Galindo-Garcia Identity-Based Signature Revisited

Sanjit Chatterjee, Chethan Kamath, and Vikas Kumar

Dept. of Computer Science and Automation,
Indian Institute of Science,
Bangalore.
{sanjit,chethan0510,vikaskumar}@csa.iisc.ernet.in

Abstract. In Africacrypt 2009, Galindo-Garcia [11] 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 [4] to solve the discrete-log problem (DLP).
In this work, we revisit the security argument given in [11]. 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 [11] fails in the standard security model for IBS [1], 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 [2] together with the programming of the random oracles and Coron's technique [7]. 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.

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)[1]. IBS or more generally, identity-based cryptosystems [19] 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 [9,13] have been proposed in the literature after the notion of IBS was introduced by Shamir in 1984 [19]. In recent times, a few pairing based constructions were also proposed [6,14,15,8]. Galindo-Garcia [11], 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 §A.2) using the random oracle methodology [3] and a variant of the forking lemma [2,4,16]. 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 [17,21].

[0] This is the full version of a paper appearing in ICISC 2012.

[1] The PKG is a trusted third party whose duty is to create and then communicate the secret keys to the users in the system through a secure channel.

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 [20] and Galindo's work on Boneh-Franklin IBE [10]. 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 [7]. In this work we revisit the security argument of Galindo-Garcia IBS [11] 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 E. The authors construct $\mathcal{B}_1$ to solve the DLP when the event E occurs. Similarly, $\mathcal{B}_2$ is used to solve the DLP in case the complement of E occurs. Both the reductions use the multiple-forking lemma [4] to show that the DLP is reduced to breaking the IBS scheme.

In this work, we make several observations about the security argument in [11]. 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 [1], 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 [11], 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 [2] and the technique first introduced by Coron [7] 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 [4], together with an algebraic technique similar to one adopted by Boneh-Boyen in [5]. 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. $s \in_R \mathbb{S}$ denotes picking an element s uniformly at random from the set $\mathbb{S}$. Similarly, $\{s_1, \ldots, s_n\} \in_R \mathbb{S}$ denotes picking elements $s_1, \ldots, s_n$ independently and uniformly at random from the set $\mathbb{S}$. $(y_1, \ldots, y_n) \xleftarrow{\$} \mathcal{A}(x_1, \ldots, x_m)$ denotes an probabilistic algorithm $\mathcal{A}$ running on input $(x_1, \ldots, x_m)$ to produce output $(y_1, \ldots, y_n)$. For an oracle H, $\#\mathrm{H}(x)$ indicates the index on which the oracle query for input x was made. In a group $\mathbb{G}$, the discrete-log to a generator g is denoted by $\log_g^{\mathbb{G}}$. Finally, in the illustrations involving the forking algorithms, $\mathtt{Q}_j^i$ denotes the j^{th} query in the i^{th} run.

Organisation. In §2, we reproduce the original security argument given in [11] and note our observations on the security argument. We give a detailed security argument for the same IBS in §3. Finally, we end with the concluding remarks in §4. For the sake of completeness, some of the standard definitions relevant to the paper are given in *Appendix A*. The forking algorithms are explained, in detail, in *Appendix B*.

2 Revisiting the Galindo-Garcia Security Argument

We first recall the construction of the Galindo-Garcia IBS and then identify several problems with the original security argument in [11].

2.1 The Construction

The scheme is based on the Schnorr signature scheme [18]. 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 §3 in [11]).

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

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

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

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

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

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

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

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

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* signing key $\mathsf{usk} = (y, R)$ will contain the same R. This also means that, in the security reduction, the simulator has to maintain the same R for a particular user; otherwise, the simulation will diverge from the actual protocol execution.

2.2 The Security Argument and Problems with it

We now reproduce the original reductions from §4 of [11] using our notation (for ease of reference, the bullets are replaced by numeric values). Then we describe in details the problems in the argument. In the following, $\mathcal{B}_i.j$ refers to the j^{th} step in the construction of $\mathcal{B}_i$, $i \in \{1, 2\}$. The description of the forking algorithms $\mathcal{F}_\mathcal{Y}$ and $\mathcal{M}_{\mathcal{Y},n}$ is given in *Appendix B*. (Some of the "typos" in the original security argument, that were corrected, are mentioned in the footnotes.)

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

2.2.1 Reduction $\mathcal{B}_1$

$\mathcal{B}_1$ takes as argument the description of a group $(\mathbb{G}, p, g)$ and a challenge g^α with $\alpha \in_R \mathbb{Z}_p$ and tries to extract the discrete logarithm α. The 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_G is the maximum number of queries that the adversary $\mathcal{A}$ performs to the G-oracle. Let $\hat{\mathsf{id}}$ (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 $(\mathsf{mpk}, \mathsf{msk}) := ((\mathbb{G}, g, p, \mathrm{G}, \mathrm{H}, g^z), z)$, 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 id, $\mathcal{B}_1$ chooses $c, y \in_R \mathbb{Z}_p$, sets $R := g^{-zc}g^y$ and adds $\langle R, \mathsf{id}, c \rangle$ 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 call to the signature oracle $\mathcal{O}_s$ for (id, m) with $\mathsf{id} \neq \hat{\mathsf{id}}$, $\mathcal{B}_1$ simply computes id's secret key as described in the previous bullet. Then it runs the signing algorithm $\mathcal{S}$ and returns the produced signature to $\mathcal{A}$.

$\mathcal{B}_1$.4 When $\mathcal{A}$ makes a call to the signature oracle $\mathcal{O}_s$ for (id, m) with $\mathsf{id} = \hat{\mathsf{id}}$, $\mathcal{B}_1$ chooses $t \in_R \mathbb{Z}_p, B \in_R \mathbb{G}$, sets $R := g^{-zc}(g^\alpha)^t, c := \mathrm{H}(\mathsf{id}, R)$,[2] and $A := B(g^\alpha g^{zc})^{-d}$.[3] Then it returns the signature (A, B, R) to $\mathcal{A}$.

$\mathcal{B}_1$.5 $\mathcal{B}_1$ runs the algorithm $\mathcal{M}_{\mathcal{Y},1}(\mathsf{mpk})$ as described in Lemma 1 (§4 in [11]). Here algorithm $\mathcal{Y}$ 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 calls to the random oracles G, H with the target identity $\hat{\mathsf{id}}$.

```
algorithm $\mathcal{M}_{\mathcal{Y},1}(\mathsf{mpk})$ –
    Pick random coins $\rho$ for $\mathcal{Y}$
    $s^0_1, \ldots, s^0_{q_\mathrm{G}} \in_R \mathbb{Z}_p$
    $(I_0, J_0, \sigma_0) \xleftarrow{\$} \mathcal{Y}(\mathsf{mpk}, s^0_1, \ldots, s^0_{q_\mathrm{G}}, \rho)$
    If $(I_0 = 0 \vee J_0 = 0)$ then return $\perp$
    $s^1_{I_0}, \ldots, s^1_{q_\mathrm{G}} \in_R \mathbb{Z}_p$
    $(I_1, J_1, \sigma_1) \xleftarrow{\$} \mathcal{Y}(\mathsf{mpk}, s^0_1, \ldots, s^0_{I_0-1}, s^1_{I_0}, \ldots, s^1_{q_\mathrm{G}}, \rho)$
    If $((I_1, J_1) \neq (I_0, J_0) \vee s^1_{I_0} = s^0_{I_0})$ then return $\perp$
    Otherwise return $(\sigma_0, \sigma_1)$
```

In this way we get two forgeries of the form $\sigma_0 = (\mathsf{id}, m, (A, B_0, R))$ and $\sigma_1 = (\mathsf{id}, m, (A, B_1, R))$. Let d_0 be the answer from the G-oracle given to $\mathcal{A}$ in the first execution, $s^0_{I_0}$ in $\mathcal{M}_{\mathcal{Y},1}$ and let d_1 be the second answer $s^1_{I_0}$. If the identity id is not equal to the target identity $\hat{\mathsf{id}}$ 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.

(i) *Correctness of signatures on* $\hat{\mathsf{id}}$: In $\mathcal{B}_1$.4, when $\mathcal{A}$ makes a signature query on $\hat{\mathsf{id}}$, $\mathcal{B}_1$ 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 $\hat{\mathsf{id}}$ 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 $\mathcal{A}$ makes a call to the signature oracle $\mathcal{O}_s$ for (id, m) with $\mathsf{id} = \hat{\mathsf{id}}$, $\mathcal{B}_1$ chooses $t, b \in_R \mathbb{Z}_p$, *sets* $B := g^b$, $R := g^{-zc}(g^\alpha)^t, c := \mathrm{H}(\mathsf{id}, R)$ 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 $\hat{\mathsf{id}}$ 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

$$(\alpha t - zc - \alpha)\, d \equiv 0 \bmod p. \tag{1}$$

It is easy to check that the LHS in (1) is a random element of $\mathbb{Z}_p$. Hence, the signatures on $\hat{\mathsf{id}}$ given by $\mathcal{B}_1$ will fail to verify with an overwhelming probability of $1 - \frac{1}{p}$. The equality

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

[3] Here, d is not assigned a value, though from the protocol we may infer that $d := \mathrm{G}(\mathsf{id}, A, m)$. 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 $\mathrm{G}(\mathsf{id}, A, m) := d$.

holds if we set $t := 1 + \frac{zc}{\alpha}$, instead of selecting t uniformly at random from $\mathbb{Z}_p$.[4] However, setting $t := 1 + \frac{zc}{\alpha}$ results in R being set to the problem instance g^α, removing t from the picture altogether. Thus, $\mathcal{B}_1$.4 would finally look like:

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

(ii) *Ambiguity due to the choice of* $\hat{\mathsf{id}}$: $\mathcal{B}_1$ sets the identity involved in the $\hat{\imath}^{\text{th}}$ G-oracle query as the target identity $\hat{\mathsf{id}}$ (see $\mathcal{B}_1$.1). Hence, the target identity can be fixed only *after* the $\hat{\imath}^{\text{th}}$ 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 $\hat{\imath}^{\text{th}}$ G-oracle call, $\mathcal{B}_1$ has no way to decide whether to proceed to $\mathcal{B}_1$.3 or $\mathcal{B}_1$.4 (as it depends on whether $\mathsf{id} = \hat{\mathsf{id}}$ 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 $\hat{\mathsf{id}}$ *before* the $\hat{\imath}^{\text{th}}$ G-oracle call. 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 $\hat{\mathsf{id}}$ before the $\hat{\imath}^{\text{th}}$ G-oracle query and one more signature query on $\hat{\mathsf{id}}$ after the $\hat{\imath}^{\text{th}}$ G-oracle query.
One way to fix the problem noted above is to guess the "index" of the target identity *instead* of guessing the index of the G-oracle query in which the target identity is involved. Suppose n distinct identities are involved in the queries to the G-oracle, where $1 \leq n \leq q_G$.[5] The strategy would be to guess the index $\hat{\imath}$ of the target identity $\hat{\mathsf{id}}$ among all the identities, i.e. if $\{\mathsf{id}_1, \ldots, \mathsf{id}_n\}$ were the *distinct* identities involved in the queries to the G-oracle (in that order), we set $\mathsf{id}_{\hat{\imath}}$ with $1 \leq \hat{\imath} \leq n$ as the target identity. Now, by assumption no identity queried to the G-oracle prior to $\mathsf{id}_{\hat{\imath}}$ can be the target identity. Hence, the ambiguity noted before can be avoided. Although this strategy works well with the "mended" reduction which we ended up in *Observation (i)*, 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 §3, 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.

