A Closer Look at Multiple Forking: Leveraging (*In*)dependence for a Tighter Bound

Sanjit Chatterjee and Chethan Kamath,

Indian Institute of Science, Bangalore.

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

December 12, 2013

Abstract

Boldyreva *et al.* introduced the notion of multiple forking (MF) as an extension of (general) forking to accommodate nested oracle replay attacks. The primary objective of a (multiple) forking algorithm is to separate out the oracle replay attack from the actual simulation of protocol to the adversary, and this is achieved through the intermediary of a so-called wrapper algorithm. Multiple forking has turned out to be a useful tool in the security argument of several cryptographic protocols. However, a reduction employing the MF Algorithm incurs a significant degradation of $\mathrm{O}\left(q^{2n}\right)$, where q denotes the upper bound on the underlying random oracle calls and n, the number of forkings.

In this work we take a closer look at the reasons for the degradation with a tighter security bound in mind. We nail down the exact set of conditions for the success of the MF Algorithm. A careful analysis of the protocols (and corresponding security argument) employing multiple forking allow us to relax the overly restrictive conditions of the original MF Algorithm. To achieve this, we club two consecutive invocations of the underlying wrapper into a single logical unit of wrapper $\mathcal{Z}$. We then use $\mathcal{Z}$ to formulate the notion of "dependence" and "independence" among different rounds of the wrapper in the MF Algorithm. The (*in*)dependence conditions lead to a general framework for multiple forking and significantly better bound for the MF Algorithm. Leveraging (*in*)dependence to the full *reduces* the degradation from $\mathrm{O}\left(q^{2n}\right)$ to $\mathrm{O}\left(q^{n}\right)$.

Finally, we study the effect of these observations on the security of the existing schemes. We conclude that by careful design of the protocol (and the wrapper in the security reduction) it is possible to harness our observations to the full extent.

Keywords: Oracle Replay Attack, Forking Lemma, Multiple-Forking Lemma, Provable Security, Tightness.

Contents

1 Introduction

The machinery of oracle replay attack of Pointcheval and Stern [PS00] plays a pivotal role in the security argument of a large class[1] of signature schemes [ElG85, Sch91, Oka93]. In the *elementary* version of replay attack[2], the simulator runs the adversary twice on related inputs in order to solve the underlying hard problem. The probability of success of the replay attack is then bounded by the Forking Lemma [PS00]. Bellare and Neven [BN06], however, 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 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 [BR93] altogether. 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.

Multiple forking. The concept of forking was further generalised by Boldyreva *et al.* leading to the Multiple Forking (MF) Algorithm [BPW12]. The immediate motivation behind this new abstraction was to argue the security of a proxy signature scheme that uses *more* than one hash function (modelled as random oracles). The MF Algorithm allows one to mount the so-called *nested* replay attacks by rewinding the adversary several times on related inputs. In particular, a nested oracle replay attack involves multiple *augmented* forkings[3]. The MF Algorithm retains the modularity advantage of GF Algorithm and has been applied in several other security arguments [GG09, CMW12, CKK13] in a more-or-less black-box fashion. Note that the generalisation of forking due to Bagherzhandi *et al.* [BCJ08] is different from that in [BPW12].

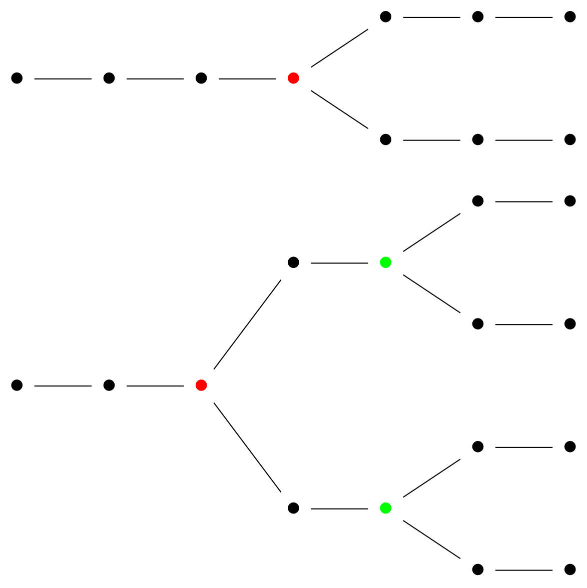

Figure 1: Elementary forking (top) vs. multiple augmented forking with three forkings (bottom): Augmented forking involves two random oracles and, hence, to be successful, the additional target index (in green) also should match.

The modularity of the (General/Multiple) Forking Lemma allows one to abstract out the

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

[2]We will use the terms forking and oracle replay attack interchangeably.

[3]We clearly distinguish between augmented forking from elementary forking: the former involves replay of two random oracles whereas the latter, only one random oracle. Henceforth, whenever we refer to multiple forkings, we are implicitly referring to multiple "augmented" forking as multiple elementary forking is not quite interesting.

probabilistic 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/MF 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 consists of invoking the appropriate 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/MF 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.

Role of the wrapper. Let's now take a simplistic[4] look at how the GF Algorithm (given in **Appendix B**), together with the wrapper $\mathcal{Y}$, is used to launch the elementary replay attack. The input to $\mathcal{Y}$ consists of some external randomness $(s_1, \ldots, s_q)$ and the internal coins (ρ) for adversary; the output, whereas, consists of an index I.

Consider the first invocation of $\mathcal{Y}$ (on $s_1, \ldots, s_q; \rho$) within the GF Algorithm: $\mathcal{Y}$ simulates the protocol environment to the actual adversary $\mathcal{A}$ having access to ρ. $\mathcal{Y}$ responds to the random oracle queries of $\mathcal{A}$ using $s_1, \ldots, s_q$. At the end of the simulation, $\mathcal{Y}$ outputs an index I that refers to the *target query*[5]; if the adversary did not make the target query, I is set to 0 (indicating failure). Next, the GF Algorithm invokes $\mathcal{Y}$ on an input that is *related* to the first invocation $(s_1, \ldots, s_{I-1}, s'_I, \ldots, s'_q; \rho)$. 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., $I' = I$. 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.

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 augmented 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 calls 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.) As the name suggests, the MF Algorithm allows the possibility of more than one forking–this adds another level of complexity in the overall structure.

The cost of multiple forking. The Forking 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 $\mathrm{O}(q)$ (and evidence suggests that the bound is more or less optimal [Seu12]). As for multiple forking, the cost according to the MF Lemma (see **Lemma 8** in **Appendix C**), is roughly $\mathrm{O}\left(q^{2n}\right)$, where q is the sum of the upper bound on the queries to the random oracles involved and n is, loosely speaking, the number of forking (so the wrapper is called $n+1$ times).[6] Consequently, the cost

[4]For the time being, let's not consider the input string x or the side-output σ.

[5]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*.

[6] To be precise, if *acc* denotes the probability with which the wrapper is successful for one round, then the cost of general forking is $\mathrm{O}(q/acc)$ and that of multiple forking is $\mathrm{O}\left(q^{2n}/acc^n\right)$. But we ignore the *acc* factor in the discussion.

of single augmented forking is O $\left(q^{2}\right)$. As we see, the bound is quite loose and naturally, the protocols employing the MF Lemma for their security suffer from this bound. This is indicated in the concluding statement of [BPW12] where the authors mention that the concrete security bound of their scheme is *not particularly tight* and they leave the possibility of a *tighter* reduction as an open question. In fact, for all security reductions that employ the MF Lemma [BPW12, GG09, CMW12, CKK13], the degradation primarily stems from the loose lower bound of the lemma. Hence, it's important to ask whether and to what extent it is possible to improve upon this bound. Investigating these questions in the concrete context of cryptographic protocols forms the focal point of this work.

1.1 Our Contribution

Multiple forking is a further generalisation of general forking and once formulated, this abstract notion has been applied in the analysis of concrete cryptographic protocols essentially in a black-box way. Here we undertake a journey from the concrete to the abstract. This has two (complementary) parts. We take a critical look at the various conditions that decide the success of the MF Algorithm as well as revisit the concrete security arguments employing the MF Lemma. This study allows us to come up with a better abstraction of multiple forking.

By the very nature of the rewinding mechanism, the wrapper algorithm is always invoked in pairs. Based on this simple observation, we club two consecutive invocations of the wrapper into a single *logical unit* of wrapper $\mathcal{Z}$. Intuitively, wrapper $\mathcal{Z}$ can be viewed as one invocation of the GF Algorithm of [BN06]. Utilizing the extra level of modularity provided by $\mathcal{Z}$, we nail down the exact conditions for the success of multiple forking. This, coupled with our investigation of the actual security arguments employing MF Lemma allow us to formulate two crucial observations: called, respectively, the "independence" condition (O_{I}) and the "dependence" condition (O_{D}). O_{I} is formulated by a careful abstraction of the exact requirements in the security reductions. In short, it allows the relaxation of success condition related to the index I across the logical wrapper $\mathcal{Z}$. O_{D}, on the other hand, has its root in the notion of hash function dependence, which is actually observed in the concrete protocols.

A general framework. Based on the above observations, we propose a general framework for the application of the MF Algorithm and a General Multiple Forking Lemma. Our framework captures the original MF as well as the observations O_{I} and O_{D} (separately and together) leading to four different versions of the lemma. We prove the new versions of the lemma. Naturally, the analysis becomes more involved as we incorporate the above two conditions–the most involved case occurs when one captures both the observations O_{I} and O_{D} ($\mathsf{O}_{\{\mathrm{I,D}\}}$, in short). We draw from existing techniques [BN06, BPW12] as well as introduce some new optimisations to *significantly* improve upon the existing bound. To be exact, the degradation *reduces* from O $\left(q^{2n}\right)$ to O $\left(q^{n}\right)$ when both the observations of are incorporated in the analysis (see **Table 2** for a summary). Thus, informally, we have:

Main Result (Lemma 2). *By carefully designing the protocol, multiple forking can be carried out with a success probability of* $\Omega(\epsilon^{n+1}/q^{n})$.

Corollary. *A single augmented forking can be launched as efficiently as an elementary forking (see **Remark 4**).*

Recall that the GF Algorithm of [BN06] captures a single elementary forking with a degradation of O (q). Our MF Algorithm with $\mathsf{O}_{\{\mathrm{I,D}\}}$, according to the above corollary, seems to be the best possible generalisation of general forking. We give an informal argument regarding the optimality of our result in §**3.2**.

Effect on protocols. Finally, we study the applicability of the observations on the security of the existing schemes that employ multiple forking [BPW12, GG09, CMW12]. We conclude that by careful design of the protocol (or, for that matter, by easily modifying existing protocols) it is possible to harness both the observations to the full extent. Thus, we end up with tighter security arguments for the protocols and under the same (hardness) assumptions (see **Table 1**). Plus, the notion of random-oracle dependence may be of independent interest as it is handy in certain situations other than multiple forking [YZ13, YADV$^+$12].

On a related note, we also emphasise on the importance of the proper design of the wrapper algorithm taking into account the intricacies of the actual security argument.

Protocol	Degradation	
	Before	After
[BPW12]	$\mathrm{O}\left(q^{10}/\epsilon^5\right)$	$\mathrm{O}\left(q^{5}/\epsilon^5\right)$
[GG09]	$\mathrm{O}\left(q^{6}/\epsilon^3\right)$	$\mathrm{O}\left(q^{3}/\epsilon^3\right)$
[CMW12]	$\mathrm{O}\left(q^{10}/\epsilon^5\right)$	$\mathrm{O}\left(q^{5}/\epsilon^5\right)$

Table 1: Comparison of the security degradation for protocols before and after our result. q denotes the upper bound on the respective hash oracle queries; ϵ is the advantage that the adversary has in the respective protocols.

Organisation of the paper. We take a closer look at the MF Algorithm in §**2**. In §**3**, we give an improved analysis of the MF Algorithm, while in §**4**, we apply the improvements to some of the existing schemes. Finally, we end with the concluding remarks in §**5**. As for the appendix, we begin with the GF Algorithm and the original MF Algorithm in **Appendix B** and **C** respectively. We give the deferred analyses of MF Algorithm with $\mathrm{O_I}$ and $\mathrm{O_D}$ (separately) in **Appendix D**. The construction of the schemes referred in the paper is given in **Appendix E**. We conclude with a section dedicated to the detailed security argument of the (modified) GG-IBS. Refer to **Appendix A** for notations used in the paper.

2 Multiple-Forking: A Closer Look

We begin with a critical look at the MF Algorithm and the associated lemma. A *slightly revised* version of the algorithm is given below (the original algorithm of [BPW12] is reproduced in **Appendix C**).

The Multiple-Forking Algorithm. Fix $q \in \mathbb{Z}^+$ and a set $\mathbb{S}$ such that $|\mathbb{S}| \geq 2$. Let $\mathcal{Y}$ be a randomised algorithm that on input a string x and elements $s_1, \ldots, s_q \in \mathbb{S}$ 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 MF Algorithm $\mathcal{M}_{\mathcal{Y},n}$ associated to $\mathcal{Y}$ and n is defined in **Algorithm 1**.

Algorithm 1 $\mathcal{M}_{\mathcal{Y},n}(x)$

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

$\{s_1^0, \ldots, s_q^0\} \overset{U}{\leftarrow} \mathbb{S}$;
$(I_0, J_0, \sigma_0) \leftarrow \mathcal{Y}(x, s_1^0, \ldots, s_q^0; \rho)$   //round 0
$\{s_{I_0}^1, \ldots, s_q^1\} \overset{U}{\leftarrow} \mathbb{S}$;
$(I_1, J_1, \sigma_1) \leftarrow \mathcal{Y}(x, s_1^0, \ldots, s_{I_0-1}^1, s_{I_0}^1, \ldots, s_q^1; \rho)$   //round 1

if $((I_0 = 0) \vee (J_0 = 0))$ then return $(0, \perp)$   //Condition $\neg$B
if $\left((I_1, J_1) \neq (I_0, J_0) \vee (s_{I_0}^1 = s_{I_0}^0)\right)$ then return $(0, \perp)$   //Condition $\neg \mathsf{C}_0$

$k := 2$
while $(k < n)$ do
    $\{s_{J_0}^k, \ldots, s_q^k\} \overset{U}{\leftarrow} \mathbb{S}$;
    $(I_k, J_k, \sigma_k) \leftarrow \mathcal{Y}(x, s_1^0, \ldots, s_{J_0-1}^0, s_{J_0}^k, \ldots, s_q^k; \rho)$   //round k
    $\{s_{I_k}^{k+1}, \ldots, s_q^{k+1}\} \overset{U}{\leftarrow} \mathbb{S}$;
    $(I_{k+1}, J_{k+1}, \sigma_{k+1}) \leftarrow \mathcal{Y}(x, s_1^0, \ldots, s_{J_0-1}^0, s_{J_0}^k, \ldots, s_{I_k-1}^k, s_{I_k}^{k+1}, \ldots, s_q^{k+1}; \rho)$   //round k+1

    if $\left((I_{k+1}, J_{k+1}) \neq (I_k, J_k) \vee (s_{I_k}^{k+1} = s_{I_k}^k)\right)$ then return $(0, \perp)$   //Condition $\neg \mathsf{C}_k$
    if $\left((I_k, J_k) \neq (I_0, J_0) \vee \boxed{\vee_{\ell:=0,2,\ldots,k-2} (s_{J_0}^k = s_{J_0}^\ell)}\right)$ then return $(0, \perp)$   //Condition $\neg \mathsf{D}_k$

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

Note that we have introduced some conceptual changes in the MF Algorithm. Before delving into the structure of the algorithm and ways to improve upon the bound on its probability of success, we briefly comment on a *subtle* (but minor in effect) logical flaw that we have fixed in the original version of [BPW12].

Remark 1. For the convenience of the readers we have boxed the modification in **Algorithm 1**. The original algorithm checked for $(s_{J_0}^k = s_{J_0}^{k-1})$. However, this is not *sufficient* for some of the reduction algorithms that use the original $\mathcal{M}_{\mathcal{Y},n}$. For example, consider the application of $\mathcal{M}_{\mathcal{Y},5}$ in the security argument of the CMW protocol. At the end of the simulation, the MF Algorithm outputs six "cheating" transcripts $\{(v_1^i, c_1^i, s_1^i), (v_1^i, c_2^i, s_2^i)\}$, for $i := 1, 2, 3$, and the reduction, subsequently, finds the solution to the DLP by computing

$$\left(\frac{z_1 w_1 - z_2 w_2}{z_1 - z_2} - \frac{z_1 w_1 - z_3 w_3}{z_1 - z_3}\right) \Big/ \left(\frac{w_1 - w_3}{z_1 - z_3} - \frac{w_1 - w_2}{z_1 - z_2}\right) \mod p \tag{1}$$

where $w_i = (s_1^i - s_2^i)/(c_2^i - c_1^i)$ and where $z_1 := s_{J_0}^0$, $z_2 := s_{J_0}^2$ and $z_3 := s_{J_0}^4$ in the original $\mathcal{M}_{\mathcal{Y},5}$. Thus, for a correct solution of the discrete-log problem (DLP), the reduction requires to compute $(z_1 - z_2)^{-1}$ and $(z_1 - z_3)^{-1}$. [CMW12] asserts "[a]ccording to the probing strategy z_1, z_2, z_3 are all distinct". However, as per the original proposition, $\mathcal{M}_{\mathcal{Y},5}$ of [BPW12] will *only* ensure that $z_2 \neq z_1$ and $z_3 \neq z_2$ but *not necessarily* $z_1 \neq z_3$. Hence, the probing strategy *does not* guarantee that all the z_is are *distinct* and the reduction may fail even though the MF Algorithm returns success. Clearly, the fault lies in the condition D_k within the **while** loop of original MF Algorithm, which checks for equality *only* with the round just preceding it and comes into the picture for $\mathcal{M}_{\mathcal{Y},n}$ for $n \geq 5$. A simple fix is to introduce pairwise check for equality, *i.e.*, by changing the proposition from $(s_{J_0}^k = s_{J_0}^{k-1})$ to $(\vee_{l:=0,2,\ldots,k-2}\ s_{J_0}^k = s_{J_0}^\ell)$. The change has a (small) bearing on the security bound given in the original MF Lemma due to an increase in the number of (inequality) checks from n to $(n+1)(n+3)/8$. This is captured in the *revised* version given below.

