# On Tight Security Proofs for Schnorr Signatures

Nils Fleischhacker Tibor Jager Dominique Schröder

April 7, 2014

**Abstract**

The Schnorr signature scheme is the most efficient signature scheme based on the discrete logarithm problem and a long line of research investigates the existence of a *tight* security reduction for this scheme. Almost all recent works present lower tightness bounds and most recently Seurin (Eurocrypt 2012) showed that under certain assumptions the *non*-tight security proof for Schnorr signatures by Pointcheval and Stern (Eurocrypt 1996) is essentially optimal. All previous works in this direction rule out tight reductions from the (one-more) discrete logarithm problem. In this paper we introduce a new meta-reduction technique, which shows lower bounds for the large and very natural class of *generic* reductions. A generic reduction is independent of a particular representation of group elements and most reductions in state-of-the-art security proofs have this desirable property. Our approach shows *unconditionally* that there is no tight generic reduction from any *natural* computational problem $\Pi$ defined over algebraic groups (including even interactive problems) to breaking Schnorr signatures, unless solving $\Pi$ is easy.

**Keywords:** Schnorr signatures, black-box reductions, generic reductions, algebraic reductions, tightness.

## 1 Introduction

The security of a cryptosystem is nowadays usually confirmed by giving a security proof. Typically, such a proof describes a *reduction* from some (assumed-to-be-)hard computational problem to breaking a defined security property of the cryptosystem. A reduction is considered as *tight*, if the reduction solving the hard computational problem has essentially the same running time and success probability as the attacker on the cryptosystem. Essentially, a tight reduction means that a successful attacker can be turned into an efficient algorithm for the hard computational problem *without* any significant increase in the running time and/or significant loss in the success probability.[1] The tightness of a reduction thus determines the strength of the security guarantees provided by the security proof: a non-tight reduction gives weaker security guarantees than a tight one. Moreover, tightness of the reduction affects the efficiency of the cryptosystem when instantiated in practice: a tighter reduction allows to securely use smaller parameters (shorter moduli, a smaller group size, etc.). Therefore it is a very desirable property of a cryptosystem to have a tight security reduction.

In the domain of digital signatures tight reductions are known for many fundamental schemes, like Rabin/Williams signatures (Bernstein, Eurocrypt 2008 [5]), many strong-RSA-based signatures (Schäge, Eurocrypt 2011 [23]), and RSA Full-Domain Hash (Kakvi and Kiltz, Eurocrypt 2012 [16]). The Schnorr

[1] Usually even a polynomially-bounded increase/loss is considered as significant, if the polynomial may be large. An increase/loss by a small constant factor is not considered as significant.

signature scheme [24, 25] is one of the most fundamental public-key cryptosystems. Pointcheval and Stern have shown that Schnorr signatures are provably secure, assuming the hardness of the discrete logarithm (DL) problem [20], in the Random Oracle Model (ROM) [3]. However, the reduction of Pointcheval and Stern from DL to breaking Schnorr signatures is not tight: it loses a factor of $q$ in the time-to-success ratio, where $q$ is the number of random oracle queries performed by the forger.

A long line of research investigates the existence of tight security proofs for Schnorr signatures. At Asiacrypt 2005 Paillier and Vergnaud [19] gave a first lower bound showing that any algebraic reduction (even in the ROM) converting a forger for Schnorr signatures into an algorithm solving some computational problem $\Pi$ must lose a factor of at least $q^{1/2}$. Their result is quite strong, as they rule out reductions even for adversaries that do not have access to a signing oracle and receive as input the message for which they must forge (UF-NM, see Section 2.1 for a formal definition). However, their result also has some limitations: It holds only under the interactive one-more discrete logarithm assumption, they only consider algebraic reductions, and they only rule out tight reductions from the (one-more) discrete logarithm problem. At Crypto 2008 Garg *et al.* [14] refined this result, by improving the bound from $q^{1/2}$ to $q^{2/3}$ with a new analysis and show that this bound is optimal if the meta-reduction follows a particular approach for simulating the forger. At Eurocrypt 2012 Seurin [26] finally closed the gap between the security proof of [20] and known impossibility results, by describing an elaborate simulation strategy for the forger and providing a new analysis. All previous works [19, 14, 26] on the existence of tight security proofs for Schnorr signatures have the following in common:

