Constructing Provably Secure Identity-Based Signature Schemes.

A Thesis

Submitted for the Degree of

Master of Science (Engineering)

in the Faculty of Engineering

by

Chethan Kamath H

Computer Science and Automation
INDIAN INSTITUTE OF SCIENCE
BANGALORE – 560 012, INDIA

MAY 2014

To my Teachers.

I say unto you: one must still have chaos in oneself to be able to give birth to a dancing star.

–Friedrich Nietzsche

Sir Bedevere: There are ways of telling whether she is a witch.
Peasant 1: Are there? Oh well, tell us.
Sir Bedevere: Tell me. What do you do with witches?
Peasant 1: Burn them.
Sir Bedevere: And what do you burn, apart from witches?
Peasant 1: More witches.
Peasant 2: Wood.
Sir Bedevere: Good. Now, why do witches burn?
Peasant 3: ...because they're made of... wood?
Sir Bedevere: Good. So how do you tell whether she is made of wood?
Peasant 1: Build a bridge out of her.
Sir Bedevere: But can you not also build bridges out of stone?
Peasant 1: Oh yeah.
Sir Bedevere: Does wood sink in water?
Peasant 1: No, no, it floats!... It floats! Throw her into the pond!
Sir Bedevere: No, no. What else floats in water?
Peasant 1: Bread.
Peasant 2: Apples.
Peasant 3: Very small rocks.
Peasant 1: Cider.
Peasant 2: Gravy.
Peasant 3: Cherries.
Peasant 1: Mud.
Peasant 2: Churches.
Peasant 3: Lead! Lead!
King Arthur: A Duck.
Sir Bedevere: ...Exactly. So, logically...
Peasant 1: If she weighed the same as a duck... she's made of wood.
Sir Bedevere: And therefore...
Peasant 2: ...A witch!

–Monty Python and the Holy Grail

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my primary advisor, Dr. Sanjit Chatterjee. I owe him not only for helping develop my academic nous, but also for enlightening me on the subtleties of life, in general. Working under his guidance has been an absolute joy! I am also grateful to Prof. C. E. Veni Madhavan, my co-advisor, for his constant support. I am indebted to faculty of mathematics (in particular, Prof. Dilip Patil and Dr. Chandan Saha) who made me appreciate the elegance, the beauty of mathematics once again. Finally, I would like to thank Prof. C. Pandu Rangan for the kind reviews.

I am thankful to my labmates: Vikas, Nitish, Srikanth, Kumar, Srinivas Vivek, Sudarshan, Kabaleesh, the two Siddharthas, Sayantan, Indradeep, Manjunath, Yogesh, Aniket and Nithin, for their support and forbearance; in particular, I am indebted to Vikas–we were partners in the buddy system of academia. I would also like to thank Ashwin, Maria, Ninad, Subramanya (CSA, IISc), Somindu, Sanjay and Rishiraj (ISI, Kolkata) for some insightful discussions. I am also grateful to my batchmates and our football club for some pleasant, lasting memories.

Last but not the least, I would like to thank my family to whom I owe everything.

Notation and Abbreviations

We adopt the notations that is commonly used in the literature (see [AB09]). Sets are denoted by blackboard bold letters in upper case (*e.g.*, $\mathbb{S}, \mathbb{Z}$), algorithms using calligraphic letters in upper case (*e.g.*, $\mathcal{A}$, $\mathcal{B}$) and tables, schemes *etc*. using upper case letters in fraktur (*e.g.*, $\mathfrak{C}$). Events are denoted using typewriter font (*e.g.*, E, $\mathsf{abort}_{1,1}$). We use $\mathbb{Z}^+$ to indicate the set of positive integers. For an event E, the complementary event is denoted by $\neg\mathsf{E}$. In a group, the discrete-log to a generator g is denoted by $\log_g$.

$s \xleftarrow{\mathsf{U}} \mathbb{S}$ denotes picking an element s uniformly at random from the set $\mathbb{S}$. In general, $\{s_1, \ldots, s_n\} \xleftarrow{\mathsf{U}} \mathbb{S}$ denotes picking elements $s_1, \ldots, s_n$ independently and uniformly at random from the set $\mathbb{S}$. In a similar manner, $s \xleftarrow{\$} \mathbb{S}$ and $\{s_1, \ldots, s_n\} \xleftarrow{\$} \mathbb{S}$ denote random sampling, but with some underlying probability distribution on $\mathbb{S}$.

$(y_1, \ldots, y_n) \xleftarrow{\$} \mathcal{A}(x_1, \ldots, x_m)$ denotes a probabilistic algorithm $\mathcal{A}$ which takes as input $(x_1, \ldots, x_m)$ to produce output $(y_1, \ldots, y_n)$. The internal coins ρ of an algorithm, when explicitly given as input, is distinguished from the normal input using a semi-colon, *e.g.*, $y \leftarrow \mathcal{A}(x; \rho)$. Oracle access is indicated using superscripts, *e.g.*, access to an oracle H taking one parameter as inputs is denoted by $y \xleftarrow{\$} \mathcal{A}^{\mathsf{H}(\cdot)}(x)$. The number of dots indicate the number of parameters, *e.g.*, $\mathsf{H}(\cdot, \cdot)$ is an oracle taking two parameters as input. Throughout the thesis, we denote the adversary by $\mathcal{A}$.

In reductionist security arguments, $\Pi \leq_\gamma \mathsf{M}[\mathfrak{P}]$ denotes that Π is γ-reducible to $\mathfrak{P}$: given an adversary against $\mathfrak{P}$ in the security model M, one can construct an algorithm that breaks Π with a degradation of γ. $\Pi \xLeftarrow{\gamma} \mathsf{M}[\mathfrak{P}]$ conveys the same message. A straightforward generalisation is $\mathsf{M}'[\mathfrak{P}'] \leq_\gamma \mathsf{M}[\mathfrak{P}]$: given an adversary against $\mathfrak{P}$ in the security model M, one can construct an algorithm that breaks $\mathfrak{P}'$ in the security model M' with a degradation of γ. (Note that we use long arrows to denote logical implication and short arrows to denote reductions.)

Next, we introduce some notations pertaining to random oracles. The symbol $<$ is used to order the random oracle calls; *e.g.*, $\mathsf{H}(x) < \mathsf{G}(y)$ indicates that the random oracle call $\mathsf{H}(x)$ precedes the random oracle call $\mathsf{G}(y)$. More generally, $\mathsf{H} < \mathsf{G}$ indicates that the target H-oracle call precedes the target G-oracle call. The convention applies to hash functions as well. The symbol, on the other hand, $\prec$ is used to indicate random oracle dependency; *e.g.* $\mathsf{H} \prec \mathsf{G}$ indicates that the random oracle G is dependent on the random oracle H. In the discussion involving the forking algorithms, Q_j^k denotes the j^{th} random oracle query in

round k of simulation.

Finally, in schematics involving IBC, dashed lines indicate an open channel, whereas solid lines indicate a secure channel.

Publications based on this Thesis

1. Sanjit Chatterjee, Chethan Kamath and Vikas Kumar. Galindo-Garcia Identity-Based Signature Revisited. In *Information Security and Cryptology–ICISC'12*, volume 7839 of *Lecture Notes in Computer Science*, pages 456-471, Springer, 2013.

2. Sanjit Chatterjee and Chethan Kamath. From Selective-ID to Full-ID IBS without Random Oracles. To appear in *Security, Privacy and Cryptography–SPACE'13*, volume 8204 of *Lecture Notes in Computer Science*, pages 172-190, Springer, 2013.

Abstract

An identity-based cryptosystem is a public-key system where the public key can be represented by any arbitrary string such as an e-mail address. This notion was introduced by Shamir with the primary goal of simplifying certificate management. In this thesis we study identity-based signatures (IBS)–the identity-based counterpart of digital signatures. In particular, we focus on two techniques–one concrete and the other, generic–used to construct IBS. The concrete scheme we look into is the one proposed by Galindo and Garcia in Africacrypt 2009. This constitutes the primary contribution of the thesis. The generic technique, on the other hand, involves constructing fully-secure IBS given an IBS secure in the selective-identity model.

The Galindo-Garcia IBS (GG-IBS) is derived through a simple and elegant concatenation of two Schnorr signatures. It works in the discrete-log setting but does not require pairing. The security is established through two algorithms (both of) which use the Multiple-Forking (MF) Algorithm to *reduce* the problem of computing the discrete-log to breaking the IBS. The primary contribution of the thesis pertains to the security argument: it turns out that the argument is ridden with flaws and ambiguities and, as a remedy, we provide a new, detailed, security argument. However, the resulting security bound is still quite *loose*, chiefly down to the usage of the MF Algorithm. We explore possible avenues for improving this bound and, to this end, introduce two notions pertaining to random oracles termed *dependency* and *independency*. Incorporating (in)dependency allows us to launch the nested replay attack more effectively than in the MF Algorithm leading to a cleaner, (significantly) tighter security argument for GG-IBS, completing the final piece of the GG-IBS jigsaw.

In the second part of the thesis we look into the notion of *selective-identity* (sID) in the context of IBS. Since its induction, the sID model for identity-based cryptosystems and its relationship with various other notions of security has been extensively studied. As a result, it is a general consensus that the sID model is much weaker than the full-identity (ID) model. The main focus is on the problem of constructing an ID-secure IBS given an sID-secure IBS without using random oracles–the so-called *standard* model–and with reasonable security degradation. We accomplish this by devising a generic construction which uses as black-box: *i*) a *chameleon* hash function and *ii*) a *weakly*-secure public-key signature. We argue that the resulting IBS is ID-secure but with a tightness gap of $\mathrm{O}(q_s)$, where q_s is the upper bound on the number of signature queries that the adversary is allowed to make.

Contents

Keywords

Provable Security, Random Oracles, Schnorr Signature, Oracle Replay Attack, Forking, Identity-Based Signature, Galindo-Garcia IBS, Multiple-Forking, Random-Oracle Dependency, Selective-Identity Model

Chapter 1

Introduction

The concept of identity-based cryptosystems (IBC) was introduced by Shamir in 1984 [Sha85]. In IBC, any arbitrary string such as an e-mail address can act as the public key. In traditional public-key cryptosystems (PKC), users have to exchange public-key certificates before being able to communicate securely. These certificates provide the external binding between the public key and the identity of a user. In some scenarios certificates can prove to be cumbersome. Using IBC one can avoid the complicated certificate management–this, precisely, was Shamir's foresight.

Identity-based signatures (IBS) extend the notion 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 the "master" public key of the private-key generator (PKG)[1]. Since IBS, as in IBC, does not require any certificates to be exchanged, it can be advantageous over the traditional public-key signature (PKS) systems in certain scenarios. IBS, in particular, turns out to be quite practical in wireless sensor networks [LBZ+10, TWZL08], BGP protocol [KLS00], MANET routing [HWS+06] *etc*.. Therefore, the question of designing efficient and secure IBS is important in the context of applied cryptography.

Although Shamir, in his 1984 paper, gave a simple RSA-based construction for IBS he was unable to establish the security of the scheme. Several "secure" RSA based IBS [FS87, GQ90] have been proposed in the literature since. In recent times, the advent of pairing ushered the way to an increased interest in research on IBC, and consequently IBS mushroomed [CC, Her05, Hes03].

1.1 Identity-Based Signatures

Our aim, in this section, is to give an intuitive feel of IBS. A more formal definition follows in §**1.3.2**. But first we proceed to describe a basic IBC setup.

A key component in IBC is the private-key generator (PKG). The role of the PKG, as the name suggests, is to generate the private keys[2] corresponding to the users in the system.

[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.

[2]The terms 'private key' and 'secret key' will be used interchangeably throughout the thesis.

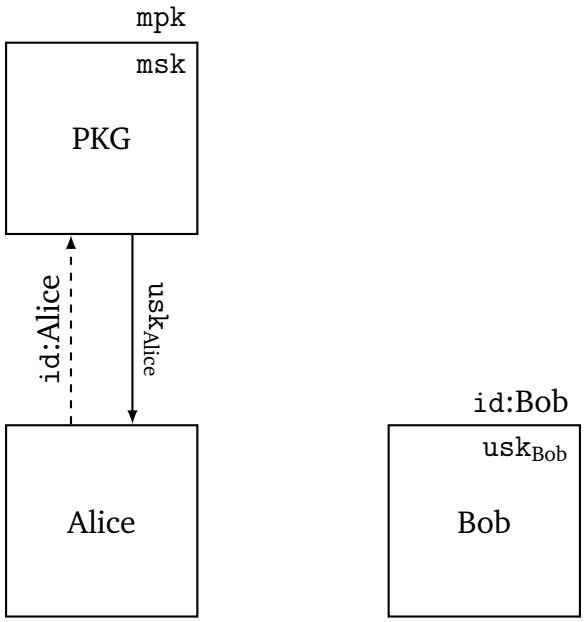

Figure 1.1: A schematic for IBC The dashed line indicates an open channel, whereas the solid line indicates a secure channel.

This is done through a secure channel as shown for the user Alice with the identity "Alice" in **Figure 1.1**. It also generates the master public key `mpk` which can, in some sense, considered to be the "public key" of the PKG. Therefore, on a high level, the PKG can be regarded as a trusted third party that provides the binding between the identity of a user an her secret key– passably analogous to the certifying authority in the traditional public-key infrastructure (PKI).

The key-escrow problem. An inherent problem with IBC, as one could make out, is the "key-escrow" problem. As the PKG can compute secret key corresponding to any user, it can decrypt ciphertext meant for any user (or sign message on behalf of any user) in the system. Some of the alternatives are distributed PKG [BF01] and certificateless PKC [ARP03].

The notion of IBS, if you recall, is the extension of the idea of digital signatures to the identity-based setting. It consists of four probabilistic, polynomial-time (PPT) algorithms Set-up, Key Extraction, Signing and Verification. The Set-up algorithm is used to generate the master public key `mpk` with the exact level of security dictated by a security parameter[3]. Key Extraction is used by the PKG to generate the user secret key `usk`. The user secret key can considered as the signature by the PKG with the identity of the user being the message and using the master secret key as the signing key. The algorithms Signing and Verification

[3]The security parameter is a variable which measures the level of security in which a cryptographic protocol is to be deployed. It is expressed using the *unary* notation; *e.g.*, for n-bit security level, $\kappa = 1^n$. κ determines the resource requirement of the protocol and, consequently, the capability adversary requires for breaking the protocol. For example, if a protocol is deployed in 80-bit security level, it would take an adversary $\mathsf{O}\left(2^{80}\right)$ operations to break the protocol.

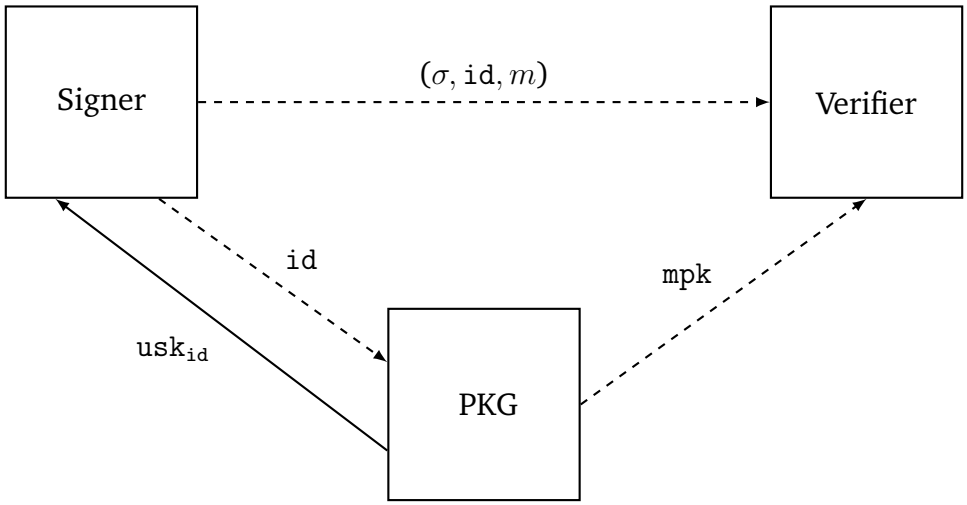

Figure 1.2: A schematic for IBS.

correspond to the namesake algorithms in the traditional PKS setting. A signer uses Sign to compute a signature σ on a message m with the user secret key being the signing key. Finally, this signature can be verified by the verifier using the Verification algorithm, using just the signer's identity and the master public key.

1.2 Provable Security

Classical cryptography relied heavily on cryptanalysts. Thus, most of the classical cryptosystems that are still in use are the time-tested ones. But nowadays, provable security–the "art" of formally arguing the security of cryptosystems–is fast becoming a *de facto* standard [KM07, Sho04]. Protocol designers are expected to support their construction with an argument establishing its security, *i.e*. the security argument. In a nutshell, security is argued through a "game" between a "challenger"–the party that strives to prove the protocol secure–and an "adversary"–the party that attempts to break the protocol. Let's start with the description of the adversary, in particular, the abstraction that is used to capture its behaviour.

Modelling the adversary. The adversary is modelled as a probabilistic, polynomial-time (PPT) Turing machine (PTM).[4] An adversary taking an input x and producing an output y is denoted by $y \xleftarrow{\$} \mathcal{A}(x)$. The adversary may have access to some oracles. These are listed in the superscript; *e.g*., $y \xleftarrow{\$} \mathcal{A}^{\mathcal{O}(\cdot)}(x)$. A PTM handles its internal decision making through a series of (internal) coin tosses (with some underlying probility distribution). It is desirable, sometimes, to control the internal decision making of an adversary (modelled as a PTM). This can be accomplished by passing the internal coins explicitly as an input, say, on a separate tape to the adversary. In this particular way of modelling, the adversary (in some sense) acts in a deterministic manner. However, it has to be, then, analysed over the

[4]We consider only computationally-bounded adversaries.

space of these internal coins as in the case of a randomised algorithm. As for notation, these internal coins are distinguished from the normal input using a semi-colon, *i.e.* $y \leftarrow \mathcal{A}(x;\rho)$. We are now equipped to formally define the notion of "an adversary breaking a protocol".

Definition 1. Consider an adversary $\mathcal{A}$ trying to break a protocol $\mathfrak{P}$ deployed in a κ-bit security level. $\mathcal{A}$ is said to $(\epsilon(\kappa), t(\kappa))$-*break* $\mathfrak{P}$ if it takes time *at most* $t(\kappa)$ and is successful with a probability *at least* $\epsilon(\kappa)$. We say that $\mathcal{A}$ is an $(\epsilon(\kappa), t(\kappa))$-adversary against $\mathfrak{P}$.[5]

Security models. The security models lay down the schema to be followed while giving a security argument. It is usually described using a security game between the challenger and adversary. The security model sets some ground rules regarding the powers the adversary has and the obligation the challenger has, to the adversary, in the security game. Security models are characterised [PV05] through an *adversarial goal* and an *attack model*–*e.g.*, in the EU-CMA model, the adversary has to produce an existential forgery (the adversarial goal) under the chosen-message attack (the attack model). A detailed discussion on security models and security games is deferred to §**1.3**.

1.2.1 Reductionist Security Arguments

Now that we are aware of the notions relevant to security arguments, let's see how the security is actually argued. A prevalent approach in provable security is argument using the *reductio ad absurdum* principle. Suppose $\mathfrak{P}$ is the cryptographic protocol that we want to argue secure. Let Π be the hard problem[6] that $\mathfrak{P}$ is based on. Our objective is to argue that $\Pi \implies \mathsf{M}[\mathfrak{P}]$: if Π is a hard problem, then the protocol $\mathfrak{P}$ is secure in the security model M.[7] Another way to establish the same would be through the contrapositive $\neg\mathsf{M}[\mathfrak{P}] \implies \neg\Pi$. Thus, the strategy is to assume a hypothetical (polynomial-time and possibly probabilistic) adversary $\mathcal{A}$ against $\mathfrak{P}$ and construct an algorithm $\mathcal{B}$–the so-called *reduction* algorithm–that harnesses $\mathcal{A}$, to solve the hard problem Π. Since we presumed Π to be hard, this leads to a contradiction and there cannot exist such an adversary. Or, in classical complexity theory jargon, solving the hard problem Π is *reduced* to breaking the protocol $\mathfrak{P}$, *i.e.*, $\Pi \leq \mathsf{M}[\mathfrak{P}]$ (or, $\Pi \Leftarrow \mathsf{M}[\mathfrak{P}]$). This is illustrated[8] in **Figure 1.3**.

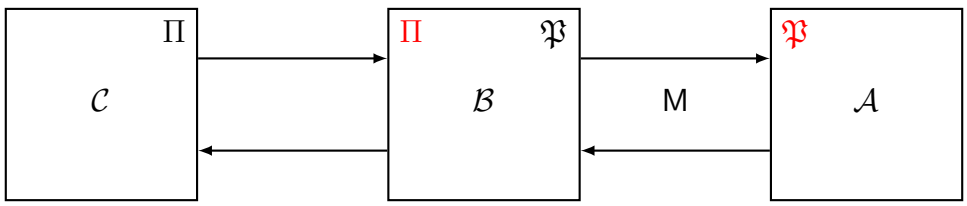

Figure 1.3: Argument through contradiction.

[5]Whenever the context is clear, we abuse the notation by omitting κ; *i.e.*, we say that $\mathcal{A}$ is (ϵ, t)-adversary against $\mathfrak{P}$.

[6]Computationally infeasible problem. See §**1.3.3** for an example of hard problem.

[7]Recall that we use long arrow to denote logical implication and short arrow to denote reductions.

[8]We follow this convention of block diagrams throughout the thesis. Moreover, oracle access is indicated in the bottom right corner of the blocks.

The notion can be easily extended to base security of one protocol on another. In order to base the security of a protocol $\mathfrak{P}$ (in a security model M) on another protocol $\mathfrak{P}'$ (which is secure in a security model M′), one has to establish $\mathsf{M}'[\mathfrak{P}'] \leq \mathsf{M}[\mathfrak{P}]$: given an adversary against $\mathfrak{P}$ in the security model M, one has to construct an algorithm that breaks $\mathfrak{P}'$ in the security model M′ (resulting in a contradiction).

The reduction algorithm. Let's see how the reduction algorithm $\mathcal{B}$ works. Recall, from the preceding discussion, that $\mathcal{B}$'s objective is to harness $\mathcal{A}$ to solve Π. Suppose that $\mathcal{B}$ receives the instance of the hard problem from its challenger $\mathcal{C}$. It initiates the security game M with the adversary $\mathcal{A}$ by sending a challenge instance of $\mathfrak{P}$. Next, $\mathcal{B}$ *simulates* the environment of protocol $\mathfrak{P}$ to the adversary $\mathcal{A}$. Note that the simulation has to be faithful[9] to the actual execution of the protocol. At the end of the simulation, if $\mathcal{A}$ succeeds in breaking $\mathfrak{P}$ then $\mathcal{B}$ has to (somehow) to solve Π. If $\mathcal{B}$ is indeed successful, then we would have reduced Π to $\mathfrak{P}$. Thus, $\mathcal{B}$ plays the role of *both* challenger and adversary (in the respective security games).

Tightness of security reductions. Returning to the preceding discussion, let's suppose that $\mathcal{A}$ is an (ϵ, t)-adversary against $\mathfrak{P}$. Also, let the reduction $\mathcal{B}$ that we designed (ϵ', t')-solve Π. The tightness gap for $\mathcal{B}$ is defined to be

$$\gamma_{\mathcal{B}} = \frac{t'/\epsilon'}{t/\epsilon}.$$

We say that the security of $\mathfrak{P}$ "degrades" by a factor of $\gamma_{\mathcal{B}}$. A reduction $\mathcal{B}$ is said to be "tight" if $t' \approx t$ and $\epsilon' \approx \epsilon$ and, by implication, $\gamma_{\mathcal{B}}$ is of constant order ($\mathrm{O}(1)$). In other words, $\mathcal{B}$ has to simulate $\mathcal{A}$ only a constant number of times before solving Π with a "considerable" advantage. Likewise, $\mathcal{B}$ has polynomial degradation if $\gamma_{\mathcal{B}}$ is a polynomial in κ (and so forth).

1.2.2 The Random-Oracle Methodology

Hash functions are indispensable to cryptography. The random-oracle methodology was introduced with the aim of simplifying the security arguments of cryptographic protocols that use hash functions. Although the methodology was first used, in the context of security arguments, by Fiat and Shamir in their zero-knowledge proof [FS87], it was formalised (and popularised) by Bellare and Rogaway [BR93].

The idea is to, first, design an ideal system in which all parties, including the adversary $\mathcal{A}$, have oracle access to a truly random function that represents the hash function–the so-called *random oracle*–and prove the security of this ideal system. During the simulation, the random oracle is under the control of the challenger $\mathcal{C}$ (roughly speaking, see [FLR+10] for refinements) and the adversary is allowed to make (a bounded number of) queries ($\mathtt{Q}_i$) to this oracle. The random oracle is then instantiated by using a cryptographic hash

[9] Technically speaking, the adversary should not be able to distinguish the simulation of the protocol environment from the actual execution of the protocol.

function, say MD5 or SHA-2, giving us an instance of the ideal system in the real world (see **Figure 1.4**).

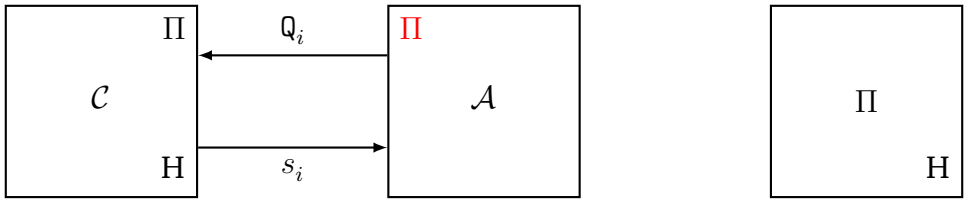

Figure 1.4: Random Oracle Methodology

Protocols without random oracles–in the so-called *standard model*, that is–turn out to be comparatively involved; random oracle assumption is known to simplify the construction as well as the security argument. Plus, the security arguments are tighter (cf. [BF01] and [Wat05]). Moreover, there exist protocols (Schnorr signature [Sch91] for example) for which the only known security arguments are the ones based on the random oracle assumption. However, proofs in the random-oracle model provide only a *heuristic* support for actual security [FLR+10].

1.3 Preliminaries

In this section, we give the formal definition of PKS and IBS, along with their relevant security models. We also define the discrete-logarithm assumption–the hardness assumption on which the objects of our interest (Schnorr signature and Galindo-Garcia IBS) are based on. We cover some of the existing (and relevant) techniques for construction of IBS as well.

1.3.1 Public-Key Signatures

Definition 2 (Public-Key Signature)**.** A PKS scheme consists of three PPT algorithms $\{\mathcal{K}, \mathcal{S}, \mathcal{V}, \ldots,\}$.

> Key Generation, $\mathcal{K}(1^\kappa)$: It takes as input the security parameter κ (in unary) and outputs the public key $\mathtt{pk}$ and the secret key $\mathtt{sk}$.
>
> Signing, $\mathcal{S}(m, \mathtt{sk})$: It takes as input a message m and the secret key of the user $\mathtt{sk}$ to generate a signature σ.
>
> Verification, $\mathcal{V}(\sigma, m, \mathtt{pk})$: It takes as input the signature σ, the message m and the public key of the user $\mathtt{pk}$. It outputs a bit b which is 1 if σ is valid signature on m or 0 if the signature is invalid.

The standard *correctness* condition: $1 \leftarrow \mathcal{V}(\mathcal{S}(m, \mathtt{sk}), m, \mathtt{pk})$, should be satisfied for all m and $(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathcal{K}(1^\kappa)$. The standard security notion for PKS schemes: existential unforgeability under chosen-message attack (EU-CMA) [GMR88], is defined below.

Definition 3 (EU-CMA Game)**.** The security of a PKS scheme in the EU-CMA model is argued in terms of the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Set-up: $\mathcal{C}$ runs $\mathcal{K}$ to obtain a public key $\mathtt{pk}$ and a secret key $\mathtt{sk}$. It passes $\mathtt{pk}$ as the challenge public key to $\mathcal{A}$.

Signature query, $\mathcal{O}_s(m)$: $\mathcal{A}$ can adaptively make signature queries to an oracle $\mathcal{O}_s$. $\mathcal{C}$ responds by running the algorithm $\mathcal{S}$ to obtain a signature corresponding to m. It then sends σ to $\mathcal{A}$.

Forgery: $\mathcal{A}$ outputs a signature $\hat{\sigma}$ on a message $\hat{m}$ and *wins* the game if the forgery is *i*) *valid:* $\hat{\sigma}$ passes the verification on $\hat{m}$; and *ii*) *non-trivial:* $\mathcal{A}$ has not queried the signature oracle with $\hat{m}$.

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

$$\Pr\left[1 \leftarrow \mathcal{V}(\hat{\sigma}, \hat{m}, \mathtt{pk}) : (\mathtt{sk}, \mathtt{pk}) \xleftarrow{\$} \mathcal{K}(1^{\kappa}); (\hat{\sigma}, \hat{m}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_s(\cdot)}(\mathtt{pk})\right]$$

provided $\hat{\sigma}$ is a non-trivial forgery on $\hat{m}$. $\mathcal{A}$ is said to be an (ϵ, t, q_s)-forger of an PKS scheme if it has advantage of at least ϵ in the above game, runs in time at most t and queries the signature oracle at most q_s times.

Definition 4 (Full-Security of PKS)**.** We say that a PKS is existentially unforgeable against a chosen-message attack (fully-secure, in short) if for any PPT adversary $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EU\text{-}CMA}}(\kappa)$ is negligible[10] in κ.

Remark 1 (No-Message Attack)**.** The adversarial task of producing an existential forgery under the no-message attack (EU-NMA), also known as the key-only attack (EU-KOA), can be regarded as a particular, weaker, case of the task of producing an existential forgery under chosen-message attack, with $q_s = 0$.

Weakly-secure PKS. A weaker notion of PKS security that we are interested in is existential forgery under *generic* chosen-message attack (EU-GCMA) [GMR88]. The distinguishing feature of EU-GCMA model is that the adversary commits, beforehand, to a set of messages $\{m_1, \ldots, m_{q_s}\}$ to the challenger. In response, the challenger gives it, along with challenge public key, a set of signatures corresponding to the committed messages. At the end of the game, the adversary has to forge on a message that is *not* part of the committed set. A more formal definition follows.

Definition 5 (EU-GCMA Game)**.** The security of a PKS scheme in the EU-GCMA model is argued in terms of the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Commitment: $\mathcal{A}$ commits to a set of messages $\tilde{\mathbb{M}} := \{m_1, \ldots, m_{q_s}\}$.

Response: $\mathcal{C}$ runs $\mathcal{G}$ to obtain the public key $\mathtt{pk}$ and the secret key $\mathtt{sk}$. It runs the signing algorithm $\mathcal{S}$ to obtain a set of signatures $\{\sigma_1, \ldots, \sigma_{q_s}\}$ corresponding to $\tilde{\mathbb{M}}$. Finally, it sends the challenge public key $\mathtt{pk}$ along with $\{\sigma_1, \ldots, \sigma_{q_s}\}$ to $\mathcal{A}$.

[10] A function $\mathrm{f} : \mathbb{R} \mapsto \mathbb{R}$ is *negligible* if for any $n > 0$, we have $|\mathrm{f}(x)| < 1/k^n$ for sufficiently large x [BF01].

Forgery: $\mathcal{A}$ outputs a signature $\hat{\sigma}$ on a message $\hat{m}$ and *wins* the game if the forgery is *i*) *valid:* $\hat{\sigma}$ passes the verification on $\hat{m}$; and *ii*) *non-trivial:* $\hat{m}$ is not a part of the committed set $\tilde{\mathbb{M}}$.

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

$$\Pr\left[1 \leftarrow \mathcal{V}(\hat{\sigma}, \hat{m}, \mathtt{pk}) \wedge \hat{m} \notin \tilde{\mathbb{M}} : \tilde{\mathbb{M}} \xleftarrow{\$} \mathcal{A}(q_s); (\mathtt{sk}, \mathtt{pk}) \xleftarrow{\$} \mathcal{K}(1^{\kappa}); (\hat{\sigma}, \hat{m}) \xleftarrow{\$} \mathcal{A}(\mathtt{pk}, \sigma_1, \ldots, \sigma_{q_s})\right]$$

An adversary $\mathcal{A}$ is said to be an (ϵ, t, q_s)-forger of an PKS scheme if it has advantage of at least ϵ in the above game, runs in time at most t, after initially having committed to a set of q_s messages.

Definition 6 (Weak-Security of PKS)**.** We say that a PKS is existentially unforgeable against a generic chosen-message attack (*weakly*-secure, in short) if for any PPT adversary $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EU\text{-}GCMA}}(\kappa)$ is negligible in κ.

1.3.2 Identity-Based Signatures

Definition 7 (Identity-Based Signature)**.** An IBS scheme consists of four PPT algorithms $\{\mathcal{G}, \mathcal{E}, \mathcal{S}, \mathcal{V}\}$ described below.

Set-up, $\mathcal{G}(1^{\kappa})$: It takes as input the security parameter κ (in unary). It outputs the master secret key $\mathtt{msk}$ and the master public key $\mathtt{mpk}$.

Key Extraction, $\mathcal{E}(\mathtt{id}, \mathtt{msk})$: It takes as input the user's identity $\mathtt{id}$, the master secret key $\mathtt{msk}$ to generate a secret key $\mathtt{usk}$ for the user.

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

Verification, $\mathcal{V}(\sigma, \mathtt{id}, m, \mathtt{mpk})$: It takes as input a signature σ, a message m, an identity $\mathtt{id}$ and the master public key $\mathtt{mpk}$. If outputs a bit $\mathtt{b}$ which is 1 if σ is a valid signature on $(\mathtt{id}, m)$ or 0 if the signature is invalid.

The standard *correctness* condition: $1 \leftarrow \mathcal{V}(\mathcal{S}(\mathtt{id}, m, \mathtt{usk}), \mathtt{id}, m, \mathtt{mpk})$, should be satisfied for all $\mathtt{id}$, m, $(\mathtt{msk}, \mathtt{mpk}) \xleftarrow{\$} \mathcal{G}(1^{\kappa})$ and $\mathtt{usk} \xleftarrow{\$} \mathcal{E}(\mathtt{id}, \mathtt{msk})$. As for security, we use the detailed EU-ID-CMA model given in [BNN04].

Definition 8 (EU-ID-CMA Game[11])**.** The security of an IBS scheme in the EU-ID-CMA model is argued in terms of the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Set-up: $\mathcal{C}$ runs $\mathcal{G}$ to obtain the master public key $\mathtt{mpk}$ and the master secret key $\mathtt{msk}$. It passes $\mathtt{mpk}$ as the challenge master public key to $\mathcal{A}$.

[11]The security game in [BNN04], *i.e*. $\mathsf{Exp}_{\mathsf{IBS},\mathsf{F}}^{\text{uf-cma}}$, 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.

Queries: $\mathcal{A}$ can adaptively make extract queries to an oracle $\mathcal{O}_{\varepsilon}$ and signature queries to an oracle $\mathcal{O}_{s}$. These queries are handled as follows.

- **Extract query**, $\mathcal{O}_{\varepsilon}(\mathtt{id})$: $\mathcal{A}$ asks for the secret key of a user with identity $\mathtt{id}$. If there has already been an extract query on $\mathtt{id}$, $\mathcal{C}$ returns the user secret key that was generated during the earlier query. Otherwise, $\mathcal{C}$ uses the knowledge of $\mathtt{msk}$ to run $\mathcal{E}$ and generate the user secret key $\mathtt{usk}$, which is then passed on to $\mathcal{A}$.
- **Signature query**, $\mathcal{O}_{s}(\mathtt{id}, m)$: $\mathcal{A}$ asks for the signature of a user with identity $\mathtt{id}$ on a message m. $\mathcal{C}$ first obtains, as specified the extract query, a user secret key $\mathtt{usk}$ corresponding to $\mathtt{id}$. Next, it uses the knowledge of $\mathtt{usk}$ to run $\mathcal{S}$ and generate a signature σ, which is passed to $\mathcal{A}$.

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

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

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

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

Definition 9 (ID-Security of IBS)**.** We say that an IBS is EU-ID-CMA-secure (ID-secure, in short) if for any PPT adversary $\mathcal{A}$ the function $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{EU\text{-}ID\text{-}CMA}}(\kappa)$ is negligible in κ.

The notion of *selective-identity*. The selective-identity (sID) model for identity-based cryptographic schemes was introduced in [CHK03]. The distinguishing feature of this model is that the adversary has to commit, beforehand, to a "target" identity–*i.e.*, the identity which it intends to eventually forge on.

Definition 10 (EU-sID-CMA Game)**.** The security of an IBS scheme in the EU-sID-CMA model is argued in terms of the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Commitment: $\mathcal{A}$ commits to a target identity $\tilde{\mathtt{id}}$.

Set-up: $\mathcal{C}$ runs $\mathcal{G}$ to obtain the master keys ($\mathtt{mpk}$,$\mathtt{msk}$). It passes $\mathtt{mpk}$ as the challenge master public key to $\mathcal{A}$.

Queries: $\mathcal{A}$ can adaptively make extract queries to an oracle $\mathcal{O}_{\varepsilon}$ and signature queries to an oracle $\mathcal{O}_{s}$. It is restricted though from making extract query on the target identity $\tilde{\mathtt{id}}$. These queries are handled as in the EU-ID-CMA game.

Forgery: $\mathcal{A}$ outputs a signature $\hat{\sigma}$ on a message $\hat{m}$ and an identity $\tilde{\mathtt{id}}$, and *wins* the game if the forgery is *i*) *valid:* $\hat{\sigma}$ passes the verification on $(\tilde{\mathtt{id}}, \hat{m})$; and *ii*) *non-trivial:* $\mathcal{A}$ has queried the signature oracle with $(\tilde{\mathtt{id}}, \hat{m})$.

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

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

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

Definition 11 (sID-Security of IBS)**.** We say that an IBS is EU-sID-CMA-secure (sID-secure, in short) if for any PPT adversary $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EU\text{-}sID\text{-}CMA}}(\kappa)$ is negligible in κ.

Remark 2. If the security argument models the hash functions as random oracles, the security games are to be modified, appropriately, to accommodate queries to the random oracles.

1.3.2.1 Existing Techniques for Constructing IBS

The task of constructing IBS is generally considered to be much easier than constructing IBE. [BNN04] contains a comprehensive list of such techniques, along with the security arguments. We discuss, although briefly, three techniques–one concrete and two generic–that are relevant to the thesis.

From Schnorr signatures. The concrete technique that we are interested in is based on the Schnorr signature scheme [Sch91]. Galindo and Garcia [GG09] devised a way concatenate two Schnorr signatures to produce an identity-based signature. 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 resultant IBS is simple and efficient. We analyse its construction and security argument in the first part of our thesis.

The "folklore" construction. Although the cryptographic research community was long aware of this (generic) technique, it was formalised by Bellare *et al*. [BNN04]. It involves two applications of PKS (certificate). In a nutshell, the PKG generates a (public-private) keypair for a user and then *binds* it to the identity of the user by using a certificate generated using its (master) secret key. Thus the PKG, to a large extent, plays the role of a certifying authority in PKI. Interestingly, no random oracles are involved.

The (existence of) folklore construction reduces the task of constructing an IBS to that of constructing a PKS. A PKS, on the other hand, can be derived from a weakly-secure PKS using a chameleon hash function (CHF) [KR00]. This approach was used implicitly in [ST01] and, later, formalised in [HW09]. This, in turn, implies the task of constructing IBS

is reduced to that of constructing a weakly-secure PKS and a CHF. It also reiterates the fact that it is much easier (even in the standard model) to construct IBS schemes.

