Limits in the Provable Security of ECDSA Signatures

Dominik Hartmann and Eike Kiltz

Ruhr University Bochum
{fistname.lastname}@rub.de

Abstract. Digital Signatures are ubiquitous in modern computing. One of the most widely used digital signature schemes is ECDSA due to its use in TLS, various Blockchains such as Bitcoin and Etherum, and many other applications. Yet the formal analysis of ECDSA is comparatively sparse. In particular, all known security results for ECDSA rely on some idealized model such as the generic group model or the programmable (bijective) random oracle model.
In this work, we study the question whether these strong idealized models are necessary for proving the security of ECDSA. Specifically, we focus on the programmability of ECDSA's "conversion function" which maps an elliptic curve point into its x-coordinate modulo the group order. Unfortunately, our main results are negative. We establish, by means of a meta reductions, that an algebraic security reduction for ECDSA can only exist if the security reduction is allowed to program the conversion function. As a consequence, a meaningful security proof for ECDSA is unlikely to exist without strong idealization.

Keywords. ECDSA, random oracle model, programmability, meta reductions

1 Introduction

The digital signature algorithm (DSA) [34][1] and its elliptic curve variant ECDSA [28] are two of the most prevalent digital signatures to date, used in TLS [4] and various cryptocurrencies such as Bitcoin and Ethereum. Furthermore, there has been a lot of recent interest in designing threshold signing protocols for ECDSA such as [31,25,18,32,19,14,15,12,16,17,30].

Despite its practical importance, only a few research papers so far have studied the provable security of (EC)DSA. Further, all security proofs require some form of strong idealization, e.g. the random oracle model [5,21], the bijective random oracle model [20], the (elliptic curve) generic group model (GGM) [8,27], or the algebraic bijective random oracle model [37].

Generic DSA signatures (GenDSA) are defined over a group $\mathbb{G}$ of prime order p with generator g, a hash function H, and a so-called "conversion function"

[1] FIPS 186-5 from February 2023 does no longer approve DSA signatures for digital signature generation. However, DSA may still be used for signature verification.

$f : \mathbb{G}^* \to \mathbb{Z}_p$ mapping group elements to exponents. A public key consists of a single group element $X = g^x$ with its discrete logarithm x being the secret key. To sign a message m, the signer samples a random exponent r, computes g^r and then applies the conversion function to compute $t = f(g^r)$. Next, it hashes the message m and computes $s = \frac{H(m)+tx}{r} \bmod p$. The signature is the tuple $(s, t) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$. DSA and ECDSA can be seen as special cases of GenDSA, instantiated over specific groups and with specific conversion functions. For ECDSA, $\mathbb{G}$ is an elliptic curve group and the conversion function f maps an element on the elliptic curve to its x-coordinate modulo p; for DSA, $\mathbb{G}$ is a subgroup of order p of a prime order field, and f reduces a group element modulo p.

Whereas in most previous security proofs of GenDSA, the hash function H was modeled as a perfect random function (i.e., a random oracle [3]), properly modeling the conversion function $f : \mathbb{G}^* \to \mathbb{Z}_p$ turned out to be problematic. In both DSA and ECDSA, the conversion functions in essence reduce (part of) the group element modulo the order of the group generator, but they completely disrupt the algebraic meaning of their input. Specifically, for ECDSA, f is "almost invertible", i.e., inverting f only loses the sign of the elliptic curve point. However, all known security proofs (of full UF-CMA security) have the following drawback:

Observation: *All existing security proofs of (EC)DSA model (parts of) the conversion function f as some programmable, idealized function.*

Specifically, they either model f *explicitly* using a random function/bijection [5,37,20] or they hide the group elements via the random encoding of the GGM and hence model f *implicitly* as a random function [8,27]. While this approach does mimic how f breaks the algebraic meaning of its inputs, it actually does even more. By using a random oracle/bijection, a reduction is allowed to *program* the random function, as long as the adversary cannot detect the changes. Similarly, the GGM allows for programming of the representations of the group elements, which makes any non-algebraic function f mapping from $\mathbb{G}$ implicitly programmable. Programmability is already somewhat debatable when modeling hash functions [22], but it is especially unsatisfying when modeling the conversion function f, as the function is conceptually very simple and not random at all, which makes programmability especially unrealistic. Yet, all known proofs for (EC)DSA crucially rely (explicitly or implicitly) on programming the conversion function f. So in this work, we ask the following natural question:

Question: *Is (EC)DSA provably secure without relying on the programmability of the conversion function f?*[2]

[2] We stress that we do not question the modeling of GenDSA's hash function H as a programmable random oracle. Even though the programmable random oracle model has received valid criticism (e.g., [13]), it is generally viewed as a valid heuristic for a modern hash function which was designed to behave randomly.

1.1 Main Contributions

Unfortunately, our results are negative: We show that without modeling (part of) the conversion function f as a programmable idealized object, there is little hope for proving (EC)DSA secure. Together with the known positive results on (EC)DSA by Fersch, Kiltz, and Poettering [20,21], we obtain the following complete picture on the provable security of (EC)DSA:

> ***Main result (informal):*** *(EC)DSA is provably secure if and only if the conversion function f is modeled using a programmable idealized function.*

To be a bit more precise, we follow the framework of [20] and decompose the conversion function f into three functions φ, ψ, Π, with $f = \psi \circ \Pi \circ \varphi$, where only $\Pi : \{0,1\}^L \to \{0, \ldots, 2^L - 1\}$ is an idealized random bijection and φ and ψ are standard model functions. (More details on the modeling of f will be given in Section 3.2.) To simplify the presentation of our results, in the remainder of this introduction we call conversion function f programmable iff the random bijection Π is programmable.

Our main results are summarized in Figure 1, where the considered security notions are Unforgeability against Chosen-Message Attack (UF-CMA), Unforgeability against No-Message Attack (UF-NMA – no signing queries allowed), and Multi-User UF-CMA (n-UF-CMA).

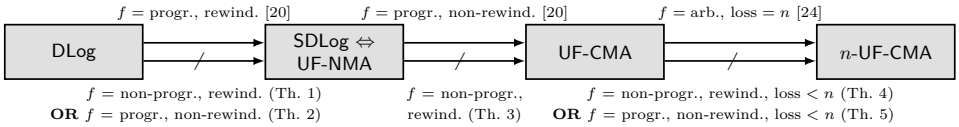

Fig. 1. Relations between hardness assumptions DLog and SDLog (relative to conversion function f) and security notions for GenDSA (relative to hash function H and conversion function f). A crossed arrow refers to an impossibility result, together with the condition under which it holds. "f = progr." means that the conversion function f is modeled as programmable, "f = non-progr." means that f is modeled as non-programmable, and "f = arb." means that f can be arbitrary; "rewind." and "non-rewind." means that the security reduction is rewinding and non-rewinding, respectively. In all negative relations, GenDSA's hash function H (if available) is modeled as a programmable random oracle.

Relation Between DLog and UF-NMA security of GenDSA. We first study the question whether the standard Discrete Logarithm assumption (DLog) can provably imply UF-NMA security of GenDSA (i.e., no signing queries allowed). Since for arbitrary conversion functions f, UF-NMA security is known to be equivalent to the SDLog assumption when modeling the hash function H as a programmable random oracle [6,21], we equivalently study the (simpler) relation between the DLog and the SDLog assumption. Here SDLog refers to the *Semi*

Discrete Logarithm assumption relative to conversion function f [6].[3] On the positive side, [20] use a rewinding reduction to show that DLog is equivalent to SDLog if f is programmable. We contrast their positive result with two negative results.

Firstly, we present a meta reduction showing that if the conversion function f is *non-programmable*, then there exists no *algebraic* reduction that reduces the SDLog to the DLog assumption, as long as the *Free-Base One-More Discrete Logarithm assumption* (FBOMDL) holds in the group. FBOMDL, introduced by Paillier and Vergnaud [36], is a stronger variant of the regular One-More Discrete Logarithm assumption (OMDL) [2]. (See below for a general discussion on assumptions in the context of meta reductions.) Secondly, we show that even if the conversion function f is *programmable*, SDLog cannot be reduced to the DLog assumption by an algebraic, *non-rewinding* reduction, as long as FBOMDL holds.

By the equivalence of SDLog and the UF-NMA security of GenDSA, the relation between DLog and UF-NMA security of GenDSA can be summarized as follows: DLog provably implies UF-NMA security of GenDSA if and only if f is programmable *and* the reduction is allowed to rewind the adversary.

Relation Between UF-NMA and UF-CMA Security. On the positive side, [20] use a non-rewinding reduction to show that UF-NMA implies UF-CMA security of GenDSA if f is programmable. Contrasting their positive result, we prove that there cannot exist an algebraic reduction from the UF-CMA security of GenDSA to its UF-NMA security, if the DLog assumption holds and f is *non-programmable*.

Note that UF-NMA and UF-1CMA security are known to be equivalent for GenDSA [21], where in UF-1CMA security the adversary is only allowed to query *one* signature for each message. By transitivity, our negative result also shows that UF-1CMA does not imply full UF-CMA security, unless f is programmable. This gives a negative answer to an open question of [21].

Relation between Single-User and Multi-User Security. We also consider the multi-user security of GenDSA. While single-user and multi-user security are tightly equivalent for other schemes such as Schnorr signatures [29], we prove that the generic security loss linear in the number of users proven in [24] is indeed optimal for GenDSA, if the reduction is algebraic and f is non-programmable. We also show optimality of the linear loss for the case if the conversion function f is *programmable* but the reduction is restricted to be *non-rewinding*. Our interpretation is that GenDSA's multi-user security inherently loses a linear factor compared to single-user security, unless the reduction can program f and is allowed to rewind the adversary. But rewinding reductions usually require the forking lemma and therefore result in an even worse (quadratic) security loss.

Interpretation of our Results Our results show that there can be no *algebraic* reductions under the conditions mentioned above. More specifically, we

[3] The SDLog assumption essentially says that it is hard to forge a GenDSA signature relative to a message m with $H(m) = 1$, see Definition 3.

have to restrict the class of reductions even further (see Section 3.2). So how relevant are our results then? First, note that almost all security reductions are *generic*, i.e. they do not use anything other than the group structure of a cryptographic group. All generic algorithms are also algebraic [23] and have to fulfill the additional requirements we make on our reductions, so most reductions commonly used are covered by our impossibility results. Another formulation of our results would be that in order to prove GenDSA secure without programming f, one would either have to break FBOMDL, find some new, non-algebraic proof technique or apply additional idealized model such as the AGM or GGM.

Generic Hardness of the FBOMDL Assumption. As we are dealing with meta-reductions, all our negative results are naturally conditioned on certain computational assumptions. Some of them are non-standard and quite strong assumptions, such as the FBOMDL assumption. We would like to stress that we only prove "unprovability" based on such strong assumptions. That is, if the strong assumption gets broken, we only lose the impossibility result and (EC)DSA signatures are most likely still secure. The natural interpretation of our results is that a formal security proof is impossible unless one finds a way to break the assumptions. In that sense for impossibility results from meta-reductions there is only a small risk in using strong assumptions. Other examples following a similar approach include the impossibility result of [10], which is conditioned on the existence of indistinguishability obfuscation (iO). This is in stark contrast to an actual security proofs from strong assumptions where the security of the whole system would be jeopardized in case the assumption is broken.

The OMDL assumption was recently proven secure in the GGM [1]. Their proof mainly uses that each query to the discrete logarithm oracle reveals a linear relation. While the free-base variant allows queries to a different basis point, answers still only reveal linear relations, so their proof applies to the FBOMDL assumption as well. For completeness, we provide a formal proof of the FBOMDL assumption in the GGM in Supplementary Material A.

1.2 Related Work

The first formal security results on *unmodified* (EC)DSA are due to Brown [7,8,9] and prove security of ECDSA in the generic group model. However, the proof is somewhat problematic in that it not only idealizes the underlying group $\mathbb{G}$ but also implicitly the conversion function f. This allows Brown to prove that ECDSA is *strongly* unforgeable (as observed by Stern et al. [39]), which it obviously is not. Additionally, Brown shows different necessary and sufficient conditions for the security of ECDSA, however the sufficient conditions are significantly stronger than the discrete logarithm problem.

As already mentioned, [20,21] take a different approach in that they do not idealize the underlying group but conversion function f and hash function H. Modeling f as a programmable idealized object, [20] show that DLog implies UF-NMA security, which in turn implies full UF-CMA security. Furthermore, [21] model only H as a random oracle, and show that, for any function f, the Semi

Discrete Logarithm assumption (SDLog) implies the UF-NMA security of GenDSA, which in turn implies UF-1CMA security, i.e. an adversary is only allowed a single signature on each message.

Recently, Groth and Shoup [27] revisited the security of ECDSA in the generic group model. They avoid the shortcomings of Brown's proof by considering ECDSA in the so-called Elliptic Curve GGM (EC-GGM). This variant of the GGM keeps some relationships in the elliptic curve intact, such as the fact which points share the same x-coordinate. Their proof still implicitly idealizes the conversion function f, but only requires standard properties for the hash function H. They also consider the security of ECDSA with additive key derivation and when using presignatures (i.e. an adversary gets a number of random group elements and can choose which the challenger has to use to answer signature queries).

In terms of impossibility results, Pailler and Vergnaud [36] show that a large class of signatures including DSA and ECDSA cannot be proven secure in the standard model via a meta reduction to a variant of the One-More Discrete Logarithm problem. We prove a stronger impossibility result, since we model f as in [20] with a (non-programmable) ideal bijection and make no restrictions on the modeling of H (i.e., even allow it to be a *programmable* RO).

Further research either considered variants of GenDSA (or DSA and ECDSA) [33] or analyzed their behavior in the presence of specific faults such as (partial) randomness reuse [35] or collisions [40,41]. Blind signatures and threshold signatures based on ECDSA were proposed in [11,37] and [26,15] respectively.

2 Preliminaries

For integers $m, n \in \mathbb{Z}$, let $[m : n] = \{m, m+1, \ldots, n\}$. If $m = 1$, we write $[n]$ instead. Let S be a (finite) set, then we write $s \xleftarrow{\$} S$ for sampling an element s uniformly at random from S.

2.1 Algebraic Algorithms

We model all algorithms (i.e. adversaries, reductions and meta reductions) as stateful and probabilistic (the specific computational model is not relevant, but one can think of interactive Turing machines). For a probabilistic algorithm $\mathcal{A}$, we write $y \xleftarrow{\$} \mathcal{A}(x_1, \ldots, x_n)$ when $\mathcal{A}$ outputs y on input $(x_1, \ldots, x_n)$ using fresh random coins. For deterministic algorithms, we use $y \leftarrow \mathcal{A}(x_1, \ldots, x_n)$. We write $\mathcal{A}^{\mathbf{O}(\cdot)}$ to indicate that $\mathcal{A}$ gets black-box access to algorithm $\mathbf{O}$ which is also called an oracle. That is, $\mathcal{A}$ can make arbitrary many queries x_i to $\mathbf{O}$ and obtains $\mathbf{O}(x_i)$ as answers.

Throughout this paper, we fix a cryptographic group $\mathcal{G} = (\mathbb{G}, p, g)$, where $\mathbb{G}$ is a finite multiplicative group $\mathbb{G}$ of prime order p, generated by g. An algorithm $\mathcal{A}$ is called *algebraic* [36,23], if it outputs every group element together with a representation relative to its inputs. We also consider oracle-aided algorithms, where an oracle can output (random) group elements. If this is the case, an

algebraic algorithm treats these group elements as additional inputs. On the other hand, if an oracle takes group elements as input, an algebraic algorithm is also required to output a representation for these inputs.

Definition 1 (Algebraic Algorithm). *An algorithm $\mathcal{A}$ is called* algebraic *relative to a group $\mathcal{G} = (\mathbb{G}, p, g)$, if whenever it outputs a group element $X \in \mathbb{G}$ it additionally outputs a representation vector $\vec{z} \in \mathbb{Z}_p^n$ s.t. $X = \prod_{i=1}^{n} Y_i^{z_i}$, where $Y_1, \ldots, Y_n$ denote the group elements previously known to $\mathcal{A}$.*

Our meta reductions will require that the assumed reduction is algebraic, yet the meta reductions themselves (and therefore also the adversaries simulated by them) will not be algebraic. This is analogue to security reductions in the AGM, where an adversary is assumed to be algebraic, but the reduction and the oracles it provides are not algebraic.

As noted by in [42,43], it is somewhat imprecise to talk about the group elements "known" to an algorithm, because additional group elements might be encoded in its inputs. However, all adversaries that we consider only receive group elements as inputs (most only a single group element), so we assume that those are the only group elements known.

2.2 Idealized Functions

In our security reductions we will model the conversion function f and the hash function H of GenDSA as publicly accessible, idealized functions. The most prominent idealized function is a random oracle [3], where a hash function $H : \mathcal{X} \to \mathcal{Y}$ is modeled as a perfect random function $\mathbf{H} \xleftarrow{\$} \{H : \mathcal{X} \to \mathcal{Y}\}$ that can only be accessed through oracle queries. Note that oracle $\mathbf{H}$ can be efficiently implemented using lazy sampling.

Let $\mathrm{Func}(\mathcal{X}, \mathcal{Y})$ be the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$ and let $\mathrm{Bij}(\mathcal{X}, \mathcal{Y})$ be the set of all bijections from $\mathcal{X}$ to $\mathcal{Y}$. The idealized functions that we will use in this work, together with their oracle interfaces, are listed in the following table.

Idealized object	Function type	Oracle	Distribution
Random Oracle (RO)	Function $H : \mathcal{X} \to \mathcal{Y}$	$\mathbf{H}$	$\mathbf{H} \xleftarrow{\$} \mathrm{Func}(\mathcal{X}, \mathcal{Y})$
Bijective Random Oracle (bRO)	Bijection $\Pi : \mathcal{X} \to \mathcal{Y}$	$(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$	$\mathbf{\Pi} \xleftarrow{\$} \mathrm{Bij}(\mathcal{X}, \mathcal{Y})$

2.3 Security Reductions and Programmability

Proving that a hard problem P (e.g., DLog) implies the security of some cryptographic protocol S (e.g., the ECDSA signature scheme) is commonly done via a security reduction. In a security reduction, an adversary $\mathcal{A}$ against S is executed by a reduction $\mathcal{R}$, denoted as $\mathcal{R}^{\mathcal{A}}$, which uses $\mathcal{A}$ to solve its own hard problem P. If further $\mathcal{R}$'s success is meaningfully related to that of $\mathcal{A}$, then one can deduce $\mathsf{P} \Rightarrow \mathsf{S}$. In this work, we always consider *Fully Black Box* reductions [38], i.e. the reduction can not depend on any internal properties of the adversary like its code. However, we assume that upper bounds on the number of oracle queries of all algorithms are known.