Lemma 1 (Revised Multiple-Forking Lemma). *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

$$mfrk := \Pr\left[(b=1) \mid x \xleftarrow{\$} \mathcal{G}_I; (b, \{\sigma_0, \ldots, \sigma_n\}) \xleftarrow{\$} \mathcal{M}_{\mathcal{Y},n}(x)\right] \quad \textit{and}$$
$$acc := \Pr\left[(I \geq 1) \wedge (J \geq 1) \mid x \xleftarrow{\$} \mathcal{G}_I; \{s_1, \ldots, s_q\} \xleftarrow{\mathrm{U}} \mathbb{S}; (I, J, \sigma) \xleftarrow{\$} \mathcal{Y}(x, s_1, \ldots, s_q)\right]$$

then

$$mfrk \geq acc \cdot \left(\frac{acc^n}{q^{2n}} - \frac{(n+1)(n+3)}{8|\mathbb{S}|}\right). \tag{2}$$

2.1 Tightness: An Intuitive Picture

Each run of $\mathcal{M}_{\mathcal{Y},n}$ consists of $n+1$ runs of the corresponding wrapper $\mathcal{Y}$ (called round 0 to round n), for some odd n. Informally speaking, $\mathcal{M}_{\mathcal{Y},n}$ is successful provided $\mathcal{Y}$ is successful in each of the $n+1$ rounds and some additional *conditions* are satisfied. We call these set of conditions $\mathbb{A}_0 := \{\mathsf{B}, \mathsf{C}_0, \ldots, \mathsf{C}_{n-1}, \mathsf{D}_2, \ldots, \mathsf{D}_{n-1}\}$ where

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

Let E be the event that all the conditions in $\mathbb{A}_0$ are satisfied, *i.e.*,

$$\mathsf{E} : \mathsf{B} \wedge (\mathsf{C}_0 \wedge \mathsf{C}_2 \wedge \cdots \wedge \mathsf{C}_{n-1}) \wedge (\mathsf{D}_2 \wedge \mathsf{D}_4 \wedge \cdots \wedge \mathsf{D}_{n-1}). \tag{3}$$

What the MF Lemma then gives us is, essentially, a lower bound of the probability of this event.

The logical wrapper $\mathcal{Z}$. A simple *but* crucial observation at this point is that the wrapper algorithm $\mathcal{Y}$ is *always* invoked in pairs (that's the reason for n being odd in the description of $\mathcal{M}_{\mathcal{Y},n}$). Note that, the conditions C_k and D_k above pertain to a single invocation of $\mathcal{Z}$. This brings us to the *conceptual* change introduced in the revised version given in **Algorithm 1**. Two consecutive invocations of $\mathcal{Y}$ (i.e., round k and round k+1, for even $k \geq 0$) have been clubbed together so that it can be visualised (see **Figure 2**) as the invocation of a single logical unit $\mathcal{Z}$ such that $((I_k, J_k, \sigma_k), (I_{k+1}, J_{k+1}, \sigma_{k+1})) \leftarrow \mathcal{Z}\left(x, \mathsf{S}^k, \mathsf{S}^{k+1}; \rho\right)$ (for even k). Here, S^k and S^{k+1} denote the external random for the rounds k and $k+1$ of $\mathcal{Y}$, *i.e.*,

$$\mathsf{S}^k := (s^0_1, \ldots, s^0_{J_0-1}, s^k_{J_0}, \ldots, s^k_q) \text{ and } \mathsf{S}^{k+1} := (s^0_1, \ldots, s^0_{J_0-1}, s^k_{J_0}, \ldots, s^k_{I_k-1}, s^{k+1}_{I_k}, \ldots, s^{k+1}_q).$$

Accordingly, the MF Algorithm constitutes of $m = (n+1)/2$ rounds of invocation of $\mathcal{Z}$. Intuitively, a single invocation of $\mathcal{Z}$ is similar to the GF Algorithm [BN06] as the objective of both is to launch the "elementary" oracle replay attack. The notion of wrapper $\mathcal{Z}$, along with the necessary restructuring of the MF Algorithm–especially, the conditions–provide us the right handle for an improved analysis of its probability of success (*mfrk*).

Among the set of conditions $\mathbb{A}_0$, B is relatively easy to deal with as it is checked only at the beginning and contributes a factor of *acc* in the final expression for *mfrk*. So let's look at the effect of the other (more involved) conditions. Consider the event

$$\mathsf{C}_k : (I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (s^{k+1}_{I_0} \neq s^k_{I_0}).$$

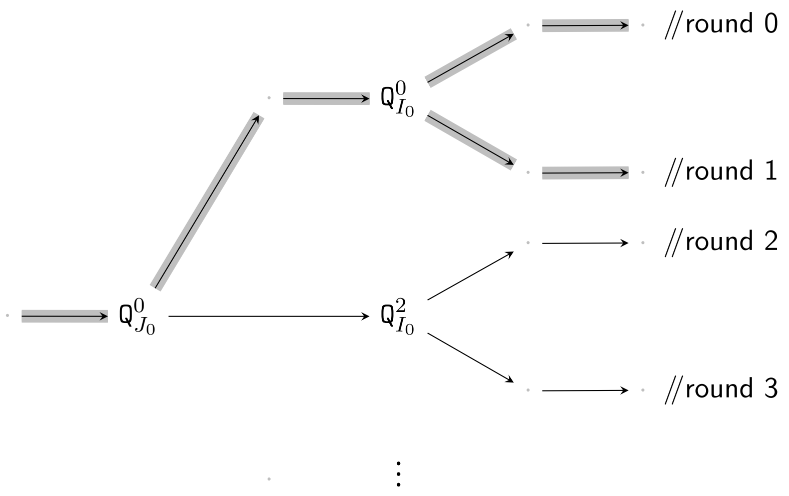

Figure 2: The logical wrapper $\mathcal{Z}$. One such logical wrapper (for round 0 and round 1) has been highlighted.

Clearly, the check for equality $(I_{k+1}, J_{k+1}) = (I_k, J_k)$ is predominant in the final expression for *mfrk* (the equality holds only with a probability of $1/q^2$). On the other hand, $(s^{k+1}_{I_0} \neq s^k_{I_0})$ is almost always true[7]. It's a similar case for the event D_k: $(I_k, J_k) = (I_0, J_0)$ holds only with a probability of $1/q^2$. Hence, the probability of the events C and D is dominated respectively by

$$\wedge_{k:=0,2,\ldots,n-1} (I_{k+1}, J_{k+1}) = (I_k, J_k) \text{ and } \wedge_{k:=2,4,\ldots,n-1} (I_k, J_k) = (I_0, J_0). \tag{4}$$

Consequently, the *degradation* for the MF Algorithm stems, predominantly, from what we term as the "core" event[8]

$$\mathsf{F}_0 : (I_n, J_n) = (I_{n-1}, J_{n-1}) = \cdots = (I_0, J_0) \tag{5}$$

formed by combining the two expressions in (4). Each of the n checks for equality in the condition contributes (roughly speaking) a factor of $\mathsf{O}\left(q^2\right)$, resulting in an overall degradation of $\mathsf{O}\left(q^{2n}\right)$. That's the intuitive reason of the (loose) lower bound one gets for *mfrk*. Naturally, any reduction employing $\mathcal{M}_{\mathcal{Y},n}$ also loses tightness by a factor of $\mathsf{O}\left(q^{2n}\right)$.

2.2 Road-map to a Better Analysis

By now it should be evident that in order to achieve a better bound for *mfrk*, one needs to revisit the conditions associated with the number of checks involved in the success event F_0. For a better understanding of the exact role of these conditions, we revisit the concrete protocols and their security argument that employ the MF Lemma [BPW12, GG09, CMW12]. Based on a careful analysis of these protocols and their security arguments, we formulate the following two observations–called, respectively, the *independence* and the *dependence* conditions.

Observation 1 (Independence condition, O_I). *It is not necessary for the I indices across the logical wrapper $\mathcal{Z}$ to be the same, i.e., I_k need not be equal to $I_{k-2}, I_{k-4}, \ldots, I_0$ for $k = 2, 4, \ldots, n-1$.*

Observation 2 (Dependence condition, O_D). *It is possible to design protocols, in particular, define hash functions used in the protocol, such that, for the k^{th} invocation of the logical wrapper $\mathcal{Z}$, $(I_{k+1} = I_k)$ implies (with very high probability) that $(J_{k+1} = J_k)$.*[9]

[7]It holds with a probability of $(1 - (1/|\mathbb{S}|))$ and for any reasonable security level, $|\mathbb{S}| \gg 1$.

[8]Note that the event F_0 corresponds to the first term of (C.3) in the analysis of the MF Algorithm given in [BPW12].

[9]Assume, without loss of generality, that among the two indices, J always precedes I in a particular run of the wrapper.

Remark 2. The concrete motivation for the above two observations will become clear when we revisit the actual security argument of the existing schemes in §**4**. $\mathsf{O_I}$ is based on a precise analysis of what is actually required from the process of (multiple) forking in the security argument of [BPW12, GG09, CMW12]. In particular, all known applications of MF Algorithm satisfies $\mathsf{O_I}$. $\mathsf{O_D}$ finds its root in the more concrete notion of hash function dependence. 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. The BPW proxy signature scheme of [BPW12] is one example where such dependence exists (see §**4.1** for further details). As in the case of $\mathsf{O_I}$, we'll show that either $\mathsf{O_D}$ is naturally satisfied [CMW12] or one can easily modify existing construction [GG09] to suit the condition (see §**4.4** and §**4.3.1** resp.). The abstraction is required in the context of MF Algorithm due to the absence of the notion of hash function (or random oracles, which model the hash functions in the actual security argument). A formal definition of the notion of index dependence, keeping in mind the observation $\mathsf{O_D}$, is given below.

Definition 1 (Index Dependence). Let $((I, J), (I', J'))$ be the two pair of indices that are part of the output of the wrapper $\mathcal{Z}$ (associated with two rounds of $\mathcal{Y}$). The index J is said to be η-dependent on the index I (denoted by $J \prec I$) if $\Pr[J' \neq J \mid I' = I] \leq \eta$.

In other words, $(I' = I) \implies (J' = J)$ with probability at least $(1 - \eta)$. A little more specifically, J is said to be *fully-dependent* on I 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[10] in the security parameter κ.

Consequences. Let's look at the consequences of the observations one by one. Based on $\mathsf{O_I}$, the condition in (4) can be relaxed to: $\wedge_{k:=0,2,\ldots,n-1}(I_{k+1}, J_{k+1}) = (I_k, J_k)$ and $\wedge_{k:=2,4,\ldots,n-1}(J_k = J_0)$, and, by implication, the "core" event in (5) is relaxed to

$$\mathsf{F}_1 : \wedge_{k=0,2,\ldots,n-1}\big((I_{k+1}, J_{k+1}) = (I_k, J_k)\big) \wedge (J_{n-1} = J_{n-3} = \cdots = J_0). \qquad (6)$$

Hence, the number of overall checks (in the "core" event) is reduced from $2n$ to $(2 \cdot (n+1)/2 + (n-1)/2) = (3n+1)/2$ and the complexity of launching the nested oracle replay attack is brought down to $\mathrm{O}\left(q^{(3n+1)/2}\right)$. A similar analysis shows that based on only $\mathsf{O_D}$ (and, assuming η to be negligible in κ), the "core" event in (5) relaxes to

$$\mathsf{F}_2 : (I_n = I_{n-1} = \cdots = I_1 = I_0) \wedge (J_{n-1} = J_{n-3} = \cdots = J_0) \qquad (7)$$

In this case the number of overall checks is reduced from $2n$ to $(3n-1)/2$ and the complexity to $\mathrm{O}\left(q^{(3n-1)/2}\right)$. The interesting point is that $\mathsf{O_I}$ and $\mathsf{O_D}$ can be employed *in conjunction*. This leads to the "core" event being further simplified to

$$\mathsf{F}_3 : (I_n = I_{n-1}) \wedge \cdots \wedge (I_1 = I_0) \wedge (J_{n-1} = J_{n-3} = \cdots = J_0). \qquad (8)$$

Observe that the number of checks is now reduced to n and the complexity to $\mathrm{O}(q^n)$. Hence, the complexity of launching the nested oracle replay attack is reduced from $\mathrm{O}\left(q^{2n}\right)$ to $\mathrm{O}(q^n)$, but, without losing the modularity of the MF Algorithm.

2.3 A *General* Multiple-Forking Algorithm

As we just observed, depending on whether (or not) the observations $\mathsf{O_I}$ and $\mathsf{O_D}$ are taken into account, we end up with four different sets of "core" conditions F_0 through F_3. The "non-core" conditions, though, remain the same for all the four cases. The resulting set of "full" conditions,

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

$\mathbb{A}_0$ through $\mathbb{A}_3$, (along with the associated degradation) are given in **Table 2**. In order to capture this extra level of abstraction, we describe a general framework of the MF Algorithm that has, in addition to the algorithm $\mathcal{Y}$, an associated (ordered) set of conditions $\mathbb{A}$. Note that we have assumed full-dependence.

MF	Set of Conditions	Degradation
Original	$\mathbb{A}_0 = \begin{cases} \mathsf{B}: & (I_0 \geq 1) \wedge (J_0 \geq 1) \\ \mathsf{C}_k: & (I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (s_{I_k}^{k+1} \neq s_{I_k}^{k}) \\ \mathsf{D}_k: & (I_k, J_k) = (I_0, J_0) \wedge (\wedge_{\ell:=0,2,\ldots,k-2}\, s_{J_0}^{k} \neq s_{J_0}^{\ell}) \end{cases}$	$\mathsf{O}\left(q^{2n}\right)$
with O_{I}	$\mathbb{A}_1 = \begin{cases} \mathsf{B}: & (I_0 \geq 1) \wedge (J_0 \geq 1) \\ \mathsf{C}_k: & (I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (s_{I_k}^{k+1} \neq s_{I_k}^{k}) \\ \mathsf{D}_k: & (J_k = J_0) \wedge (I_k \geq 1) \wedge (\wedge_{\ell:=0,2,\ldots,k-2}\, s_{J_0}^{k} \neq s_{J_0}^{\ell}) \end{cases}$	$\mathsf{O}\left(q^{(3n+1)/2}\right)$
with O_{D}	$\mathbb{A}_2 = \begin{cases} \mathsf{B}: & (1 \leq J_0 < I_0 \leq q) \\ \mathsf{C}_k: & (I_{k+1} = I_k) \wedge (s_{I_k}^{k+1} \neq s_{I_k}^{k}) \\ \mathsf{D}_k: & (I_k, J_k) = (I_0, J_0) \wedge (\wedge_{\ell:=0,2,\ldots,k-2}\, s_{J_0}^{k} \neq s_{J_0}^{\ell}) \end{cases}$	$\mathsf{O}\left(q^{(3n-1)/2}\right)$
with $\mathsf{O}_{\{\mathrm{I,D}\}}$	$\mathbb{A}_3 = \begin{cases} \mathsf{B}: & (1 \leq J_0 < I_0 \leq q) \\ \mathsf{C}_k: & (I_{k+1} = I_k) \wedge (s_{I_k}^{k+1} \neq s_{I_k}^{k}) \\ \mathsf{D}_k: & (J_k = J_0) \wedge (J_k < I_k \leq q) \wedge (\wedge_{\ell:=0,2,\ldots,k-2}\, s_{J_0}^{k} \neq s_{J_0}^{\ell}) \end{cases}$	$\mathsf{O}\left(q^{n}\right)$

Table 2: The set of conditions is $\mathbb{A} := \{\mathsf{B}, \mathsf{C}, \mathsf{D}\}$ where C denotes $\mathsf{C}_0, \mathsf{C}_2, \ldots, \mathsf{C}_{n-1}$ and D denotes $\mathsf{D}_2, \mathsf{D}_4, \ldots, \mathsf{D}_{n-1}$. Also, note that we have ignored the (common) ϵ^n factor in the degradation and assumed full-dependence.

The General Multiple-Forking Algorithm Fix $q \in \mathbb{Z}^+$ and a set $\mathbb{S}$ such that $|\mathbb{S}| \geq 2$. Let $\mathcal{Y}$ be a randomised algorithm that on input a string x and elements $s_1, \ldots, s_q \in \mathbb{S}$ 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. In addition, let $\mathbb{A}$ denote the set of conditions, and be of the form $\{\mathsf{B}, \mathsf{C}, \mathsf{D}\}$ with $\mathsf{C} := \mathsf{C}_0, \mathsf{C}_2, \ldots, \mathsf{C}_{n-1}$ and $\mathsf{D} := \mathsf{D}_2, \mathsf{D}_4, \ldots, \mathsf{D}_{n-1}$. The General MF Algorithm $\mathcal{N}_{\mathbb{A},\mathcal{Y},n}$ associated to $\mathbb{A}$, $\mathcal{Y}$ and n is defined as **Algorithm 2** below.

Algorithm 2 $\mathcal{N}_{\mathbb{A},\mathcal{Y},n}(x)$

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

$\{s_1^0,\ldots,s_q^0\} \xleftarrow{U} \mathbb{S}$;
$(I_0,J_0,\sigma_0) \leftarrow \mathcal{Y}(x,s_1^0,\ldots,s_q^0;\rho)$   //round 0
$\{s_{I_0}^1,\ldots,s_q^1\} \xleftarrow{U} \mathbb{S}$;
$(I_1,J_1,\sigma_1) \leftarrow \mathcal{Y}(x,s_1^0,\ldots,s_{I_0-1}^1,s_{I_0}^1,\ldots,s_q^1;\rho)$   //round 1

if $\neg(\mathsf{B} \wedge \mathsf{C}_0)$ then return $(0,\perp)$

$k := 2$
while $(k < n)$ do
    $\{s_{J_0}^k,\ldots,s_q^k\} \xleftarrow{U} \mathbb{S}$;
    $(I_k,J_k,\sigma_k) \leftarrow \mathcal{Y}(x,s_1^0,\ldots,s_{J_0-1}^0,s_{J_0}^k,\ldots,s_q^k;\rho)$   //round k
    $\{s_{I_k}^{k+1},\ldots,s_q^{k+1}\} \xleftarrow{U} \mathbb{S}$;
    $(I_{k+1},J_{k+1},\sigma_{k+1}) \leftarrow \mathcal{Y}(x,s_1^0,\ldots,s_{J_0-1}^0,s_{J_0}^k,\ldots,s_{I_k-1}^k,s_{I_k}^{k+1},\ldots,s_q^{k+1};\rho)$   //round k+1

    if $\neg(\mathsf{C}_k \wedge \mathsf{D}_k)$ then return $(0,\perp)$

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

Remark 3 (On usage)**.** We use the set of conditions $\mathbb{A}_2$ and $\mathbb{A}_3$ *only* in the case of *i*) full-dependence; or *ii*) η-dependence with negligible η. For η-dependence, in general, we use the set of conditions $\mathbb{A}_0$ and $\mathbb{A}_1$ respectively (*e.g.*, see **Lemma 2** and its proof).