From sID **IBS.** Recall that the selective-identity (sID) model for IBC is considered to be weaker than the full-identity (ID) security model. However, it is much easier to design a protocol that is sID-secure than one that is ID-secure. Therefore, transforming an sID IBS to an ID IBS, in a *generic* manner, is an interesting problem to pursue, both from theoretical and practical point of view. An efficient (comparatively) black-box method to convert a sID-secure IBE scheme to ID security was suggested in [BF01]. But the method relies on random oracles [BR93]. This was followed by [BB04a], in which the problem is solved in the standard model, albeit with an *exponential* loss of tightness. Both these methods can be adapted to IBS. To the best of our knowledge, there doesn't exist an efficient generic transformation from sID IBS to ID IBS in the standard model. In the second half of our thesis we concentrate on this exact problem.

1.3.3 Discrete-Logarithm Assumption

Let $\mathcal{G}_{\mathsf{DL}}$ be a (randomised) group generator: it takes as input a security parameter κ (in unary) and outputs $(\mathbb{G}, g, p)$, where $\mathbb{G}$ is a cyclic group of prime order p generated by g.[12] The discrete-logarithm problem (DLP) is defined through the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ (see **Figure 1.5**).

Definition 12 (DLP Game)**.** $\mathcal{C}$ invokes the group generator to generate a cyclic group: $(\mathbb{G}, g, p) \xleftarrow{\$} \mathcal{G}_{\mathsf{DL}}(\kappa)$. Next, it sends to $\mathcal{A}$ the challenge DLP instance $(\mathbb{G}, g, p, h)$, where $h := g^{\alpha}$ with $\alpha \xleftarrow{\mathsf{U}} \mathbb{Z}_p$. $\mathcal{A}$ wins the game if it correctly finds the discrete logarithm of h (with respect to the generator g), *i.e.*, $\alpha' = \alpha$. The advantage $\mathcal{A}$ has solving the DLP is the probability with which it wins the game, *i.e.*,

$$\mathsf{Adv}^{\mathsf{DLP}}_{\mathcal{A}}(\kappa) \stackrel{\mathsf{def}}{=} \Pr\left[\alpha' = \alpha : (\mathbb{G}, g, p) \xleftarrow{\$} \mathcal{G}_{\mathsf{DL}}(\kappa); \alpha \xleftarrow{\mathsf{U}} \mathbb{Z}_p; \alpha' \xleftarrow{\$} \mathcal{A}(\mathbb{G}, p, g, g^{\alpha})\right].$$

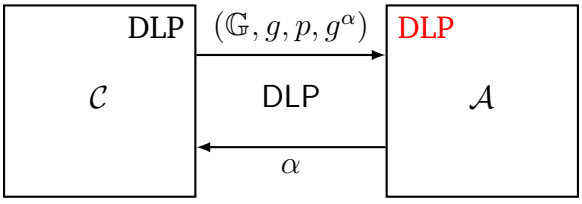

Figure 1.5: The DLP Game.

Definition 13 (Discrete-Logarithm Assumption)**.** The (ϵ, t)-discrete-log assumption holds if no adversary that takes takes time at most t has advantage at least ϵ in the DLP game. In general, the (asymptotic) discrete-log assumption holds if for all PPT adversaries $\mathcal{A}$ the function $\mathsf{Adv}^{\mathsf{DLP}}_{\mathcal{A}}(\kappa)$ is negligible in κ; we say that $\mathcal{G}_{\mathsf{DL}}$ generates DL-secure groups [GG09].

[12] We sometimes omit the description of the group generator for sake of convenience and, also, consistency.

1.4 Organisation of the Thesis

The Galindo-Garcia IBS (GG-IBS) is based on the Schnorr signature scheme [Sch91]. Hence, we devote **Chapter 2** to the particulars of Schnorr signature: its construction and security argument. This, in turn, lends us some insight into GG-IBS, especially its security argument. The machinery of (oracle) replay attack[13] [PS00] plays a pivotal role in the security argument of Schnorr signature and, therefore, we place emphasis on its development. This includes a discourse on the two techniques that are used to launch the replay attack: the Rewinding Technique and the Replicating Technique. Plus, we discuss an abstraction of the replay attack, called General Forking [BN06], that aids in simplifying security arguments. Finally, we augment the "elementary" replay attack (which involves only *one* random oracle) to include multiple random oracles and nested forkings. This yields the Multiple-Forking Algorithm [BPW12].

With the know-how on Schnorr signature in our arsenal, we revisit GG-IBS itself in **Chapter 3**. GG-IBS is derived through a simple and elegant concatenation of two Schnorr signatures. It works in the discrete-log setting but does not require pairing. The security is established by reducing the problem of computing the discrete-log to breaking the IBS, through two (reduction) algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$. Both $\mathcal{B}_1$ and $\mathcal{B}_2$ employ the Multiple-Forking (MF) Algorithm. The focus of the chapter is on this security argument: we find that it is ridden with flaws and ambiguities. In particular, we show that the reduction $\mathcal{B}_1$ in [GG09] *fails* in the standard security model for IBS [BNN04], whereas the reduction $\mathcal{B}_2$ is *incomplete*. As a remedy, we provide a new, detailed, security argument which allows tighter reductions [CKK13].

In **Chapter 4**, we explore possible avenues for further improvement in the tightness of GG-IBS security argument. As already stated, the machinery of MF Algorithm plays a central role in the security argument of GG-IBS. However, it is also responsible for the huge degradation in the concrete security of the IBS. Thus, a natural question arises: *Can the nested replay attack (on the GG-IBS adversary) be launched more effectively than in the MF Algorithm?* To this end, we introduce the two notions pertaining to random oracles: *dependency* and *independency*. Independency follows naturally for GG-IBS, whereas, dependency has to be induced through a tweak in construction. As it turns out, incorporating dependency and independency, in conjunction, results in a cleaner, (significantly) tighter security argument for GG-IBS completing the final piece of the GG-IBS jigsaw.

In **Chapter 5**, the second part of the thesis, we look into the concept of *selective-identity* (sID) [CHK03] in the context of IBS. Since its induction, the sID model for identity-based cryptosystems and its relationship with various other notions of security has been extensively studied. As a result, it is a general consensus that the sID model is much weaker than the full-identity (ID) model. The main focus is on the problem of constructing an ID-secure IBS given an sID-secure IBS in the *standard model* and with reasonable security degradation. We accomplish this by devising a generic construction which uses as black-box: *i*) a chameleon hash function and *ii*) a weakly-secure public-key signature [CK13b]. We argue that the resulting IBS is ID-secure but with a tightness gap of $\mathsf{O}(q_s)$, where q_s is the upper

[13] We will use the terms forking and (the process of launching) oracle replay attack interchangeably.

bound on the number of signature queries that the adversary is allowed to make.

Chapter 2

Schnorr Signature and the Oracle Replay Attack

2.1 Introduction

We devote this chapter to the particulars of Schnorr signature scheme: its construction and security argument. We also emphasise on developing the machinery of *oracle replay attack* [PS00] which plays a pivotal role in the security argument of Schnorr signature scheme (as well as other class[1] of signature schemes [ElG85, Oka93]). Since the Galindo-Garcia IBS (GG-IBS) scheme is based on Schnorr signature, the discussion lends us some insight into GG-IBS. Plus, we discuss an abstraction of the replay attack, called General Forking [BN06], that aids in simplifying security arguments. Finally, we augment the "elementary" replay attack (which involves only *one* random oracle) to include multiple random oracles and nested forkings. This yields the Multiple-Forking Algorithm [BPW12].

2.2 Schnorr Signature

The Schnorr signature scheme is based on the discrete-log problem (see **Definition 12**). It can be concocted from the Schnorr identification scheme [Sch90] by using the Fiat-Shamir transformation [FS87]. We now describe a (minor) variant of this signature scheme that helps us in studying GG-IBS.

2.2.1 Construction

Parameter Generation[2], $\mathcal{G}(1^\kappa)$: Invoke the group generator $\mathcal{G}_{\mathsf{DL}}$ (on 1^κ) to obtain $(\mathbb{G}, g, p)$. In addition, let H be a hash function $\mathrm{H} : \{0,1\}^* \mapsto \mathbb{Z}_p$. Return $\mathtt{pp} := (\mathbb{G}, g, p, \mathrm{H})$ as public parameters.

[1] To be precise, the signature schemes obtained from three-round identification schemes (Σ-protocols) through the Fiat-Shamir transformation [FS87].

Key Generation, $\mathcal{K}(\mathtt{pp})$: Select $z \xleftarrow{\text{U}} \mathbb{Z}_p$ and set $Z = g^z$. Return $\mathtt{sk} := (z, \mathtt{pp})$ as the secret key and $\mathtt{pk} := (Z, \mathtt{pp})$ as the public key.

Signing, $\mathcal{S}(m, \mathtt{sk})$: Select $r \xleftarrow{\text{U}} \mathbb{Z}_p$ and set $R := g^r$. Return $\sigma := (y, R) \in \mathbb{Z}_p \times \mathbb{G}$ as a signature on the message m, where

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

Verification, $\mathcal{V}(\sigma, m, \mathtt{pk})$: Parse σ as (y, R) and compute $c := \mathrm{H}(m, R)$. The signature is valid if

$$g^y = R \cdot Z^c.$$

Figure 2.1: The Schnorr Signature Scheme.

Remark 3. We use the convention $\mathcal{S}_{\mathrm{H}}$ to denote the usage of hash function H by the Signing algorithm $\mathcal{S}$. We use this convention as the Signing algorithm may be used with different hash functions. Simply put, the hash function too is considered to be a parameter to the Signing algorithm. This applies to the Verification algorithm $\mathcal{V}$ as well.

2.2.2 Security of Schnorr Signature: An Intuition

Suppose that we are working in a group $\mathbb{G}$ where the discrete-log assumption holds (see **Definition 13**). Consider an adversary $\mathcal{A}$ with the capability to carry out chosen-message attack on the Schnorr signature scheme. Let $\{\sigma_1, \ldots, \sigma_n\}$ be the signatures that $\mathcal{A}$ procures. Moreover, (for each $i \in \{1, \ldots, n\}$) let: *i*) σ_i be a signature on m_i, and *ii*) be of the form (y_i, R_i) with, $R_i = g^{r_i}$, $y_i = r_i + c_i z$ and $c_i = \mathrm{H}(m_i, R_i)$. The adversary's view is captured by the following system of n linear congruences (modulo p) in the $(n+1)$ unknowns $\{r_1, \ldots, r_n, z\}$.

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & c_1 \\ 0 & 1 & \cdots & 0 & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & c_n \end{pmatrix} \times \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \\ z \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{2.1}$$

It is evident from basic linear algebra that the system in (2.1) has p solutions. Besides, the discrete-log assumption ensures that it is infeasible for $\mathcal{A}$ to extract information regarding the r_is from the corresponding R_is. Thus, it is hard for $\mathcal{A}$ to learn the value of the secret key z. *However*, if (by some means) it secures two equations that use the same randomiser r but with different hash value c, it can easily solve for z (as the system is reduced to n linear congruences in n unknowns). Keeping this in mind, we argue the security of Schnorr

signature.

2.2.3 Basic Security

Let's start with the simpler (but weaker, see **Remark 1**) EU-NMA model for PKS. For the time being, we restrict ourselves to a particular class of (strong) adversaries which we refer to as "perfect" adversaries. A perfect adversary, against a particular scheme, has the maximum (possible) advantage in breaking the scheme (in some appropriate security model). Therefore, for the perfect adversary $\mathcal{A}$ that we consider against the Schnorr signature scheme, in the EU-NMA model, we have $\mathsf{Adv}_{\mathcal{A}}^{\text{EU-NMA}}(\kappa) = 1$.

Claim 1 (Basic security of Schnorr signature)**.** *Given a perfect adversary $\mathcal{A}$ against the Schnorr signature scheme in the EU-NMA model, it is possible to construct an algorithm $\mathcal{B}$ that solves the DLP, provided the hash function H is modelled as a random oracle with an upper bound of q queries. In other words,*

$$\text{DLP} \leq_{O(1)} \text{EU-NMA [Schnorr Signature]}.$$

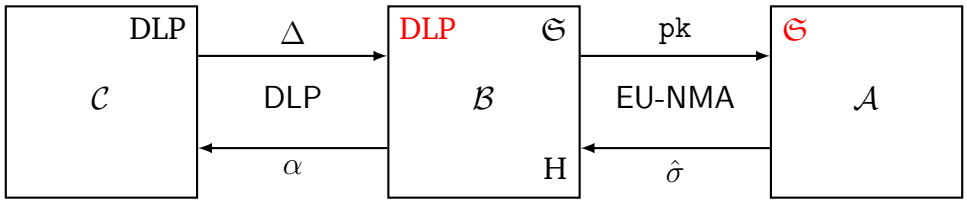

Figure 2.2: (Basic) Security Argument for Schnorr PKS (denoted by $\mathfrak{S}$).

Argument. Our objective is to construct an algorithm $\mathcal{B}$ that simulates the protocol environment for the Schnorr signature adversary $\mathcal{A}$ in the EU-NMA model. At the end of the simulation, $\mathcal{A}$ (being a perfect adversary) forges successfully and $\mathcal{B}$, in turn, has to somehow break the DLP. Thus, $\mathcal{B}$ plays the role of a challenger to $\mathcal{A}$ in the EU-NMA game; at the same time, it plays the role of an adversary in the DLP game (see **Figure 2.2**).

First round of simulation. The simulator $\mathcal{B}$ receives a DLP instance $\Delta = (\mathbb{G}, g, p, g^{\alpha})$ from its own challenger $\mathcal{C}$ and its objective is to extract α. $\mathcal{B}$ embeds the problem instance g^{α} in Z and, thus, passes $\mathtt{pk} := (\mathbb{G}, g, p, g^{\alpha})$ as the challenge public key to $\mathcal{A}$. $\mathcal{A}$ is allowed access to a random oracle H but it *does not* have access to the signature oracle. The adversary $\mathcal{A}$ makes a series of hash oracle queries $\{\mathtt{Q}_1, \ldots, \mathtt{Q}_q\}$ (with each $\mathtt{Q}_i$ of the form $\mathrm{H}(m_i, R_i)$) which the challenger responds to with $\{c_1, \ldots, c_q\} \xleftarrow{\text{U}} \mathbb{Z}_p$. At the end of the simulation, it successfully forges $\hat{\sigma} = (y, R)$ on some message $\hat{m}$ (with $R = g^r$, $c = \mathrm{H}(\hat{m}, R)$ and $y = r + \alpha c$). Since H is a random oracle, for $\mathcal{A}$ to have produced the forgery $\hat{\sigma}$ with a non-negligible advantage, it

must have made the query $\mathrm{H}(\hat{m}, R)$ at some juncture during its simulation[3]. In other words, there *must* exist an index $1 \leq I \leq q$ such that the query $\mathtt{Q}_I$ is $\mathrm{H}(\hat{m}, R)$ (see **Figure 2.3**). This index is termed the "target" H-index and the query, the "target" H-oracle query. To paraphrase, the random oracle query that is involved in the fabrication of the forgery, by the adversary, is called the target oracle query; the index of this query is called the target index.

$$\mathtt{Q}_1 \xrightarrow{c_1} \mathtt{Q}_2 \dashrightarrow \mathtt{Q}_I : \mathrm{H}(\hat{m}, R) \overset{c = c_I}{\longrightarrow} \mathtt{Q}_{I+1} \dashrightarrow \mathtt{Q}_q \xrightarrow{c_q} \hat{\sigma} = (y = r + \alpha c, R)$$

Figure 2.3: The "target" random oracle query $\mathrm{H}(\hat{m}, R)$.

Second round of simulation. Now, let's suppose that the simulator $\mathcal{B}$ (by some means) manages to "turn back the clock" to the juncture when $\mathcal{A}$ made the target oracle query $\mathtt{Q}_I$. Next, it simulates $\mathcal{A}$ again from $\mathtt{Q}_I$ but, this time, with a different set of outputs to the random oracle queries (starting with $\mathtt{Q}_I$)[4]. This constitutes $\mathcal{B}$ carrying out an *oracle replay attack* on the adversary $\mathcal{A}$–we say that the adversary $\mathcal{A}$ is "forked" at the point $\mathtt{Q}_I$.[5] Let's denote this new round of simulation of the adversary by round 1 and the initial round by round 0. Suppose, at the end of round 1, the adversary forges $\hat{\sigma}'$ on some message $\hat{m}'$ (with $R' = g^{r'}$, $c' = \mathrm{H}(\hat{m}', R')$ and $y' = r' + \alpha c'$). Thus the challenger has secured two forgeries, $\hat{\sigma}$ and $\hat{\sigma}'$ with $y = r + \alpha c$ and $y' = r' + \alpha c'$. Let I' be the target H-index in round 1. Now, there are two possibilities with respect to I and I': *i*) they could be different; or *ii*) they could match ($I' = I$). In the first case, $\mathcal{B}$ cannot extract α as it possesses (only) two congruences in three unknowns. However, in the second case the adversary has made the same commitment $(\hat{m}, R)$, but received two, different, oracle outputs (see **Figure 2.4**). Thus, the simulator ends up with two forgeries $\{\hat{\sigma}, \hat{\sigma}'\}$ with $\{y = r + \alpha c, y' = r + \alpha c'\}$–a system of two linear congruences in two unknowns–and it easily solves for α (by computing $(y - y')/(c - c')$). This completes our argument. □

[3] Let S denote the event that $\mathcal{A}$ successfully forges $\hat{\sigma} = (y, R)$ on some message $\hat{m}$. In addition, let H denote the event that $\mathcal{A}$ queries the random oracle with $\mathrm{H}(\hat{m}, R)$–the "target" H-oracle query. Thus, we have

$$\begin{aligned} \Pr[\mathsf{S}] &= \Pr[\mathsf{S} \wedge \mathsf{H}] + \Pr[\mathsf{S} \wedge \neg\mathsf{H}] \\ &= \Pr[\mathsf{S} \wedge \mathsf{H}] + \Pr[\mathsf{S} \mid \neg\mathsf{H}] \Pr[\neg\mathsf{H}]. \end{aligned}$$

The term $\mathsf{S}|\neg\mathsf{H}$ indicates that $\mathcal{A}$ has forged successfully, but, without actually having made the target H-oracle query $\mathrm{H}(\hat{m}, R)$. However, the value of $\mathrm{H}(\hat{m}, R)$ is necessary in order to forge. Therefore, in this situation, the best that the adversary can do to conjure up with the value of $\mathrm{H}(\hat{m}, R)$ is to guess it. However, the probability with which the adversary could have *correctly* anticipated the output is negligible ($1/p$ to be precise). Thus the $\Pr[\mathsf{S} \mid \neg\mathsf{H}]$ term (weighed down by the $1/p$ factor) is negligible and as a result $\Pr[\mathsf{S}] \approx \Pr[\mathsf{S} \wedge \mathsf{H}]$. Simply put, $\mathcal{A}$ *has* to make the target random oracle query to be able to forge with a non-negligible probability.

[4] Note that a random oracle is distinguished by its outputs–the random function that represents the oracle. Therefore, we can conceive a "new" random oracle by changing the outputs from the "original" random oracle. However, in this particular case, the new random oracle has to *agree* with the original random oracle up to the target oracle query. Simply put, one has to use two different random functions that agree up to a particular point (the point being the target query). We elaborate on how to accomplish this in the next section.

[5] We will use the terms forking and (the process of launching) oracle replay attack interchangeably.

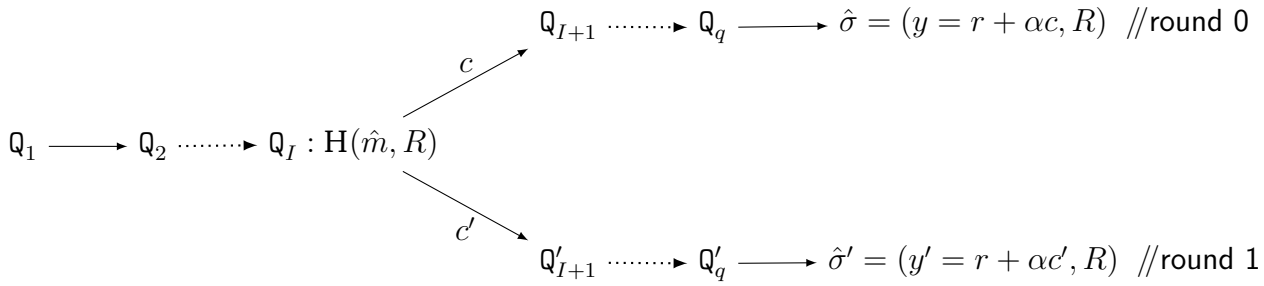

Figure 2.4: A successful oracle replay attack on $\mathcal{A}$.

With a bit more effort, it is possible to cater to the signature queries as well. Thus, the above argument can be extended to the full model for PKS, *i.e.*, EU-CMA. We omit this discussion to avoid digression. However, we *are* interested in security of Schnorr signature against more "general" probabilistic, polynomial-time (PPT) adversaries. But, this transition requires some additional tools. We discuss this in the next section, along with the discourse on the oracle replay attack and its analysis (which we had omitted in the proof).

2.3 The Oracle Replay Attack

In the previous section, we demonstrated how the (oracle) replay attack was used in order to carefully manipulate the adversary and obtain two related signatures. However, we had refrained from divulging the details on the replay attack. It was used at a "black-box" level. In this section, our focus is on the replay attack itself. We discuss two approaches: the "Replicating" Technique and the "Rewinding" Technique, that are used to carry out replay attacks. In addition to this, we "lift" the (basic) security argument given in §**2.2.3** to accommodate a more "general" PPT adversary. We end up with:

Theorem 1 (The Forking Lemma, Pointcheval-Stern [PS00][6])**.** *Given an ϵ-adversary $\mathcal{A}$ against the Schnorr signature scheme in the EU-CMA model, it is possible to construct an algorithm $\mathcal{B}$ that solves the DLP with an advantage $\epsilon' \geq \epsilon^2/4q$, provided the hash function H is modelled as a random oracle with an upper bound of q queries. In other words,*

$$\text{DLP} \leq_{O(q)} \text{EU-CMA [Schnorr Signature]}.$$

The security argument is enabled, primarily, by the "Splitting" Lemma [PS00] which we discuss next.

2.3.1 The Splitting Lemma

Let $\mathbb{X}$ and $\mathbb{Y}$ be two finite sets (with some underlying probability distribution) with $|\mathbb{X}| = m$ and $|\mathbb{Y}| = n$. Also, let $\mathbb{T}$ denote the "universal" set $\mathbb{X} \times \mathbb{Y}$. A pair $(x, y) \in \mathbb{X} \times \mathbb{Y}$ is deemed to

[6]It is possible to give a simpler, cleaner, security argument using the notion of "general forking".

be good if it satisfies a certain property. Let $\mathbb{V}$ denote the set of all such good pairs. Suppose that at least γ fraction of the pairs in $\mathbb{X} \times \mathbb{Y}$ are good, *i.e.*, $\Pr[\mathbb{V}] \geq \gamma$. We define a function $\mathrm{k} : \mathbb{X} \mapsto \mathfrak{P}(\mathbb{Y})$ as

$$\mathrm{k}(x) \stackrel{\text{def}}{=} \{y \in \mathbb{Y} \mid (x, y) \in \mathbb{V}\}.$$

k can be used to count, for a particular x, the number of elements $y \in \mathbb{Y}$ such that (x, y) forms a good pair. For some $\beta < \gamma$, we define the (sub)set of better pairs of $\mathbb{V}$ as

$$\mathbb{V}^* \stackrel{\text{def}}{=} \{(x, y) \in \mathbb{V} \mid |\mathrm{k}(x)| \geq (\gamma - \beta)n\}.$$

Thus, a good pair (x, y) is better if x forms good pairs with at least $(\gamma - \beta)$ fraction of the elements of $\mathbb{Y}$.

Lemma 1 (The Splitting Lemma, Pointcheval-Stern [PS00][7])**.** *The following propositions hold on the aforementioned assumptions:*

1. $\Pr[\mathbb{V}^*] \geq \beta$*, i.e., at least* β *fraction of pairs in* $\mathbb{T}$ *are better.*
2. $\Pr\left[(x, y') \in \mathbb{V} \mid (x, y) \in \mathbb{V}^*; y' \stackrel{\$}{\leftarrow} \mathbb{Y}\right] \geq (\gamma - \beta)$*, i.e., given a better pair* (x, y)*, the probability with which the pair* (x, y') *(with* y' *sampled randomly from* $\mathbb{Y}$*) is good is at least* $(\gamma - \beta)$*.*
3. $\Pr[\mathbb{V}^* \mid \mathbb{V}] \geq \beta/\gamma$*, i.e., a good pair is better with a probability of at least* β/γ*.*

Argument. We begin with **Proposition 2**. It is not difficult to see that it follows from the definition of good and better sets. Moving on, we establish **Proposition 1** through contradiction. Suppose $\Pr[\mathbb{V}] \geq \gamma$ and $\Pr[\mathbb{V}^*] < \beta$, then we have

$$\Pr[\mathbb{V}] = \Pr[\mathbb{V}^*] + \Pr[\mathbb{V} \setminus \mathbb{V}^*] < \beta + \Pr[\mathbb{V} \setminus \mathbb{V}^*]. \tag{2.2}$$

Now, there can be at most m distinct x elements in $\mathbb{V} \setminus \mathbb{V}^*$. In addition, for each such $x \in \mathbb{V} \setminus \mathbb{V}^*$, $|\mathrm{k}(x)| < (\gamma - \beta)n$ (by definition). Therefore (2.2) yields,

$$\Pr[\mathbb{V}] < \beta + \frac{|\mathrm{k}(x)|.m}{mn} < \beta + \frac{(\gamma - \beta)n.m}{mn} = \gamma.$$

This contradicts our assumption that at least γ fraction of elements in $\mathbb{T}$ are good ($\Pr[\mathbb{V}] \geq \gamma$), thus, establishing **Proposition 1**. Finally, we establish the **Proposition 3** using Bayes' theorem as shown below.

$$\begin{aligned} \Pr[\mathbb{V}^* \mid \mathbb{V}] &= \Pr[\mathbb{V} \mid \mathbb{V}^*] \Pr[\mathbb{V}^*] / \Pr[\mathbb{V}] \quad \text{(by Bayes' theorem)} \\ &\geq \beta/\gamma \quad \text{(using } \textbf{Proposition 1}\text{)} \end{aligned} \tag{2.3}$$

Note that the second step ($\Pr[\mathbb{V}^* \mid \mathbb{V}] = 1$) follows from the fact that a better pair is always good. This concludes the argument. □

[7] Also refer to [Kia07].

The Splitting Lemma: An alternative formulation. Koblitz-Menezes [KM07] explained the Splitting Lemma using a different perspective from that in **Lemma 1**. This comes in handy in certain scenarios. The sets $\mathbb{X}$, $\mathbb{Y}$ (hence $\mathbb{T}$) and $\mathbb{V}$, as well as the function k, are defined as in the previous formulation. For some $\beta < \gamma$, we define the (sub)set of better elements of $\mathbb{X}$ as

$$\mathbb{X}^* = \{x \in \mathbb{X} \mid |\mathrm{k}(x)| \geq (\gamma - \beta)n\}.$$

Thus, an element $x \in \mathbb{X}$ is better if it forms good pairs with at least $(\gamma - \beta)$ fraction of the elements of $\mathbb{Y}$.

Lemma 2 (The Splitting Lemma, Koblitz-Menezes [KM07])**.** *The following propositions hold on the aforementioned assumptions:*

1. $\Pr[\mathbb{X}^*] \geq \beta$*, i.e., at least* β *fraction of elements in* $\mathbb{X}$ *are better.*
2. $\Pr\left[(x, y) \in \mathbb{V} \mid x \in \mathbb{X}^* \wedge y' \xleftarrow{\$} \mathbb{Y}\right] \geq \gamma - \beta$*, i.e., given a better element* x*, the probability with which the pair* (x, y) *(with* y *sampled randomly from* $\mathbb{Y}$*) is good is at least* $\gamma - \beta$*.*

Argument. As in **Lemma 1**, **Proposition 2** follows by definition. **Proposition 1** is, again, established through contradiction. Provided that $\Pr[\mathbb{V}] \geq \gamma$ and $\Pr[\mathbb{X}^*] < \beta$, let's count the number of good pairs in $\mathbb{T}$. For any good pair (x, y), the coordinate x comes from either $\mathbb{X}^*$ or $\mathbb{X} \setminus \mathbb{X}^*$. Keeping this in mind, we define the two sets $\mathbb{V}_1$ and $\mathbb{V}_2$ as follows.

$$\mathbb{V}_1 = \{(x, y) \mid x \in \mathbb{X}^* \wedge y \in \mathrm{k}(x)\} \qquad \mathbb{V}_2 = \{(x, y) \mid x \in \mathbb{X} \setminus \mathbb{X}^* \wedge y \in \mathrm{k}(x)\}$$

It follows from the definition that

$$\Pr[\mathbb{V}] = \Pr[\mathbb{V}_1] + \Pr[\mathbb{V}_2] < \beta \cdot 1 + 1 \cdot (\gamma - \beta) = \beta. \tag{2.4}$$

(2.4) contradicts our initial assumption that $\Pr[\mathbb{V}] \geq \gamma$ and that concludes the argument. □

Remark 4. We refer to the Splitting Lemma by (γ, β)-Splitting Lemma. For a particular value of γ, the optimal value of β depends on the way the replay attack is carried out. This is demonstrated in §**2.3.2.3**.

The Splitting Lemma in the context of signature schemes. Recall the basic security argument for Schnorr signature scheme from §**2.2.3**. We had assumed a perfect adversary– an adversary capable of forging every time. In other words, the adversary is successful irrespective of the input (which includes the output of the random oracle queries it made). Next, let's consider a more general adversary with a success probability of ϵ. The good set can be regarded as (the set of) those particular inputs for which the adversary is able to forge a signature successfully[8]. Thus for an adversary with success probability ϵ, at least ϵ

[8] *Caveat:* Recall our modelling of the adversary. We had assumed the adversary to be a probabilistic (polynomial-time) Turing machine. Now, consider two simulations of the adversary on a specific input. Its behaviour, during these two simulations, depends on the internal coin tosses. Therefore, it may not necessarily be successful during both the simulations. This means that our classification of an input as good is a bit ambiguous. However, it serves the purpose of lending an intuitive explanation. A sound definition of good input has to take into account these internal coin tosses as well. We adopt this approach from the next section.

fraction of the inputs results in a valid forgery; in other words, $\gamma = \epsilon$.[9] As for the perfect adversary, we can consider any input to be good, *i.e.*, $\gamma = \epsilon = 1$. Although the discussion pertains to Schnorr signature scheme, it applies to other signature schemes as well.

2.3.2 Launching the Oracle Replay Attack

Let's go back to the (basic) security argument of the Schnorr signature scheme, especially, the way the replay attack was carried out. The simulator "turned back the clock" to a "critical" juncture and then continued with a new simulation with a different "setting". In this section, we discuss two approaches by which one can actually carry out the replay attack: the *Replicating Technique* and the *Rewinding Technique*. This would explain how to turn back the clock, what comprises a critical point and what we mean by a different setting. Finally, we analyse the two techniques using the Splitting Lemma. We stick to Schnorr signature and will be constantly referring back to the basic security argument from §**2.2.3**.

Replicating and rewinding: an analogy. Before delving into the technical details, we would like to give an intuitive overview of the two approaches. We adopt popular science-fiction subjects for analogy. Intuitively, the Rewinding Technique can be likened to the concept of *time travel*. The adversary (modelled as a probabilistic Turing machine) is executed once. At the end of this round, we have sufficient knowledge about the critical point. The "rewinding" attribute of a Turing machine (which enables us to trace its actions back to a previous state of progress) is, then, used to restore the adversary to this critical point. The adversary is then simulated again, but, with a setting which is different from the initial round of simulation. The clock is thus, literally, turned back on the adversary.

On the other hand, the Replicating Technique can be likened to the concept of *parallel universes*[10]. At some (random) point during its simulation, the adversary is paused and we produce another, identical, copy of it[11]. Then, these two instances are simulated independently, but, each with a different setting. Finally, we hope the point at which we diverged turns out to be the critical point. To sum it all up, we have two adversaries, identical up to a certain point, but with behaviour differing from that point, as if in parallel universes[12]. We now move on the technical details.

2.3.2.1 The Replicating Technique

Let's return to the security argument of the Schnorr signature scheme. The critical point in the simulation is precisely the juncture when the adversary $\mathcal{A}$ makes the target oracle query $\mathtt{Q}_I$. The simulator starts by making a guess i of this target index I. Next, $\mathcal{B}$ continues with the simulation until the adversary $\mathcal{A}$ makes the i^{th} query. At this juncture, $\mathcal{B}$ produces a

[9] We may, at times, use a more refined definition of good. See §**2.3.2.3** for an example of such a definition.

[10] It is also referred to as a *multiverse*. See the introductory chapter of [Teg03] for a intuitive explanation of parallel universes.

[11] This is possible as the adversary is modelled as a Turing machine.

[12] Interestingly, the concepts of time travel and parallel universes are closely related.

replica of $\mathcal{A}$ which we denote by $\mathcal{A}'$. The simulator, then, proceeds with the simulation of these two instances of the adversary independently–*forking, by replicating the adversary at* $\mathtt{Q}_I$. However, in order to induce a different setting in the two rounds (from $\mathtt{Q}_i$), $\mathcal{B}$ responds to the H-oracle queries with different outputs: $\mathcal{A}$'s queries are responded to with $\{c_i, \ldots, c_q\}$, whereas, those of $\mathcal{A}'$ with $\{c'_i, \ldots, c'_q\}$, as shown in **Figure 2.5** below.

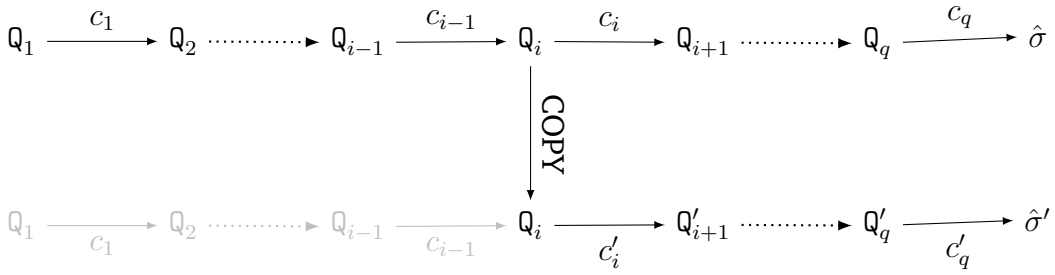

Figure 2.5: Oracle replay attack carried out by replicating.

At the end of the two rounds, the simulator is in possession of two forgeries, one each due to $\mathcal{A}$ and $\mathcal{A}'$. Let I and I' be the target indices for the two rounds. The forking is successful if the simulator's guess of the target index, *i.e.* i, turns out to be correct for both the rounds, *i.e.*, $I' = I = i$.

2.3.2.2 The Rewinding Technique

The approach adopted to launch the replay attack using the Rewinding Technique is fundamentally different from that in the Replicating Technique. Nevertheless, it still involves two simulations of the adversary. But in the Rewinding Technique, contrary to the Replicating Technique, the challenger first completes a full simulation of the adversary as shown in round 0 in the figure below.

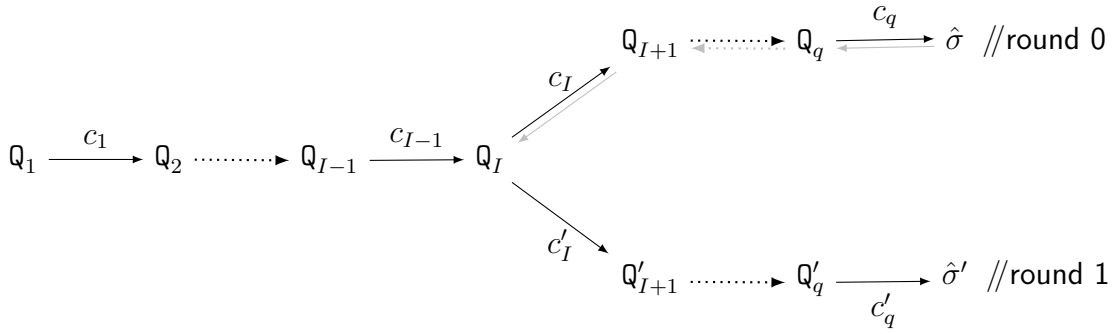

Figure 2.6: Oracle replay attack carried out by rewinding.

At the end of the round, the simulator is in possession of a forgery $\hat{\sigma} = (y, R)$ on some message $\hat{m}$. The forgery, in turn, contains sufficient information about the critical point for that particular round–it is, precisely, the juncture when the adversary made the target oracle

query $\mathrm{H}(\hat{m}, R)$, which is denoted in **Figure 2.6** by $\mathtt{Q}_I$. The simulator, next, rewinds[13] the adversary $\mathcal{A}$ (which is modelled as a Turing machine) to this critical point and then continues with the simulation of the adversary, but with a different setting–*forking, by rewinding the adversary to* $\mathtt{Q}_I$. Let's call this round 1. A different setting is induced in round 1, as in the Replicating Technique, by using a different set of outputs in response to the random oracle queries–$\{c'_I, \ldots, c'_q\}$ compared to $\{c_I, \ldots, c_q\}$ in round 0. At the end of round 1, the simulator secures another forgery $\hat{\sigma}'$. Let I' be the target index for this particular round. The forking is successful if the target index for round 1 matches the one during round 0, *i.e.*, $I' = I$.

2.3.2.3 Analysis

Forking, when carried out using the Rewinding Technique, is successful if the target indices during the two rounds of simulation of the adversary match. In the Replicating Technique, on the other hand, there is an additional requirement that the simulator's guess of the target index has to be correct. This leads to additional degradation. Thus, at first glance, the Rewinding Technique seems to be the more efficient of the two approaches. The analysis concurs with this observation. The Splitting Lemma, discussed in §**2.3.1**, plays a central role in establishing the lower bound. In addition, we also use the following result.

Lemma 3 (Hölder's inequality[14])**.** *Let $q \in \mathbb{Z}^+$, $1 \leq n < \infty$ and $x_1, \ldots, x_q \geq 0$ be real numbers. Then*

$$\sum_{k:=1}^{q} x_k^n \geq \frac{1}{q^{n-1}} \left(\sum_{k:=1}^{q} x_k \right)^n .$$

Defining the notions of "good". Consider a general adversary $\mathcal{A}$ with a non-negligible advantage of ϵ. Let $\mathbb{T}$ denote the set of all random tapes participating in a single round of simulation of the adversary–the universe of tapes. This includes the internal coins of the adversary as well as the randomness from the random functions associated with the random oracle and the signature oracle. Provided that the adversary has a non-negligible advantage of ϵ, at least ϵ fraction of the tapes in $\mathbb{T}$ lead to a successful forgery. These constitute the good tapes in $\mathbb{T}$.

[13] We discuss a rather naïve approach that could be used to rewind the adversary, which is modelled as a probabilistic Turing machine. An elegant, more abstract, approach which involves the alternative modelling the adversary (as a deterministic Turing machine) is discussed later.

The state of progress of a Turing machine is (completely) determined by its current state (from the state register), the contents of the tape and the position of the head on the tape. The basic idea involved in rewinding is as follows. The simulator explicitly saves the state of progress of the adversary every time it queries the random oracle. To be precise, it saves the current state (from the state register), the contents of the tape and the position of the head on a separate, external, tape. The simulator, thus, may have to save at most q states of progress. At the end of the round, the simulator has sufficient information about the critical point. Besides, the state of progress of the adversary at this critical point–the critical state of progress–is highly likely to be one of the saved states of progress. Therefore, in order to rewind the adversary, the simulator has to just look up for this critical state of progress and restore the adversary to it.