2.2.2 Reduction $\mathcal{B}_2$

It takes as argument, the description of a group $(\mathbb{G}, p, g)$ and a challenge g^α with $\alpha \in_R \mathbb{Z}_p$ and outputs the discrete logarithm α. 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 public parameters mpk:=$(\mathbb{G}, p, g, \mathrm{G}, \mathrm{H})$ and $\mathsf{msk} := (g^\alpha)$, 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 $c, y \in_R \mathbb{Z}_q$, sets $R := g^{-\alpha c} g^y$ and adds $\langle R, \mathsf{id}, c\rangle$ to the table $\mathfrak{L}_{\mathrm{H}}$. Then it returns the key (y, R) to $\mathcal{A}$.

$\mathcal{B}_2$.3 When $\mathcal{A}$ makes a call to the signature oracle $\mathcal{O}_s$ with (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, $((\mathsf{id}, g^a, m), d)$ to the table $\mathfrak{L}_{\mathrm{G}}$ and gives the resulting signature to the adversary.

$\mathcal{B}_2$.4 $\mathcal{B}_2$ runs the algorithm $\mathcal{M}_{\mathcal{Y},3}(\mathsf{mpk})$. In this way either $\mathcal{B}_2$ aborts prematurely or we get, for some identity id, some message m and some R, four forgeries $(\mathsf{id}, m, (A_k, b_k, R))$[6], $k :=$

[4] This modification was pointed out by an anonymous reviewer of an earlier version of this paper.
[5] $\mathcal{B}_1$ will maintain a counter and increment it by 1 each time a new identity is queried to the G-oracle.
[6] We use b_k instead of B_k, throughout the reduction, to maintain consistency with the protocol description (in §2.1).

$0, \ldots, 3$ with $A_0 = A_1$ and $A_2 = A_3$. As all these signatures are valid, the following equations hold.

$$\begin{aligned} b_0 &= \log A_0 + (\log R + c_0\alpha)d_0 \quad , & b_1 &= \log A_1 + (\log R + c_0\alpha)d_1, \\ b_2 &= \log A_2 + (\log R + c_1\alpha)d_2 \quad \text{and} & b_3 &= \log A_3 + (\log R + c_1\alpha)d_3 \end{aligned} \tag{2}$$

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

$$\alpha = \frac{b_2 + b_1 - b_0 - b_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)}. \tag{3}$$

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.

(i) *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 (2). However, on substituting the values of b_ks from (2) in (3) we get

$$\frac{b_2 + b_1 - b_0 - b_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)} = \alpha + \log_g^{\mathbb{G}} R \cdot \frac{d_2 + d_1 - d_0 - d_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)}, \tag{4}$$

which is *not* the correct solution to the DLP instance. Note that the simulator does not know the value of $\log_g^{\mathbb{G}} R$ and hence cannot extract α from the above expression. However, it is not difficult to get the correct solution as we show in (23) of §3.5. The more fundamental problem is that $\mathcal{B}_2$ fails to capture all possible adversarial strategies as we show next.

(ii) *Incompleteness of $\mathcal{B}_2$*: In Step $\mathcal{B}_2$.4, $\mathcal{B}_2$ runs $\mathcal{M}_{\mathcal{Y},3}$ to get four forged signatures with b_ks as given in (2). 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 oracle calls: $\mathrm{G}(\mathsf{id}, A, m)$ and $\mathrm{H}(R, \mathsf{id})$, during the simulation. (Here, (id, m) corresponds to the target identity and the message pair in the forgery while (A, R) are part of the forged signature.) Thus, (2) covers only one of the two possible adversarial behaviors, which we call Case 1:– $\mathcal{A}$ calling the H-oracle before the G-oracle (shown in *Figure 4* where the first branching corresponds to the forking of the H-oracle). But one cannot rule out the second case, i.e. Case 2:– $\mathcal{A}$ calling the G-oracle before the H-oracle, illustrated in *Figure 1* below, where the first branching corresponds to the forking of the G-oracle.

$$\begin{tikzcd}
 & & & & & \mathsf{Q}^0_{I_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^0_\gamma \arrow[r] & \sigma_0 = (A, b_0, R_0)\ \textsf{[Run 0]} \\
 & & & \mathsf{Q}^0_{J_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^0_{I_0} \arrow[ur, "c_0"] \arrow[dr, "c_1"'] & & & \\
 & & & & & \mathsf{Q}^1_{I_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^1_\gamma \arrow[r] & \sigma_1 = (A, b_1, R_0)\ \textsf{[Run 1]} \\
\mathsf{Q}^0_1 \arrow[r] & \mathsf{Q}^0_2 \arrow[r, dotted, no head] & \mathsf{Q}^0_{J_0} \arrow[uur, "d_0"] \arrow[ddr, "d_2"'] & & & & & \\
 & & & & & \mathsf{Q}^2_{I_1+1} \arrow[r, dotted, no head] & \mathsf{Q}^2_\gamma \arrow[r] & \sigma_2 = (A, b_2, R_2)\ \textsf{[Run 2]} \\
 & & & \mathsf{Q}^2_{J_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^2_{I_1} \arrow[ur, "c_2"] \arrow[dr, "c_3"'] & & & \\
 & & & & & \mathsf{Q}^3_{I_1+1} \arrow[r, dotted, no head] & \mathsf{Q}^3_\gamma \arrow[r] & \sigma_3 = (A, b_3, R_2)\ \textsf{[Run 3]}
\end{tikzcd}$$

$$\mathsf{Q}^0_{J_0} = \mathrm{G}(\mathsf{id}, A, m)$$
$$\mathsf{Q}^0_{I_0} = \mathrm{H}(R_0, \mathsf{id})\ ,\ \mathsf{Q}^2_{I_0} = \mathrm{H}(R_2, \mathsf{id})$$

Fig. 1. Structure of the forgeries in Case 2. (Note that the branchings indicate forking.)

Let's look into the structure of the forged signatures in Case 2. As a result of the ordering of the oracle calls, $\mathcal{Y}$ returns J_0 as the index of the G-oracle call on (id, A, m) and I_0 as the index of the H-oracle call on (R_0, id), at the end of Run 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 hand, the value A returned as part of the forged signature remains the same in all the four runs. Hence, the signatures returned by $\mathcal{M}_{\mathcal{Y},3}$ will contain b_ks of the form

$$\begin{aligned} b_0 &= \log A + (\log R_0 + c_0\alpha)d_0 \quad , \quad & b_1 &= \log A + (\log R_0 + c_1\alpha)d_0, \\ b_2 &= \log A + (\log R_2 + c_2\alpha)d_2 \quad \text{and} \quad & b_3 &= \log A + (\log R_2 + c_3\alpha)d_2. \end{aligned} \tag{5}$$

When the signatures have the structure as in (5), we cannot use (3) (more precisely, the corrected version as given in (23) of §3.5) 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, Case 1 and Case 2, simply by looking at the structure of the forged signatures. In Case 1, all the Rs will be equal, i.e. $R_3 = R_2 = R_1 = R_0$; on the other hand, in Case 2, all the As will be equal, i.e. $A_3 = A_2 = A_1 = A_0$. We could then use the appropriate relations, i.e., (23) and (18) (derived in *Appendix 3.3* and §3.2, respectively) to solve for the DLP instance. However, this results in an *unnecessary* forking (the branch consisting of Run 2 and Run 3 in *Figure 1*) being executed in Case 2. We address this in §3 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 significantly tighter reduction (see *Table 1*).

For the sake of completeness, we provide the modified security argument incorporating all the above mentioned fixes in *Appendix C*.

3 New Security Argument

On the basis of the observations made in the previous section, we now proceed to provide a detailed security argument for Galindo-Garcia 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 significantly tighter reductions in two of the three cases.

In order to address the problem in $\mathcal{B}_1$ we redefine the event E and to address the incompleteness of $\mathcal{B}_2$ we introduce another event 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 general forking algorithm, whereas $\mathcal{R}_2$ and $\mathcal{R}_3$, the counterparts of $\mathcal{B}_2$, still use the multiple-forking algorithm. The new reductions $\mathcal{R}_1$ and $\mathcal{R}_2$ are also tighter than their counterparts in [11].

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 H and G are modelled as random oracles, we can construct either*

(i) Algorithm $\mathcal{R}_1$ which (ϵ_1, t_1)-breaks the DLP, where

$$\epsilon_1 \geq \frac{\epsilon^2}{\exp(1) q_G q_\varepsilon} \quad \textit{and} \quad t_1 \leq t + 2(q_\varepsilon + 3q_s)\tau, \tag{6}$$

or

(ii) Algorithm $\mathcal{R}_2$ which (ϵ_2, t_2)-breaks the DLP, where

$$\epsilon_2 \geq \epsilon \left(\frac{\epsilon}{(q_H + q_G)^2} - \frac{1}{p} \right) \quad \textit{and} \quad t_2 \leq t + 2(2q_\varepsilon + 3q_s)\tau, \tag{7}$$

or

(iii) Algorithm $\mathcal{R}_3$ which (ϵ_3, t_3)-breaks the DLP, where

$$\epsilon_3 \geq \epsilon \left(\frac{\epsilon^3}{(q_H + q_G)^6} - \frac{3}{p} \right) \quad \textit{and} \quad t_3 \leq t + 4(2q_\varepsilon + 3q_s)\tau. \tag{8}$$

Here q_ε and q_s denote the upper bound on the number of extract and signature queries, respectively, that $\mathcal{A}$ can make; q_H and q_G denote the upper bound on the number of queries to the H-oracle and G-oracle respectively. τ is the time taken for an exponentiation in the group $\mathbb{G}$ and exp *is the base of natural logarithm.*

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

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

The complement of this event is

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

Note that the definition of the new event E (and $\bar{\text{E}}$) is slightly different from the one given in the security argument of [11], i.e. event E (and NE) discussed in §2.2.

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

F:– The event that $\mathcal{A}$ makes the oracle call $\text{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$ before the oracle call $\text{H}(\hat{R}, \hat{\mathsf{id}})$.

The complement of this event is

$\bar{\text{F}}$:– The event that $\mathcal{A}$ makes the oracle call $\text{H}(\hat{R}, \hat{\mathsf{id}})$ before the oracle call $\text{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$.

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

Remark 2. The reductions in the subsequent sections are described in two steps. In the first step, called "Handling the queries", we describe the protocol set-up and the methods in which adversarial queries – signature, extract and random oracle queries, are to be handled. This gives us sufficient know-how to simulate the protocol environment. In the second step, called "Solving the DLP", we describe how the reductions use forking algorithms to solve the underlying hard problem, which in this case is the DLP. The reductions use the forking algorithm simply as a black-box to get hold of the forgeries. The actual simulation is handled by the wrapper algorithm $\mathcal{Y}$ in the forking algorithms, according to the plan laid down by the reductions in the first step. For details on how the forking algorithms work, see *Appendix B.*

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 from 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 $\mathcal{A}$.