3 Harnessing (*In*)Dependence

We have seen in the previous section that the most effective way of launching the nested replay attack is by exploiting both $\mathsf{O_I}$ and $\mathsf{O_D}$. As expected, the analysis of this case turns out to be the most involved and in a sense encompasses the analysis of the other two set of conditions ($\mathbb{A}_1$ and $\mathbb{A}_2$). Hence in this section we focus on analysing $\mathcal{N}_{\mathbb{A}_1,\mathcal{Y},n}$ with η-dependence. The analysis of $\mathcal{N}_{\mathbb{A}_1,\mathcal{Y},n}$ and $\mathcal{N}_{\mathbb{A}_0,\mathcal{Y},n}$ with η-dependence is deferred to **Appendix D** (see **Lemma 9** and **Lemma 10**).

3.1 Multiple-Forking with Index (In)Dependence

The probability of success of the MF Algorithm with both $\mathsf{O_I}$ and $\mathsf{O_D}$ is bounded by **Lemma 2** given below. The details of some of the steps in the probability analysis is given in **Appendix D.1**.

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

$$mfrk_3 := \Pr\left[(b=1) \mid x \xleftarrow{\$} \mathcal{G}_I; (b,\{\sigma_0,\ldots,\sigma_n\}) \xleftarrow{\$} \mathcal{N}_{\mathbb{A}_1,\mathcal{Y},n}(x)\right] \quad \textit{and}$$

$$acc := \Pr\left[(1 \le J < I \le q) \mid x \xleftarrow{\$} \mathcal{G}_I; \{s_1,\ldots,s_q\} \xleftarrow{U} \mathbb{S}; (I,J,\sigma) \xleftarrow{\$} \mathcal{Y}(x,s_1,\ldots,s_q)\right].$$

On the assumption that J is η-dependent on I,

$$mfrk_3 \geq frk\left(\frac{frk^{(n-1)/2}}{q^{(n-1)/2}} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\right) \text{ where } frk \geq acc\left(\frac{acc}{q}(1-\eta) - \frac{1}{|\mathbb{S}|}\right) \tag{9}$$

The main hurdle in proving the bounds lies in exploiting the leverage offered by O_I and O_D, in the probability analysis. As it turns out, O_D is much harder to integrate. We have to appeal to the underlying sets (*i.e.* the source of randomness) of the GMF Algorithm. On the other hand, the analysis of O_I is in some sense similar to the analysis of the original MF Algorithm in [BPW12]. Naturally, the simultaneous analysis of O_I and O_D is a "hybrid" of the above two. The subtle change in the accepting condition to $(1 \leq J < I \leq q)$ from $(I \geq 1) \wedge (J \geq 1)$ in **Lemma 1** is also crucial in establishing the bounds (see **Claim 3** below). The following two inequalities are also used.

Lemma 3 (Jensen's inequality). *Let f be a convex function and X be a real-valued random variable. Then*

$$Ex[f(X)] \geq f(Ex[X]).$$

Lemma 4 (Hölder's inequality[11]). *Let $q \in \mathbb{Z}^+$, $1 \leq n < \infty$ and $x_1, \ldots, x_q \geq 0$ be real numbers. Then*

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

Conventions. Before moving on to the proof, we fix some conventions with regard to the coins involved in the GMF Algorithm. The internal coins for the algorithm $\mathcal{Y}$ is denoted by ρ. We assume that ρ is drawn from a set $\mathbb{R}$ (not to be confused with the set of real numbers). On the other hand, the external randomness for $\mathcal{Y}$ is denoted by $\mathsf{S} := \{s_1, \ldots, s_q\}$ (with the round indicated in the superscript, *e.g.* S^0). For convenience, we use the convention in **Table 3** to split up S. Hence, $\{\mathsf{S}_{(i-)}, \mathsf{S}_{(i+)}\}$ indicates S split up into two at the index i, whereas $\{\mathsf{S}_{(j-)}, \mathsf{S}_{(ji)}, \mathsf{S}_{(i+)}\}$

Symbol	Denotes	Domain
$\mathsf{S}_{(i-)}$	$s_1, \ldots, s_{i-1}$	$\lvert\mathbb{S}\rvert^{i-1}$
$\mathsf{S}_{(ji)}$	$s_j, \ldots, s_{i-1}$	$\lvert\mathbb{S}\rvert^{i-j}$
$\mathsf{S}_{(i+)}$	$s_i, \ldots, s_q$	$\lvert\mathbb{S}\rvert^{q-i+1}$

Table 3: Shorthand for external randomness.

indicates S split into three, at indices j and i respectively. Finally, $\mathbb{T}_{(2)}$ denotes the set from which coins (both internal and external) are drawn for $\mathcal{Z}$ and $\mathbb{T}_{(n)}$ denotes the set of coins for the GMF Algorithm[12].

[11]Although, the result is a corollary to a more general Hölder's inequality (see [Lemma C.3][BPW12]), another way to proving the bound is by viewing it an optimisation problem. Let $\mathsf{f}(x_1, \ldots, x_q) := \sum_{k:=1}^{q} x_k^n$ 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 $k \in \{1, \ldots, q\}$. Then f attains a minima of x^n/q^{n-1} at the point $(x/q, \ldots, x/q)$, thus, establishing **Lemma 4**.

[12]With regard to the coins, it is apparent that the source of coins for a single invocation of $\mathcal{Y}$ is the set $\mathbb{R} \times \mathbb{S}^q$. As for two invocations of $\mathcal{Y}$ (single invocation of $\mathcal{Z}$), it can be worked out that the coins are drawn from the set

$$\mathbb{T}_{(2)} := \mathbb{R} \times \bigcup_{i=2}^{q} \mathbb{S}^{i-1} \times \mathbb{S}^{(q-i+1)*2}.$$

Proof. For a fixed string x, let

$$mfrk_3(x) := \Pr\left[(\mathsf{b}=1) \mid (\mathsf{b}, \{\sigma_0, \ldots, \sigma_n\}) \xleftarrow{\$} \mathcal{N}_{\mathbb{A}_1, \mathcal{Y}, n}(x)\right] \quad \text{and}$$
$$acc(x) := \Pr\left[(1 \leq J < I \leq q) \mid \{s_1, \ldots, s_q\} \xleftarrow{\mathrm{U}} \mathbb{S}; (I, J, \sigma) \xleftarrow{\$} \mathcal{Y}(x, s_1, \ldots, s_q)\right].$$

Let $\mathbb{A}_1 := \{\mathsf{B}, \mathsf{C}_0, \mathsf{C}_2, \ldots, \mathsf{C}_{n-1}, \mathsf{D}_2, \mathsf{D}_4, \ldots, \mathsf{D}_{n-1}\}$. For ease of notation, we further break the event C_k (resp. D_k) into two subevents $\mathsf{C}_{k,c}$ and $\mathsf{C}_{k,s}$ (resp. $\mathsf{D}_{k,c}$ and $\mathsf{D}_{k,s}$) as shown below.

$$\begin{aligned} \mathsf{C}_{k,c} &: (I_{k+1} = I_k) & \mathsf{C}_{k,s} &: (s^{k+1}_{I_k} \neq s^{k}_{I_k}) \\ \mathsf{D}_{k,c} &: (J_k = J_0) \wedge (J_k < I_k \leq q) & \mathsf{D}_{k,s} &: (\wedge_{\ell:=0,2,\ldots,k-2}\, s^{k}_{J_0} \neq s^{\ell}_{J_0}) \end{aligned} \tag{10}$$

The GMF Algorithm is successful in the event $\mathsf{E} : \mathsf{B} \wedge (\mathsf{C}_0 \wedge \mathsf{C}_2 \wedge \cdots \wedge \mathsf{C}_{n-1}) \wedge (\mathsf{D}_2 \wedge \mathsf{D}_4 \wedge \cdots \wedge \mathsf{D}_{n-1})$. In other words, with the probabilities calculated over the coin tosses of the GMF Algorithm, it follows that $mfrk_3(x) = \Pr[\mathsf{E}]$. The task of bounding this probability is accomplished through three claims (**Claim 1** through **Claim 3**). The object at the centre of **Claim 1** is the logical unit $\mathcal{Z}$. It turns out that *with* index dependence, the behaviour of $\mathcal{Z}$ is similar to the GF Algorithm [BN06]. The aim of **Claim 1** is to bound the probability of success of $\mathcal{Z}$, denoted by $frk(x)$, in terms of $acc(x)$. The bound on $frk(x)$ is, then, used in **Claim 2** and **Claim 3** to bound $mfrk_3(x)$.

We start with the analysis of a single invocation of $\mathcal{Z}$. Without loss of generality, let's consider the first two rounds of the GMF Algorithm.

Claim 1. $frk(x) \geq acc(x)\left(\dfrac{acc(x)}{q}(1-\eta(x)) - \dfrac{1}{|\mathbb{S}|}\right)$.

Argument. With the probability taken over the coin tosses over the two rounds (which we denote by $\mathbb{T}_{(2)}$), it follows that

$$\begin{aligned} frk(x) &= \Pr[\mathsf{B} \wedge \mathsf{C}_0] = \Pr\left[\mathsf{B} \wedge \mathsf{C}_{0,c} \wedge \mathsf{C}_{0,s}\right] \quad \text{(using subevents given in (10))} \\ &\geq \Pr[(J_1 = J_0) \wedge (I_1 = I_0) \wedge (1 \leq J_0 < I_0 \leq q)] - \Pr\left[(1 \leq J_0 < I_0 \leq q) \wedge (s^0_{I_0} = s^1_{I_0})\right] \end{aligned} \tag{11}$$

Let's denote the two parts of the above expression by $frk_c(x)$ and $frk_s(x)$ respectively. The second part can be computed as follows:

$$frk_s(x) = \Pr[1 \leq J_0 < I_0 \leq q] \cdot \Pr\left[s^0_{I_0} = s^1_{I_0}\right] = acc(x)/|\mathbb{S}|. \tag{12}$$

The first part of (11), on the other hand, forms the "core" probability for the two rounds. In order to analyse it, we define a random variable which captures single invocation of the algorithm $\mathcal{Y}$. Let $Y_i : \mathbb{R} \times \mathbb{S}^{i-1} \mapsto [0,1]$, for each $i \in \{2, \ldots, q\}$, be defined by setting

$$Y_i(\rho, \mathsf{S}_{(i-)}) = \Pr\left[(I = i) \wedge (1 \leq J < i)) \mid \mathsf{S}_{(i+)} \xleftarrow{\mathrm{U}} \mathbb{S}; (I, J, \sigma) \leftarrow \mathcal{Y}(x, \{\mathsf{S}_{(i-)}, \mathsf{S}_{(i+)}\}; \rho)\right].$$

By a simple extrapolation, the coins for the GMF Algorithm are drawn from

$$\mathbb{T}_{(n)} := \mathbb{R} \times \bigcup_{j=1}^{q-1} \mathbb{S}^{j-1} \times \left(\bigcup_{i=j+1}^{q} \mathbb{S}^{i-j} \times \mathbb{S}^{(q-i+1)*2}\right)^{(n+1)/2}.$$

Using Y_i, $\mathit{frk}_c(x)$ can be rewritten as follows.

$$\begin{aligned}
\mathit{frk}_c(x) &= \Pr\left[(J_1 = J_0) \mid (I_1 = I_0) \wedge (1 \leq J_0 < I_0 \leq q)\right] \Pr\left[(I_1 = I_0) \wedge (1 \leq J_0 < I_0 \leq q)\right] \\
&= (1-\eta(x)) \sum_{i=2}^{q} \sum_{\rho, \mathsf{S}_{(i-)}} \frac{1}{|\mathbb{R}||\mathbb{S}|^{i-1}} Y_i^2(\rho, \mathsf{S}_{(i-)}) \text{ (see (28) in } \textbf{Appendix D.1} \text{ for details)} \\
&= (1-\eta(x)) \sum_{i=2}^{q} \mathrm{Ex}\left[Y_i^2\right] \geq \frac{1}{q}\left(\sum_{i=2}^{q} \mathrm{Ex}\left[Y_i\right]\right)^2 \text{ (by Jensen's and Hölder's inequality)} \\
&= (1-\eta(x)) \frac{1}{q} \mathit{acc}(x)^2 \text{ (by definition of } Y_i \text{ and } \mathit{acc}(x)) \qquad (13)
\end{aligned}$$

By substituting (12) and (13) in (11), we get

$$\mathit{frk}(x) = \Pr_{\mathbb{T}_{(2)}}\left[(I_1 = I_0) \wedge (1 \leq J_0 < I_0 \leq q) \wedge (s^0_{I_0} \neq s^1_{I_0})\right] \geq \mathit{acc}(x)\left(\frac{\mathit{acc}(x)}{q}(1-\eta(x)) - \frac{1}{|\mathbb{S}|}\right) \qquad (14)$$

establishing **Claim 1**. □

The fundamental difference between the argument of **Claim 1** and the proof of GF Algorithm in [BN06] is the involvement of the additional index J. The *only* reason that the analysis proceeds as in [BN06] is due to the assumption $J \prec I$. Without this assumption, the argument for **Claim 1** would require defining a random variable that takes both the indices into consideration (this is indeed the case in the proof of **Lemma 9**).

Remark 4. In **Claim 1**, what we have effectively established is that on $\mathsf{O}_{\{\mathrm{I,D}\}}$, single augmented forking can be carried out as efficiently as an elementary forking.

The remaining two claims require a random variable Z_j that captures a single invocation of $\mathcal{Z}$. Let $Z_j : \mathbb{R} \times \mathbb{S}^{j-1} \mapsto [0,1]$, for $1 \leq j \leq q-1$, be defined by setting

$$Z_j(\rho, \mathsf{S}_{(j-)}) = \Pr\left[(J' = J = j) \wedge (I' = I) \wedge (j < I \leq q) \wedge (s'_I \neq s_I)\right]$$

given

$$(\mathsf{S}_{(jI)}, (\mathsf{S}_{(I+)}, \mathsf{S}'_{(I+)})) \overset{\mathrm{U}}{\leftarrow} \mathbb{S} \text{ and}$$
$$((I, J, \sigma), (I', J', \sigma')) \leftarrow \mathcal{Z}(x, \{\mathsf{S}_{(j-)}, \mathsf{S}_{(jI)}, \mathsf{S}_{(I+)}\}, \{\mathsf{S}_{(j-)}, \mathsf{S}_{(jI)}, \mathsf{S}'_{(I+)}\}; \rho).$$

A crucial point is that the observation O_{I} has been considered in the definition of Z_j. Without this assumption, the analysis would require a random variable that takes both the indices into consideration (as we do later in the proof of **Lemma 10** using a random variable $Z_{i,j}$). Briefly, our aim is to bound $\mathit{mfrk}_3(x)$ in terms of Z_j (**Claim 2**) and then bound Z_j in terms of $\mathit{frk}(x)$ (**Claim 3**).

Claim 2.

$$\mathit{mfrk}_3(x) \geq \frac{1}{q^{(n-1)/2}}\left(\sum_{j=1}^{q-1} \mathit{Ex}\left[Z_j\right]\right)^{(n+1)/2} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\left(\sum_{j=1}^{q-1} \mathit{Ex}\left[Z_j\right]\right) \qquad (15)$$

Argument. The first step is to separate the "core" subevents out of the event E as shown below.

$$\begin{aligned}
\Pr\left[\mathsf{E}\right] &= \Pr\left[\mathsf{B} \wedge \left(\wedge_{k=0,2,\ldots,n-1} \mathsf{C}_k\right) \wedge \left(\wedge_{k=2,4,\ldots,n-1} \mathsf{D}_k\right)\right] \\
&\geq \Pr\left[\mathsf{B} \wedge \mathsf{C}_0 \wedge \left(\wedge_{k=2,4,\ldots,n-1} \mathsf{C}_k \wedge \mathsf{D}_{k,c}\right)\right] - \Pr\left[\mathsf{B} \wedge \mathsf{C}_0 \wedge \left(\vee_{k=2,4,\ldots,n-1} \neg \mathsf{D}_{k,s}\right)\right] \qquad (16)
\end{aligned}$$

We denote the first part of (16) by *mfrk*$_{3,c}(x)$ and the second part by *mfrk*$_{3,s}(x)$ and analyse them separately.

$$\begin{aligned}
\mathit{mfrk}_{3,c}(x) &= \sum_{j=1}^{q-1} \Pr\left[\,\wedge_{k=0,2,\ldots,n-1}\left((J_{k+1}=J_k=j)\wedge(I_{k+1}=I_k)\wedge(j<I_k\le q)\wedge(s^{k+1}_{I_k}\neq s^{k}_{I_k})\right)\right] \\
&= \sum_{j=1}^{q-1}\sum_{\rho,\mathtt{s}_{(j-)}} \frac{\prod_{k=0,2}^{n-1} Z_j(\rho,\mathtt{s}_{(j-)})}{|\mathbb{R}||\mathbb{S}|^{j-1}} \quad \text{(see (29) in \textbf{Appendix D.1} for details)} \\
&= \sum_{j=1}^{q-1} \mathrm{Ex}\left[Z_j^{(n+1)/2}\right] \ge \frac{1}{q^{(n-1)/2}}\left(\sum_{j=1}^{q-1}\mathrm{Ex}\left[Z_j\right]\right)^{(n+1)/2} \quad \text{(by Jensen's and Hölder's inequality)}
\end{aligned} \tag{17}$$

Using a similar line of approach as above, it is possible to bound *mfrk*$_{3,s}$ too in terms of Z_j (see (31) in **Appendix D.1** for the intermediate steps).

$$\mathit{mfrk}_{3,s}(x) = \frac{(n-1)(n+1)}{8|\mathbb{S}|}\left(\sum_{j=1}^{q-1}\mathrm{Ex}\left[Z_j\right]\right) \tag{18}$$

Substituting the value of *mfrk*$_{3,c}(x)$ from (17) and *mfrk*$_{3,s}(x)$ from (18) in (16) proves **Claim 2**. □

What remains is to relate **Claim 1** and **Claim 2** by establishing

Claim 3. $\sum_{j=1}^{q-1} Ex\left[Z_j\right] \ge \mathit{frk}(x)$.