1. They only rule out the existence of tight reductions from certain strong computational problems, namely the (one-more) discrete logarithm problem [1]. Reduction from weaker problems like, e.g., the computational or decisional Diffie-Hellman problem (CDH/DDH) are not considered.
2. The impossibility results are themselves only valid under the very strong OMDL hardness assumption.
3. They hold only with respect to a limited (but natural) class of reductions, so-called *algebraic reductions*.

It is not unlikely that first the inexistence of a tight reduction from *strong* computational problems is proven, and later a tight reduction from some *weaker* problem is found. A concrete recent example in the domain of digital signatures where this has happened is RSA Full-Domain Hash (RSA-FDH) [4]. First, at Crypto 2000 Coron [7] described a non-tight reduction from solving the RSA-problem to breaking the security of RSA-FDH, and at Eurocrypt 2002 [8] showed that under certain conditions no tighter reduction from RSA can exist. Later, at Eurocrypt 2012, Kakvi and Kiltz [16] gave a tight reduction from solving a weaker problem, the so-called Phi-Hiding problem. The leverage to circumvent the aforementioned impossibility results used by Kakvi and Kiltz was to assume hardness of a weaker computational problem. As all previous works rule out only tight reductions from strong computational problems like DL and OMDL, this might happen again with Schnorr signatures and the following question was left open for 25 years:

> *Does a tight security proof for Schnorr signatures based on any weaker computational problem exist?*

**Our contribution** In this work we answer this question in the negative ruling out the existence of tight reductions for virtually all natural computational problems defined over abstract algebraic groups. Like previous works, we consider universal unforgeability under no-message attacks (UF-NM-security). Moreover, our results hold unconditional. In contrast to previous works, we consider *generic* reductions instead of algebraic reductions, but we believe that this restriction is marginal: The motivation of considering only algebraic reductions from [19] applies equally to generic reductions. In particular, to the best of our knowledge all known examples of algebraic reductions are generic.

Our main technical contribution is a new approach for the simulation of a forger in a meta-reduction, i.e., "a reduction against the reduction", which differs from previous works [19, 14, 26] and which allows us to show the following main result:

**Theorem** (informal)**.** *For almost any* natural *computational problem* $\Pi$*, there is no tight* generic *reduction from solving* $\Pi$ *to breaking the universal unforgeability under no-message attacks of Schnorr signatures.*

**Technical approach.** We begin with the hypothesis that there exists a tight generic reduction $\mathcal{R}$ from some hard (and possibly interactive) problem $\Pi$ to the UF-NM-security of Schnorr signatures. Then we show that under this hypothesis there exists an efficient algorithm $\mathcal{M}$, a meta-reduction, which efficiently solves $\Pi$. This implies that the hypothesis is false. The meta-reduction $\mathcal{M} = \mathcal{M}^{\mathcal{R}}$ runs $\mathcal{R}$ as a subroutine, by efficiently *simulating* the forger $\mathcal{A}$ for $\mathcal{R}$.

All previous works in this direction [19, 14, 26] followed essentially the same approach. The difficulty with meta-reductions is that $\mathcal{M} = \mathcal{M}^{\mathcal{R}}$ must efficiently *simulate* the forger $\mathcal{A}$ for $\mathcal{R}$. Previous works resolved this by using a discrete logarithm oracle provided by the OMDL assumption, which allows to efficiently compute valid signatures in the simulation of forger $\mathcal{A}$. This is the reason why all previous results are only valid under the OMDL assumption, and were only able to rule out reductions from the discrete log or the OMDL problem. To overcome these limitations, a new simulation technique is necessary.