3.1 Reduction $\mathcal{R}_1$

$\mathcal{R}_1$ uses the so-called "partitioning strategy", first used by Coron in the security argument of FDH [7]. 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. The simulator 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 $\mathbb{I}_s$. Finally, the simulator hopes that the adversary produces a forgery on an identity from $\mathbb{I}_s$. The optimal size of the sets is determined on analysis.

In $\mathcal{R}_1$ the problem instance is embedded in the randomiser R, depending on the outcome of a biased coin. As $\mathcal{R}_1$ 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 details follow.

Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. $\mathcal{R}_1$ sets $z \in_R \mathbb{Z}_p$ as the *master* secret key. The public parameters $\mathsf{mpk} := (\mathbb{G}, p, g, g^z, \mathrm{H}, \mathrm{G})$ are released to the adversary. The hash functions H and G are modelled as random oracles. This is done with the aid of two tables, $\mathfrak{L}_\mathrm{H}$ and $\mathfrak{L}_\mathrm{G}$.

3.1.1 Handling the Queries

H-oracle Query. $\mathfrak{L}_\mathrm{H}$ contains tuples of the form

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

Here, (R, id) is the query to the H-oracle and c is the corresponding output. Therefore, an oracle query $\mathrm{H}(R, \mathsf{id})$ is fresh if there exists no tuple $\langle R_i, r_i, \mathsf{id}_i, c_i, \beta_i \rangle$ in $\mathfrak{L}_\mathrm{H}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (R_i = R)$. If such a tuple exists, then the oracle has to return 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 $\mathcal{R}_1$ does not have any additional information regarding R. In these tuples, the β-field is irrelevant and this is indicated by storing 'ϕ'. In tuples with $r \neq \perp$, the field β 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. Therefore, the r-field stores one of the values: (i) '$\perp$', (ii) $\log_g^{\mathbb{G}} R$ or (iii) r, if $R = (g^\alpha)^r$. As a result, $\mathfrak{L}_\mathrm{H}$ can contain three types of tuples determined by the content of r-field and β-field, viz.

1. $r = \perp$: These tuples correspond to the explicit H-oracle queries made by $\mathcal{A}$.
2. $r \neq \perp \wedge \beta = 0$: These tuples correspond to identities in $\mathbb{I}_s$. They are added by $\mathcal{R}_1$ while answering the signature queries. As the DLP instance is embedded in R, extract query fails on these identities.
3. $r \neq \perp \wedge \beta = 1$: These tuples correspond to identities in $\mathbb{I}_\varepsilon$. They are added by $\mathcal{R}_1$ while answering signature or extract queries.

We now explain how the fresh H-oracle queries are handled.

$\mathrm{H}(R, \mathsf{id})$:– The query may be

(i) H_1, Explicit query made by $\mathcal{A}$:– In this case $\mathcal{R}_1$ returns $c \in_R \mathbb{Z}_p$ as the output. $\langle R, \perp, \mathsf{id}, c, \phi \rangle$ is added to $\mathfrak{L}_\mathrm{H}$.

(ii) H_2, Explicit query made by $\mathcal{R}_1$:– As in the previous case, $\mathcal{R}_1$ returns $c \in_R \mathbb{Z}_p$ as the output. As $\mathcal{R}_1$ knows $r = \log_g^{\mathbb{G}} R$, $\langle R, r, \mathsf{id}, c, 1 \rangle$ is added to $\mathfrak{L}_\mathrm{H}$.

(iii) H_3, Implicit query by $\mathcal{R}_1$ in order to answer a signature query made by $\mathcal{A}$:– See *Sign* (iii) on how to program the random oracle in this situation.

G-oracle Query. $\mathfrak{L}_{\mathrm{G}}$ contains tuples of the form

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

Here, (id, A, m) is the query to the G-oracle and d is the corresponding output. Therefore, a random oracle query $\mathrm{G}(\mathsf{id}, A, m)$ is fresh if there exists no tuple $\langle \mathsf{id}_i, A_i, m_i, d_i\rangle$, in $\mathfrak{L}_{\mathrm{G}}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (A_i = A) \wedge (m_i = m)$. If such a tuple exists, then the oracle has to return d_i as the output.

We now explain how the fresh G-oracle queries are handled.

$\mathrm{G}(\mathsf{id}, A, m)$:– The query may be
(i) G_1, Explicit query made by the either $\mathcal{A}$ or $\mathcal{R}_1$:– In this case $\mathcal{R}_1$ returns $d \in_R \mathbb{Z}_p$ as the output. $\langle \mathsf{id}, A, m, d\rangle$ is added to $\mathfrak{L}_{\mathrm{G}}$.
(ii) G_2, Implicit query by $\mathcal{R}_1$ in order to answer a signature query made by $\mathcal{A}$:– See *Sign* (i), (iii) on how to program the random oracle in this situation.

Remark 3. In the case of the implicit calls H_3 and G_2, $\mathcal{R}_1$ has to program the respective random oracles in an appropriate way to deal with the circularity involved. For ease of understanding, they are dealt with in their respective sections.

Now that $\mathcal{R}_1$ can handle the random oracle queries, the extract and signature queries are answered as follows.

Extract Query. $\mathcal{R}_1$ first checks if id has an associated R. This is done by searching for tuples $\langle R_i, r_i, \mathsf{id}_i, c_i, \beta_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ with $(\mathsf{id}_i = \mathsf{id}) \wedge (r_i \neq \perp)$. If such a tuple exists, $\mathcal{R}_1$ checks for the value of β_i in the tuple. $\beta_i = 0$ implies the identity belongs to $\mathbb{I}_s$ and consequently the extract query fails, leading to an abort, $\mathsf{abort}_{1,1}$. On the other hand, $\beta_i = 1$ implies that there was a prior extract query on id and also that the identity belongs to $\mathbb{I}_\varepsilon$. $\mathcal{R}_1$ generates the 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, $\mathcal{R}_1$ selects a fresh r and assigns id to $\mathbb{I}_\varepsilon$. $\mathcal{R}_1$ has this freedom since the adversary cannot forge on this identity. This is captured by the oracle $\mathcal{O}_\varepsilon$ shown below.

$\mathcal{O}_\varepsilon(\mathsf{id})$:–
<u>If</u> there exists a tuple $\langle R_i, r_i, \mathsf{id}_i, c_i, \beta_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (r_i \neq \perp)$
(i) If $\beta_i = 0$, $\mathcal{R}_1$ *aborts* $(\mathsf{abort}_{1,1})$.
(ii) Otherwise, $\beta_i = 1$ and $\mathcal{R}_1$ returns $\mathsf{usk} := (r_i + zc_i, R_i)$ as the secret key for id.
<u>Otherwise</u>
(iii) $\mathcal{R}_1$ chooses $r \in_R \mathbb{Z}_p$, sets $R := g^r$ and asks the H-oracle for $c := \mathrm{H}(R, \mathsf{id})$. It returns $\mathsf{usk} := (r + zc, R)$ as the secret key.

Signature Query. As in the extract query, $\mathcal{R}_1$ checks the identity for an associated R by searching tuples $\langle R_i, r_i, \mathsf{id}_i, c_i, \beta_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ with $(\mathsf{id}_i = \mathsf{id}) \wedge (r_i \neq \perp)$. 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 β_i. If such a tuple does not exist, then the identity is unassigned and $\mathcal{R}_1$ assigns the identity to either $\mathbb{I}_\varepsilon$ or $\mathbb{I}_s$ by tossing a biased coin β. If the outcome is 0, 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^α embedded in the randomiser R. Although the private key cannot be calculated, an algebraic technique, similar to one adopted by Boneh-Boyen in [5], 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 the usk as in the extract query and then running $\mathcal{S}$. This is captured by the oracle $\mathcal{O}_s$ described below.

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

(i) If $\beta_i = 0$, $\mathcal{R}_1$ selects $s, d \in_R \mathbb{Z}_p$ and sets $A := g^s(g^\alpha)^{-r_i d}$. Then (id, A, m, d) is added to $\mathfrak{L}_{\mathrm{G}}$ (*Deferred case G_2*)[7]. The signature returned is

$$\sigma := (A, s + zcd, R_i).$$

(ii) Otherwise, $\beta_i = 1$ and the secret key for id is $\mathsf{usk} = (y, R_i)$, where $y = r_i + zc_i$ and $R_i = g^{r_i}$. $\mathcal{R}_1$ selects $a \in_R \mathbb{Z}_p$, sets $A := g^a$ and asks G-oracle for $d := \mathrm{G}(\mathsf{id}, A, m)$. The signature returned is

$$\sigma := (A, a + yd, R_i).$$

<u>Otherwise</u>, $\mathcal{R}_1$ tosses a coin β with a bias δ (i.e, $\Pr[\beta = 0]=\delta$). The value of δ will be quantified on analysis.

(iii) If $\beta = 0$, $\mathcal{R}_1$ selects $c, d, s, r \in_R \mathbb{Z}_p$ and sets $R := (g^\alpha)^r$, $A := g^s(g^\alpha)^{-rd}$. Next, it adds $\langle (g^\alpha)^r, r, \mathsf{id}, c, 0\rangle$ to $\mathfrak{L}_{\mathrm{H}}$ (*Deferred case H_3*) and $\langle \mathsf{id}, A, m, d\rangle$ to $\mathfrak{L}_{\mathrm{G}}$ (*Deferred case G_2*).[8] The signature returned is

$$\sigma := (A, s + zcd, R).$$

(iv) Otherwise, $\beta = 1$ and $\mathcal{R}_1$ selects $a, r \in_R \mathbb{Z}_p$ and sets $A := g^a, R := g^r$. It then asks the respective oracles for $c := \mathrm{H}(R, \mathsf{id})$ and $d := \mathrm{G}(\mathsf{id}, A, m)$. The signature returned is

$$\sigma := (A, a + (r + zc)d, R).$$

Correctness. For $\beta = 0$, the signature given by $\mathcal{R}_1$ is of the form (A, b, R), where $A = g^s(g^\alpha)^{-rd}$, $b = s+zcd$ and $R = (g^\alpha)^r$. $\mathcal{R}_1$ also sets $c := \mathrm{H}(R, \mathsf{id})$ and $d := \mathrm{G}(\mathsf{id}, A, m)$. The signature verifies as shown below.

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

For $\beta = 1$, the signatures are generated as in the protocol. Therefore they fundamentally verify.