Argument. Recall that the bound given in **Claim 1** for *frk*(x) is with the probability taken over the set

$$\mathbb{T}_{(2)} := \mathbb{R}\times\bigcup_{i=2}^{q}\mathbb{S}^{i-1}\times\mathbb{S}^{(q-i+1)*2}.$$

Let $\mathbb{T}_{(2,j>)}$ denote the underlying set for the random variable Z_j. From the definition of Z_j, we can infer that

$$\begin{aligned}
\mathbb{T}_{(2,j>)} &:= \left(\mathbb{R}\times\mathbb{S}^{j-1}\times\bigcup_{i=j+1}^{q}\mathbb{S}^{i-j}\times\mathbb{S}^{(q-i+1)*2}\right) = \left(\mathbb{R}\times\bigcup_{i=j+1}^{q}\mathbb{S}^{i-1}\times\mathbb{S}^{(q-i+1)*2}\right) \text{ and} \\
\mathrm{Ex}\left[Z_j\right] &= \Pr_{\mathbb{T}_{(2,j>)}}\left[(J'=J=j)\wedge(I'=I)\wedge(j<I\le q)\wedge(s'_I\neq s_I)\right].
\end{aligned} \tag{19}$$

Notice that the set $\mathbb{T}_{(2,j>)}$ is a subset of the set $\mathbb{T}_{(2)}$. In fact, $\mathbb{T}_{(2)}$ can be partitioned into the two sets $\mathbb{T}_{(2,j<)}$ and $\mathbb{T}_{(2,j>)}$ where

$$\mathbb{T}_{(2,j<)} := \mathbb{R}\times\bigcup_{i=2}^{j}\mathbb{S}^{i-1}\times\mathbb{S}^{(q-i+1)*2}.$$

At this point the subtle change in the definition of the *acc*(x) comes into play. The accepting condition had been altered[13] from $(I\ge 1)\wedge(J\ge 1)$ in **Lemma 1** to $(1\le J<I\le q)$ in **Lemma 2**. By implication

$$\Pr_{\mathbb{T}_{(2,j<)}}\left[(J'=J=j)\wedge(I'=I)\wedge(j<I\le q)\wedge(s'_I\neq s_I)\right] = 0. \tag{20}$$

[13]To be precise, we have taken into account the definition of $\mathcal{Y}$ while defining *acc*(x). Interestingly, [BPW12] could also have incorporated this.

Intuitively speaking, (20) is a consequence of the fact that the definition of $acc(x)$ prohibits a successful $\mathcal{Y}$ from returning (I, J) such that $I \leq J$. The fact that $\mathbb{T}_{(2,j>)} \subset \mathbb{T}_{(2)}$, in conjunction with (20) implies

$$\Pr_{\mathbb{T}_{(2,j>)}} \left[(J' = J = j) \wedge (I' = I) \wedge (j < I \leq q) \wedge (s'_I \neq s_I)\right] \geq \\ \Pr_{\mathbb{T}_{(2)}} \left[(J' = J = j) \wedge (I' = I) \wedge (j < I \leq q) \wedge (s'_I \neq s_I)\right]. \tag{21}$$

Finally, on taking the sum of $\text{Ex}\,[Z_j]$ over the index j, we get

$$\sum_{j=1}^{q-1} \text{Ex}\,[Z_j] \geq \sum_{j=1}^{q-1} \Pr_{\mathbb{T}_{(2)}} \left[(J' = J = j) \wedge (I' = I) \wedge (j < I \leq q) \wedge (s_I \neq s'_I)\right] \quad \text{(using (21))} \\ = \Pr_{\mathbb{T}_{(2)}} \left[(J' = J) \wedge (I' = I) \wedge (1 \leq J < I \leq q) \wedge (s_I \neq s'_I)\right] = \mathit{frk}(x) \tag{22}$$

completing the argument. □

On putting all the three claims together, we get

$$\mathit{mfrk}_3(x) \geq \frac{\mathit{frk}(x)^{(n+1)/2}}{q^{(n-1)/2}} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\mathit{frk}(x) \\ = \mathit{frk}(x) \cdot \left(\frac{\mathit{frk}(x)^{(n-1)/2}}{q^{(n-1)/2}} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\right)$$

Finally, taking the expectation over $x \xleftarrow{\$} \mathcal{G}_I$, yields

$$\mathit{frk} \geq \mathit{acc}\left(\frac{\mathit{acc}}{q}(1-\eta) - \frac{1}{|\mathbb{S}|}\right) \quad \text{and} \quad \mathit{mfrk}_3 \geq \mathit{frk} \cdot \left(\frac{\mathit{frk}^{(n-1)/2}}{q^{(n-1)/2}} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\right)$$

establishing **Lemma 2**. We conclude with the comment that on assuming $|\mathbb{S}| \gg 1$, one gets $\mathit{mfrk}_3 \approx \mathit{acc}^{n+1}/q^n$. □

3.2 Tightness of the Lower Bound

Assuming that an elementary forking cannot be launched more efficiently than in the GF Algorithm, the analysis that we obtained is optimal. This is captured by the following theorem.

Theorem 1. *Suppose both η and $1/|\mathbb{S}|$ are negligible in security parameter. Given an MF Algorithm that is successful with a probability of $O\left(\epsilon^{n+1}/q^n\right)$, one can construct a GF Algorithm that is successful with a probability $O\left(\epsilon^2/q\right)$.*

Argument. We give an informal argument which proceeds by induction on n, the number of rounds. Let's consider the base case of $n = 1$: *given an MF Algorithm that is successful with a probability of $O\left(\epsilon^2/q\right)$, one can construct a GF Algorithm that is successful with a probability $O\left(\epsilon^2/q\right)$*. The argument is straightforward: the MF Algorithm, itself, acts as the GF Algorithm. Thus, a single augmented forking can be carried out with a success probability of $O(\epsilon/q)$. Next, for the sake of induction, we assume that the statement is true[14] for $n-2$: *given an MF Algorithm that is successful with a probability $O\left(\epsilon^{n-1}/q^{n-2}\right)$, one can construct a GF Algorithm that is successful with a probability of $O\left(\epsilon^2/q\right)$*. We argue that the *same* GF Algorithm–one given for the inductive hypothesis–can be used for the case of n as well. Note that we need to carry out two additional augmented forkings given the MF Algorithm for $n-2$. These, assuming the base case, can be carried out with a combined success probability of $O\left(\epsilon^2/q^2\right)$ and, thus, the success probability of the n forks is $O\left(\epsilon^{n-1}/q^{n-2}\right) \cdot O\left(\epsilon^2/q^2\right) = O\left(\epsilon^{n+1}/q^n\right)$ completing the argument. □

[14]Recall that the MF Algorithm is defined for odd n.

4 Revisiting the Security Argument of Existing Protocols

We now take a closer look at the protocols that employ the MF Algorithm in their security argument. Our primary objective is to examine the applicability of the observations $\mathrm{O_I}$ and $\mathrm{O_D}$. We also comment on the design of the corresponding wrappers. We use the proxy signature scheme of Boldyreva *et al.* (BPW-PSS) [BPW12] to motivate the notion of (random oracle) dependence and show how both $\mathrm{O_I}$ and $\mathrm{O_D}$ can be captured in the security argument. We then suggest a small modification–coined "binding"–in the identity-based signature scheme of Galindo and Garcia (GG-IBS) [GG09] to induce hash function dependence and briefly comment on its security (a detailed argument is provided in **Appendix F**). The effect on zero-knowledge protocol of Chow *et al.* (CMW-ZKP) [CMW12] is discussed in §**4.4**.

4.1 Random-Oracle Dependence

The notion of hash function (or, random-oracle) dependence can be best appreciated through a concrete example. Consider the construction of BPW-PSS scheme (see **Figure 4** of **Appendix E.1**). The protocol uses three hash functions: G, R and H. The hash functions are called by the different algorithms in a certain *order* and here we'll mainly focus on the generation of proxy signature. Observe that Delegation uses G for producing proxy certificates and R for generating proxy secret keys; H is used in Proxy Signing for computing proxy signatures. The proxy signature is computed using the proxy secret key which, in turn, is computed using the proxy certificate. Hence, to generate a proxy signature the hash function calls must follow a *logical* order: $\mathrm{G} < \mathrm{R} < \mathrm{H}$ (here $<$ denotes 'followed by').

Now let's take a look at the structure of hash function calls:

$$c := \mathrm{G}(0\|S\|Y) \qquad r := \mathrm{R}(S\|Y\|c) \qquad h := \mathrm{H}(0\|\tilde{m}\|S\|Y\|V\|r).$$

The critical point is to observe the *binding* between the hash functions. R takes as an input c which is the output of G; H takes as an input r which is the output of R. Consequently, to produce a proxy signature, one *has to* (except, with a negligible probability of guessing) call the hash functions in the order: $\mathrm{G} < \mathrm{R} < \mathrm{H}$ (which is also the logical order). In other words, the logical order has been explicitly imposed as the only *viable* order.

Next, consider the simulation of the protocol environment for the BPW-PSS where the hash functions are modelled as random oracles. The aforesaid order among the hash functions naturally translates into an order among the corresponding random oracles. Hence, to forge a proxy signature, an adversary *has* to (except, with a negligible probability of guessing) make the target random oracle queries in the order $\mathrm{G} < \mathrm{R} < \mathrm{H}$. In other words, if K, J and I are the indices that refer to the target G, R and H random oracle queries corresponding to the forgery, then it follows that $(1 \leq K < J < I \leq q)$. Now, consider a second round of simulation of the adversary initiated by a forking at I (corresponding to the successful target H-query). Suppose the adversary is successful in the second round and, in addition, the target H index for the second round matches with the first (*i.e.*, $I' = I$). It is not difficult to see that, due to the binding between the random oracles, the R and G target indices for the two rounds also *have to* (except, as we shall see, with probability denoted by η_b) match. The advantage with which an adversary can forge a proxy signature on violating this condition is, in fact, (asymptotically) negligible. Hence, $(I' = I)$ implies $(J' = J)$ and $(K' = K)$. We say that the random oracle H is "dependent" on the random oracles G and R (denoted by $\{\mathrm{G}, \mathrm{R}\} \prec \mathrm{H}$) over these two rounds (in other words, within the wrapper of $\mathcal{Z}$ in the context of multiple forking). Using a similar line of argument, one can establish that R is dependent on G (*i.e.* $\mathrm{G} \prec \mathrm{R}$). On putting together the two observations, in the case of BPW-PSS we get $\mathrm{G} \prec \mathrm{R} \prec \mathrm{H}$.

A little more formally, the dependence among two random oracles is defined as follows.

Definition 2 (Random-Oracle Dependence)**.** 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 $\mathsf{H}_1 \prec \mathsf{H}_2$) if the following criteria are satisfied: *i)* $(1 \leq J < I \leq q)$ or, in other words $\mathsf{H}_1 < \mathsf{H}_2$; and *ii)* $\Pr[(J' \neq J) \mid (I' = I)] \leq \eta$.[15]

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 logical order among the "hash" functions in the protocol does not necessarily mean that there is a dependence among them.[16] Hence, one may need to impose explicit dependence among the hash functions. A natural way to induce the dependence $\mathsf{H}_1 \prec \mathsf{H}_2$ is through the *binding* technique used in BPW-PSS: by making the input to H_2 a function of H_1's output.

Claim 4. *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 dependence $H_1 \prec H_2$ with $\eta_b := q_1(q_1-1)/|\mathbb{R}_1|$.*

Argument. Suppose $(I' = I)$ but $(J' \neq J)$. It is not difficult to see that the only scenario that can lead to this is if the adversary comes up with a collision on the random function correspond to the oracle H_1. This, according to the birthday bound, can happen with probability at most $q_1(q_1-1)/|\mathbb{R}_1|$. □

Remark 5. 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 due to the MF Lemma, we end up choosing $|\mathbb{R}_1| := 2^{260}$ rendering η_b negligible. Hence, we assume η_b to be negligible for concrete values as well.

Remark 6. The notion of random-oracle dependence can be naturally adapted for (interactive) commitment-challenge rounds (through [FS87]) with the notion of target commitment-challenge round in place of the notion of target random oracle query. This is demonstrated for the CMW-ZKP scheme in coming section (§**4.4**).

4.2 The Boldyreva-Palacio-Warinschi Proxy Signature Scheme

We refer the reader to [BPW12] for the definition of proxy signature and the original security argument of BPW-PSS (the construction is reproduced in **Appendix E.1**). The extremely technical and long security argument of [BPW12] consists of five reductions $\mathcal{B}$ through $\mathcal{F}$, with the associated wrappers $\mathcal{Y}, \mathcal{Z}', \mathcal{U}, \mathcal{V}$ and $\mathcal{W}$ respectively[17]. The discrete-log problem is reduced to breaking the scheme in each of them. The reductions $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{F}$ use the MF Algorithm, whereas, the reductions $\mathcal{B}$ and $\mathcal{E}$ use the GF Algorithm. Some features of the relevant reductions $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{F}$ are summarised in **Table 4**.

[15]Note that the second criterion is the concrete instance of index-dependence in **Definition 1**. The first criterion, on the other hand, has been already *absorbed* in the corresponding definition of $\mathcal{Y}$ (see Condition B of $\mathbb{A}_2$ and $\mathbb{A}_3$).

[16]Dependence is induced by design in BPW-PSS and was not present in the original GG-IBS. On the other hand, dependence for the CMW-ZKP follows from the logical order.

[17]We have renamed the wrapper $\mathcal{Z}$ in [BPW12] to $\mathcal{Z}'$ for avoiding confusion with the logical wrapper $\mathcal{Z}$ used in the analysis.

Inherent dependence and wrapper design. We have already pointed out the inherent dependence of the hash functions used in BPW-PSS. This inbuilt dependence among the hash functions, along with a careful design of the wrappers, ensures that the MF Algorithm is applied properly. An integral part in the design of the wrappers is the *explicit* check for the logical order among the target random oracle queries. If the order is violated, the wrapper sets a flag and returns $(0, \perp)$. For example, consider the reduction $\mathcal{C}$ and the associated wrapper $\mathcal{Z}'$ from [BPW12]. The check ensures that the indices J and I (that the wrapper returns) *always* correspond to the target query for the random oracles R and H respectively. Moreover, due to dependence, the adversary is bound (except with a negligible probability of guessing) to make the target oracle queries in the logical order, *i.e.* R followed by H. These two factors ensure that the reduction $\mathcal{C}$ will end up with a correct solution to the DLP whenever the MF Algorithm is successful. The same strategy has been followed meticulously in the construction of $\mathcal{D}$ and $\mathcal{F}$ as well. So the notion of index dependence is to some extent used implicitly in the security argument of BPW-PSS.

However, when we come to the MF Algorithm that corresponds to this particular scheme, neither the notion of index dependence nor that of independence between the logical wrapper $\mathcal{Z}$ is considered in the analysis. This brings us to the improved security argument.

4.2.1 Improved Security Argument

The new security argument takes advantage of both the observations O_I and O_D. We have already seen that O_D is applicable due to the existing binding. As for O_I, we again consider the case of reduction $\mathcal{C}$ and its wrapper $\mathcal{Z}'$ due to their relative simplicity. A similar argument works for reductions $\mathcal{D}$ and $\mathcal{F}$ as well.

$\mathcal{Z}'$ is designed to output index I (resp. J) corresponding to the target H (resp. R) query. $\mathcal{C}$ uses the MF Algorithm $\mathcal{M}_{\mathcal{Z}',3}$ to secure a set of four forgeries $\sigma := (z, h, r)$, $\hat{\sigma} := (\hat{z}, \hat{h}, \hat{r})$, $\bar{\sigma} := (\bar{z}, \bar{h}, \bar{r})$ and $\dot{\sigma} := (\dot{z}, \dot{h}, \dot{r})$ with

$$\begin{aligned} z &= v + (r\alpha + y + c\log_g \mathtt{pk}_i)h & \bar{z} &= \bar{v} + (\bar{r}\alpha + y + c\log_g \mathtt{pk}_i)\bar{h} \\ \hat{z} &= v + (r\alpha + y + c\log_g \mathtt{pk}_i)\hat{h} & \dot{z} &= \bar{v} + (\bar{r}\alpha + y + c\log_g \mathtt{pk}_i)\dot{h}. \end{aligned} \tag{23}$$

What we have now is a system of four congruences in four (effective) unknowns $\{\alpha, (y + c\log_g \mathtt{pk}_i), v, \bar{v}\}$ with α being the solution to the DLP. The forgeries σ and $\hat{\sigma}$ (resp. $\bar{\sigma}$ and $\dot{\sigma}$) can be clubbed together as they constitute the output of the the logical wrapper $\mathcal{Z}$ that is associated to wrapper $\mathcal{Z}'$ of $\mathcal{M}_{\mathcal{Z}',3}$. From the structure of the H query, it follows that the index I corresponds to the unknown v. The process of solving for α starts by eliminating the unknown v from each of $\mathcal{Z}$s (*i.e.*, v from $(z, \hat{z})$ and $\bar{v}$ from $(\bar{z}, \dot{z})$). What is necessary at this point is that the I indices must match within $\mathcal{Z}$. The solution is *not* affected by the value of I in the second invocation of $\mathcal{Z}$. In other words, eliminating v from $(z, \hat{z})$ is not affected by the pair $(\bar{z}, \dot{z})$ 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 independence (the system of congruences one ends up with is *exactly* the same as in (23)). In fact, the reduction is unlikely to achieve anything by restricting the indices. Hence, the independence of the I indices is applicable.

Remark 7. The notion of independence 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 secures from the MF Algorithm is a set of $n + 1$ congruences in $n + 1$ unknowns (for some odd n). One of these unknowns is the solution to the hard problem that the reduction wants to solve. The MF Algorithm needs to ensure that the congruences are linearly independent of each other with a certain non-negligible probability. The claim in O_I can then be restated as: even if the condition on the I indices is relaxed as in O_I, we *still* end up with a system of $n + 1$ congruences in $n + 1$ unknowns.