Programmable Idealized Functions. If $\mathcal{A}$ is an adversary with oracle access to an idealized function $\mathbf{O}$ (see Section 2.2), then reduction $\mathcal{R}$ executes $\mathcal{A}$ by providing its inputs, randomness, and answering all queries made to the expected oracle $\mathbf{O}$. Using our notation, we write $\mathcal{R}^{\mathcal{A}^{\mathbf{O}}}$ to denote the reduction executing $\mathcal{A}^{\mathbf{O}}$. Consequently, $\mathcal{R}$ has complete control over $\mathcal{A}$'s input and output channel and therefore in particular over oracle $\mathbf{O}$. In the most general case, $\mathcal{R}$ can even rewind the adversary $\mathcal{A}$ in a black-box manner by running it again with the same initial inputs (including randomness) and altering the oracle answers at some point of the execution. Hence in the context of a security reduction, such idealized functions are also called programmable. Most prominently, if $\mathbf{O}$ is modeled as a random function $\mathbf{H}$, then $\mathbf{H}$ is the well known programmable random oracle or simply random oracle [3].

Non-Programmable Idealized Functions. One can also model $\mathcal{A}$'s oracle as an idealized function that reduction $\mathcal{R}$ does *not* control but is only able to observe. We call such an oracle a non-programmable idealized function $\bar{\mathbf{O}}$. We use the notion $\bar{\mathbf{O}}$ ("overline") to make the non-programmability of the function explicit. Here, observing means that whenever $\mathcal{A}$ makes a query X to $\bar{\mathbf{O}}$, it obtains the response $\bar{\mathbf{O}}(X)$ directly from $\bar{\mathbf{O}}$, but $\mathcal{R}$ also gets the pair $(X, \bar{\mathbf{O}}(X))$.[4] Furthermore, we also provide the reduction $\mathcal{R}$ black-box access to $\bar{\mathbf{O}}$. (In the programmable case this is not necessary since $\mathcal{R}$ has already full control over the oracle $\mathbf{O}$.) In the context of a security reduction, we use the notation $\mathcal{R}^{\bar{\mathbf{O}},\mathcal{A}^{\mathbf{O}}}$ to indicate that $\bar{\mathbf{O}}$ is non-programmable by $\mathcal{R}$.

The special case where $\bar{\mathbf{O}}$ is a perfectly random function $\bar{\mathbf{H}}$ yields the well-known non-programmable random oracle [22].

The differences between a programmable and a non-programmable idealized function in the context of a security reduction are visualized in Figure 2.

2.4 Meta Reductions

Meta reductions are a useful tool for proving the (conditional) impossibility of security reductions. On a technical level, a meta reduction $\mathcal{M}$ assumes the existence of a reduction $\mathcal{R} = \mathcal{R}^{\mathcal{A}}$ proving $\mathsf{P} \Rightarrow \mathsf{S}$, i.e., $\mathcal{R}$ solves problem P with black-box access to an adversary $\mathcal{A}$. Meta reduction $\mathcal{M}$ then uses said assumed reduction to break some (potentially different) hard problem P'. As a consequence, such a security reduction $\mathcal{R}$ proving $\mathsf{P} \Rightarrow \mathsf{S}$ cannot exist unless problem P' turns out to be easy. The above requires simulating an adversary $\mathcal{A}$ for $\mathcal{R}$ to work, which is often the main challenge of constructing a meta reduction.

The meta reduction has to provide $\mathcal{R}$ with all oracles it expects in its game as well as (programmable or non-programmable) idealized functions, while $\mathcal{R}$ in turn provides all *programmable* oracles $\mathbf{O}$ to the (simulated) adversary $\mathcal{A}$. This is depicted in Figure 3. For the case of a non-programmable ideal function, there are

[4] [22] consider a more general modeling where $\mathcal{R}$ gets X before the query and can make its own oracle queries which could influence the response $\bar{\mathbf{O}}(X)$. However, since such queries would never alter the behavior in all of our reductions, we only consider this simplified definition.

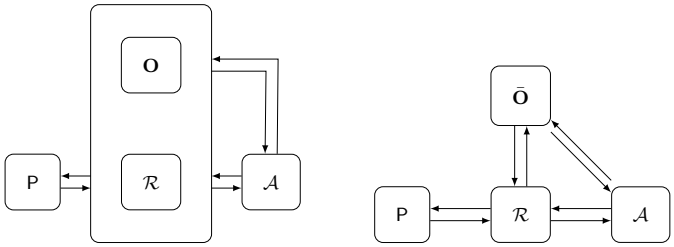

Fig. 2. Left: Security reduction $\mathcal{R}$ running adversary $\mathcal{A}$ with access to a programmable idealized function $\mathbf{O}$. Reduction $\mathcal{R}$ has full control over $\mathcal{A}$'s queries to $\mathbf{O}$. Right: Security reduction $\mathcal{R}$ running $\mathcal{A}$ with access to a non-programmable idealized function $\bar{\mathbf{O}}$. Reduction $\mathcal{R}$ can only observe $\mathcal{A}$'s queries to $\bar{\mathbf{O}}$.

some intricacies to the modeling. Specifically, if $\mathcal{R}$ expects a non-programmable idealized function, $\mathcal{M}$ can itself simulate a *programmable* idealized function (via standard lazy sampling), which is indistinguishable from a non-programmable idealized function for $\mathcal{R}$.

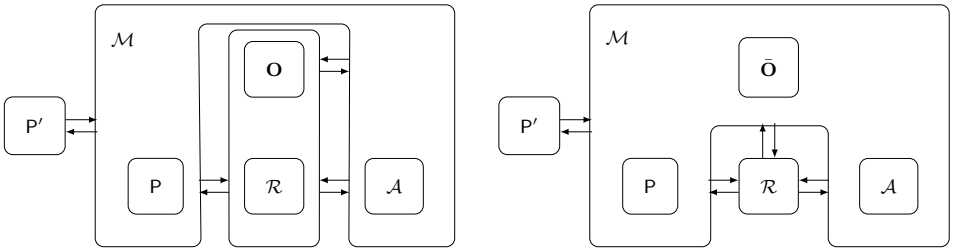

Fig. 3. Depiction of a meta reduction $\mathcal{M}$ interacting with a game P' with a *programmable* oracle $\mathbf{O}$ (left) and *non-programmable* oracle $\bar{\mathbf{O}}$ (right), and a reduction $\mathcal{R}$. In the programmable case, the oracle is "internal" to the reduction, i.e., $\mathbf{O}$ can be controlled arbitrarily by $\mathcal{R}$. In the non-programmable case, the oracle is external to $\mathcal{R}$ and hence meta-reduction $\mathcal{M}$ is able to control $\bar{\mathbf{O}}$.

Fishlin et al. [22] argue that one should allow the reduction the same programming and observability capabilities as the meta reduction, as it keeps the modeling of the function replaced by the random oracle consistent. Intuitively, if the goal is to model realistic functions, it is unreasonable to assume that one (efficient) algorithm has more control over some function than another (efficient) algorithm without any trapdoors. However in our proofs, we have to allow the meta-reduction to program while the reduction sees the random oracle as non-programmable. Indeed, programmability for the meta-reduction seems necessary

in order to extract solutions in the end of the meta reduction. Looking ahead, the meta-reduction will use the solution of the reduction together with the observed random oracle queries to extract a discrete logarithm from the solution and previously simulated signatures.

2.5 Hardness Assumptions

We now recall three hard problems over a group $\mathcal{G} = (\mathbb{G}, p, g)$: The standard Discrete Logarithm problem, the Semi Discrete Logarithm problem [6], and the Free-Base One-More Discrete Logarithm problem [36].

Definition 2 (Discrete Logarithm Problem). *The* discrete logarithm (DLog) *problem is (t, ϵ)-hard in group $\mathcal{G} = (\mathbb{G}, p, g)$, if for all adversaries $\mathcal{A}$ running in time at most t,* $\Pr[x' = x \mid x \xleftarrow{\$} \mathbb{Z}_p; X := g^x; x' \xleftarrow{\$} \mathcal{A}(X)] \leq \epsilon$.

Definition 3 (Semi-Discrete Logarithm Problem). *The* Semi-Discrete Logarithm (SDLog) *problem is (t, ϵ)-hard relative to group $\mathcal{G} = (\mathbb{G}, p, g)$ and conversion function $f : \mathbb{G} \to \mathbb{Z}_p$, if for any adversary $\mathcal{A}$ running in time at most t,* $\Pr[t^* \neq 0 \wedge s^* \neq 0 \wedge t^* = f((g \cdot X^{t^*})^{s^{*-1}}) \mid x \xleftarrow{\$} \mathbb{Z}_p; X = g^x; (s^*, t^*) \xleftarrow{\$} \mathcal{A}(X)] \leq \epsilon$.

We remark that the SDLog problem can equivalently be seen as "forging a GenDSA signature (see Definition 8) on a message m with $H(m) = 1$". While SDLog is defined relative to a specific conversion function f, it can be replaced by an idealized function in a reduction similar to how explicit hash functions can be replaced with random oracles.

Definition 4 (Free-Base One-More Discrete Logarithm Problem). *The* q-Free-Base One-More Discrete Logarithm (q-FBOMDL) *problem is (t, ϵ)-hard in group $\mathcal{G} = (\mathbb{G}, p, g)$, if for all adversaries $\mathcal{A}$ running in time at most t,*

$$\Pr\left[\forall 0 \leq i \leq q : X_i = g^{x_i} \;\middle|\; \begin{matrix} X_0 \xleftarrow{\$} \mathbb{G} \\ (x_0, \ldots, x_q) \xleftarrow{\$} \mathcal{A}^{\mathbf{DL}(\cdot,\cdot),\mathbf{Chal}}(X_0) \end{matrix}\right] \leq \epsilon,$$

where the oracles are defined as follows. The challenge oracle **Chal***, on its i-th query, outputs a group element $X_i \xleftarrow{\$} \mathbb{G}^*$. It can be queried at most q times. The free-base discrete logarithm oracle* $\mathbf{DL}(\cdot,\cdot)$*, on input $(X, R) \in \mathbb{G} \times \mathbb{G}^*$, outputs $z \in \mathbb{Z}_p$ s.t. $R^z = X$. It can be queried as many times as the* **Chal** *oracle was queried.*

q-FBOMDL is a stronger variant of the regular One-More Discrete Logarithm assumption [2], where the discrete logarithm oracle is restricted to base g. To increase the confidence in q-FBOMDL, we will prove its unconditional hardness in the GGM in Supplementary Material A.

2.6 Digital Signatures

We recall the definition of digital signatures and their standard notions of security.

Definition 5 (Digital Signatures). *A* digital signature scheme $\mathit{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *is a tuple of three algorithms with the following properties:*

- $\mathsf{Gen} \xrightarrow{\$} (\mathsf{sk}, \mathsf{vk})$*: The key generation algorithm outputs a public key/secret key pair. The public key implicitly defines the message space* $\mathcal{M}$*.*
- $\mathsf{Sign}(\mathsf{sk}, m) \xrightarrow{\$} \sigma$*: The signing algorithm takes a secret key* sk *and a message* m *as input and returns a signature* σ*.*
- $\mathsf{Ver}(\mathsf{vk}, m, \sigma)$*: The (deterministic) verification algorithm gets a public key* vk*, a message* m *and a signature* σ *and outputs* 1 *for accept and* 0 *for reject.*

A digital signature scheme is ϵ-correct*, if for all* $(\mathsf{sk}, \mathsf{vk}) \in \mathsf{Gen}$ *and all messages* $m \in \mathcal{M}$

$$Pr\left[\mathsf{Ver}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 0\right] \leq \epsilon$$

Definition 6 (Signature Security). *Let* $n \in \mathbb{N}$ *and* $\mathit{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *be a digital signature scheme.* SIG *is said to be* (t, ϵ, q_S)-n-$\mathsf{UF\text{-}ATK}$ *secure (*Multi-User Unforgeability against ATK attacks*), if for every adversary* $\mathcal{A}$ *running in time at most* t *and making at most* q_S *queries to the oracle* Sign*,*

$$Pr\left[\begin{array}{c|c} (i^*, m^*) \notin \mathcal{Q} \wedge & \textit{For } i \in [n] : (\mathsf{vk}_i, \mathsf{sk}_i) \xleftarrow{\$} \mathsf{Gen} \\ \mathsf{Ver}(\mathsf{vk}_{i^*}, m^*, \sigma^*) = 1 & (i^*, m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathrm{Sign}(\cdot,\cdot)}((\mathsf{vk}_i)_{i \in [n]}) \end{array}\right] \leq \epsilon,$$

where depending on ATK*, the oracle* Sign *is defined as follows:*

- $\mathsf{ATK} = \mathsf{NMA}$*: The oracle* Sign *always returns* $\perp$*.*
- $\mathsf{ATK} = \mathsf{CMA}$*: On input of a message* m *and an index* $i \in [n]$*,* Sign *returns a signature* σ *for* m *under secret key* sk_i *and adds* (i, m) *to* $\mathcal{Q}$*.*

If the adversary has additionally access to an idealized function via oracle $\mathbf{O}(\cdot)$*, we denote this as* $(t, \epsilon, q_S, q_{\mathbf{O}})$-$n$-$\mathsf{UF\text{-}ATK}$ *security, where* $q_{\mathbf{O}}$ *denotes the number of queries to* $\mathbf{O}$*. For* NMA *security, we omit the parameter* q *from the description and simply write* (t, ϵ)-n-$\mathsf{UF\text{-}NMA}$*. If* $n = 1$*, i.e. we are in a single-user setting, we instead write* (t, ϵ, q_S)-$\mathsf{UF\text{-}ATK} = (t, \epsilon, q_S)$-1-$\mathsf{UF\text{-}ATK}$*.*

3 Generic DSA and the Conversion Function

In this section, we recall the definition of GenDSA from [20] and discuss the modeling of the conversion function f in our meta reductions. We also recall some basic definitions beforehand.

Definition 7 (Semi-Injective Function). *Let* $\mathbb{G}$ *be a prime order group and* $\mathcal{Y}$ *a set. A function* $\varphi : \mathbb{G}^* \to \mathcal{Y}$ *is called* semi-injective *if*

1. *its range* $\varphi(\mathbb{G}^*) \subseteq \mathcal{Y}$ *is efficiently decidable and*
2. *it is either injective or a 2-to-1 function with* $\varphi(x) = \varphi(y) \implies x \in \{y, y^{-1}\}$

3.1 Generic DSA Signatures

In this work, we will study Generic DSA (GenDSA) [20], an abstract signature framework which subsumes both DSA and ECDSA. It models the conversion function as $f = \psi \circ \Pi \circ \varphi$ for efficient functions ψ, φ and a bijection Π. According to [20], the idea is to reflect in φ the structure of f that involves only its domain and to reflect in ψ the structure that involves only its range; the component that is responsible for disrupting any algebraic link between the domain and the range is modeled by Π. In security proofs we, Π will be replaced by a bijective random oracle. DSA and ECDSA can naturally be obtained, together with the specific peculiarities of their conversion functions, by correspondingly instantiating φ and ψ. For a more in-depth discussion of GenDSA, we refer the reader to [20].

Definition 8 (GenDSA). *Let $L \in \mathbb{N}$. The signature scheme $\mathsf{GenDSA} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ relative to a group $\mathcal{G} = (\mathbb{G}, p, g)$, a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$ and a conversion function $f : \mathbb{G}^* \to \mathbb{Z}_p$ with $f = \psi \circ \Pi \circ \varphi$ for efficient functions ψ, φ and bijection Π with*

$$\mathbb{G}^* \xrightarrow{\varphi} \{0,1\}^L \xrightarrow{\Pi} [0 : 2^L - 1] \xrightarrow{\psi} \mathbb{Z}_p$$

is defined as follows.

$\underline{\mathsf{Gen}}$**:**

$x \xleftarrow{\$} \mathbb{Z}_p$; $X := g^x$
$\mathsf{vk} := X$; $\mathsf{sk} := x$
return $(\mathsf{vk}, \mathsf{sk})$

$\underline{\mathsf{Sign}(\mathsf{sk} = x, m)}$**:**

$r \xleftarrow{\$} \mathbb{Z}_p^*$; $R := g^r$
$t := f(R)$; $h := H(m)$
$s := \frac{h + x \cdot t}{r}$
if $(t = 0) \vee (s = 0)$ ***then***
  return $\perp$
return (s, t)

$\underline{\mathsf{Ver}(\mathsf{vk} = X, m, \sigma = (s,t))}$**:**

if $(t = 0) \vee (s = 0)$ ***then***
  return $\perp$
$h := H(m)$
$t' = f\left(\left(g^h X^t\right)^{\frac{1}{s}}\right)$
return $t = t'$

To the function ψ from Definition 8, we associate the quantity

$$\epsilon_\psi = \max_{t \in \mathbb{Z}_p} \Pr_{y \in [0:2^L-1]}[\psi(y) = t], \tag{1}$$

ECDSA AND DSA. For ECDSA [28], $\mathcal{G}$ is instantiated with a prime-order subgroup $\mathbb{G}$ of an elliptic curve group over $\mathbb{F}_q$. For $L = \lceil \log_2(q) \rceil$, $\varphi : \mathbb{G}^* \to \{0,1\}^L$ maps an elliptic curve point $R = (R_x, R_y) \in \mathbb{F}_q \times \mathbb{F}_q$ to the binary representation of its x-coordinate R_x, which makes it a semi-injective 2-to-1 function. Note that $\varphi(\mathbb{G})$ is efficiently decidable. Function $\psi : [0 : 2^L - 1] \to \mathbb{Z}_p$ is the modular reduction modulo p. By Hasse's theorem, we get $p \leq 2q$ for all $q \geq 13$ (see [27]). If $q \geq p$, then the probability that ψ maps an element from $[0 : 2^L - 1]$ with $L = \lceil \log_2(q) \rceil$ to a specific $y \in \mathbb{Z}_p$ is at most $2/p$, since there are at most $\lceil 2^L/p \rceil$ preimages per y. On the other hand, if $q \leq p$, we get that the same probability is bounded by $1/q \leq 1/(p/2) = 2/p$, where the inequality follows from the Hasse bound. So in both cases $\epsilon_\psi \leq 2/p$.

For DSA, $\mathcal{G}$ is instantiated with a multiplicative prime order subgroup of the a prime order field $\mathbb{F}_q$, i.e. $\mathbb{G} \subset \mathbb{F}_q^*$. For $L = \lceil \log_2(q) \rceil$ $\varphi : \mathbb{G}^* \to \{0,1\}^L$ is the mapping of a group element to its binary representation, hence it is injective (so also semi-injective). $\varphi(\mathbb{G})$ is efficiently decidable since the group order is known. $\psi : [0 : 2^L - 1] \to \mathbb{Z}_p$ is the reduction modulo p. Since we always have $p \leq 2q + 1$, it is straight forward to see $\epsilon_\psi \leq 2/p$.

3.2 Modeling the Conversion Function in Proofs

As observed by Brown [9], the non-algebraic conversion function $f : \mathbb{G}^* \to \mathbb{Z}_p$ is an integral part of a security analysis of GenDSA. However, modeling the abstract properties of f in formal proofs turns out to be a non-trivial task.