To conclude the queries section, we calculate the probability of the event $\neg\mathsf{abort}_{1,1}$. $\mathcal{R}_1$ aborts only when $\mathcal{A}$ does an extract query on an identity from $\mathbb{I}_s$, i.e. an identity with $\beta = 0$. Therefore, $\mathcal{R}_1$ does not abort if all the extract queries are from $\mathbb{I}_\varepsilon$ and we have

$$\Pr[\neg\mathsf{abort}_{1,1}] = (1 - \delta)^{q_\varepsilon}. \tag{9}$$

3.1.2 Solving the DLP $\mathcal{R}_1$ now uses the general forking algorithm $\mathcal{F}_\mathcal{Y}$ (see *Appendix B.1* for details on the working of $\mathcal{F}_\mathcal{Y}$) to solve the DLP challenge. It runs $\mathcal{F}_\mathcal{Y}$ on the given DLP instance Δ,[9] with the G-oracle involved in the replay attack. If $\mathcal{F}_\mathcal{Y}$ fails, $\mathcal{R}_1$ *aborts* ($\mathsf{abort}_{1,2}$). On the other hand, if $\mathcal{F}_\mathcal{Y}$ is successful, it gets two valid forgeries

$$\hat{\sigma}_0 = (\hat{A}, \hat{b}_0, \hat{R}_0) \text{ and } \hat{\sigma}_1 = (\hat{A}, \hat{b}_1, \hat{R}_1)$$

on $(\hat{\mathsf{id}}, \hat{m})$, as illustrated in *Figure 2*. $\mathcal{R}_1$ now retrieves two tuples

$$\mathsf{t}_i := \langle R_i, r_i, \mathsf{id}_i, c_i, \beta_i\rangle \mid (\mathsf{id}_i = \hat{\mathsf{id}}) \wedge (R_i = \hat{R}_0) \text{ and}$$

[7] If there exists a tuple $\langle \mathsf{id}_i, A_i, m_i, d_i\rangle$ in $\mathfrak{L}_{\mathrm{G}}$ with $\mathsf{id}_i = \mathsf{id} \wedge A_i = A \wedge m_i = m$ but $d_i \neq d$ then $\mathrm{G}(\mathsf{id}, A, m)$ cannot be set to d. In that case $\mathcal{R}_1$ can simply choose a fresh set of randomisers s, d and repeat the process.

[8] $\mathcal{R}_1$ chooses different randomisers if there is a collision as explained in *Footnote 7*.

[9] In reductions involving the forking algorithm, the problem instance is usually embedded in mpk. Therefore the forking algorithm $\mathcal{F}_\mathcal{Y}$ is run on mpk. But in case of $\mathcal{R}_1$, the master secret is chosen by $\mathcal{R}_1$ itself. The problem instance (g^α) is embedded as part of answers to signature queries. Therefore, $\mathcal{R}_1$ runs $\mathcal{F}_\mathcal{Y}$ on Δ while $(\mathsf{msk}, \mathsf{mpk})$ is considered to be part of ρ.

$$\begin{tikzcd}
 & & & \mathsf{Q}^0_{I_0+1} \arrow[r, dash, dotted] & \mathsf{Q}^0_\gamma \arrow[r, "s^0_\gamma"] & \hat{\sigma}_0 = (\hat{A}, \hat{b}_0, \hat{R}_0)\ [\mathsf{Run\ 0}] \\
\mathsf{Q}^0_1 \arrow[r, "s^0_1"] & \mathsf{Q}^0_2 \arrow[r, dash, dotted] & \mathsf{Q}^0_{I_0} \arrow[ur, "s^0_{I_0}"] \arrow[dr, "s^1_{I_0}"'] & & & \\
 & & & \mathsf{Q}^1_{I_0+1} \arrow[r, dash, dotted] & \mathsf{Q}^1_\gamma \arrow[r, "s^1_\gamma"'] & \hat{\sigma}_1 = (\hat{A}, \hat{b}_1, \hat{R}_1)\ [\mathsf{Run\ 1}]
\end{tikzcd}$$

$\mathsf{Q}^0_{I_0} = \mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$

Fig. 2. Successful oracle replay attack by $\mathcal{R}_1$.

$$\mathfrak{t}_j := \langle R_j, r_j, \mathsf{id}_j, c_j, \beta_j \rangle \ \mid \ (\mathsf{id}_j = \hat{\mathsf{id}}) \wedge (R_j = \hat{R}_1)$$

from $\mathfrak{L}_{\mathrm{H}}$. $\mathcal{R}_1$ *aborts* ($\mathsf{abort}_{1,3}$) if both β_i and β_j are equal to 1. Otherwise it solves for α as shown below. Note that d_0 and d_1 represent the value of $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$ in the two runs, i.e., $d_0 = s^0_{I_0}$ and $d_1 = s^1_{I_0}$. Let $\hat{a} := \log_g^{\mathbb{G}} \hat{A}$.

(i) $(\beta_i = 1) \wedge (\beta_j = 0)$:– In this case, $\hat{R}_0 = g^{r_i}$ and $\hat{R}_1 = g^{r_j \alpha}$. Thus we have $\hat{b}_0 = \hat{a} + (r_i + zc_i)d_0$ and $\hat{b}_1 = \hat{a} + (r_j\alpha + zc_j)d_1$.

$$\begin{aligned} \hat{b}_0 - \hat{b}_1 &= (r_i d_0 - r_j \alpha d_1) + z(c_i d_0 - c_j d_1), \\ \alpha &= \frac{z(c_i d_0 - c_j d_1) + r_i d_0 - (\hat{b}_0 - \hat{b}_1)}{r_j d_1}. \end{aligned} \tag{10}$$

(ii) $(\beta_i = 0) \wedge (\beta_j = 1)$:– In this case, $\hat{R}_0 = g^{r_i \alpha}$ and $\hat{R}_1 = g^{r_j}$. Thus we have $\hat{b}_0 = \hat{a} + (r_i\alpha + zc_i)d_0$ and $\hat{b}_1 = \hat{a} + (r_j + zc_j)d_1$.

$$\begin{aligned} \hat{b}_1 - \hat{b}_0 &= (r_j d_1 - r_i \alpha d_0) + z(c_j d_1 - c_i d_0), \\ \alpha &= \frac{z(c_j d_1 - c_i d_0) + r_j d_1 - (\hat{b}_1 - \hat{b}_0)}{r_i d_0}. \end{aligned} \tag{11}$$

(iii) $(\beta_i = 0) \wedge (\beta_j = 0)$:– In this case, $\hat{R}_0 = g^{r_i \alpha}$ and $\hat{R}_1 = g^{r_j \alpha}$. Thus we have $\hat{b}_0 = \hat{a} + (r_i\alpha + zc_i)d_0$ and $\hat{b}_1 = \hat{a} + (r_j\alpha + zc_j)d_1$.

$$\begin{aligned} \hat{b}_0 - \hat{b}_1 &= \alpha(r_i d_0 - r_j d_1) + z(c_i d_0 - c_j d_1), \\ \alpha &= \frac{(\hat{b}_0 - \hat{b}_1) - z(c_i d_0 - c_j d_1)}{(r_i d_0 - r_j d_1)}. \end{aligned} \tag{12}$$

Remark 4. The equations (10), (11) and (12) 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 $\mathrm{H}(\hat{R}_0, \hat{\mathsf{id}})$ before the query $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$ in $\mathsf{Run\ 0}$. Hence, the order in which $\mathcal{A}$ makes the aforementioned random oracle calls is not relevant.

Structure of R. The event E guarantees the existence of the tuple $\mathfrak{t}_i$ in $\mathfrak{L}_{\mathrm{H}}$. As $\mathcal{A}$ cannot make an extract query on $\hat{\mathsf{id}}$, the choice of $\hat{R}_i$ must have been made during a signature query, where the structure of R is determined by the coin β. Therefore

$$\hat{R}_i = \begin{cases} g^{\hat{r}_i \alpha} & \text{If } \beta_i = 0 \\ g^{\hat{r}_i} & \text{Otherwise} \end{cases}$$

This, in turn, is determined by the bias in β, i.e. δ. A similar argument holds for $\mathfrak{t}_j$. $\mathcal{R}_1$ is successful in solving the DLP if either of the forgeries have $\beta = 0$. If $(\beta_i = 1) \wedge (\beta_j = 1)$, $\mathcal{R}_1$

fails and does $\mathsf{abort}_{1,3}$. We conclude by calculating the probability of $\mathsf{abort}_{1,3}$ provided $\mathsf{abort}_{1,2}$ has not occurred. It is same as the probability with which $(\beta_i = 1) \wedge (\beta_j = 1)$, i.e.

$$\Pr[\mathsf{abort}_{1,3} \mid \neg\mathsf{abort}_{1,2}] = (1-\delta)^2. \tag{13}$$

Let gfrk be the probability with which $\mathcal{F}_{\mathcal{Y}}$ is successful. Since $\mathsf{abort}_{1,2}$ occurs if $\mathcal{F}_{\mathcal{Y}}$ fails, we have

$$\Pr[\neg\mathsf{abort}_{1,2}] = \mathsf{gfrk}. \tag{14}$$

3.1.3 Analysis The probability analysis is done in terms of the aborts $\mathsf{abort}_{1,1}$, $\mathsf{abort}_{1,2}$ and $\mathsf{abort}_{1,3}$. From (9), (14) and (13), we have

$$\Pr[\neg\mathsf{abort}_{1,1}] = (1-\delta)^{q_\varepsilon} \ , \ \Pr[\neg\mathsf{abort}_{1,2}] = \mathsf{gfrk} \ \text{ and } \ \Pr[\mathsf{abort}_{1,3} \mid \neg\mathsf{abort}_{1,2}] = (1-\delta)^2.$$

$\mathcal{F}_{\mathcal{Y}}$ is successful during $\mathsf{Run}\ 0$ if there is no abort during the query phase ($\neg\mathsf{abort}_{1,1}$) and $\mathcal{A}$ produces a valid forgery. We denote this probability as acc_1. Thus

$$\begin{aligned}\mathsf{acc}_1 &\geq \Pr[\neg\mathsf{abort}_1] \cdot \epsilon \\ &\geq (1-\delta)^{q_\varepsilon} \cdot \epsilon.\end{aligned}$$

Applying (25) from the general forking lemma with $|\ \mathbb{S}\ | = p$ and $\gamma = q_{\mathrm{G}}$, we get

$$\begin{aligned}\mathsf{gfrk} &\geq \mathsf{acc}_1 \cdot \left(\frac{\mathsf{acc}_1}{q_{\mathrm{G}}} - \frac{1}{p}\right) \\ &\geq (1-\delta)^{q_\varepsilon}\epsilon \cdot \left(\frac{(1-\delta)^{q_\varepsilon}\epsilon}{q_{\mathrm{G}}} - \frac{1}{p}\right).\end{aligned}$$

If we assume $p \gg 1$, the above expression approximates to

$$\mathsf{gfrk} \geq \frac{(1-\delta)^{2q_\varepsilon}\epsilon^2}{q_{\mathrm{G}}}.$$

Now, $\mathcal{R}_1$ is successful in solving DLP if neither of the aborts, $\mathsf{abort}_{1,2}$ and $\mathsf{abort}_{1,3}$, occur. Thus the advantage it has is

$$\begin{aligned}\epsilon_1 &= \Pr[\neg\mathsf{abort}_{1,3} \wedge \neg\mathsf{abort}_{1,2}] \\ &= \Pr[\neg\mathsf{abort}_{1,3} \mid \neg\mathsf{abort}_{1,2}] \cdot \Pr[\neg\mathsf{abort}_{1,2}] \\ &\geq (1-(1-\delta)^2) \cdot \mathsf{gfrk} \\ &\geq (2\delta - \delta^2)\frac{(1-\delta)^{2q_\varepsilon}\epsilon^2}{q_{\mathrm{G}}}.\end{aligned} \tag{15}$$

The above expression is maximised when $\delta = \left(1 - \sqrt{\frac{q_\varepsilon - 2}{q_\varepsilon - 1}}\right)$, at which we get

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

Here, exp is the base of natural logarithm. Assuming $q_\varepsilon \gg 1$, we get the approximation

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

Remark 5. The above reduction is tighter than the reduction $\mathcal{B}_1$ given by Galindo-Garcia [11]. This can be attributed to two reasons: (i) $\mathcal{R}_1$ using the general forking algorithm $\mathcal{F}_{\mathcal{Y}}$ instead of the multiple-forking algorithm $\mathcal{M}_{\mathcal{Y},1}$ and (ii) $\mathcal{B}_1$ in [11] randomly chooses one of the identities involved in the G-oracle call as the target identity (Refer to §2.2.1) which contributes a factor of q_{G} to the degradation in $\mathcal{B}_1$. In contrast, we apply Coron's technique in $\mathcal{R}_1$ to partition the identity space in some optimal way.