[14] Although, the result is a corollary to a more general Hölder's inequality (see [BPW12, Lemma C.3]), another way to proving the bound is by viewing it an optimisation problem. Let $\mathrm{f}(x_1, \ldots, x_q) := \sum_{k:=1}^{q} x_k^n$ be the objective function under the set of constraints: *i*) $\sum_{k=1}^{q} x_k = x$, and *ii*) $(0 \leq x_k \leq 1)$ for $k \in \{1, \ldots, q\}$. The function f attains a minima of x^n/q^{n-1} at the point $(x/q, \ldots, x/q)$ which completes the proof.

Recall, from §**2.2.3**, that there must exist a target H-index I for the adversary to have produced the forgery with a non-negligible advantage. On this assumption, we define a more refined notion of good–termed $\mathsf{good}_{(i)}$–for the tapes in $\mathbb{T}$. For an arbitrary index $i \in \{1, \ldots, q\}$, a tape in $\mathbb{T}$ is deemed to be $\mathsf{good}_{(i)}$ if it leads to the adversary forging successfully *and* with a target H-index of i. It follows that a tape $\mathtt{T}$ is good is $\mathsf{good}_{(i)}$ for some i. Let's assume that ϵ_i fraction of the tapes in $\mathbb{T}$ are $\mathsf{good}_{(i)}$. Thus, we have

$$\sum_{i=1}^{q} \epsilon_i = \sum_{i=1}^{q} \Pr[I = i] = \Pr[I > 0] = \epsilon. \tag{2.5}$$

Replay attack, as already stated, involves simulating the adversary on two tapes that agree up to a certain juncture–the target oracle query. In order to accommodate this, we split the universe $\mathbb{T}$ into the two sets $\mathbb{T}_{(i-)}$ and $\mathbb{T}_{(i+)}$ using the Cartesian product. Here, $\mathbb{T}_{(i-)}$ denotes the (sub)universe random tapes involved in the simulation before the adversary makes the query $\mathtt{Q}_i$, whereas $\mathbb{T}_{(i+)}$, those after the query $\mathtt{Q}_i$. We assume that $\mathbb{T}$ is bijective to $\mathbb{T}_{(i-)} \times \mathbb{T}_{(i+)}$ through a "join" function (denoted by $\|$) and a "split" function.[15] The notion of $\mathsf{good}_{(i)}$, which was originally defined for the set $\mathbb{T}$, can be easily adapted for the set $\mathbb{T}_{(i-)} \times \mathbb{T}_{(i+)}$ using the bijection: a tape $(\mathtt{T}_{(i-)}, \mathtt{T}_{(i+)}) \in \mathbb{T}_{(i-)} \times \mathbb{T}_{(i+)}$ is deemed to be $\mathsf{good}_{(i)}$ if its counterpart in $\mathbb{T}$ (*i.e.*, $\mathtt{T}_{(i-)} \| \mathtt{T}_{(i+)}$) is $\mathsf{good}_{(i)}$. We denote the (sub)set of all such tapes in $\mathbb{T}_{(i-)} \times \mathbb{T}_{(i+)}$ by $\mathbb{V}_i$.

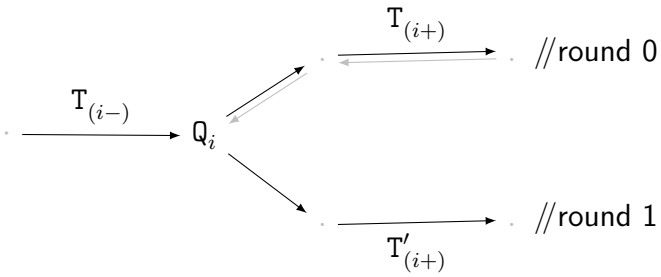

Figure 2.7: Tapes involved in the replay attack.

Applying the Splitting Lemma to the Replicating Technique. The Koblitz-Menezes version of the Splitting Lemma (**Lemma 2**), seems to be the more natural choice for analysing replay attack launched using the Replicating Technique. Let's assume that an arbitrary β_i fraction of the tape segments in $\mathbb{T}_{(i-)}$ are $\mathsf{better}_{(i)}$ in the sense that is described in **Lemma 2**. We denote this set by $\mathbb{T}^*_{(i-)}$. The exact value of β_i is determined later. Let $(\mathtt{T}_{(i-)}, \mathtt{T}_{(i+)})$ and $(\mathtt{T}_{(i-)}, \mathtt{T}'_{(i+)})$ be the tapes involved in the two rounds of simulation in a particular instance of

[15]The "join" function f_i concatenates the two constituent tape segments from the set $\mathbb{T}_{(i-)} \times \mathbb{T}_{(i+)}$ into a single tape in set $\mathbb{T}$, *i.e.*, $\mathsf{f}_i(\mathtt{T}_{(i-)}, \mathtt{T}_{(i+)}) := \mathtt{T}_{(i-)} \| \mathtt{T}_{(i+)}$. Provided that f_i is bijective, the inverse of f_i would be the function which "splits" a tape $\mathtt{T}$ into the two constituent tape segments $\mathtt{T}_{(i-)}$ and $\mathtt{T}_{(i+)}$ depending upon the random oracle query $\mathtt{Q}_i$, *i.e.*, $\mathsf{f}_i^{-1}(\mathtt{T}) := (\mathtt{T}_{(i-)}, \mathtt{T}_{(i+)})$. Note that the definition of f_i does not dependent of i and hence we consider a single join function f.

the replay attack, with

$$\mathtt{T}_{(i-)} \xleftarrow{\$} \mathbb{T}_{(i-)} \text{ and } (\mathtt{T}_{(i+)}, \mathtt{T}'_{(i+)}) \xleftarrow{\$} \mathbb{T}_{(i+)}. \tag{2.6}$$

Recall that i, here, is the simulator's guess of the target index. The forking is successful if both $(\mathtt{T}_{(i-)}, \mathtt{T}_{(i+)})$ and $(\mathtt{T}_{(i-)}, \mathtt{T}'_{(i+)})$ turn out to be $\mathsf{good}_{(I)}$ pairs and the guess of the target index turns out to be correct, *i.e.*, $I = i$. Let's denote the probability of this event by ϵ'_i. Thus, the probability with which the forking is successful (with the tapes sampled as shown in (2.6)) is given by:

$$\begin{aligned}\epsilon'_i &= \Pr\left[(\mathtt{T}_{(i-)}, \mathtt{T}_{(i+)}) \in \mathbb{V}_I \wedge (\mathtt{T}_{(i-)}, \mathtt{T}'_{(i+)}) \in \mathbb{V}_I \wedge I = i\right] \\ &= \Pr\left[(\mathtt{T}_{(i-)}, \mathtt{T}_{(i+)}) \in \mathbb{V}_i \wedge (\mathtt{T}_{(i-)}, \mathtt{T}'_{(i+)}) \in \mathbb{V}_i \mid I = i\right] \cdot \Pr\left[I = i\right]. \end{aligned} \tag{2.7}$$

As the index i is chosen uniformly at random from $\{1, \ldots, q\}$, the probability that it matches I is at least $1/q$. As per our definition, the tape $(\mathtt{T}_{(i-)}, \mathtt{T}_{(i+)})$ is $\mathsf{good}_{(i)}$ if $\mathtt{T}_{(i-)}$ is $\mathsf{better}_{(i)}$ and $\mathtt{T}_{(i+)}$ belongs to $\mathrm{k}(\mathtt{T}_{(i-)})$. The same applies to the tape $(\mathtt{T}_{(i-)}, \mathtt{T}'_{(i+)})$. Thus, (2.7) can be rewritten as follows:

$$\begin{aligned}\epsilon'_i &= \Pr\left[\mathtt{T}_{(i-)} \in \mathbb{T}^*_{(i-)} \wedge \mathtt{T}_{(i+)} \in \mathrm{k}(\mathtt{T}_{(i-)}) \wedge \mathtt{T}'_{(i+)} \in \mathrm{k}(\mathtt{T}_{(i-)}) \mid I = i\right] \cdot 1/q \\ &= \Pr\left[\mathtt{T}_{(i+)} \in \mathrm{k}(\mathtt{T}_{(i-)}) \wedge \mathtt{T}'_{(i+)} \in \mathrm{k}(\mathtt{T}_{(i-)}) \mid \mathtt{T}_{(i-)} \in \mathbb{T}^*_{(i-)}\right] \cdot \Pr\left[\mathtt{T}_{(i-)} \in \mathbb{T}^*_{(i-)}\right] \cdot 1/q \\ &= \Pr\left[\mathtt{T}_{(i+)} \in \mathrm{k}(\mathtt{T}_{(i-)}) \mid \mathtt{T}_{(i-)} \in \mathbb{T}^*_{(i-)}\right] \cdot \Pr\left[\mathtt{T}'_{(i+)} \in \mathrm{k}(\mathtt{T}_{(i-)}) \mid \mathtt{T}_{(i-)} \in \mathbb{T}^*_{(i-)}\right] \cdot \beta_i \cdot 1/q \\ &\qquad \text{(using } \textbf{Proposition 1} \text{ of } \textbf{Lemma 2}\text{)} \\ &\geq (\epsilon_i - \beta_i)^2 \beta_i / q \quad \text{(using } \textbf{Proposition 2} \text{ of } \textbf{Lemma 2}\text{)} \end{aligned} \tag{2.8}$$

The above expression attains a maxima[16] of $4\epsilon_i^3/27q$ at the point $\beta_i = \epsilon_i/3$. Thus, the probability with which the forking is successful for any i is given by

$$\epsilon' = \sum_{i=1}^{q} \epsilon'_i = \sum_{i=1}^{q} \frac{4}{27q} \epsilon_i^3 = \frac{4}{27q} \sum_{i=1}^{q} \epsilon_i^3. \tag{2.9}$$

Under the constraint given in (2.5), (2.9) attains a minima[17] at the point $\epsilon_i = \epsilon/q$ for all $1 \leq i \leq q$ and thus, we get the lower bound

$$\epsilon' \geq \frac{4}{27q} \sum_{i=1}^{q} (\epsilon/q)^3 = \frac{4}{27} (\epsilon/q)^3. \tag{2.10}$$

[16]The objective function is $\mathrm{f}(\beta_i) = (\epsilon_i - \beta_i)^2 \beta_i / q$ under the constraint $\beta_i < \epsilon_i$. The function attains maximum value at the point $\beta_i = \epsilon_i/3$.

[17]The objective function is

$$\mathrm{f}(\epsilon_1, \ldots, \epsilon_q) = \frac{4}{27q} \sum_{i=1}^{q} \epsilon_i^3$$

under the set of constraints $\sum_{i=1}^{q} \epsilon_i = \epsilon$ and $\epsilon_i \in [0, 1]$ for each $i \in \{1, \ldots, q\}$. The function attains minimum value at the point $(\epsilon/q, \ldots, \epsilon/q)$. This can also be established using Hölder's inequality (**Lemma 3**).

Applying the Splitting Lemma to the Rewinding technique. For analysing the replay attack launched using the Rewinding Technique, we use **Lemma 1**. Let's presume that an arbitrary β_i fraction of the tape pairs in $\mathbb{T}$ are $\mathsf{better}_{(i)}$ in the sense described in **Lemma 1**. We denote this set by $\mathbb{V}_i^*$. The exact value of β_i is, again, determined later. Let $\mathtt{T}$ (sampled at random from $\mathbb{T}$) be the tape involved in the simulation of the adversary in $\mathsf{round}\ 0$. If the adversary forges successfully, the simulator can determine the target index I. We split $\mathtt{T}$ into the two tapes $\mathtt{T}_{(I-)}$ and $\mathtt{T}_{(I+)}$ depending on this index I. Here, $\mathtt{T}_{(I-)}$ is the tape involved in the simulation before the adversary makes the query $\mathtt{Q}_I$, whereas $\mathtt{T}_{(I+)}$, that after the query $\mathtt{Q}_I$. Thus, $(\mathtt{T}_{(I-)}, \mathtt{T}_{(I+)})$ is a $\mathsf{good}_{(I)}$ pair. The simulator, next, rewinds the adversary to the point $\mathtt{Q}_I$ and simulates it on a fresh tape $\mathtt{T}'_{(I+)}$ (sampled at random from $\mathbb{T}_{(I+)}$). The replay attack is successful if $(\mathtt{T}_{(I-)}, \mathtt{T}'_{(I+)})$ also turns out to be a $\mathsf{good}_{(I)}$ pair. However, this is a necessary condition. For ease of analysis, we use the sufficient condition that $(\mathtt{T}_{(I-)}, \mathtt{T}_{(I+)})$ be better and $(\mathtt{T}_{(I-)}, \mathtt{T}'_{(I+)})$ be good. Thus, the probability with which the Rewinding Technique is successful (with the tapes sampled as described above) is given by:

$$\begin{aligned}
\epsilon'_I &= \Pr\left[(\mathtt{T}_{(I-)}, \mathtt{T}_{(I+)}) \in \mathbb{V}_I^* \wedge (\mathtt{T}_{(I-)}, \mathtt{T}'_{(I+)}) \in \mathbb{V}_I\right] \\
&= \Pr\left[(\mathtt{T}_{(I-)}, \mathtt{T}'_{(I+)}) \in \mathbb{V}_I \mid (\mathtt{T}_{(I-)}, \mathtt{T}_{(I+)}) \in \mathbb{V}_I^*\right] \cdot \Pr\left[(\mathtt{T}_{(I-)}, \mathtt{T}_{(I+)}) \in \mathbb{V}_I^*\right] \\
&= (\epsilon_I - \beta_I)\beta_I \quad \text{(using \textbf{Proposition 1} and \textbf{2} of \textbf{Lemma 1})} \qquad (2.11)
\end{aligned}$$

The above expression attains a maxima[18] of $\epsilon_I^2/4$ at the point $\beta_I = \epsilon_I/2$. Thus, the probability that the oracle replay attack is successful for any I is given by

$$\begin{aligned}
\epsilon' &= \sum_{I=1}^{q} \epsilon'_I \geq \sum_{I=1}^{q} \frac{\epsilon_I^2}{4} \\
&\geq \frac{\epsilon^2}{4q} \quad \text{(using Hölder's inequality)} \qquad (2.12)
\end{aligned}$$

That concludes the analysis. It is evident that the Rewinding Technique is more *efficient* than the Replicating Technique–the security degradation incurred (by the former) is just $\mathsf{O}(q)$ compared to $\mathsf{O}(q^3)$ (for the latter). Thus, it the Rewinding Technique that yields **Theorem 1**. Henceforth, by forking we *always* refer (unless explicitly mentioned) to replay attack carried out by rewinding the adversary.

We wrap up the discussion by introducing the notion of "characteristic" expressions and, also, with some remarks on the analyses. In the next section, we describe how the intricacy of forking (and also its analysis) can be separated out from the simulation of the adversary through the notion of general forking. We also describe the nested oracle replay attack which, in simple terms, involves two random oracles and (as the name suggests) multiple, nested, forkings.

[18]The objective function is $\mathsf{f}(\beta_I) = (\epsilon_I - \beta_I)\beta_I$ under the constraint $\beta_I < \epsilon_I$. The function attains maximum value at the point $\beta_I = \epsilon_I/2$.

The *characteristic* expression. The replay attack launched using Replicating Technique is captured, essentially, by the expression

$$\epsilon' \geq \frac{4}{27q} \sum_{i=1}^{q} \epsilon_i^3 \tag{2.13}$$

under the set of constraints: *i*) $\sum_{i=1}^{q} \epsilon_i = \epsilon$, and *ii*) $(0 \leq \epsilon_i \leq 1)_{i \in \{1,\ldots,q\}}$. We say that (2.13) is the *characteristic* expression for the Replicating Technique. Similarly, the characteristic expression for the Rewinding Technique is

$$\epsilon' \geq \frac{1}{4} \sum_{i=1}^{q} \epsilon_i^2 \tag{2.14}$$

under the same set of constraints. This abstraction comes in handy (in coming chapters) for analysing nested replay attacks.

Remark 5. After optimisation, we arrived at two different values of β_i for the Splitting Lemma: for the Replicating Technique, we used an $(\epsilon_i, \epsilon_i/3)$-Splitting Lemma, whereas, for the Rewinding Technique, an $(\epsilon_i, \epsilon_i/2)$-Splitting Lemma.

Remark 6. We denote the universe of tapes for two rounds of simulations, associated with a forking of the adversary, by $\mathbb{T}_{(1)}$ ('1' in the subscript denotes one forking), *i.e.*,

$$\begin{aligned} \mathbb{T}_{(1)} &= \bigcup_{i=1}^{q} \mathbb{T}_{(i-)} \times \mathbb{T}^2_{(i+)} \\ &= \bigcup_{i=1}^{q} \mathbb{T}_{(i-)} \times \mathbb{T}_{(1,i+)} \quad \text{(using shorthand notation)} \end{aligned}$$

The introduction of $\mathbb{T}_{(1)}$ allows us to define the "higher" notion of $\mathsf{good}_{(1)}$ based on the notions of good and $\mathsf{good}_{(i)}$. The triplet of tape segments $(\mathtt{T}_{(I-)}, (\mathtt{T}_{(I+)}, \mathtt{T}'_{(I+)})) \in \mathbb{T}_{(1)}$ is considered to be $\mathsf{good}_{(1)}$ if both the tapes $(\mathtt{T}_{(I-)}, \mathtt{T}_{(I+)})$ and $(\mathtt{T}_{(I-)}, \mathtt{T}'_{(I+)})$ are $\mathsf{good}_{(I)}$, *i.e.*, if the forking is successful. From the analyses, we had concluded the forking is successful with probability at least ϵ'. This result can be interpreted in terms of $\mathsf{good}_{(1)}$ as follows: a randomly sampled triplet from $\mathbb{T}_{(1)}$ is $\mathsf{good}_{(1)}$ with probability at least ϵ'. These notions come into play in the analysis of modified GG-IBS in §**4.3.2**.

2.4 General Forking

In the previous section, we saw how the machinery of oracle replay attack was used in the security argument of Schnorr signature scheme. However, we also saw that the probability analysis of the security argument, which culminated with the Forking Lemma (**Lemma 1**), was quite *involved*. Bellare and Neven [BN06] observed that

> "the Forking Lemma is something purely probabilistic, not about signatures"

and proposed a more abstract version called the General Forking (GF) Lemma. The motivation is to demarcate the probability analysis of the rewinding from that of the actual simulation, allowing for more modular security arguments. The concept of general forking is formulated in terms of randomised algorithms and its outputs, leaving out the notions of signature scheme as well as random oracles altogether. Thus, the alternative modelling of the adversary (as a deterministic Turing machine with the internal coins being passed explicitly as input.) which we had discussed in §**1.2**, comes into play. The claimed advantage is to allow for more modular and easy to verify proof of cryptographic schemes that apply the notion of forking in their security argument.

The modularity of the GF Lemma allows one to abstract out the probability analysis of the rewinding process from the actual simulation in the security argument. The gap between the abstract and the concrete is, then, bridged using the so-called "wrapper" algorithm. While the GF Algorithm takes care of the replay attack, it is the wrapper that handles the simulation of the protocol environment to the actual adversary. The reduction involves invoking the General Forking Algorithm (on the associated wrapper) and utilising its outputs to solve the underlying hard problem. So the design of the wrapper is central to any security argument involving GF Algorithm. In fact, the design depends on the actual protocol and the security model used–see, e.g., [BN06, BPW12] for the concrete design of the wrappers in their respective contexts.

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

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

Pick coins ρ for $\mathcal{W}$ at random

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

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

Role of the wrapper. Let's take a *simplistic* look[19] at how the GF Algorithm, together with the wrapper $\mathcal{W}$, is used to launch the oracle replay attack. The input to $\mathcal{W}$ comprises of some external randomness $(s_1, \ldots, s_q)$ and the internal coins (ρ) for the adversary, whereas the output, an index I. Consider the first invocation of $\mathcal{W}$ (on $s_1, \ldots, s_q; \rho$) within the GF Algorithm: $\mathcal{W}$ simulates the protocol environment to the actual adversary $\mathcal{A}$ having access

[19]For the time being, we do not consider the input string x or the side-output σ.

to ρ. $\mathcal{W}$ responds to the random oracle queries of $\mathcal{A}$ by using $s_1, \ldots, s_q$. At the end of the simulation, $\mathcal{W}$ outputs an index I that refers to the target query[20].

Next, the GF Algorithm invokes $\mathcal{W}$ on an input $(s_1, \ldots, s_{I-1}, s'_I, \ldots, s'_q; \rho)$ that is *related* to the first invocation. The behaviour of $\mathcal{A}$ remains identical to the first round of simulation, right up to the I^{th} random oracle query, at which point it diverges (assuming $s'_I \neq s_I$). This is tantamount to forking $\mathcal{A}$ at the index I. The forking is successful, if the target index for the second round of simulation is the same as that for the first, *i.e.*, $I' = I$. The probability of this event is governed by the General Forking Lemma given below. One can clearly see how the wrapper acts as an intermediary between the abstract GF Algorithm and the adversary in the concrete setting of the reduction.

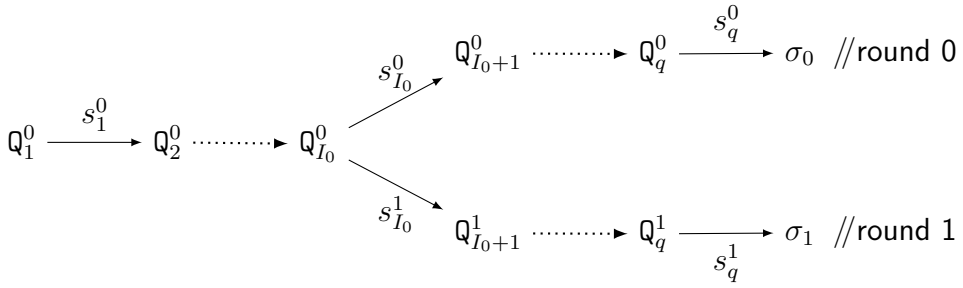

Figure 2.8: A successful forking of the wrapper $\mathcal{W}$ by $\mathcal{F}_{\mathcal{W}}$.

The cost of forking. The GF Lemma gives us a lower bound on the probability of success of the forking algorithm in terms of the success probability of the associated wrapper (and hence, the underlying adversary). Roughly speaking, the cost of forking can be measured in terms of the degradation incurred in the forking process. Let q denote the upper bound on the number of random oracle queries, then the cost of general forking is roughly $\mathsf{O}(q)$.

Lemma 4 (General Forking Lemma [BN06])**.** *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

$$\mathit{gfrk} := \Pr\left[(b = 1) : x \xleftarrow{\$} \mathcal{G}_I; (b, \sigma, \sigma) \xleftarrow{\$} \mathcal{F}_{\mathcal{W}}(x)\right] \quad \textit{and}$$
$$\mathit{acc} := \Pr\left[I \geq 1 : x \xleftarrow{\$} \mathcal{G}_I; \{s_1, \ldots, s_q\} \xleftarrow{\cup} \mathbb{S}; (I, \sigma) \xleftarrow{\$} \mathcal{W}(x, s_1, \ldots, s_q)\right],$$

then

$$\mathit{gfrk} \geq \mathit{acc} \cdot \left(\frac{\mathit{acc}}{q} - \frac{1}{|\mathbb{S}|}\right). \tag{2.15}$$

Remark 7. The analysis of the GF Algorithm is quite different from that of Rewinding Technique. It is done through a random variable that models $\mathcal{W}$ as a randomised algorithm. However, the resulting characteristic expression is the *same* as that for the Rewinding Technique given in (2.14). Thus, the characteristic expression *links* the two approaches.

[20]Recall that, the random oracle query that is used by $\mathcal{A}$ to produce its desired output is termed the target query and the index of this query is the target index.

2.5 Nested Oracle Replay Attacks and Multiple Forking

So far, we have seen replay attacks involving *one* random oracle and a *single* forking of the adversary–an *elementary* oracle replay attack. Boldyreva *et al*. [BPW12] generalised the concept of forking, further, leading to the Multiple-Forking (MF) Lemma. The immediate motivation behind this new abstraction was to argue the security of a proxy signature scheme that uses more than one hash functions. The hash functions are modelled as random oracles and the MF Algorithm allows one to mount the nested replay attacks by rewinding the adversary several times on related inputs. In particular, a nested replay attack involves two random oracles and multiple forkings on the two oracles. Hence, unlike elementary replay attack, there are *two* critical points for a round.

A (successful) nested replay attack with three forkings is illustrated in **Figure 2.9**. The adversary is forked at the critical points $\mathtt{Q}_i^0$, $\mathtt{Q}_j^0$ and $\mathtt{Q}_i^2$ (in that order[21]) and the forking is successful if the target indices (of the two random oracles involved) in all the four rounds is *same* (see (2.18) for the exact condition). The more abstract notion of multiple forking still retains the modularity advantage of general forking and has been applied in several other security arguments [GG09, CMW12, CKK13] in a more-or-less black-box fashion.

We begin with a Pointcheval-Stern-style analysis of the nested replay attack (given in **Figure 2.9**) and then describe the original MF Algorithm [BPW12] (but, with some notational changes).

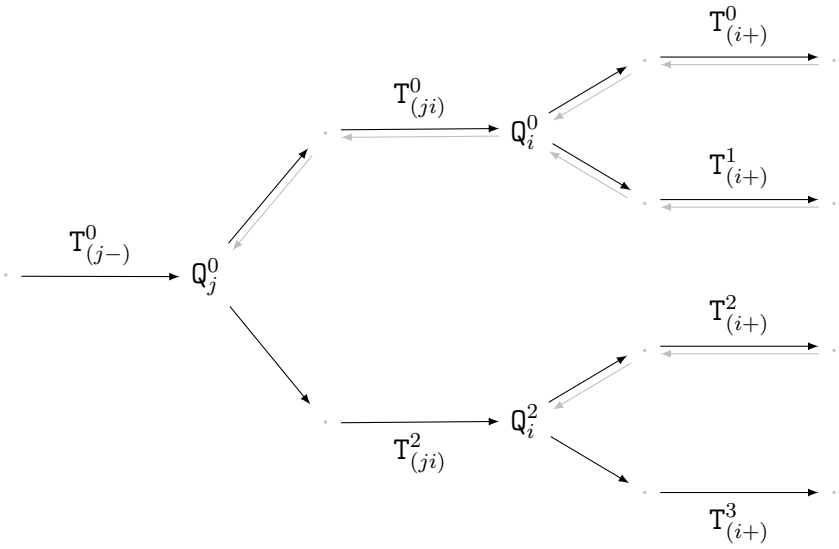

Figure 2.9: Nested replay attack with three forkings.

2.5.1 Analysis

The nested replay attack given in **Figure 2.9** cannot be direct analysed using the Splitting Lemma (at least, not at the moment: see §**4.3.2**). We need to extend the Splitting Lemma

[21] Further forkings can be carried out at $\mathtt{Q}_j^0$ and $\mathtt{Q}_i^4$ (in that order) and so forth.

(**Lemma 1**) to suit the purpose.

2.5.1.1 The Extended Splitting Lemma

Let $\mathbb{W}$, $\mathbb{X}$ and $\mathbb{Y}$ be finite sets (with some underlying probability distribution) with $|\mathbb{W}| = \ell$, $|\mathbb{X}| = m$ and $|\mathbb{Y}| = n$. Also, let $\mathbb{T}$ denote the "universal" set $\mathbb{W} \times \mathbb{X} \times \mathbb{Y}$. A triple $(w, x, y) \in \mathbb{W} \times \mathbb{X} \times \mathbb{Y}$ is deemed to be good if it satisfies a certain property. Let $\mathbb{V}$ denote the set of all such good triples. Suppose that at least γ fraction of the triples in $\mathbb{W} \times \mathbb{X} \times \mathbb{Y}$ are good, *i.e.*, $\Pr[\mathbb{V}] \geq \gamma$. We define a function $\mathrm{k} : \mathbb{W} \times \mathbb{X} \mapsto \mathfrak{P}(\mathbb{Y})$ as

$$\mathrm{k}(w, x) \stackrel{\text{def}}{=} \{y \in \mathbb{Y} \mid (w, x, y) \in \mathbb{V}\}.$$

k can be used to count, for a particular (w, x), the number of elements $y \in \mathbb{Y}$ such that (w, x, y) forms a good triple. For some $\beta < \gamma$, we define the (sub)set of better triples of $\mathbb{V}$ as

$$\mathbb{V}^* \stackrel{\text{def}}{=} \{(w, x, y) \in \mathbb{V} \mid |\mathrm{k}(w, x)| \geq (\gamma - \beta)\}.$$

Thus, a good triple (w, x, y) is better if (w, x) forms good triples with at least $(\gamma - \beta)$ fraction of the elements of $\mathbb{Y}$. We define another function $\mathrm{g} : \mathbb{W} \mapsto \mathfrak{P}(\mathbb{X} \times \mathbb{Y})$ as

$$\mathrm{g}(w) \stackrel{\text{def}}{=} \{(x, y) \in \mathbb{X} \times \mathbb{Y} \mid (w, x, y) \in \mathbb{V}^*\}.$$

g can be used to count, for a particular w, the number of elements $(x, y) \in \mathbb{X} \times \mathbb{Y}$ such that (w, x, y) forms a better triple. For some $\alpha < \beta$, we define the best triples of $\mathbb{V}^*$ as

$$\mathbb{V}^{**} \stackrel{\text{def}}{=} \{(w, x, y) \in \mathbb{V}^* \mid |\mathrm{g}(w)| \geq (\beta - \alpha)\}.$$

Thus, a better triple (w, x, y) is best if (w) forms better triples with at least $(\beta - \alpha)$ fraction of the elements of $\mathbb{X} \times \mathbb{Y}$.

Lemma 5 (The Extended Splitting Lemma)**.** *The following propositions hold on the aforementioned assumptions:*

1. *i)* $\Pr[\mathbb{V}^*] \geq \beta$*, i.e., at least* β *fraction of triples in* $\mathbb{T}$ *are better;* ***and*** *ii)* $\Pr[\mathbb{V}^{**}] \geq \alpha$*, i.e., at least* α *fraction of triples in* $\mathbb{T}$ *are best.*

2. *i)* $\Pr[(w, x, y') \in \mathbb{V} \mid (w, x, y) \in \mathbb{V}^*; y' \stackrel{\$}{\leftarrow} \mathbb{Y}] \geq (\gamma - \beta)$*, i.e., given a better triple* (w, x, y)*, the probability with which the triple* (w, x, y') *(with* y' *sampled randomly from* $\mathbb{Y}$*) is good is at least* $(\gamma - \beta)$*;* ***and*** *ii)* $\Pr[(w, x', y') \in \mathbb{V}^* \mid (w, x, y) \in \mathbb{V}^{**}]; x' \stackrel{\$}{\leftarrow} \mathbb{X}; y' \stackrel{\$}{\leftarrow} \mathbb{Y}] \geq (\beta - \alpha)$ *(interpreted as in* ***2.i****).*

3. *i)* $\Pr[\mathbb{V}^* \mid \mathbb{V}] \geq \beta/\gamma$*, i.e., a good pair is better with a probability of at least* β/γ*;* ***and*** *ii)* $\Pr[\mathbb{V}^{**} \mid \mathbb{V}^*] \geq \alpha/\beta$*, i.e., a better pair is best with a probability of at least* β/γ*;* ***and*** *iii)* $\Pr[\mathbb{V}^{**} \mid \mathbb{V}] \geq \alpha/\gamma$*, i.e., a good pair is best with a probability of at least* α/γ*.*

Argument. The argument is quite similar to that of the Splitting Lemma. It is not difficult to see that **Proposition 2.i** and **2.ii** follow from the definitions (of better and best sets, respectively). The proof of **Proposition 1.i** is quite similar to that of **Proposition 1** of the Splitting

Lemma (*i.e.*, through contradiction). But it is given next for the sake of completeness. Suppose $\Pr[\mathbb{V}] \geq \gamma$ and $\Pr[\mathbb{V}^*] < \beta$, then we have

$$\Pr[\mathbb{V}] = \Pr[\mathbb{V}^*] + \Pr[\mathbb{V} \setminus \mathbb{V}^*] < \beta + \Pr[\mathbb{V} \setminus \mathbb{V}^*]. \tag{2.16}$$

Now, there can be at most ℓm distinct (w, x) elements in $\mathbb{V} \setminus \mathbb{V}^*$. In addition, for each such $(w, x) \in \mathbb{V} \setminus \mathbb{V}^*$, $|\mathsf{k}(w, x)| < (\gamma - \beta)n$ (by definition). Therefore (2.16) yields,

$$\Pr[\mathbb{V}] < \beta + \frac{|\mathsf{k}(x, w)|.\ell m}{\ell mn} < \beta + \frac{(\gamma - \beta)n.\ell m}{\ell mn} = \gamma.$$

This contradicts our assumption that at least γ fraction of elements in $\mathbb{T}$ are good ($\Pr[\mathbb{V}] \geq \gamma$), thus, establishing the proposition. The argument for **Proposition 2.ii** is similar as well. Suppose $\Pr[\mathbb{V}^*] \geq \beta$ and $\Pr[\mathbb{V}^{**}] < \alpha$, then we have

$$\Pr[\mathbb{V}^*] = \Pr[\mathbb{V}^{**}] + \Pr[\mathbb{V}^* \setminus \mathbb{V}^{**}] < \alpha + \Pr[\mathbb{V}^* \setminus \mathbb{V}^{**}]. \tag{2.17}$$

Now, there can be at most ℓ distinct w elements in $\mathbb{V}^* \setminus \mathbb{V}^{**}$. In addition, for each such $w \in \mathbb{V} \setminus \mathbb{V}^*$, $|\mathsf{g}(w)| < (\gamma - \beta)mn$ (by definition). Therefore (2.17) yields,

$$\Pr[\mathbb{V}^*] < \alpha + \frac{|\mathsf{g}(w)|.\ell}{\ell mn} < \alpha + \frac{(\beta - \alpha)mn.\ell}{\ell mn} = \beta.$$

This contradicts our assumption ($\Pr[\mathbb{V}^*] \geq \gamma$), thus, establishing **Proposition 1.ii**. Finally, we turn to **Proposition 3.i** through **3.iii**. They can be established, as in **Proposition 3** of the Splitting Lemma, using Bayes' theorem. This concludes the argument. □

2.5.1.2 Probability Analysis

Let H and G be the two random oracles involved in the nested replay attack. Nested forking, when carried out using rewinding, is successful if the target indices (of both G and H) during all four rounds of simulation of the adversary *match*. Thus, if (J_k, I_k) denote the target H and G indices, respectively, for round k, then the success condition is:

$$(I_3, J_3) = (I_2, J_2) = (I_1, J_1) = (I_0, J_0) \tag{2.18}$$

Defining the notions of "good". Consider a general adversary $\mathcal{A}$ with the universe of tapes denoted by $\mathbb{T}$ (as in §**2.3.2.3**). Provided that $\mathcal{A}$ has a non-negligible advantage of ϵ, at least ϵ fraction of the tapes in $\mathbb{T}$ lead to its success (*e.g.*, it forging successfully). These constitute the good tapes in $\mathbb{T}$. The refined notion for good is a bit different as there are *two* random oracles in consideration. Let j [resp. i] denote the target H-index [resp. G-index] (see **Footnote 3**). On this assumption, for arbitrary indices $(j, i) \in \{1, \ldots, q\}^2$, a tape in $\mathbb{T}$ is deemed to be $\mathsf{good}_{(j,i)}$ if it leads to the adversary being successful *and* with a target H-index of j and a target G-index of i. It follows that a tape T is good is $\mathsf{good}_{(j,i)}$ for some $(j, i) \in \{1, \ldots, q\}^2$. Let's assume that $\epsilon_{j,i}$ fraction of the tapes in $\mathbb{T}$ are $\mathsf{good}_{(j,i)}$. Thus, we

have

$$\sum_{j=1}^{q}\sum_{i=1}^{q}\epsilon_{j,i} = \sum_{j=1}^{q}\sum_{i=1}^{q}\Pr\left[J=j\wedge I=i\right] = \Pr\left[J>0\wedge I>0\right] = \epsilon. \tag{2.19}$$

To analyse forking involving two oracles, we need split the universe $\mathbb{T}$ into the three sets $\mathbb{T}_{(j-)}$, $\mathbb{T}_{(ji)}$ and $\mathbb{T}_{(i+)}$ using the Cartesian product. As usual, we assume that $\mathbb{T}$ is bijective to $\mathbb{T}_{(j-)} \times \mathbb{T}_{(ji)} \times \mathbb{T}_{(i+)}$ through a join and a split function and use this bijection to tailor the notion of $\mathsf{good}_{(j,i)}$ for this set. We denote the (sub)set of all such tapes by $\mathbb{V}_{j,i}$.

Applying the Extended Splitting Lemma. Let's presume that an arbitrary $\beta_{j,i}$ and $\alpha_{j,i}$ fraction of the tape triples in $\mathbb{T}$ are $\mathsf{better}_{(j,i)}$ and $\mathsf{best}_{(j,i)}$, respectively. We denote these sets, respectively, by $\mathbb{V}^{*}_{j,i}$ and $\mathbb{V}^{**}_{j,i}$. The exact value of $\beta_{j,i}$ and $\alpha_{j,i}$ is, again, determined later. The tapes involved are shown in **Figure 2.9**. The probability with which the nested replay attack is successful is calculated as follows. Refer to **Table 2.1** for notation.

Symbol	Denotes
$\mathtt{T}^0$	$(\mathtt{T}^0_{(j-)}, \mathtt{T}^0_{(ji)}, \mathtt{T}^0_{(i+)})$
$\mathtt{T}^1$	$(\mathtt{T}^0_{(j-)}, \mathtt{T}^0_{(ji)}, \mathtt{T}^1_{(ji+)})$
$\mathtt{T}^2$	$(\mathtt{T}^0_{(j-)}, \mathtt{T}^2_{(ji)}, \mathtt{T}^2_{(ji+)})$
$\mathtt{T}^3$	$(\mathtt{T}^0_{(j-)}, \mathtt{T}^2_{(ji)}, \mathtt{T}^3_{(ji+)})$

Table 2.1: Shorthand for tapes.

$$\begin{aligned}
\epsilon'_{j,i} &= \Pr\left[\mathtt{T}^0 \in \mathbb{V}^{**}_{j,i} \wedge \mathtt{T}^1 \in \mathbb{V}_{j,i} \wedge \mathtt{T}^2 \in \mathbb{V}^{*}_{j,i} \wedge \mathtt{T}^3 \in \mathbb{V}_{j,i}\right] \\
&= \Pr\left[\mathtt{T}^1 \in \mathbb{V}_{j,i} \mid \mathtt{T}^0 \in \mathbb{V}^{**}_{j,i}\right] \cdot \Pr\left[\mathtt{T}^3 \in \mathbb{V}_{j,i} \wedge \mathtt{T}^2 \in \mathbb{V}^{*}_{j,i} \mid \mathtt{T}^0 \in \mathbb{V}^{**}_{j,i}\right] \cdot \Pr\left[\mathtt{T}^0 \in \mathbb{V}^{**}_{j,i}\right] \\
&\geq \Pr\left[\mathtt{T}^1 \in \mathbb{V}_{j,i} \mid \mathtt{T}^0 \in \mathbb{V}^{*}_{j,i}\right] \cdot \Pr\left[\mathtt{T}^3 \in \mathbb{V}_{j,i} \mid \mathtt{T}^2 \in \mathbb{V}^{*}_{j,i}\right] \cdot \Pr\left[\mathtt{T}^2 \in \mathbb{V}^{*}_{j,i} \mid \mathtt{T}^0 \in \mathbb{V}^{**}_{j,i}\right] \cdot \alpha_{j,i} \\
&\qquad \text{(using } \textbf{Proposition 1.ii}\text{)} \\
&= \alpha_{j,i}(\beta_{j,i}-\alpha_{j,i})(\gamma_{j,i}-\beta_{j,i})^2 \quad \text{(using } \textbf{Proposition 2.i} \text{ and } \textbf{2.ii}\text{)}
\end{aligned}$$

The expression attains a maxima[22] of $\epsilon^4_{j,i}/4^3$ at the point $(\epsilon_{j,i}/4, \epsilon_{j,i}/2, \epsilon_{j,i})$. Thus, the probability that the nested replay attack is successful for any (j,i) is given by

$$\begin{aligned}
\epsilon' = \sum_{j=1}^{q}\sum_{i=1}^{q}\epsilon'_{j,i} &\geq \sum_{j=1}^{q}\sum_{i=1}^{q}\frac{\epsilon^4_{j,i}}{4^3} \\
&\geq \frac{\epsilon^4}{4^3 q^6} \quad \text{(using Hölder's inequality)}
\end{aligned} \tag{2.20}$$

[22]The objective function is $\mathrm{f}(\beta_{j,i}, \alpha_{j,i}) = \alpha_{j,i}(\beta_{j,i}-\alpha_{j,i})(\epsilon_{j,i}-\beta_{j,i})^2$ under the constraint $(0 < \alpha_{j,i} < \beta_{j,i} < \epsilon_{j,i}) < 1$. The function attains maximum value at the point $(\epsilon_{j,i}/4, \epsilon_{j,i}/2, \epsilon_{j,i})$.