We use Definition 8 and decompose f as $f = \psi \circ \Pi \circ \varphi$. In our meta reductions, permutation $\Pi : \{0,1\}^L \to [0 : 2^L - 1]$ is modeled as a *bijective Random Oracle* (bRO) $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ and the two functions $\varphi : \mathbb{G}^* \to \{0,1\}^L$, $\psi : [0 : 2^L - 1] \to \mathbb{Z}_p$ are standard model functions. This allows to accurately model the biases introduced by the real conversion functions while having a clear interface to work with in (meta-)reductions.

The Conversion Function in the AGM. In our results we will provide an *algebraic* algorithm $\mathcal{A}$ (i.e., a reduction) access to a bijective Random Oracle $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$. Since algorithm $\mathcal{A}$ is algebraic, we make the following constraints:

1. Whenever an algebraic algorithm $\mathcal{A}$ queries $\mathbf{\Pi}^{-1}(y)$ for some $y \in [0 : 2^L - 1]$, it receives the output $x = \mathbf{\Pi}^{-1}(y) \in \{0,1\}^L$ of the bRO. If $x \in \varphi(\mathbb{G}^*)$ (which can be efficiently decided by Definition 7), then we add $\varphi^{-1}(x)$ to the list of known group elements relative to which the algebraic algorithm can output representations. Note that, by using lazy sampling, queries to $\mathbf{\Pi}^{-1}(y)$ can be efficiently simulated even if φ is *not* efficiently invertible: We first sample a fresh random element $x \in \{0,1\}^L$. If $x \notin \varphi(\mathbb{G}^*)$, we define $\mathbf{\Pi}^{-1}(y) := x$. If $x \in \varphi(\mathbb{G})$, we discard x and define $\mathbf{\Pi}^{-1}(y) := \varphi(g^r)$ for $r \xleftarrow{\$} \mathbb{Z}_p$.
2. Whenever an algebraic algorithm $\mathcal{A}$ queries $\mathbf{\Pi}(x)$ for some $x \in \varphi(\mathbb{G}^*)$, it also has to provide a group element $R \in \varphi^{-1}(x)$, together with an algebraic representation of R. (Note that φ is semi-injective, and hence the exact choice of $R \in \varphi^{-1}(x)$ is irrelevant.)

The latter constraint above was previously introduced in [37] as the *Algebraic Bijective Random Oracle model*. However they only consider algebraic use of the bRO interface $\mathbf{\Pi}$ used in f in contrast to the fully algebraic reductions in our meta reductions. Therefore, we make the need for a representation relative to φ part of the algebraic adversary instead of $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ and add the potentially new group elements computable from $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ to $\mathcal{A}$'s inputs. Specifically, this is *not* an additional idealization of φ for $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ but a restriction on the class of algorithms we consider.

Why is this a reasonable modeling of the conversion function in the algebraic group model? First, as argued in [20], the composition of the three functions allows to accurately model the biases and intricacies of the conversion function

in many different settings such as DSA and ECDSA. However, since the bijective random oracle $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ does not receive group elements as inputs, it is generally incompatible with the AGM. The second constraint bridges this gap somewhat artificially by demanding a representation of a preimage under φ whenever $\mathbf{\Pi}$ is queried, which implicitly assumes that an algorithm will always use $\mathbf{\Pi}$ in composition with φ (it can still use φ without querying $\mathbf{\Pi}$). This assumption can be seen as a simply another convenient constraint on algorithms similar to the ones already made by the AGM. From this point of view, we simply consider an even more limited class of algorithms.

Note that all *generic* algorithms are still covered by this limited class. Specifically, in the GGM, a representation of a preimage under φ is always known for every $z \in \{0,1\}^L$ as long as $\varphi^{-1}(z)$ is a valid label. This is due to the fact that the GGM can simply track all calls to the group oracle and therefore knows a representation for all defined labels relative to the inputs. If φ^{-1} produces a new, valid label, the GGM can internally simply assign it a random group element for which it knows a representation, keeping this invariant intact. For invalid labels, nothing has to be done.

However, this restriction is indeed also a reasonable assumption if we look at the concrete instantiations of the conversion function. For ECDSA, φ is essentially the projection of an elliptic curve point (R_x, R_y) to is x-coordinate. While it is easy for many curves to sample x-coordinates uniformly at random, doing so samples the curve point obliviously, i.e. without knowledge of the discrete logarithm or a representation relative to other points. If an algorithm were to use such a group element in its solution, it would have to produce a non-trivial equation in the secret key or challenge *and* this unknown discrete logarithm. Worse still, this choice of an x-coordinate also does not give any more control over the output of $\mathbf{\Pi}$ or $\psi \circ \mathbf{\Pi}$, since $\mathbf{\Pi}$ is a random function, so using $\mathbf{\Pi}$ without φ only loses information.

4 Impossibility Results

4.1 DLog $\not\Rightarrow$ SDLog

Our first result is to show that DLog does not imply SDLog if we consider only non-programming, algebraic reductions. Specifically, the following theorem states that there cannot exist an algebraic reduction from SDLog to DLog that does not program the random bijection $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$, unless the FBOMDL assumption is false.

Theorem 1 ($\mathsf{DLog} \overset{\Pi \text{ non-progr.}}{\not\Rightarrow} \mathsf{SDLog}$). *Let $f = \psi \circ \Pi \circ \varphi$ be a conversion function with semi-injective $\varphi : \mathbb{G}^* \to \{0,1\}^L$, $\psi : [0 : 2^L - 1] \to \mathbb{Z}_p$ and $\Pi : \{0,1\}^L \to [0 : 2^L - 1]$ modeled by a* non-programmable *bijective random oracle $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$. If there exists an algebraic reduction $\mathcal{R}$ that $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, q'_{\bar{\mathbf{\Pi}}}, q'_{\bar{\mathbf{\Pi}}^{-1}})$-breaks the* DLog *assumption given q_P-times access to an adversary $\mathcal{A}$ that $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_{\bar{\mathbf{\Pi}}}, q_{\bar{\mathbf{\Pi}}^{-1}})$-breaks*

the SDLog *assumption, then there exists a meta-reduction* $\mathcal{M}$ *which* $(t_{\mathcal{M}}, \epsilon_{\mathcal{M}})$*-breaks the* q_P-FBOMDL *assumption with*

$$\epsilon_{\mathcal{M}} \geq \epsilon_{\mathcal{R}} - 2q_P\epsilon_{\psi} - \frac{q^2}{2^L}, \quad t_{\mathcal{M}} \approx t_{\mathcal{R}} + q_P t_{\mathcal{A}},$$

where $q = q_P(q_{\bar{\mathbf{\Pi}}} + q_{\bar{\mathbf{\Pi}}^{-1}}) + q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}}$ *is the total amount of queries to* $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ *made by* $\mathcal{A}$ *and* $\mathcal{R}$*.*

We state an orthogonal impossibility result for non-rewinding reductions which have access to a *programmable* bRO $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$. Specifically, the following theorem states that there cannot exist an algebraic reduction $\mathcal{R}$ that reduces SDLog to DLog without rewinding the adversary, unless the FBOMDL assumption is false.

Theorem 2 (DLog $\overset{\text{non-rew.}}{\not\Rightarrow}$ SDLog)**.** *Let* $f = \psi \circ \mathbf{\Pi} \circ \varphi$ *be a conversion function with semi-injective* $\varphi : \mathbb{G}^* \to \{0,1\}^L$*,* $\psi : [0 : 2^L - 1] \to \mathbb{Z}_p$ *and* $\mathbf{\Pi} : \{0,1\}^L \to [0 : 2^L - 1]$ *modeled by a* programmable *bijective random oracle* $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$*. If there exists an algebraic reduction* $\mathcal{R}$ *that* $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}})$*-breaks the* DLog *assumption given one-times access to an adversary* $\mathcal{A}$ *that* $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_{\mathbf{\Pi}}, q_{\mathbf{\Pi}^{-1}})$*-breaks the* SDLog *assumption, then there exists a meta-reduction* $\mathcal{M}$ *which* $(t_{\mathcal{M}}, \epsilon_{\mathcal{M}})$*-breaks the* 1-FBOMDL *assumption with*

$$\epsilon_{\mathcal{M}} \geq \epsilon_{\mathcal{R}}, \quad t_{\mathcal{M}} \approx t_{\mathcal{R}} + t_{\mathcal{A}}.$$

We now prove Theorem 1. The proof of Theorem 2 is similar and can be found in Supplementary Material B.1.

Proof (of Theorem 1). Let $\mathcal{A}$ be an adversary that $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_{\bar{\mathbf{\Pi}}}, q_{\bar{\mathbf{\Pi}}^{-1}})$-breaks the SDLog assumption, and let $\mathcal{R}$ be the security reduction that $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, q'_{\bar{\mathbf{\Pi}}}, q'_{\bar{\mathbf{\Pi}}^{-1}})$-breaks the DLog assumption. $\mathcal{R} = \mathcal{R}^{(\bar{\mathbf{\Pi}}(\cdot), \bar{\mathbf{\Pi}}^{-1}(\cdot))}$ has access to the non-programmable bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ and executes up to q_P-times adversary $\mathcal{A}$. (Recall that the bars of $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ indicate non-programmability.) We denote the i-th execution of $\mathcal{A}$ as $\mathcal{A}_i$. $\mathcal{A}_i = \mathcal{A}_i^{(\bar{\mathbf{\Pi}}(\cdot), \bar{\mathbf{\Pi}}^{-1}(\cdot))}$ has access to the non-programmable bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ of the SDLog game and $\mathcal{R}$ can observer all queries made by $\mathcal{A}$ to $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$. We will construct a meta-reduction $\mathcal{M}$ trying to solve the q_P-FBOMDL game by running the reduction $\mathcal{R}$, simulating the DLog experiment, the non-programmable bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ via lazy sampling, and the q_P executions $\mathcal{A}_i$ of adversary $\mathcal{A}$.

$\mathcal{M}^{\mathbf{Chal}, \mathbf{DL}(\cdot,\cdot)}(X)$ receives X as its q_P-FBOMDL input, has access to FBOMDL's challenge oracle **Chal** and discrete logarithm oracle **DL** and executes the reduction $\mathcal{R}^{\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1}}(X)$ with X as its DLog challenge. Reduction $\mathcal{R}$ in turn executes up to q_P adversaries $\mathcal{A}_i(\hat{X}_i)$ (simulated by $\mathcal{M}$) on an arbitrary group element $\hat{X}_i$ as its SDLog challenge. Meta reduction $\mathcal{M}$ and $\mathcal{A}_i(\hat{X}_i)$ simulated by $\mathcal{M}$ are shown in Figure 4. Since $\mathcal{R}$ is algebraic, it provides $\mathcal{A}_i$ (and hence also $\mathcal{M}$) with the representations $(a_i, b_i, c_{i,1}, \ldots, c_{i,i-1})_{i \in [q_P]}$ of $\hat{X}_i$. That is, $\hat{X}_i = g^{a_i} X^{b_i} \prod_{j<i} X_j^{c_{i,j}}$ for $a_i, b_i, c_{i,j} \in \mathbb{Z}_p$, where X_j is the j-th challenge returned by the **Chal** oracle. $\mathcal{R}$ will learn exactly one new X_j for each execution of $\mathcal{A}$ through its queries to $\mathbf{\Pi}$, so $\hat{X}_i$ can only depend on g, X, and $X_1, \ldots, X_{i-1}$.

Meta Reduction $\mathcal{M}^{\mathbf{Chal},\mathbf{DL}(\cdot,\cdot)}(X)$

1: Run $\mathcal{R}(X)$
2: **for** $i \in [q_P]$ **do**
3: Receive $\hat{X}_i$ from $\mathcal{R}$
4: Simulate $\mathcal{A}_i^{\bar{\mathbf{\Pi}}(\cdot),\bar{\mathbf{\Pi}}^{-1}(\cdot)}(\hat{X}_i)$ for $\mathcal{R}$
5: Receive solution x from $\mathcal{R}$
6: Solve equations for $x_1, \dots x_{q_P}$
7: **return** $(x, x_1, \dots, x_{q_P})$

Adversary $\mathcal{A}_i^{\bar{\mathbf{\Pi}}(\cdot),\bar{\mathbf{\Pi}}^{-1}(\cdot)}(\hat{X}_i)$ // $\hat{X}_i = g^{a_i} X^{b_i} \prod_{j<i} X_j^{c_{i,j}}$

8: $X_i \leftarrow \mathbf{Chal}$
9: $t_i^* \leftarrow \psi(\bar{\mathbf{\Pi}}(\varphi(X_i)))$ // via oracle query to $\bar{\mathbf{\Pi}}$
10: Make remaining $q_{\bar{\mathbf{\Pi}}} - 1$ dummy queries to $\bar{\mathbf{\Pi}}$
11: Make $q_{\bar{\mathbf{\Pi}}^{-1}}$ dummy queries to $\bar{\mathbf{\Pi}}^{-1}$
12: **if** $t_i^* = 0 \vee g \cdot \hat{X}_i^{\,t_i^*} = 1$ **then** // $\Leftrightarrow s_i^* = 0$
13: abort
14: $s_i^* \leftarrow \mathbf{DL}(g \cdot \hat{X}_i^{\,t_i^*}, X_i)$ // from FBOMDL game
15: **return** $\begin{cases} (s_i^*, t_i^*) & \text{with prob. } \epsilon_{\mathcal{A}} \\ \bot & \text{else} \end{cases}$

Fig. 4. Meta reduction $\mathcal{M}$ and simulated adversaries $\mathcal{A}_i$ against the SDLog assumption. **DL** and **Chal** are $\mathcal{M}$'s oracles from the FBOMDL assumption. If the $\mathcal{A}_i$ is run with the same randomness and public key as $\mathcal{A}_j$, then $\mathcal{M}$ simply replays the queries made by $\mathcal{A}_j$ and does not query the **Chal** or **DL** oracle.

$\mathcal{A}_i$ embeds one of the FBOMDL challenges X_i into its first query to $\bar{\mathbf{\Pi}}$.[5] Next, it makes $q_{\bar{\mathbf{\Pi}}} - 1$ dummy queries to $\bar{\mathbf{\Pi}}$ and $q_{\bar{\mathbf{\Pi}}^{-1}}$ queries to $\bar{\mathbf{\Pi}}^{-1}$. Here, all queries are chosen such that the group elements corresponding to them via φ are independent of the X_i, so the representation of all public keys for future executions of $\mathcal{A}$ still collapses as described above. After making all of its queries, $\mathcal{A}_i$ checks if the challenge X_i results in an invalid solution (line 12) and ends its execution if so. Otherwise, it uses one query to the **DL** oracle to obtain $s_i^* = \mathbf{DL}(g \cdot \hat{X}_i^{\,t_i^*}, X_i) = \frac{1+\hat{x}_i t_i^*}{x_i} \neq 0$, where $\hat{X}_j = g^{\hat{x}_j}$ and $X_j = g^{x_j} \in \mathbb{G}^*$. Next, it returns a valid SDLog solution $(s_i^*, t_i^*) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ relative to public key $\hat{X}_i$. Note that every $\mathcal{A}_i$ makes one **Chal** query and at most one **DL** query.

We can assume w.l.o.g. that all executions of $\mathcal{A}_i$ are distinct. (If $\mathcal{A}_i$ is run with the same randomness and public key as $\mathcal{A}_j$ for $j < i$, then $\mathcal{M}$ simply replays the same queries as made in $\mathcal{A}_j$. Specifically, it does not query the **Chal** oracle and sets $X_i = X_j$. This results in a correct simulation, since $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ is a non-programmable bRO, so $\mathcal{R}$ can only control the randomness and input of $\mathcal{A}_i$ and nothing else.)

The simulation of $\mathcal{A}_i$ is perfect, as it wins with exactly probability $\epsilon_{\mathcal{A}}$, unless $t_i^* = 0$ or $t_i^* \cdot \hat{x}_i + 1 = 0$ with $\hat{X}_i = g^{\hat{x}_i}$ (line 12). We call the overall abort BAD and the two sub-events BAD_1 and BAD_2 respectively. In order to analyze $\Pr[\text{BAD}]$, we assume that $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ is a random *function* instead of a random permutation. This allows us to assume that all output values are always chosen uniformly at random, but might incur an error if a collision occurs. The latter can be bounded by the birthday bound as $q^2/2^L$, where q is the total number of queries to $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ observed by $\mathcal{R}$.

We now analyze BAD_1 and BAD_2 for each individual execution of $\mathcal{A}_i$. Event BAD_1 occurs iff $t_i^* = f(X_i) = \psi(\bar{\mathbf{\Pi}}(\varphi(X_i))) = 0$. The random variable $y_i := \bar{\mathbf{\Pi}}(\varphi(X_i))$ is distributed uniformly random since $\bar{\mathbf{\Pi}}$ is a random function. Therefore, $\Pr[\text{BAD}_1] = \Pr[f(X_i) = 0] = \Pr[\psi(\bar{\mathbf{\Pi}}(\varphi(X_i))) = 0] = \Pr_{y_i \xleftarrow{\$} [0:2^L-1]}[\psi(y_i) = 0] \leq \epsilon_\psi$. Similarly, for each execution of $\mathcal{A}_i$, event BAD_2 occurs iff $\hat{x}_i t_i^* + 1 = 0$ with $t_i^* \neq 0$ and $\hat{x}_i \neq 0$. $\hat{x}_i$ is (implicitly) chosen by $\mathcal{R}$ *before* t_i^* is computed

[5] This fixed embedding is not exploitable by $\mathcal{R}$ since $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ is non-programmable

by $\mathcal{A}_i$ with its first query to $\bar{\mathbf{\Pi}}$, so it is independent of t_i^*. So in order for BAD_2 to occur, we have to bound the probability that $t_i^* = -\hat{x}_i^{-1}$. With the same argument as for BAD_1, we get $\Pr[\text{BAD}_2] = \Pr[t_i^* = -\hat{x}_i] \leq \epsilon_\psi$. So overall, $\Pr[\text{BAD}] \leq \Pr[\text{BAD}_1] + \Pr[\text{BAD}_2] \leq 2\epsilon_\psi$ for each execution of $\mathcal{A}_i$. A union bound over all executions of $\mathcal{A}$ and the birthday bound yields that the probability that BAD occurs in *any* execution is

$$\Pr[\text{BAD}] \leq 2q_P\epsilon_\psi + q^2/2^L.$$

Now assume that none of the $\mathcal{A}_i$ aborts, i.e., BAD does not happen. Then all $\mathcal{A}_i$ are simulated perfectly, so $\mathcal{R}$ will be successful with probability $\epsilon_\mathcal{R}$ and output x s.t. $g^x = X$. Let x_i ($i \in [q_P]$) be the unknown discrete logarithms of the FBOMDL challenges $X_i = g^{x_i}$ from line 8. Using the representations $a_i, b_i, (c_{i,j})_{j\in[q_P]}$ of $\hat{X}_i = g^{a_i}X^{b_i}\prod_{j<i} X_j^{c_{i,j}}$, $\mathcal{M}$ gets the equations

$$s_i^* x_i = 1 + t_i^* a_i + t_i^* b_i x + t_i^* \sum_{1\leq j<i} c_{i,j}x_j,$$

for $i \in [q_P]$ in the variables $x_1, \ldots, x_{q_P}$.