Time Analysis. If τ 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.

3.2 Reduction $\mathcal{R}_2$

The reduction $\mathcal{R}_2$ is similar in some aspects to the (incomplete) reduction argument $\mathcal{B}_2$ in [11]. However, a *major* difference is that $\mathcal{R}_2$ uses the multiple-forking algorithm $\mathcal{M}_{\mathcal{Y},1}$ instead of $\mathcal{M}_{\mathcal{Y},3}$ to solve the DLP challenge. Therefore, only one forking is involved leading to a much tighter reduction than $\mathcal{B}_2$. The details follow.

Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. $\mathcal{R}_2$ sets $\mathsf{mpk} := (\mathbb{G}, p, g, g^\alpha, \mathrm{H}, \mathrm{G})$ as public parameters and releases it to $\mathcal{A}$. Note that $\mathcal{R}_2$ does not know the *master* secret key msk, which is α, the solution to the DLP challenge. The hash functions H and G are modelled as random oracles. This is done with the aid of two tables, $\mathfrak{L}_\mathrm{H}$ and $\mathfrak{L}_\mathrm{G}$.

3.2.1 Handling the Queries

H-oracle Query. $\mathfrak{L}_\mathrm{H}$ contains tuples of the form

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

Here, (R, id) is the query to the H-oracle and c the corresponding output. The y-field stores either the corresponding component of the secret key for id or '$\perp$' if the field is invalid. A random oracle query $\mathrm{H}(R, \mathsf{id})$ is fresh if there exists no tuple $\langle R_i, \mathsf{id}_i, c_i, y_i \rangle$ in $\mathfrak{L}_\mathrm{H}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (R_i = R)$. If such a tuple exists, then the oracle has to return c_i as the output. We now explain how the H-oracle queries are answered.

$\mathrm{H}(R, \mathsf{id})$:– The query may be
(i) H_1, Explicit query made by $\mathcal{A}$:– In this case $\mathcal{R}_2$ returns $c \in_R \mathbb{Z}_p$ as the output. $\langle R, \mathsf{id}, c, \perp \rangle$ is added to $\mathfrak{L}_\mathrm{H}$.
(ii) H_2, Implicit query by $\mathcal{R}_2$ in order to answer an extract query made by $\mathcal{A}$:– See *Extract* (ii) on how to program the random oracle in this situation.

G-oracle Query. $\mathfrak{L}_\mathrm{G}$ has the same structure as in $\mathcal{R}_1$ (See §3.1.1). The queries to G-oracle are handled as shown below.

$\mathrm{G}(\mathsf{id}, A, m)$:– $\mathcal{R}_2$ returns $d \in_R \mathbb{Z}_p$ as the output. $\langle \mathsf{id}, A, m, d \rangle$ is added to $\mathfrak{L}_\mathrm{G}$.

Signature and Extract Queries. Since $\mathcal{R}_2$ does not know the *master* secret key α, it has to use the algebraic technique used in $\mathcal{R}_1$ to come up with the secret key corresponding to an identity. The choice of R and c enables it to give the secret key. The circularity involved in this choice is resolved by programming the H-oracle appropriately. Signature queries are answered by generating the usk as in the extract query, followed by calling $\mathcal{S}$.

Extract query. $\mathcal{O}_\varepsilon(\mathsf{id})$:–
(i) If there exists a tuple $\langle R_i, \mathsf{id}_i, c_i, y_i \rangle$ in $\mathfrak{L}_\mathrm{H}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (y_i \neq \perp)$, $\mathcal{R}_2$ returns $\mathsf{usk} := (y_i, R_i)$ as the secret key.
(ii) Otherwise, $\mathcal{R}_2$ chooses $c, y \in_R \mathbb{Z}_p$, sets $R := (g^\alpha)^{-c} g^y$ and adds $\langle R, \mathsf{id}, c, y \rangle$ to $\mathfrak{L}_\mathrm{H}$(*Deferred case* H_2). It returns $\mathsf{usk} := (y, R)$ as the secret key.

Signature query. $\mathcal{O}_s(\mathsf{id}, m)$:–
(i) If there exists a tuple $\langle R_i, \mathsf{id}_i, c_i, y_i \rangle$ in $\mathfrak{L}_\mathrm{H}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (y_i \neq \perp)$, then $\mathsf{usk} = (y_i, R_i)$. $\mathcal{R}_2$ now uses the knowledge of usk to run $\mathcal{S}$ and returns the signature.
(ii) Otherwise, $\mathcal{R}_2$ generates the usk as in *Extract*(ii) and runs $\mathcal{S}$ to return the signature.

We conclude the queries section with the remark that $\mathcal{R}_2$ never aborts during the query stage.

$$\begin{tikzcd}
 & & & & & \mathsf{Q}^0_{I_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^0_{\gamma} \arrow[r, "s^0_{\gamma}"] & \hat{\sigma}_0 = (\hat{A}, \hat{b}_0, \hat{R}) \text{ [Run 0]} \\
\mathsf{Q}^0_1 \arrow[r, "s^0_1"] & \mathsf{Q}^0_2 \arrow[r, dotted, no head] & \mathsf{Q}^0_{J_0} \arrow[r, "s^0_{J_0}"] & \mathsf{Q}^0_{J_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^0_{I_0} \arrow[ur, "s^0_{I_0}"] \arrow[dr, "s^1_{I_0}"'] & & & \\
 & & & & & \mathsf{Q}^1_{I_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^1_{\gamma} \arrow[r, "s^1_{\gamma}"'] & \hat{\sigma}_1 = (\hat{A}, \hat{b}_1, \hat{R}) \text{ [Run 1]}
\end{tikzcd}$$

$$\mathsf{Q}^0_{J_0} = \mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$$
$$\mathsf{Q}^0_{I_0} = \mathrm{H}(\hat{R}, \hat{\mathsf{id}})$$

Fig. 3. Successful oracle replay attack by $\mathcal{R}_2$.

3.2.2 Solving the DLP $\mathcal{R}_2$ now uses the multiple-forking algorithm $\mathcal{M}_{\mathcal{Y},1}$ (see *Appendix B.2* for details on the working of $\mathcal{M}_{\mathcal{Y},n}$) to solve the DLP challenge. It runs $\mathcal{M}_{\mathcal{Y},1}$ on mpk, with both H and G-oracle involved in the replay attack. If $\mathcal{M}_{\mathcal{Y},1}$ fails, $\mathcal{R}_2$ *aborts* ($\mathsf{abort}_{2,1}$). On the other hand, if $\mathcal{M}_{\mathcal{Y},1}$ is successful, $\mathcal{R}_2$ gets two valid forgeries

$$\hat{\sigma}_0 = (\hat{A}, \hat{b}_0, \hat{R}) \text{ and } \hat{\sigma}_1 = (\hat{A}, \hat{b}_1, \hat{R}) \tag{16}$$

on $(\hat{\mathsf{id}}, \hat{m})$, as illustrated in *Figure 3*, with

$$\hat{b}_0 = \hat{a} + (\hat{r} + \alpha c_0)\hat{d} \text{ and } \hat{b}_1 = \hat{a} + (\hat{r} + \alpha c_1)\hat{d}, \tag{17}$$

where $\hat{a} := \log_g^{\mathbb{G}} \hat{A}$ and $\hat{r} := \log_g^{\mathbb{G}} \hat{R}$. Note that c_0 and c_1 represent the value of $\mathrm{H}(\hat{R}, \hat{\mathsf{id}})$ in the two runs, i.e., $c_0 = s^0_{I_0}$ and $c_1 = s^1_{I_0}$. The event F guarantees that $\mathcal{A}$ makes the G-oracle call $\mathsf{Q}^0_{J_0} = \mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$, before the H-oracle call $\mathsf{Q}^0_{I_0} = \mathrm{H}(\hat{R}, \hat{\mathsf{id}})$. Finally it outputs the solution to the DLP instance,

$$\alpha = \frac{\hat{b}_0 - \hat{b}_1}{\hat{d}(c_0 - c_1)}. \tag{18}$$

Structure of the forgeries. Now we justify the structure of the component b of the forgeries given in (17). 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 $\mathsf{usk} = (y, R)$, 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 α, and such forgeries are of no use to $\mathcal{R}_2$. But the event $\bar{\mathrm{E}}$ 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:

$$b = a + (r + \alpha c)d. \tag{19}$$

We conclude with the remark that the event $\mathsf{abort}_{2,1}$ does not occur if the multiple-forking algorithm is successful (let this probability be mfrk). Therefore

$$\Pr[\neg \mathsf{abort}_{2,1}] = \mathsf{mfrk}. \tag{20}$$

3.2.3 Analysis The only abort involved in $\mathcal{R}_2$ is $\mathsf{abort}_{2,1}$, which occurs when $\mathcal{M}_{\mathcal{Y},1}$ fails. Therefore $\mathcal{R}_2$ is successful if $\mathcal{M}_{\mathcal{Y},1}$ is and from (20) we have

$$\epsilon_2 = \Pr[\neg \mathsf{abort}_{2,1}] = \mathsf{mfrk}.$$

We denote the probability with which $\mathcal{M}_{\mathcal{Y},1}$ is successful during $\mathsf{Run\ 0}$ as acc_2. Since there is no abort involved during query phase, $\mathcal{M}_{\mathcal{Y},1}$ is successful during $\mathsf{Run\ 0}$ if $\mathcal{A}$ produces a valid forgery, i.e. $\mathsf{acc}_2 = \epsilon$. Applying (26) from the multiple-forking lemma with $n := 1$, $\gamma := q_{\mathrm{H}} + q_{\mathrm{G}}$,[10] and $|\ \mathbb{S}\ | = p$, we have

$$\begin{aligned}\epsilon_2 = \mathsf{mfrk} &\geq \mathsf{acc}_2 \cdot \left(\frac{\mathsf{acc}_2}{(q_{\mathrm{H}} + q_{\mathrm{G}})^2} - \frac{1}{p}\right) \\ &\geq \epsilon \left(\frac{\epsilon}{(q_{\mathrm{H}} + q_{\mathrm{G}})^2} - \frac{1}{p}\right).\end{aligned}$$

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

3.3 Reduction $\mathcal{R}_3$

As mentioned in the previous sections, the approach used in $\mathcal{R}_3$ is the same as in the reduction $\mathcal{B}_2$ in [11].