To sum it up, in order to harness both $\mathsf{O_I}$ and $\mathsf{O_D}$, the only change in the security argument of [BPW12] is to use the GMF Algorithm $\mathcal{N}_{\mathbb{A}_3,\mathcal{Y},n}$ (with **Lemma 2**) instead of the original MF Algorithm. The resulting changes are summarised in **Table 4**.

Security Argument	Particulars			
	Reduction	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{F}$
	Oracles involved	R and H	G and H	G and H
Original	Forking Algorithm	$\mathcal{M}_{\mathcal{Z}',3}$	$\mathcal{M}_{\mathcal{U},5}$	$\mathcal{M}_{\mathcal{W},5}$
	Degradation	$\mathsf{O}\left((q_{\mathrm{R}}+q_{\mathrm{H}})^6\right)$	$\mathsf{O}\left((q_{\mathrm{G}}+q_{\mathrm{H}})^{10}\right)$	$\mathsf{O}\left((q_{\mathrm{R}}+q_{\mathrm{H}})^{10}\right)$
Improved	Forking Algorithm	$\mathcal{N}_{\mathbb{A}_3,\mathcal{Z}',3}$	$\mathcal{N}_{\mathbb{A}_3,\mathcal{U},5}$	$\mathcal{N}_{\mathbb{A}_3,\mathcal{W},5}$
	Degradation	$\mathsf{O}\left((q_{\mathrm{R}}+q_{\mathrm{H}})^3\right)$	$\mathsf{O}\left((q_{\mathrm{G}}+q_{\mathrm{H}})^5\right)$	$\mathsf{O}\left((q_{\mathrm{R}}+q_{\mathrm{H}})^5\right)$

Table 4: Comparison of the original and improved security argument for BPW-PSS. q_{G}, q_{R} and q_{H} denote the upper bound on the respective hash oracle queries.

4.3 The Galindo-Garcia Identity-Based Signature

The original construction from [GG09] is reproduced in **Figure 5** of **Appendix E.2** and we refer the reader to for the definition and security model of IBS schemes. Two hash functions are used in the scheme: H and G (both map arbitrary length strings to the set $\mathbb{Z}_p^*$). The structure of hash function calls is given below.

$$c := \mathrm{H}(R\|\mathtt{id}) \qquad d := \mathsf{G}(\mathtt{id}\|A\|m)$$

H is used to generate the user secret key which, in turn, is required to sign on a message (using G). Hence H $<$ G constitutes the logical order for the hash functions. However, *no* binding is in place by construction. The absence of binding (and consequently, the absence of dependence) is precisely the reason for the “incompleteness” of the original security argument that was pointed out in [CKK13]. The incompleteness was addressed in [CKK13] by using the *two-reduction* strategy– give separate reductions for each of the orders of the target H and G calls.

4.3.1 Modified Galindo-Garcia IBS

A better alternative to the *two-reduction* fix is to enforce the logical order on the adversary by binding the G-oracle to the H-oracle.[18] A modified GG-IBS with the aforesaid binding is given in **Appendix E.3**. In short, the structure of the hash function call to G to something like the following:

$$d := \mathsf{G}(\mathtt{id}\|A\|m\|c) \quad \text{where} \quad c := \mathrm{H}(R\|\mathtt{id}).$$

Once the binding is in place, the adversary (except with a negligible probability) is bound to make the target queries in the logical order. In addition, through an argument similar to that of BPW-PSS, one can show that $\mathsf{O_I}$ is also applicable. Accordingly, the security argument for the

[18]A noteworthy observation is that, binding the H-oracle to the G-oracle–*i.e.*, setting up the dependence G $\prec$ H instead of H $\prec$ G–in the GG-IBS allows more efficient reductions to DLP (using general forking). However, this disturbs the logical order of the hash functions. In such a protocol, the PKG will have to issue user secret keys for each message to be signed, rendering it impractical.

modified GG-IBS consists of two reductions $\mathcal{R}'_1$ and $\mathcal{R}'_3$. The core of these reductions remains the same as in $\mathcal{R}_1$ and $\mathcal{R}_3$ respectively of [CKK13]. The only major change is the use of GMF Algorithm $\mathcal{N}_{\mathbb{A}_3,\mathcal{Y},n}$ (with **Lemma 2**) instead of the original MF Algorithm in $\mathcal{R}_3$. For the sake of completeness, we give a detailed description of $\mathcal{R}'_3$ (including the wrapper) in **Appendix F**. The overall effect is summarised in **Table 5**.

Scheme	Security Argument			
GG-IBS [CKK13]	Reduction	$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$
	Forking Algorithm	$\mathcal{F}_{\mathcal{Y}}$	$\mathcal{M}_{\mathcal{Y},1}$	$\mathcal{M}_{\mathcal{Y},3}$
	Degradation	$\mathrm{O}(q_{\mathrm{G}}q_{\varepsilon})$	$\mathrm{O}\left((q_{\mathrm{H}}+q_{\mathrm{G}})^2\right)$	$\mathrm{O}\left((q_{\mathrm{H}}+q_{\mathrm{G}})^6\right)$
Modified GG-IBS (**Figure 5**)	Reduction	$\mathcal{R}'_1$	$\mathcal{R}'_3$	
	Forking Algorithm	$\mathcal{F}_{\mathcal{Y}}$	$\mathcal{N}_{\mathbb{A}_3,\mathcal{Y},3}$	
	Degradation	$\mathrm{O}(q_{\mathrm{G}}q_{\varepsilon})$	$\mathrm{O}\left((q_{\mathrm{H}}+q_{\mathrm{G}})^3\right)$	

Table 5: Degradation for Galindo-Garcia IBS and its variants. q_{G} and q_{H} denote the upper bound on the respective hash oracle queries, whereas, q_{ε} denotes upper bound on the extract queries. We have assumed η to be negligible (see **Remark 5**).

4.4 The Chow-Ma-Weng ZKP for Simultaneous Discrete Logarithms

We confine ourselves to the basic protocol ($n = 2$) which is given in **Figure 7** of **Appendix E.4**. The argument can be easily extended for arbitrary values of n. There are two objects in consideration: the hash function H and the (interactive) commitment-challenge round[19] which we denote by C. The soundness of the CMW-ZKP is based on the hardness of the DLP: the reduction, denoted by $\mathcal{B}$, uses the (original) MF Algorithm $\mathcal{M}_{\mathcal{Y},5}$ to launch a nested replay attack involving H and C.

Lemma 5 (Soundness, *Lemma 4* in [CMW12]). *In the random-oracle model (the hash function H will be modelled as a random oracle), if there exists an adversary $\mathcal{A}$ that can ϵ-break the soundness of the CMW protocol (i.e. $\mathcal{V}$ accepts but $\log_g y_1 \neq \log_g y_2$), there exists an algorithm $\mathcal{B}$ which can ϵ'-solve the DLP with*

$$\epsilon' \geq \epsilon \cdot \left(\frac{\epsilon^5}{(q_H + q_C)^{10}} - \frac{5}{p}\right),$$

where q_H is the number of random oracle query made by $\mathcal{A}$ and q_C is the number of interactions between $\mathcal{A}$ and $\mathcal{B}$.[20]

The notion of binding/dependence for the CMW-ZKP is, interestingly (and contrary to the previous two examples), between a random oracle (H) and an interactive commitment-challenge round (C).

[19]The round of interaction can be replaced with a hash function (also denoted by C) to make the protocol non-interactive [FS87].

[20]We correct a small error in the original expression: the degradation should be by a factor of $(q_{\mathrm{H}} + q_{\mathrm{C}})^{10}$ instead of $(q_{\mathrm{H}} \cdot q_{\mathrm{C}})^{10}$.

4.4.1 A Case for (In)Dependence

Condition $\mathsf{O_I}$. Recall that the commitment v is of the form $(g^z h)^k$, where $z := \mathrm{H}(y_1, y_2)$. By construction, the prover has to compute the value of z before making the commitment and the verifier returns the challenge c only *after* receiving the commitment. Hence the logical order of $\mathrm{H} < \mathrm{C}$. However, it also results in a natural binding between C and H which leads to the dependence $\mathrm{H} \prec \mathrm{C}$.[21] Next, we consider the simulation of the protocol, in particular, the first invocation of $\mathcal{Z}$. At the end of round 0, the adversary produces a cheating transcript (v_1^1, c_1^1, s_1^1), where $v_1^1 := (g^{z_1} h)^{k_1}$. Let (I_0, J_0) be the target indices with J_0 corresponding to the H-oracle output z_1 and I_0 to the commitment v_1^1. round 1 involves forking the adversary at I_0 and let's assume that, at the end of it, the adversary produces another cheating transcript. If we follow the success conditions of the original MF Algorithm, then this particular forking is successful with probability roughly $1/q^2$ because the cheating transcript has to be on the same commitment v_1^1 (*i.e.* $I_1 = I_0$) and, also, on the same H-oracle output z_1 (*i.e.* $J_1 = J_0$).

However, it is easy to observe that, due to the natural *binding* discussed above, an adversary cheating on the commitment v_1^1 (at the end of round 1) has to (except, with a negligible probability of guessing) cheat using the H-oracle output of z_1. In other words, the adversary commits to z_1 *indirectly* through v_1^1. Hence, $(I_1 = I_0)$ *has to* imply $(J_1 = J_0)$ and, as a consequence $\mathrm{H} \prec \mathrm{C}$. The same argument holds for the other two invocations of $\mathcal{Z}$ as well.

Condition $\mathsf{O_D}$. The line of argument is basically similar to the one adopted for the BPW-PSS. Consider the first invocation of $\mathcal{Z}$ in the simulation. For the reduction to successfully solve the DLP, the adversary, over these two rounds, has to produce two cheating transcripts on the *same* commitment (*i.e.* $I_1 = I_0$). This applies to the rounds round k and round k+1, for $k = 2$ and $k = 4$ as well. However, it is not required that the I indices should match *across* these rounds (*i.e.* $(I_4 = I_2 = I_0)$). To see this, consider the effect of relaxing the condition on the simulation which is shown in **Figure 3**.
Even though the I indices across $\mathcal{Z}$ *do not* match (I_0, I_2 and I_4), the set of cheating transcripts that the reduction obtains is still of the form $\{(v_1^i, c_1^i, s_1^i), (v_1^i, c_2^i, s_2^i)\}$, for $i := 1, 2, 3$. The technique used to solve the DLP in (1) still works. Hence, it suffices that I for the cheating transcripts within the two rounds of a particular invocation of $\mathcal{Z}$ match, but not necessarily across the $\mathcal{Z}$s.

4.4.2 Improved Argument

Before commenting on the improved argument, we would like to point out some issues with the design of the wrapper. The authors have made the following observation:

> "The algorithm $\mathcal{Y}$ here is a wrapper that takes as explicit input the answers from the random oracle H and the random challenges given by $\mathcal{B}$, calls $\mathcal{A}$ and returns its output together with two integers I, J. *One of the integers* is the index of $\mathcal{A}$'s calls to the random oracle $\mathrm{H}(\cdot, \cdot)$ and *the other* is the index of the challenge corresponding to the cheat given by $\mathcal{A}$."[emphasis added]

Notice that the correspondence between the indices (I, J) and the target H-oracle call and C-round is not clearly spelt out. The reduction, however, proceeds to solve the DLP under the (implicit) assumption that the wrapper returns a J (resp. I) which refers to the target H-oracle query (resp. C round). If the correspondence is reversed, the reduction will end up computing

[21] Let's consider a resource constrained variant of the verifier which, instead of picking the challenge *upon* receiving a commitment, picks all of the challenges *beforehand*. From the point of view of the prover, the change is purely conceptual. However, the aforesaid logical order, and also the dependence induced by that logical order, no longer holds.

$$\begin{tikzcd}
 & & & & \cdot \arrow[r] & (v_1^1, c_1^1, s_1^1) \;\text{//round 0} \\
 & & \cdot \arrow[r] & \mathsf{Q}^0_{I_0} : \mathsf{C}(v_1^1) \arrow[ur, "c_1^1"] \arrow[dr, "c_2^1"'] & & \\
 & & & & \cdot \arrow[r] & (v_1^1, c_2^1, s_2^1) \;\text{//round 1} \\
 & & & & \cdot \arrow[r] & (v_1^2, c_1^2, s_1^2) \;\text{//round 2} \\
\cdot \arrow[r] & \mathsf{Q}^0_{J_0} : \mathrm{H}(y_1, y_2) \arrow[uuur, "z_1"] \arrow[rr, "z_2"] \arrow[dddr, "z_3"'] & & \mathsf{Q}^2_{I_2} : \mathsf{C}(v_1^2) \arrow[ur, "c_1^2"] \arrow[dr, "c_2^2"'] & & \\
 & & & & \cdot \arrow[r] & (v_1^2, c_2^2, s_2^2) \;\text{//round 3} \\
 & & & & \cdot \arrow[r] & (v_1^2, c_1^2, s_1^2) \;\text{//round 4} \\
 & & \cdot \arrow[r] & \mathsf{Q}^4_{I_4} : \mathsf{C}(v_1^3) \arrow[ur, "c_1^3"] \arrow[dr, "c_2^3"'] & & \\
 & & & & \cdot \arrow[r] & (v_1^3, c_2^3, s_2^3) \;\text{//round 5}
\end{tikzcd}$$

Figure 3: A successful nested replay attack on a CMW-ZKP adversary using $\mathcal{N}_{\mathbb{A}_3,\mathcal{Y},5}$.

an incorrect solution to the DLP.[22] To avoid any ambiguity, we emphasise that the wrapper *should* be explicitly designed to return I (resp. J) which refers to the target C-round (resp. H-query). Now, the MF Algorithm $\mathcal{M}_{\mathcal{Y},5}$ in the original security argument can be replaced with the GMF Algorithm $\mathcal{N}_{\mathbb{A}_3,\mathcal{Y},5}$ (with **Lemma 2**) resulting in the following lemma:

Lemma 6 (Soundness)**.** *In the random-oracle model, if there exists an adversary $\mathcal{A}$ that can ϵ-break the soundness of the CMW protocol, there exists an algorithm $\mathcal{B}$ which can ϵ'-solve the DLP with*

$$\epsilon' = O\left(\frac{\epsilon^6}{(q_H + q_C)^5}\right),$$

where q_H is the number of random oracle query made by $\mathcal{A}$, while q_C is the number of interactions between $\mathcal{A}$ and $\mathcal{B}$.

5 Conclusion

In this paper we have proposed a general framework for the application of the Multiple Forking Lemma. The framework and the corresponding algorithm is derived based on a careful analysis of the original Multiple Forking Lemma and its application in the security argument of various schemes. We prove that our notions of (*in*)dependence significantly improve upon the bound of the original Lemma. We also show that all known instances of the application of the original Multiple Forking Lemma satisfy the notion of (*in*)dependence and hence benefit from our improved bound. Whether the new bounds in the security arguments are optimal or can be improved further will be an interesting open question from a theoretical perspective.

[22]It is, in fact, not difficult to conceive such a wrapper algorithm. Upon receiving the values $\{s_1, \ldots, s_q\}$ as parameters, the wrapper is designed to fix $\{s_1, \ldots, s_{q_c}\}$ as its random challenges before proceeding with the actual simulation. This leads to the wrapper *always* returning a J index that refers to the target C-round (and an I index that refers to the target H-query). The "artificial" design of the wrapper has the same effect as that of an adversary making the target calls in the *wrong* order. Hence, in spite of the simulation being faithful (from the standpoint of the adversary) and the MF Algorithm returning success, the strategy adopted for solving DLP will *fail*. Interestingly, this strategy corresponds to the "resource constrained" verifier that we had discussed in **Footnote 21**.

References

[BCJ08] Ali Bagherzandi, Jung-Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 449–458, New York, NY, USA, 2008. ACM. (Cited on page 3.)

[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 page 10.)

[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 3, 4, 5, 8, 14, 15, 26 and 27.)

[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 3, 4, 5, 6, 7, 9, 10, 13, 16, 18, 19, 20, 21, 27, 28 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 page 3.)

[CKK13] Sanjit Chatterjee, Chethan Kamath, and Vikas Kumar. Galindo-Garcia identity-based signature revisited. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology - ICISC 2012*, volume 7839 of *Lecture Notes in Computer Science*, pages 456–471. Springer Berlin / Heidelberg, 2013. Full version available in Cryptology ePrint Archive, Report 2012/646, http://eprint.iacr.org/2012/646. (Cited on pages 3, 5, 21, 22 and 39.)

[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 pages 3, 5, 6, 7, 9, 10, 18 and 22.)

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 89–105, London, UK, UK, 1993. Springer-Verlag. (Cited on page 38.)

[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 3.)

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Berlin / Heidelberg, 1987. (Cited on pages 3, 19 and 22.)

[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 3, 5, 6, 9, 10, 18 and 21.)

[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 3.)

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

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991. 10.1007/BF00196725. (Cited on page 3.)

[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 4.)

[YADV$^+$12] Sidi-Mohamed Yousfi-Alaoui, Özgür Dagdelen, Pascal Véron, David Galindo, and Pierre-Louis Cayrel. Extended security arguments for signature schemes. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012*, volume 7374 of *Lecture Notes in Computer Science*, pages 19–34. Springer Berlin Heidelberg, 2012. (Cited on page 6.)

[YZ13] Andrew Chi-Chih Yao and Yunlei Zhao. Online/offline signatures for low-power devices. *IEEE Transactions on Information Forensics and Security*, 8(2):283–294, 2013. (Cited on page 6.)

A Notations

We adopt the notations commonly used in the literature. $s \overset{\text{U}}{\leftarrow} \mathbb{S}$ denotes picking an element s uniformly at random from the set $\mathbb{S}$. In general, $\{s_1, \ldots, s_n\} \overset{\text{U}}{\leftarrow} \mathbb{S}$ denotes picking elements $s_1, \ldots, s_n$ independently and uniformly at random from the set $\mathbb{S}$. In a similar manner, $s \overset{\$}{\leftarrow} \mathbb{S}$ and $\{s_1, \ldots, s_n\} \overset{\$}{\leftarrow} \mathbb{S}$ denote random sampling, but with some underlying probability distribution on $\mathbb{S}$. $(y_1, \ldots, y_n) \overset{\$}{\leftarrow} \mathcal{A}(x_1, \ldots, x_m)$ denotes a probabilistic algorithm $\mathcal{A}$ which takes as input $(x_1, \ldots, x_m)$ to produce output $(y_1, \ldots, y_n)$. Sometimes the internal coins ρ of this algorithm is given explicitly as an input. This is distinguished from the normal input using a semi-colon, *e.g.*, $y \leftarrow \mathcal{A}(x; \rho)$.