There are q_P equations in q_P variables, so there exists a unique solution unless its determinant is zero. Looking at these equations as a matrix, we see that it is a triangle matrix with the s_i^* on the diagonal. Its determinant is $\prod_{i\in[q_P]} s_i^* \neq 0$ since $s_i^* \neq 0$. $\mathcal{M}$ computes the x_i using standard linear algebra and returns all $q_P + 1$ discrete logarithms $(x, x_1, \ldots, x_{q_P})$ as its solution. Hence, $\mathcal{M}$ wins the q_P-FBOMDL game if $\mathcal{R}$ is successful and BAD occurs in no execution of $\mathcal{A}$, so

$$\epsilon_\mathcal{M} \geq \epsilon_\mathcal{R} - \Pr[\text{BAD}] \geq \epsilon_\mathcal{R} - 2q_P\epsilon_\psi - \frac{q^2}{2^L},$$

which concludes the proof. □

4.2 NMA $\not\Rightarrow$ CMA

Next, we prove that with non-programmable bijective random oracles, GenDSA's UF-NMA security does not imply UF-CMA security. By transitivity, this also implies that UF-1CMA security does not imply UF-CMA security without programmability of $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$.

Specifically, the following theorem states that there cannot exist an algebraic reduction $\mathcal{R}$ that reduces the UF-CMA security of GenDSA to its UF-NMA security that does not program the random bijection $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$, unless the DLog assumption does not hold in the group.

Theorem 3 (UF-NMA $\overset{\Pi \text{ non-progr.}}{\not\Rightarrow}$ UF-CMA)**.** *Let $\mathbf{H}, \mathbf{H}' : \{0,1\}^* \to \mathbb{Z}_p$ be hash functions modeled as a programmable random oracle and $f = \psi \circ \mathbf{\Pi} \circ \varphi$ be a conversion function with semi-injective $\varphi : \mathbb{G}^* \to \{0,1\}^L$, $\psi : [0 : 2^L - 1] \to \mathbb{Z}_p$ and $\mathbf{\Pi} : \{0,1\}^L \to [0 : 2^L - 1]$ a* non-programmable *bijective random oracle*

$(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$. *If there exists an algebraic reduction* $\mathcal{R}$ *that* $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, q'_{\bar{\mathbf{\Pi}}}, q'_{\bar{\mathbf{\Pi}}^{-1}}, q'_{\mathbf{H}})$-*breaks the* UF-NMA *security of* GenDSA *(instantiated with* $\mathbf{H}$*) with* q_P*-time access to an adversary* $\mathcal{A}$ *that* $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_s, q_{\bar{\mathbf{\Pi}}}, q_{\bar{\mathbf{\Pi}}^{-1}}, q_{\mathbf{H}'})$-*breaks the* UF-CMA *security of* GenDSA *(instantiated with* $\mathbf{H}'$*), then there exists an algorithm* $\mathcal{M}$ *that* $(t_{\mathcal{M}}, \epsilon_{\mathcal{M}})$-*breaks the* DLog *assumption where*

$$\epsilon_{\mathcal{M}} \geq \epsilon_{\mathcal{R}} - (2(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}})q'_{\mathbf{H}} + 1)\epsilon_{\psi} - q'_{\mathbf{H}}/p - q^2/2^L, \quad t_{\mathcal{M}} \approx t_{\mathcal{R}},$$

where $q = q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}} + q_P(q_{\bar{\mathbf{\Pi}}} + q_{\bar{\mathbf{\Pi}}^{-1}})$ *is total number of queries made to* $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ *by* $\mathcal{R}$ *and* $\mathcal{A}$*.*

Since the goals of $\mathcal{R}$ and $\mathcal{A}$ are identical, i.e. producing a valid signature for a given key, we cannot generally use the output of $\mathcal{R}$ to make $\mathcal{M}$ work, since if $\mathcal{R}$ just forwards its own challenge key, the solution of $\mathcal{A}$ is also a solution for $\mathcal{R}$. So in this case, the idea of the meta-reduction is to ask for two signatures on the same message and then extract a DLog solution from the oracle queries and the signatures that $\mathcal{R}$ produces. Disallowing the reduction from programming the bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ is necessary, as otherwise the proof from [20] applies. For the same reason, an analogue of Theorem 2 is impossible here, as the reduction of [20] is non-rewinding.

Proof. Let $\mathcal{A}$ be an adversary that $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_s, q_{\bar{\mathbf{\Pi}}}, q_{\bar{\mathbf{\Pi}}^{-1}}, q_{\mathbf{H}'})$-breaks the UF-CMA security of GenDSA and let $\mathcal{R}$ be the security reduction that $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, q'_{\bar{\mathbf{\Pi}}}, q'_{\bar{\mathbf{\Pi}}^{-1}}, q'_{\mathbf{H}})$-breaks the UF-NMA security of GenDSA. $\mathcal{A} = \mathcal{A}^{\mathbf{H}', \bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1}}(\mathsf{vk}')$ has access to the non-programmable bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ and the programmable random oracle $\mathbf{H}'$ of the UF-CMA game. $\mathcal{R} = \mathcal{R}^{\mathbf{H}, \bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1}}$ has access to the non-programmable bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ and the random oracle $\mathbf{H}$ of the UF-NMA game, internally simulates the programmable random oracle $\mathbf{H}'$ of the UF-CMA game (accessed by $\mathcal{A}$) and observes all queries made by $\mathcal{A}$ to $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$. Meta-reduction $\mathcal{M}$ tries to solve the DLog game by running the reduction $\mathcal{R}$, simulating the UF-NMA experiment together with its RO $\mathbf{H}$ and the non-programmable bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$, and simulating adversary $\mathcal{A}$.

$\mathcal{M}(X)$ receives a DLog challenge $X = g^x$ as input and sets $\mathsf{vk} = X$. It runs $\mathcal{R}^{\mathbf{H}, \bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1}}(\mathsf{vk})$ with the signature key vk as input and simulates the random oracle $\mathbf{H}$ and the bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ via lazy sampling. It keeps a list L where it stores previous queries and assorted information on the queries. When $\mathcal{R}$ queries $\bar{\mathbf{\Pi}}(z)$ for a fresh $z \in \{0,1\}^L$, $\mathcal{M}$ samples a fresh random $y \in [0 : 2^L - 1]$. If $z \notin \varphi(\mathbb{G}^*)$, it stores $(\bot, z, \bot, y)$ in L and returns y. If $z \in \varphi(\mathbb{G}^*)$, $\mathcal{R}$ has to provide a representation a, b s.t. $\varphi(g^a \cdot X^b) = z$ and $\mathcal{M}$ stores $(g^a X^b, z, (a,b), y)$ (and $(g^{-a} X^{-b}, z, (-a,-b), y)$ if φ is 2-to-1) in L. When $\mathcal{R}$ queries $\bar{\mathbf{\Pi}}^{-1}(z')$ on a fresh $z' \in [0 : 2^L - 1]$, $\mathcal{M}$ samples a random $x \in \{0,1\}^L$. If $x \notin \varphi(\mathbb{G}^*)$, $\mathcal{M}$ sets $\bar{\mathbf{\Pi}}^{-1}(z') = x$ (i.e. stores $(\bot, x, \bot, z')$ in L) and returns x. Otherwise, it samples $(a,b) \xleftarrow{\$} \mathbb{Z}_p^2$ s.t. $a + xb \neq 0$, computes $R := g^a \cdot X^b$ and sets $\bar{\mathbf{\Pi}}^{-1}(z') := \varphi(R)$. In the latter case, it additionally stores $(R, \varphi(R), (a,b), z')$ (and $(R^{-1}, \varphi(R), (-a,-b), \psi(z'))$ if φ is 2-to-1) in L and returns $\varphi(R)$ to $\mathcal{R}$. Note that this simulation of $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ implements a random *function* instead of a random bijection. By the birthday bound, this

incurs a statistical loss $q^2/2^L$, where $q = q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}} + q_P(q_{\bar{\mathbf{\Pi}}} + q_{\bar{\mathbf{\Pi}}^{-1}})$ is the total number of queries made to $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ by $\mathcal{R}$ and $\mathcal{A}$ due to it possibly introducing collisions.

When $\mathcal{R}$ invokes the UF-NMA adversary $\mathcal{A} = \mathcal{A}^{\mathbf{H}', \bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1}}$, $\mathcal{M}$ simulates it for $\mathcal{R}$. Note that $\mathcal{A}$ has access to the same oracles $\bar{\mathbf{\Pi}}$ and $\bar{\mathbf{\Pi}}^{-1}$ as $\mathcal{R}$, since we model $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ as a non-programmable bRO, but $\mathbf{H}'$ is fully controlled by $\mathcal{R}$. $\mathcal{R}$ starts $\mathcal{A}$ by sending a public key vk'. Since $\mathcal{R}$ is algebraic, it also outputs a representation of vk' relative to all its inputs, including group elements calculated from answers to $\bar{\mathbf{\Pi}}^{-1}$ queries. Since $\mathcal{M}$ knows all discrete logarithms except for X from the simulation described above, this representation implicitly collapses to one relative to g and X:

$$\mathsf{vk}' = g^{\alpha} X^{\beta} \in \mathbb{G} \; . \tag{2}$$

At this point, $\mathcal{M}$ reacts differently depending on whether $\beta = 0$ or not.

Case 1: $\beta = 0$. In this case, $\mathcal{R}$ effectively does not need the adversary $\mathcal{A}$ as it can compute any signature produced by $\mathcal{A}$ on its own. (Note that the simulation of adversary $\mathcal{A}$ is trivial in this case as $\mathcal{R}$ is algebraic and therefore $\mathcal{M}$ knows $\mathsf{sk}' = \alpha$, which also makes rewinding/multiple executions trivial.) Meta-reduction $\mathcal{M}$ continues the execution of $\mathcal{R}$ until it terminates and produces a forgery $(m^*, \sigma^* = (s^*, t^*))$. Assume the forgery is valid, i.e., it satisfies

$$t^* = f(R^*), \text{ where } R^* = \left(g^{\mathbf{H}(m^*)} X^{t^*} \right)^{\frac{1}{s^*}} \; . \tag{3}$$

We we will now use another case distinction to show that $\mathcal{M}$ can solve its DLog challenge in Case 1 except with probability $2\epsilon_\psi q'_{\mathbf{H}}(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}})$.

Case 1.a: $\beta = 0$ and $\nexists (R^*, *, (a^*, b^*), y^*) \in L$ with $\psi(y^*) = t^*$. In this case, t^* does not correspond to a query made by $\mathcal{R}$ to $\bar{\mathbf{\Pi}}$ or $\bar{\mathbf{\Pi}}^{-1}$ and $\mathcal{M}$ *can not* compute a discrete logarithm solution in this case and aborts. We call this case Bad_1. We will show that $\Pr[\text{Bad}_1] \leq \epsilon_\psi$. First, note that since $\bar{\mathbf{\Pi}}$ is random, the random variable $y^* = \bar{\mathbf{\Pi}}(\varphi(R^*))$ is distributed uniformly over $[0 : 2^L - 1]$. Therefore, we have

$$\Pr[\text{Bad}_1] = \Pr[\psi(\bar{\mathbf{\Pi}}(\varphi(R^*)) = t^*] = \Pr[\psi(y^*) = t^*] \leq \epsilon_\psi.$$

Case 1.b: $\beta = 0$ and $\exists (R^*, *, (a^*, b^*), y^*) \in L$ with $\psi(y^*) = t^*$. Since $R^* = g^{a^*} X^{b^*}$, (3) simplifies to

$$s^*(a^* + b^* x) = \mathbf{H}(m^*) + t^* x \; . \tag{4}$$

Consequently, unless $s^* b^* = t^*$ (which we denote as event Bad_2), $\mathcal{M}$ can extract $x = \frac{s^* a^* - \mathbf{H}(m^*)}{t^* - s^* b^*}$ and solve its DLog challenge. To finish the analysis of Case 1.b, it remains to show

$$\Pr[\text{Bad}_2] = \Pr[t^* = s^* b^*] \leq 2\epsilon_\psi q'_{\mathbf{H}}(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}}) + q'_{\mathbf{H}}/p. \tag{5}$$

Assuming $t^* = s^* b^*$, the verification equation (4) can be rearranged as

$$t^* a^* = \mathbf{H}(m^*) b^*. \tag{6}$$

For every fixed m^*, a^*, b^*, the value t^* satisfying (6) is unique unless $a^* = 0$. However in that case, (6) implies that either $b^* = 0$ or $\mathbf{H}(m^*) = 0$. The first case already reveals $x = \mathbf{H}(m^*)/t^*$ from (4). The second case only occurs with probability $q'_{\mathbf{H}}/p$, since $\mathbf{H}$ is a random oracle that $\mathcal{R}$ can not program and we also abort in this case. Hence, conditioned on $\mathbf{H}(m^*) \neq 0$, $\Pr[\psi(\bar{\mathbf{\Pi}}(\varphi(g^{a^*}X^{b^*})) = t^*] \leq \epsilon_\psi$, where the probability is over the random function $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$. $\mathcal{R}$ can chose from $q'_{\mathbf{H}}$ many tuples $(m^*, \mathbf{H}(m^*))$ ($\mathbf{H}$ is the oracle provided by the UF-NMA game to $\mathcal{R}$ so it is not programmable) and from $2(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}})$ many tuples (a^*, b^*) (φ is semi-injective, i.e., there are potentially 2 preimages for every t^*). So a union bound over the number of tuples yields that the probability that $\mathcal{R}$ finds a tuple (a^*, b^*, t^*, m^*) satisfying (6) is at most $2\epsilon_\psi q'_{\mathbf{H}}(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}}) + q'_{\mathbf{H}}/p$.

Overall, $\mathcal{M}$ is successful in Case 1 iff BAD_1 and BAD_2 do not happen.

Case 2: $\beta \neq 0$. In this case, $\mathcal{R}$ effectively uses the adversary $\mathcal{A}$. $\mathcal{M}$ continues simulating $\mathcal{A}$ by picking an arbitrary message m satisfying $\mathbf{H}'(m) \neq 0$ and asks for two signatures $\sigma_1 = (s_1, t_1)$ and $\sigma_2 = (s_2, t_2)$ on the same m. Without loss of generality, we assume that $\sigma_1 \neq \sigma_2$. Next, $\mathcal{M}$ aborts the execution of $\mathcal{A}$ and $\mathcal{R}$. Note that $\mathcal{M}$ has simulated the UF-NMA game and an UF-CMA adversary perfectly up to this point. Again, rewinding/multiple executions are handled trivially, as every instance of $\mathcal{A}$ only queries two signatures on a random message m with $\mathbf{H}'(m) \neq 0$.

Let $r_1, r_2 \in \mathbb{Z}_p$ s.t. $R_i := g^{r_i}$ and $f(R_i) = t_i$ for $i \in \{1, 2\}$. Note that there might be multiple such r_1, r_2 as f is generally not injective. We assume that the two values chosen here are the ones used by the reduction in its answer and its queries to f (if they were made, see case distinction below). Assuming the signatures σ_1 and σ_2 are valid, we have

$$\forall i \in \{1, 2\} : \quad s_i r_i - \mathbf{H}'(m) = t_i\alpha + t_i\beta x, \tag{7}$$

with $s_i \neq 0$, $t_i \neq 0$ and α, β are from (2). Note that $\mathcal{M}$ only knows $R_i = g^{r_i}$ and not r_i. We now distinguish two cases depending on whether R_1 and R_2 have been recorded as a query to $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ or not.

- Case 2.A: $\nexists (R_i, z_i, (a_i, b_i), z'_i) \in L$ s.t. $g^{r_i} = R_i$ for at least one $i \in \{1, 2\}$
- Case 2.B: $\exists (R_i, z_i, (a_i, b_i), z'_i) \in L$ s.t. $g^{r_i} = R_i$, i.e. $r_i = a_i + xb_i$ for $i \in \{1, 2\}$

Case 2.A: This case is almost identical to Case 1.A, as $\mathcal{M}$ can again not extract a DLog solution and aborts. We call this event BAD_3. As in Case 1.A, we have that the probability of $\mathcal{R}$ succeeding in this case is upper bounded by ϵ_ψ, because $\mathcal{R}$ has to simulate a proper signing oracle, so the signatures σ_1, σ_2 need to verify and with the same argument as in the other case, we get

$$\Pr[\text{BAD}_3] \leq \epsilon_\psi.$$

Case 2.B: In this case, $\mathcal{R}$ made queries to $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ to compute $f(R_i)$ for both signatures (or $f^{-1}(t_i)$ respectively). Therefore due to the way $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ is programmed and the fact that the reduction is algebraic, $\mathcal{M}$ knows a representation

of $R_i = g^{a_i} X^{b_i}$, which it can use to solve for x. Plugging $r_i = a_i + xb_i$ for $i \in \{1, 2\}$ into (7) we obtain

$$\forall i \in \{1, 2\} : \quad -\mathbf{H}'(m) - t_i\alpha + s_i a_i = (t_i\beta - s_i b_i)x \; . \tag{8}$$

$\mathcal{M}$ can solve any of these two equations for x (and hence solve its own challenge), unless

$$\forall i \in \{1, 2\} : \quad : t_i\beta = s_i b_i \; . \tag{9}$$

We call this event BAD_4. Assuming (9) is the case, (8) simplifies to $-\mathbf{H}'(m) - t_i\alpha + s_i a_i = 0$ which implies

$$\mathbf{H}'(m)b_i = t_i \Delta_i \tag{10}$$

$$t_2 b_1 \Delta_2 = t_1 b_2 \Delta_1, \tag{11}$$

where $\Delta_i := a_i\beta - b_i\alpha$.

Assume $b_1 \neq 0$ and $\Delta_2 \neq 0$. By (11), t_2 directly depends on t_1 and the exponents a_2, b_2, which are used in the bRO query that outputs t_2 (or chosen at random by $\mathcal{M}$ when a $\psi^{-1}(t_2)$ is queried to $\bar{\mathbf{\Pi}}^{-1}$). Most importantly, $\mathcal{R}$ has to fix w.l.o.g. t_1 before it can choose t_2. So similar to Case 1.B, the probability that $\mathcal{R}$ computes a t_2 which satisfies equation (11) is upper bounded by $2\epsilon_\psi(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}})$ since it can only evaluate $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ up to $q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}}$ many times. Therefore

$$\Pr[\text{BAD}_4] \leq 2\epsilon_\psi(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}})$$

and $\mathcal{M}$ extracts a DLog solution unless BAD_4 occurs. Note that the bound does not depend on $q_{\mathbf{H}}$, because both signatures are for the same message and $\mathbf{H}(m^*)$ cancels in (11).

It remains to argue why we could assume $b_1 \neq 0$ and $\Delta_2 \neq 0$. If $b_1 = 0$, then (9) and $t_1 \neq 0$ implies $\beta = 0$ contradicting (2). If $\Delta_2 = 0$, then (10) implies $b_2 = 0$ (and hence $\beta = 0$ as in the case above) or $\mathbf{H}'(m) = 0$ (contradicting $\mathbf{H}'(m) \neq 0$).