Remark 8. The result can be extended for an arbitrary n number of forkings and we get $\epsilon' \geq \epsilon^{n+1}/4^n q^{2n}$.

The characteristic expression. The characteristic expression, evidently, is:

$$\epsilon' \geq \frac{1}{4^n} \sum_{j=1}^{q} \sum_{i=1}^{q} \epsilon_{ij}^{n+1} \tag{2.21}$$

under the set of constraints

$$\sum_{j=1}^{q} \sum_{i=1}^{q} \epsilon_{ij} = \epsilon \text{ and } (0 \leq \epsilon_{ij} \leq 1)_{(i,j)\in\{1,\ldots,q\}^2}. \tag{2.22}$$

The expression attains a minima of $\epsilon^{n+1}/4^n q^{2n}$ at the point $\epsilon_{ij} = \epsilon/q^2$ for $1 \leq i, j \leq q$, and hence the success probability of $\Omega(\epsilon^{n+1}/q^{2n})$.

2.5.2 Multiple Forking

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

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

Pick coins ρ for $\mathcal{W}$ at random

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

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

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

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

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

While the role of wrapper remains the same in the MF Algorithm, there are a few significant changes in its actual structure. The wrapper now simulates two random oracles and hence its output contains a pair of indices (I, J) with $J < I$. The two indices are usually associated to the target queries made to the two random oracles involved in the replay attack. For reductions employing the MF Algorithm, the design of the wrapper becomes a bit more involved because of the additional index in its output. In particular, the relative "order" among the two target oracle queries must be taken into consideration in the design of the wrapper. (We'll later see how neglecting the order of the indices, or even worse, using the MF Algorithm as a black-box may lead the reductions to fail.)

The cost of multiple forkings. According to the MF Lemma (see **Lemma 6**), the cost of n forkings (so the wrapper is called $n + 1$ times), carried out as per the specifications in **Algorithm 2**, is roughly $\mathrm{O}(q^{2n})$, where q is the sum of the upper bound on the queries to the random oracles involved. The degradation, in turn, can be attributed to the set of conditions

$\mathbb{A} := \{\mathsf{B}, \mathsf{C}_0, \ldots, \mathsf{C}_{n-1}, \mathsf{C}_2, \ldots, \mathsf{D}_{n-1}\}$ where

$$\begin{aligned} \mathsf{B} &: (I_0 \geq 1) \wedge (J_0 \geq 1) \\ \mathsf{C}_k &: (I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (s^{k+1}_{I_k} \neq s^k_{I_k}) \ \text{(for } k = 0, 2, \ldots, n-1) \\ \mathsf{D}_k &: (I_k, J_k) = (I_0, J_0) \wedge (s^k_{J_0} \neq s^{k-1}_{J_0}) \ \text{(for } k = 2, 4, \ldots, n-1) \end{aligned} \tag{2.23}$$

To be precise, the MF Algorithm is successful in the event E that all of the conditions in $\mathbb{A}_0$ are satisfied, *i.e.*,

$$\mathsf{E} : \mathsf{B} \wedge \left(\mathsf{C}_0 \wedge \mathsf{C}_2 \wedge \cdots \wedge \mathsf{C}_{n-1}\right) \wedge \left(\mathsf{D}_2 \wedge \mathsf{D}_4 \wedge \cdots \wedge \mathsf{D}_{n-1}\right). \tag{2.24}$$

The probability of this event, which is denoted by *mfrk*, is bounded by the MF lemma given below.

Lemma 6 (Multiple-Forking Lemma [BPW12])**.** *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

$$\begin{aligned} \mathit{mfrk} &:= \Pr\left[(b = 1) : x \xleftarrow{\$} \mathcal{G}_I; (b, \{\sigma_0, \ldots, \sigma_n\}) \xleftarrow{\$} \mathcal{M}_{\mathcal{W},n}(x)\right] \ \textit{and} \\ \mathit{acc} &:= \Pr\left[(I \geq 1) \wedge (J \geq 1) : x \xleftarrow{\$} \mathcal{G}_I; \{s_1, \ldots, s_q\} \xleftarrow{\cup} \mathbb{S}; (I, J, \sigma) \xleftarrow{\$} \mathcal{W}(x, s_1, \ldots, s_q)\right] \end{aligned}$$

then

$$\mathit{mfrk} \geq \mathit{acc} \cdot \left(\frac{\mathit{acc}^n}{q^{2n}} - \frac{n}{|\mathbb{S}|}\right). \tag{2.25}$$

The characteristic expression. The approach used for analysing the MF Algorithm is quite similar to that used for the GF Algorithm–*i.e.*, through random variables. However, as in the case of General Forking/Rewinding, the characteristic expression that we end up with is the same as for the nested replay attack (*i.e.*, (2.21) under the set of constraints given in (2.22)).

Remark 9 (On the degradation)**.** Note that the MF Algorithm causes a substantial amount of degradation to the reduction. The dominant part of degradation incurred is due to the condition

$$(I_n, J_n) = (I_{n-1}, J_{n-1}) = \cdots = (I_0, J_0) \tag{2.26}$$

which is necessary for the success of the MF Algorithm. We call (2.26) the "core" condition for the MF Algorithm. Each of the n equality checks in the expression (on a high level) contributes a factor of $\mathrm{O}(q^2)$, leading to the overall degradation of $\mathrm{O}(q^{2n})$. In **Chapter 4**, we will revisit the expression with the objective of minimising the degradation on mind.

Remark 10 (Forking in terms of congruences and unknowns.)**.** In the security argument of the Schnorr signature (which was discussed in the previous section), forking was used, in a vague sense, to obtain two linear congruences in two unknowns, given a linear congruence in two unknowns. In the same vein, Multiple-Forking (for n=3) can be considered to be a mechanism to obtain four linear congruence in four unknowns given a linear congruence in three unknowns.

Chapter 3

Galindo-Garcia IBS, Revisited

3.1 Introduction

In Africacrypt 2009, Galindo and Garcia [GG09] proposed a lightweight IBS scheme based on the Schnorr signature scheme. The construction is simple and claimed to be the most efficient IBS till date. The security argument 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. Likewise, $\mathcal{B}_2$ is used to solve the DLP in case the complement of E occurs. Both the reductions use the MF Lemma (**Lemma 6**) to show that the DLP is reduced to breaking the IBS scheme. 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.

Revisiting the security argument. Critical examination of the security argument of a cryptographic construction to see whether the claimed security is indeed achieved or not is an important topic in cryptographic research. Two such well-known examples are Shoup's work on OAEP [Sho01] and Galindo's work on Boneh-Franklin IBE [Gal05]. Another important question in the area of provable security is to obtain tighter security reduction for existing constructions. One such classical example is Coron's analysis of FDH [Cor00].

With this in mind, in this chapter, we revisit the security argument given in [GG09]. Our contributions are two fold; we: *i*) identify several problems in the original argument, and *ii*) provide a detailed new security argument that allows significantly tighter reductions. In particular, we show that the reduction $\mathcal{B}_1$ in [GG09] *fails* in the standard security model for IBS [BNN04], while the reduction $\mathcal{B}_2$ is *incomplete*. Moreover, the details of the wrapper algorithms has been *neglected* in both the reductions–the forking algorithms have been used in a more or less black-box manner. 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. The reduction $\mathcal{R}_1$ uses the GF Lemma (**Lemma 4**) and the technique first introduced by Coron [Cor00] to prove the security of FDH. We show that this results in a significantly tighter security reduction. On the other hand, both $\mathcal{R}_2$ and $\mathcal{R}_3$ are

structurally similar to $\mathcal{B}_2$ but uses two different versions of the MF Lemma, together with an algebraic technique similar to one adopted by Boneh-Boyen in [BB04a]. The security reduction $\mathcal{R}_2$ is also significantly tighter than the original $\mathcal{B}_2$. All the three reductions use the programmability of the random oracles in a crucial way.

As the *bottom-up* approach–building complex systems from simpler, more primitive, systems–is prevalent in cryptography, sound security arguments are crucial in preventing the logical fallacies from getting carried over. Galindo-Galindo IBS (GG-IBS), due to its efficiency and simplicity, has been used as a building block for numerous systems [RS11, XW12]. Thus, to ensure that they rest on solid ground, it is important to give a sound argument for GG-IBS.

3.2 Revisiting the Galindo-Garcia Security Argument

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

3.2.1 The Construction

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

Set-up, $\mathcal{G}(1^\kappa)$: Invoke the group generator $\mathcal{G}_{\mathsf{DL}}$ (on 1^κ) to obtain $(\mathbb{G}, g, p)$. Select $z \xleftarrow{\text{U}} \mathbb{Z}_p$ and set $Z = g^z$. Return z as the master secret key $\mathtt{msk}$ and $(\mathbb{G}, p, g, Z, \mathsf{H}, \mathsf{G})$ as the master public key $\mathtt{mpk}$, where H and G are hash functions

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

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

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

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

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

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

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

Figure 3.1: The (Original) Galindo-Garcia IBS.

Remark 11. Note that, although R is a part of the secret key of a user it is actually public information. In fact, R also forms a part of the signatures given by that user. Hence, by construction, all signatures generated using the user secret key $\mathtt{usk} = (y, R)$ will contain the same R. This also means that, in the security argument, the simulator *has* to maintain the same R for a particular user; otherwise, the simulation will diverge from the actual protocol execution.

Remark 12. There are two hash functions in consideration: H and G. H is used to generate the user secret key which, in turn, is required to sign on a message (using G). Hence H $<$ G ($<$ denotes 'followed by') constitutes the *logical* order for calling the hash functions.

GG-IBS in terms of Schnorr Signature. In the beginning of the section, we had given a (vague) description of GG-IBS in terms of the Schnorr signature. Let's elaborate on the idea. Let $\{\mathcal{G}_s, \mathcal{K}_s, \mathcal{S}_s, \mathcal{V}_s\}$ be the Schnorr signature scheme. The Key Extraction algorithm of GG-IBS can be realised using the Schnorr Signing algorithm $\mathcal{S}_s$ and the hash function H. The identity of the user acts as the message, the master secret key as the signing key and, *i.e.*,

$$\mathcal{E}(\mathtt{id}, \mathtt{msk}) = \mathcal{S}_{s,\mathrm{H}}(\mathtt{id}, \mathtt{msk}).$$

Similarly, if $\mathtt{usk}$ denotes the secret key for the user corresponding to $\mathtt{id}$, the Signing algorithm of GG-IBS is realised using $\mathcal{S}_s$ as shown below.

$$\mathcal{S}(\mathtt{id}, m, \mathtt{usk}) = \mathcal{S}_{s,\mathrm{G}}(m, \mathtt{usk})$$

3.2.2 The Security Argument and Problems with it

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

In reduction $\mathcal{B}_2$, the problem instance is embedded as a part of the master public key and hence the master secret key is not known to $\mathcal{B}_2$. The extract queries are answered by using an *algebraic technique* similar to the one in [BB04a]. The signature queries, on the other hand, are answered by invoking the signing algorithm $\mathcal{S}$ *after* the user secret key has been generated as in the extract query. Finally, $\mathcal{B}_2$ uses the MF Algorithm $\mathcal{M}_{\mathcal{W},3}$ (see §**2.5**) to obtain four related forgeries and uses these forgeries to solve the DLP challenge.

On the other hand, the strategy adopted in $\mathcal{B}_1$ is quite different from $\mathcal{B}_2$. The central idea is to embed the problem instance in the randomiser R while choosing its own master keys. In the simulation, $\mathcal{B}_1$ randomly chooses one of the identities involved in G-oracle query as the target identity. For signature queries involving this target identity, $\mathcal{B}_1$ embeds the problem instance in R and then uses the aforementioned algebraic technique to give the signature. The circularity involved is resolved by programming the random oracles. If $\mathcal{A}$ makes an extract query on the target identity, $\mathcal{B}_1$ fails; for all other identities, $\mathcal{B}_1$ uses the knowledge of the master secret key to respond to the queries. Finally it hopes that $\mathcal{A}$ returns a forgery containing R (in which the problem instance is embedded). In order to solve the DLP, $\mathcal{B}_1$ needs to obtain two such forgeries. This is accomplished with the help of the MF Algorithm $\mathcal{M}_{\mathcal{W},1}$.

The original argument. We now reproduce the original reductions from [GG09, §4] using our notation (for ease of reference, the bullets are replaced by numeric values). Each reduction is followed, immediately, by the observations that cause its failure. In the following, $\mathcal{B}_i.j$ refers to the j^{th} step in the construction of $\mathcal{B}_i$, $i \in \{1, 2\}$. The description of the forking algorithms $\mathcal{F}_{\mathcal{W}}$ and $\mathcal{M}_{\mathcal{W},n}$ is given in §**2.4** and §**2.5** respectively. (Some of the "typos" in the original security argument, that were corrected, are mentioned in the footnotes.)

Let $\mathcal{A}$ be an adversary against GG-IBS in the EU-ID-CMA model. Eventually, $\mathcal{A}$ outputs an attempted forgery of the form $\sigma = (A, B, R)$. Let 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 and Garcia construct an algorithm $\mathcal{B}_1$ [resp. $\mathcal{B}_2$] that breaks the DLP in case of event E [resp. NE].

3.2.2.1 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 $\alpha \xleftarrow{\text{U}} \mathbb{Z}_p$ and tries to extract the discrete logarithm α. The protocol environment is simulated as shown below.

$\mathcal{B}_1.1$ $\mathcal{B}_1$ picks $\hat{i} \xleftarrow{\text{U}} \{1, \ldots, q_{\text{G}}\}$, where q_{G} is the maximum number of queries that the adversary $\mathcal{A}$ performs to the G-oracle. Let $\hat{\texttt{id}}$ (the target identity) be the identity in the $\hat{i}^{\text{th}}$ query to the G-oracle. Next, $\mathcal{B}_1$ chooses $z \xleftarrow{\text{U}} \mathbb{Z}_p$ and sets $(\texttt{mpk}, \texttt{msk}) := ((\mathbb{G}, g, p, \text{G}, \text{H}, g^z), z)$, 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 $\mathfrak{L}_{\text{G}}$ and $\mathfrak{L}_{\text{H}}$ 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 $\mathcal{O}_e$, for user $\texttt{id}$, $\mathcal{B}_1$ chooses $c, y \xleftarrow{\text{U}} \mathbb{Z}_p$, sets $R := g^{-zc} g^y$ and adds $\langle R, \texttt{id}, c\rangle$ to the table $\mathfrak{L}_{\text{H}}$. Then it returns the key (y, R) to $\mathcal{A}$.

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

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

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

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

Pick random coins ρ for $\mathcal{W}$

$s_1^0, \ldots, s_{q_\mathrm{G}}^0 \xleftarrow{U} \mathbb{Z}_p$

$(I_0, J_0, \sigma_0) \leftarrow \mathcal{W}(\mathtt{mpk}, s_1^0, \ldots, s_{q_\mathrm{G}}^0; \rho)$

If $(I_0 = 0 \vee J_0 = 0)$ then return $\perp$

$s_{I_0}^1, \ldots, s_{q_\mathrm{G}}^1 \xleftarrow{U} \mathbb{Z}_p$

$(I_1, J_1, \sigma_1) \leftarrow \mathcal{W}(\mathtt{mpk}, s_1^0, \ldots, s_{I_0-1}^0, s_{I_0}^1, \ldots, s_{q_\mathrm{G}}^1, \rho)$

If $((I_1, J_1) \neq (I_0, J_0) \vee s_{I_0}^1 = s_{I_0}^0)$ then return $\perp$

Otherwise return (σ_0, σ_1)

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

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

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

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

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

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

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

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

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

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

However, it holds only if

$$(\alpha t - zc - \alpha)\, d \equiv 0 \bmod p. \tag{3.1}$$

It is easy to check that the LHS in (3.1) is a random element of $\mathbb{Z}_p$. Hence, the signatures on $\hat{\mathtt{id}}$ given by $\mathcal{B}_1$ will fail to verify with an overwhelming probability of $1 - 1/p$. The equality holds if we set $t := 1 + zc/\alpha$, instead of selecting t uniformly at random from $\mathbb{Z}_p$.[3] However, setting $t := 1 + zc/\alpha$ results in R being set to the problem instance g^α, removing t from the picture altogether. Thus, $\mathcal{B}_1.4$ would finally look like:

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

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

Observation 2 (Ambiguity due to the choice of $\hat{\mathtt{id}}$)**.** *$\mathcal{B}_1$ sets the identity involved in the $\hat{i}^{th}$ G-oracle query as the target identity $\hat{\mathtt{id}}$ (see $\mathcal{B}_1.1$). Hence, the target identity can be fixed only* after *the $\hat{i}^{th}$ 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 $\hat{i}^{th}$ G-oracle query, $\mathcal{B}_1$ has no way to decide whether to proceed to $\mathcal{B}_1.3$ or $\mathcal{B}_1.4$ (as it depends on whether $\mathtt{id} = \hat{\mathtt{id}}$ or not).*

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

One way to fix the problem noted above is to guess the "index" of the target identity *instead* of guessing the target G-index. Suppose n distinct identities are involved in the

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

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

In our alternative security argument given in §**3.3**, 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.

3.2.2.2 Reduction $\mathcal{B}_2$

It takes as argument, the description of a group $(\mathbb{G}, p, g)$ and a challenge g^α with $\alpha \xleftarrow{\mathrm{U}} \mathbb{Z}_p$ 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 the master public key $\mathtt{mpk}{:=}(\mathbb{G}, p, g, \mathrm{G}, \mathrm{H})$ and $\mathtt{msk} := (g^\alpha)$, where G, H are description of hash functions modelled as random oracles. As usual, $\mathcal{B}_2$ simulates these oracles with the help of two tables $\mathfrak{L}_{\mathrm{G}}$ and $\mathfrak{L}_{\mathrm{H}}$ containing the queried values together with the answers given to $\mathcal{A}$.

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

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

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

$$\begin{aligned}\{b_0 = \log A_0 + (\log R + c_0\alpha)d_0, b_1 = \log A_1 + (\log R + c_0\alpha)d_1,\\ b_2 = \log A_2 + (\log R + c_1\alpha)d_2, b_3 = \log A_3 + (\log R + c_1\alpha)d_3\}\end{aligned} \tag{3.2}$$

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

$$\alpha = \frac{b_2 + b_1 - b_0 - b_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)}. \tag{3.3}$$

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

[5] We use b_k instead of B_k, throughout the reduction, to maintain consistency with the protocol description (in §**3.2.1**).

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

Observation 3 (Incorrect solution of the DLP instance.)**.** *In Step $\mathcal{B}_2.4$, the reduction obtains the solution of the DLP instance by solving the four equations given in (3.2). However, on substituting the values of b_ks from (3.2) in (3.3) we get*

$$\frac{b_2 + b_1 - b_0 - b_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)} = \alpha + \log_g R \cdot \frac{d_2 + d_1 - d_0 - d_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)}, \tag{3.4}$$

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

Note that the simulator does not know the value of $\log_g R$ and hence cannot extract α from the above expression. However, it is not difficult to get the correct solution (as we show in (3.12) of §**3.3.3**). The more fundamental problem is that $\mathcal{B}_2$ fails to capture all possible adversarial strategies as we show next.

Observation 4 (Incompleteness of $\mathcal{B}_2$.)**.** *In Step $\mathcal{B}_2.4$, $\mathcal{B}_2$ invokes $\mathcal{M}_{\mathcal{W},3}$ to get four forged signatures with b_ks as given in (3.2). 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 target oracle queries $\mathsf{G}(\mathtt{id}, A, m)$ and $\mathsf{H}(R, \mathtt{id})$ during the simulation. (Here, $(\mathtt{id}, m)$ corresponds to the target identity and the message pair in the forgery while (A, R) are components of the forged signature.) Thus, (3.2) covers only one of the two possible adversarial behaviours: $\mathcal{A}$ querying the random oracles in the logical order $\mathsf{H} < \mathsf{G}$ (shown in **Figure A.3** where the first branching corresponds to the forking of the H-oracle). But one cannot rule out the complementary case of $\mathsf{G} < \mathsf{H}$: $\mathcal{A}$ querying the G-oracle before the H-oracle (see **Figure 3.2**) where the first branching corresponds to the forking of the G-oracle.[6] Let's look into the structure of the forged signatures in the case $\mathsf{G} < \mathsf{H}$. As a result of the ordering of the oracle queries, $\mathcal{W}$ returns J_0 as the index of the G-oracle query on $(\mathtt{id}, A, m)$ and I_0 as the index of the H-oracle query on $(R_0, \mathtt{id})$, at the end of round 0. As G-oracle is forked before the H-oracle, we get $d_1 = d_0$, $d_3 = d_2$ and $R_1 = R_0$, $R_3 = R_2$ in the subsequent forkings, while all the c_i, $0 \leq i \leq 3$ will be different. On the other hand, the value A returned as part of the forged signature remains the same in all the four rounds. Hence, the signatures returned by $\mathcal{M}_{\mathcal{W},3}$ will contain b_ks of the form:

$$\begin{aligned} \{b_0 = \log A + (\log R_0 + c_0\alpha)d_0, b_1 = \log A + (\log R_0 + c_1\alpha)d_0, \\ b_2 = \log A + (\log R_2 + c_2\alpha)d_2, b_3 = \log A + (\log R_2 + c_3\alpha)d_2\} \end{aligned} \tag{3.5}$$

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

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

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

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

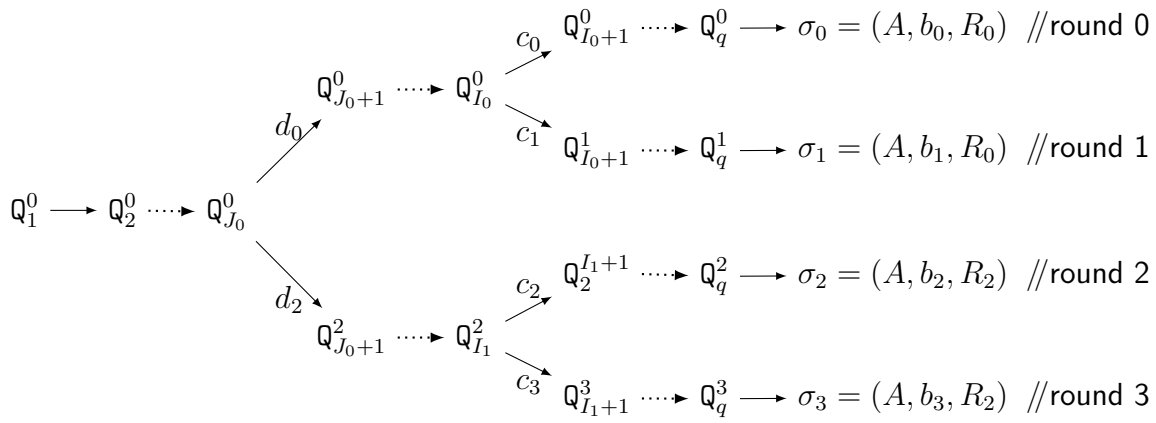

Figure 3.2: Structure of the forgeries in the case G $<$ H. $\mathtt{Q}_{J_0}^0$ denotes the target G-query $\mathrm{G}(\mathtt{id}, A, m)$; $\mathtt{Q}_{I_0}^0$ [resp. $\mathtt{Q}_{I_0}^2$] denotes the target H-query $\mathrm{H}(R_0, \mathtt{id})$ [resp. $\mathrm{H}(R_2, \mathtt{id})$].

When the signatures have the structure as in (3.5), we cannot use (3.3) (more precisely, the corrected version as given in (A.18) of **Appendix A.2.3.2**) to get a solution of the DLP. This is because $d_1 = d_0$ and $d_3 = d_2$ makes the denominator part in the corresponding expression zero. As we cannot rule out this particular adversary, the reduction does not address all the cases, rendering it incomplete.

To summarize, the same strategy to solve the DLP will *not* work for the two aforementioned complementary cases. Still it is possible to distinguish between the two cases (H $<$ G and G $<$ H) simply by looking at the structure of the forged signatures. In the case H $<$ G, all the Rs will be equal, *i.e.* $R_3 = R_2 = R_1 = R_0$; as for G $<$ H, all the As will be equal, *i.e.* $A_3 = A_2 = A_1 = A_0$. We could then use appropriate relations[7] to solve for the DLP instance. However, this results in an *unnecessary* forking (the branch consisting of round 2 and round 3 in **Figure 3.2**) being carried out in the case G $<$ H. We address this in §**3.3** 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 tighter reduction (see **Table 3.1**).

3.3 New Security Argument

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

In order to address the problem in $\mathcal{B}_1$ we redefine the event 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 GF Algorithm, whereas $\mathcal{R}_2$ and $\mathcal{R}_3$, the

[7] *i.e.*, (3.12) and (3.10) derived in **Appendix A.2.3** and **A.2.2**, respectively. For the sake of completeness, we provide the modified security argument incorporating all the above mentioned fixes in the **Appendix A.1**.

counterparts of $\mathcal{B}_2$, still use the MF algorithm. The new reductions $\mathcal{R}_1$ and $\mathcal{R}_2$ are also tighter than their counterparts in the original argument. We also use wrappers in all the three reductions.

Theorem 2. *Let $\mathcal{A}$ be an $(\epsilon, t, q_\varepsilon, q_s, q_H, q_G)$-adversary against the IBS in the EU-ID-CMA model. If the hash functions H and G are modelled as random oracles, we can construct either*

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

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

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

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

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

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

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

Argument. $\mathcal{A}$ is successful if it produces a valid forgery $\hat{\sigma} = (\hat{A}, \hat{b}, \hat{R})$ on $(\hat{\mathtt{id}}, \hat{m})$. Consider the following event[8] in the case that $\mathcal{A}$ is successful.

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

The complement of this event is

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

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

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

[8] Note that the definition of the new event E (and ¬E) is slightly different from the one given in the security argument of [GG09], *i.e.* event E (and NE) discussed in §**3.2.2**.

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

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

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

3.3.1 Reduction $\mathcal{R}_1$

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

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

Select $z \xleftarrow{\mathsf{U}} \mathbb{Z}_p^*$ as the $\mathtt{msk}$ and set $\mathtt{mpk} := (\mathbb{G}, g, p, g^z)$.
$(\mathsf{b}, \sigma_0, \sigma_1) \xleftarrow{\$} \mathcal{F}_{\mathcal{Y}}((\mathtt{mpk}, \mathtt{msk}), g^{\alpha})$
if $(\mathsf{b} = 0)$ **then return** $\perp$ //$\mathsf{abort}_{1,2}$
Parse σ_i as $(\hat{b}_i, c_i, r_i, \beta_i, d_i)$.
if $(\beta_0 = 1) \wedge (\beta_1 = 0)$ **then return** $(z(c_0 d_0 - c_1 d_1) + r_0 d_0 - (\hat{b}_0 - \hat{b}_1))/r_1 d_1$
else if $(\beta_0 = 0) \wedge (\beta_1 = 1)$ **then return** $(z(c_1 d_1 - c_0 d_0) + r_1 d_1 - (\hat{b}_1 - \hat{b}_0))/r_0 d_0$
else if $(\beta_0 = 0) \wedge (\beta_1 = 0)$ **then return** $((\hat{b}_0 - \hat{b}_1) - z(c_0 d_0 - c_1 d_1))/(r_0 d_0 - r_1 d_1)$
else return $\perp$ //$\mathsf{abort}_{1,3}$
end if

The Wrapper

The main ingredient is the so-called "partitioning strategy", first used by Coron in the security argument of FDH [Cor00]. The basic idea is to divide the identity-space $\mathbb{I}$ into two disjoint sets, $\mathbb{I}_{\varepsilon}$ and $\mathbb{I}_{s}$, depending upon the outcome of a biased coin. $\mathcal{Y}$ is equipped to respond to both extract and signature queries on identities from $\mathbb{I}_{\varepsilon}$. But it fails if the adversary

does an extract query on any identity from $\mathbb{I}_s$; it can answer only to signature queries on identities from $\mathbb{I}_s$. The problem instance is embedded in the randomiser R, depending on the outcome of the biased coin. As $\mathcal{Y}$ maintains a unique R for each identity, the structure of R decides whether that identity belongs to $\mathbb{I}_\varepsilon$ or to $\mathbb{I}_s$. The optimal size of the sets is determined on analysis. The details follow.

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

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

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

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

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

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

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

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

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

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

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

as the output. We now explain how the fresh G-oracle queries are handled. The query may be

(i) G_1, Explicit query made by either $\mathcal{A}$ or $\mathcal{Y}$: In this case $\mathcal{Y}$ returns $d := s_\ell$ as the output. $\langle \mathtt{id}, A, m, d, \ell \rangle$ is added to $\mathfrak{L}_{\mathsf{G}}$ and ℓ is incremented by one.

(ii) G_2, Implicit query by $\mathcal{Y}$ in order to answer a signature query made by $\mathcal{A}$: See steps (i) and (iii) of **Signature query** on how to program the random oracle in this situation.

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

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

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

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

Otherwise

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

(d) **Signature query,** $\mathcal{O}_s(\mathtt{id}, m)$: As in **Extract query**, $\mathcal{Y}$ checks the identity for an associated R by searching tuples $\langle R_i, \mathtt{id}_i, c_i, r_i, \beta_i \rangle$ in $\mathfrak{L}_{\mathsf{H}}$ with $(\mathtt{id}_i = \mathtt{id}) \wedge (r_i \neq \perp)$. 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 $\mathcal{Y}$ assigns the identity to either $\mathbb{I}_\varepsilon$ or $\mathbb{I}_s$ by tossing a (biased) coin β. If the outcome is 0, $\mathtt{id}$ is assigned to $\mathbb{I}_s$; else it is assigned to $\mathbb{I}_\varepsilon$. Identities assigned to $\mathbb{I}_s$ have the problem instance g^α embedded in the randomiser R. Although the private key cannot be calculated, an algebraic technique, similar to one adopted by Boneh-Boyen in [BB04a], coupled with random oracle programming enables us to give the signature. On the other hand, signature queries involving identities from $\mathbb{I}_\varepsilon$ are answered by first generating $\mathtt{usk}$ as in **Extract query** and then invoking $\mathcal{S}$. A more formal description follows.

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

(i) If $\beta_i = 0$, $\mathcal{Y}$ selects $s \xleftarrow{\mathsf{U}} \mathbb{Z}_p$ and sets $d := s_\ell$, $A := g^s (g^\alpha)^{-r_i d}$. It then adds $\langle \mathtt{id}, A, m, d, \ell \rangle$ to $\mathfrak{L}_{\mathsf{G}}$ (deferred case G_2)[9] and increments ℓ by one. The signature returned is $\sigma := (A, s + zcd, R_i)$.

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

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

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

(iii) If $\beta = 0$, $\mathcal{Y}$ selects $c, s, r \xleftarrow{\text{U}} \mathbb{Z}_p$ and sets $d := s_\ell$, $R := (g^\alpha)^r$, $A := g^s(g^\alpha)^{-rd}$. Next, it adds $\langle \mathtt{id}, (g^\alpha)^r, c, r, 0\rangle$ to $\mathfrak{L}_{\text{H}}$ (deferred case H_3), $\langle \mathtt{id}, A, m, d, \ell\rangle$ to $\mathfrak{L}_{\text{G}}$ (deferred case G_2) and increments ℓ by one.[10] The signature returned is $\sigma := (A, s + zc_id, R)$.

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

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

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

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

That concludes the description of the wrapper.

3.3.1.1 Correctness of the Discrete-Log.

In the event of successful forking, $\mathcal{R}_1$ obtains two (related) sets of side-outputs σ_0 and σ_1, where σ_i (for $i = 1, 2$) is of the form $(\hat{b}_i, c_i, r_i, \beta_i, d_i)$. It aborts in the event that $\beta_1 = \beta_0 = 1$ ($\mathsf{abort}_{1,3}$). In the rest of the cases, we claim that $\mathcal{R}_1$ ends up with a system of two congruences in two unknowns $\{\hat{a}, \alpha\}$. In the following discussion, let $\hat{a}$ denote $\log_g \hat{A}_1 = \log_g \hat{A}_0$.

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

[10] $\mathcal{Y}$ chooses different randomisers if there is a collision as explained in **Footnote 9**.

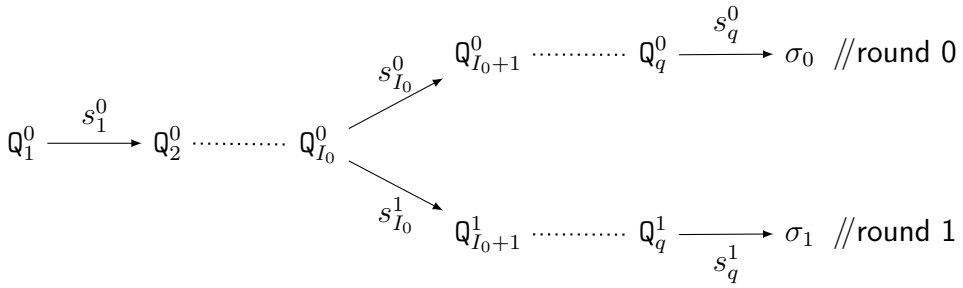

Figure 3.3: Successful forking by $\mathcal{R}_1$. $\mathtt{Q}_{I_0}^0$ denotes the target query $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$.

two unknowns $\{\hat{a}, \alpha\}$. α can be solved for as shown below.

$$\alpha = \frac{z(c_0 d_0 - c_1 d_1) + r_0 d_0 - (\hat{b}_0 - \hat{b}_1)}{r_1 d_1} \tag{3.6}$$

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

$$\alpha = \frac{z(c_1 d_1 - c_0 d_0) + r_1 d_1 - (\hat{b}_1 - \hat{b}_0)}{r_0 d_0} \tag{3.7}$$

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

$$\alpha = \frac{(\hat{b}_0 - \hat{b}_1) - z(c_0 d_0 - c_1 d_1)}{(r_0 d_0 - r_1 d_1)} \tag{3.8}$$

Notice that (3.6), (3.7) and (3.8) is precisely what $\mathcal{R}_1$ outputs in **Algorithm 4**.

Remark 13. The equations (3.6), (3.7) and (3.8) 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 $\mathsf{H}(\hat{R}_0, \hat{\mathtt{id}})$ before the query $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$ ($\mathsf{H} < \mathsf{G}$) in round 0. Hence, the order in which $\mathcal{A}$ makes the aforementioned random oracle queries is not relevant.

3.3.1.2 Analysis

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

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

simulation. Therefore $acc_{\mathcal{Y}} \geq (1-\delta)^{q_\varepsilon}\epsilon$.

The reduction is successful in the event that neither $\mathsf{abort}_{1,2}$ and $\mathsf{abort}_{1,3}$ occurs. The first of the aborts ($\mathsf{abort}_{1,2}$) pertains to the GF Algorithm: $\mathcal{R}_1$ aborts in the event that the forking ended up being a failure. On applying the GF Lemma (**Lemma 4**) with $acc = acc_{\mathcal{Y}}$, $|\mathbb{S}| = p$ and $q = q_{\mathrm{G}}$, we get the following lower bound:

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

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

$$\begin{aligned}
\epsilon_1 &= \Pr\left[\neg\mathsf{abort}_{1,3} \wedge \neg\mathsf{abort}_{1,2}\right] \\
&\geq (1-(1-\delta)^2) \cdot (1-\delta)^{q_\varepsilon}\epsilon \cdot \left(\frac{(1-\delta)^{q_\varepsilon}\epsilon}{q_{\mathrm{G}}} - \frac{1}{p}\right) \\
&= (2\delta - \delta^2) \cdot (1-\delta)^{q_\varepsilon}\epsilon \cdot \left(\frac{(1-\delta)^{q_\varepsilon}\epsilon}{q_{\mathrm{G}}} - \frac{1}{p}\right) \qquad (3.9)
\end{aligned}$$

Assuming $p \gg 1$, (3.9) attains maximum value at the point $\delta = \left(1 - \sqrt{q_\varepsilon/(q_\varepsilon+1)}\right)$, at which

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

Here, $\exp$ is the base of natural logarithm.

Remark 14. The above reduction is tighter than the original reduction $\mathcal{B}_1$ given in [GG09]. This can be attributed to two reasons: *i*) $\mathcal{R}_1$ uses the GF Algorithm $\mathcal{F}_{\mathcal{W}}$ instead of the MF Algorithm $\mathcal{M}_{\mathcal{W},1}$; and *ii*) $\mathcal{B}_1$ in [GG09] randomly chooses one of the identities involved in the G-oracle query as the target identity (refer to §**3.2.2.1**) which contributes a factor of q_{G}^2 to the degradation in $\mathcal{B}_1$. By contrast, we apply Coron's technique in $\mathcal{R}_1$ to partition the identity space in an optimal way.

Time complexity. If 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.

3.3.2 Reduction $\mathcal{R}_2$

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

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

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

The Wrapper

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

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

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

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

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

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

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

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

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

(ii) Otherwise, $\mathcal{W}$ chooses $y \xleftarrow{U} \mathbb{Z}_p$, sets $c := s_\ell$ and $R := (g^\alpha)^{-c} g^y$. It then adds $\langle R, \mathtt{id}, c, \ell, y \rangle$[11] to $\mathfrak{L}_{\text{H}}$ and increments ℓ by one (an implicit H-oracle query). Finally, it returns $\mathtt{usk} := (y, R)$ as the secret key.

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

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

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

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

Structure of the forgery. Recall that the signature queries are answered by doing an extract query on the identity followed by calling $\mathcal{S}$. Therefore, the resultant secret keys are of the form $\mathtt{usk} = (y, R)$, 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 $\neg$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 $b = a + (r + \alpha c)d$.