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

B General Forking

We reproduce the GF Algorithm from [BN06] followed by the statement of the GF lemma. We use slightly different notations to maintain uniformity.

Forking Algorithm. Fix $q \in \mathbb{Z}^+$ and a set $\mathbb{S}$ such that $|\mathbb{S}| \geq 2$. Let $\mathcal{Y}$ be a randomised algorithm that on input a string x and elements $s_1, \ldots, s_q \in \mathbb{S}$ returns a pair (I, σ) consisting of an integer $0 \leq I \leq q$ and a string σ. The forking algorithm $\mathcal{F}_{\mathcal{Y}}$ associated to $\mathcal{Y}$ is defined as **Algorithm 3** below.

Algorithm 3 $\mathcal{F}_{\mathcal{Y}}(x)$

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

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

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

Lemma 7 (General Forking Lemma [BN06]). *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

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

then

$$gfrk \geq acc \cdot \left(\frac{acc}{q} - \frac{1}{|\mathbb{S}|}\right). \tag{24}$$

C (Original) Multiple-Forking Algorithm

We describe the original Multiple-Forking Algorithm [BPW12] with some notational changes. The success condition and the lemma that governs the success probability follows the algorithm.

The Multiple-Forking Algorithm Fix $q \in \mathbb{Z}^+$ and a set $\mathbb{S}$ such that $|\mathbb{S}| \geq 2$. Let $\mathcal{Y}$ be a randomised algorithm that on input a string x and elements $s_1, \ldots, s_q \in \mathbb{S}$ 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 MF Algorithm $\mathcal{M}_{\mathcal{Y},n}$ associated to $\mathcal{Y}$ and n is defined as **Algorithm 4** below.

Algorithm 4 $\mathcal{M}_{\mathcal{Y},n}(x)$

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

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

$\{s_{I_0}^1,\ldots,s_q^1\} \xleftarrow{U} \mathbb{S}$;
$(I_1,J_1,\sigma_1) \leftarrow \mathcal{Y}(x,s_1^0,\ldots,s_{I_0-1}^0,s_{I_0}^1,\ldots,s_q^1;\rho)$  //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 $\neg$C$_0$

$k:=2$
while $(k<n)$ do
    $\{s_{J_0}^k,\ldots,s_q^k\} \xleftarrow{U} \mathbb{S}$;
    $(I_k,J_k,\sigma_k) \leftarrow \mathcal{Y}(x,s_1^0,\ldots,s_{J_0-1}^0,s_{J_0}^k,\ldots,s_q^k;\rho)$  //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 $\neg$D$_k$

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

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

The success condition. The success of the MF Algorithm is determined by the set of conditions $\mathbb{A}_0 := \{\mathsf{B},\mathsf{C}_0,\ldots,\mathsf{C}_{n-1},\mathsf{C}_2,\ldots,\mathsf{D}_{n-1}\}$ where

$$\begin{aligned}
&\mathsf{B} : (I_0\geq 1)\wedge(J_0\geq 1)\\
&\mathsf{C}_k : (I_{k+1},J_{k+1})=(I_k,J_k)\wedge(s_{I_k}^{k+1}\neq s_{I_k}^k)\ \text{ (for } k=0,2,\ldots,n-1)\\
&\mathsf{D}_k : (I_k,J_k)=(I_0,J_0)\wedge(s_{J_0}^k\neq s_{J_0}^{k-1})\ \text{ (for } k=2,4,\ldots,n-1)
\end{aligned}\tag{25}$$

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.*,

$$\mathsf{E} : \mathsf{B}\wedge\big(\mathsf{C}_0\wedge\mathsf{C}_2\wedge\cdots\wedge\mathsf{C}_{n-1}\big)\wedge\big(\mathsf{D}_2\wedge\mathsf{D}_4\wedge\cdots\wedge\mathsf{D}_{n-1}\big)\,. \tag{26}$$

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

Lemma 8 ((Original) Multiple-Forking Lemma [BPW12]). *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

$$\mathit{mfrk} := \Pr\Big[(b=1)\mid x\xleftarrow{\$}\mathcal{G}_I;(b,\{\sigma_0,\ldots,\sigma_n\})\xleftarrow{\$}\mathcal{M}_{\mathcal{Y},n}(x)\Big]\quad \textit{and}$$

$$\mathit{acc} := \Pr\Big[(I\geq 1)\wedge(J\geq 1)\mid x\xleftarrow{\$}\mathcal{G}_I;\{s_1,\ldots,s_q\}\xleftarrow{U}\mathbb{S};(I,J,\sigma)\xleftarrow{\$}\mathcal{Y}(x,s_1,\ldots,s_q)\Big]$$

then

$$\mathit{mfrk}\geq \mathit{acc}\cdot\left(\frac{\mathit{acc}^n}{q^{2n}}-\frac{n}{|\mathbb{S}|}\right). \tag{27}$$

D Harnessing (In)Dependence

D.1 Detailed Steps for Lemma 2

Intermediate steps for (13)

$$
\begin{aligned}
&\mathit{frk}_c(x)\\
&= \Pr\left[(I_1 = I_0) \wedge (J_1 = J_0) \wedge (1 \le J_0 < I_0 \le q)\right]\\
&= \Pr\left[(J_1 = J_0) \mid (I_1 = I_0) \wedge (1 \le J_0 < I_0 \le q)\right] \Pr\left[(I_1 = I_0) \wedge (1 \le J_0 < I_0 \le q)\right]\\
&= (1-\eta) \Pr\left[(I_1 = I_0) \wedge (1 \le J_0 < I_0 \le q)\right]\\
&= (1-\eta) \sum_{i=2}^{q} \Pr\left[(I_1 = i) \wedge (1 \le J_1 < i) \mid (I_0 = i) \wedge (1 \le J_0 < i)\right] \cdot \Pr\left[(I_0 = i) \wedge (1 \le J_0 < i)\right]\\
&= (1-\eta) \sum_{i=2}^{q} \sum_{\rho, \mathsf{s}_{(i-)}} \frac{1}{|\mathbb{R}||\mathbb{S}|^{j-1}} Y_i^2(\rho, \mathsf{s}_{(i-)})\\
&= (1-\eta) \sum_{i=2}^{q} \mathrm{Ex}\left[Y_i^2\right] \ge \sum_{i=2}^{q} \mathrm{Ex}\left[Y_i\right]^2 \quad \text{(by Jensen's inequality)}\\
&\ge \frac{(1-\eta)}{q} \left(\sum_{i=2}^{q} \mathrm{Ex}\left[Y_i\right] \right)^2 \quad \text{(by Hölder's inequality)}\\
&= \frac{(1-\eta)}{q} \mathit{acc}(x)^2 \quad \text{(by definition of } Y_i \text{ and } \mathit{acc}(x)\text{)}
\end{aligned}
\tag{28}
$$

Intermediate steps for (17)

$$
\begin{aligned}
&\mathit{mfrk}_{3,c}(x)\\
&= \Pr\big[(1 \le J_0 < I_0 \le q) \wedge (I_1 = I_0) \wedge (s^1_{I_0} \ne s^0_{I_0}) \wedge\\
&\qquad \big(\wedge_{k=2,4,\dots,n-1} (I_{k+1} = I_k) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k}) \wedge (J_k = J_0) \wedge (J_k < I_k \le q)\big)\big]\\
&= \Pr\big[(1 \le J_0 < I_0 \le q) \wedge (I_1 = I_0) \wedge (J_1 = J_0) \wedge (s^1_{I_0} \ne s^0_{I_0}) \wedge\\
&\qquad \big(\wedge_{k=2,4,\dots,n-1} (I_{k+1} = I_k) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k}) \wedge (J_{k+1} = J_k = J_0) \wedge (J_k < I_k \le q)\big)\big] \quad \text{(since } J \prec I\text{)}\\
&= \sum_{j=1}^{q-1} \Pr\big[\wedge_{k=0,2,\dots,n-1} ((J_{k+1} = J_k = j) \wedge (I_{k+1} = I_k) \wedge (j < I_k \le q) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k}))\big]\\
&= \sum_{j=1}^{q-1} \sum_{\rho, \mathsf{s}_{(j-)}} \frac{\Pr\big[\wedge_{k=0,2,\dots,n-1} (J_{k+1} = J_k = j) \wedge (I_{k+1} = I_k) \wedge (j < I_k \le q) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k})\big]}{|\mathbb{R}||\mathbb{S}|^{j-1}}\\
&= \sum_{j=1}^{q-1} \sum_{\rho, \mathsf{s}_{(j-)}} \frac{\prod_{k=0,2}^{n-1} \Pr\big[(J_{k+1} = J_k = j) \wedge (I_{k+1} = I_k) \wedge (j < I_k \le q) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k}) \mid \bigwedge_{l=0,2}^{k-2} \mathsf{G}_\ell\big]}{|\mathbb{R}||\mathbb{S}|^{j-1}}
\end{aligned}
\tag{29}
$$

In the above expression, G_ℓ denotes the event

$$((J_{\ell+1} = J_\ell = j) \wedge (I_{\ell+1}, J_{\ell+1}) = (I_\ell, J_\ell) \wedge (j < I_\ell \le q) \wedge (s^{\ell+1}_{I_\ell} \ne s^\ell_{I_\ell}).$$

Using the random variable Z_j, the equation (17) can be rewritten as

$$\begin{aligned}
&\mathit{mfrk}_{3,c}(x) \\
&= \sum_{j=1}^{q-1} \sum_{\rho, \mathsf{s}_{(j-)}} \frac{\prod_{k=0,2}^{n-1} Z_j(\rho, \mathsf{s}_{(j-)})}{|\mathbb{R}||\mathbb{S}|^{j-1}} = \sum_{j=1}^{q-1} \left(\sum_{\rho, \mathsf{s}_{(j-)}} \frac{Z_j^{(n+1)/2}(\rho, \mathsf{s}_{(j-)})}{|\mathbb{R}||\mathbb{S}|^{j-1}} \right) \\
&= \sum_{j=1}^{q-1} \mathrm{Ex}\left[Z_j^{(n+1)/2}\right] \geq \sum_{j=1}^{q-1} \mathrm{Ex}\left[Z_j\right]^{(n+1)/2} \quad \text{(by Jensen's inequality)} \\
&\geq \frac{1}{q^{(n-1)/2}} \left(\sum_{j=1}^{q-1} \mathrm{Ex}\left[Z_j\right] \right)^{(n+1)/2} \qquad \text{(by Jensen's and Hölder's inequality)} \qquad (30)
\end{aligned}$$

Intermediate steps for (18)

$$\begin{aligned}
&\mathit{mfrk}_{3,s}(x) = \\
&= \Pr\left[(1 \leq J_0 < I_0 \leq q) \wedge (I_1 = I_0) \wedge (s^1_{I_0} \neq s^0_{I_0})\right] \cdot \Pr\left[\left(\vee_{k=2,4,\ldots,n-1} \neg \mathsf{D}_{k,s}\right)\right] \\
&= \frac{(n-1)(n+1)}{8|\mathbb{S}|} \cdot \Pr\left[(1 \leq J_0 < I_0 \leq q) \wedge (I_1 = I_0) \wedge (J_1 = J_0) \wedge (s^1_{I_0} \neq s^0_{I_0})\right] \\
&= \frac{(n-1)(n+1)}{8|\mathbb{S}|} \left(\sum_{j=1}^{q-1} (J_1 = J_0 = j) \wedge (I_1 = I_0) \wedge (j < I_0 \leq q) \wedge (s^1_{I_0} \neq s^0_{I_0}) \right) \\
&= \frac{(n-1)(n+1)}{8|\mathbb{S}|} \left(\sum_{j=1}^{q-1} \mathrm{Ex}\left[Z_j\right] \right) \qquad (31)
\end{aligned}$$

D.2 Multiple-Forking with Index Independence

Lemma 9 (Multiple-Forking Lemma with Index Independence). *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

$$\mathit{mfrk}_1 := \Pr\left[(\mathsf{b} = 1) \mid x \xleftarrow{\$} \mathcal{G}_I; (\mathsf{b}, \{\sigma_0, \ldots, \sigma_n\}) \xleftarrow{\$} \mathcal{N}_{\mathbb{A}_1, \mathcal{Y}, n}(x)\right] \quad \textit{and}$$

$$\mathit{acc} := \Pr\left[(I \geq 1) \wedge (J \geq 1) \mid x \xleftarrow{\$} \mathcal{G}_I; \{s_1, \ldots, s_q\} \xleftarrow{\mathrm{U}} \mathbb{S}; (I, J, \sigma) \xleftarrow{\$} \mathcal{Y}(x, s_1, \ldots, s_q)\right]$$

then

$$\mathit{mfrk}_1 \geq \mathit{acc} \cdot \left(\frac{\mathit{acc}^n}{q^{(3n+1)/2}} - \frac{(n+1)(n+3)}{8|\mathbb{S}|} \right). \qquad (32)$$

Proof. The analysis, especially its logical flow, is quite similar to the original analysis. We stick to the conventions adopted in §**3.1**. For a fixed string x, let

$$\mathit{mfrk}_1(x) := \Pr\left[(\mathsf{b} = 1) \mid (\mathsf{b}, \{\sigma_0, \ldots, \sigma_n\}) \xleftarrow{\$} \mathcal{N}_{\mathbb{A}_1, \mathcal{Y}, n}(x)\right] \quad \text{and}$$

$$\mathit{acc}(x) := \Pr\left[(I \geq 1) \wedge (J \geq 1) \mid \{s_1, \ldots, s_q\} \xleftarrow{\mathrm{U}} \mathbb{S}; (I, J, \sigma) \xleftarrow{\$} \mathcal{Y}(x, s_1, \ldots, s_q)\right]$$

Let $\mathbb{A}_1 := \{\mathsf{B}, \mathsf{C}_0, \mathsf{C}_2, \ldots, \mathsf{C}_{n-1}, \mathsf{D}_2, \mathsf{D}_4, \ldots, \mathsf{D}_{n-1}\}$. For ease of notation, we further break the event C_k (resp. D_k) into two subevents $\mathsf{C}_{k,c}$ and $\mathsf{C}_{k,s}$ (resp. $\mathsf{D}_{k,c}$ and $\mathsf{D}_{k,s}$) as follows:

$$\begin{aligned}
\mathsf{C}_{k,c} : (I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (I_k \geq 1) \quad & \mathsf{C}_{k,s} : (s^{k+1}_{I_k} \neq s^k_{I_k}) \\
\mathsf{D}_{k,c} : (J_k = J_0) \quad & \mathsf{D}_{k,s} : (\wedge_{\ell:=0,2,\ldots,k-2}\, s^k_{J_0} \neq s^\ell_{J_0}) \qquad (33)
\end{aligned}$$

The GMF Algorithm is successful in the event $\mathsf{E} : \mathsf{B} \wedge (\mathsf{C}_0 \wedge \mathsf{C}_2 \wedge \cdots \wedge \mathsf{C}_{n-1}) \wedge (\mathsf{D}_2 \wedge \mathsf{D}_4 \wedge \cdots \wedge \mathsf{D}_{n-1})$. In other words, with the probabilities calculated over the randomness of the GMF Algorithm, it follows that $\mathit{mfrk}_1(x) = \Pr[\mathsf{E}]$. The first step in calculating the probability is to separate the "core" subevents out of the event E. This is accomplished as follows.

$$\begin{aligned}
\Pr[\mathsf{E}] &= \Pr\left[\mathsf{B} \wedge \left(\wedge_{k=0,2,\ldots,n-1} \mathsf{C}_k\right) \wedge \left(\wedge_{k=2,4,\ldots,n-1} \mathsf{D}_k\right)\right] \\
&= \Pr\left[\mathsf{B} \wedge \left(\wedge_{k=0,2,\ldots,n-1} \mathsf{C}_{k,c} \wedge \mathsf{C}_{k,s}\right) \wedge \left(\wedge_{k=2,4,\ldots,n-1} \mathsf{D}_{k,c} \wedge \mathsf{D}_{k,s}\right)\right] \quad \text{(using (33))} \\
&\geq \Pr\left[\mathsf{B} \wedge \left(\wedge_{k=0,2,\ldots,n-1} \mathsf{C}_{k,c}\right) \wedge \left(\wedge_{k=2,4,\ldots,n-1} \mathsf{D}_{k,c}\right)\right] - \\
&\quad \Pr\left[\mathsf{B} \wedge \left(\left(\vee_{k=0,2,\ldots,n-1} \mathsf{C}_{k,s}\right) \vee \left(\vee_{k=2,4,\ldots,n-1} \mathsf{D}_{k,s}\right)\right)\right] \qquad (34)
\end{aligned}$$

It can be shown that the second part of (34) equals

$$\frac{(n+1)(n+3) \cdot \mathit{acc}(x)}{8|\mathbb{S}|}$$

by following the analysis in (18). As for the first part, it constitutes the "core" event for the analysis, and is denoted by $\mathit{mfrk}_{1,c}(x)$. The event corresponding to $\mathit{mfrk}_{1,c}(x)$ is closely related to the event given in (6). The next step is to show that

$$\mathit{mfrk}_{1,c}(x) \geq \frac{\mathit{acc}(x)^{n+1}}{q^{(3n+1)/2}}. \qquad (35)$$