As in Case 1, $\mathcal{M}$ is successful in Case 2, if BAD_3 and BAD_4 do not occur.

Combining the probabilities for the two cases and the collision bound, the theorem follows:

$$\begin{aligned}\epsilon_{\mathcal{M}} &\geq \epsilon_{\mathcal{R}} - \max_{i \in \{1,3\}} (\text{BAD}_i + \text{BAD}_{i+1}) - q^2/2^L \\ &\geq \epsilon_{\mathcal{R}} - \epsilon_\psi(2q'_{\mathbf{H}}(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}}) + 1) - q'_{\mathbf{H}}/p - q^2/2^L\end{aligned}$$

□

Remark 1. Note that it is necessary for the meta reduction to program the responses to $\bar{\mathbf{\Pi}}^{-1}$ queries in order to know the discrete logarithms of the R corresponding to the responses. Suppose that this was not allowed. Then if the reduction makes a query $\bar{\mathbf{\Pi}}^{-1}(z)$ for some t, $\mathcal{M}$ does not know a representation for the resulting group element R with $\varphi(R) = \bar{\mathbf{\Pi}}^{-1}(z)$. So if a signature provided by the reduction includes this R, the meta-reduction will not be able extract a solution. Yet, it seems intuitive that an adversary that uses such a random group

element already needs to know its discrete logarithm, because a valid solution fulfills a non-trivial equation with this group element. So such an adversary would break the DLog problem in $\mathbb{G}$. However, in order to "use" this (implicit) DLog adversary, we need to embed a DLog challenge in the random oracle. But without programming, the challenge will only occur with negligible probability, leaving us with the same problem.

We avoid this seemingly technical problem by allowing the meta-reduction to program the random oracle. Note that we only use it for exactly this purpose, namely so that the meta-reduction knows the discrete logarithms of all results of $\bar{\mathbf{\Pi}}^{-1}$ queries.

4.3 Single-User security does not tightly imply multi-user security

Lastly, we show that for GenDSA, single user security does not tightly imply multi-user security with algebraic, non-programming reductions.

The following theorem states that there cannot exist an algebraic reduction from the multi-user UF-NMA-security of GenDSA to its single-user security that does not program the random bijection $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ and loses less then a factor linear in the number of users, unless the FBOMDL assumption does not hold.

Theorem 4 (UF-NMA $\overset{\mathbf{loss}<n}{\not\Rightarrow}$ n-UF-NMA). *Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function modeled as a programmable random oracle $\mathbf{H}$ and $f = \psi \circ \mathbf{\Pi} \circ \varphi$ be a conversion function with semi-injective $\varphi : \mathbb{G}^* \to \{0,1\}^L$, $\psi : [0 : 2^L - 1] \to \mathbb{Z}_p$ and $\mathbf{\Pi} : \{0,1\}^L \to [0 : 2^L - 1]$ modeled by a* non-programmable *bijective random oracle $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$. If there exists an algebraic reduction $\mathcal{R}$ that $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, q'_{\mathbf{\Pi}}, q'_{\mathbf{\Pi}^{-1}}, q'_{\mathbf{H}})$-breaks the* UF-NMA *security of GenDSA with q_P-time black-box access to an adversary $\mathcal{A}$ that $(n, t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_{\bar{\mathbf{\Pi}}}, q_{\bar{\mathbf{\Pi}}^{-1}}, q_{\mathbf{H}'})$-breaks the n-UF-NMA security of GenDSA, then there exists an algorithm $\mathcal{M}$ that $(t_{\mathcal{M}}, \epsilon_{\mathcal{M}})$-breaks the q_P-FBOMDL assumption where*

$$\epsilon_{\mathcal{M}} \geq \epsilon_{\mathcal{R}} - \frac{q_P}{n}\epsilon_{\mathcal{A}} - \left(2(q'_{\mathbf{\Pi}} + q'_{\mathbf{\Pi}^{-1}})q'_{\mathbf{H}} + q_P + 2\right)\epsilon_{\psi} - q^2/2^L, t_{\mathcal{M}} \approx t_{\mathcal{R}} + q_P t_{\mathcal{A}}$$

where $q = (q'_{\mathbf{\Pi}} + q'_{\mathbf{\Pi}^{-1}} + q_P(q_{\bar{\mathbf{\Pi}}} + q_{\bar{\mathbf{\Pi}}^{-1}}))$ is the total amount of queries to $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ made by $\mathcal{A}$ and $\mathcal{R}$.

As an orthogonal impossibility result we prove that single user security does not tightly imply multi-user security with algebraic, non-rewinding reductions. Specifically, the following theorem states that there cannot exist an algebraic reduction from the multi-user UF-NMA-security of GenDSA to its single-user security that does not rewind the multi-user adversary $\mathcal{A}$ and loses less than a factor linear in the number of users, unless the FBOMDL assumption does not hold.

Theorem 5. *Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function modeled as a programmable random oracle $\mathbf{H}$ and $f = \psi \circ \mathbf{\Pi} \circ \varphi$ be a conversion function with semi-injective $\varphi : \mathbb{G}^* \to \{0,1\}^L$, $\psi : [0 : 2^L - 1] \to \mathbb{Z}_p$ and $\mathbf{\Pi} : \{0,1\}^L \to [0 : 2^L - 1]$ modeled by a* programmable *bijective random oracle $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$. If there exists an algebraic*

reduction $\mathcal{R}$ *that* $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, q_{\mathbf{\Pi}}, q_{\mathbf{\Pi}^{-1}}, q_{\mathbf{H}})$*-breaks the* UF-NMA *security of* GenDSA *with one-time black-box access to an adversary* $\mathcal{A}$ *that* $(n, t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_{\mathbf{\Pi}'}, q_{\mathbf{\Pi}'^{-1}}, q_{\mathbf{H}'})$*-breaks the* n*-*UF-NMA *security of* GenDSA*, then there exists an algorithm* $\mathcal{M}$ *that* $(t_{\mathcal{M}}, \epsilon_{\mathcal{M}})$*-breaks the* 1-FBOMDL *assumption where*

$$\epsilon_{\mathcal{M}} \geq \epsilon_{\mathcal{R}} - \frac{1}{n}\epsilon_{\mathcal{A}} - (2(q_{\mathbf{\Pi}} + q_{\mathbf{\Pi}^{-1}})q_{\mathbf{H}} + 1)\,\epsilon_{\psi} - q^2/2^L, \quad t_{\mathcal{M}} \approx t_{\mathcal{R}} + t_{\mathcal{A}}$$

where $q = q_{\mathbf{\Pi}} + q_{\mathbf{\Pi}^{-1}}$ is the total number of queries to $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ made by $\mathcal{R}$. We now prove Theorem 4. The proof of Theorem 5 is similar and can be found in Supplementary Material B.2.

Remark 2. The results of Theorem 4 and Theorem 5 also hold for UF-CMA and n-UF-CMA security with small adjustments. The main difference is that the meta reduction now uses the $q_S + q_P$-FBOMDL (resp. q_S + 1-FBOMDL) assumption, as we need the additional oracle queries to answer signature queries. The additional challenges used in the signatures do not pose a problem when eventually extracting the discrete logarithm solutions. Similar to the proof of Theorem 1, they form a random upper triangular matrix, so unless the forgery lies in the span of the new signatures (which can only happen with negligible probability) a solution can be extracted in the same way as in Theorem 4 (resp. Theorem 5).

Proof (of Theorem 4). Let $\mathcal{A}$ be an adversary that $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_{\bar{\mathbf{\Pi}}}, q_{\bar{\mathbf{\Pi}}^{-1}}, q_{\mathbf{H}})$-breaks the n-UF-NMA security of GenDSA, and let $\mathcal{R}$ be the algebraic security reduction that $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, q'_{\bar{\mathbf{\Pi}}}, q'_{\bar{\mathbf{\Pi}}^{-1}}, q'_{\mathbf{H}})$-breaks the UF-NMA security of GenDSA. $\mathcal{A} = \mathcal{A}^{\mathbf{H}'(\cdot),\bar{\mathbf{\Pi}}(\cdot),\bar{\mathbf{\Pi}}^{-1}(\cdot)}$ has access to the non-programmable bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ and the random oracle $\mathbf{H}'$ of the n-UF-NMA game. $\mathcal{R} = \mathcal{R}^{\mathbf{H}(\cdot),\bar{\mathbf{\Pi}}(\cdot),\bar{\mathbf{\Pi}}^{-1}(\cdot)}$ has access to the same non-programmable bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ and the programmable random oracle $\mathbf{H}$ of the UF-NMA game and internally simulates the programmable RO $\mathbf{H}'$, which is accessed by $\mathcal{A}$. Meta reduction $\mathcal{M}$ tries to solve the q_P-FBOMDL assumption by running the reduction $\mathcal{R}$, simulating the UF-NMA game, the adversary $\mathcal{A}$ up to q_P times and the oracles $\mathbf{H}$ and $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$.

Meta-reduction $\mathcal{M}$ and simulated $\mathcal{A}$ are described in Figure 5. Note that since $\mathcal{A}$ is simulated by $\mathcal{M}$, it has access to the **DL** and **Chal** oracles. The meta reduction $\mathcal{M}$ gets a FBOMDL challenge X as input and has to output $x, x_1, \ldots, x_{q_P}$ s.t. $X = g^x$ and $X_i = g^{x_i}$, where the X_i are returned by its **Chal** oracle. It simulates the UF-NMA game for $\mathcal{R}$ by setting $\mathsf{vk} = X$ and runs $\mathcal{R}^{\mathbf{H},\bar{\mathbf{\Pi}},\bar{\mathbf{\Pi}}^{-1}}(\mathsf{vk})$. $\mathcal{M}$ simulates the bRO $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ and the regular random oracle $\mathbf{H}$ via lazy sampling. When $\mathcal{R}$ queries $\bar{\mathbf{\Pi}}$ on a fresh bitstring z with preimage group element R under φ, it has to provide a representation $a, b, c_1, \ldots, c_{q_P}$ s.t. $R = g^a \cdot X^b \cdot \prod_{j<i} X_j^{c_j}$, which $\mathcal{M}$ stores together with the random answer z'. If no such representation exists, $\mathcal{M}$ only samples z' and returns it. Similarly, when $\mathcal{R}$ queries $\bar{\mathbf{\Pi}}^{-1}$ on some z', $\mathcal{M}$ first samples a random $z \in \{0,1\}^L \setminus L$ (i.e. a random z that it hasn't returned yet) and if $z \in \varphi(\mathbb{G}^*)$ $\mathcal{M}$ chooses random $(a, b) \in \mathbb{Z}_p^2$ s.t. $R' = g^a X^b \neq 1$ and returns $\varphi(R')$. Otherwise, it simply returns z. In both cases, $\mathcal{M}$ stores the input/output pairs together with the potential representation in a

Meta Reduction $\mathcal{M}^{\mathbf{DL}(\cdot,\cdot),\mathbf{Chal}}(X)$

$L = \emptyset$
Run $\mathcal{R}^{\mathbf{H}(\cdot),\bar{\mathbf{\Pi}}(\cdot),\bar{\mathbf{\Pi}}^{-1}(\cdot)}(X)$
for $i \in [q_P]$ **do**
  Simulate $\mathcal{A}_i^{\mathbf{H}'(\cdot),\bar{\mathbf{\Pi}}(\cdot),\bar{\mathbf{\Pi}}^{-1}(\cdot)}((\mathsf{vk}_{i,j})_{j\in[n]})$
  if $\mathcal{A}_i$ aborts **then**
    `abort`
Receive forgery $(m^*, \sigma^* = (s^*, t^*))$ from $\mathcal{R}$
if $\exists i \in [q_P]$ s.t. $\sigma^* = (s_i, t_i)$ **then** *// $\mathcal{R}$ guessed n_i^**
  `abort`
if $\nexists (R^*, \varphi(R^*), (*, \dots, *), \psi^{-1}(t^*)) \in L$ **then**
  `abort`
Solve verification equations for $x, x_1, \dots, x_{q_P}$
return $(x, x_1, \dots, x_{q_P})$

Oracle $\bar{\mathbf{\Pi}}(z)$

if $\nexists (*, z, *, z') \in L$ **then**
  $z' \xleftarrow{\$} [0 : 2^L - 1]$
  if $z \in \varphi(\mathbb{G}^*)$ **then** *// $\mathcal{R}$ provides representation*
    Let $z = \varphi(Y) = \varphi(g^a X^b \prod_{j\in[q_P]} X_j^{c_j})$
    $L \xleftarrow{+} (Y, z, (a, b, (c_j)_{j\in[q_P]}), z')$
    $L \xleftarrow{+} (Y^{-1}, z, (-a, -b, (-c_j)_{j\in[q_P]}), z')$
  else
    $L \xleftarrow{+} (\bot, z, \bot, z')$
Let $(*, z, *, z') \in L$
return z'

Adv. $\mathcal{A}_i^{\mathbf{H}'(\cdot),\bar{\mathbf{\Pi}}(\cdot),\bar{\mathbf{\Pi}}^{-1}(\cdot)}(\mathsf{vk}_{i,j})_{j\in[n]})$

// $\mathsf{vk}_{i,j} = g^{a_{i,j}} X^{b_{i,j}} \prod_{k<i} X_k^{c_{i,j,k}}$

for $j \in [q_{\mathbf{H}}]$ **do**
  $m_j \xleftarrow{\$} \mathcal{M}$, $h_j \leftarrow \mathbf{H}'(m_j)$
$n_i^* \xleftarrow{\$} [n]$; $k_i^* \xleftarrow{\$} [q_{\bar{\mathbf{\Pi}}}]$; $j_i^* \xleftarrow{\$} [q_{\mathbf{H}}]$
for $j \in [q_{\bar{\mathbf{\Pi}}}] \setminus k_i^*$ **do**
  $r_j \xleftarrow{\$} \mathbb{Z}_p, R_j \leftarrow g^{r_j}$
  $t_j \leftarrow \psi(\bar{\mathbf{\Pi}}(\varphi(R_j)))$ *// via oracle query to $\bar{\mathbf{\Pi}}$*
$X_i \leftarrow \mathbf{Chal}$
$R_{k_i^*} \leftarrow X_i$, $t_{k_i^*} \leftarrow \psi(\bar{\mathbf{\Pi}}(\varphi(X_i)))$ *// Embed challenge*
if $t_{k_i^*} = 0$ **then**
  `abort`
Make $q_{\bar{\mathbf{\Pi}}^{-1}}$ dummy queries to $\bar{\mathbf{\Pi}}^{-1}$
$t_i \leftarrow t_{k_i^*}$
$h_i \leftarrow h_{j_i^*}$; $m_i \leftarrow m_{j_i^*}$
if $g^{h_i} \cdot \mathsf{vk}_{i,n_i^*}^{t_i} = 1$ **then**
  `abort`
$s_i \leftarrow \mathbf{DL}(g^{h_i} \cdot \mathsf{vk}_{i,n_i^*}^{t}, X_i)$ *// From* FBOMDL *game*
$\sigma_i \leftarrow (s_i, t_i)$
return $\begin{cases} (n_i^*, m_i, \sigma_i) & \text{with prob. } \epsilon_{\mathcal{A}} \\ \bot & \text{else} \end{cases}$

Oracle $\bar{\mathbf{\Pi}}^{-1}(z')$

if $\nexists (*, z, *, z') \in L$ **then**
  $x \xleftarrow{\$} \{0,1\}^L$
  if $x \notin \varphi(\mathbb{G}^*)$ **then**
    $L \xleftarrow{+} (\bot, x, \bot, z')$
  else
    $(a, b) \xleftarrow{\$} \mathbb{Z}_p^2$ s.t. $g^a \cdot X^b \neq 1$
    $z \leftarrow \varphi(g^a X^b)$
    $L \xleftarrow{+} (g^a X^b, z, (a, b, 0, \dots, 0), z')$
    $L \xleftarrow{+} (g^{-a} X^{-b}, z, (-a, -b, 0, \dots, 0), z')$
Let $(*, z, *, z') \in L$
return z

Fig. 5. Meta reduction $\mathcal{M}$ and simulated n-UF-NMA adversary for Theorem 4. For ease of notation, we group queries to $\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1}$ and $\mathbf{H}$ and write the challenge queries separately, but assume that $\mathcal{A}$ interweaves them at position k_i^* and j_i^* respectively. If two executions of $\mathcal{A}$ receive the same randomness, public keys and answers to all $\mathbf{H}$ queries, then $\mathcal{M}$ replays the previous execution of $\mathcal{A}$ and makes no additional **Chal** query.

list L in order to keep the random oracles consistent. Note that it is necessary to first sample a uniformly random $x \in \{0,1\}^L$ and check if it lies in $\varphi(\mathbb{G}^*)$ to keep the distribution of $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ consistent. Furthermore, $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ is simulated as a random *function*, instead of a random permutation. By the birthday bound, this incurs a statistical error of $q^2/2^L$, where $q = q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}} + q_P(q_{\bar{\mathbf{\Pi}}} + q_{\bar{\mathbf{\Pi}}^{-1}})$ is the total number of queries made to $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ by $\mathcal{A}$ and $\mathcal{R}$ due to the possibility of collisions. The random oracle $\mathbf{H}$ is simulated honestly, i.e. via lazy sampling without programming any values.

Eventually, $\mathcal{R}$ invokes up to q_P copies $\mathcal{A}_i^{\mathbf{H}', \bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1}}((\mathsf{vk}_{i,j})_{j\in[n]})$ of the adversary $\mathcal{A}$ with n verification keys of its choosing. Note that all adversaries get access to the same $\bar{\mathbf{\Pi}}$ and $\bar{\mathbf{\Pi}}^{-1}$ because it is non-programmable for $\mathcal{R}$, but $\mathbf{H}'$ is fully controlled by $\mathcal{R}$. Since $\mathcal{R}$ is algebraic, it also outputs representations of all keys, so let $\mathsf{vk}_{i,j} = g^{a_{i,j}} \cdot X^{b_{i,j}} \cdot \prod_{k\in[q_P]} X_k^{c_{i,j,k}}$ for $i \in [q_P], j \in [n]$. To be precise, $\mathcal{R}$ provides a representation relative to $g, X, X_1, \ldots, X_{q_P}$ and all elements it can compute from outputs of $\bar{\mathbf{\Pi}}^{-1}$. Due to the programming described above, $\mathcal{M}$ knows a representation of all these group elements relative to X and $X_1, \ldots, X_{q_P}$, so we can assume that $\mathcal{M}$ gets a representation only relative to them. Note however that at the start of the first execution of $\mathcal{A}$, $\mathcal{R}$ did not receive any of the X_i yet, so the first verification keys only depend on g and X. In the second, it has only seen X_1 and so on (w.l.o.g. we order the adversary executions such that the i-th adversary is the adversary that queried its challenge after $i-1$ other adversaries queried theirs). So for the i-th adversary, $c_{i,j,k} = 0$ for $k \geq i$.