3.3.2.1 Correctness of the Discrete-Log.

In the event of successful forking, $\mathcal{R}_2$ obtains two (related) sets of side-outputs $\{\sigma_0, \sigma_1\}$, where σ_i (for $i = 0, 1$) is of the form $(\hat{b}_i, c_i, d_i)$. Let $\hat{a}$ denote $\log_g \hat{A}_1 = \log_g \hat{A}_0$; let $\hat{r}$ denote $\log_g \hat{R}_1 = \log_g \hat{R}_0$; and let d denote $d_1 = d_0$. Since the multiple-forking was successful, we have: $\{\hat{b}_0 = \hat{a} + (\hat{r} + \alpha c_0)d, \hat{b}_1 = \hat{a} + (\hat{r} + \alpha c_1)d\}$–a system of two congruences in the two (effective) unknowns $\{\hat{a} + \hat{r}d, \alpha\}$. α can be solved for by using the expression given below.

$$\alpha := (\hat{b}_0 - \hat{b}_1) \big/ (d(c_0 - c_1)) \tag{3.10}$$

Notice that (3.10) is precisely what $\mathcal{R}_2$ outputs in **Algorithm 5**.

3.3.2.2 Analysis

Since there is no abort involved in the simulation of the protocol, we may conclude that the accepting probability of $\mathcal{W}$ is the same as the advantage of the adversary, *i.e.* $acc_{\mathcal{W}} = \epsilon$.

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

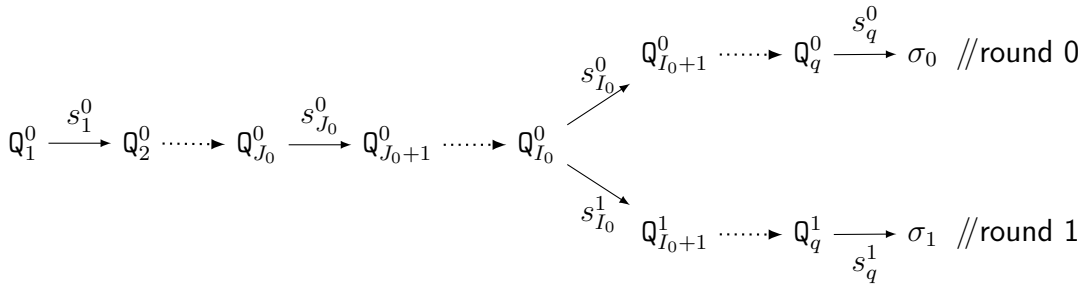

Figure 3.4: Successful forking by $\mathcal{R}_2$. $\mathtt{Q}_{J_0}^0$ denotes the target G-query $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$ while $\mathtt{Q}_{I_0}^0$ denotes the target H-query $\mathsf{H}(\hat{R}, \hat{\mathtt{id}})$.

The probability of success of the reduction $\mathcal{R}_2$ is computed by using MF Lemma (**Lemma 6**) with $q := q_{\mathsf{H}} + q_{\mathsf{G}}$, $|\mathbb{S}| := p$ and $n = 1$.[12] Hence, we get $\epsilon_2 = \mathsf{O}\left(\epsilon^2/(q_{\mathsf{H}} + q_{\mathsf{G}})^2\right)$.

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

Remark 15. $\mathcal{R}_2$ is similar in some aspects to the (incomplete) reduction $\mathcal{B}_2$ in [GG09]. However, a *major* difference is that $\mathcal{R}_2$ uses the MF Algorithm $\mathcal{M}_{\mathcal{W},1}$ instead of $\mathcal{M}_{\mathcal{W},3}$ to solve the DLP challenge. Therefore, only one forking is involved leading to a much tighter reduction than $\mathcal{B}_2$.

3.3.3 Reduction $\mathcal{R}_3$

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

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

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

3.3.3.1 Correctness of the discrete-log.

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

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

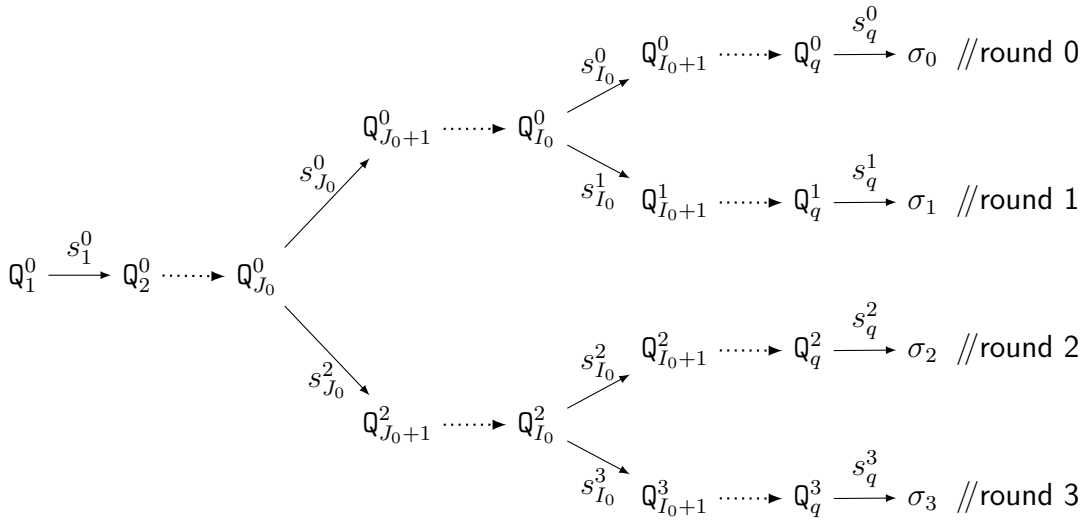

Figure 3.5: Successful multiple forkings by $\mathcal{R}_3$. $\mathtt{Q}_{J_0}^0$ denotes the target H-query $\mathsf{H}(\hat{R}, \hat{\mathtt{id}})$; $\mathtt{Q}_{I_0}^0$ [resp. $\mathtt{Q}_{I_0}^2$] denotes the target G-query $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}_0, \hat{m})$ [resp. $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}_2, \hat{m})$].

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

$$\{\hat{b}_0 = \hat{a}_0 + (\hat{r} + \alpha c_0)d_0, \hat{b}_1 = \hat{a}_0 + (\hat{r} + \alpha c_0)d_1, \hat{b}_2 = \hat{a}_2 + (\hat{r} + \alpha c_2)d_2, \hat{b}_3 = \hat{a}_2 + (\hat{r} + \alpha c_2)d_3\} \quad (3.11)$$

We claim that (3.11) forms a system of four congruences in the four unknowns $\{\hat{r}, \hat{a}_0, \hat{a}_2, \alpha\}$. α can be solved for by using the expression given below.

$$\alpha = \frac{(\hat{b}_0 - \hat{b}_1)(d_2 - d_3) - (\hat{b}_2 - \hat{b}_3)(d_0 - d_1)}{(c_0 - c_1)(d_0 - d_1)(d_2 - d_3)} \quad (3.12)$$

Notice that (3.12) is precisely what $\mathcal{R}_3$ outputs in **Algorithm 6**.

3.3.3.2 Analysis

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

3.3.4 A Comparison with the Original Reduction.

Recall that we replaced the reduction $\mathcal{B}_1$ in the original security argument with the new reduction $\mathcal{R}_1$. Likewise, $\mathcal{B}_2$ was replaced with the two reductions $\mathcal{R}_2$ and $\mathcal{R}_3$. The resulting effect on tightness is tabulated below. The security degradation involved in original $\mathcal{B}_1$ is of the order $\mathsf{O}\left(q_{\mathsf{G}}^3\right)$. In comparison, $\mathcal{R}_1$ incurs a degradation of order $\mathsf{O}\left(q_{\mathsf{G}} q_{\varepsilon}\right)$ which is much lower than that of $\mathcal{B}_1$. Note that $q_{\mathsf{G}} \gg q_{\varepsilon}$, *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 $\mathrm{O}\left((q_{\mathrm{G}} + q_{\mathrm{H}})^6\right)$ (as pointed out in **Footnote 12**). In comparison, the security degradation involved in $\mathcal{R}_2$ and $\mathcal{R}_3$ is of order $\mathrm{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^2\right)$ and $\mathrm{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^6\right)$ respectively. Thus, the effective degradation[13] is *still* of the order $\mathrm{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^6\right)$.

Original reductions [GG09]	$\mathcal{B}_1$	$\mathcal{B}_2$	
Degradation	$\mathrm{O}\left(q_{\mathrm{G}}^3\right)$	$\mathrm{O}\left((q_{\mathrm{G}} q_{\mathrm{H}})^6\right)$	
Our new reductions	$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$
Degradation	$\mathrm{O}\left(q_{\mathrm{G}} q_{\varepsilon}\right)$	$\mathrm{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^2\right)$	$\mathrm{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^6\right)$

Table 3.1: A comparison of degradation in the original [GG09] and the new security argument.

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

Chapter 4

Galindo-Garcia IBS, Improved

4.1 Introduction

Even though we had managed to fix the security argument for GG-IBS in the previous chapter, the security bound that we ended up with in **Theorem 2** is still quite loose. The loss of tightness is *inherited* from the MF Algorithm (**Algorithm 2**) which is used to launch nested replay attack on the GG-IBS adversary. To be precise, the use of $\mathcal{M}_{\mathcal{W},3}$ in reduction $\mathcal{R}_3$ leads to an effective degradation of $\mathrm{O}\left(q^6\right)$ (where q denotes $q_{\mathrm{H}} + q_{\mathrm{G}}$). In this chapter, we contemplate a better security bound for GG-IBS. To that end, we introduce two notions pertaining to simulation of random oracles: "dependency" and "independency". The notion of independency follows naturally for GG-IBS; dependency, on the other hand, has to be induced by modifying the construction (to be precise, the hash functions H and G) of the protocol in a clever manner. It turns out that the two notions can be applied in conjunction and this leads to the nested replay attack being launched far more *effectively* than specified in the MF Algorithm. As a result, the effective degradation is reduced to $\mathrm{O}\left(q^3\right)$. The non-trivial aspect is to leverage these two notions in the security argument.

In the previous chapter, we had concluded that the security bound for GG-IBS is quite loose. The loss of tightness of $\mathrm{O}\left(q^6\right)$ (where q denotes $q_{\mathrm{H}}+q_{\mathrm{G}}$) is *inherited*, primarily, from the MF Algorithm $\mathcal{M}_{\mathcal{W},3}$ which is used for launching a nested oracle replay attack on the GG-IBS adversary in reduction $\mathcal{R}_3$. In this chapter, we address the question of whether the nested oracle replay attack can be launched more effectively than specified in the MF Algorithm. To that end, we exploit the notions of "dependency" and "independency"–(in)dependency, in conjunction–among the target indices of the two random oracles H and G. The notion of independency follows naturally for GG-IBS; dependency, on the other hand, has to be induced by modifying the construction (to be precise, the hash functions H and G) of the protocol in a clever manner. The non-trivial aspect is to leverage these two notions in the security argument.

4.2 Degradation: A Closer Look

Let's return to the primary source of degradation for GG-IBS: reduction $\mathcal{R}_3$. For $i = \{0, \ldots, 3\}$, let J_i [resp. I_i] denote the target H-index [resp. G-index] for round i of simulation. (see **Figure 4.1**). The condition for the success of the forkings can be derived from

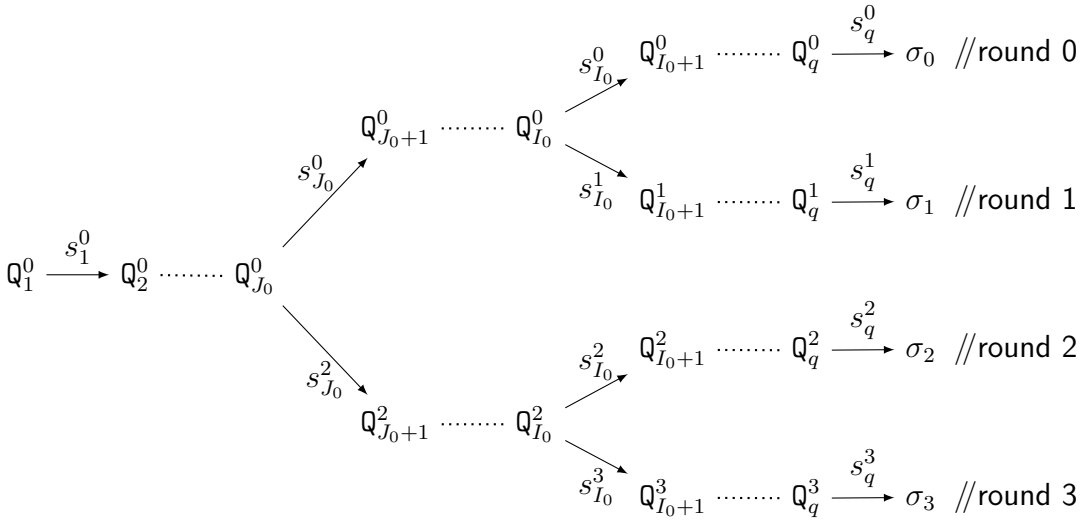

Figure 4.1: Successful Multiple-Forking for $\mathcal{R}_3$. $\mathtt{Q}_{J_0}^0$ denotes the target H-query $\mathsf{H}(\hat{R}, \hat{\mathtt{id}})$ for round 0; $\mathtt{Q}_{I_0}^0$ [resp. $\mathtt{Q}_{I_0}^2$] denotes the target G-query $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}_0, \hat{m})$ [resp. $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}_1, \hat{m})$] for round 0 [resp. round 2].

(2.23) to be $\mathsf{E} : \mathsf{B} \wedge \mathsf{C}_0 \wedge \mathsf{C}_2 \wedge \mathsf{D}_2$, with the *core* condition of

$$(I_3, J_3) = (I_2, J_2) = (I_1, J_1) = (I_0, J_0). \tag{4.1}$$

Each of the equality checks, loosely speaking, contributes to a factor of $\mathsf{O}(q^2)$ leading to the overall degradation of $\mathsf{O}(q^6)$. Our objective of bringing down the overall degradation relies on relaxing the core condition. This, in turn, is owed to the following observations.

Observation 5 (Independency of I_2 and I_0)**.** *It is not necessary for the target indices of the H oracle to be the same in round 0 and round 2 (i.e., I_2 needn't match I_0).*

In order to see this, let's return to the system of congruences in (3.11), *i.e.*,

$$\{\hat{b}_0 = \hat{a}_0 + (\hat{r} + \alpha c_0)d_0, \hat{b}_1 = \hat{a}_0 + (\hat{r} + \alpha c_0)d_1, \hat{b}_2 = \hat{a}_2 + (\hat{r} + \alpha c_2)d_2, \hat{b}_3 = \hat{a}_2 + (\hat{r} + \alpha c_2)d_3\} \tag{4.2}$$

What we have is a system of four congruences in the four unknowns $\{\hat{r}, \hat{a}_0, \hat{a}_2, \alpha\}$ with α being the solution to the DLP. From the structure of the hash function H, it follows that the index I corresponds to the unknown $\hat{a}$. The process of solving for α starts by eliminating the unknown $\hat{a}_0$ [resp. $\hat{a}_2$] from $\{\hat{b}_0, \hat{b}_1\}$ [resp. $\{\hat{b}_2, \hat{b}_3\}$]. What is *necessary* at this point is that the I indices must match for round 0 and round 1 [resp. round 2 and round 3]. However, eliminating $\hat{a}_0$ from $\{\hat{b}_0, \hat{b}_1\}$ is not affected by the pair $\{\hat{b}_2, \hat{b}_3\}$ and vice versa. Hence, from the point of view of the reduction, it doesn't make any difference whether we relax the condition

to accommodate independency–the system of congruences one ends up with is *exactly* the same as in (4.2). In fact, the reduction is unlikely to achieve anything by restricting the indices. Thus, incorporating independency simplifies the condition in (4.1) to

$$(I_3, J_3) = (I_2, J_2) \wedge (I_1, J_1) = (I_0, J_0) \wedge (J_2 = J_0). \tag{4.3}$$

Remark 16. The notion of independency can be better appreciated if we visualise the process of multiple-forking in terms of congruences and unknowns. At a high level, what the reduction algorithm $\mathcal{R}_3$ secures from the MF Algorithm is a set of four congruences in four unknowns. One of these unknowns is the solution to the DLP. The MF Algorithm needs to ensure that the congruences are linearly independent of each other with a certain non-negligible probability. **Observation 5** can then be restated as: even if the condition on the I indices is relaxed, we *still* end up with a system of four congruences in four unknowns.

Observation 6 (Dependency between H and G)**.** *It is possible to modify GG-IBS, the structure of the hash function G to be precise, such that $I_1 = I_0$ implies $J_1 = J_0$ (and similarly $I_3 = I_2$ implies $J_3 = J_2$) with a non-negligible probability.*

Recall that the hash functions in the construction of GG-IBS are of the form $\mathrm{H}(\mathtt{id}, R)$ and $\mathrm{G}(\mathtt{id}, A, m)$. Suppose that we modify the structure of G to $\mathrm{G}(\mathtt{id}, A, m, c)$ where $c := \mathrm{H}(\mathtt{id}, R)$. Let's consider round 1 of simulation for the "modified" GG-IBS initiated by a forking at I_0 as in reduction $\mathcal{R}_3$. Suppose that the adversary is successful and, in addition, the

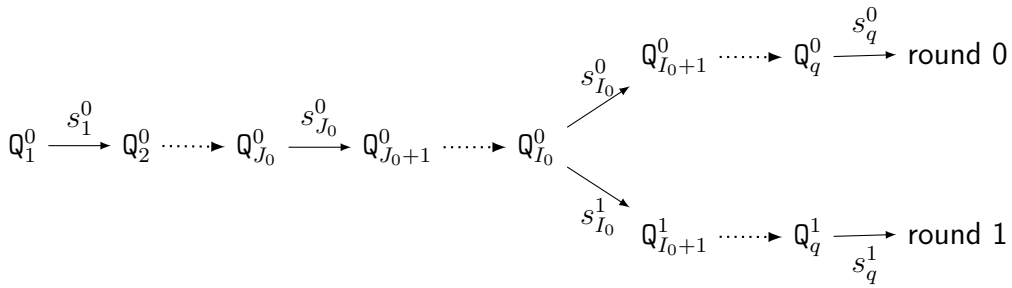

Figure 4.2: Oracle replay attack after setting up dependency between the hash functions. $\mathtt{Q}_{J_0}^0$ denotes the target H-query $\mathrm{H}(\hat{x})$ while $\mathtt{Q}_{I_0}^0$ corresponds to the target $\mathrm{G}(\hat{y}, \hat{c})$ where $\hat{c} = \mathrm{H}(\hat{x})$.

target H-index for the round 1 matches with that in round 0 (*i.e.*, $I_1 = I_0$). We claim: due to the *binding* between the hash functions that we have introduced, the target G-indices for the two rounds also *have to* match. The advantage with which an adversary can forge a signature having violated this condition is, in fact, negligible (see **Claim 2**). Hence, $(I_1 = I_0)$ *implies* $(J_1 = J_0)$. We say that the random oracle G is "dependent" on the random oracle H (denoted by $\mathrm{H} \prec \mathrm{G}$) over these two rounds[1]. The property holds for round 2 and round 3 as well (*i.e.* $(I_3 = I_2)$ implies $(J_3 = J_2)$). As a result, the condition in (4.3) is further simplified to

$$(I_3 = I_2) \wedge (I_1 = I_0) \wedge (J_2 = J_0). \tag{4.4}$$

[1]Intuitively, if a protocol uses two hash functions H_1 and H_2 in such a way that the input to H_2 is a *function* of the output of H_1, then we say that the H_2-call is *dependent* on the H_1-call. This is possible only when there is an inherent order, a *hierarchy*, among the hash functions used in the construction of the protocol.

The introduction of binding, in fact, brings more to the table. In a particular round of simulation, to forge a signature, an adversary (except with a negligible probability of guessing) *has* to make the target random oracle queries in the order $\mathrm{H} < \mathrm{G}$ (see **Claim 2**). For example, it follows in round 0 that $(1 \leq J_0 < I_0 \leq q)$. Thus, as a result of the binding, the logical order is *imposed* on the adversary. That brings us to the formal definition of the notion of random-oracle dependency.

Definition 14 (Random-Oracle Dependency)**.** Consider the oracle replay attack in the context of a cryptographic protocol that employs two hash functions H_1 and H_2 modelled as random oracles. Let J [resp. I] denote the target H_1-index [resp. H_2-index] for the first round of simulation of the protocol. Also, let J' [resp. I'] denote the target H_1-index [resp. H_2-index] for the second round of simulation that was initiated by a forking at I. Suppose that the adversary was successful in both the rounds. The random oracle H_2 is defined to be η-dependent on the random oracle H_1 on the target query (denoted by $\mathrm{H}_1 \prec \mathrm{H}_2$) if the following criteria are satisfied: *i*) $(1 \leq J < I \leq q)$ or, in other words $\mathrm{H}_1 < \mathrm{H}_2$; and *ii*) $\Pr[(J' \neq J) \mid (I' = I)] \leq \eta$.

Inducing dependency. Let's return to the random oracles in **Definition 14**. A natural way to set up the dependency $\mathrm{H}_1 \prec \mathrm{H}_2$ is by redefining the random oracle H_2 as a function

$$\mathrm{H}_2 : \mathbb{D}_2 \times \mathbb{R}_1 \mapsto \mathbb{R}_2,$$

where the second input to the oracle H_2 is chosen as the output of the H_1-oracle for an appropriate input. The trick is to choose the appropriate input. This, in turn, will require changes in the protocol, to be precise, changes in the hash functions corresponding to the random oracles.

The second criterion, in other words, requires $I' = I \implies J' = J$ with probability at least $(1 - \eta)$; for the criterion to hold with overwhelming probability, η should be negligible in κ. A little more specifically, H_2 is said to be *fully-dependent* on H_1 if $\eta = 0$, *i.e.* $(I' = I) \implies (J' = J)$. But for most applications, it suffices that J be η-dependent on I for some η which is negligible.

Although dependency forces an order among the hash functions, the reverse may not be true–a logical order among the hash functions in the protocol does not necessarily translate into dependency between them. Hence, one may need to impose explicit dependency among the hash functions. A natural way to induce the dependency $\mathrm{H}_1 \prec \mathrm{H}_2$ is by making the input to H_2 a function of H_1's output (like the binding we introduced for GG-IBS).

Next, we formally argue that the binding in GG-IBS does indeed translate into a dependency between the random oracles (see **Figure 4.3** for a detailed construction).

Claim 2 (Dependency for modified GG-IBS)**.** *The random-oracle dependency of $H \prec G$ with $\eta := q(q-1)/p$ applies to the modified GG-IBS.*

Argument. Consider an adversary $\mathcal{A}$ against the modified GG-IBS. Let's suppose that it produces a valid, non-trivial forgery $\hat{\sigma} = (\hat{b}, \hat{R}, \hat{A})$ on $(\hat{\mathtt{id}}, \hat{m})$ at the end of one round of simulation. Let J [resp. I] denote the index of the target H-query $\mathrm{H}(\hat{\mathtt{id}}, \hat{R})$ [resp. G-query $\mathrm{G}(\hat{m}, \hat{A}, c)$, where $c := \mathrm{H}(\hat{\mathtt{id}}, \hat{R})$] for the round. By using an argument *à la* that in **Footnote 3**, it can be established that $0 < I, J \leq q$. Now, there are two possibilities: $\mathrm{H} < \mathrm{G}$

and G $<$ H. Let's look at the second of the cases. Since G takes c as an input, G $<$ H is plausible only if $\mathcal{A}$ correctly *anticipates* the value of c. Thus, for $\mathcal{A}$ to have a non-negligible advantage, it has to make the two target queries *and* in the order H $<$ G. Simply put, the random oracles satisfy the first criterion.

Next, we simulate $\mathcal{A}$ again after forking at I. Let (I', J') be the target indices for this round. In order to establish that the random oracles satisfy the second criterion, let's assume the contrary: $I' = I$ *but* $J' \neq J$. Since $I' = I$, $\mathcal{A}$ has to forge a signature $\hat{\sigma}' = (\hat{b}', \hat{R}', \hat{A})$ on $(\hat{\mathtt{id}}', \hat{m})$ such that $\mathsf{H}(\hat{\mathtt{id}}', \hat{R}') = c$. This, in turn, is tantamount to a collision of the random function corresponding to H and, as per the birthday bound, the probability of this event is $q(q-1)/p$. Moreover, provided that $q \ll p$ the value is negligible[2]. By implication, the second criterion is also satisfied. □

More generally:

Claim 3 (Binding induces dependency)**.** *Consider the hash functions (and the corresponding random oracles) described in* ***Definition 2****. Let q_1 denote the upper bound on the number of queries to the random oracle H_1. In addition, let $\mathbb{R}_1$ denote the range of H_1. Binding H_2 to H_1 (by making the input to H_2 a function of H_1's output) induces a random-oracle dependency $H_1 \prec H_2$ with $\eta_b := q_1(q_1 - 1)/|\mathbb{R}_1|$.*

Argument. The line of argument is the same as in **Claim 2**. □

Remark 17. Assuming q_1 to be polynomial and $|\mathbb{R}_1|$ to be exponential in κ, the value of η_b is asymptotically negligible. Now, let's consider a concrete setting, say 80-bit security level. Assuming typical values of $q_1 := 2^{60}$ and $|\mathbb{R}_1| := 2^{80}$, the value can no longer be ignored. However, if we take into consideration the degradation in **Theorem 3**, we end up choosing $|\mathbb{R}_1| := 2^{260}$ rendering η_b negligible. Hence, we can safely assume full-dependency in subsequent discussions.

The consequences of (in)dependency. In a nutshell, our observations affect the security of GG-IBS in the following ways.

(i) We saw that, once (in)dependency is taken into consideration, the condition for success of the multiple forkings is simplified to (4.4). Intuitively, this would bring down the degradation to $\mathsf{O}(q^3)$.

(ii) The incompleteness of $\mathcal{B}_2$ stems from the fact that the simulator cannot really restrict the order in which the adversary makes the two target random oracle queries. In the previous chapter, the issue was fixed using the *two-reduction* strategy. By imposing the logical order H $<$ G through dependency, we can do away with the events F and $\neg$F and hence the reduction $\mathcal{R}_2$. This leads to a simpler event structure.

As we will see in the next section, we end up with a cleaner, tighter security argument for GG-IBS.

[2]To be precise, q is polynomial in κ whereas $1/p$ is a negligible function in κ. Plus, the product of a polynomial and a non-negligible function is still non-negligible.

4.3 Galindo-Garcia IBS, Improved

We begin by describing, in detail, the GG-IBS scheme in which the binding required to induce the dependency $\mathsf{H} \prec \mathsf{G}$ has been set up.[3] The construction is same as in **Figure 3.1** *save* for the structure of the hash function G, which is now a function of $c := \mathsf{H}(\mathtt{id}, R)$ (boxed in **Figure 4.3**). The binding that we have introduced is more refined[4] than the one suggested in **Observation 6**.

Set-up, $\mathcal{G}(1^\kappa)$: Invoke the group generator $\mathcal{G}_{\mathsf{DL}}$ (on 1^κ) to obtain $(\mathbb{G}, g, p)$. Select $z \xleftarrow{\mathsf{U}} \mathbb{Z}_p$ and set $Z = g^z$. Return z as the master secret key $\mathtt{msk}$ and $(\mathbb{G}, p, g, Z, \mathsf{H}, \mathsf{G})$ as the master public key $\mathtt{mpk}$, where H and G are hash functions

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

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

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

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

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

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

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

Figure 4.3: Galindo-Garcia IBS with binding.

4.3.1 Security Argument

Once the binding is in place, the adversary is bound to make the target queries in the logical order (except with a negligible probability, see **Claim 2**). This simplifies the event

[3] A noteworthy observation is that, setting up the random-oracle dependency $\mathsf{G} \prec \mathsf{H}$ in GG-IBS allows more efficient reductions to DLP. However, this disturbs the "logical" order of the random oracles from the protocol's perspective. In such a protocol, the PKG will have to issue private keys for each message to be signed, rendering it impractical.

[4] Recall that the suggestion in **Observation 6** was to set $d := \mathsf{G}(\mathtt{id}, A, m, c)$ where $c := \mathsf{H}(R, \mathtt{id})$. However, $\mathtt{id}$ is redundant here as it is anyways captured indirectly by c.

structure of the security argument to quite an extent and, accordingly, it consists of only two reductions $\mathcal{R}_1'$ and $\mathcal{R}_3'$.

Theorem 3. *Let $\mathcal{A}$ be an $(\epsilon, q_\varepsilon, q_H, q_G)$-adversary against GG-IBS in the EU-ID-CMA model. If the hash functions H and G are modelled as random oracles, we can construct either*

(i) Algorithm $\mathcal{R}_1'$ that $\epsilon^2/(2\exp(1)q_G q_\varepsilon)$-breaks the DLP, or

(ii) Algorithm $\mathcal{R}_3'$ that $\epsilon^4/64(q_H+q_G)^3$-breaks the DLP.

Here q_ε denotes the upper bound on the number of extract queries that $\mathcal{A}$ can make; q_H [resp. q_G] denotes the upper bound on the number of queries to the H-oracle [resp. G-oracle]. $\exp$ is the base of natural logarithm. (Plus, for simplicity, we have assumed both $1/p$ and $q(q-1)/p$ to be negligible).

Argument. Recall the events E and $\neg\mathsf{E}$ that were defined in the security argument of GG-IBS. In the case of the event E [resp. $\neg\mathsf{E}$] we give a reduction $\mathcal{R}_1'$ [resp. $\mathcal{R}_3'$] to the DLP. $\mathcal{R}_1'$ can be regarded to be a simplified version of the reduction $\mathcal{R}_1$ confined to handling the order $\mathsf{H} < \mathsf{G}$ (recall that $\mathcal{R}_1$ is equipped to handle both the orders). As the gist of the reduction $\mathcal{R}_1'$ is the same as in $\mathcal{R}_1$ given in §**3.3.3**, we relegate it to **Appendix A.3**. As for $\mathcal{R}_3'$, the simulation of one round of the adversary, as well as the manner in which it is forked is the same as plotted in $\mathcal{R}_3$. However, we use the Rewinding Technique (see §**2.3.2.2** and §**2.5**) instead of the MF Algorithm, as shown in **Figure 4.4**. The novelty lies in exploiting (in)dependency for the fruition of a better security bound. Hence, we focus on the probability analysis. □

4.3.2 Analysis

The aim is to avoid the use of Extended Splitting Lemma in the analysis. This requires two iterations of the Splitting Lemma (**Lemma 1**) on different, but correlated, underlying sets. Consequently, we have two base notions of "good": good and $\mathsf{good}_{(1)}$. In addition, we have a higher notion of $\mathsf{good}_{(3)}$ which we, ultimately, intend to bound.

Conventions. We stick to the conventions followed in the analysis of elementary oracle replay attack in §**2.3.2.3**. $\mathbb{T}$ denotes the universe of random tapes participating in a single round of simulation of the adversary. This includes the internal coins of the adversary as well as the randomness from the random functions associated to the two random oracles, the signature oracle and the extract oracle. $\mathbb{T}_{(1)}$ denotes the universe of random tapes involved in two rounds of simulation resulting from the forking of the G-oracle. Thus, $\mathbb{T}_{(1)}$ is defined as

$$\mathbb{T}_{(1)} = \bigcup_{i=2}^{q} \mathbb{T}_{(i-)} \times \mathbb{T}^2_{(i+)}. \tag{4.5}$$

Here, $\mathbb{T}_{(i-)}$ denotes the set of all random tapes involved in the simulation before the adversary makes the query $\mathtt{Q}_i^0$, whereas $\mathbb{T}_{(i+)}$, those after the query $\mathtt{Q}_i^0$.[5] Note that $\mathbb{T}_{(1)}$ consists of the partitions $(\mathbb{T}_{(2-)} \times \mathbb{T}^2_{(2+)})$, $(\mathbb{T}_{(3-)} \times \mathbb{T}^2_{(3+)})$ and $(\mathbb{T}_{(q-)} \times \mathbb{T}^2_{(q+)})$. We use the shorthand $\mathbb{T}_{(1,j<)}$ [resp. $\mathbb{T}_{(1,j>)}$] to denote the union $\cup_{i=2}^{j}(\mathbb{T}_{(i-)} \times \mathbb{T}^2_{(i+)})$ [resp. $\cup_{i=j+1}^{q}(\mathbb{T}_{(i-)} \times \mathbb{T}^2_{(i+)})$]. Thus, $\mathbb{T}_{(1)} = \mathbb{T}_{(1,j<)} \cup \mathbb{T}_{(1,j>)}$. Finally, $\mathbb{T}_{(3)}$ denotes the universe of random tapes for all four rounds of simulation[6] with the adversary forked as per the illustration in **Figure 4.4**. We follow the same subscripting convention for individual tapes as well with the round is indicated in the superscript; *e.g.*, $\mathtt{T}^0_{(i-)}$ denotes the random tape involved in the first round of simulation up till the adversary makes the query $\mathtt{Q}_i^0$ (see **Figure 4.4**).

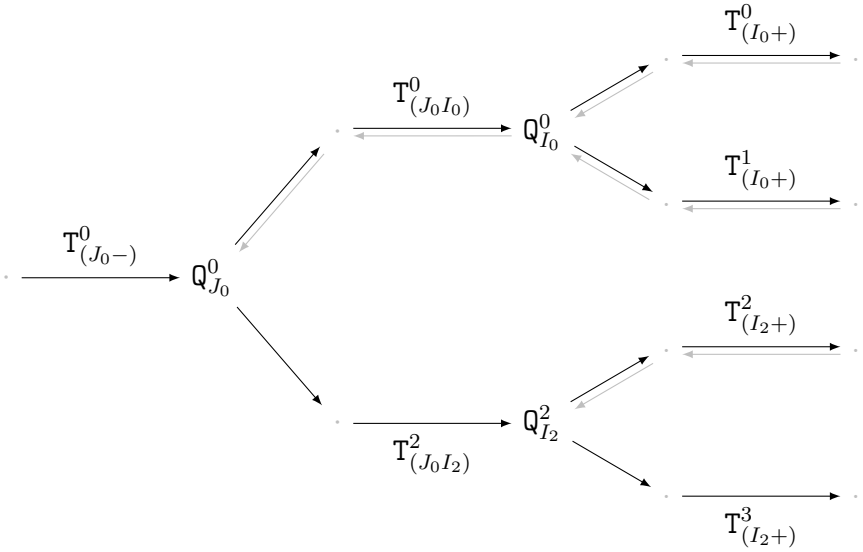

Figure 4.4: The tapes involved in the nested replay attack involving *three* forkings (carried out using the Rewinding Technique).

[5]Keep in mind that the set $\mathbb{T}_{(i-)}$ can be further represented by the Cartesian product of the two sets $\mathbb{T}_{(j-)}$ and $\mathbb{T}_{(ji)}$ as shown below.

$$\mathbb{T}_{(1)} = \bigcup_{i=2}^{q} \left(\mathbb{T}_{(j-)} \times \mathbb{T}_{(ji)} \right) \times \mathbb{T}^2_{(i+)}$$

Here $\mathbb{T}_{(j-)}$ denotes the set of all random tapes involved in the simulation before the adversary makes the query $\mathtt{Q}_j^0$, whereas $\mathbb{T}_{(ji)}$ denotes those from the query $\mathtt{Q}_j^0$ to the query $\mathtt{Q}_i^0$. In fact, this representation is in agreement with the **Figure 4.4** more than the one in (4.5).

[6]It can be worked out that the universe of tapes for three forkings is

$$\mathbb{T}_{(3)} = \bigcup_{j=1}^{q-1} \left(\mathbb{T}_{(j-)} \times \left(\bigcup_{i=j+1}^{q} \left(\mathbb{T}_{(ji)} \times \mathbb{T}^2_{(i+)} \right) \right)^2 \right).$$

.

Defining the notions of "good". The notion of good is defined akin to the notion of "good" for the Schnorr signature scheme in §**2.3.2.3**: a tape in set $\mathbb{T}$ is deemed to be good if it leads to the adversary successfully forging. Thus, by definition, at least ϵ fraction of the tapes in $\mathbb{T}$ are good. However, the refined notion for good is a bit different as there are *two* random oracles in consideration. A tape in set $\mathbb{T}$ is deemed to be $\mathsf{good}_{(j,i)}$ if it leads to the adversary forging successfully and with a target H-index of j and a target G-index of i. In addition, $\mathsf{good}_{(j,*)}$ denotes a tape which leads to the adversary forging successfully with a target H-index i and *any* target G-index. In a analogous manner, we define the notion of $\mathsf{good}_{(*,i)}$. The key to an improved analysis requires circumventing the notion of $\mathsf{good}_{(j,i)}$ through the notion of $\mathsf{good}_{(1)}$ and, its refinements, $\mathsf{good}_{(1,j,*)}$ and $\mathsf{good}_{(1,*,i)}$. This, in turn, is made possible by our observations in §**4.2**–*i.e.*, through (in)dependency.

The notion of $\mathsf{good}_{(1)}$ is also related to the "higher" notion of $\mathsf{good}_{(1)}$ that was defined for Schnorr Signature (in **Remark 6**). Recall that, for Schnorr Signature, the notion of $\mathsf{good}_{(1)}$ applies to tapes participating in two rounds of simulations associated with (the only) forking. However, in the case of GG-IBS, there are two random oracles in consideration. Thus, the notion of $\mathsf{good}_{(1)}$ could be applied to tapes participating in replay of either of these oracles. But, as we are concerned with the replay of the G-oracle in the first two rounds of the nested replay attack (see **Figure 4.4**), the underlying set for which the notion of $\mathsf{good}_{(1)}$ applies is defined to be $\mathbb{T}_{(1)}$. A triplet of tape segments $(\mathtt{T}_{(I-)}, (\mathtt{T}_{(I+)}, \mathtt{T}'_{(I+)})) \in \mathbb{T}_{(1)}$ is considered to be $\mathsf{good}_{(1)}$ if the tapes $\mathtt{T} = (\mathtt{T}_{(I-)}, \mathtt{T}_{(I+)})$ and $\mathtt{T}' = (\mathtt{T}_{(I-)}, \mathtt{T}'_{(I+)})$ are both $\mathsf{good}_{(*,I)}$. To paraphrase, the tape is $\mathsf{good}_{(1)}$ if the replay of the G-oracle using tapes $\mathtt{T}$ and $\mathtt{T}'$ is successful. We define the refined notions of $\mathsf{good}_{(1,j,i)}$, $\mathsf{good}_{(1,*,i)}$ and $\mathsf{good}_{(1,j,*)}$ the same way we had defined it for the notion of good. Finally, a tape in $\mathbb{T}_{(3)}$ is $\mathsf{good}_{(3)}$ if the nested replay attack launched as in **Figure 4.4** using the tape is successful. Keeping these notions in mind, we proceed to the probability analysis of the nested replay attack.

Probability Analysis. Our objective is to lower bound the fraction of $\mathsf{good}_{(3)}$ tapes in $\mathbb{T}_{(3)}$ in terms of the fraction of good tapes in $\mathbb{T}$. This is accomplished through three claims: **Claim 4** through **Claim 6**, which, in turn, requires two iterations of the Splitting Lemma (**Lemma 1**). The objective of the first iteration of the Splitting Lemma (in **Claim 4**) is to establish a lower bound on the fraction of $\mathsf{good}_{(1)}$ tapes in $\mathbb{T}_{(1)}$, in terms of the fraction of good tapes in $\mathbb{T}$. This bound is, in turn, to be used in the second iteration (in **Claim 6**) to lower bound the fraction of $\mathsf{good}_{(3)}$ tapes in $\mathbb{T}_{(3)}$. However, the second iteration is far more involved than the first iteration–we need an intermediate **Claim 5**. Nevertheless, the usage of the Splitting Lemma, on a high level, is (self) similar to that in the first iteration; it's the underlying set, the associated probabilities and the notion of "good" that differs. Let's focus on the first two rounds of simulation of the adversary (see **Figure 4.4**).