We revisit the simulation strategy of $\mathcal{A}$ applied in known meta-reductions, and put forward a new technique for proving impossibility results. It turns out that considering *generic* reductions provides a new leverage to simulate a successful forger efficiently, essentially by suitably re-programming the group representation to compute valid signatures. The technical challenge is to prove that the reduction does not notice that the meta-reduction changes the group representation during the simulation, except for some negligible probability. We show how to prove this by adopting the "low polynomial degree" proof technique of Shoup [27], which originally was introduced to analyze the complexity of certain algorithms for the discrete logarithm problem, to the setting considered in this paper.

This new approach turns out to be extremely powerful, as it allows to rule out reductions from any (even *interactive*) *representation-invariant* computational problem. Since almost all common hardness assumptions in algebraic groups (e.g., DL, CDH, DDH, OMDL, DLIN, etc.) are based on representation-invariant computational problems, we are able to rule out tight generic reductions from virtually any natural computational problem, without making any additional assumption. Even though we apply it specifically to Schnorr signatures, the overall approach is general. We expect that it is applicable to other cryptosystems as well.

**Generic reductions vs. algebraic reductions** Similar to algebraic reductions, a generic reduction performs only group operations. The main difference is that the sequence of group operations performed by an algebraic reduction may (but, to our best knowledge, in all known examples does not) depend on a particular representation of group elements. A generic reduction, however, is required to work essentially identical for any representation of group elements. Generic reductions are by definition more restrictive than algebraic ones, however, we explain below why we do not consider this restriction as very significant.

An obvious question arising with our work is the relation between algebraic and generic reductions. Is a lower bound for generic reductions much less meaningful than a bound for algebraic reductions? We argue that the difference is not very significant. The restriction to algebraic reductions was motivated by the fact most reductions in known security proofs treat the group as a black-box, and thus are algebraic [19, 14, 26]. However, the same motivation applies to generic reductions as well, with exactly the same arguments. In particular, virtually all examples of algebraic reductions in the literature are also generic.

The vast majority of reductions in common security proofs for group-based cryptosystems treats the underlying group as a black-box (i.e., works for any representation of the group), and thus is generic. This is a very desirable feature, because then the cryptosystem can securely be instantiated with *any* group in which the underlying computational problem is hard. In contrast, representation-specific security proofs would require to re-prove security for any particular group representation the scheme is used with. Therefore considering generic reductions seems very reasonable.

**Generic reductions vs. security proofs in the generic group model.** We stress that we model only the reduction $\mathcal{R}$ as a generic algorithm. We do not restrict the forger $\mathcal{A}$ in this way, as commonly done in security proofs in the generic group model. It may not be obvious that this is possible, because $\mathcal{A}$ expects as input group elements in some specific encoding, while $\mathcal{R}$ can only specify them in the form of random encodings. However, the reduction only gets access to the adversary as a blackbox, which means that the adversary is external to the reduction, and the environment in which the reduction is run can easily translate between the encodings used by reduction and adversary. Further note, that while some reduction from a problem $\Pi$ may be generic, the actual algorithm solving said problem is not $\mathcal{R}$ itself, but the composition of $\mathcal{R}$ and $\mathcal{A}$ which may very well be non-generic. In particular, this means that any results about equivalence of interesting problems in the generic group model do not apply to the reduction. See Section 2.4 and Figure 2 for further explanation.

**Further related work.** Dodis *et al.* [9] showed that it is impossible to reduce any computational problem to breaking the security of RSA-FDH in a model where the RSA-group $\mathbb{Z}_N^*$ is modeled as a generic group. This result extends [10]. Coron [8] considered the existence of tight security reductions for RSA-FDH signatures [4]. This result was generalized by Dodis and Reyzin [11] and later refined by Kiltz and Kakvi [16].

In the context of Schnorr signatures, Neven *et al.* [18] described necessary conditions the hash function must meet in order to provide existential unforgeability under chosen-message attacks (EUF-CM), and showed that these conditions are sufficient if the forger (not the reduction!) is modeled as a generic group algorithm.