Note that $\mathcal{R}$ can only program $\mathbf{H}'$. So if $\mathcal{R}$ executes an adversary $\mathcal{A}_i$ with the same randomness and inputs as $\mathcal{A}_j$, $\mathcal{A}_i$ makes the same queries to $\mathbf{H}'$ as $\mathcal{A}_j$ and if they are answered identically, then $\mathcal{A}_i$ uses the same challenge as $\mathcal{A}_j$ and does not query **Chal** or **DL**. However if the answers to $\mathbf{H}'$ differ in at least one place, then $\mathcal{A}_i$ behaves as depicted in Figure 5 (i.e. queries a fresh challenge, etc.). To detect which is the case, $\mathcal{A}$ has to make all queries to $\mathbf{H}'$ *before* querying a new challenge. Otherwise, $\mathcal{M}$ would have to use two queries to **DL** with the same challenge if $\mathcal{R}$ reprograms $\mathbf{H}'$ on the message $\mathcal{A}_i$ intends to forge on *after* all queries to $\bar{\mathbf{\Pi}}$ have already been made, making $\mathcal{M}$ unable to win the FBOMDL game. So w.l.o.g., we assume that all adversaries $\mathcal{A}_i$ for $i \in [q_P]$ are distinct.

Each $\mathcal{A}_i$ clearly $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_{\bar{\mathbf{\Pi}}}, q_{\bar{\mathbf{\Pi}}^{-1}}, q_{\mathbf{H}})$-breaks the n-UF-NMA security of GenDSA, unless the aborts in line 33 or 38 occur and $\mathcal{M}$ makes exactly one **Chal** query and at most one **DL** query per simulation of $\mathcal{A}$. The abort in line 33 occurs if $t_{k_i^*} = \psi(\bar{\mathbf{\Pi}}(\varphi(X_i))) = 0$ for any $i \in [q_P$ and we call this event $\textsc{Bad}_1$. Since $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ is a random function, it follows that the random variable $y^* = \bar{\mathbf{\Pi}}(\varphi(X_i))$ is uniformly random over $[0 : 2^L - 1]$. Therefore, we have $\Pr[\psi(\bar{\mathbf{\Pi}}(\varphi(X_i) = 0] = \Pr[\psi(y^*) = 0] \leq \max_{t\in\mathbb{Z}_p} \Pr_{y\xleftarrow{\$}[0:2^L-1]}[\psi(y) = t] \leq \epsilon_\psi$, so $\Pr[\textsc{Bad}_1] \leq \epsilon_\psi$ for any single execution of $\mathcal{A}_i$. A union bound over the q_P executions of $\mathcal{A}$ allows us to bound the probability of $\textsc{Bad}_1$ occurring in *any* execution of $\mathcal{A}_i$ by

$$\Pr[\textsc{Bad}_1] \leq q_P \epsilon_\psi.$$

Now consider the abort in line 38, i.e. $g^{h_i} \cdot \mathsf{vk}_{i,n_i^*}^{t_i} = 1$, which we call BAD_2. If BAD_2 occurs, it implies that

$$\mathbf{H}'(m_i^*) = t_i(a_{i,n_i^*} + b_{i,n_i^*}x + \sum_{j\in[i-1]} c_{i,n_i^*,j}x_j).$$

Note that $\mathcal{R}$ can program $\mathbf{H}'$, so it can trigger BAD_2 if it programs $\mathbf{H}'(m_i^*)$ correctly. However, $\mathcal{A}_i$ makes all of its queries to $\mathbf{H}'$ *before* it makes any queries to $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$. Therefore, in order to correctly program $\mathbf{H}'(m_i^*)$, $\mathcal{R}$ has to guess the correct indices n_i^*, j_i^* and a group element R' s.t. $f(R') = t_i$, since they are all information-theoretically hidden from $\mathcal{R}$. With the same argument as for BAD_1, the probability of finding a R' that maps to t_i can be bounded by ϵ_ψ, so together with guessing the indices, we get that

$$\Pr[\text{BAD}_2] \leq \frac{\epsilon_\psi}{nq_{\mathbf{H}}}$$

If $\mathcal{R}$ aborts at any point or does not return a valid signature, $\mathcal{M}$ aborts as well. So assume $\mathcal{R}$ eventually outputs a valid forgery $(m^*, \sigma^* = (s^*, t^*))$ for key vk^*. Since $\mathcal{R}$ can choose all verification keys $\mathsf{vk}_{i,j}$, it can set one to its own challenge key for each execution of $\mathcal{A}$ (It can only do this at one point since the challenge keys need to be distinct). So with probability at most $1/n$, an $\mathcal{A}_i$ produced a forgery for the public key that $\mathcal{M}$ gave to $\mathcal{R}$, in which case $\mathcal{R}$ can just forwards the signature it got from $\mathcal{A}_i$ and $\mathcal{M}$ aborts (line 10). We the event that this happens for any execution of $\mathcal{A}$ BAD_3. This abort occurs with probability $\frac{1}{n}\epsilon_{\mathcal{A}}$ for each execution of $\mathcal{A}$, so via a union bound, the probability of BAD_3 occurring in *any* execution of $\mathcal{A}$ is

$$\Pr[\text{BAD}_3] \leq \frac{q_P}{n}\epsilon_{\mathcal{A}}.$$

Otherwise, $\mathcal{R}$ produces a fresh forgery and together with the signatures from the simulated adversaries $\mathcal{A}_i$ and the verification equation of GenDSA, we get that

$$s_1 \cdot x_1 - b_{n_1^*}t_1 x = h_1 + a_{n_1^*}t_1 \tag{12}$$

$$\vdots$$

$$s_{q_P} \cdot x_{q_P} - t_{q_P} \sum_{j\in[q_P-1]} c_{q_P,n_{q_P}^*,j}x_j - b_{n_{q_P}^*}t_{q_P}x = h_{q_P} + a_{n_{q_P}^*}t_{q_P} \tag{13}$$

$$s^* \cdot r^* - t^* x = \mathbf{H}(m^*) \tag{14}$$

with $f(g^{r^*}) = t^*$.

Now $\mathcal{M}$ checks L for an entry $(g^{r^*}, z, (\alpha, \beta, (\gamma_i)_{i\in[q_P]}), z')$ with $\psi(z') = t^*$. If such an entry exists, we have $r^* = \alpha + \beta x + \sum_{i\in[q_P]} \gamma_i x_i$, yielding the equation

system

$$\begin{aligned} s_1 \cdot x_1 - b_{n_1^*} t_1 x &= h_1 + a_{n_1^*} t_1 \\ &\vdots \\ s_{q_P} \cdot x_{q_P} - t_{q_P} \sum_{j \in [q_P - 1]} c_{q_P, n_{q_P}^*, j} x_j - b_{n_{q_P}^*} t_{q_P} x &= h_{q_P} + a_{n_{q_P}^*} t_{q_P} \\ s^* \sum_{j \in [q_P]} \gamma_j x_j + (\beta s^* - t^*) x &= \mathbf{H}(m^*) - \alpha s^* \end{aligned}$$

If no such r^* exists, then $\mathcal{R}$ never queried the bRO on a preimage of t^* or received a preimage of t^* from the bRO. We call this case BAD_4 and $\mathcal{M}$ aborts (line 12) as it can not solve its FBOMDL challenge in this case. By the same argument as for BAD_1, we get

$$\Pr[\text{BAD}_4] \leq \epsilon_\psi.$$

Otherwise, the signatures form a system of $q_P + 1$ equations with $q_P + 1$ variables, where each variable represents a solution to one of the discrete logarithm challenges. So unless the determinant of the system is 0, $\mathcal{M}$ can extract a solution for its FBOMDL instance. So all that is left is to analyze the probability of this bad case.

In matrix notation, we can write this condition as

$$\mathbf{det} \begin{bmatrix} s_1 & 0 & 0 & \cdots & b_{n_1^*} t_1 \\ t_2 c_{2, n_2^*, 1} & s_2 & 0 & \ddots & \vdots \\ \vdots & & \ddots & \vdots & \vdots \\ t_{q_P} c_{q_P, n_{q_P}^*, 1} & \cdots & t_{q_P} c_{q_P, n_{q_P}^*, q_P - 1} & s_{q_P} & b_{n_{q_P}^*} t_{q_P} \\ s^* \gamma_1 & & \cdots & s^* \gamma_{q_P} & \beta s^* - t^* \end{bmatrix} \neq 0$$

Note that every entry of the matrix except for the main diagonal includes factors which are under the adversaries control, namely the $c_{i, n_j^*, k}$ and $b_{n_i^*}$. So for simplicity, we assume that the adversary can have them take an arbitrary value. The only problematic part is the main diagonal, which depends on all the s_i, which are given by the adversary $\mathcal{A}$ and $\beta s^* - t^*$ given by the reduction. Next, note that all the s_i are non-zero, as otherwise the signatures would be invalid. Therefore, for the determinant to be zero, $\beta s^* - t^* = 0$ is required.[6] We call this event BAD_5.

So assume that $\beta s^* = t^*$. Substituting in Equation (14) yields the equation

$$t^* \Big(\sum_{j \in [q_P]} \gamma_j x_j + \alpha \Big) = \beta \cdot \mathbf{H}(m^*) \tag{15}$$

[6] Since the adversary can control all other terms, it could also make the diagonal match the sum of all other values, but that is just as hard as making the main diagonal zero.

Since $t^* \neq 0$ and $s^* \neq 0$ (otherwise $\mathcal{R}$ didn't produce a valid forgery), $\beta \neq 0$. Then we get $\mathbf{H}(m^*) = \frac{t^*(\sum_{j\in[q_P]} \gamma_j x_j + \alpha)}{\beta}$ where $\mathbf{H}$ is the random oracle that $\mathcal{R}$ can *not* program.

For every fixed $m^*, \alpha, \beta, (\gamma_i)_{i\in[q_P]}$, there is a unique t^* that satisfies Equation (15) and since $(\bar{\mathbf{\Pi}}, \bar{\mathbf{\Pi}}^{-1})$ is a random function, we have $\Pr[\psi(\bar{\mathbf{\Pi}}(\varphi(R) = t^*] \leq \epsilon_\psi$ for every $R \in \mathbb{G}$. $\mathcal{R}$ can choose from $q_{\mathbf{H}}$ tuples $(m^*, \mathbf{H}(m^*)$ and from $2(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}})$ tuples $(\alpha, \beta, (\gamma_i)_{i\in[q_P]})$ (φ is semi-injective, so there are 2 tuples for every query). So a union bound over the number of tuples yields that the probability of $\mathcal{R}$ finding a tuple $(\mathbf{H}(m^*), \alpha, \beta, (\gamma_i)_{i\in[q_P]}, t^*)$ that satisfies Equation (15) is at most $2\epsilon_\psi q_{\mathbf{H}}(q_{\bar{\mathbf{\Pi}}} + q_{\bar{\mathbf{\Pi}}^{-1}})$ and

$$\Pr[\textsc{Bad}_5] \leq 2\epsilon_\psi q_{\mathbf{H}}(q_{\bar{\mathbf{\Pi}}} + q_{\bar{\mathbf{\Pi}}^{-1}}).$$

Combining all probabilities, we get

$$\begin{aligned}\epsilon_{\mathcal{M}} &\geq \epsilon_{\mathcal{R}} - \sum_{i=1}^{5} \Pr[\textsc{Bad}_i] - q^2/2^L \\ &\geq \epsilon_{\mathcal{R}} - \frac{2q_P}{n}\epsilon_{\mathcal{A}} - \epsilon_\psi \left(2(q'_{\bar{\mathbf{\Pi}}} + q'_{\bar{\mathbf{\Pi}}^{-1}})q'_{\mathbf{H}} + q_P + 2\right) - q^2/2^L\end{aligned}$$

which concludes the proof. □

References

1. Bauer, B., Fuchsbauer, G., Plouviez, A.: The one-more discrete logarithm assumption in the generic group model. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 587–617. Springer (2021)
2. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. Journal of Cryptology 16(3), 185–215 (Jun 2003)
3. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93: 1st Conference on Computer and Communications Security. pp. 62–73. ACM Press, Fairfax, Virginia, USA (Nov 3–5, 1993)
4. Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., Moeller, B.: Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls). RFC 4492, RFC Editor (2016)
5. Brickell, E.F., Pointcheval, D., Vaudenay, S., Yung, M.: Design validations for discrete logarithm based signature schemes. In: Imai, H., Zheng, Y. (eds.) PKC 2000: 3rd International Workshop on Theory and Practice in Public Key Cryptography. Lecture Notes in Computer Science, vol. 1751, pp. 276–292. Springer, Heidelberg, Germany, Melbourne, Victoria, Australia (Jan 18–20, 2000)
6. Brown, D.: On the Provable Security of ECDSA, p. 21–40. London Mathematical Society Lecture Note Series, Cambridge University Press (2005)
7. Brown, D.R.L.: The exact security of ECDSA. Contributions to IEEE P1363a (Jan 2001), `http://grouper.ieee.org/groups/1363/`

8. Brown, D.R.L.: Generic groups, collision resistance, and ECDSA. Cryptology ePrint Archive, Report 2002/026 (2002), https://eprint.iacr.org/2002/026
9. Brown, D.R.L.: Generic groups, collision resistance, and ECDSA. Contributions to IEEE P1363a (Feb 2002), updated version for "The Exact Security of ECDSA." Available from http://grouper.ieee.org/groups/1363/
10. Brzuska, C., Farshim, P., Mittelbach, A.: Random-oracle uninstantiability from indistinguishability obfuscation. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015: 12th Theory of Cryptography Conference, Part II. Lecture Notes in Computer Science, vol. 9015, pp. 428–455. Springer, Heidelberg, Germany, Warsaw, Poland (Mar 23–25, 2015)
11. Camenisch, J., Piveteau, J.M., Stadler, M.: Blind signatures based on the discrete logarithm problem (rump session). In: Santis, A.D. (ed.) Advances in Cryptology – EUROCRYPT'94. Lecture Notes in Computer Science, vol. 950, pp. 428–432. Springer, Heidelberg, Germany, Perugia, Italy (May 9–12, 1995)
12. Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., Peled, U.: UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020: 27th Conference on Computer and Communications Security. pp. 1769–1787. ACM Press, Virtual Event, USA (Nov 9–13, 2020)
13. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: 30th Annual ACM Symposium on Theory of Computing. pp. 209–218. ACM Press, Dallas, TX, USA (May 23–26, 1998)
14. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ECDSA from hash proof systems and efficient instantiations. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 191–221. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019)
15. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-efficient threshold EC-DSA. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II. Lecture Notes in Computer Science, vol. 12111, pp. 266–296. Springer, Heidelberg, Germany, Edinburgh, UK (May 4–7, 2020)
16. Dalskov, A.P.K., Orlandi, C., Keller, M., Shrishak, K., Shulman, H.: Securing DNSSEC keys via threshold ECDSA from generic MPC. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020: 25th European Symposium on Research in Computer Security, Part II. Lecture Notes in Computer Science, vol. 12309, pp. 654–673. Springer, Heidelberg, Germany, Guildford, UK (Sep 14–18, 2020)
17. Damgård, I., Jakobsen, T.P., Nielsen, J.B., Pagter, J.I., Østergaard, M.B.: Fast threshold ECDSA with honest majority. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20: 12th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 12238, pp. 382–400. Springer, Heidelberg, Germany, Amalfi, Italy (Sep 14–16, 2020)
18. Doerner, J., Kondi, Y., Lee, E., shelat, a.: Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE Symposium on Security and Privacy. pp. 980–997. IEEE Computer Society Press, San Francisco, CA, USA (May 21–23, 2018)
19. Doerner, J., Kondi, Y., Lee, E., shelat, a.: Threshold ECDSA from ECDSA assumptions: The multiparty case. In: 2019 IEEE Symposium on Security and Privacy. pp. 1051–1066. IEEE Computer Society Press, San Francisco, CA, USA (May 19–23, 2019)

20. Fersch, M., Kiltz, E., Poettering, B.: On the provable security of (EC)DSA signatures. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016: 23rd Conference on Computer and Communications Security. pp. 1651–1662. ACM Press, Vienna, Austria (Oct 24–28, 2016)
21. Fersch, M., Kiltz, E., Poettering, B.: On the one-per-message unforgeability of (EC)DSA and its variants. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017: 15th Theory of Cryptography Conference, Part II. Lecture Notes in Computer Science, vol. 10678, pp. 519–534. Springer, Heidelberg, Germany, Baltimore, MD, USA (Nov 12–15, 2017)
22. Fischlin, M., Lehmann, A., Ristenpart, T., Shrimpton, T., Stam, M., Tessaro, S.: Random oracles with(out) programmability. In: Abe, M. (ed.) Advances in Cryptology – ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 303–320. Springer, Heidelberg, Germany, Singapore (Dec 5–9, 2010)
23. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 33–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)
24. Galbraith, S., Malone-Lee, J., Smart, N.: Public key signatures in the multi-user setting. Information Processing Letters 83(5), 263–266 (2002), https://www.sciencedirect.com/science/article/pii/S0020019001003386
25. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018: 25th Conference on Computer and Communications Security. pp. 1179–1194. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018)
26. Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) ACNS 16: 14th International Conference on Applied Cryptography and Network Security. Lecture Notes in Computer Science, vol. 9696, pp. 156–174. Springer, Heidelberg, Germany, Guildford, UK (Jun 19–22, 2016)
27. Groth, J., Shoup, V.: On the security of ECDSA with additive key derivation and presignatures. In: EUROCRYPT 2022. Lecture Notes in Computer Science, Springer (2022)
28. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International journal of information security 1(1), 36–63 (2001)
29. Kiltz, E., Masny, D., Pan, J.: Optimal security proofs for signatures from identification schemes. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 33–61. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016)
30. Kondi, Y., Magri, B., Orlandi, C., Shlomovits, O.: Refresh when you wake up: Proactive threshold wallets with offline devices. In: 2021 IEEE Symposium on Security and Privacy. pp. 608–625. IEEE Computer Society Press, San Francisco, CA, USA (May 24–27, 2021)
31. Lindell, Y.: Fast secure two-party ECDSA signing. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 613–644. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
32. Lindell, Y., Nof, A.: Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018: 25th Conference on Computer and Communications Security. pp. 1837–1854. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018)

33. Malone-Lee, J., Smart, N.P.: Modifications of ECDSA. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 2595, pp. 1–12. Springer, Heidelberg, Germany, St. John's, Newfoundland, Canada (Aug 15–16, 2003)
34. National Institute of Standards and Technology: Digital signature standard (DSS) - fips 186-4. Tech. rep., U.S. Department of Commerce (2013)
35. Nguyen, P.Q., Shparlinski, I.: The insecurity of the digital signature algorithm with partially known nonces. Journal of Cryptology 15(3), 151–176 (Jun 2002)
36. Paillier, P., Vergnaud, D.: Discrete-log-based signatures may not be equivalent to discrete log. In: Roy, B.K. (ed.) Advances in Cryptology – ASIACRYPT 2005. Lecture Notes in Computer Science, vol. 3788, pp. 1–20. Springer, Heidelberg, Germany, Chennai, India (Dec 4–8, 2005)
37. Qin, X., Cai, C., Yuen, T.H.: One-more unforgeability of blind ecdsa. Cryptology ePrint Archive, Report 2021/1449 (2021), `https://ia.cr/2021/1449`
38. Reingold, O., Trevisan, L., Vadhan, S.P.: Notions of reducibility between cryptographic primitives. In: Naor, M. (ed.) TCC 2004: 1st Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 2951, pp. 1–20. Springer, Heidelberg, Germany, Cambridge, MA, USA (Feb 19–21, 2004)
39. Stern, J., Pointcheval, D., Malone-Lee, J., Smart, N.P.: Flaws in applying proof methodologies to signature schemes. In: Yung, M. (ed.) Advances in Cryptology – CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 93–110. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2002)
40. Vaudenay, S.: Hidden collisions on DSS. In: Koblitz, N. (ed.) Advances in Cryptology – CRYPTO'96. Lecture Notes in Computer Science, vol. 1109, pp. 83–88. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 1996)
41. Vaudenay, S.: The security of DSA and ECDSA. In: Desmedt, Y. (ed.) PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography. Lecture Notes in Computer Science, vol. 2567, pp. 309–323. Springer, Heidelberg, Germany, Miami, FL, USA (Jan 6–8, 2003)
42. Zhandry, M.: To label, or not to label (in generic groups). In: Advances in Cryptology – CRYPTO 2022, Part IV. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (Aug 2022)
43. Zhang, C., Zhou, H.S., Katz, J.: An analysis of the algebraic group model. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022. pp. 310–322. Springer Nature Switzerland, Cham (2022)

Supplementary Material

A Hardness of FBOMDL in Generic Groups

Recently, [1] proved a lower bound in the GGM on the q-OMDL assumption which is independent of q. We now show how to adapt this proof to show the same bound on the q-FBOMDL assumption.