Let $\varDelta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. $\mathcal{R}_3$ sets $\mathsf{mpk} := (\mathbb{G}, p, g, g^\alpha, \mathrm{H}, \mathrm{G})$ as the public parameters and releases it to $\mathcal{A}$. As in $\mathcal{R}_2$, $\mathcal{R}_3$ does not know the master secret msk, which is α. The hash functions H and G are modelled as random oracles. This is done with the aid of two tables, $\mathfrak{L}_{\mathrm{H}}$ and $\mathfrak{L}_{\mathrm{G}}$.

3.4 Handling the Queries

The queries are handled in the same way as in $\mathcal{R}_2$. So we refer to §3.2.1 for details.

3.5 Solving the DLP

“

$$\begin{tikzcd}
 & & & & & \mathsf{Q}^0_{I_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^0_\gamma \arrow[r, "s^0_\gamma"] & \hat{\sigma}_0 = (\hat{A}_0, \hat{b}_0, \hat{R})\ [\mathsf{Run\ 0}] \\
 & & & \mathsf{Q}^0_{J_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^0_{I_0} \arrow[ur, "s^0_{I_0}"'] \arrow[dr, "s^1_{I_0}"] & & & \\
 & & & & & \mathsf{Q}^1_{I_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^1_\gamma \arrow[r, "s^1_\gamma"] & \hat{\sigma}_1 = (\hat{A}_0, \hat{b}_1, \hat{R})\ [\mathsf{Run\ 1}] \\
\mathsf{Q}^0_1 \arrow[r, "s^0_1"] & \mathsf{Q}^0_2 \arrow[r, dotted, no head] & \mathsf{Q}^0_{J_0} \arrow[uur, "s^0_{J_0}"] \arrow[ddr, "s^2_{J_0}"'] & & & & & \\
 & & & & & \mathsf{Q}^2_{I_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^2_\gamma \arrow[r, "s^2_\gamma"] & \hat{\sigma}_2 = (\hat{A}_1, \hat{b}_2, \hat{R})\ [\mathsf{Run\ 2}] \\
 & & & \mathsf{Q}^2_{J_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^2_{I_0} \arrow[ur, "s^2_{I_0}"'] \arrow[dr, "s^3_{I_0}"] & & & \\
 & & & & & \mathsf{Q}^3_{I_0+1} \arrow[r, dotted, no head] & \mathsf{Q}^3_\gamma \arrow[r, "s^3_\gamma"] & \hat{\sigma}_3 = (\hat{A}_1, \hat{b}_3, \hat{R})\ [\mathsf{Run\ 3}]
\end{tikzcd}$$

$$\mathsf{Q}^0_{J_0} = \mathrm{H}(\hat{R}, \hat{\mathsf{id}})$$
$$\mathsf{Q}^0_{I_0} = \mathrm{G}(\hat{\mathsf{id}}, \hat{A}_0, \hat{m})\ ,\ \mathsf{Q}^2_{I_0} = \mathrm{G}(\hat{\mathsf{id}}, \hat{A}_1, \hat{m})$$

Fig. 4. Successful oracle replay attack by $\mathcal{R}_3$.

[10] In the analysis of $\mathcal{B}_2$ in [11], γ was assumed to be $q_{\mathrm{H}} \cdot q_{\mathrm{G}}$. However, γ 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 $q_{\mathrm{H}} \cdot q_{\mathrm{G}}$.

$\mathcal{R}_3$ now uses the multiple-forking algorithm $\mathcal{M}_{\mathcal{Y},3}$, to solve the DLP challenge. It runs $\mathcal{M}_{\mathcal{Y},3}$ on mpk, with both H and G-oracle involved in the replay attack. If $\mathcal{M}_{\mathcal{Y},3}$ fails, $\mathcal{R}_3$ *aborts* ($\mathsf{abort}_{3,1}$). On the other hand, if $\mathcal{M}_{\mathcal{Y},3}$ is successful, as illustrated in *Figure 4*, $\mathcal{R}_3$ gets four valid forgeries

$$\begin{aligned} \hat{\sigma}_0 = (\hat{A}_0, \hat{b}_0, \hat{R}) \quad &, \quad \hat{\sigma}_1 = (\hat{A}_0, \hat{b}_1, \hat{R}), \\ \hat{\sigma}_2 = (\hat{A}_1, \hat{b}_2, \hat{R}) \quad &\text{and} \quad \hat{\sigma}_3 = (\hat{A}_1, \hat{b}_3, \hat{R}) \end{aligned} \tag{21}$$

with $\hat{\sigma}_0$ and $\hat{\sigma}_1$ on $(\hat{\mathsf{id}}, \hat{m}_0)$ and $\hat{\sigma}_2$ and $\hat{\sigma}_3$ on $(\hat{\mathsf{id}}, \hat{m}_1)$, where

$$\begin{aligned} \hat{b}_0 = \hat{a}_0 + (\hat{r} + \alpha c_0) d_0 \quad &, \quad \hat{b}_1 = \hat{a}_0 + (\hat{r} + \alpha c_0) d_1, \\ \hat{b}_2 = \hat{a}_1 + (\hat{r} + \alpha c_1) d_2 \quad &\text{and} \quad \hat{b}_3 = \hat{a}_1 + (\hat{r} + \alpha c_1) d_3, \end{aligned} \tag{22}$$

where $\hat{r} := \log_g^{\mathbb{G}} \hat{R}$, $\hat{a}_i := \log_g^{\mathbb{G}} \hat{A}_i$. Note that c_0 and c_1 represent the value of $\mathrm{H}(\hat{R}, \hat{\mathsf{id}})$ in the H-oracle forks; d_0 and d_1 represent the value of $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}_0, \hat{m})$ in the first two runs; d_2 and d_3 represent the value of $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}_1, \hat{m})$ in the last two runs. Finally it outputs the solution to the DLP challenge,

$$\alpha = \frac{(\hat{b}_0 - \hat{b}_1)(d_2 - d_3) - (\hat{b}_2 - \hat{b}_3)(d_0 - d_1)}{(c_0 - c_1)(d_0 - d_1)(d_2 - d_3)}. \tag{23}$$

We conclude with the remark that the event $\mathsf{abort}_{3,1}$ does not occur if the multiple-forking algorithm is successful (let this probability be mfrk). Therefore

$$\Pr\left[\neg \mathsf{abort}_{3,1}\right] = \mathsf{mfrk}. \tag{24}$$

3.6 Analysis

As in $\mathcal{R}_2$, the only abort involved in $\mathcal{R}_3$ is $\mathsf{abort}_{3,1}$, which occurs when $\mathcal{M}_{\mathcal{Y},3}$ fails. Therefore $\mathcal{R}_3$ is successful if $\mathcal{M}_{\mathcal{Y},3}$ is and from (24) we have

$$\epsilon_3 = \Pr\left[\neg \mathsf{abort}_{3,1}\right] = \mathsf{mfrk}.$$

We denote the probability with which $\mathcal{M}_{\mathcal{Y},3}$ is successful during the first run as acc_3. Since there is no abort involved during query phase, $\mathcal{M}_{\mathcal{Y},3}$ is successful during the first run if $\mathcal{A}$ produces a valid forgery, i.e. $\mathsf{acc}_3 = \epsilon$. Applying (26) from the multiple-forking lemma with $n = 3$, $\gamma = q_{\mathrm{H}} + q_{\mathrm{G}}$ and $|\ \mathbb{S}\ | = p$, we have

$$\begin{aligned} \epsilon_3 = \mathsf{mfrk} &\geq \mathsf{acc}_3 \cdot \left(\frac{\mathsf{acc}_3^3}{(q_{\mathrm{H}} + q_{\mathrm{G}})^6} - \frac{3}{p} \right) \\ &\geq \epsilon \left(\frac{\epsilon^3}{(q_{\mathrm{H}} + q_{\mathrm{G}})^6} - \frac{3}{p} \right). \end{aligned}$$

Time Analysis. The time taken by $\mathcal{R}_3$ is easily seen to be bounded by $t_3 \leq t + 4(2q_\varepsilon + 3q_s)\tau$.

3.7 A Comparison with the Original Reduction.

Recall that we replaced the reduction $\mathcal{B}_1$ in the original security argument with the new reduction $\mathcal{R}_1$. Similarly, $\mathcal{B}_2$ 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 10*). In comparison, the security degradation involved in $\mathcal{R}_2$ and $\mathcal{R}_3$ is of order $\mathcal{O}\left((q_{\mathrm{G}} + q_{\mathrm{H}})^2\right)$ and $\mathcal{O}\left((q_{\mathrm{G}} + q_{\mathrm{H}})^6\right)$ respectively.

Original reductions [11]	$\mathcal{B}_1$	$\mathcal{B}_2$	
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	$\mathcal{O}\left(q_{\mathrm{G}}q_{\varepsilon}\right)$	$\mathcal{O}\left((q_{\mathrm{G}}+q_{\mathrm{H}})^2\right)$	$\mathcal{O}\left((q_{\mathrm{G}}+q_{\mathrm{H}})^6\right)$

Table 1. A comparison of degradation in the original [11] and the new security argument.

4 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 [11]. 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.

References

1. 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, 2 and 20.)
2. 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, 2, 21 and 22.)
3. 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 pages 1 and 20.)
4. 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, 2, 21, 22 and 24.)
5. 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 2 and 10.)
6. 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 1.)
7. 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, 2 and 9.)
8. 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 1.)
9. 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 1.)
10. 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 2.)

11. 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, 2, 3, 4, 7, 8, 13, 14, 16, 18 and 25.)
12. 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 20.)
13. 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 1.)
14. Javier Herranz. Deterministic identity-based signatures for partial aggregation. *The Computer Journal*, 49(3):322–330, 2005. (Cited on page 1.)
15. 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 1.)
16. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13:361–396, 2000. (Cited on pages 1 and 21.)
17. V. Radhakishan and S. Selvakumar. Prevention of man-in-the-middle attacks using id based signatures. In *Networking and Distributed Computing (ICNDC), 2011 Second International Conference on*, pages 165 –169, sept. 2011. (Cited on page 1.)
18. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer Berlin / Heidelberg, 1990. (Cited on page 2.)
19. 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 1.)
20. 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 2.)
21. 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 1.)