In [12] Fischlin and Fleischhacker presented a result also about the security of Schnorr signatures which is orthogonal to our result. They show, again under the OMDL assumption, that a large class of reductions has to rely on re-programming the random oracle. Essentially they prove that in the *non-programmable* ROM [13] no reduction from the discrete logarithm problem can exist that invokes the adversary only ever on the same input. This class is limited, but encompasses all forking-lemma style reductions used to prove Schnorr signatures secure in the programmable ROM. As said before, the result is orthogonal to our main result, as it considers reductions in the *non-programmable* ROM.

## 2 Preliminaries

**Notation.** If $S$ is a set, we write $s \leftarrow_\$ S$ to denote the action of sampling a uniformly random element $s$ from $S$. If $A$ is a probabilistic algorithm, we denote with $a \leftarrow_\$ A$ the action of computing $a$ by running $A$. We denote with $\emptyset$ the empty string, the empty set, as well as the empty list, the meaning will always be clear from the context. We write $[n]$ to denote the set of integers from 1 to $n$, i.e., $[n] := \{1, \ldots, n\}$.

## 2.1 Schnorr Signatures

Let $\mathbb{G}$ be a group of order $p$ with generator $g$, and let $H : \mathbb{G} \times \{0,1\}^k \to \mathbb{Z}_p$ be a hash function. The Schnorr signature scheme [24, 25] consists of the following efficient algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

$\mathsf{Gen}(g)$**:** The key generation algorithm takes as input a generator $g$ of $\mathbb{G}$. It chooses $x \leftarrow_\$ \mathbb{Z}_p$, computes $X := g^x$, and outputs $(X, x)$.

$\mathsf{Sign}(x, m)$**:** The input of the signing algorithm is a private key $x$ and a message $m \in \{0,1\}^k$. It chooses a random integer $r \leftarrow_\$ \mathbb{Z}_p$, sets $R := g^r$ as well as $c := H(R, m)$, and computes $y := x \cdot c + r \bmod p$.

$\mathsf{Vrfy}(X, m, (R, y))$**:** The verification algorithm outputs the truth value of $g^y \stackrel{?}{=} X^c \cdot R$, where $c = H(R, m)$.

**Universal Unforgeability under No-Message Attacks.** Consider the following security experiment involving a signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, an attacker $\mathcal{A}$, and a challenger $\mathcal{C}$.

1. The challenger $\mathcal{C}$ computes a key-pair $(X, x) \leftarrow_\$ \mathsf{Gen}(g)$ and chooses a message $m \leftarrow_\$ \{0,1\}^k$ uniformly at random. It sends $(X, m)$ to the adversary $\mathcal{A}$.
2. Eventually, $\mathcal{A}$ stops, outputting a signature $\sigma$.

**Definition 1.** We say that $\mathcal{A}$ $(\epsilon, t)$-breaks the UF-NM-security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, if $\mathcal{A}$ runs in time at most $t$ and

$$\Pr\left[\mathcal{A}(X, m) = \sigma : \mathsf{Vrfy}(X, m, \sigma) = 1\right] \geq \epsilon.$$

Note that UF-NM-security is a very weak security goal for digital signatures. Since we are going to prove a negative result, this is not a limitation, but makes our result only stronger. In fact, if we rule out reductions from some problem $\Pi$ to forging signatures in the sense of UF-NM, then the impossibility clearly holds for stronger security goals, like existential unforgeability under adaptive chosen-message attacks [15], too.

## 2.2 Computational Problems

Let $\mathbb{G}$ be a cyclic group of order $p$ and $g \in \mathbb{G}$ a generator of $\mathbb{G}$. We write $\mathsf{desc}(\mathbb{G}, g)$ to denote the list of group elements $\mathsf{desc}(\mathbb{G}, g) = (g, g^2, \ldots, g^p) \in \mathbb{G}^p$. We say that $\mathsf{desc}(\mathbb{G}, g)$ is the *enumerating description* of $\mathbb{G}$ with respect to $g$.

**Definition 2.** A *computational problem* $\Pi$ in $\mathbb{G}$ is specified by three (computationally unbounded) procedures $\Pi = (\mathcal{G}_\Pi, \mathcal{S}_\Pi, \mathcal{V}_\Pi)$, with the following syntax.