A.1 Shoup's Generic Group Model

In the Generic Group Model (GGM), the (cyclic) cryptographic group is replaced by a generic group, i.e. instead of receiving group elements and doing all group arithmetic themselves, all algorithms instead receive random *labels* (e.g. bit-strings from a large enough domain) for group elements. Since these labels do not contain any semantic meaning, they are also provided with a group operation oracle **GGMOp**, which takes two labels as input and returns the label of the group element that is the result of the group operation applied to the group elements corresponding to the input labels.

A standard technique, which we will also use, is to replace group elements with polynomials and only choosing the acutal group element after the execution is done. Therefore, we extend the labeling to arbitrary polynomials in the straightforward way. Replacing group elements with polynomials is indistinguishable for an adversary, unless it finds two polynomials, which are different but evaluate to the same group element for the randomly chosen point. Generally, the probability of this bad case occurring is bounded using the Schwartz-Zippel lemma. However, as [1] observed, this approach only works if all variables are replaced *after* the execution. However, since we want to prove the FBOMDL assumption in the GGM, we have to handle **DL** queries *during* the execution, which makes applying the lemma hard. To circumvent this, we use the lemma proven in [1], which we recall in Lemma 1.

Lemma 1 ([1]). *Let $p \in \mathbb{N}$, $D_1, \ldots, D_m, Q_1, \ldots, Q_{q+1} \in \mathbb{Z}_p[X_1, \ldots, X_n]$ of degree 1 and*

$$\begin{aligned} \forall i \in [q] : \mathcal{Q}_i &\leftarrow \{\vec{x} \in \mathbb{Z}_p^n : Q_i(\vec{x}) = 0\} \\ \forall i \in [m] : \mathcal{D}_i &\leftarrow \{\vec{x} \in \mathbb{Z}_p^n : D_i(\vec{x}) = 0\} \\ \mathcal{C} &\leftarrow \left(\bigcap_{i\in[q]} \mathcal{Q}_i\right) \setminus \left(\bigcup_{i\in[m]} \mathcal{D}_i\right). \end{aligned}$$

Assume $\mathcal{Q}_{q+1} \cap \mathcal{C} \neq \emptyset$ and Q_{q+1} is linearly independent of $(Q_i)_{i\in[q]}$. Then

$$\frac{p-m}{p^2} \leq \Pr[Q_{q+1}(\vec{x}) = 0 \mid \vec{x} \xleftarrow{\$} \mathcal{C}] \leq \frac{1}{p-m}$$

Intuitively, we will use the lemma to bound the probability that an adversary finds a **DL** query (Q_{q+1}) which will cause an inconsistency (i.e., it is linearly independent of the previous queries (Q_i), but $Q_{q+1}(\vec{x}) = 0$) for a choice of $\vec{x}$ that is consistent with its previous view (i.e., the D_i and Q_i).

A.2 FBOMDL holds in the GGM

We now state the formal theorem and proceed with the proof that the FBOMDL assumption holds unconditionally in the generic group model.

Theorem 6. *Let $\mathcal{A}$ be an adversary (t, ϵ)-breaks the q-*FBOMDL *problem. Then*

$$\epsilon \leq \frac{m^2}{p - m^2} + \frac{1}{p},$$

where $m \approx t$ is the number of queries to **GGMOp** *made by $\mathcal{A}$.*

Proof. The proof is almost identical to the proof of Theorem 1 in [1]. The main change is made in the simulation of the **DL** oracle, since it now gets two labels as input. However, all abort conditions are identical, so the overall bound is the same as in [1].

For the proof consider the games in Figure 6. We keep the notation from [1]. The **Chal**, **DL** and **GGMOp** oracles have the same function as in the regular FBOMDL game. Encoding a group element is done via the function **Enc**. This function takes no explicit input, but instead produces an encoding for the last group element generated by another function. This group element is indexed by the variable j, which is always set by the other oracles before calling **Enc**.

All other oracles use **Enc**when they generate a new group element, specifically **Chal** and **GGMOp** end with a call to **Enc**. Note that all encodings ξ_i are associated with their respective discrete logarithm a_i (and the polynomial P_i in the later games). Since the challenge oracle **Chal** can be queried at any time, all challenges x_i have an associated a_{j_i} (and later P_{j_i}).

The idea of the proof is to gradually remove the need to know the challenge $\vec{x}$ from the game. This is done by replacing them with variables. This will introduce a small error probability because two distinct polynomials might coincide on the challenge $\vec{x}$, but we can use Lemma 1 to bound it. After some rearranging to make all checks using only the polynomials, we can eventually choose all challenges *after* the execution of the adversary in a way that is consistent with the random answers we gave before.

We will now describe the changes between the different games and analyse the probability of an adversary distinguishing two subsequent games.

G_0: This is the regular FBOMDL game in the generic group model. We assign each group element seen by the adversary an increasing number j in addition to its label as in [1]. The group generator 1 gets index 0 and label ξ_0. The n-th challenge gets index j_n and is called x_n.

We have

$$\Pr[G_0^{\mathcal{A}} = 1] = \epsilon.$$

Games G_0-G_4

$j := 0; q := 0; n := 0$
$a_0 \leftarrow 1$ // G_0-G_1
$P_0 \leftarrow 1$; $L \leftarrow \emptyset$ // G_1-G_4
$\xi_0 = \mathbf{Enc}()$
$\xi_1 \xleftarrow{\$} \mathbf{Chal}$
$\vec{y} \xleftarrow{\$} \mathcal{A}^{\mathbf{DL},\mathbf{Chal},\mathbf{GGMOp}}(\xi_0, \xi_1)$
if $q > n$ **then**
  return 0
for $i \in [n]$ **do** // G_3-G_4
  $x_i \leftarrow \mathbf{DL}(\xi_{j_i}, \xi_0)$
return $\vec{x} = \vec{y}$

Oracle Chal

$n \leftarrow n + 1$
$j \leftarrow j + 1$
$j_n \leftarrow j$
$x_n \xleftarrow{\$} \mathbb{Z}_p$ // G_0-G_3
$a_j \leftarrow x_n$ // G_0-G_1
$P_j \leftarrow X_n$ // G_1-G_4
return $\mathbf{Enc}()$

Oracle Enc

if $\exists k \in [0 : j-1] : a_k = a_j$ **then** // G_0-G_1
  $\xi_j \leftarrow \xi_k$
if $\exists k \in [0 : j-1] : P_j(\vec{x}) = P_k(\vec{x})$ $\wedge\, P_j - P_k \notin \mathrm{span}(L)$ **then** // G_1-G_2
  `abort`
if $\exists k \in [0 : j-1] : P_j - P_k \in \mathrm{span}(L)$ **then** // G_2-G_4
  $\xi_j \leftarrow \xi_k$
else
  $\xi_j \xleftarrow{\$} \{0,1\}^* \setminus (\xi_k)_{k \in [0:j-1]}$
return ξ_j

Oracle DL(ξ, ξ')

if $\xi \notin \{\xi_i\}_{i \in [0:j]} \vee \xi' \notin \{\xi_i\}_{i \in [0:j]}$ **then**
  return $\perp$
$i \leftarrow \min\{k : \xi_k = \xi\}$
$i' \leftarrow \min\{k : \xi_k = \xi'\}$
$q \leftarrow q + 1$
$v \leftarrow a_i$ // G_0-G_1
$v' \leftarrow a_{i'}$
$v \leftarrow P_i(\vec{x})$; $v' \leftarrow P_{i'}(\vec{x})$ // G_2-G_4
$v, v' \xleftarrow{\$} \mathbb{Z}_p^2$ // G_4
if $P_i \in \mathrm{span}(1, L)$ **then**
  Let $P_i = \alpha_0 + \sum_{Q \in L} \alpha_Q Q$
  $v \leftarrow \alpha_0$
if $P_{i'} \in \mathrm{span}(1, L)$ **then**
  Let $P_{i'} = \alpha_0 + \sum_{Q \in L} \alpha_Q Q$
  $v' \leftarrow \alpha_0$
if $v' = 0$ **then**
  $Q_q \leftarrow P_{i'} - v$
else
  $Q_q \leftarrow P_i - \frac{v}{v'} P_{i'}$
$L \leftarrow L \cup \{Q_q\}$ // G_1-G_4
if $\exists i_1, i_2 \in [0 : j]^2 : P_{i_1} - P_{i_2} \in \mathrm{span}(L)$ $\wedge\, \xi_{i_1} \neq \xi_{i_2}$ **then** // G_3-G_4
  `abort`
if $v' = 0$ **then**
  return $\perp$
return $\frac{v}{v'}$

Oracle GGMOp(ξ, ξ')

if $\xi \notin \{\xi_i\}_{i \in [0:j]} \vee \xi' \notin \{\xi_i\}_{i \in [0:j]}$ **then**
  return $\perp$
$i \leftarrow \min\{k : \xi_k = \xi\}$
$i' \leftarrow \min\{k : \xi_k = \xi'\}$
$j \leftarrow j + 1$
$a_j \leftarrow a_i + a_{i'}$ // G_0
$P_j \leftarrow P_i + P_{i'}$ // G_1-G_4
return $\mathbf{Enc}()$

Fig. 6. Games G_0-G_4 for the proof of Theorem 6. Each group element has a unique index i, its discrete logarithm is denoted by a_i and is associated to a polynomial P_i, which is constant for all regular group elements and a monomial for all challenge group elements.

G_1: In G_1, we introduce polynomials. Specifically, each challenge x_i is associated with a monomial X_i. In parallel to the scalars a_i, we now also track polynomials P_i, to which we apply the same operations as the a_i. Here, the polynomial P_0 is the constant polynomial 1 corresponding to $a_0 = 1$. Specifically, $P_i(\vec{x}) = a_i$ for all $i \leq j$. We still sample the x_i whenever a new challenge is requested and use the real a_i to answer discrete logarithm queries.

Additionally, the (initially empty) set L is introduced. Intuitively, L will contain all relations $\mathcal{A}$ knows on the challenge vector $\vec{x}$. Specifically, it will contain all polynomials that $\mathcal{A}$ queried to its **DL** oracle rearranged to equal 0. So whenever $\mathcal{A}$ makes a **DL** query on two labels ξ, ξ', the corresponding values $a_i, a_{i'}$ are looked up. If $a_{i'} = 0$, then the discrete logarithm query fails, so the only

information that $\mathcal{A}$ gets is that $a_{i'} = 0$. So we return $\perp$ and add $P_{i'} - P_{i'}(\vec{x}) = P_{i'}$ to L. Otherwise, we return v/v' and add $P_i - \frac{v}{v'}P_{i'}$ to L. In both cases, the dimension of L can increase by at most 1 (which will be relevant in the last game).

This is the main divergence from the original proof of [1] for the regular OMDL assumption, because we need to handle queries relative to an arbitrary basis. However note that although the adversary provides two labels for polynomials, it still only gets a solution for one affine equation, so the proof can continue as in [1].

Lastly, we introduce the first abort condition, namely the game aborts if the encoding function is called on a polynomial which, evaluated on $\vec{x}$ is equal to another polynomial evaluated on $\vec{x}$, but the difference of the two polynomials is not in the span of L. This case is indeed problematic, as it results in two polynomials that agree when evaluated on $\vec{x}$ having different labels, which the adversary would be able to detect. We call this event BAD.

Since this is the only non-conceptual change, we see that

$$\Pr[G_0^{\mathcal{A}} = 1] \leq \Pr[G_1^{\mathcal{A}} = 1] + m \cdot \Pr[\text{BAD}]$$

With verbatim the same argument as in [1] (p. 17f), we can use Lemma 1 and see that

$$\Pr[\text{BAD}] \leq \frac{m}{p - m^2},$$

so overall we get

$$\Pr[G_0^{\mathcal{A}} = 1] \leq \Pr[G_1^{\mathcal{A}} = 1] + \frac{m^2}{p - m^2}.$$

G_2: Next, we change how the **DL** oracle is implemented. First, instead of looking for $a_i, a_{i'}$, we instead search for the corresponding polynomials $P_i, P_{i'}$ and evaluate them on $\vec{x}$ to calculate the v, v'. This does not change the game, because for all $i \in [0 : j-1] : a_i = P_i(\vec{x})$. However, eventually in G_4, we want to remove the need to know $\vec{x}$ entirely. To prepare for this step, we additionally check if one of the polynomials can be represented as a linear combination of the $Q \in L$. Since all Q evaluate to 0 on $\vec{x}$, the constant part of this representation is exactly v (resp. v').

All these changes are only conceptual, so we get

$$\Pr[G_1^{\mathcal{A}} = 1] = \Pr[G_2^{\mathcal{A}} = 1]$$

G_3: In this game, we move the abort condition from the encoding function to the **DL** oracle. In order to abort in the same cases as G_2, we also now query the **DL** oracle on every challenge *after* the adversary returned its solution.

In G_2, we check for every polynomial that we encode whether it coincides with another polynomial on the challenge $\vec{x}$ while this relation is not mirrored in L and abort in this case.

On the other hand, in G_3, we check in **DL** if there are pairs of polynomials such that their difference is in L but they have different labels. Most importantly,

this condition can be checked without knowing $\vec{x}$. Note that we call **DL** at the end of the protocol for all challenges X_i, so this check is always done at the end of the execution.

First, we argue that if G_3 aborts, then G_2 aborts as well. So assume that there are P_{i_1}, P_{i_2} s.t. $P = P_{i_1} - P_{i_2} \in L$ but $\xi_{i_1} \neq \xi_{i_2}$. Since $P \in L$, we have $P_{i_1}(\vec{x}) = P_{i_2}(\vec{x})$. Let $i_1 < i_2$. Then when the encoding ξ_2 was computed, the algorithm chose a fresh ξ_2, so $P \notin L$. But since $P_{i_1}(\vec{x}) = P_{i_2}(\vec{x})$, G_2 had to abort at that point.

On the other hand, assume that G_2 aborts. Then there are i_1, i_2 s.t. $P = P_{i_1} - P_{i_2} \notin L$ but $P_{i_1}(\vec{x}) = P_{i_2}(\vec{x})$. So in G_3, we do not abort at this point but the two polynomials get different labels $\xi_{i_1} \neq \xi_{i_2}$. At the end of the game, the challenger calls **DL** on all X_i relative to 1, so L contains the polynomials $X_i - x_i$ for $i \in [n]$. Let $P = P(\vec{0}) + \sum_{i\in[n]} \alpha_i X_i$. Rewriting with the polynomials in L, we get $P = P(\vec{0}) + \sum_{i\in[n]} \alpha_i(X_i - x_i) + x_i$. Since G_2 would abort, we have $P(\vec{x}) = P_{i_1}(\vec{x}) + P_{i_2}(\vec{x}) = 0$ and $P(\vec{x}) = P(0) + \sum_{i\in[n]} \alpha_i x_i$. Therefore, $P = \sum_{i\in[n]} \alpha_i(X_i - x_i)$, so at the end of the game $P \in L$ and G_3 aborts because $\xi_{i_1} \neq \xi_{i_2}$.

Therefore, we have

$$\Pr[G_2^{\mathcal{A}} = 1] = \Pr[G_3^{\mathcal{A}} = 1]$$

G_4: In this last game, we do not sample the x_i in calls to **Chal** and only return a label for the new monomial X_n. Since the x_i are now undefined, we can finally replace v and v' with random exponents instead of evaluating the corresponding $P_i, P_{i'}$, unless the answers to previous queries to **DL** already define the answer, i.e. one of the P_i queries is in the span of L. Again, identically to [1], we can use Lemma 1 to see that the v and v' are distributed uniformly in G_3, so choosing them at random does not change the distribution in G_4 and we have

$$\Pr[G_3^{\mathcal{A}} = 1] = \Pr[G_4^{\mathcal{A}} = 1]$$

Finally, we have to bound the winning probability in G_4. To this end, note that the dimension of L is at most q, but there are $n > q$ challenges X_i. Since $\vec{x}$ is chosen adaptively to be consistent with L, the challenger chooses a vector of dimension n with q constraints, so at least one component is not defined by the constraints in L and is therefore chosen *after* $\mathcal{A}$ outputs its solution $\vec{y}$ in the final queries to the **DL** oracle done by the challenger. Since this last x is chosen uniformly at random, we have

$$\Pr[G_4^{\mathcal{A}} = 1] \leq \frac{1}{p}.$$

□

B Non-Rewinding Reductions

B.1 Proof of Theorem 2

Proof. The idea of the proof is similar to Theorem 1, except that we now only need one additional FBOMDL challenge as the meta reduction now only simulates one execution of $\mathcal{A}$. However, we have to take extra care how we embed the second challenge X' as the reduction can now program the bRO $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$.

Let $\mathcal{A}$ be an adversary that $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_{\mathbf{\Pi}}, q_{\mathbf{\Pi}^{-1}})$-breaks the SDLog assumption, and let $\mathcal{R}$ be the security reduction that $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}})$-breaks the DLog assumption. $\mathcal{A} = \mathcal{A}^{(\mathbf{\Pi}(\cdot), \mathbf{\Pi}^{-1}(\cdot))}$ has access to the programmable bRO $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ of the SDLog game. $\mathcal{R}$ executes adversary $\mathcal{A}$ *once* and simulates the bRO $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ for $\mathcal{A}$. We will construct a meta-reduction $\mathcal{M}$ trying to solve the 1-FBOMDL game by running the reduction $\mathcal{R}$, simulating the DLog experiment and the one execution of adversary $\mathcal{A}$.