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* probabilistic polynomial-time *algorithms* $\{\mathcal{G}, \mathcal{E}, \mathcal{S}, \mathcal{V}\}$ *described below.*

Set-up. $\mathcal{G}(\kappa)$:– It takes as input the security parameter κ. It outputs the *master* secret key msk and public parameters mpk.

$$(\mathsf{msk}, \mathsf{mpk}) \overset{\$}{\leftarrow} \mathcal{G}(\kappa)$$

Key Extraction. $\mathcal{E}(\mathsf{id}, \mathsf{msk}, \mathsf{mpk})$:– It takes as input the user's identity id, the *master* secret key msk and public parameters mpk to generate the *user* secret key usk.

$$\mathsf{usk} \overset{\$}{\leftarrow} \mathcal{E}(\mathsf{id}, \mathsf{msk}, \mathsf{mpk})$$

Signing. $\mathcal{S}(\mathsf{id}, m, \mathsf{usk}, \mathsf{mpk})$:– It takes as input the user's identity id, a message m, public parameters mpk and the *user* secret key usk to generate a signature σ.

$$\sigma \overset{\$}{\leftarrow} \mathcal{S}(\mathsf{id}, m, \mathsf{usk}, \mathsf{mpk})$$

Verification. $\mathcal{V}(\sigma, \mathsf{id}, m, \mathsf{mpk})$:– It takes as input a signature σ, a message m, an identity id and public parameters mpk. It outputs the result which is 1 if σ is a valid signature on (id, m) or 0 if the signature is invalid.

$$\mathsf{result} \leftarrow \mathcal{V}(\sigma, \mathsf{id}, m, \mathsf{mpk})$$

The standard *correctness* condition –

$$1 \leftarrow \mathcal{V}(\mathcal{S}(\mathsf{id}, m, \mathsf{usk}, \mathsf{mpk}), \mathsf{id}, m, \mathsf{mpk}),$$

where $(\mathsf{msk}, \mathsf{mpk}) \xleftarrow{\$} \mathcal{G}(\kappa)$ and $\mathsf{usk} \xleftarrow{\$} \mathcal{E}(\mathsf{id}, \mathsf{msk}, \mathsf{mpk})$, should be satisfied.

A.2 Security Model

Goldwasser et al. [12] 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 [1].

Definition 2 (EU-ID-CMA Game[11]). *The security of an IBS scheme in the* EU-ID-CMA *model is argued 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 public parameters mpk and the *master* secret key msk. $\mathcal{A}$ is given public parameters but the master secret key is kept by $\mathcal{C}$.
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.** $\mathcal{O}_\varepsilon(\mathsf{id})$:– $\mathcal{A}$ asks for the secret key of a user with identity id. If there has already been an extract query on id, $\mathcal{C}$ returns the *user* secret key that was generated during the earlier query. Otherwise, $\mathcal{C}$ uses the knowledge of msk to run $\mathcal{E}$ and generate the *user* secret key usk, which is passed on to $\mathcal{A}$.
- **Signature Query.** $\mathcal{O}_s(\mathsf{id}, m)$:– $\mathcal{A}$ asks for the signature of a user with identity id on a message m. $\mathcal{C}$ first generates a *user* secret key for id, as in the extract query. Next, it uses the knowledge of usk to run $\mathcal{S}$ and generates a signature σ, which is passed to $\mathcal{A}$.

Forgery. $\mathcal{A}$ outputs a signature $\hat{\sigma}$ on an identity $\hat{\mathsf{id}}$ and a message $\hat{m}$, and wins the game if

1. $\hat{\sigma}$ is a valid signature on $\hat{m}$ by $\hat{\mathsf{id}}$.
2. $\mathcal{A}$ has not made an extract query on $\hat{\mathsf{id}}$.
3. $\mathcal{A}$ has not made a signature query on $(\hat{\mathsf{id}}, \hat{m})$.

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

$$\mathrm{Adv}_{\mathcal{A}}^{\mathtt{EU-ID-CMA}} = \Pr\left[(\mathsf{msk}, \mathsf{mpk}) \xleftarrow{\$} \mathcal{G}(\kappa); (\hat{\sigma}, \hat{\mathsf{id}}, \hat{m}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_\varepsilon, \mathcal{O}_s}(\mathsf{mpk}) \mid 1 \leftarrow \mathcal{V}(\hat{\sigma}, \hat{\mathsf{id}}, \hat{m}, \mathsf{mpk})\right]$$

provided $\hat{\sigma}$ is a valid forgery on $(\hat{\mathsf{id}}, \hat{m})$. 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 ϵ in the above game, runs in time at most t and makes at most q_ε and q_s extract and signature queries respectively. If the security argument uses the random oracle methodology [3], the adversary is also allowed to make queries to the random oracle(s).

A.3 Discrete-Log Assumption

Definition 3. *Consider a group $\mathbb{G}$ of prime order p, generated by g. The* discrete-log problem *(DLP) in $\mathbb{G}$ is to find α given g^α, where $\alpha \in_R \mathbb{Z}_p$. An adversary $\mathcal{A}$ has advantage ϵ in solving the DLP if*

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

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

[11] The security game in [1], i.e. $\mathrm{Exp}_{\mathrm{IBS,F}}^{\text{uf-cma}}$, 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.

B Forking Lemma

Pointcheval-Stern introduced the forking lemma [16] 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 [2] and the multiple-forking lemma [4]. 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.

B.1 General Forking Lemma

We first reproduce the general forking algorithm from [2] and then explain its working. This is followed by the statement of general forking lemma. We use slightly different notations to maintain uniformity.

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

```
Algorithm $\mathcal{F}_\mathcal{Y}(x)$
  Pick coins $\rho$ for $\mathcal{Y}$ at random
  $s_1^0, \ldots, s_\gamma^0 \in_R \mathbb{S}$; $(I_0, \sigma_0) \xleftarrow{\$} \mathcal{Y}(x, s_1^0, \ldots, s_\gamma^0; \rho)$ [Run 0]
  if $(I_0 = 0)$ then
    return $(0, \perp, \perp)$
  end if
  $s_{I_0}^1, \ldots, s_\gamma^1 \in_R \mathbb{S}$; $(I_1, \sigma_1) \xleftarrow{\$} \mathcal{Y}(x, s_1^0, \ldots, s_{I_0-1}^0, s_{I_0}^1, \ldots, s_\gamma^1; \rho)$ [Run 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 $(0, \perp, \perp)$
  end if
```

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

Fig. 5. A successful oracle replay attack by $\mathcal{F}_\mathcal{Y}$.

Working. The reductions in our security argument use $\mathcal{F}_\mathcal{Y}$, as a black-box, to secure the forgeries required to solve the underlying hard problem. $\mathcal{F}_\mathcal{Y}$, in turn, uses $\mathcal{Y}$ to (i) simulate the protocol environment for the adversary and (ii) launch the *oracle replay attacks.*

$\mathcal{Y}$ takes as input: (i) x, the parameters passed to $\mathcal{F}_\mathcal{Y}$, (ii) $\mathbb{T} := \{s_1, \ldots, s_\gamma\}$, the set of answers to the random oracle queries and (iii) ρ, the randomness associated with the protocol, excluding $\mathbb{T}$. Here, γ is the upper bound on the number of queries to the random oracle. At the end of a simulation run, it returns: σ, the forgery produced by the adversary and I, the index to the oracle query which was used by the adversary to forge.

$\mathsf{Run\ 0}$. $\mathcal{F}_\mathcal{Y}$ picks: the randomness ρ and the responses to the random oracle queries $\{s_1^0, \ldots, s_\gamma^0\}$. Then it runs $\mathcal{Y}$ on $(x, s_1^0, \ldots, s_\gamma^0, \rho)$ to get (I_0, σ_0). This constitutes the *first forgery.*

$\mathsf{Run\ 1}$. $\mathcal{F}_\mathcal{Y}$ picks $\{s_{I_0}^1, \ldots, s_\gamma^1\}$ and re-runs $\mathcal{Y}$ on $(x, s_1^0, \ldots, s_{I_0-1}^0, s_{I_0}^1, \ldots, s_\gamma^1, \rho)$ to get (I_1, σ_1). This constitutes $\mathcal{F}_\mathcal{Y}$ launching an *oracle replay attack* by re-winding the input tape to $\mathsf{Q}_{I_0}^0$ and then re-running $\mathcal{A}$ using a different random oracle.

$\mathcal{F}_\mathcal{Y}$ now hopes that $\mathcal{A}$, during $\mathsf{Run\ 1}$, uses the same random oracle index to produce its output, i.e. $I_1 = I_0$. This results in $\mathcal{A}$ committing to an input, but with different random oracle outputs as shown in *Figure 5*. Finally, $\mathcal{F}_\mathcal{Y}$ returns $(1, \sigma_0, \sigma_1)$ as an indication of a successful replay attack . On the other hand, if $I_1 \neq I_0$, $\mathcal{F}_\mathcal{Y}$ fails and returns $(0, \perp, \perp)$ as an indicating failure. The probability with which $\mathcal{F}_\mathcal{Y}$ succeeds, in terms of the adversarial advantage, is governed by the forking lemma given below.

Lemma 1 (General Forking Lemma [2]). *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

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

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

then

$$\mathsf{gfrk} \geq \mathsf{acc} \cdot \left(\frac{\mathsf{acc}}{\gamma} - \frac{1}{\mid \mathbb{S} \mid}\right). \tag{25}$$

B.2 Multiple-Forking Lemma

We first reproduce the multiple-forking algorithm from [4], followed by the statement of multiple-forking lemma. Again, we use slightly different notations to maintain uniformity.

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

```
Algorithm $\mathcal{M}_{\mathcal{Y},n}(x)$
  Initialise an empty array $\mathsf{results}[0,\ldots,n]$
  Pick coins $\rho$ for $\mathcal{Y}$ at random
  $s_1^0,\ldots,s_\gamma^0 \in_R \mathbb{S}$; $(I_0,J_0,\sigma_0) \xleftarrow{\$} \mathcal{Y}(x,s_1^0,\ldots,s_\gamma^0;\rho)$ [Run 0]
  if $(I_0 = 0 \vee J_0 = 0)$ then
    return $(0,\mathsf{results})$
  end if
  $s_{I_0}^1,\ldots,s_\gamma^1 \in_R \mathbb{S}$; $(I_1,J_1,\sigma_1) \xleftarrow{\$} \mathcal{Y}(x,s_1^0,\ldots,s_{I_0-1}^0,s_{I_0}^1,\ldots,s_\gamma^1;\rho)$ [Run 1]
  if $((I_1,J_1) \neq (I_0,J_0) \vee s_{I_0}^1 = s_{I_0}^0)$ then
    return $(0,\mathsf{results})$
  end if
  $i \leftarrow 2$
  while $(i < n)$ do
    $s_{J_0}^i,\ldots,s_\gamma^i \in_R \mathbb{S}$; $(I_i,J_i,\sigma_i) \xleftarrow{\$} \mathcal{Y}(x,s_1^0,\ldots,s_i^0,s_{J_0}^i,\ldots,s_\gamma^i;\rho)$ [Run i]
    if $((I_i,J_i) \neq (I_0,J_0) \vee s_{J_0}^i = s_{J_0}^{i-1})$ then
      return $(0,\mathsf{results})$
    end if
    $s_{I_0}^{i+1},\ldots,s_\gamma^{i+1} \in_R \mathbb{S}$;
    $(I_{i+1},J_{i+1},\sigma_{i+1}) \xleftarrow{\$} \mathcal{Y}(x,s_1^0,\ldots,s_i^0,s_{J_0}^i,\ldots,s_{I_0-1}^i,s_{I_0}^{i+1},\ldots,s_\gamma^{i+1};\rho)$ [Run i + 1]
    if $((I_{i+1},J_{i+1}) \neq (I_0,J_0) \vee s_{J_0}^{i+1} = s_{J_0}^i)$ then
      return $(0,\mathsf{results})$
    end if
    $i \leftarrow i + 2$
  end while
  for $i := 0$ to $n$ do
    $\mathsf{results}[i] \leftarrow \sigma_i$
  end for
  return (1, results)
```