Claim 4. *At least $\epsilon^2/4q$ fraction of the tapes in $\mathbb{T}_{(1)}$ are good$_{(1)}$.*

Argument. The analysis, because of dependency, turns out to be similar to analysing the two rounds of the elementary replay attack *bar* the definition of the notion of "good"–we use the notion of $\mathsf{good}_{(*,i)}$ in place of $\mathsf{good}_{(i)}$. Let's presume that ϵ_i fraction of the tapes in $\mathbb{T}$ are

$\mathsf{good}_{(*,i)}$. Thus, we have

$$\begin{aligned}\sum_{i=1}^{q}\epsilon_i &= \sum_{i=2}^{q}\Pr\left[I=i\wedge J>0\right]\\ &= \Pr\left[1\leq J<I\leq q\right]=\epsilon \ \text{ (by definition)}.\end{aligned} \tag{4.6}$$

The notion of $\mathsf{good}_{(*,i)}$, which we defined for the set $\mathbb{T}$, is adapted for the set $\mathbb{T}_{(i-)}\times\mathbb{T}_{(i+)}$ through the split and join functions (see §**2.3.2.3**). A tape $(\mathtt{T}_{(i-)},\mathtt{T}_{(i+)})\in\mathbb{T}_{(i-)}\times\mathbb{T}_{(i+)}$ is deemed to be $\mathsf{good}_{(*,i)}$ if its counterpart in $\mathbb{T}$ is $\mathsf{good}_{(*,i)}$. We denote the (sub)set of all such tapes in $\mathbb{T}_{(i-)}\times\mathbb{T}_{(i+)}$ by $\mathbb{V}_i$. Also, suppose that at least β_i fraction of tapes in $\mathbb{T}_{(i-)}\times\mathbb{T}_{(i+)}$ are $\mathsf{better}_{(*,i)}$. On these underlying assumptions, the analysis proceeds in a manner (almost) similar to that of the elementary replay attack (using the Rewinding Technique) in §**2.3.2.3**. Thus, we may conclude that the first two rounds of the nested replay attack are successful with a probability of at least $\epsilon^2/4q$. Moreover, it follows from the definition (of $\mathsf{good}_{(1)}$) that at least $\epsilon^2/4q$ fraction of tape segments in $\mathbb{T}_{(1)}$ are $\mathsf{good}_{(1)}$. □

The second iteration of the Splitting Lemma is a bit trickier than the first one. We have to prime the set $\mathbb{T}_{(1)}$ before actually applying the Splitting Lemma. The partitions of the set $\mathbb{T}_{(1)}$ come into play. Let's focus on a particular partition $(\mathbb{T}_{(i-)}\times\mathbb{T}^2_{(i+)})$. It is not difficult to infer that the set of $\mathsf{good}_{(1,*,i)}$ tapes in $\mathbb{T}_{(1)}$, in fact, all belong[7] to the partition $(\mathbb{T}_{(i-)}\times\mathbb{T}^2_{(i+)})$. These $\mathsf{good}_{(1,*,i)}$ elements can be further partitioned, depending on their target H-index, into subsets containing the $\mathsf{good}_{(1,j,i)}$ elements. The set of all such $\mathsf{good}_{(1,j,i)}$ elements, across the partitions (of $\mathbb{T}_{(1)}$), form the $\mathsf{good}_{(1,j,*)}$ elements of $\mathbb{T}_{(1)}$. Let δ_j denote the fraction of all such tapes in $\mathbb{T}_{(1)}$. By implication,

$$\begin{aligned}\sum_{j=1}^{q-1}\delta_j &= \sum_{j=1}^{q-1}\Pr\left[J_1=J_0=j\wedge I_1=I_0\wedge 1<I_0\leq q\right]\\ &= \Pr\left[J_1=J_0\wedge I_1=I_0\wedge 1\leq J_0<I_0\leq q\right]\\ &\geq \epsilon^2/4q \ \text{ (using } \textbf{Claim 4}\text{)}\end{aligned} \tag{4.7}$$

That brings us to our second claim.

Claim 5. *At least δ_j fraction of tapes in $\mathbb{T}_{(1,j>)}$ are good$_{(1,j,*)}$.*

Argument. Let's presume that the tape involved in the first two rounds of simulation (the replay of G-oracle) was $\mathsf{good}_{(1,j,i)}$. Our argument relies on two observations: *i*) the candidate tapes for the replay of the H-oracle belong to $\mathbb{T}_{(1,j>)}$; and *ii*) a $\mathsf{good}_{(1,j,*)}$ element *cannot* belong to $\mathbb{T}_{(1,j<)}$. The first of the observations follows by definition, whereas the second–a consequence of dependency between the two random oracles–requires some explanation. Let's assume the contrary: a tape is $\mathsf{good}_{(1,j,*)}$ and it belongs to $\mathbb{T}_{(1,j<)}$. Consequently, it is $\mathsf{good}_{(1,j,i)}$ for some $i<j$. However, the dependency H $\prec$ G means that an adversary *has to* make the target queries in the order H $<$ G which, in turn, implies $j<i$ (leading to a contradiction). Thus, the $\mathsf{good}_{(1,j,*)}$ elements all belong to $\mathbb{T}_{(1,j>)}$.

[7] Strictly speaking, we have to take into consideration the non-core conditions.

The two observations, in conjunction with the fact that $\mathbb{T}_{(1,j>)}$ is a subset of the set $\mathbb{T}_{(1)}$ (of which at least δ_j fraction are $\mathsf{good}_{(1,j,*)}$ elements) completes the proof. □

Claim 6. *The fraction of $\mathsf{good}_{(3)}$ tapes in $\mathbb{T}_{(3)}$ is at least $\epsilon^4/64q^3$.*

Argument. Let's assume that the tape involved in the first two rounds of simulation $\mathtt{T}^0_{(1)} := (\mathtt{T}^0_{(J_0-)}, (\mathtt{T}^0_{(I_0+)}, \mathtt{T}^1_{(I_0+)}))$ is $\mathsf{good}_{(1,J_0,I_0)}$. We further split the tape segment $\mathtt{T}^0_{(I_0-)}$ into two: $\mathtt{T}^0_{(J_0-)}$ and $\mathtt{T}^0_{(J_0I_0)}$. Here, $\mathtt{T}^0_{(J_0-)}$ is the tape involved in the simulation before the adversary makes the target query $\mathtt{Q}^0_{J_0}$, whereas $\mathtt{T}^0_{(J_0I_0)}$, that from the query $\mathtt{Q}^0_{J_0}$ up to the query $\mathtt{Q}^0_{I_0}$ (see **Figure 4.4**). Thus, an alternative representation of the tape $\mathtt{T}^0_{(1)}$ is

$$\begin{aligned}\mathtt{T}^0_{(1)} &:= \left(\mathtt{T}^0_{(J_0-)}, \mathtt{T}^0_{(J_0I_0)}, \left(\mathtt{T}^0_{(I_0+)}, \mathtt{T}^1_{(I_0+)}\right)\right)\\ &:= \left(\mathtt{T}^0_{(J_0-)}, \mathtt{T}^0_{(1,J_0I_0+)}\right) \quad \text{(using shorthand notation)}\end{aligned}$$

The simulator, next, proceeds with the remaining two rounds of simulations by forking at $\mathtt{Q}^0_{J_0}$ and $\mathtt{Q}^2_{I_2}$ (in that order). Let $\mathtt{T}^2_{(1,J_0I_2+)} := (\mathtt{T}^2_{(J_0I_2)}, (\mathtt{T}^2_{(I_2+)}, \mathtt{T}^3_{(I_2+)})) \in \mathbb{T}_{(1,J_0I_2+)}$ denote the tapes involved in the aforementioned simulation. Thus, we end up with the tapes given in **Figure 4.4**. The nested replay attack, all four rounds of it, is successful if the tape

$$\mathtt{T}^2_{(1)} := \left(\mathtt{T}^0_{(J_0-)}, \left(\mathtt{T}^2_{(J_0I_2)}, \left(\mathtt{T}^2_{(I_2+)}, \mathtt{T}^3_{(I_2+)}\right)\right)\right)$$

turns out to be $\mathsf{good}_{(1,J_0,I_2)}$ (note that I_2 need not be the same as I_0). Our objective is to find the probability of this particular event using the Splitting Lemma. This requires $\mathbb{T}_{(1)}$ to be split into $\mathbb{T}_{(J_0-)}$ and $\cup^q_{i=J_0}(\mathbb{T}_{(J_0i)} \times \mathbb{T}^2_{(i+)})$. This, in turn, can be achieved by splitting the individual partitions of $\mathbb{T}_{(1)}$. But note that *not all* of the partitions in $\mathbb{T}_{(1)}$ can be split in such a way. In fact, by definition, it is viable only to the partitions $(\mathbb{T}_{(i-)} \times \mathbb{T}^2_{(i+)})$ for $i > J_0$, *i.e.* the set $\mathbb{T}_{(1,J_0>)}$. At this point **Claim 5** comes into play. It allows us to apply the Splitting Lemma to the "reduced" universe of $\mathbb{T}_{(1,J_0>)}$ (by ensuring that at least δ_{J_0} fraction of the tapes in $\mathbb{T}_{(1,J_0>)}$ *too* are $\mathsf{good}_{(1,J_0,*)}$). We are now ready to apply the Splitting Lemma.

Let's assume that an arbitrary λ_j fraction of the tapes in $\mathbb{T}_{(1)}$ (and hence $\mathbb{T}_{(1,J_0>)}$) are $\mathsf{better}_{(1,j,*)}$. We denote the (sub)set of $\mathsf{good}_{(1,J_0,*)}$ elements and the $\mathsf{better}_{(1,J_0,*)}$ elements in $\mathbb{T}_{(1,J_0>)}$ by $\mathbb{U}_{J_0}$ and $\mathbb{U}^*_{J_0}$ respectively. For ease of analysis, we use the sufficient condition that the tape $\mathtt{T}^0_{(1)}$ be $\mathsf{good}_{(1,J_0,*)}$ and the $\mathtt{T}^2_{(1)}$ be $\mathsf{better}_{(1,J_0,*)}$. Let's denote the probability of this event (with the tapes sampled as described above) by δ'_{J_0}, *i.e.*,

$$\begin{aligned}\delta'_{J_0} &= \Pr\left[\left(\mathtt{T}^0_{(J_0-)}, \mathtt{T}^0_{(1,J_0I_0+)}\right) \in \mathbb{U}^*_{J_0} \wedge \left(\mathtt{T}^0_{(J_0-)}, \mathtt{T}^2_{(1,J_0I_2+)}\right) \in \mathbb{U}_{J_0}\right]\\ &= \Pr\left[\left(\mathtt{T}^0_{(J_0-)}, \mathtt{T}^2_{(1,J_0I_2+)}\right) \in \mathbb{U}_{J_0} \mid \left(\mathtt{T}^0_{(J_0-)}, \mathtt{T}^0_{(1,J_0I_0+)}\right) \in \mathbb{U}^*_{J_0}\right] \cdot \Pr\left[\left(\mathtt{T}^0_{(J_0-)}, \mathtt{T}^0_{(1,J_0I_0+)}\right) \in \mathbb{U}^*_{J_0}\right]\\ &= (\delta_{J_0} - \lambda_{J_0})\lambda_{J_0} \quad \text{(using propositions 1 and 2 of \textbf{Lemma 1})} \qquad (4.8)\end{aligned}$$

The above expression attains a maxima of $\delta_{J_0}^2/4$ at the point $\lambda_{J_0} = \delta_{J_0}/2$. Now, the probability that the nested oracle replay attack is successful for any J_0 is given by

$$\begin{aligned}\delta' = \sum_{J_0=1}^{q-1} \delta'_{J_0} &\geq \sum_{J_0=1}^{q-1} \frac{\delta_{J_0}^2}{4} \\ &\geq \frac{\epsilon^4}{64q^3} \quad \text{(using Hölder's inequality and (4.7))} \end{aligned} \tag{4.9}$$

□

Significance of (in)dependency. Dependency plays a crucial role in the first two claims. It is the first criterion of **Definition 14** which allows us to apply the Splitting Lemma to a reduced universe of tapes in **Claim 5**. The second criterion of dependency comes into play in **Claim 4**. It ensures: if, indeed, the two tapes $(\mathtt{T}_{(I-)}, \mathtt{T}_{(I+)})$ and $(\mathtt{T}_{(I-)}, \mathtt{T}'_{(I+)})$ are both $\mathsf{good}_{(1,*,I)}$, then they have the same target H-index as well. In fact, this is the *sole* reason why the analysis proceeds as in §**2.3.2.3**. These two handles, together, reduce the degradation by $\mathsf{O}(q^2)$. Independency, on the other hand, is used in **Claim 6**. Recall that a $\mathsf{good}_{(1,*,i)}$ can only belong to the partition $(\mathbb{T}_{(i-)} \times \mathbb{T}^2_{(i+)})$. Without independency, the set of candidate tapes (for replay of the H-oracle at a particular index j) is restricted to $(\mathbb{T}_{(i-)} \times \mathbb{T}^2_{(i+)})$. But, relaxing this condition (*i.e.*, incorporating independency) "expands" the set of candidate tapes to $\mathbb{T}_{(1,j>)}$. On a high level, this reduces the degradation by $\mathsf{O}(q)$. Apply the two observations together, the degradation is reduced by $\mathsf{O}(q^3)$.

The characteristic expression. An interesting observation is that the second iteration of the Splitting Lemma, on a high level, seems to be similar to the first one. This is also captured by the fact that the analysis is abstracted out by the characteristic expressions[8]

$$\delta' \geq \sum_{j=1}^{q-1} \delta_j^2 \;\text{ and }\; \epsilon' \geq \sum_{i=2}^{q} \epsilon_i^2$$

under the set of constraints

$$\sum_{j=1}^{q-1} \delta_j = \epsilon' \;,\; \sum_{i=2}^{q} \epsilon_i = \epsilon \;\text{ and }\; (0 \leq \epsilon_i, \delta_j \leq 1)_{i,j \in \{1,\ldots,q\}}.$$

Clearly, there is a degree of *self-similarity* in the two expressions and, in addition, the expressions themselves are quite disparate from that of the nested replay attack (and the MF algorithm for $n = 3$) in (2.21).

[8] In fact, it was the characteristic expression that guided us to the analysis (and not the other way around).

4.3.3 Taking Stock

It is evident that the modified GG-IBS allows far tighter reductions. The effective degradation is reduced from $\mathsf{O}\left((q_{\mathrm{H}}+q_{\mathrm{G}})^{6}\right)$ to $\mathsf{O}\left((q_{\mathrm{H}}+q_{\mathrm{G}})^{3}\right)$ (see **Table 4.1**). The "gain", which is substantial, is by the virtue of the multiple forkings being replaced, on a high level, by (two) general forkings. This, in turn, is endowed by our observations in §**4.2**.

Scheme	Security Argument			
GG-IBS (**Chapter 3**)	Reduction	$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$
	Forking	$\mathcal{F}_{\mathcal{Y}}$	$\mathcal{M}_{\mathcal{W},1}$	$\mathcal{M}_{\mathcal{W},3}$
	Degradation	$\mathsf{O}\left(q_{\mathrm{G}}q_{\varepsilon}\right)$	$\mathsf{O}\left((q_{\mathrm{H}}+q_{\mathrm{G}})^{2}\right)$	$\mathsf{O}\left((q_{\mathrm{H}}+q_{\mathrm{G}})^{6}\right)$
Modified GG-IBS (**Figure 4.3**)	Reduction	$\mathcal{R}'_1$	$\mathcal{R}'_3$	
	Forking	$\mathcal{F}_{\mathcal{Y}}$	Nested rewinding	
	Degradation	$\mathsf{O}\left(q_{\mathrm{G}}q_{\varepsilon}\right)$	$\mathsf{O}\left((q_{\mathrm{H}}+q_{\mathrm{G}})^{3}\right)$	

Table 4.1: Degradation for the original and modified GG-IBS. q_{G} [resp. q_{H}] denotes the upper bound on the H-oracle [resp. G-oracle] queries, whereas, q_{ε} denotes upper bound on the extract queries.

Chapter 5

From sID IBS to ID IBS without Random Oracles

5.1 Introduction

The selective-identity (sID) model for identity-based cryptographic schemes was introduced in [CHK03]. The distinguishing feature of this model is that the adversary has to commit, beforehand, to a "target" identity–*i.e.*, the identity which it eventually forges on (see **Definition 10**). Since its induction, the relationship of the sID notion with various other notions of security has been extensively studied [CS06, CFH+09, Gal06, GH05]. One of the interesting results is the separation between the sID models and ID models for IBE in the standard model [Gal06]. Therefore, it is a general consensus that the sID model is much weaker than the full-identity (ID) model. However, it is *easier* to design efficient schemes, based on weaker assumptions, that are secure in the sID model compared to the ID model. This is, in particular, highlighted by the disparity in the construction of IBE schemes given in [BB04a] and [Wat05]. The former is simple and efficient, whereas the latter, involved. Therefore, a generic transformation from an sID scheme to ID scheme would be a problem worth pursuing. We could design efficient sID-secure schemes and then just bootstrap it to ID-security using the transformation[1]. In fact, this is a long-standing open problem.

The primary focus of this chapter is on the question of constructing an ID-secure IBS, given an sID-secure IBS, in the standard model, and with reasonable security degradation (say polynomial). We accomplish this through a generic transformation which uses a CHF and a *weakly*-secure PKS as black-box [CK13b]. We go one step further by applying the same construction technique to a *relaxed* notion of IBS security which we call the *weak* selective-identity (wID) model. The distinguishing feature of the wID model is that the adversary, apart from committing to the target identity, has to commit to a set of "query" identities–the set of identities which it wishes to query the signature and extract oracle with (see §**5.4** for the definition of the security model). Thus, we reduce the problem of constructing an ID-secure IBS to that of constructing wID-secure IBS, a EU-GCMA-secure PKS and a CHF. Our approach can be considered to be an alternative paradigm to the aforementioned folklore

[1] A well-known example of such an approach is the design of CCA-secure encryption schemes from CPA-secure ones using the Fujisaki-Okamato transformation [FO99], as demonstrated in [BF01].

construction of IBS.

The security argument constitutes the main hurdle–the construction itself is quite straightforward. The line of argument, roughly, is: given an adversary that breaks the ID-IBS, we construct algorithms to break either the sID/wID-IBS, the PKS or the CHF. It leads to a tightness gap of $\mathsf{O}(q_s)$, where q_s is the upper bound on the number of signature queries that the adversary is allowed to make.

5.2 Chameleon Hash Function

A chameleon hash function (CHF) is a randomised trapdoor hash function. Apart from the *collision resistance* property, it has an additional "chameleon" property which enables anyone with the trapdoor information to efficiently generate collisions.

Definition 15 (Chameleon Hash Function [BCC88, KR00, Moh11])**.** A family of CHF $\mathfrak{H}$ consists of three PPT algorithms $\{\mathcal{G}, \mathsf{h}, \mathsf{h}^{-1}\}$ described below.

> Key Generation, $\mathcal{G}(1^\kappa)$: It takes as input the security parameter κ (in unary). It outputs the evaluation key $\mathtt{ek}$ and the trapdoor key $\mathtt{td}$.
>
> Hash Evaluation, $\mathsf{h}(\mathtt{ek}, m, r)$: It takes as input the evaluation key $\mathtt{ek}$, a message m from the message-space $\mathbb{M}$ and a randomiser r from the domain $\mathbb{R}$. It outputs the hash value y from the range $\mathbb{Y}$.
>
> Collision Generation, $\mathsf{h}^{-1}(\mathtt{td}, m, r, m')$: It takes as input the trapdoor key $\mathtt{td}$, two messages $m, m' \in \mathbb{M}$ and $r \in \mathbb{R}$. It outputs $r' \in \mathbb{R}$ such that $\mathsf{h}(\mathtt{ek}, m, r) = \mathsf{h}(\mathtt{ek}, m', r')$; in other words, (m, r) and (m', r') is a collision.

Any CHF should satisfy the following two properties.

(i) *Uniformity.* The distribution induced by $\mathsf{h}(\mathtt{ek}, m, r)$ for all messages m and a randomly chosen r should be the same. In other words, the distributions $(\mathtt{ek}, \mathsf{h}(\mathtt{ek}, m, r))$ and $(\mathtt{ek}, y)$ should be computationally indistinguishable, where $(\mathtt{ek}, \mathtt{td}) \xleftarrow{\$} \mathcal{G}(1^\kappa), r \xleftarrow{\mathsf{U}} \mathbb{R}$ and $y \xleftarrow{\mathsf{U}} \mathbb{Y}$.

(ii) *Collision Resistance.* Given the evaluation key $\mathtt{ek}$, it should be hard to compute a pair $(m, r) \neq (m', r')$ such that $\mathsf{h}(\mathtt{ek}, m, r) = \mathsf{h}(\mathtt{ek}, m', r')$, *i.e.* the probability given below should be negligible for all PPT adversaries $\mathcal{A}$.

$$\Pr\left[\mathsf{h}(\mathtt{ek}, m, r) = \mathsf{h}(\mathtt{ek}, m', r') \wedge (m, r) \neq (m', r') : (\mathtt{ek}, \mathtt{td}) \xleftarrow{\$} \mathcal{G}(1^\kappa); (m, r, m', r') \xleftarrow{\$} \mathcal{A}(\mathtt{ek})\right]$$

5.3 The Generic Transformation

The transformation takes as input: *i*) an sID-secure IBS $\mathfrak{I}_s := \{\mathcal{G}_s, \mathcal{E}_s, \mathcal{S}_s, \mathcal{V}_s\}$; *ii*) an EU-GCMA-secure PKS $\mathfrak{P} := \{\mathcal{K}, \mathcal{S}_p, \mathcal{V}_p\}$; and *iii*) a CHF $\mathfrak{H} := \{\mathcal{G}_h, \mathsf{h}, \mathsf{h}^{-1}\}$, to output an ID-secure IBS $\mathfrak{I} := \{\mathcal{G}, \mathcal{E}, \mathcal{S}, \mathcal{V}\}$. The basic idea is to *map* an identity $\mathtt{id}$ in $\mathfrak{I}$ to an identity $\mathtt{id}_s$ in $\mathfrak{I}_s$ using the CHF. These two identities are then *bound* by using the PKS. A formal description follows.

Assumptions. We denote the identity-space of $\mathfrak{I}_s$ (and that of resulting $\mathfrak{I}$) by $\mathbb{I}$ and its message-space by $\mathbb{M}$. For simplicity, we assume that *i*) the message-space of $\mathfrak{P}$ *ii*) the message-space of $\mathfrak{H}$ (denoted by $\mathbb{M}_h$) and *iii*) the range of $\mathfrak{H}$ (denoted by $\mathbb{Y}$) *are all* the same set $\mathbb{I}$, *i.e.*, $\mathbb{M}_h = \mathbb{Y} = \mathbb{I}$.[2] In addition, the randomness space of $\mathfrak{H}$ is denoted by $\mathbb{R}$. Therefore, for a particular evaluation key $\mathtt{ek}$, the hash evaluation algorithm can be considered as a function $\mathsf{h} : \mathbb{I} \times \mathbb{R} \mapsto \mathbb{I}$. The description of the transformation is given in **Figure 5.1** [CK13b]. It is followed by the argument that the resultant IBS $\mathfrak{I}$ is secure in the ID model.

$\mathfrak{I} \leftarrow \mathfrak{T}(\mathfrak{I}_s, \mathfrak{P}, \mathfrak{H})$

Set-up, $\mathcal{G}(1^\kappa)$: Invoke the algorithms $\mathcal{G}_s$, $\mathcal{K}$ and $\mathcal{G}_h$ (all) on 1^κ to obtain $(\mathtt{msk}_s, \mathtt{mpk}_s)$, $(\mathtt{sk}, \mathtt{pk})$ and $(\mathtt{ek}, \mathtt{td})$ respectively. Return $\mathtt{msk} := (\mathtt{msk}_s, \mathtt{sk})$ as the master secret key and $\mathtt{mpk} := (\mathtt{mpk}_s, \mathtt{pk}, \mathtt{ek})$ as the master public key.

Key Extraction, $\mathcal{E}(\mathtt{id}, \mathtt{msk})$: Select $r \xleftarrow{\mathsf{U}} \mathbb{R}$ and compute $\mathtt{id}_s \leftarrow \mathsf{h}(\mathtt{ek}, \mathtt{id}, r)$. Next, run $\mathcal{E}_s(\mathtt{id}_s, \mathtt{msk}_s)$ and $\mathcal{S}_p(\mathtt{id}_s, \mathtt{sk})$ to obtain $\mathtt{usk}_s$ and σ_p respectively. Finally, return $\mathtt{usk} := (\mathtt{usk}_s, r, \sigma_p)$ as the user secret key.

Signing, $\mathcal{S}(\mathtt{id}, m, \mathtt{usk})$: Parse the user secret key $\mathtt{usk}$ as $(\mathtt{usk}_s, r, \sigma_p)$ and compute $\mathtt{id}_s \leftarrow \mathsf{h}(\mathtt{ek}, \mathtt{id}, r)$. Next, run $\mathcal{S}_s(\mathtt{id}_s, m, \mathtt{usk}_s)$ to obtain σ_s. Finally, return $\sigma := (\sigma_s, r, \sigma_p)$ as the signature.

Verification, $\mathcal{V}(\sigma, \mathtt{id}, m, \mathtt{mpk})$: Parse σ as (σ_s, r, σ_p) and compute $\mathtt{id}_s \leftarrow \mathsf{h}(\mathtt{ek}, \mathtt{id}, r)$. Return 1 only if σ_p is a valid signature on $\mathtt{id}_s$ and σ_s is a valid signature on $(\mathtt{id}_s, m)$. In other words, if $\mathsf{b}_p \leftarrow \mathcal{V}_p(\sigma_p, \mathtt{id}_s, \mathtt{pk})$ and $\mathsf{b}_s \leftarrow \mathcal{V}_s(\sigma_s, \mathtt{id}_s, m, \mathtt{mpk})$, return $(\mathsf{b}_p \wedge \mathsf{b}_s)$.

Figure 5.1: Constructing ID-secure IBS from an sID-secure IBS.

The Hash Evaluation function h is used to maintain a map and identity $\mathtt{id}$ in $\mathfrak{I}$ on to and identity $\mathtt{id}_s$ in $\mathfrak{I}_s$. The mapped identities are then bound using σ_p. This is reflected in the structure of the user secret key for $\mathtt{id}$ which is of the form $(\mathtt{usk}_s, r, \sigma_p)$.

Remark 18. Note that we have omitted $\mathtt{td}$ from the master secret key. The trapdoor key $\mathtt{td}$–hence the collision generation function h^{-1}–is not used *per se* in the transformation. However, it *does* play a crucial role in its security argument.

[2]This assumption can be relaxed–to accommodate a CHF with $\mathbb{M}_h \neq \mathbb{Y} \neq \mathbb{I}$–using two collision resistant hash functions H and G defined as follows:

$$\mathsf{H} : \mathbb{I} \mapsto \mathbb{M}_h \text{ and } \mathsf{G} : \mathbb{Y} \mapsto \mathbb{I}$$

These hash functions can be used in the protocol, and also in the security argument, to couple the CHF with the IBS.

5.3.1 Security Argument

For simplicity, we consider the security of the specific case of EU-sID-CMA model; we argue that the resulting IBS is EU-ID-CMA-secure. The details of both the security models is given in §**1.3.2**. The line of argument can be easily extended to other models as well[3].

Theorem 4. *Let $\mathcal{A}$ be an $(\epsilon, t, q_\varepsilon, q_s)$-adversary against the IBS $\mathfrak{I}$ in the EU-ID-CMA model. We can construct either*

(i) Algorithm $\mathcal{B}_s$ which $(\epsilon_s, t_s, q_\varepsilon, q_s)$-breaks $\mathfrak{I}_s$ in the EU-sID-CMA model, where

$$\epsilon_s \geq \frac{1}{3q_s}\epsilon \text{ and } t_s \leq t + (q_\varepsilon + q_\varepsilon)\tau_1, \text{ or}$$

(ii) Algorithm $\mathcal{B}_p$ which $(\epsilon_p, t_p, q_\varepsilon + q_s)$-breaks $\mathfrak{P}$ in the EU-GCMA model, where

$$\epsilon_p = \frac{1}{3}\epsilon \text{ and } t_p \leq t + (q_\varepsilon\tau_2 + q_s\tau_3), \text{ or}$$

(iii) Algorithm $\mathcal{B}_h$ which (ϵ_h, t_h)-breaks $\mathfrak{H}$, where

$$\epsilon_h = \frac{1}{3}\epsilon \text{ and } t_h \leq t + (q_\varepsilon + q_s)\tau_1 + (q_\varepsilon\tau_2 + q_s\tau_3).$$

Here, q_ε [resp. q_s] denotes the upper bound on the number of extract [resp. signature] queries that $\mathcal{A}$ can make. τ_1 is the time taken for generating a signature in $\mathfrak{P}$; τ_2 [resp. τ_3] denotes the time taken to generate a user secret key [resp. signature] in $\mathfrak{I}_s$.

Proof. We classify the forgeries (mutually-exclusively) into three: type 1,type 2 and type 3. A forgery qualifies as type 1 if the adversary makes *at least* one signature query on the "target" identity–the identity which $\mathcal{A}$ eventually forges on–and produces a forgery with the binding (provided by the simulator) *intact*. In both type 2 and type 3 forgeries, the binding is violated by the adversary by some means. The strategy adopted in each of the three cases is different; we give a reduction $\mathcal{B}_s$ for type 1, $\mathcal{B}_p$ for type 2 and $\mathcal{B}_h$ for type 3 adversary. The details follow.

Classifying the forgery. Consider an adversary $\mathcal{A}$ in the EU-ID-CMA model. At the beginning of the security game, $\mathcal{A}$ is given the challenge master public key $\mathtt{mpk}$ by (its challenger) $\mathcal{C}$. $\mathcal{A}$ produces a forgery after making a series of queries–extract and signature–adaptively with $\mathcal{C}$. Let $\mathtt{id}_i$ denote the i^{th} extract query made by $\mathcal{A}$, which is responded to with $\mathtt{usk}_i = (\mathtt{usk}_{s,i}, \dot{r}_i, \dot{\sigma}_{p,i})$ by $\mathcal{C}$. Similarly, $(\mathtt{id}_i, m_i)$ denotes the i^{th} signature query by $\mathcal{A}$, which is responded to with $\sigma_i = (\sigma_{s,i}, r_i, \sigma_{p,i})$ by $\mathcal{C}$. Note that the number of extract [resp.

[3] *e.g.*, consider the sM-sID-CMA model - the *selective-message, selective-identity chosen-message attack* model. It is similar to the EU-sID-CMA model, except that the adversary–in addition to committing to the target identity–has to commit to the target message too. If we start from sM-sID-CMA-secure IBS, we end up with an sM-ID-CMA-secure IBS.

signature] queries is bounded by q_ε [resp. q_s]. Finally, let $\hat{\sigma} = (\hat{\sigma}_s, \hat{r}, \hat{\sigma}_p)$ be the forgery produced by $\mathcal{A}$ on $(\hat{\mathtt{id}}, \hat{m})$. The identity $\hat{\mathtt{id}}$ is the so-called target identity. The forgeries can be partitioned into three types,*viz.*:

(i) type 1 forgery. $\mathcal{A}$ produces the forgery with $(\hat{\mathtt{id}}, \hat{r}) = (\mathtt{id}_i, r_i)$ for some $i \in \{1, \ldots, q_s\}$.

(ii) type 2 forgery. $\mathcal{A}$ produces the forgery with $(\hat{\mathtt{id}}, \hat{r}) \neq (\mathtt{id}_i, r_i)$ for all $i \in \{1, \ldots, q_s\}$, and with

  (a) $\mathsf{h}(\mathtt{ek}, \hat{\mathtt{id}}, \hat{r}) \neq \mathsf{h}(\mathtt{ek}, \dot{\mathtt{id}}_i, \dot{r}_i)$ for all $i \in \{1, \ldots, q_\varepsilon\}$, and
  (b) $\mathsf{h}(\mathtt{ek}, \hat{\mathtt{id}}, \hat{r}) \neq \mathsf{h}(\mathtt{ek}, \mathtt{id}_i, r_i)$ for all $i \in \{1, \ldots, q_s\}$.

(iii) type 3 forgery. $\mathcal{A}$ produces the forgery with $(\hat{\mathtt{id}}, \hat{r}) \neq (\mathtt{id}_i, r_i)$ for all $i \in \{1, \ldots, q_s\}$, but with

  (a) $\mathsf{h}(\mathtt{ek}, \hat{\mathtt{id}}, \hat{r}) = \mathsf{h}(\mathtt{ek}, \dot{\mathtt{id}}_i, \dot{r}_i)$ for some $i \in \{1, \ldots, q_\varepsilon\}$, or
  (b) $\mathsf{h}(\mathtt{ek}, \hat{\mathtt{id}}, \hat{r}) = \mathsf{h}(\mathtt{ek}, \mathtt{id}_i, r_i)$ for some $i \in \{1, \ldots, q_s\}$.

If $\mathcal{A}$ produces a forgery of type 1, we construct an algorithm $\mathcal{B}_s$ which breaks the IBS scheme $\mathfrak{I}_s$; whereas, in case of type 2 forgery, we construct an algorithm $\mathcal{B}_p$ which breaks the PKS scheme $\mathfrak{P}$; and finally, in case of type 3 forgery, we construct an algorithm $\mathcal{B}_h$ that breaks collision resistance property of the CHF $\mathfrak{H}$. We describe these reductions in the subsequent sections. □

5.3.1.1 Reduction $\mathcal{B}_s$.

Recall that in type 1 forgeries, $\mathcal{A}$ makes at least one signature query on the target identity $\hat{\mathtt{id}}$. The strategy is to guess the index of this identity and map it to the identity that $\mathcal{B}_s$ *commits* to (initially) in the EU-sID-CMA game. This leads to a degradation of $\mathsf{O}(q_s)$.

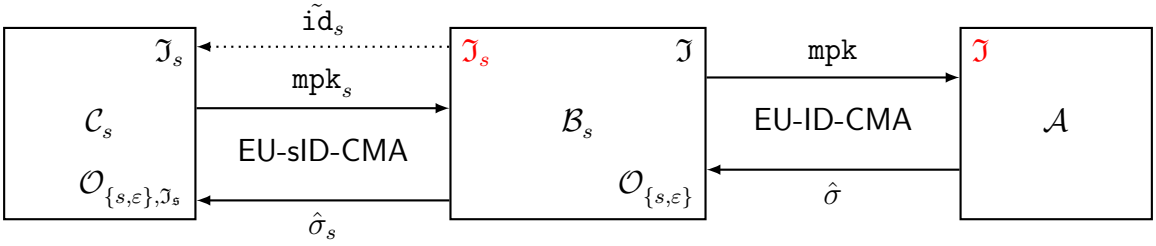

Figure 5.2: Reduction $\mathcal{B}_s$

Let $\mathcal{C}_s$ be the challenger in the EU-sID-CMA game. $\mathcal{B}_s$ plays the role of the adversary in the EU-sID-CMA game and, at the same time, the role of the challenger to $\mathcal{A}$ in the EU-ID-CMA game. It starts by running the Key Generation algorithms $\mathcal{K}$ and $\mathcal{G}_h$ to obtain $(\mathtt{pk}, \mathtt{sk})$ and $(\mathtt{ek}, \mathtt{td})$ respectively. In order to initiate the EU-sID-CMA game, $\mathcal{B}_s$ has to commit to a target identity. It does so by selecting an identity $\tilde{\mathtt{id}} \xleftarrow{\mathsf{U}} \mathbb{I}$ and a randomiser $\tilde{r} \xleftarrow{\mathsf{U}} \mathbb{R}$, and committing $\tilde{\mathtt{id}}_s \leftarrow \mathsf{h}(\mathtt{ek}, \tilde{\mathtt{id}}, \tilde{r})$ to $\mathcal{C}_s$. As a result, $\mathcal{C}_s$ releases the challenge master public key $\mathtt{mpk}_s$ to $\mathcal{B}_s$. $\mathcal{B}_s$ is also allowed access to a signature oracle $\mathcal{O}_{\mathcal{S},\mathfrak{I}_s}$. Now, $\mathcal{B}_s$ passes $\mathtt{mpk} := (\mathtt{mpk}_s, \mathtt{pk}, \mathtt{ek})$ as its own challenge master public key to $\mathcal{A}$. Next, $\mathcal{B}_s$ guesses $1 \leq \tilde{\ell} \leq q_s$ as the index of the target identity.

Mapping the identities. In order to track the mapping between the identities in $\mathfrak{I}$ and $\mathfrak{I}_s$, $\mathcal{B}_s$ maintains a table $\mathfrak{L}$. It also maintains a counter ℓ (initially 1) to track the index of these identities. $\mathfrak{L}$ contains tuples of the form $\langle \mathtt{id}, \mathtt{id}_s, \ell, \mathtt{usk}\rangle$. Here, $\mathtt{id}$ and $\mathtt{id}_s$ are the related identities from $\mathfrak{I}$ and $\mathfrak{I}_s$ respectively; ℓ is the index of the identity $\mathtt{id}$. The $\mathtt{usk}$-field stores the user secret key for $\mathtt{id}$ and hence contains elements of the form $(\mathtt{usk}_s, r, \sigma_p)$. If any component of the $\mathtt{usk}$-field is yet to be generated, it is indicated by a '$\perp$'.

An identity $\mathtt{id}$ has already been mapped if there exists $\langle \mathtt{id}_i, \mathtt{id}_{s,i}, \ell_i, \mathtt{usk}\rangle$ in $\mathfrak{L}$ such that $\mathtt{id}_i = \mathtt{id}$. For mapping a *fresh* identity $\mathtt{id}$, $\mathcal{B}_s$ chooses $r \xleftarrow{U} \mathbb{R}$ and sets $\mathtt{id}_s \leftarrow \mathrm{h}(\mathtt{ek}, \mathtt{id}, r)$.[4] Finally, it adds $\langle \mathtt{id}, \mathtt{id}_s, \ell, (\perp, r, \perp)\rangle$ to $\mathfrak{L}$ and increments ℓ by one. A more formal description of the mapping function is given below.

$\mathrm{M}_s(\mathtt{id})$:
if $\exists$ a tuple $\langle \mathtt{id}_i, \mathtt{id}_{s,i}, \ell_i, \mathtt{usk}_i\rangle \in \mathfrak{L}$ such that $(\mathtt{id}_i = \mathtt{id})$ **then**
  Set $\tau := (\mathtt{id}_{s,i}, \ell_i, \mathtt{usk}_i)$
else
  if $(\ell = \tilde{\ell})$ **then return** set $r \leftarrow \mathrm{h}^{-1}(\mathtt{td}, \tilde{\mathtt{id}}, \tilde{r}, \mathtt{id})$
  else return choose $r \xleftarrow{U} \mathbb{R}$
  Compute $\mathtt{id}_s \leftarrow \mathrm{h}(\mathtt{ek}, \mathtt{id}, r)$ and set $\tau := (\mathtt{id}_s, \ell, (\perp, r, \perp))$
  Add $\langle \mathtt{id}, \mathtt{id}_s, \ell, (\perp, r, \perp)\rangle$ to $\mathfrak{L}$ and increment ℓ by one
end if
return τ

Queries. The extract and signature queries by $\mathcal{A}$ are answered as per the following specifications.

Extract query, $\mathcal{O}_{\varepsilon,\mathfrak{I}}(\mathtt{id})$: Invoke the function $\mathrm{M}_s(\mathtt{id})$ to obtain $(\mathtt{id}_s, \ell, (\mathtt{usk}_s, r, \sigma_p))$.

(i) If $(\ell = \tilde{\ell})$ then $\mathcal{B}_s$ *aborts* (abort_1).