The intermediate steps to achieving it are as follows:

$$\begin{aligned}
\mathit{mfrk}_{1,c}(x) &= \Pr\left[\mathsf{B} \wedge \left(\wedge_{k=0,2,\ldots,n-1} \mathsf{C}_{k,c}\right) \wedge \left(\wedge_{k=2,4,\ldots,n-1} \mathsf{D}_{k,c}\right)\right] \\
&= \Pr\left[(I_0 \geq 1) \wedge (J_0 \geq 1) \wedge \bigwedge_{k=0,2}^{n-1} ((I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (I_k \geq 1)) \wedge \bigwedge_{k=2,4}^{n-1} (J_k = J_0)\right] \\
&= \sum_{j=1}^{q} \Pr\left[(I_0 \geq 1) \wedge (J_0 = j) \wedge \bigwedge_{k=0,2}^{n-1} ((I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (I_k \geq 1)) \wedge \bigwedge_{k=2,4}^{n-1} (J_k = j)\right] \\
&= \sum_{j=1}^{q} \sum_{\rho, \mathsf{s}_{(j-)}} \frac{\prod_{k=0,2}^{n-1} \Pr\left[(I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (J_k = j) \wedge (I_k \geq 1) \mid \bigwedge_{l=2,4}^{k-2} \mathsf{G}_\ell\right]}{|\mathbb{R}||\mathbb{S}|^{j-1}}
\end{aligned}$$

(36)

In the above expression, G_ℓ denotes the event $(I_{\ell+1}, J_{\ell+1}) = (I_\ell, J_\ell) \wedge (J_\ell = j) \wedge (I_\ell \geq 1)$. Let's focus on the probability part of (36).

$$\begin{aligned}
&\Pr\left[(I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (J_k = j) \wedge (I_k \geq 1)\right] \\
&= \sum_{i=1}^{q} \Pr\left[(I_{k+1}, J_{k+1}) = (I_k, J_k) = (i, j)\right] \\
&= \sum_{i=1}^{q} \sum_{\mathsf{s}_{(ji)}} \frac{\Pr\left[(I_{k+1}, J_{k+1}) = (I_k, J_k) = (i, j)\right]}{|\mathbb{S}|^{i-j}} \\
&= \sum_{i=1}^{q} \sum_{\mathsf{s}_{(ji)}} \frac{\Pr\left[(I_{k+1}, J_{k+1}) = (i, j)\right] \cdot \Pr\left[(I_k, J_k) = (i, j)\right]}{|\mathbb{S}|^{i-j}} \qquad (37)
\end{aligned}$$

At this point, we define a random variable $Y_{i,j,\rho,\mathsf{s}_{(j-)}}$ which captures a single invocation of the wrapper $\mathcal{Y}$, but with the internal randomness ρ and the external randomness $\mathsf{s}_{(j-)}$ being fixed. Let $Y_{i,j,\rho,\mathsf{s}_{(j-)}} : \mathbb{S}^{i-j} \mapsto [0,1]$, for each $i, j \in \{1, \ldots, q\}$, be defined by setting

$$Y_{i,j,\rho,\mathsf{s}_{(j-)}}(\mathsf{s}_{(ji)}) := \Pr\left[(I, J) = (i, j) \mid \mathsf{s}_{(i+)} \xleftarrow{\$} \mathbb{S}^{q-i+1}; (I, J, \sigma) \leftarrow \mathcal{Y}(x, \mathsf{s}_{(j-)}, \mathsf{s}_{(ji)}, \mathsf{s}_{(i+)}; \rho)\right]$$

Using the random variable Y, the equation (37) can be rewritten as

$$\begin{aligned}
&\sum_{i=1}^{q}\sum_{\mathrm{S}_{(ji)}} \frac{Y^2_{i,j,\rho,\mathrm{S}_{(j-)}}(\mathrm{S}_{(ji)})}{|\mathbb{S}|^{i-j}} \\
&= \sum_{i=1}^{q} \mathrm{Ex}\left[Y^2_{i,j,\rho,\mathrm{S}_{(j-)}}\right] \geq \sum_{i=1}^{q} \mathrm{Ex}\left[Y_{i,j,\rho,\mathrm{S}_{(j-)}}\right]^2 \quad \text{(by Jensen's inequality)} \\
&\geq \frac{1}{q}\left(\sum_{i=1}^{q} \mathrm{Ex}\left[Y_{i,j,\rho,\mathrm{S}_{(j-)}}\right]\right)^2 \quad \text{(by Hölder's inequality)} \qquad (38)
\end{aligned}$$

Next, we define a random variable Z_j which captures one invocation of the logical wrapper $\mathcal{Z}$. Let $Z_j : \mathbb{R} \times \mathbb{S}^{j-1} \mapsto [0,1]$, for each $j \in \{1,\dots,q\}$, be defined by setting

$$Z_j(\rho, \mathrm{S}_{(j-)}) := \sum_{i=1}^{q} \mathrm{Ex}\left[Y_{i,j,\rho,\mathrm{S}_{(j-)}}\right].$$

Hence, on representing (38) in terms of the random variable Z_j, we get

$$\Pr\left[(I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (J_k = j) \wedge (I_k \geq 1)\right] \geq Z_j(\rho, \mathrm{S}_{(j-)})^2/q$$

Substituting the above expression, further, in (36) yields

$$\begin{aligned}
&\sum_{j=1}^{q}\sum_{\rho,\mathrm{S}_{(j-)}} \frac{\prod_{k=0,2}^{n-1}\Pr\left[(I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (J_k = j) \wedge (I_k \geq 1)\right]}{|\mathbb{R}||\mathbb{S}|^{j-1}} \\
&\geq \sum_{j=1}^{q}\sum_{\rho,\mathrm{S}_{(j-)}} \frac{\left(Z_j(\rho,\mathrm{S}_{(j-)})^2/q\right)^{(n+1)/2}}{|\mathbb{R}||\mathbb{S}|^{j-1}} \\
&= \frac{1}{q^{(n+1)/2}}\sum_{j=1}^{q}\sum_{\rho,\mathrm{S}_{(j-)}} \frac{Z_j(\rho,\mathrm{S}_{(j-)})^{n+1}}{|\mathbb{R}||\mathbb{S}|^{j-1}} \\
&= \frac{1}{q^{(n+1)/2}}\sum_{j=1}^{q}\mathrm{Ex}\left[Z_j^{n+1}\right] \geq \frac{1}{q^{(n+1)/2}}\sum_{j=1}^{q}\mathrm{Ex}\left[Z_j\right]^{n+1} \quad \text{(by Jensen's inequality)} \\
&\geq \frac{1}{q^{(n+1)/2}\cdot q^n}\left(\sum_{j=1}^{q}\mathrm{Ex}\left[Z_j\right]\right)^{n+1} \quad \text{(by Hölder's inequality)} \\
&= \frac{1}{q^{(3n+1)/2}}\mathit{acc}(x)^{n+1}.
\end{aligned}$$

That completes the analysis of the "core" event and establishes our initial claim in (35). On combining the two parts of the equation (34), we get

$$\begin{aligned}
\mathit{mfrk}_1(x) &\geq \frac{\mathit{acc}(x)^{n+1}}{q^{(3n+1)/2}} - \frac{(n+1)(n+3)\cdot \mathit{acc}(x)}{8|\mathbb{S}|} \\
&= \mathit{acc}(x)\left(\frac{\mathit{acc}(x)^n}{q^{(3n+1)/2}} - \frac{(n+1)(n+3)}{8|\mathbb{S}|}\right).
\end{aligned}$$

With expectation taken over $x \xleftarrow{\$} \mathcal{G}_I$,

$$\mathit{mfrk}_1 \geq \mathit{acc}\left(\frac{\mathit{acc}^n}{q^{(3n+1)/2}} - \frac{(n+1)(n+3)}{8|\mathbb{S}|}\right).$$

hence, proving the lemma. We conclude with the comment that on assuming $|\mathbb{S}| \gg 1$, one gets $\mathit{mfrk}_2 \approx \mathit{acc}^{n+1}/q^{(3n1)/2}$. $\square$

D.3 Multiple-Forking with Index Dependence

Lemma 10 (Multiple-Forking Lemma with Index Dependence). *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

$$mfrk := \Pr\left[(b=1) \mid x \xleftarrow{\$} \mathcal{G}_I; (b, \{\sigma_0, \ldots, \sigma_n\}) \xleftarrow{\$} \mathcal{N}_{\mathbb{A}_0, \mathcal{Y}, n}(x)\right] \text{ and}$$
$$acc := \Pr\left[(1 \le J < I \le q) \mid x \xleftarrow{\$} \mathcal{G}_I; \{s_1, \ldots, s_q\} \xleftarrow{\text{U}} \mathbb{S}; (I, J, \sigma) \xleftarrow{\$} \mathcal{Y}(x, s_1, \ldots, s_q)\right]$$

On the assumption that J is η-dependent on I,

$$mfrk \ge frk\left(\frac{frk^{(n-1)/2}}{q^n} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\right) \text{ where } frk \ge acc\left(\frac{acc}{q} - \frac{1}{|\mathbb{S}|}\right) \tag{39}$$

Proof. For a fixed string x, let

$$mfrk(x) := \Pr\left[(\mathsf{b}=1) \mid (\mathsf{b}, \{\sigma_0, \ldots, \sigma_n\}) \xleftarrow{\$} \mathcal{N}_{\mathbb{A}_0, \mathcal{Y}, (}x)\right] \text{ and}$$
$$acc(x) := \Pr\left[(1 \le J < I \le q) \mid \{s_1, \ldots, s_q\} \xleftarrow{\text{U}} \mathbb{S}; (I, J, \sigma) \xleftarrow{\$} \mathcal{Y}(x, s_1, \ldots, s_q)\right].$$

Let $\mathbb{A}_1 := \{\mathsf{B}, \mathsf{C}_0, \mathsf{C}_2, \ldots, \mathsf{C}_{n-1}, \mathsf{D}_2, \mathsf{D}_4, \ldots, \mathsf{D}_{n-1}\}$. For ease of notation, we further break the event C_k (resp. D_k) into two subevents $\mathsf{C}_{k,c}$ and $\mathsf{C}_{k,s}$ (resp. $\mathsf{D}_{k,c}$ and $\mathsf{D}_{k,s}$) as follows:

$$\begin{aligned} &\mathsf{C}_{k,c} : (I_{k+1} = I_k) \quad && \mathsf{C}_{k,s} : (s^{k+1}_{I_k} \ne s^k_{I_k}) \\ &\mathsf{D}_{k,c} : (I_k, J_k) = (I_0, J_0) \quad && \mathsf{D}_{k,s} : (\wedge_{\ell:=0,2,\ldots,k-2}\ s^k_{J_0} \ne s^\ell_{J_0}) \end{aligned} \tag{40}$$

The GMF Algorithm is successful in the event $\mathsf{E} : \mathsf{B} \wedge (\mathsf{C}_0 \wedge \mathsf{C}_2 \wedge \cdots \wedge \mathsf{C}_{n-1}) \wedge (\mathsf{D}_2 \wedge \mathsf{D}_4 \wedge \cdots \wedge \mathsf{D}_{n-1})$. In other words, with the probability calculated over the randomness of the algorithm GMF Algorithm, it follows that $mfrk_2(x) = \Pr[\mathsf{E}]$. The task of bounding this probability is accomplished through three claims (**Claim 1**, **Claim 5** and **Claim 6**). We reuse the bound on $frk(x)$ given in **Claim 1**. In order to establish **Claim 5** and **Claim 6** we have to define a random variable $Z_{i,j}$ that captures a single invocation of the logical wrapper $\mathcal{Z}$. Let $Z_{i,j} : \mathbb{R} \times \mathbb{S}^{j-1} \mapsto [0,1]$, for $1 \le j \le q-1$ and $j < i \le q$, be defined by setting

$$Z_{i,j}(\rho, \mathsf{S}_{(j-)}) = \Pr\left[(J' = J = j) \wedge (I' = I = i) \wedge (s'_I \ne s_I)\right]$$

given

$$(\mathsf{S}_{(ji)}, (\mathsf{S}_{(i+)}, \mathsf{S}'_{(i+)})) \xleftarrow{\text{U}} \mathbb{S} \text{ and}$$
$$(I, J, \sigma), (I', J', \sigma')) \leftarrow \mathcal{Z}(x, \{\mathsf{S}_{(j-)}, \mathsf{S}_{(ji)}, \mathsf{S}_{(i+)}\}, \{\mathsf{S}_{(j-)}, \mathsf{S}_{(ji)}, \mathsf{S}'_{(i+)}\}; \rho).$$

Briefly, our aim is to bound $mfrk_2(x)$ in terms of $Z_{i,j}$ (**Claim 5**) and bound $Z_{i,j}$ in terms of $frk(x)$ (**Claim 6**).

Claim 5.

$$mfrk_2(x) \ge \frac{1}{q^{n-1}}\left(\sum_{j=1}^{q-1}\sum_{i=j+1}^{q} Ex[Z_{i,j}]\right)^{(n+1)/2} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\left(\sum_{j=1}^{q-1}\sum_{i=j+1}^{q} Ex[Z_{i,j}]\right) \tag{41}$$

Argument. The first step is to separate the "core" subevents out of the event E as shown below.

$$\begin{aligned} \Pr[\mathsf{E}] &= \Pr\left[\mathsf{B} \wedge (\wedge_{k=0,2,\ldots,n-1} \mathsf{C}_k) \wedge (\wedge_{k=2,4,\ldots,n-1} \mathsf{D}_k)\right] \\ &= \Pr\left[\mathsf{B} \wedge \mathsf{C}_0 \wedge (\wedge_{k=2,4,\ldots,n-1} \mathsf{C}_k \wedge \mathsf{D}_{k,c} \wedge \mathsf{D}_{k,s})\right] \text{ (using subevents given in (33))} \\ &= \Pr\left[\mathsf{B} \wedge \mathsf{C}_0 \wedge (\wedge_{k=2,4,\ldots,n-1} \mathsf{C}_k \wedge \mathsf{D}_{k,c}) \wedge (\wedge_{k=2,4,\ldots,n-1} \mathsf{D}_{k,s})\right] \\ &\ge \Pr\left[\mathsf{B} \wedge \mathsf{C}_0 \wedge (\wedge_{k=2,4,\ldots,n-1} \mathsf{C}_k \wedge \mathsf{D}_{k,c})\right] - \Pr\left[\mathsf{B} \wedge \mathsf{C}_0 \wedge (\vee_{k=2,4,\ldots,n-1} \mathsf{D}_{k,s})\right] \end{aligned} \tag{42}$$

We denote the first part of (42) by *mfrk*$_{2,c}(x)$ and the second part by *mfrk*$_{2,s}(x)$ and analyse them separately.

$$\begin{aligned}
mfrk_{2,c}(x) &= \Pr\left[\mathsf{B} \wedge \mathsf{C}_0 \wedge \left(\wedge_{k=2,4,\ldots,n-1} \mathsf{C}_k \wedge \mathsf{D}_{k,c}\right)\right] \\
&= \Pr\left[\mathsf{B} \wedge \left(\wedge_{k=0,2,\ldots,n-1} \mathsf{C}_k\right) \wedge \left(\wedge_{k=2,4,\ldots,n-1} \mathsf{D}_{k,c}\right)\right] \\
&= \Pr\left[(1 \le J_0 < I_0 \le q) \wedge \left(\wedge_{k=0,2,\ldots,n-1} (I_{k+1} = I_k) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k})\right) \wedge \right. \\
&\qquad \left. \left(\wedge_{k=2,4,\ldots,n-1} (I_k, J_k) = (I_0, J_0)\right)\right] \\
&= \Pr\left[(1 \le J_0 < I_0 \le q) \wedge \left(\wedge_{k=0,2,\ldots,n-1} (I_{k+1}, J_{k+1}) = (I_k, J_k) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k})\right) \wedge \right. \\
&\qquad \left. \left(\wedge_{k=2,4,\ldots,n-1} (I_k, J_k) = (I_0, J_0)\right)\right] \ \text{(since } J \prec I\text{)} \\
&= \sum_{j=1}^{q-1} \sum_{i=j+1}^{q} \Pr\left[\wedge_{k=0,2,\ldots,n-1} (I_{k+1}, J_{k+1}) = (I_k, J_k) = (i,j) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k}))\right] \\
&= \sum_{j=1}^{q-1} \sum_{i=j+1}^{q} \sum_{\rho, \mathsf{S}_{(j-)}} \frac{\Pr\left[\wedge_{k=0,2,\ldots,n-1} (I_{k+1}, J_{k+1}) = (I_k, J_k) = (i,j) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k})\right]}{|\mathbb{R}||\mathbb{S}|^{j-1}} \\
&= \sum_{j=1}^{q-1} \sum_{i=j+1}^{q} \sum_{\rho, \mathsf{S}_{(j-)}} \frac{\prod_{k=0,2}^{n-1} \Pr\left[(I_{k+1}, J_{k+1}) = (I_k, J_k) = (i,j) \wedge (s^{k+1}_{I_k} \ne s^k_{I_k}) \mid \bigwedge_{l=0,2}^{k-2} \mathsf{G}_\ell\right]}{|\mathbb{R}||\mathbb{S}|^{j-1}}
\end{aligned} \tag{43}$$

In the above expression, G_ℓ denotes the event

$$(I_{\ell+1}, J_{\ell+1}) = (I_\ell, J_\ell) = (i,j) \wedge (s^{\ell+1}_{I_\ell} \ne s^\ell_{I_\ell}).$$

Using the random variable $Z_{i,j}$, (43) can be rewritten as

$$\begin{aligned}
mfrk_{2,c}(x) &= \sum_{j=1}^{q-1} \sum_{i=j+1}^{q} \sum_{\rho, \mathsf{S}_{(j-)}} \frac{\prod_{k=0,2}^{n-1} Z_{i,j}(\rho, \mathsf{S}_{(j-)})}{|\mathbb{R}||\mathbb{S}|^{j-1}} \\
&= \sum_{j=1}^{q-1} \sum_{i=j+1}^{q} \sum_{\rho, \mathsf{S}_{(j-)}} \frac{Z^{(n+1)/2}_{i,j}(\rho, \mathsf{S}_{(j-)})}{|\mathbb{R}||\mathbb{S}|^{j-1}} \\
&= \sum_{j=1}^{q-1} \sum_{i=j+1}^{q} \mathrm{Ex}\left[Z^{(n+1)/2}_{i,j}\right] \ge \sum_{j=1}^{q-1} \sum_{i=j+1}^{q} \mathrm{Ex}\left[Z_{i,j}\right]^{(n+1)/2} \quad \text{(by Jensen's inequality)} \\
&\ge \frac{1}{q^{n-1}} \left(\sum_{j=1}^{q-1} \sum_{i=j+1}^{q} \mathrm{Ex}\left[Z_{i,j}\right] \right)^{(n+1)/2} \qquad \text{(by Hölder's inequality)}
\end{aligned} \tag{44}$$

Using a similar line of approach (as in (43) and (44)), it is possible to establish that

$$mfrk_{2,s}(x) = \frac{(n-1)(n+1)}{8|\mathbb{S}|} \left(\sum_{j=1}^{q-1} \sum_{i=j+1}^{q} \mathrm{Ex}\left[Z_{i,j}\right] \right). \tag{45}$$

Substituting the value of *mfrk*$_{2,c}(x)$ from (44) and *mfrk*$_{2,s}(x)$ from (45) in (42), yields the bound in **Claim 5**. □

What remains is to relate **Claim 1** and **Claim 5**

Claim 6. $\sum_{j=1}^{q-1} \sum_{i=j+1}^{q} Ex\left[Z_{i,j}\right] \ge frk(x)$.