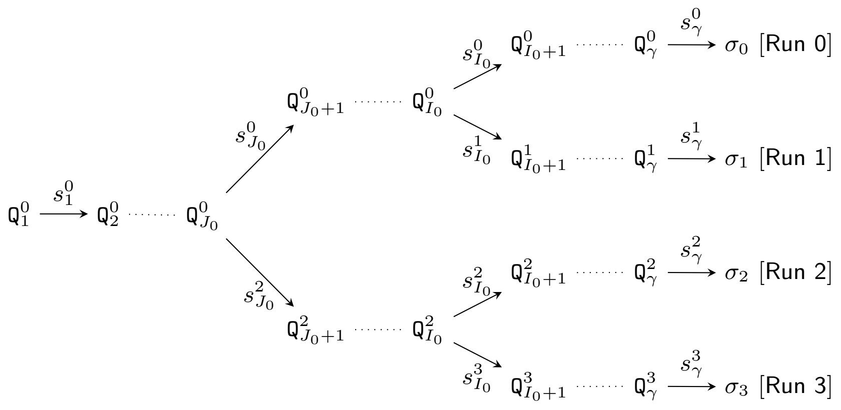
$\mathsf{Q}_1^0 \xrightarrow{s_1^0} \mathsf{Q}_2^0 \cdots \mathsf{Q}_{J_0}^0$
$s_{J_0}^0$
$\mathsf{Q}_{J_0+1}^0 \cdots \mathsf{Q}_{I_0}^0$
$s_{I_0}^0$ $\mathsf{Q}_{I_0+1}^0 \cdots \mathsf{Q}_\gamma^0 \xrightarrow{s_\gamma^0} \sigma_0$ [Run 0]
$s_{I_0}^1$ $\mathsf{Q}_{I_0+1}^1 \cdots \mathsf{Q}_\gamma^1 \xrightarrow{s_\gamma^1} \sigma_1$ [Run 1]
$s_{J_0}^2$
$\mathsf{Q}_{J_0+1}^2 \cdots \mathsf{Q}_{I_0}^2$
$s_{I_0}^2$ $\mathsf{Q}_{I_0+1}^2 \cdots \mathsf{Q}_\gamma^2 \xrightarrow{s_\gamma^2} \sigma_2$ [Run 2]
$s_{I_0}^3$ $\mathsf{Q}_{I_0+1}^3 \cdots \mathsf{Q}_\gamma^3 \xrightarrow{s_\gamma^3} \sigma_3$ [Run 3]

Fig. 6. A successful oracle replay attack by $\mathcal{M}_{\mathcal{Y},3}$.

Working. As in the case of $\mathcal{F}_\mathcal{Y}$, the reductions in our security argument use $\mathcal{M}_{\mathcal{Y},n}$, as a blackbox, to secure the forgeries required to solve the underlying hard problem. The role of $\mathcal{Y}$ remains as described in $\mathcal{F}_\mathcal{Y}$. However, as two random oracles – denoted by H and G – are now involved in the replay, $\mathcal{F}_\mathcal{Y}$ now takes answers to both the random oracles as it input. We explain the working of $\mathcal{M}_{\mathcal{Y},n}$ for the case of $n := 3$ (which corresponds to the reduction $\mathcal{R}_3$ in our security argument).

$\mathsf{Run\ 0}$. $\mathcal{M}_{\mathcal{Y},3}$ picks the randomness ρ and the responses to the random oracle queries $\{s_1^0, \ldots, s_\gamma^0\}$. Then it runs $\mathcal{Y}$ on $(x, s_0^0, \ldots, s_\gamma^0, \rho)$ to get (I_0, J_0, σ_0). Without any loss of generality, we may assume that the index I_0 involved the H-oracle and J_0, the G-oracle. This constitutes the *first forgery.*

$\mathsf{Run\ 1}$. Now, $\mathcal{M}_{\mathcal{Y},3}$ picks $\{s_{I_0}^1, \ldots, s_\gamma^1\}$ and re-runs $\mathcal{Y}$ on $(\mathsf{mpk}, s_1^0, \ldots, s_{I_0-1}^0, s_{I_0}^1, \ldots, s_\gamma^1, \rho)$ to get (I_1, J_1, σ_1). This constitutes $\mathcal{F}_\mathcal{Y}$ launching the *first oracle replay attack* by re-winding the input tape to $\mathsf{Q}_{I_0}^0$ and then re-running $\mathcal{A}$ using a different G-oracle.

$\mathcal{F}_\mathcal{Y}$ now hopes that $\mathcal{A}$, during $\mathsf{Run\ 1}$, forges on the "correct" indices, i.e. $I_1 = I_0$ and $J_1 = J_0$. If $\mathcal{A}$ does not, $\mathcal{M}_{\mathcal{Y},3}$ fails. Otherwise, we have $\mathcal{A}$ committing to an input, but with different G-oracle answers $s_{I_0}^0$ and $s_{I_0}^1$. In a similar manner, $\mathcal{M}_{\mathcal{Y},3}$ launches two more oracle replay attacks.

$\mathsf{Run\ 2}$. The *second oracle replay attack* is launched by re-winding the input tape to $\mathsf{Q}_{J_0}^0$ and then re-running $\mathcal{A}$ using a different H-oracle. $\mathcal{M}_{\mathcal{Y},3}$ is successful during this run if the forgery is on the "correct" indices, i.e., $J_2 = J_0$ and $I_2 = I_0$.

$\mathsf{Run\ 3}$. The *third (and final) oracle replay attack* is analogous to the first replay attack in the sense that the re-winding is done to the point where the G-oracle call was made in the previous run.

If $\mathcal{M}_{\mathcal{Y},3}$ is successful at the end of $\mathsf{Run\ 3}$ – i.e., it is successful in all the four runs – it returns $(1, \mathsf{results})$ indicating a successful execution. The probability with which $\mathcal{M}_{\mathcal{Y},n}$ succeeds, in terms of the adversarial advantage, is governed by the multiple-forking lemma given below.

Lemma 2 (Multiple-Forking Lemma [4]). *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

$$\mathsf{acc} := \Pr\left[x \xleftarrow{\$} \mathcal{G}_I; s_1^0, \ldots, s_\gamma^0 \in_R \mathbb{S}; (I_0, J_0, \sigma) \xleftarrow{\$} \mathcal{Y}(x, s_1^0, \ldots, s_\gamma^0) \mid I_0 \geq 1 \wedge J_0 \geq 1\right] \quad \textit{and}$$

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

then

$$\mathsf{mfrk} \geq \mathsf{acc} \cdot \left(\frac{\mathsf{acc}^n}{\gamma^{2n}} - \frac{n}{|\mathbb{S}|}\right). \tag{26}$$

C The Fixed Security Argument

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

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

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

$\mathcal{B}_1.1$ $\mathcal{B}_1$ picks $\hat{i} \in_R \{1, \ldots, q_\mathrm{G}\}$,[12] where q_G is the maximum number of queries that the adversary $\mathcal{A}$ makes to the G-oracle. Let $\hat{\mathsf{id}}$ (the target identity) be the $\hat{i}^{\mathrm{th}}$ distinct identity queried to the G-oracle. Next, $\mathcal{B}_1$ chooses $z \in_R \mathbb{Z}_p$ and sets $(\mathsf{mpk}, \mathsf{msk}) := ((\mathbb{G}, g, p, \mathrm{G}, \mathrm{H}, g^z), z)$, 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 $\mathcal{A}$.

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

$\mathcal{B}_1$.2 Every time $\mathcal{A}$ queries the key extraction oracle $\mathcal{O}_\varepsilon$, for user id, $\mathcal{B}_1$ chooses $c, y \in_R \mathbb{Z}_p$, sets $R := g^{-zc}g^y$ and adds $\langle R, \mathsf{id}, c\rangle$ 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 call to the signature oracle $\mathcal{O}_s$ for (id, m) with $\mathsf{id} \neq \hat{\mathsf{id}}$, $\mathcal{B}_1$ simply computes id's secret key as described in the previous bullet. Then it runs the signing algorithm $\mathcal{S}$ and returns the produced signature to $\mathcal{A}$.

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

$\mathcal{B}_1$.5 $\mathcal{B}_1$ runs the algorithm $\mathcal{M}_{\mathcal{Y},1}(\mathsf{mpk})$ as described in Lemma 1 (§4 in [11]). Here algorithm $\mathcal{Y}$ 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 calls to the random oracles G, H with the target identity $\hat{\mathsf{id}}$.

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

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

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

It takes as argument, the description of a group $(\mathbb{G}, p, g)$ and a challenge g^α with $\alpha \in_R \mathbb{Z}_p$ and outputs the discrete logarithm α. 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 public parameters mpk:=$(\mathbb{G}, p, g, \mathrm{G}, \mathrm{H})$ and $\mathsf{msk} := (g^\alpha)$, 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 $c, y \in_R \mathbb{Z}_q$, sets $R := g^{-\alpha c}g^y$ and adds $\langle R, \mathsf{id}, c\rangle$ to the table $\mathfrak{L}_{\mathrm{H}}$. Then it returns the key (y, R) to $\mathcal{A}$.

$\mathcal{B}_2$.3 When $\mathcal{A}$ makes a call to the signature oracle $\mathcal{O}_s$ with (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, $((\mathsf{id}, g^a, m), d)$ to the table $\mathfrak{L}_{\mathrm{G}}$ and gives the resulting signature to the adversary.

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

(a) If $R_3 = R_2 = R_1 = R_0$ ($\mathsf{Case\ 1}$) then, the signatures will be of the form

$$\begin{aligned} b_0 = \log A_0 + (\log R + c_0\alpha)d_0 \quad &, \quad b_1 = \log A_0 + (\log R + c_0\alpha)d_1, \\ b_2 = \log A_2 + (\log R + c_2\alpha)d_2 \quad &\text{and} \quad b_3 = \log A_2 + (\log R + c_2\alpha)d_3 \end{aligned} \tag{27}$$

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

$$\alpha = \frac{(b_0 - b_1)(d_2 - d_3) - (b_2 - b_3)(d_0 - d_1)}{(c_0 - c_1)(d_0 - d_1)(d_2 - d_3)}. \tag{28}$$

(b) Else, if $A_3 = A_2 = A_1 = A_0$ ($\mathsf{Case\ 2}$) then, the signatures will be of the form

$$\begin{aligned} b_0 = \log A + (\log R_0 + c_0\alpha)d_0 \quad &, \quad b_1 = \log A + (\log R_0 + c_1\alpha)d_0, \\ b_2 = \log A + (\log R_2 + c_2\alpha)d_2 \quad &\text{and} \quad b_3 = \log A + (\log R_2 + c_3\alpha)d_2. \end{aligned} \tag{29}$$

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

$$\alpha = \frac{b_0 - b_1}{d_0(c_0 - c_1)}. \tag{30}$$