(ii) Otherwise, if $(\mathtt{usk}_s \neq \perp)$ then return $\mathtt{usk} := (\mathtt{usk}_s, r, \sigma_p)$ as the user secret key.

(iii) Otherwise, $\mathcal{B}_s$ makes an extract query with $\mathcal{O}_{\varepsilon,\mathfrak{I}_s}$ on $\mathtt{id}_s$ to obtain $\mathtt{usk}_s$. Next, it uses the knowledge of $\mathtt{sk}$ to compute $\sigma_p := \mathcal{S}_p(\mathtt{id}_s, \mathtt{sk})$. Finally, it returns $\mathtt{usk} := (\mathtt{usk}_s, r, \sigma_p)$ as the user secret key and updates the $\mathtt{usk}$-field of the tuple corresponding to $\mathtt{id}$ in $\mathfrak{L}$.

Signature query, $\mathcal{O}_{\mathcal{S},\mathfrak{I}}(\mathtt{id}, m)$: Invoke the function $\mathrm{M}_s(\mathtt{id})$ to obtain $(\mathtt{id}_s, \ell, (\mathtt{usk}_s, r, \sigma_p))$.

(i) If $((\ell = \tilde{\ell}) \vee (\mathtt{usk}_s = \perp))$ then $\mathcal{B}_s$ then makes a signature query with $\mathcal{O}_{\mathcal{S},\mathfrak{I}_s}$ on $(\mathtt{id}_s, m)$ to obtain σ_s. It uses the knowledge of $\mathtt{sk}$ to compute $\sigma_p := \mathcal{S}_p(\mathtt{id}_s, \mathtt{sk})$. Finally, it returns $\sigma := (\sigma_s, r, \sigma_p)$ as the signature.

(ii) Otherwise, $\mathcal{B}_s$ uses the knowledge of the user secret key $\mathtt{usk}$ to generate the signature, *i.e.* it returns $\sigma := \mathcal{S}(\mathtt{id}, m, \mathtt{usk})$.

[4] If there already exists a tuple $\langle \mathtt{id}_i, r_i, \mathtt{id}_{s,i}, \ell_i, \mathtt{usk}_i\rangle$ such that $\mathtt{id}_{s,i} = \mathtt{id}_s$, to maintain injection in the mapping, $\mathcal{B}_s$ repeats the process with a fresh r.

Forgery. At the end of the simulation, $\mathcal{A}$ produces a type 1 forgery $\hat{\sigma} = (\hat{\sigma}_s, \hat{r}, \hat{\sigma}_p)$ on $(\hat{\mathtt{id}}, \hat{m})$. Let $\langle \mathtt{id}_{\hat{i}}, \mathtt{id}_{s,\hat{i}}, l_{\hat{i}}, \mathtt{usk}_{\hat{i}} \rangle$ be the tuple in $\mathfrak{L}$ such that $\mathtt{id}_{\hat{i}} = \hat{\mathtt{id}}$. If $\ell_{\hat{i}}$ matches $\mathcal{B}_s$'s initial guess for the target index (*i.e.* $\ell_{\hat{i}} = \tilde{\ell}$), it wins the EU-sID-CMA game with $\mathcal{C}_s$ by passing $\hat{\sigma}_s$ as a forgery on $(\tilde{\mathtt{id}}_s, \hat{m})$ to $\mathcal{C}_s$; otherwise it *aborts* (abort_2).

Analysis. The probability of success of the reduction $\mathcal{B}_s$ is governed by the two events abort_1 and abort_2. To be precise,

$$\begin{aligned} \epsilon_s &= \Pr\left[\neg\mathsf{abort}_1 \wedge \neg\mathsf{abort}_2\right] \epsilon \\ &= \Pr\left[\neg\mathsf{abort}_1 \mid \neg\mathsf{abort}_2\right] \Pr\left[\neg\mathsf{abort}_2\right] \epsilon. \end{aligned}$$

Since $\tilde{\ell}$ is hidden from the adversary, it is easy to see that

$$\Pr\left[\neg\mathsf{abort}_2\right] = \Pr\left[\ell_{\hat{i}} = \tilde{\ell}\right] = 1/q_s.$$

On the other hand, $\Pr\left[\neg\mathsf{abort}_1 \mid \neg\mathsf{abort}_2\right] = 1$. This follows from the fact that if the simulator's guess of the target index was indeed correct ($\neg\mathsf{abort}_2$), then the adversary *would not* have made an extract query on that identity (which causes abort_1). Thus, $\epsilon_s = \epsilon/q_s$. As for the time complexity, if τ_1 is the time taken for generating a signature in $\mathfrak{P}$, then the time taken by $\mathcal{B}_s$ can be easily seen as $t_s \leq t + (q_\varepsilon + q_s)\tau_1$.

5.3.1.2 Reduction $\mathcal{B}_p$.

The strategy adopted in $\mathcal{B}_p$ is similar to that in security arguments of [HW09, ST01]. It is also, on a high level, related to the technique used in [BB04b] for proving the security of the EU-CMA-secure PKS scheme constructed from EU-GCMA-secure PKS scheme (using CHF implicitly). The details follow.

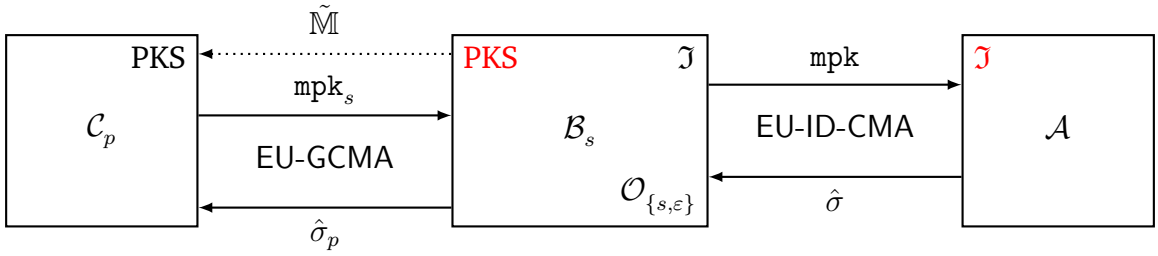

Figure 5.3: Reduction $\mathcal{B}_p$

Let $\mathcal{C}_p$ be the challenger in the EU-GCMA game. $\mathcal{B}_p$ plays the role of the adversary in the EU-GCMA game and, at the same time, the role of the challenger to $\mathcal{A}$ in the EU-ID-CMA game. It starts by running the Key Generation algorithms $\mathcal{G}_s$ and $\mathcal{G}_h$ to obtain $(\mathtt{mpk}_s, \mathtt{msk}_s)$ and $(\mathtt{ek}, \mathtt{td})$ respectively. In order to initiate the EU-GCMA game, $\mathcal{B}_p$ has to commit to a set of q_s messages to $\mathcal{C}_p$. On the other hand, it also has to answer the adaptive queries by $\mathcal{A}$. Let's see how this is accomplished using the CHF. $\mathcal{B}_p$ first selects pairs $(\tilde{\mathtt{id}}_1, \tilde{r}_1), \ldots, (\tilde{\mathtt{id}}_{q_s}, \tilde{r}_{q_s})$ independently and uniformly at random from $\mathbb{I} \times \mathbb{R}$. Next, it commits $\tilde{\mathbb{M}} := \{\tilde{\mathtt{id}}_{s,1}, \ldots, \tilde{\mathtt{id}}_{s,q_s}\}$ to $\mathcal{C}_s$, where $\tilde{\mathtt{id}}_{s,i} \leftarrow \mathrm{h}(\mathtt{ek}, \tilde{\mathtt{id}}_i, \tilde{r}_i)$. As a result, $\mathcal{C}_p$ releases the challenge public key $\mathtt{pk}$ to $\mathcal{B}_p$

along with the set of signatures $\{\sigma_{p,1}, \ldots, \sigma_{p,q_s}\}$ on the (respective) committed messages. All this information is stored in a table $\mathfrak{C}$ as tuples $\langle \tilde{\mathtt{id}}_i, \tilde{r}_i, \tilde{\mathtt{id}}_{s,i}, \sigma_{p,i} \rangle$. Now, $\mathcal{B}_p$ initiates the EU-ID-CMA game by passing $\mathtt{mpk} := (\mathtt{mpk}_s, \mathtt{pk}, \mathtt{ek})$ as the challenge master public key to $\mathcal{A}$.

Mapping the identities. $\mathcal{B}_p$ too maintains the table $\mathfrak{L}$; but, it's structure is slightly different from that in $\mathcal{B}_s$. $\mathfrak{L}$ contains tuples of the form $\langle \mathtt{id}, \tilde{\mathtt{id}}_s, \mathtt{usk} \rangle$. Here, $\mathtt{id}$ and $\mathtt{id}_s$ are the related identities from $\mathfrak{I}$ and $\mathfrak{I}_s$ respectively. The $\mathtt{usk}$-field stores the user secret key for $\mathtt{id}$ and hence contains elements of the form $(\mathtt{usk}_s, r, \sigma_p)$. If any component of the $\mathtt{usk}$-field is yet to be generated, it is indicated by a '$\perp$'.

The way in which the mapping is maintained between the identities is somewhat different from that in $\mathcal{B}_s$. For mapping a *fresh* identity $\mathtt{id}$, $\mathcal{B}_p$ first picks an tuple $\mathfrak{t} = \langle \tilde{\mathtt{id}}_s, \tilde{\mathtt{id}}, \tilde{r}, \sigma_p \rangle$ randomly from $\mathfrak{C}$. It then computes $r \leftarrow \mathrm{h}^{-1}(\mathtt{td}, \tilde{\mathtt{id}}, \tilde{r}, \mathtt{id})$, and adds the tuple $\langle \mathtt{id}, \tilde{\mathtt{id}}_s, (\perp, r, \sigma_p) \rangle$ to $\mathfrak{L}$. Finally it removes the tuple $\mathfrak{t}$ from $\mathfrak{C}$. As a result of these actions, $\mathtt{id}$ is effectively mapped to $\tilde{\mathtt{id}}_s$ since $\mathrm{h}(\mathtt{ek}, \mathtt{id}, r) = \mathrm{h}(\mathtt{ek}, \tilde{\mathtt{id}}, \tilde{r}) = \tilde{\mathtt{id}}_s$. A more formal description follows.

$\mathrm{M}_p(\mathtt{id})$:
if $\exists$ a tuple $\langle \mathtt{id}_i, \mathtt{id}_{s,i}, \mathtt{usk}_i \rangle \in \mathfrak{L}$ such that $(\mathtt{id}_i = \mathtt{id})$ **then**
  Set $\tau := (\mathtt{id}_{s,i}, \mathtt{usk}_i)$
else
  Pick $\mathfrak{t} \xleftarrow{\$} \mathfrak{C}$ and parse it as $\langle \tilde{\mathtt{id}}, \tilde{r}, \tilde{\mathtt{id}}_s, \sigma_p \rangle$
  Compute $r \leftarrow \mathrm{h}^{-1}(\mathtt{td}, \tilde{\mathtt{id}}, \tilde{r}, \mathtt{id})$ and set $\tau := (\mathtt{id}_s, (\perp, r, \sigma_p))$
  Add $\langle \mathtt{id}, \mathtt{id}_s, (\perp, r, \sigma_p) \rangle$ to $\mathfrak{L}$ and remove $\mathfrak{t}$ from $\mathfrak{C}$
end if
return τ

Queries: The extract and signature queries by $\mathcal{A}$ are answered as follows.

Extract query, $\mathcal{O}_{\varepsilon,\mathfrak{I}}(\mathtt{id})$: Invoke the function $\mathrm{M}_p(\mathtt{id})$ to obtain $(\tilde{\mathtt{id}}_s, \ell, (\mathtt{usk}_s, r, \sigma_p))$.

(i) If $(\mathtt{usk}_s \neq \perp)$ then return $\mathtt{usk} := (\mathtt{usk}_s, r, \sigma_p)$ as the user secret key.

(ii) Otherwise, $\mathcal{B}_s$ uses the knowledge of the master secret key $\mathtt{msk}_s$ to generate the user secret key $\mathtt{usk}_s := \mathcal{E}_s(\tilde{\mathtt{id}}_s, \mathtt{msk}_s)$ for $\tilde{\mathtt{id}}_s$. It returns $\mathtt{usk} := (\mathtt{usk}_s, r, \sigma_p)$ as the user secret key for $\mathtt{id}$ and updates the $\mathtt{usk}_s$-field of the tuple corresponding to $\mathtt{id}$ in $\mathfrak{L}$.

Signature query, $\mathcal{O}_{\mathcal{S},\mathfrak{I}}(\mathtt{id}, m)$: Invoke the function $\mathrm{M}_p(\mathtt{id})$ to obtain $(\tilde{\mathtt{id}}_s, \ell, (\mathtt{usk}_s, r, \sigma_p))$.

(i) If $(\mathtt{usk}_s \neq \perp)$ then $\mathcal{B}_p$ uses the knowledge of $\mathtt{usk}$ to return the signature $\sigma := \mathcal{S}(\mathtt{id}, m, \mathtt{usk})$

(ii) Otherwise, $\mathcal{B}_s$ uses step (ii) of **Extract query** to generate a user secret key $\mathtt{usk}$ for $\mathtt{id}$ and then use this $\mathtt{usk}$ to return a signature $\sigma := \mathcal{S}(\mathtt{id}, m, \mathtt{usk})$.

Forgery. Finally, $\mathcal{A}$ produces a forgery $\hat{\sigma} = (\hat{\sigma}_s, \hat{r}, \hat{\sigma}_p)$ on $(\hat{\mathtt{id}}, \hat{m})$. As the forgery is of type 2, it implies $\hat{\mathtt{id}}_s := \mathrm{h}(\mathtt{ek}, \hat{\mathtt{id}}, \hat{r}) \notin \tilde{\mathbb{M}}$. Therefore $\hat{\sigma}_p$ is a valid forgery in the EU-GCMA game and $\mathcal{B}_p$ passes it to $\mathcal{C}_s$ to win the game.

Analysis. Since no abort is involved in $\mathcal{S}_p$, there is no degradation involved either. Thus, its advantage in attacking $\mathfrak{P}$ is $\epsilon_p = \epsilon$. If τ_2 and τ_3 denote the time taken for generating a secret key and a signature respectively in $\mathfrak{I}_s$, then the time taken by $\mathcal{B}_p$ is $t_p \leq t + (q_\varepsilon \tau_2 + q_s \tau_3)$.

5.3.1.3 Reduction $\mathcal{B}_h$.

$\mathcal{B}_h$ first obtains the challenge evaluation key $\mathtt{ek}$ for $\mathfrak{H}$ from its challenger $\mathcal{C}_h$. Then it invokes the algorithms $\mathcal{G}_s$ and $\mathcal{K}$ to generate $(\mathtt{msk}_s, \mathtt{mpk}_s)$ and $(\mathtt{sk}, \mathtt{pk})$ respectively. Finally, it passes $(\mathtt{mpk}_s, \mathtt{pk}, \mathtt{ek})$ as the challenge master public key to $\mathcal{A}$.

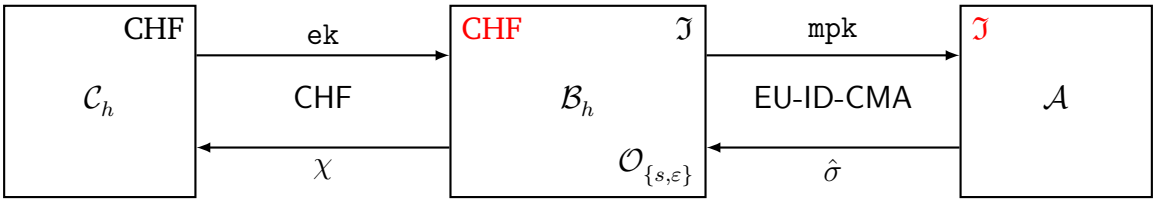

Figure 5.4: Reduction $\mathcal{B}_h$

Mapping the identities. The table used for maintaining mapping has the same structure as in $\mathcal{B}_p$. However, the actual method used for mapping identities is far simpler than in $\mathcal{B}_p$ as shown below.

$\mathrm{h}M(\mathtt{id})$:
if $\exists$ a tuple $\langle \mathtt{id}_i, \mathtt{id}_{s,i}, \mathtt{usk}_i \rangle \in \mathfrak{L}$ such that $(\mathtt{id}_i = \mathtt{id})$ **then**
  Set $\tau := (\mathtt{id}_{s,i}, \mathtt{usk}_i)$
else
  Pick $r \xleftarrow{U} \mathbb{R}$ and compute $\mathtt{id}_s := \mathrm{h}(\mathtt{ek}, \mathtt{id}, r)$
  Set $\tau := (\mathtt{id}_s, (\perp, r, \sigma_p))$ add $\langle \mathtt{id}, \mathtt{id}_s, (\perp, r, \perp) \rangle$ to $\mathfrak{L}$
end if
return τ

Queries: The extract and signature queries by $\mathcal{A}$ are answered as follows.

Extract query, $\mathcal{O}_{\varepsilon,\mathfrak{I}}(\mathtt{id})$: Invoke the function $\mathrm{h}M(\mathtt{id})$ to obtain $(\mathtt{id}_s, (\mathtt{usk}_s, r, \sigma_p))$.

(i) If $(\mathtt{usk}_s \neq \perp)$ then return $\mathtt{usk} := (\mathtt{usk}_s, r, \sigma_p)$ as the user secret key.

(ii) Otherwise, $\mathcal{B}_s$ uses the knowledge of the master secret key $\mathtt{msk}_s$ to generate the user secret key $\mathtt{usk}_s := \mathcal{E}_s(\mathtt{id}_s, \mathtt{msk}_s)$ for $\mathtt{id}_s$. It also uses $\mathtt{sk}$ to generate $\sigma_p := \mathcal{S}_p(\mathtt{id}_s)$. Finally, $\mathcal{B}_h$ returns $\mathtt{usk} := (\mathtt{usk}_s, r, \sigma_p)$ as the user secret key for $\mathtt{id}$ and updates the $\mathtt{usk}_s$-field and σ_p-field of the tuple corresponding to $\mathtt{id}$ in $\mathfrak{L}$.

Signature query, $\mathcal{O}_{\mathcal{S},\mathfrak{I}}(\mathtt{id}, m)$: Invoke the function $\mathrm{h}M(\mathtt{id})$ to obtain $(\mathtt{id}_s, (\mathtt{usk}_s, r, \sigma_p))$.

(i) If $(\mathtt{usk}_s \neq \perp)$ then $\mathcal{B}_p$ uses the knowledge of $\mathtt{usk}$ to return the signature $\sigma := \mathcal{S}(\mathtt{id}, m, \mathtt{usk})$

(ii) Otherwise, $\mathcal{B}_s$ uses step (ii) of **Extract query** to generate a user secret key $\mathtt{usk}$ for $\mathtt{id}$ and then use this $\mathtt{usk}$ to return a signature $\sigma := \mathcal{S}(\mathtt{id}, m, \mathtt{usk})$.

Queries. The extract and signature queries by $\mathcal{A}$ are answered as follows.

Extract query, $\mathcal{O}_{\varepsilon,\mathfrak{I}}(\mathtt{id})$: Let $(\mathtt{id}, \mathtt{id}_s, r)$ be the mapping for $\mathtt{id}$. $\mathcal{B}_s$ uses the knowledge of the master secret key $\mathtt{msk}_s$ to generate the user secret key for $\mathtt{id}_s$ in $\mathfrak{I}_s$, *i.e.* $\mathtt{usk}_s := \mathcal{E}_s(\mathtt{id}_s, \mathtt{msk}_s, \mathtt{mpk}_s)$. It then uses the knowledge of secret key $\mathtt{sk}$ to generate a signature σ_p on $\mathtt{id}_s$. It returns the user secret key for $\mathtt{id}$

$$\mathtt{usk} := (\mathtt{usk}_s, r, \sigma_p).$$

Signature query, $\mathcal{O}_{\mathcal{S},\mathfrak{I}}(\mathtt{id}, m)$: First, $\mathcal{B}_p$ generates the user secret key for $\mathtt{id}$ by running $\mathcal{E}$, *i.e.* $\mathtt{usk} = \mathcal{E}(\mathtt{id}, \mathtt{msk}, \mathtt{mpk})$. Then, $\mathcal{B}_p$ uses $\mathtt{usk}$ to return the signature

$$\sigma := \mathcal{S}(\mathtt{id}, m, \mathtt{usk}).$$

Forgery. Finally, $\mathcal{A}$ produces a type 3 forgery $\hat{\sigma} = (\hat{\sigma}_s, \hat{r}, \hat{\sigma}_p)$ on $(\hat{\mathtt{id}}, \hat{m})$. Recall that this implies $\mathcal{A}$ produces the forgery with $(\hat{\mathtt{id}}, \hat{r}) \neq (\mathtt{id}_i, r_i)$ for all $i \in \{1, \ldots, q_s\}$, but with

(a) $\mathrm{h}(\mathtt{ek}, \hat{\mathtt{id}}, \hat{r}) = \mathrm{h}(\mathtt{ek}, \dot{\mathtt{id}}_i, \dot{r}_i)$ for some $i \in \{1, \ldots, q_\varepsilon\}$, or

(b) $\mathrm{h}(\mathtt{ek}, \hat{\mathtt{id}}, \hat{r}) = \mathrm{h}(\mathtt{ek}, \mathtt{id}_i, r_i)$ for some $i \in \{1, \ldots, q_s\}$.

Both the cases tantamount to breaking the collision resistance property of $\mathfrak{H}$. In case (a) [resp. case (b)] $\mathcal{B}_h$ passes $\chi := ((\hat{\mathtt{id}}, \hat{r}), (\dot{\mathtt{id}}_i, \dot{r}_i))$ [resp. $\chi := ((\hat{\mathtt{id}}, \hat{r}), (\mathtt{id}_i, r_i))$] as a collision to the challenger $\mathcal{C}_h$ to win the game.

Analysis. As in $\mathcal{B}_p$, there is no abort involved in $\mathcal{B}_h$. Therefore, its advantage in attacking $\mathfrak{H}$ is $\epsilon_p = \epsilon$. Again, if τ_2 and τ_3 denote the time taken for generating a secret key and a signature respectively in $\mathfrak{I}_s$, then the time taken by $\mathcal{B}_h$ is $t_p \leq t + (q_\varepsilon \tau_2 + q_s \tau_3)$.

5.4 Transforming from the EU-wID-CMA model

The construction technique described in the previous section can as well be used with a *relaxed* version of the selective-identity model which we call the *weak selective-identity* (wID) model. In this model the adversary, apart from committing to the "target" identity $\tilde{\mathtt{id}}$, has to commit a set of "query" identities $\tilde{\mathbb{I}}$. The adversary is allowed to query the extract oracle only on identities belonging to $\tilde{\mathbb{I}}$; whereas, it is allowed to query the signature oracle with identities from $\tilde{\mathbb{I}}$ *as well as* the target identity. Finally, as in the sID model, the adversary

has to produce a forgery on $\tilde{\mathtt{id}}$. One may see the analogy between the EU-GCMA model for PKS and the wID model–both involve the adversary committing, beforehand, to the identities/messages that it wants to queries to. The only change involved is in the security argument–the way in which mapping is handled by the simulator. We elaborate on this later. But first, let's formally define the EU-wID-CMA model for IBS.

Definition 16 (EU-wID-CMA Game)**.** The security of an IBS scheme in the EU-wID-CMA model is argued in terms of the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Commitment: $\mathcal{A}$ commits to a target identity $\tilde{\mathtt{id}}$ and a set of query identities $\tilde{\mathbb{I}} := \{\tilde{\mathtt{id}}_1, \ldots, \tilde{\mathtt{id}}_{\tilde{q}}\} \subset \mathbb{I} \setminus \{\tilde{\mathtt{id}}\}$.

Set-up: $\mathcal{C}$ runs the set-up algorithm $\mathcal{G}$ to obtain the master keys ($\mathtt{mpk}$,$\mathtt{msk}$). It passes $\mathtt{mpk}$ as the challenge master public key to $\mathcal{A}$.

Queries: $\mathcal{A}$ can adaptively make extract queries on identities from $\tilde{\mathbb{I}}$ to an oracle $\mathcal{O}_{\varepsilon}$ and signature queries involving identities from $\tilde{\mathbb{I}} \cup \{\tilde{\mathtt{id}}\}$ to an oracle $\mathcal{O}_{s}$. These queries are handled as follows.

Extract query, $\mathcal{O}_{\varepsilon}(\mathtt{id})$: $\mathcal{A}$ asks for the secret key of a user with identity $\mathtt{id} \in \tilde{\mathbb{I}}$. $\mathcal{C}$ computes $\mathtt{usk} := \mathcal{E}(\mathtt{id})$ and passes it to $\mathcal{A}$.

Signature query, $\mathcal{O}_{s}(\mathtt{id}, m)$: $\mathcal{A}$ asks for the signature of a user with identity $\mathtt{id} \in \tilde{\mathbb{I}} \cup \{\tilde{\mathtt{id}}\}$ on a message m. $\mathcal{C}$ first runs $\mathcal{E}$ on $\mathtt{id}$ to obtain the user secret key $\mathtt{usk}$. Next, it computes $\sigma := \mathcal{S}(\mathtt{id}, m, \mathtt{usk})$ and forwards it to $\mathcal{A}$.

Forgery: $\mathcal{A}$ outputs a signature $\hat{\sigma}$ on a message $\hat{m}$ and the target identity $\tilde{\mathtt{id}}$ wins the game if:

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

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

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

where the oracles $\mathcal{O}_{\varepsilon}$ and $\mathcal{O}_{s}$ are restricted to answering queries involving identities from $\tilde{\mathbb{I}}$ and $\tilde{\mathbb{I}} \cup \{\tilde{\mathtt{id}}\}$ respectively. An adversary $\mathcal{A}$ is said to be an $(\epsilon, t, q_{\varepsilon}, q_{s}, \tilde{q})$-forger of an IBS scheme in the EU-wID-CMA model 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, provided the number of identities involved in the signature and extract queries, excluding the target identity, is at most $\tilde{q}$. It is easy to see that $\tilde{q} \leq q_{\varepsilon} + q_{s}$. As we pointed out, the same transformation technique applies; the only change is in the security argument.

Theorem 5. *Let $\mathcal{A}$ be an $(\epsilon, t, q_{\varepsilon}, q_{s})$-adversary against the IBS $\mathfrak{I}$ in the EU-ID-CMA model. We can construct either*

(i) Algorithm $\mathcal{B}_w$ which $(\epsilon_w, t_w, q_\varepsilon, q_s, q_\varepsilon + q_s)$-breaks $\mathfrak{I}_w$ in the EU-wID-CMA model, where

$$\epsilon_w \geq \frac{1}{3q_s}\epsilon \text{ and } t_w \leq t + (q_\varepsilon + q_\varepsilon)\tau_1, \text{ or}$$

(ii) Algorithm $\mathcal{B}_p$ which $(\epsilon_p, t_p, q_\varepsilon + q_s)$-breaks $\mathfrak{P}$ in the EU-GCMA model, where

$$\epsilon_p = \frac{1}{3}\epsilon \text{ and } t_p \leq t + (q_\varepsilon\tau_2 + q_s\tau_3), \text{ or}$$

(iii) Algorithm $\mathcal{B}_h$ which (ϵ_h, t_h)-breaks $\mathfrak{H}$, where

$$\epsilon_h = \frac{1}{3}\epsilon \text{ and } t_h \leq t + (q_\varepsilon + q_s)\tau_1 + (q_\varepsilon\tau_2 + q_s\tau_3).$$

Here, q_ε [resp. q_s] denotes the upper bound on the number of extract [resp. signature] queries that $\mathcal{A}$ can make. τ_1 is the time taken for generating a signature in $\mathfrak{P}$; τ_2 [resp. τ_3] denotes the time taken to generate a user secret key [resp. signature] in $\mathfrak{I}_s$.

Proof. The security argument is similar to the one discussed in §**5.3.1**. The only difference lies in the way in which the mapping of identities is handled in $\mathcal{B}_w$.

Let $\mathcal{C}_w$ be the challenger in the EU-wID-CMA game. $\mathcal{B}_w$ plays the role of the adversary in the EU-wID-CMA game and, at the same time, the role of the challenger to $\mathcal{A}$ in the EU-ID-CMA game. In order to initiate the EU-wID-CMA game, $\mathcal{B}_w$ has to commit to a target identity and a target set. It selects an identity $\tilde{\mathtt{id}} \xleftarrow{U} \mathbb{I}$ and a randomiser $\tilde{r} \xleftarrow{U} \mathbb{R}$, and commits $\tilde{\mathtt{id}}_w \leftarrow \mathrm{h}(\mathtt{ek}, \tilde{\mathtt{id}}, \tilde{r})$ as the target identity to $\mathcal{C}_w$. Similarly, it selects $\{\tilde{\mathtt{id}}_i, \ldots, \tilde{\mathtt{id}}_{\tilde{q}}\} \xleftarrow{\$} \mathbb{I}$, $\{\tilde{r}_1, \ldots, \tilde{r}_{\tilde{q}}\} \xleftarrow{\$} \mathbb{R}$ and commits $\hat{\mathbb{I}} := \{\tilde{\mathtt{id}}_{1,w}, \ldots, \tilde{\mathtt{id}}_{\tilde{q},w}\}$, where $\tilde{\mathtt{id}}_{i,w} \leftarrow \mathrm{h}(\mathtt{ek}, \tilde{\mathtt{id}}_i, \tilde{r}_i)$, as the target set to $\mathcal{C}_w$. As a result, $\mathcal{C}_w$ releases the challenge master public key $\mathtt{mpk}_w$ to $\mathcal{B}_s$. All this information is stored in a table, denoted by $\mathfrak{D}$, as tuples $\langle \tilde{\mathtt{id}}_i, \tilde{r}_i, \tilde{\mathtt{id}}_{w,i} \rangle$.

Mapping. $\mathcal{B}_w$ maintains a table $\mathfrak{L}$ with structure the same as that in reduction $\mathcal{B}_p$. For mapping a *fresh* identity $\mathtt{id}$, $\mathcal{B}_s$ chooses a tuple $\mathfrak{t} = \langle \tilde{\mathtt{id}}, \tilde{r}, \tilde{\mathtt{id}}_w \rangle$ randomly from $\mathfrak{D}$. Next, it computes $r := \mathrm{h}^{-1}(\mathtt{td}, \tilde{\mathtt{id}}, \tilde{r}, \mathtt{id})$ and adds $\langle \mathtt{id}, \tilde{\mathtt{id}}_w, (\perp, r, \perp) \rangle$ to $\mathfrak{L}$. Finally, it removes the tuple $\mathfrak{t}$ from $\mathfrak{D}$. As a result of these actions, $\mathtt{id}$ is effectively mapped to $\tilde{\mathtt{id}}_w$ as $\mathrm{h}(\mathtt{ek}, \mathtt{id}, r) = \mathrm{h}(\mathtt{ek}, \tilde{\mathtt{id}}, \tilde{r}) = \tilde{\mathtt{id}}_w$. A more formal description follows.

$\mathrm{M}_w(\mathtt{id})$:
if $\exists$ a tuple $\langle \mathtt{id}_i, \mathtt{id}_{w,i}, \mathtt{usk}_i \rangle \in \mathfrak{L}$ such that $(\mathtt{id}_i = \mathtt{id})$ **then**
  Set $\tau := (\mathtt{id}_{w,i}, \mathtt{usk}_i)$
else
  Pick $\mathfrak{t} \xleftarrow{\$} \mathfrak{C}$ and parse it as $\langle \tilde{\mathtt{id}}, \tilde{r}, \tilde{\mathtt{id}}_w \rangle$
  Compute $r \leftarrow \mathrm{h}^{-1}(\mathtt{td}, \tilde{\mathtt{id}}, \tilde{r}, \mathtt{id})$ and set $\tau := (\mathtt{id}_w, (\perp, r, \perp))$
  Add $\langle \mathtt{id}, \mathtt{id}_w, (\perp, r, \perp) \rangle$ to $\mathfrak{L}$ and remove $\mathfrak{t}$ from $\mathfrak{C}$
end if
return τ

□

Remark 19 (Comparison with the folklore paradigm)**.** The (identities-based) signature of an IBS scheme constructed using the folklore technique consists of *two* (public-key) signatures and *one* public key of the underlying (fully-secure) PKS. In contrast, the signature of an IBS scheme using our approach consists of *one* signature each of the underlying (wID-secure) IBS and (weakly-secure) PKS and *one* randomiser from the CHF. The time taken for signing and verification is comparable, bar the time taken to compute the hash value.

Chapter 6

Conclusions

In this thesis we have identified certain shortcomings in the original security argument of GG-IBS in [GG09]. Based on our observations, we provide a new elaborate security argument for the same scheme. Although, the reductions are tighter than their counterparts in the original security argument, they are still quite loose due to the usage of the MF Algorithm. This motivated us to explore means to launch the nested replay attack in a more effective manner than in the MF Algorithm and our (re)search culminated with the notion of (in)dependency. The result was a cleaner, tighter security argument for GG-IBS with the effective degradation down from O (q^6) to O (q^3).

The bound on security reductions for the Schnorr signature has been well-studied [PV05, GBL08, Seu12]. In the same vein, giving a bound on the reductions for GG-IBS would truly complete the jigsaw (of GG-IBS). In addition, studying the effect of (in)dependency on the MF Algorithm should also be worthwhile [CK13a].

On a separate note, we described a generic transformation from sID/wID IBS to full-identity IBS using a chameleon hash function and a GCMA-secure PKS scheme. It was also supported by an argument, without using random oracles, that the resulting IBS is secure in the full-identity model with only linear degradation incurred. An interesting problem would be to replace the EU-GCMA PKS with a more primitive construct. Extending the transformation for Hierarchical IBS could be yet another challenging task.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. (Cited on page iv.)

[ARP03] Sattam Al-Riyami and Kenneth Paterson. Certificateless public key cryptography. In Chi-Sung Laih, editor, *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473. Springer Berlin / Heidelberg, 2003. (Cited on page 2.)

[BB04a] 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 11, 38, 39, 49, 71 and 97.)

[BB04b] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer Berlin / Heidelberg, 2004. (Cited on page 77.)

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156 – 189, 1988. (Cited on page 72.)

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin / Heidelberg, 2001. (Cited on pages 2, 6, 7, 11 and 71.)

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 390–399, New York, NY, USA, 2006. ACM. (Cited on pages 12, 14, 27, 28 and 29.)

[BNN04] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Security proofs for identity-based identification and signature schemes. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 268–286. Springer Berlin / Heidelberg, 2004. (Cited on pages 8, 10, 12 and 37.)

[BPW12] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. *Journal of Cryptology*, 25:57–115, 2012. (Cited on pages 12, 14, 23, 28, 30 and 36.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM. (Cited on pages 5 and 11.)

[CC] Jae Choon and Jung-Hee Cheon. An identity-based signature from gap Diffie-Hellman groups. In Yvo Desmedt, editor, *Public Key Cryptography — PKC 2003*, Lecture Notes in Computer Science. (Cited on page 1.)

[CFH+09] Yang Cui, Eiichiro Fujisaki, Goichiro Hanaoka, Hideki Imai, and Rui Zhang. Formal security treatments for ibe-to-signature transformation: Relations among security notions. *IEICE Transactions*, 92-A(1):53–66, 2009. (Cited on page 71.)

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 646–646. Springer Berlin / Heidelberg, 2003. (Cited on pages 9, 12 and 71.)

[CK13a] Sanjit Chatterjee and Chethan Kamath. A closer look at multiple-forking: Leveraging (in)dependence for a tighter bound. Cryptology ePrint Archive, Report 2013/651, 2013. `http://eprint.iacr.org/`. (Cited on page 84.)

[CK13b] Sanjit Chatterjee and Chethan Kamath. From selective-id to full-id IBS without random oracles. In Benedikt Gierlichs, Sylvain Guilley, and Debdeep Mukhopadhyay, editors, *Security, Privacy, and Applied Cryptography Engineering*, volume 8204 of *Lecture Notes in Computer Science*, pages 172–190. Springer Berlin Heidelberg, 2013. (Cited on pages 12, 71 and 73.)

[CKK13] Sanjit Chatterjee, Chethan Kamath, and Vikas Kumar. Galindo-Garcia identity-based signature revisited. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology - ICISC 2012*, volume 7839 of *Lecture Notes in Computer Science*, pages 456–471. Springer Berlin / Heidelberg, 2013. Full version available in Cryptology ePrint Archive, Report 2012/646, http://eprint.iacr.org/2012/646. (Cited on pages 12 and 30.)

[CMW12] Sherman S. M. Chow, Changshe Ma, and Jian Weng. Zero-knowledge argument for simultaneous discrete logarithms. *Algorithmica*, 64(2):246–266, 2012. (Cited on page 30.)

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer Berlin / Heidelberg, 2000. (Cited on pages 37, 47 and 94.)

[CS06] Sanjit Chatterjee and Palash Sarkar. Generalization of the selective-id security model for HIBE protocols. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 241–256. Springer Berlin / Heidelberg, 2006. (Cited on page 71.)

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In GeorgeRobert Blakley and David Chaum, editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer Berlin Heidelberg, 1985. (Cited on page 14.)

[FLR+10] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 303–320. Springer Berlin / Heidelberg, 2010. (Cited on pages 5 and 6.)

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 79–79. Springer Berlin / Heidelberg, 1999. (Cited on page 71.)

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Berlin / Heidelberg, 1987. (Cited on pages 1, 5 and 14.)

[Gal05] David Galindo. Boneh-Franklin identity based encryption revisited. In Luís Caires, Giuseppe Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 102–102. Springer Berlin / Heidelberg, 2005. (Cited on page 37.)

[Gal06] David Galindo. A separation between selective and full-identity security notions for identity-based encryption. In Marina Gavrilova, Osvaldo Gervasi, Vipin Kumar, C. Tan, David Taniar, Antonio Laganá, Youngsong Mun, and Hyunseung Choo, editors, *Computational Science and Its Applications - ICCSA 2006*, volume 3982 of *Lecture Notes in Computer Science*, pages 318–326. Springer Berlin / Heidelberg, 2006. (Cited on page 71.)

[GBL08] Sanjam Garg, Raghav Bhaskar, and Satyanarayana V. Lokam. Improved bounds on security reductions for discrete log based signatures. In *Proceedings of the 28th Annual conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, pages 93–107, Berlin, Heidelberg, 2008. Springer-Verlag. (Cited on page 84.)

[GG09] David Galindo and Flavio Garcia. A Schnorr-like lightweight identity-based signature scheme. In Bart Preneel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, volume 5580 of *Lecture Notes in Computer Science*, pages 135–148.

Springer Berlin / Heidelberg, 2009. (Cited on pages 10, 11, 12, 30, 37, 38, 40, 41, 46, 52, 55, 57, 84, 93, 101 and 104.)

[GH05] David Galindo and Ichiro Hasuo. Security notions for identity based encryption. Cryptology ePrint Archive, Report 2005/253, 2005. (Cited on page 71.)

[GMR88] Shafi Goldwasser, Silvio Micali, and Ron Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988. (Cited on pages 6 and 7.)

[GQ90] Louis Guillou and Jean-Jacques Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO' 88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231. Springer Berlin / Heidelberg, 1990. (Cited on page 1.)

[Her05] Javier Herranz. Deterministic identity-based signatures for partial aggregation. *The Computer Journal*, 49(3):322–330, 2005. (Cited on page 1.)

[Hes03] Florian Hess. Efficient identity based signature schemes based on pairings. In Kaisa Nyberg and Howard Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 310–324. Springer Berlin / Heidelberg, 2003. (Cited on page 1.)