Argument. This argument makes use of the partitions of $\mathbb{T}_{(2)}$ that we used in **Claim 3**. Let $\mathbb{T}_{(2,ji)}$ denote the underlying set for the random variable $Z_{i,j}$. From the definition of $Z_{i,j}$, we can infer that

$$\begin{aligned}\mathbb{T}_{(2,ji)} &:= \left(\mathbb{R} \times \mathbb{S}^{j-1} \times \mathbb{S}^{i-j} \times \mathbb{S}^{(q-i+1)*2}\right) = \left(\mathbb{R} \times \mathbb{S}^{i-1} \times \mathbb{S}^{(q-i+1)*2}\right) \text{ and} \\ \operatorname{Ex}[Z_{i,j}] &= \Pr_{\mathbb{T}_{(2,ji)}}\left[(J' = J = j) \wedge (I' = I = i) \wedge (s'_I \neq s_I)\right].\end{aligned}$$

Notice that $\mathbb{T}_{(2,ji)}$ is the subset of $\mathbb{T}_{(2)}$. Let $\bar{\mathbb{T}}_{(2,ji)}$ denote $\mathbb{T}_{(2)} \setminus \mathbb{T}_{(2,ji)}$. It follows by definition that

$$\Pr_{\bar{\mathbb{T}}_{(2,ji)}}\left[(J' = J = j) \wedge (I' = I = i) \wedge (s'_I \neq s_I)\right] = 0. \tag{46}$$

The fact that $\mathbb{T}_{(2,ji)} \subset \mathbb{T}_{(2)}$ and (46) implies

$$\begin{aligned}&\Pr_{\mathbb{T}_{(2,ji)}}\left[(J' = J = j) \wedge (I' = I = i) \wedge (j < I \leq q) \wedge (s'_I \neq s_I)\right] \geq \\ &\quad \Pr_{\mathbb{T}_{(2)}}\left[(J' = J = j) \wedge (I' = I = i) \wedge (j < I \leq q) \wedge (s'_I \neq s_I)\right].\end{aligned}$$

On taking the sum of $\operatorname{Ex}[Z_{i,j}]$ over the indices (i, j), we get

$$\begin{aligned}\sum_{j=1}^{q-1}\sum_{i=j+1}^{q} \operatorname{Ex}[Z_{i,j}] &\geq \sum_{j=1}^{q-1}\sum_{i=j+1}^{q} \Pr_{\mathbb{T}_{(2)}}\left[(J' = J = j) \wedge (I' = I = i) \wedge (j < I \leq q) \wedge (s_I \neq s'_I)\right] \\ &= \Pr_{\mathbb{T}_{(2)}}\left[(J' = J) \wedge (I' = I) \wedge (1 \leq J < I \leq q) \wedge (s_I \neq s'_I)\right] = \mathit{frk}(x)\end{aligned}$$

completing the argument. □

On putting all the three claims together, we get

$$\begin{aligned}\mathit{mfrk}_2(x) &\geq \frac{1}{q^{n-1}}\mathit{frk}(x)^{(n+1)/2} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\mathit{frk}(x) \\ &= \mathit{frk}(x) \cdot \left(\frac{\mathit{frk}(x)^{(n-1)/2}}{q^{n-1}} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\right)\end{aligned}$$

Finally, taking the expectation over $x \xleftarrow{\$} \mathcal{G}_I$, yields

$$\mathit{frk} \geq \mathit{acc}\left(\frac{\mathit{acc}}{q}(1-\eta) - \frac{1}{|\mathbb{S}|}\right) \text{ and } \mathit{mfrk}_2 \geq \mathit{frk} \cdot \left(\frac{\mathit{frk}^{(n-1)/2}}{q^{n-1}} - \frac{(n-1)(n+1)}{8|\mathbb{S}|}\right)$$

establishing **Lemma 10**. We conclude with the comment that on assuming $|\mathbb{S}| \gg 1$, one gets $\mathit{mfrk}_2 \approx \mathit{acc}^{n+1}/q^{(3n-1)/2}$. □

E Constructions

E.1 The Boldyreva-Palacio-Warinschi Proxy Signature Scheme

Let $\mathfrak{S} = \{\mathcal{G}_s, \mathcal{K}_s, \mathcal{S}_s, \mathcal{V}_s\}$ denote the Schnorr signature scheme. The Triple-Schnorr proxy signature scheme $\mathfrak{B}$ consists of the algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V}, (\mathcal{D}, \mathcal{P}), \mathcal{S}_p, \mathcal{V}_p, \mathcal{I})$, each of which is defined below.

Parameter Generation, $\mathcal{G}(\kappa)$: Run the Schnorr Parameter Generation algorithm $\mathcal{G}_s$ on κ to obtain $\mathtt{pp}_s = (\mathbb{G}, g, p, \mathrm{G})$. Return $\mathtt{pp} := (\mathtt{pp}_s, \mathrm{H}, \mathrm{R})$ as public parameters, where H and R are two hash functions

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

Key Generation, $\mathcal{K}(\mathtt{pp})$: Run the Schnorr Key Generation algorithm $\mathcal{K}_s$ on $\mathtt{pp}_s$ to obtain $((\mathtt{pp}_s, X), (\mathtt{pp}_s, x))$. Return $\mathtt{pk} := (\mathtt{pp}, X)$ as the public key and $\mathtt{sk} := (\mathtt{pp}, x)$ as the secret key. For convenience, we drop pp from the keys; thus, (X, x) serves as the key-pair.

Signing, $\mathcal{S}(m, \mathtt{sk})$: Return $\mathcal{S}_s^{\mathrm{G}}(1\|m, \mathtt{sk})$ as the signature on the message m.

Verification, $\mathcal{V}(m, \sigma, \mathtt{pk})$: Return $\mathcal{V}_s^{\mathrm{G}}(\sigma, 1\|m, \mathtt{pk})$.

Delegation, $(\mathcal{D}(\mathtt{pk}_i, \mathtt{sk}_i, j, \mathtt{pk}_j, \tilde{\mathbb{M}}), \mathcal{P}(\mathtt{pk}_j, \mathtt{sk}_j, \mathtt{pk}_i))$: Let S be a shorthand for the string $(\mathtt{pk}_i\|j\|\mathtt{pk}_j\|\tilde{\mathbb{M}})$. User i computes

$$\mathtt{cert} := (Y, s) \xleftarrow{\$} \mathcal{S}_s^{\mathrm{G}}(\mathtt{sk}_i, 0\|S)$$

and sends it to user j. User j, in turn, verifies cert and computes the proxy signing key $\mathtt{skp} := (S, Y, t)$ where

$$t = r \cdot \mathtt{sk}_j + s \text{ with } r := \mathrm{R}(S\|Y\|c) \text{ and } c := \mathrm{G}(0\|S\|Y).$$

Proxy Signing, $\mathcal{S}_p(\mathtt{skp}, \tilde{m})$: Let $r := \mathrm{R}(S\|Y\|c)$ and $c := \mathrm{G}(0\|S\|Y)$. If $\tilde{m}$ is indeed from the message space $\tilde{\mathbb{M}}$ then return

$$\sigma_p := \Big(j, \mathtt{pk}_j, \tilde{\mathbb{M}}, Y, \mathcal{S}_s^{\mathrm{H}}(t, 0\|\tilde{m}\|S\|Y\|r)\Big)$$

as the proxy signature on $\tilde{m}$ on behalf of user j by the user i. Else, return $\perp$.

Proxy Verification, $\mathcal{V}_p(\sigma_p, \tilde{m}, \mathtt{pk}_i)$: Let σ_p be of the form $(j, \mathtt{pk}_j, \tilde{\mathbb{M}}, Y, \sigma)$. Compute the stamp $S := (\mathtt{pk}_i\|j\|\mathtt{pk}_j\|\tilde{\mathbb{M}})$. In addition, compute the proxy public key

$$\mathtt{pkp} := \mathtt{pk}_j^r \cdot Y \cdot \mathtt{pk}_i^c$$

where $r := \mathrm{R}(S\|Y\|c)$ and $c := \mathrm{G}(0\|S\|Y)$. Return $\mathcal{V}_s^{\mathrm{H}}(\sigma, 0\|\tilde{m}\|S, \mathtt{pkp})$

Proxy Identification, $\mathcal{I}(\sigma_p)$: Let σ_p be of the form $(j, \mathtt{pk}_j, \tilde{\mathbb{M}}, Y, \sigma)$. Return j as the identity of the proxy signer.

Figure 4: The BPW Proxy Signature Scheme. (We use $\tilde{\mathbb{M}}$ in place of ω (used in [BPW12]) to maintain uniformity of notation.)

Remark 8. Self-delegation can be achieved by invoking the interactive algorithm $(\mathcal{D}, \mathcal{P})$ on an alternative key-pair of the designator (itself), in place of the key-pair of the proxy signer. For example, a user i with an alternative key-pair $(\mathtt{pk}_i', \mathtt{sk}_i')$ can delegate itself by invoking

$$(\mathtt{skp}, \perp) \xleftarrow{\$} (\mathcal{D}(\mathtt{pk}_i, \mathtt{sk}_i, i, \mathtt{pk}_i', \tilde{\mathbb{M}}), \mathcal{P}(\mathtt{pk}_i', \mathtt{sk}_i', \mathtt{pk}_i)).$$

E.2 The (Original) Galindo Garcia IBS

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

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

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

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

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

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

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

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

Figure 5: The (Original) Galindo-Garcia IBS.

E.3 The Modified Galindo Garcia IBS

The construction is same as in **Figure 5** except for the structure of the hash functions. We have introduced a binding between H and G (through $d := \mathsf{G}(m, A, c)$ where $c := \mathrm{H}(\texttt{id}, R)$) to induce a dependence of $\mathrm{H} \prec \mathrm{G}$. The binding that we have introduced is more refined than the one suggested in §**4.3.1** (*i.e.*, $d := \mathsf{G}(\texttt{id} \| A \| m \| c)$ where $c := \mathrm{H}(R \| \texttt{id})$).

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

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

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

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

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

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

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

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

Figure 6: The Modified Galindo-Garcia IBS.

E.4 Chow-Ma-Weng Zero-Knowledge Argument

Chaum and Pederson devised a zero-knowledge protocol for proving the equality of two discrete logarithms [CP93]. The protocol due to Chow *et al.* improves on that of Chaum and Pederson by increasing efficiency and, also, by including provision for simultaneous checking of $n \geq 2$ discrete logarithms.

Setting. Let $\mathbb{G}$ be a cyclic group of order a prime p. Let $g, h \in \mathbb{G}$ be two random generators for $\mathbb{G}$. In addition, let $\mathrm{H} : \mathbb{G}^2 \mapsto \mathbb{Z}_p^*$ be a cryptographic hash function.

Protocol. The objective of the prover $\mathcal{P}$, after publishing $y_1 = g^x$ and $y_2 = h^x$, is to convince the verifier $\mathcal{V}$ that $\log_g y_1 = \log_h y_2$. This is accomplished through the following sequence of steps:

(i) $\mathcal{P}$ picks $k \xleftarrow{\mathrm{U}} \mathbb{Z}_p^*$ and sends the commitment $\boxed{v := (g^z h)^k}$ to $\mathcal{V}$, where $\boxed{z := \mathrm{H}(y_1, y_2)}$.

(ii) Upon receiving v, $\mathcal{V}$ sends challenge $c \xleftarrow{\mathrm{U}} \mathbb{Z}_p^*$ to $\mathcal{P}$.

(iii) $\mathcal{P}$ sends the response $s := k - cx \mod p$ to $\mathcal{V}$.

(iv) $\mathcal{V}$ accepts *iff* $\boxed{v = (g^z h)^s (y_1^z y_2)^c}$ holds, where $z := \mathrm{H}(y_1, y_2)$.

Figure 7: The Chow-Ma-Weng argument for the statement $\log_g y_1 = \log_g y_2$.

F Security Argument for Modified Galindo-Garcia IBS

Theorem 2. *Let $\mathcal{A}$ be an $(\epsilon, t, q_\varepsilon, q_s, q_{\mathrm{H}}, q_{\mathrm{G}})$-adversary against the modified GG-IBS. If H and G are modelled as random oracles, we can construct either*

(i) Algorithm $\mathcal{R}_1'$ which ϵ_1-breaks the DLP, where $\epsilon_1 = O\left(\epsilon^2/(\exp(1) q_{\mathrm{G}} q_\varepsilon)\right)$ or

(ii) Algorithm $\mathcal{R}_3'$ which ϵ_3-breaks the DLP, where $\epsilon_3 = O\left(\epsilon^4/(q_{\mathrm{H}} + q_{\mathrm{G}})^3\right)$.

Here q_ε and q_s denote the upper bound on the number of extract and signature queries, respectively, that $\mathcal{A}$ can make; q_{H} and q_{G} denote the upper bound on the number of queries to the H-oracle and G-oracle respectively.

Argument. $\mathcal{A}$ is successful if it produces a valid non-trivial forgery $\hat{\sigma} = (\hat{b}, \hat{R}, \hat{A})$ on $(\hat{\texttt{id}}, \hat{m})$. Consider the following complementary events in the case that $\mathcal{A}$ is successful.

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

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

In the event E we give a reduction $\mathcal{R}'_1$, whereas in the event $\bar{\mathsf{E}}$, we give $\mathcal{R}'_3$. Apart from the need of a wrapper, $\mathcal{R}'_1$ is similar to the reduction $\mathcal{R}_1$ given in [CKK13]. $\mathcal{R}'_3$, on the other hand, employs the GMF Algorithm $\mathcal{N}_{\mathbb{A}_3,\mathcal{Y},3}$ (with **Lemma 2**), in place of $\mathcal{M}_{\mathcal{Y},3}$. Hence, we confine the security argument to the details of reduction $\mathcal{R}'_3$. $\square$

Remark 9. The dependence among the hash functions H and G (modelled as random oracles) has been induced by the binding between them. The notion of independence, on the other hand, can be argued in terms of the system of congruences as we have done for BPW-PSS and CMW-ZKP.

F.1 Reduction $\mathcal{R}'_3$

Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. The reduction involves invoking the GMF Algorithm $\mathcal{N}_{\mathbb{A}_3,\mathcal{Y},n}$ on the wrapper $\mathcal{Y}$ as shown in **Algorithm 5**. 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 (see full version of [CKK13] for details). The design of the wrapper $\mathcal{Y}$ follows.

Algorithm 5 Reduction $\mathcal{R}'_3(\Delta)$

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

The Wrapper

Suppose that $q := q_{\mathrm{H}} + q_{\mathrm{G}}$ and $\mathbb{S} := \mathbb{Z}_p$. $\mathcal{Y}$ takes as input the master public key mpk and $s_1, \ldots, s_q$, and returns a triple (I, J, σ) where J and I are integers that refer to the target H and G query respectively and σ is the side-output. In order to track the index of the current random 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 simulation of the protocol environment 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) *H-oracle Query.* $\mathfrak{L}_{\mathrm{H}}$ contains tuples of the form $\langle \mathtt{id}, R, c, \ell, y\rangle$. Here, $(\mathtt{id}, R)$ is the query to the H-oracle with c being the corresponding output. The index of the oracle call is stored in the ℓ-field. Finally, the y-field stores either (a component of) the secret key for id, or a '$\perp$' in case the field is invalid. A fresh H-oracle query is handled as follows: *i*) return $c := s_\ell$ as the output; and *ii*) add $\langle \mathtt{id}, R, c, \ell, \perp\rangle$ to $\mathfrak{L}_{\mathrm{H}}$ and increment ℓ by one.

(b) *G-oracle Query.* $\mathfrak{L}_{\mathrm{G}}$ contains tuples of the form $\langle m, A, c, d, \ell\rangle$. Here, (m, A, c) is the query to the G-oracle with d being the corresponding output. The index of the oracle call is stored in the ℓ-field. A fresh G-oracle query is handled as follows: *i*) return $d := s_\ell$ as the output; and *ii*) add $\langle m, A, c, d, \ell\rangle$ to $\mathfrak{L}_{\mathrm{G}}$ and increment ℓ by one.

(c) *Signature and Extract Queries.* Since the master secret key α is unknown to $\mathcal{Y}$, it has to carefully program the H-oracle in order to generate the user secret key usk. The signature queries, on the other hand, are answered by first generating the usk (as in the extract query), followed by invoking $\mathcal{S}$.

Extract query. $\mathcal{O}_\varepsilon(\mathtt{id})$:

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

(ii) Otherwise, $\mathcal{Y}$ chooses $y \overset{\mathrm{U}}{\leftarrow} \mathbb{Z}_p$, sets $c := s_\ell$ and $R := (g^\alpha)^{-c} g^y$. It then adds $\langle \mathtt{id}, R, c, \ell, y \rangle$[23] to $\mathfrak{L}_{\mathrm{H}}$ and increments ℓ by one (an implicit H-oracle call). Finally, it returns $\mathtt{usk} := (y, R)$ as the secret key.

Signature query. $\mathcal{O}_s(\mathtt{id}, m)$:

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

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

The Output. 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 $\langle \mathtt{id}_j, R_j, c_j, \ell_j, y_j \rangle$ be the tuple in $\mathfrak{L}_{\mathrm{H}}$ that corresponds to the target H-query. Similarly, let $\langle m_i, A_i, c_i, d_i, \ell_i \rangle$ be the tuple in $\mathfrak{L}_{\mathrm{G}}$ that corresponds to the target G-query. $\mathcal{Y}$ returns $(l_i, \ell_j, (\hat{b}, c_j, d_i))$ as its own output. Note that the side-output σ consists of $(\hat{b}, c_j, d_i)$.

F.1.1 Analysis.

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

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