$\mathcal{M}^{\mathbf{Chal}, \mathbf{DL}(\cdot,\cdot)}(X)$ receives X as its 1-FBOMDL input, has access to FBOMDL's challenge oracle **Chal** and discrete logarithm oracle **DL** and executes the reduction $\mathcal{R}(X)$ with X as its DLog challenge. Reduction $\mathcal{R}$ in turn executes $\mathcal{A}^{\mathbf{\Pi}(\cdot), \mathbf{\Pi}^{-1}(\cdot)}(\hat{X})$ with SDLog challenge $\hat{X} = g^a X^b$. It simulates the SDLog experiment and the bRO $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$. Since $\mathcal{R}$ is algebraic, it provides $\mathcal{A}$ (and therefore $\mathcal{M}$) with the representation a, b of $\hat{X}$. Note that the representation only depends on the generator and X as no other FBOMDL challenges have been queried so far and $\mathcal{R}$ does not have access to an external oracle that could provide new group elements. We assume that $b \neq 0$. This is without loss of generality, because if $b = 0$, then the representation of $\hat{X}$ reveals the corresponding discrete logarithm a and $\mathcal{A}$ can honestly compute a SDLog solution, which $\mathcal{R}$ could also compute without using $\mathcal{A}$.

$\mathcal{A}$ is simulated by $\mathcal{M}$ as shown in Figure 7.

Meta Reduction $\mathcal{M}^{\mathbf{Chal}, \mathbf{DL}(\cdot,\cdot)}(X)$

1: Run $\mathcal{R}(X)$
2: Receive $\hat{X} = g^a X^b$ from $\mathcal{R}$ // w.l.o.g. $b \neq 0$
3: Simulate $\mathcal{A}^{\mathbf{\Pi}(\cdot), \mathbf{\Pi}^{-1}(\cdot)}(\hat{X})$ for $\mathcal{R}$
4: **if** $\mathcal{A}$ aborts in line 18 **then**
5: $\quad x \leftarrow -\frac{1+at^*}{bt^*}$
6: $\quad x' \leftarrow \mathbf{DL}(X', g)$
7: $\quad$ **return** (x, x')
8: Receive solution x from $\mathcal{R}$
9: $x' \leftarrow \frac{1+(a+bx)t^*}{s^*} - r'_{j^*}$
10: **return** (x, x')

Adversary $\mathcal{A}^{\mathbf{\Pi}(\cdot), \mathbf{\Pi}^{-1}(\cdot)}(\hat{X})$ // $\hat{X} = g^a X^b$, $b \neq 0$

11: $X' \leftarrow \mathbf{Chal}$
12: **for** $j \in [q_{\mathbf{\Pi}}]$ **do**
13: $\quad r'_j \xleftarrow{\$} \mathbb{Z}_p$; $R_j \leftarrow X' \cdot g^{r'_j}$ // w.l.o.g. $R_j \xleftarrow{\$} \mathbb{G}^*$
14: $\quad t_j \leftarrow f(R_j)$ // via oracle query to $\mathbf{\Pi}$
15: Make $q_{\mathbf{\Pi}^{-1}}$ dummy queries to $\mathbf{\Pi}^{-1}$
16: Let j^* s.t. $t_{j^*} \neq 0$
17: $t^* \leftarrow t_{j^*}$; $R^* \leftarrow R_{j^*}$
18: **if** $g \cdot \hat{X}^{t^*} = 1$ **then** // $\Leftrightarrow s^* = 0$
19: $\quad$ **abort**
20: $s^* \leftarrow \mathbf{DL}(g \cdot \hat{X}^{t^*}, R^*)$ // from FBOMDL game
21: **return** $\begin{cases} (s^*, t^*) & \text{with prob. } \epsilon_{\mathcal{A}} \\ \perp & \text{else} \end{cases}$

Fig. 7. Meta reduction $\mathcal{M}$ and simulated adversary $\mathcal{A}$ against the SDLog assumption. **DL** and **Chal** are $\mathcal{M}$'s oracles from the FBOMDL assumption.

$\mathcal{A}$ begins by querying a fresh **DL** challenge X' and embeds it in all of its queries $R_j = X' \cdot g^{r'_j}$ to f (i.e. it queries $\varphi(R_i)$ to $\mathbf{\Pi}$) with an additive blinding term $g^{r'_j}$ for $j \in [q_{\mathbf{\Pi}}]$. Here we assume that R_j is uniformly distributed in $\mathbb{G}^*$, as

in GenDSA. (If $R_j = 1_{\mathbb{G}}$ we simply resample r'_j.) Denote the answers as z_j and $t_j \leftarrow \psi(z_j)$.

In Line 16 we assume that there exists an index j^* with $t_{j^*} \neq 0$. This is without loss of generality, as otherwise all valid signatures are information theoretically hidden from *any* adversary and the reduction can forge a signature with at least the same probability as the adversary without using it. $\mathcal{A}$ picks such a $t^* = t_{j^*}$ and checks whether $g \cdot \hat{X}^{t^*} = 1$. If this is the case, then $\mathcal{A}$ is not able to extract a SDLog solution using t^*. However, this implies

$$0 = 1 + (a + bx)t^* \Leftrightarrow x = -\frac{1 + at^*}{bt^*},$$

where the latter is well-defined since $b \neq 0$ and $t^* \neq 0$ by assumption. $\mathcal{M}$ can then use its **DL** query to get x' and wins the game.

So assume that $g \cdot \hat{x}^{t^*} \neq 1$. Then $\mathcal{A}$ uses the **DL** query to compute $s^* = \mathbf{DL}(g \cdot \hat{X}^{t^*}, R^*) = \frac{1+\hat{x}t^*}{r^*}$, where $R^* = g^{r^*}$ and $\hat{X} = g^{\hat{x}}$. Note that $(s^*, t^*) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ is a valid solution to the SDLog game. This simulation is perfect, so $\mathcal{R}$ eventually outputs the discrete logarithm x of its challenge X with probability $\epsilon_{\mathcal{R}}$. Finally, using $r^* = x' + r'_{j^*}$ and $\hat{x} = a + bx$, $\mathcal{M}$ can can recover x' as from x as

$$x' = \frac{1 + (a + bx)t^*}{s^*} - r'_{j^*}.$$

To conclude, $\mathcal{M}$ wins the 1-FBOMDL game exactly if $\mathcal{R}$ is successful. □

B.2 Proof of Theorem 5

Proof. Let $\mathcal{A}$ be an adversary that $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}}, q_{\mathbf{\Pi}'}, q_{\mathbf{\Pi}'^{-1}}, q_{\mathbf{H}'})$-breaks the n-UF-NMA security of GenDSA, and let $\mathcal{R}$ be the algebraic security reduction that $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, q_{\mathbf{\Pi}}, q_{\mathbf{\Pi}^{-1}}, q_{\mathbf{H}})$-breaks the UF-NMA security of GenDSA. $\mathcal{A} = \mathcal{A}^{\mathbf{H}'(\cdot),\mathbf{\Pi}'(\cdot),\mathbf{\Pi}'^{-1}(\cdot)}$ has access to the programmable bRO $(\mathbf{\Pi}', \mathbf{\Pi}'^{-1})$ and the random oracle $\mathbf{H}'$ of the n-UF-NMA game. $\mathcal{R} = \mathcal{R}^{\mathbf{H}(\cdot),\mathbf{\Pi}(\cdot),\mathbf{\Pi}^{-1}(\cdot)}$ has access to the programmable bRO $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ and the programmable random oracle $\mathbf{H}$ of the UF-NMA game and internally simulates the programmable RO $\mathbf{H}'$ and programmable bRO $(\mathbf{\Pi}', \mathbf{\Pi}'^{-1})$, which are accessed by $\mathcal{A}$. Meta reduction $\mathcal{M}$ tries to solve the 1-FBOMDL assumption by running the reduction $\mathcal{R}$, simulating the UF-NMA game, the adversary $\mathcal{A}$ once and the oracles $\mathbf{H}$ and $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$.

Meta reduction $\mathcal{M}$ and adversary $\mathcal{A}$ are described in Figure 8.

$\mathcal{M}(X)$ receives a FBOMDL challenge $X = g^x$ and sets $\mathsf{vk} = X$. It runs $\mathcal{R}^{\mathbf{H}(\cdot),\mathbf{\Pi}(\cdot),\mathbf{\Pi}^{-1}(\cdot)}(\mathsf{vk})$ on the signing key vk and simulates the random oracle $\mathbf{H}$ and the bRO $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$ via lazy sampling. $\mathbf{H}$ is simulated perfectly random without programming. When $\mathcal{R}$ queries $\mathbf{\Pi}(z)$ for a fresh z with $z = \varphi(Y)$, it has to provide the representation a, b, c s.t. $Y = g^a \cdot X^b \cdot X'^c$ and $\mathcal{M}$ stores $(Y, z, (a, b, c), \mathbf{\Pi}(z))$ in a list L, where X' is the additional **DL** challenge that $\mathcal{A}$ queries at some point. If $z \notin \varphi(\mathbb{G}^*)$, then $\mathcal{M}$ only samples a random value to return. For duplicate queries, $\mathcal{M}$ answers consistently using the list L. When $\mathcal{R}$ queries $\mathbf{\Pi}^{-1}(z')$ on a fresh z', $\mathcal{M}$ samples a fresh $z \in \{0,1\}^L$. If $z \in \varphi(\mathbb{G}^*)$,

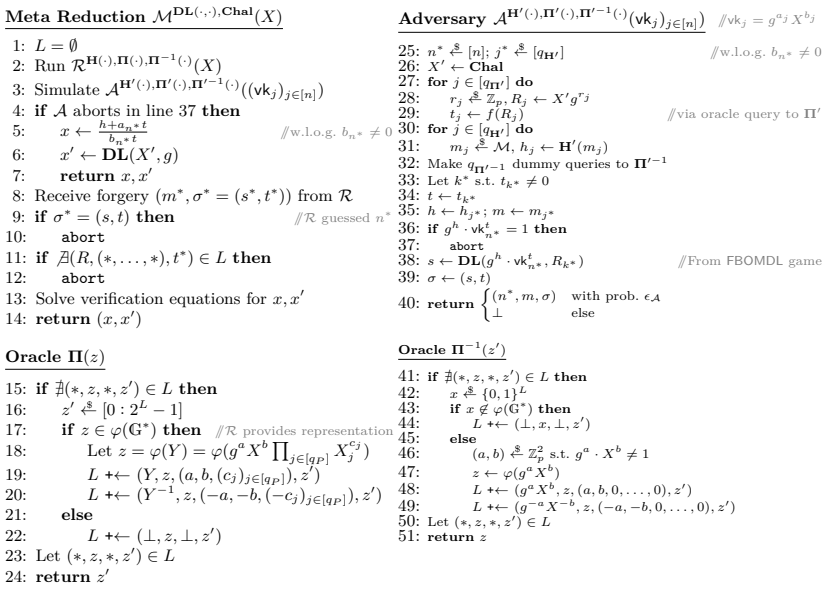

Fig. 8. Meta reduction $\mathcal{M}$ and simulated n-UF-NMA adversary $\mathcal{A}$ for Theorem 5.

it samples $(a, b) \xleftarrow{\$} \mathbb{Z}_p^2$ s.t. $R = g^a X^b \neq 1$ and sets $\mathbf{\Pi}^{-1}(z') = \varphi(R)$. It stores $(R, \varphi(R), (a, b, 0), z')$ (and $(R^{-1}, \varphi(R), (-a, -b, 0), z')$ if φ is 2-to-1) in L and returns $\varphi(R)$ to $\mathcal{R}$. Otherwise, it sets $\mathbf{\Pi}^{-1}(z') = z$, stores $(\perp, z, \perp, z')$ in L and returns z. Note that this simulates a random *function* instead of a random permutation. By the birthday bound, this is only incurs a statistical error of $q^2/2^L$ due to possibly causing collisions, where $q = q_{\mathbf{\Pi}} + q_{\mathbf{\Pi}^{-1}}$ is the number of queries made by $\mathcal{R}$ to $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$.

Eventually, $\mathcal{R}$ invokes the n-UF-NMA adversary $\mathcal{A} = \mathcal{A}^{\mathbf{H}'(\cdot),\mathbf{\Pi}'(\cdot),\mathbf{\Pi}'^{-1}(\cdot)}(\mathsf{vk}_1, \dots, \mathsf{vk}_n)$ on signing keys $(\mathsf{vk}_i)_{i\in[n]}$. All oracles for $\mathcal{A}$ are under the reductions control. Since $\mathcal{R}$ is algebraic, it also provides representations for the signing keys, i.e. $\mathsf{vk}_i = g^{a_i} X^{b_i}$ for $i \in [n]$. Note that all keys are independent of X' since it is unknown to $\mathcal{R}$ at this point.

$\mathcal{A}$ is simulated by $\mathcal{M}$ and begins by sampling random indices $n^* \in [n]$ and $j^* \in [q_{\mathbf{H}'}]$. Without loss of generality, we assume that $b_{n^*} \neq 0$, as otherwise the secret key corresponding to vk_{n^*} is a_{n^*} and $\mathcal{A}$ can simply compute an honest signature using said key.

$\mathcal{A}$ begins by invoking the challenge oracle **Chal** and embeds the challenge X' in all queries to $\mathbf{\Pi}'$ by randomizing it with R_j for $j \in [q_{\mathbf{\Pi}'}]$. It then proceeds to make the appropriate number of queries to its other oracles. Next, $\mathcal{A}$ picks an index k^* s.t. $t_{k^*} \neq 0$. Again, without loss of generality, we assume that such an index exists, as otherwise all valid signatures are information theoretically hidden from *any* adversary and the reduction could compute a valid forgery with at least the same probability as the adversary without using it.

It then attempts to create a forgery on m_{j^*} using t_{k^*} for the key vk_{n^*}. First, it checks whether

$$g^{h_{j^*}} \cdot \mathsf{vk}_{n^*}^{t_{k^*}} = 1.$$

This would imply that the corresponding signature part $s = 0$, which results in an invalid signature. However, the above equation also implies

$$h_{j^*} + (a_{n^*} + b_{n^*}x)t_{k^*} = 0,$$

and since by assumption $b_{n^*} \neq 0$ and $t_{k^*} \neq 0$, $\mathcal{M}$ can solve this equation for x and get x' with a query to **DL**, solving its FBOMDL challenge.

So assume that $g^{h_{j^*}} \cdot \mathsf{vk}_{n^*}^{t_{k^*}} \neq 1$. Then $\mathcal{A}$ uses its query to **DL** to produce a valid forgery and returns it with the appropriate probability. Then the simulation of $\mathcal{A}$ is perfect. Therefore, $\mathcal{R}$ will eventually produce a forgery $\sigma^* = (s^*, t^*)$ with probability $\epsilon_{\mathcal{R}}$. If $\mathsf{vk}_{n^*} = X$, then $\mathcal{M}$ aborts, as in this case $\mathcal{R}$ will just forward the signature created by $\mathcal{A}$ and $\mathcal{M}$ won't be able to solve its challenge. We call this event BAD_1. However, this abort can only occur with probability $\frac{1}{n}\epsilon_{\mathcal{A}}$, as $\mathcal{R}$ can embed its own key only once in $\mathcal{A}$'s challenge, as the keys have to be distinct, so

$$\Pr[\text{BAD}_1] \leq \frac{1}{n}\epsilon_{\mathcal{A}}$$

So assume that $\sigma^* \neq \sigma$. Then we have the two equations

$$x's = h + (a_{n^*} + b_{n^*}x)t \tag{16}$$

$$r^*s^* = \mathbf{H}(m^*) + xt^* \tag{17}$$

with $g^{r^*} =: R^*$. We consider two cases depending on the queries $\mathcal{R}$ made to $(\mathbf{\Pi}, \mathbf{\Pi}^{-1})$:

CASE 1: $\nexists(R^*, z, (a^*, b^*, c^*), z') \in L$ WITH $\psi(z') = t^*$. In this case, $\mathcal{R}$ did not compute t^* from $\mathbf{\Pi}$ output or queried $\psi^{-1}(t^*)$ to $\mathbf{\Pi}^{-1}$. $\mathcal{M}$ aborts in this case as it will not be able to solve its FBOMDL instance. We call this event BAD_2. Consider the random variable $y^* = \mathbf{\Pi}(\varphi(R^*))$. Since $\mathbf{\Pi}$ is a random function, y^* is distributed uniform over $[0 : 2^L - 1]$, so $\Pr[\psi(y^*) = t^*] \leq \epsilon_\psi$. This allows us to bound

$$\Pr[\text{BAD}_2] \leq \epsilon_\psi.$$

CASE 2 : $\exists(R^*, z, (a^*, b^*, c^*), z') \in L$ WITH $\psi(z') = t^*$. Here, $\mathcal{R}$ queried either $\psi^{-1}(t^*)$ to $\mathbf{\Pi}^{-1}$ or computed t^* from a query to $\mathbf{\Pi}$ on the image of φ of a group element. In both cases, we get a representation for $R^* = g^{r^*}$ with $\psi(\mathbf{\Pi}(\varphi(R^*))) = t^*$. Substituting in the above equation and rearranging slightly yields

$$-h - a_{n^*}t = b_{n^*}tx - sx'$$
$$a^*s^* - \mathbf{H}(m^*) = (t^* - b^*s^*)x - s^*c^*x'$$

The equation system has determinant

$$D = st^* - b_{n^*}ts^*c^* - sb^*s^*$$

and is solvable if and only if $D \neq 0$. Since σ and σ' are valid signatures, we know that t^*, t, s^* and s are non-zero. Assume that $D = 0$, which we call BAD_3. Then

$$s^*(b^*s + b_{n^*}tc^*) = st^*$$

Solving for s^*, we have $s^* = \frac{st^*}{\Delta}$ with $\Delta = b^*s + b_{n^*}tc^*$. Note that $\Delta \neq 0$ as otherwise $s = 0$ or $t^* = 0$. Replacing s^* in Equation (17), we get

$$\frac{(a^* + b^*x + c^*x')st^*}{\Delta} = \mathbf{H}(m^*) + t^*x.$$

Every term not including t^* contains either a^*, b^* or c^*. So if we solve for t^*, we get

$$t^* = \frac{\mathbf{H}(m^*)\Delta}{a^*s + (b^*s - \Delta)x + c^*x's},$$

where the denominator is non-zero with probability $1 - \frac{q_{\mathbf{H}}}{p}$, as it implies that $\mathbf{H}(m^*) = 0$. With the same argument as in Theorem 4, we get that the probability that $\mathcal{R}$ finds a^*, b^*, c^*, t^* and m^* which fulfil the above equation is bounded by $2\epsilon_\psi(q_{\mathbf{\Pi}} + q_{\mathbf{\Pi}^{-1}})q_{\mathbf{H}}$, so

$$\Pr[\text{BAD}_3] \leq 2\epsilon_\psi(q_{\mathbf{\Pi}} + q_{\mathbf{\Pi}^{-1}})q_{\mathbf{H}}.$$

Combining all probabilities of the bad cases and the birthday bound yields the theorem. □