[HW09] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 654–670. Springer Berlin / Heidelberg, 2009. (Cited on pages 10 and 77.)

[HWS+06] Anne Marie Hegland, Eli Winjum, Pal Spilling, Chunming Rong, and Oivind Kure. Analysis of IBS for MANET security in emergency and rescue operations. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 2, pages 155–159, 2006. (Cited on page 1.)

[Kia07] Aggelos Kiayias. Cryptography: Primitives and protocols, 2007. (Cited on page 19.)

[KLS00] Stephen Kent, Charles Lynn, and Karen Seo. Secure border gateway protocol (S-BGP). *Selected Areas in Communications, IEEE Journal on*, 18(4):582–592, 2000. (Cited on page 1.)

[KM07] Neal Koblitz and Alfred J. Menezes. Another look at "provable security". *Journal of Cryptology*, 20:3–37, 2007. (Cited on pages 3 and 20.)

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA*. The Internet Society, 2000. (Cited on pages 10 and 72.)

[LBZ+10] Joseph K. Liu, Joonsang Baek, Jianying Zhou, Yanjiang Yang, and JunWen Wong. Efficient online/offline identity-based signature for wireless sensor network. *International Journal of Information Security*, 9(4):287–296, 2010. (Cited on page 1.)

[Moh11] Payman Mohassel. One-time signatures and chameleon hash functions. In Alex Biryukov, Guang Gong, and Douglas Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 302–319. Springer Berlin / Heidelberg, 2011. (Cited on page 72.)

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In ErnestF. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer Berlin Heidelberg, 1993. (Cited on page 14.)

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13:361–396, 2000. (Cited on pages 12, 14, 18 and 19.)

[PV05] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2005. (Cited on pages 4 and 84.)

[RS11] V. Radhakishan and S. Selvakumar. Prevention of man-in-the-middle attacks using id-based signatures. In *Second International Conference on Networking and Distributed Computing - ICNDC, 2011*, pages 165 –169. 2011. (Cited on page 38.)

[Sch90] 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 14.)

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991. 10.1007/BF00196725. (Cited on pages 6, 10, 12 and 38.)

[Seu12] Yannick Seurin. On the exact security of Schnorr-type signatures in the random oracle model. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 554–571. Springer Berlin / Heidelberg, 2012. (Cited on page 84.)

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In George Blakley and David Chaum, editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer Berlin / Heidelberg, 1985. (Cited on page 1.)

[Sho01] Victor Shoup. OAEP reconsidered. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 239–259. Springer Berlin / Heidelberg, 2001. (Cited on page 37.)

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004. (Cited on page 3.)

[ST01] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer Berlin / Heidelberg, 2001. (Cited on pages 10 and 77.)

[Teg03] Max Tegmark. Parallel universes. In John Barrow, Paul Davies, and Charles Harper Jr., editors, *Science and Ultimate Reality: Quantum Theory, Cosmology, and Complexity*, pages 459–491. Cambridge University Press, 2003. (Cited on page 21.)

[TWZL08] Chiu C. Tan, Haodong Wang, Sheng Zhong, and Qun Li. Body sensor network security: an identity-based cryptography approach. In *Proceedings of the first ACM conference on Wireless network security*, WiSec '08, pages 148–153, New York, NY, USA, 2008. ACM. (Cited on page 1.)

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 557–557. Springer Berlin / Heidelberg, 2005. (Cited on pages 6 and 71.)

[XW12] Min Xie and Libin Wang. One-round identity-based key exchange with perfect forward security. *Information Processing Letters*, 112(14–15):587 – 591, 2012. (Cited on page 38.)

Appendices

Appendix A

Galindo-Garcia IBS

A.1 The Fixed Security Argument

Let $\mathcal{A}$ be an adversary against GG-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 and 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.

A.1.1 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 $\alpha \xleftarrow{\text{U}} \mathbb{Z}_p$ and tries to extract the discrete logarithm α. The environment is simulated as shown below.

$\mathcal{B}_1.1$ $\mathcal{B}_1$ picks $\hat{i} \xleftarrow{\text{U}} \{1, \ldots, q_{\text{G}}\}$,[1] where q_{G} is the maximum number of queries that the adversary $\mathcal{A}$ makes to the G-oracle. Let $\hat{\texttt{id}}$ (the target identity) be the $\hat{i}^{\text{th}}$ distinct identity queried to the G-oracle. Next, $\mathcal{B}_1$ chooses $z \xleftarrow{\text{U}} \mathbb{Z}_p$ and sets $(\texttt{mpk}, \texttt{msk}) := ((\mathbb{G}, g, p, \text{G}, \text{H}, g^z), z)$, 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 $\mathfrak{L}_{\text{G}}$ and $\mathfrak{L}_{\text{H}}$ 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 $\texttt{id}$, $\mathcal{B}_1$ chooses $c, y \xleftarrow{\text{U}} \mathbb{Z}_p$, sets $R := g^{-zc} g^y$ and adds $\langle R, \texttt{id}, c \rangle$ to the table $\mathfrak{L}_{\text{H}}$. Then it returns the key (y, R) to $\mathcal{A}$.

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

[1] 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.

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

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

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

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

A.1.2 Reduction $\mathcal{B}_2$

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

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

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

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

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

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

$$\begin{aligned}\{b_0 = \log A_0 + (\log R + c_0\alpha)d_0, b_1 = \log A_0 + (\log R + c_0\alpha)d_1,\\ b_2 = \log A_2 + (\log R + c_2\alpha)d_2, b_3 = \log A_2 + (\log R + c_2\alpha)d_3\}\end{aligned} \tag{A.1}$$

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

$$\alpha = \frac{(b_0 - b_1)(d_2 - d_3) - (b_2 - b_3)(d_0 - d_1)}{(c_0 - c_1)(d_0 - d_1)(d_2 - d_3)}. \tag{A.2}$$

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

$$\begin{aligned}\{b_0 = \log A + (\log R_0 + c_0\alpha)d_0, b_1 = \log A + (\log R_0 + c_1\alpha)d_0,\\ b_2 = \log A + (\log R_2 + c_2\alpha)d_2, b_3 = \log A + (\log R_2 + c_3\alpha)d_2\}.\end{aligned} \tag{A.3}$$

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

$$\alpha = \frac{b_0 - b_1}{d_0(c_0 - c_1)}. \tag{A.4}$$

A.2 A Security Argument without Wrappers

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 §**2.4**.

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 $\mathcal{A}$.

A.2.1 Reduction $\mathcal{R}_1$

$\mathcal{R}_1$ uses the so-called "partitioning strategy", first used by Coron in the security argument of FDH [Cor00]. The basic idea is to divide the identity-space $\mathbb{I}$ into two disjoint sets, $\mathbb{I}_\varepsilon$ and $\mathbb{I}_s$, depending upon the outcome of a biased coin. 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 $z \xleftarrow{U} \mathbb{Z}_p$ as the master secret key. The master public key $\mathtt{mpk} := (\mathbb{G}, p, g, g^z, \mathrm{H}, \mathrm{G})$ 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 $\mathfrak{L}_{\mathrm{G}}$.

A.2.1.1 Handling the Queries

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

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

Here, $(R, \mathtt{id})$ is the query to the H-oracle and c is the corresponding output. Therefore, an oracle query $\mathrm{H}(R, \mathtt{id})$ is fresh if there exists no tuple $\langle R_i, r_i, \mathtt{id}_i, c_i, \beta_i \rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\mathtt{id}_i = \mathtt{id}) \wedge (R_i = R)$. 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*) '$\perp$', or *ii*) $\log_g R$, 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.

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

(i) H_1, Explicit query made by $\mathcal{A}$: In this case $\mathcal{R}_1$ returns $c \xleftarrow{U} \mathbb{Z}_p$ as the output. $\langle R, \perp, \mathtt{id}, c, \phi \rangle$ 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 $c \xleftarrow{U} \mathbb{Z}_p$ as the output. As $\mathcal{R}_1$ knows $r = \log_g R$, $\langle R, r, \mathtt{id}, c, 1 \rangle$ 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

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

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

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

$\mathrm{G}(\mathtt{id}, A, m)$: 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 $d \xleftarrow{\mathrm{U}} \mathbb{Z}_p$ as the output. $\langle \mathtt{id}, A, m, d\rangle$ 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 20. In the case of the implicit queries 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 $\mathtt{id}$ has an associated R. This is done by searching for tuples $\langle R_i, r_i, \mathtt{id}_i, c_i, \beta_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ with $(\mathtt{id}_i = \mathtt{id}) \wedge (r_i \neq \perp)$. If such a tuple exists, $\mathcal{R}_1$ 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 $\mathtt{id}$ 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 $\mathtt{id}$ 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.

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

(i) If $\beta_i = 0$, $\mathcal{R}_1$ *aborts* ($\mathsf{abort}_{1,1}$).

(ii) Otherwise, $\beta_i = 1$ and $\mathcal{R}_1$ returns $\mathtt{usk} := (r_i + zc_i, R_i)$ as the secret key for $\mathtt{id}$.

Otherwise

(iii) $\mathcal{R}_1$ chooses $r \xleftarrow{\mathrm{U}} \mathbb{Z}_p$, sets $R := g^r$ and asks the H-oracle for $c := \mathrm{H}(R, \mathtt{id})$. It returns $\mathtt{usk} := (r + zc, R)$ as the secret key.

Signature query. As in the extract query, $\mathcal{R}_1$ checks the identity for an associated R by searching tuples $\langle R_i, r_i, \mathtt{id}_i, c_i, \beta_i\rangle$ in $\mathfrak{L}_{\text{H}}$ with $(\mathtt{id}_i = \mathtt{id}) \wedge (r_i \neq \perp)$. 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,$\mathtt{id}$ is assigned to $\mathbb{I}_s$; else it is assigned to $\mathbb{I}_\varepsilon$. Identities assigned to $\mathbb{I}_s$ have the problem instance g^α embedded in the randomiser R. Although the private key cannot be calculated, an algebraic technique, similar to one adopted by Boneh-Boyen in [BB04a], coupled with random oracle programming enables us to give the signature. On the other hand, signature queries involving identities from $\mathbb{I}_\varepsilon$ are answered by first generating $\mathtt{usk}$ as in the extract query and then invoking $\mathcal{S}$. This is captured by the oracle $\mathcal{O}_s$ described below.

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

(i) If $\beta_i = 0$, $\mathcal{R}_1$ selects $s, d \xleftarrow{\text{U}} \mathbb{Z}_p$ and sets $A := g^s(g^\alpha)^{-r_i d}$. Then $(\mathtt{id}, A, m, d)$ is added to $\mathfrak{L}_{\text{G}}$ (*Deferred case G_2*)[2]. The signature returned is

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

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

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

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

(iii) If $\beta = 0$, $\mathcal{R}_1$ selects $c, d, s, r \xleftarrow{\text{U}} \mathbb{Z}_p$ and sets $R := (g^\alpha)^r$, $A := g^s(g^\alpha)^{-rd}$. Next, it adds $\langle (g^\alpha)^r, r, \mathtt{id}, c, 0\rangle$ to $\mathfrak{L}_{\text{H}}$ (*Deferred case H_3*) and $\langle \mathtt{id}, A, m, d\rangle$ to $\mathfrak{L}_{\text{G}}$ (*Deferred case G_2*).[3] The signature returned is

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

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

[2]If there exists a tuple $\langle \mathtt{id}_i, A_i, m_i, d_i\rangle$ in $\mathfrak{L}_{\text{G}}$ with $\mathtt{id}_i = \mathtt{id} \wedge A_i = A \wedge m_i = m$ but $d_i \neq d$ then $\text{G}(\mathtt{id}, A, m)$ 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.

[3]$\mathcal{R}_1$ chooses different randomisers if there is a collision as explained in **Footnote 2**.

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

$$\Pr\left[\neg\mathsf{abort}_{1,1}\right] = (1-\delta)^{q_\varepsilon}. \tag{A.5}$$

A.2.1.2 Solving the DLP

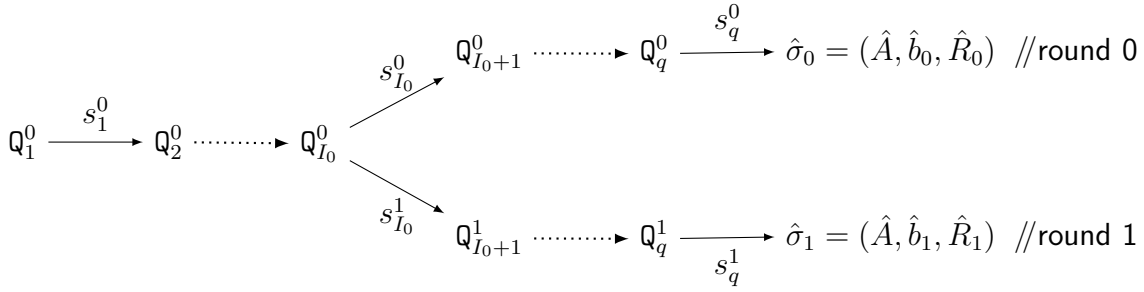

Figure A.1: Successful forking by $\mathcal{R}_1$. $\mathtt{Q}_{I_0}^0$ denotes the target G-query $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$.

$\mathcal{R}_1$ now uses the general forking algorithm $\mathcal{F}_\mathcal{W}$ (see §**2.4** for details on the working of $\mathcal{F}_\mathcal{W}$) to solve the DLP challenge. It invokes $\mathcal{F}_\mathcal{W}$ on the given DLP instance Δ,[4] with the G-oracle involved in the replay attack. If $\mathcal{F}_\mathcal{W}$ fails, so does $\mathcal{R}_1$ and it *aborts* ($\mathsf{abort}_{1,2}$). On the other hand, if $\mathcal{F}_\mathcal{W}$ is successful, it gets two valid forgeries

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

on $(\hat{\mathtt{id}}, \hat{m})$ (see **Figure A.1**). $\mathcal{R}_1$ now retrieves two tuples

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

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

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

from $\mathfrak{L}_{\text{H}}$. $\mathcal{R}_1$ *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 $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$ in the two rounds, *i.e.*, $d_0 = s^0_{I_0}$ and $d_1 = s^1_{I_0}$. Let $\hat{a} := \log_g \hat{A}$.

(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 $\hat{b}_1 = \hat{a} + (r_j\alpha + zc_j)d_1$.

$$\begin{aligned} \hat{b}_0 - \hat{b}_1 &= (r_i d_0 - r_j \alpha d_1) + z(c_i d_0 - c_j d_1), \\ \alpha &= \frac{z(c_i d_0 - c_j d_1) + r_i d_0 - (\hat{b}_0 - \hat{b}_1)}{r_j d_1}. \end{aligned} \tag{A.6}$$

(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 $\hat{b}_1 = \hat{a} + (r_j + zc_j)d_1$.

$$\begin{aligned} \hat{b}_1 - \hat{b}_0 &= (r_j d_1 - r_i \alpha d_0) + z(c_j d_1 - c_i d_0), \\ \alpha &= \frac{z(c_j d_1 - c_i d_0) + r_j d_1 - (\hat{b}_1 - \hat{b}_0)}{r_i d_0}. \end{aligned} \tag{A.7}$$

(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 $\hat{b}_1 = \hat{a} + (r_j\alpha + zc_j)d_1$.

$$\begin{aligned} \hat{b}_0 - \hat{b}_1 &= \alpha(r_i d_0 - r_j d_1) + z(c_i d_0 - c_j d_1), \\ \alpha &= \frac{(\hat{b}_0 - \hat{b}_1) - z(c_i d_0 - c_j d_1)}{(r_i d_0 - r_j d_1)}. \end{aligned} \tag{A.8}$$

Remark 21. The equations (A.6), (A.7) and (A.8) 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 $\mathsf{H}(\hat{R}_0, \hat{\mathtt{id}})$ before the query $\mathsf{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$ in round 0. Hence, the order in which $\mathcal{A}$ makes the aforementioned random oracle queries is not relevant.

Structure of R. The event E guarantees the existence of the tuple $\mathfrak{t}_i$ in $\mathfrak{L}_{\text{H}}$. As $\mathcal{A}$ cannot make an extract query on $\hat{\mathtt{id}}$, 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.*

$$\Pr\left[\mathsf{abort}_{1,3} \mid \neg\mathsf{abort}_{1,2}\right] = (1-\delta)^2. \tag{A.9}$$

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

$$\Pr\left[\neg\mathsf{abort}_{1,2}\right] = \mathit{gfrk}. \tag{A.10}$$

A.2.1.3 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 (A.5), (A.10) and (A.9), we have $\Pr\left[\neg\mathsf{abort}_{1,1}\right] = (1-\delta)^{q_\varepsilon}$, $\Pr\left[\neg\mathsf{abort}_{1,2}\right] = \mathit{gfrk}$ and $\Pr\left[\mathsf{abort}_{1,3} \mid \neg\mathsf{abort}_{1,2}\right] = (1-\delta)^2$. $\mathcal{F}_{\mathcal{W}}$ is successful during round 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 by acc_1. Thus

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

Applying the GF lemma (**Lemma 2.4**) with $|\mathbb{S}| = p$ and $q = q_{\mathrm{G}}$, we get

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

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

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

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

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

Assuming $p \gg 1$, (3.9) attains maximum value at the point $\delta = \left(1 - \sqrt{q_\varepsilon/(q_\varepsilon+1)}\right)$, at which

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

Here, $\exp$ is the base of natural logarithm.

Remark 22. The above reduction is tighter than the reduction $\mathcal{B}_1$ given by Galindo and Garcia [GG09]. This can be attributed to two reasons: *i*) $\mathcal{R}_1$ using the GF Algorithm $\mathcal{F}_{\mathcal{W}}$ instead of the MF Algorithm $\mathcal{M}_{\mathcal{W},1}$; and *ii*) $\mathcal{B}_1$ in [GG09] randomly chooses one of the identities involved in the G-oracle query as the target identity (refer to §**3.2.2.1**) 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 complexity. 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 simulating the adversary twice.

A.2.2 Reduction $\mathcal{R}_2$

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

Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. $\mathcal{R}_2$ sets $\texttt{mpk} := (\mathbb{G}, p, g, g^\alpha, \text{H}, \text{G})$ as the master public key and releases it to $\mathcal{A}$. Note that $\mathcal{R}_2$ does not know the master secret key $\texttt{msk}$, 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, $\mathfrak{L}_{\text{H}}$ and $\mathfrak{L}_{\text{G}}$.

A.2.2.1 Handling the Queries

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

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

Here, $(R, \texttt{id})$ 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 $\texttt{id}$ or '$\perp$' if the field is invalid. A random oracle query $\text{H}(R, \texttt{id})$ is fresh if there exists no tuple $\langle R_i, \texttt{id}_i, c_i, y_i \rangle$ in $\mathfrak{L}_{\text{H}}$ such that $(\texttt{id}_i = \texttt{id}) \wedge (R_i = R)$. 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.

$\text{H}(R, \texttt{id})$: The query may be

(i) H_1, Explicit query made by $\mathcal{A}$: In this case $\mathcal{R}_2$ returns $c \xleftarrow{\text{U}} \mathbb{Z}_p$ as the output. $\langle R, \texttt{id}, c, \perp \rangle$ is added to $\mathfrak{L}_{\text{H}}$.

(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. $\mathfrak{L}_{\text{G}}$ has the same structure as in $\mathcal{R}_1$ (See §**A.2.1.1**). The queries to G-oracle are handled as shown below.

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

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 $\texttt{usk}$ as in the extract query, followed by calling $\mathcal{S}$.

Extract query, $\mathcal{O}_{\varepsilon}(\texttt{id})$:

(i) If there exists a tuple $\langle R_i, \texttt{id}_i, c_i, y_i \rangle$ in $\mathfrak{L}_{\text{H}}$ such that $(\texttt{id}_i = \texttt{id}) \wedge (y_i \neq \perp)$, $\mathcal{R}_2$ returns $\texttt{usk} := (y_i, R_i)$ as the secret key.

(ii) Otherwise, $\mathcal{R}_2$ chooses $c, y \xleftarrow{\text{U}} \mathbb{Z}_p$, sets $R := (g^{\alpha})^{-c} g^{y}$ and adds $\langle R, \texttt{id}, c, y \rangle$ to $\mathfrak{L}_{\text{H}}$(*Deferred case* H_2). It returns $\texttt{usk} := (y, R)$ as the secret key.

Signature query, $\mathcal{O}_{s}(\texttt{id}, m)$:

(i) If there exists a tuple $\langle R_i, \texttt{id}_i, c_i, y_i \rangle$ in $\mathfrak{L}_{\text{H}}$ such that $(\texttt{id}_i = \texttt{id}) \wedge (y_i \neq \perp)$, then $\texttt{usk} = (y_i, R_i)$. $\mathcal{R}_2$ now uses the knowledge of $\texttt{usk}$ to invoke $\mathcal{S}$ and returns the signature.

(ii) Otherwise, $\mathcal{R}_2$ generates $\texttt{usk}$ as in **Extract**(ii) and invokes $\mathcal{S}$ to return the signature.

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

A.2.2.2 Solving the DLP

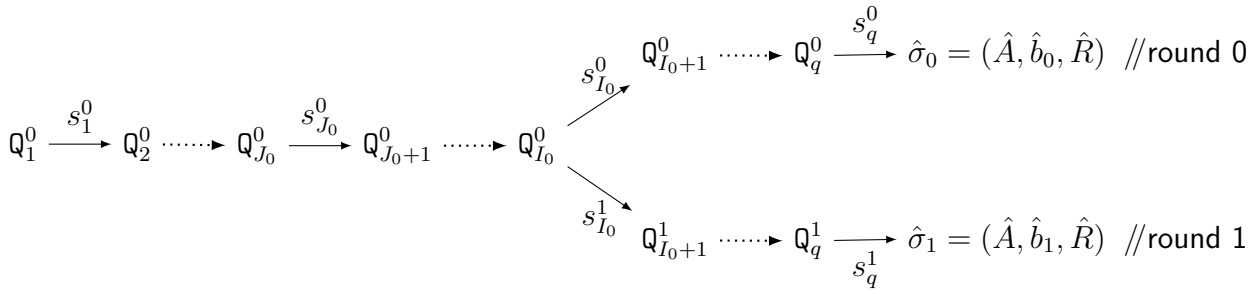

Figure A.2: Successful forking by $\mathcal{R}_2$. $\mathtt{Q}^0_{J_0}$ denotes the target G-query G($\hat{\texttt{id}}, \hat{A}, \hat{m}$) while $\mathtt{Q}^0_{I_0}$ denotes the target H-query H($\hat{R}, \hat{\texttt{id}}$).

$\mathcal{R}_2$ now uses the multiple-forking algorithm $\mathcal{M}_{\mathcal{W},1}$ (see §**2.5** for details on the working of $\mathcal{M}_{\mathcal{W},n}$) to solve the DLP challenge. It invokes $\mathcal{M}_{\mathcal{W},1}$ on $\texttt{mpk}$, with both H and G-oracle involved in the replay attack. If $\mathcal{M}_{\mathcal{W},1}$ fails, so does $\mathcal{R}_2$ and it *aborts* ($\texttt{abort}_{2,1}$). On the other

hand, if $\mathcal{M}_{\mathcal{W},1}$ is successful, $\mathcal{R}_2$ gets two valid, non-trivial forgeries $\{\hat{\sigma}_0, \hat{\sigma}_1\}$ on $(\hat{\mathtt{id}}, \hat{m})$ with $\hat{\sigma}_i := (\hat{A}, \hat{b}_i, \hat{R})$ for each $i = 0, 1$ and

$$\{\hat{b}_0 = \hat{a} + (\hat{r} + \alpha c_0)\hat{d}, \hat{b}_1 = \hat{a} + (\hat{r} + \alpha c_1)\hat{d}\} \tag{A.12}$$

where $\hat{a} := \log_g \hat{A}$ and $\hat{r} := \log_g \hat{R}$. Note that c_0 and c_1 represent the value of $\mathsf{H}(\hat{R}, \hat{\mathtt{id}})$ in the two rounds, *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 query $\mathtt{Q}^0_{J_0} : \mathsf{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$, before the H-oracle query $\mathtt{Q}^0_{I_0} : \mathsf{H}(\hat{R}, \hat{\mathtt{id}})$. Finally it outputs the solution to the DLP instance,

$$\alpha = \frac{\hat{b}_0 - \hat{b}_1}{\hat{d}(c_0 - c_1)}. \tag{A.13}$$

Structure of the forgeries. Now we justify the structure of the component b of the forgeries given in (A.12). 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 $\mathtt{usk} = (y, R)$, 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 $\neg\mathsf{E}$ guarantees that $\mathcal{A}$ does not forge using an R which was given as part of the signature query on id and hence, for the forgery to be valid b will necessarily be of the form:

$$b = a + (r + \alpha c)d. \tag{A.14}$$

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

$$\Pr\left[\neg\mathsf{abort}_{2,1}\right] = \mathit{mfrk}. \tag{A.15}$$

A.2.2.3 Analysis

The only abort involved in $\mathcal{R}_2$ is $\mathsf{abort}_{2,1}$, which occurs when $\mathcal{M}_{\mathcal{W},1}$ fails. Therefore $\mathcal{R}_2$ is successful if $\mathcal{M}_{\mathcal{W},1}$ is and from (A.15) we have

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

We denote the probability with which $\mathcal{M}_{\mathcal{W},1}$ is successful during round 0 as acc_2. Since there is no abort involved during query phase, $\mathcal{M}_{\mathcal{W},1}$ is successful during round 0 if $\mathcal{A}$ produces a valid forgery, *i.e.* $\mathit{acc}_2 = \epsilon$. Applying the MF lemma (**Lemma 6**) with $n := 1$, $q := q_{\mathrm{H}} + q_{\mathrm{G}}$ and $|\mathbb{S}| = p$, we get

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

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

A.2.3 Reduction $\mathcal{R}_3$

The approach used in $\mathcal{R}_3$ is the same as in the reduction $\mathcal{B}_2$ in [GG09]. Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. $\mathcal{R}_3$ sets $\mathtt{mpk} := (\mathbb{G}, p, g, g^\alpha, \mathrm{H}, \mathrm{G})$ as the master public key and releases it to $\mathcal{A}$. As in $\mathcal{R}_2$, $\mathcal{R}_3$ does not know the master secret $\mathtt{msk}$, 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 $\mathfrak{L}_{\mathrm{G}}$.

A.2.3.1 Handling the Queries

The queries are handled in the same way as in $\mathcal{R}_2$. So we refer to §**A.2.2.1** for details.

A.2.3.2 Solving the DLP

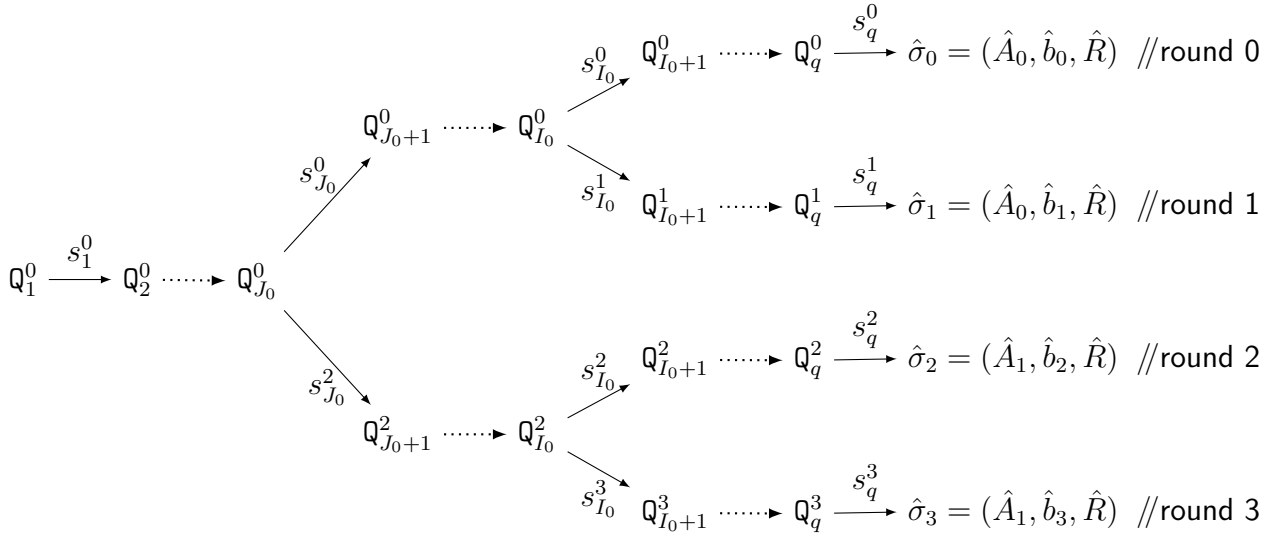

Figure A.3: Successful multiple forkings by $\mathcal{R}_3$. $\mathtt{Q}_{J_0}^0$ denotes the target H-query $\mathrm{H}(\hat{R}, \hat{\mathtt{id}})$; $\mathtt{Q}_{I_0}^0$ [resp. $\mathtt{Q}_{I_0}^2$] denotes the target G-query $\mathrm{G}(\hat{\mathtt{id}}, \hat{A}_0, \hat{m})$ [resp. $\mathrm{G}(\hat{\mathtt{id}}, \hat{A}_1, \hat{m})$].

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

$$\begin{aligned} &\hat{\sigma}_0 = (\hat{A}_0, \hat{b}_0, \hat{R}) \ , \ \hat{\sigma}_1 = (\hat{A}_0, \hat{b}_1, \hat{R}), \\ &\hat{\sigma}_2 = (\hat{A}_1, \hat{b}_2, \hat{R}) \ \text{ and } \ \hat{\sigma}_3 = (\hat{A}_1, \hat{b}_3, \hat{R}) \end{aligned} \tag{A.16}$$

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

$$\begin{aligned} \hat{b}_0 &= \hat{a}_0 + (\hat{r} + \alpha c_0)d_0 \ , \ \hat{b}_1 = \hat{a}_0 + (\hat{r} + \alpha c_0)d_1, \\ \hat{b}_2 &= \hat{a}_1 + (\hat{r} + \alpha c_1)d_2 \ \text{ and } \ \hat{b}_3 = \hat{a}_1 + (\hat{r} + \alpha c_1)d_3, \end{aligned} \tag{A.17}$$

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

$$\alpha = \frac{(\hat{b}_0 - \hat{b}_1)(d_2 - d_3) - (\hat{b}_2 - \hat{b}_3)(d_0 - d_1)}{(c_0 - c_1)(d_0 - d_1)(d_2 - d_3)}. \tag{A.18}$$

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

$$\Pr\left[\neg \mathsf{abort}_{3,1}\right] = \mathit{mfrk}. \tag{A.19}$$

A.2.3.3 Analysis

As in $\mathcal{R}_2$, the only abort involved in $\mathcal{R}_3$ is $\mathsf{abort}_{3,1}$, which occurs when $\mathcal{M}_{\mathcal{W},3}$ fails. Therefore $\mathcal{R}_3$ is successful if $\mathcal{M}_{\mathcal{W},3}$ is and from (A.19) we have

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

We denote the probability with which $\mathcal{M}_{\mathcal{W},3}$ is successful during the first round as acc_3. Since there is no abort involved during query phase, $\mathcal{M}_{\mathcal{W},3}$ is successful during the first round if $\mathcal{A}$ produces a valid forgery, *i.e.* $\mathit{acc}_3 = \epsilon$. Applying the MF lemma (**Lemma 6**) with $n = 3$, $q = q_{\mathsf{H}} + q_{\mathsf{G}}$ and $|\mathbb{S}| = p$, we have

$$\begin{aligned} \epsilon_3 = \mathit{mfrk} &\geq \mathit{acc}_3 \cdot \left(\frac{\mathit{acc}_3^3}{(q_{\mathsf{H}} + q_{\mathsf{G}})^6} - \frac{3}{p} \right) \\ &\geq \epsilon \left(\frac{\epsilon^4}{(q_{\mathsf{H}} + q_{\mathsf{G}})^6} - \frac{3}{p} \right). \end{aligned}$$

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

A.3 Reduction $\mathcal{R}'_1$

Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. The reduction involves invoking the GF Algorithm on the wrapper $\mathcal{Y}$ as shown in **Algorithm 7**. As a result, it obtains a set of two congruences in two unknowns and solves for α.

Algorithm 7 Reduction $\mathcal{R}'_1(\Delta)$

Select $z \xleftarrow{\text{U}} \mathbb{Z}_p^*$ as the $\texttt{msk}$ and set $\texttt{mpk} := (\mathbb{G}, g, p, g^z)$.
$(\mathsf{b}, \sigma_0, \sigma_1) \xleftarrow{\$} \mathcal{F}_{\mathcal{Y}}((\texttt{mpk}, \texttt{msk}), g^\alpha)$
if $(\mathsf{b} = 0)$ **then return** $\perp$ //$\mathsf{abort}_{1,2}$
Parse σ_i as $(\hat{b}_i, c_i, r_i, \beta_i, d_i)$.
Let $\beta := \beta_0$, $c := c_0$ and $r := r_0$
if $\beta = 0$ **then return** $((\hat{b}_0 - \hat{b}_1) - zc(d_0 - d_1))/r(d_0 - d_1)$
else return $\perp$ //$\mathsf{abort}_{1,3}$
end if

The Wrapper

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

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

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

Here, $(\texttt{id}, R)$ is the query to the H-oracle and c is the corresponding output. Therefore, a query $\text{H}(\texttt{id}, R)$ is fresh if there exists no tuple $\langle \texttt{id}_i, R_i, c_i, r_i, \beta_i \rangle$ in $\mathfrak{L}_{\text{H}}$ such that $(\texttt{id}_i = \texttt{id}) \wedge (R_i = R)$. If such a tuple exists, then the oracle has to return the corresponding c_i as the output. The role of the r and β-field is the same as in $\mathcal{R}_1$. We now explain how the fresh H-oracle queries are handled. The query may be

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

(ii) H_2, Explicit query made by $\mathcal{Y}$: As in the previous case, $\mathcal{Y}$ returns $c \xleftarrow{\text{U}} \mathbb{Z}_p$ as the output. As $\mathcal{Y}$ knows $r = \log_g R$, $\langle \texttt{id}, R, c, r, 1 \rangle$ is added to $\mathfrak{L}_{\text{H}}$.

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

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

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

Here, (m, A, c) is the query to the G-oracle and d is the corresponding output. The index of the query is stored in the ℓ-field. Therefore, a random oracle query $\text{G}(m, A, c)$ is fresh if there exists no tuple $\langle m_i, A_i, c_i, d_i, \ell_i \rangle$, in $\mathfrak{L}_{\text{G}}$ such that $(m_i = m) \wedge (A_i = A) \wedge (c_i = c)$.

If such a tuple exists, then the oracle has to return the corresponding d_i as the output. We now explain how the fresh G-oracle queries are handled. The query may be

(i) G_1, Explicit query made by either $\mathcal{A}$ or $\mathcal{Y}$: In this case $\mathcal{Y}$ returns $d := s_\ell$ as the output. $\langle m, A, c, d, \ell\rangle$ is added to $\mathfrak{L}_{\mathrm{G}}$ and ℓ is incremented by one.

(ii) G_2, Implicit query by $\mathcal{Y}$ in order to answer a signature query made by $\mathcal{A}$: See steps (i) and (iii) of **Signature query** on how to program the random oracle in this situation.

(c) **Extract query,** $\mathcal{O}_{\varepsilon}(\mathtt{id})$:

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

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

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

Otherwise

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

(d) **Signature query,** $\mathcal{O}_s(\mathtt{id}, m)$:

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

(i) If $\beta_i = 0$, $\mathcal{Y}$ selects $s, \xleftarrow{\mathrm{U}} \mathbb{Z}_p$ and sets $d := s_\ell$, $A := g^s(g^\alpha)^{-r_i d}$. It then adds $\langle m, A, c, d, \ell\rangle$ to $\mathfrak{L}_{\mathrm{G}}$ (deferred case G_2)[5] and increments ℓ by one. The signature returned is $\sigma := (A, s + zcd, R_i)$.

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

Otherwise, $\mathcal{Y}$ 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 $\beta = 0$, $\mathcal{Y}$ selects $c, s, r \xleftarrow{\mathrm{U}} \mathbb{Z}_p$ and sets $d := s_\ell$, $R := (g^\alpha)^r$, $A := g^s(g^\alpha)^{-rd}$. Next, it adds $\langle \mathtt{id}, (g^\alpha)^r, c, r, 0\rangle$ to $\mathfrak{L}_{\mathrm{H}}$ (deferred case H_3), $\langle m, A, c, d, \ell\rangle$ to $\mathfrak{L}_{\mathrm{G}}$ (deferred case G_2) and increments ℓ by one.[6] The signature returned is $\sigma := (s + zc_i d, R, A)$.

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

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

[6] $\mathcal{Y}$ chooses different randomisers if there is a collision as explained in **Footnote 5**.

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

Correctness of the discrete-log. In the event of successful forking, $\mathcal{R}_1'$ obtains two (related) sets of side-outputs σ_0 and σ_1, where σ_i (for $i = 1, 2$) is of the form $(\hat{b}_i, c_i, r_i, \beta_i, d_i)$. Due to dependency, the adversary is forced to follow the order H $<$ G. As a result, $\beta_0 = \beta_1$ (denoted by β), $c_0 = c_1$ (denoted by c) and $r_0 = r_1$ (denoted by r). In addition, let $\hat{a}$ denote $\log_g \hat{A}_1 = \log_g \hat{A}_0$. $\mathcal{Y}$ aborts in the event that $\beta = 1$ ($\mathsf{abort}_{1,3}$). In the case $\beta = 0$, $\mathcal{R}_1'$ ends up with a system of two congruences $\{\hat{b}_0 = \hat{a} + (r\alpha + zc)d_0, \hat{b}_1 = \hat{a} + (r\alpha + zc)d_1\}$ in two unknowns $\{\hat{a}, \alpha\}$. α can be solved for as shown below.

$$\alpha = \frac{(\hat{b}_0 - \hat{b}_1) - zc(d_0 - d_1)}{r(d_0 - d_1)} \tag{A.20}$$

Notice that (A.20) is precisely what $\mathcal{R}_1'$ outputs in **Algorithm 7**.

A.3.1 Analysis

The probability analysis is governed by the three events $\mathsf{abort}_{1,1}$, $\mathsf{abort}_{1,2}$ and $\mathsf{abort}_{1,3}$. The accepting probability of $\mathcal{Y}$ is the same as for $\mathcal{R}_1$ and, as a result, so is the probability of $\mathsf{abort}_{1,2}$. The probability of event $\mathsf{abort}_{1,3}$, on the other hand, is the same as that with which $(\beta = 1)$, *i.e.* $\Pr\left[\mathsf{abort}_{1,3} \mid \neg\mathsf{abort}_{1,2}\right] = (1 - \delta)$. On putting it all together, we get

$$\begin{aligned}
\epsilon_1' &= \Pr\left[\neg\mathsf{abort}_{1,3} \wedge \neg\mathsf{abort}_{1,2}\right] \\
&\geq (1 - (1 - \delta)) \cdot (1 - \delta)^{q_\varepsilon}\epsilon \cdot \left(\frac{(1 - \delta)^{q_\varepsilon}\epsilon}{q_{\text{G}}} - \frac{1}{p}\right) \\
&= \delta \cdot (1 - \delta)^{q_\varepsilon}\epsilon \cdot \left(\frac{(1 - \delta)^{q_\varepsilon}\epsilon}{q_{\text{G}}} - \frac{1}{p}\right)
\end{aligned} \tag{A.21}$$

Assuming $p \gg 1$, (A.21) attains maximum value at the point $\delta = 1/(1 + 2q_\varepsilon)$, at which

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

Here, $\exp$ is the base of natural logarithm.