$\mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ takes as input an enumerating description of $\mathbb{G}$, and outputs a state $st$ and a problem instance (the challenge) $C = (C_1, \ldots, C_u, C') \in \mathbb{G}^u \times \{0,1\}^*$. We assume in the sequel that at least $C_1$ is a generator of $\mathbb{G}$.

$\mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$ takes as input $\mathsf{desc}(\mathbb{G}, g)$, a state $st$, and $Q = (Q_1, \ldots, Q_v, Q') \in \mathbb{G}^v \times \{0,1\}^*$, and outputs $(st', A)$ where $st'$ is an updated state and $A = (A_1, \ldots, A_\nu, A') \in \mathbb{G}^\nu \times \{0,1\}^*$.

$\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, S, C)$ takes as input $(\mathsf{desc}(\mathbb{G}, g), st, C)$ as defined above, and $S = (S_1, \ldots, S_w, S') \in \mathbb{G}^w \times \{0,1\}^*$. It outputs 0 or 1.

If $\mathcal{S}_\Pi$ always responds with $A = \emptyset$ (i.e., the empty string), then we say that $\Pi$ is *non-interactive*. Otherwise it is *interactive*. The exact description and distribution of $st, C, Q, A, S$ depends on the considered computational problem.

**Definition 3.** An algorithm $\mathcal{A}$ $(\epsilon, t)$-solves the computational problem $\Pi$ if $\mathcal{A}$ has running time at most $t$ and wins the following interactive game against a (computationally unbounded) challenger $\mathcal{C}$ with probability at most $\epsilon$, where the game is defined as follows:

1. The challenger $\mathcal{C}$ generates an instance of the problem $(st, C) \leftarrow_\$ \mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ and sends $C$ to $\mathcal{A}$.
2. $\mathcal{A}$ is allowed to issue an arbitrary number of *oracle queries* to $\mathcal{C}$. To this end, $\mathcal{A}$ provides $\mathcal{C}$ with a query $Q$. $\mathcal{C}$ runs $(st', A) \leftarrow_\$ \mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$, updates the state $st := st'$, and responds with $A$.
3. Finally, algorithm $\mathcal{A}$ outputs a candidate solution $S$. The algorithm $\mathcal{A}$ wins the game (i.e., solves the computational problem correctly) iff $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S) = 1$.

*Example* 4. The *discrete logarithm problem* in $\mathbb{G}$ is specified by the following procedures. $\mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ outputs $(st, C)$ with $st = \emptyset$ and $C = (g, h)$, where $h \leftarrow_\$ \mathbb{G}$ is a random group element. $\mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$ always outputs $(st', A) = (st, \emptyset)$. $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S)$ interprets $S = S' \in \{0,1\}^*$ canonically as an integer in $\mathbb{Z}_p$, and outputs 1 iff $h = g^{S'}$.

*Example* 5. We describe the *$u$-one-more discrete logarithm problem* ($u$-OMDL) [2, 1] in $\mathbb{G}$ with the following algorithms. $\mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ outputs $(st, C)$ where $C = (C_1, \ldots, C_u) \leftarrow_\$ \mathbb{G}^u$ consists of $u$ random group elements and $st = 0$. The algorithm $\mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$ takes as input state $st$ and group element $Q \in \mathbb{G}$. It responds with $st' := st + 1$ and $A = A' \in \{0,1\}^*$, where $A'$ canonically interpreted as an integer in $\mathbb{Z}_p$ satisfies $g^{A'} = Q$. The verification algorithm $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S)$ interprets $S = (S'_1, \ldots, S'_u) \in \{0,1\}^*$ canonically as a vector of $u$ integers in $\mathbb{Z}_p$, and outputs 1 iff $st < u$ and $g_i = g^{S'_i}$ for all $i \in [u]$.

*Example* 6. The UF-NM-forgery problem for Schnorr signatures in $\mathbb{G}$ with hash function $H$ is specified by the following procedures. $\mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ outputs $(st, C)$ with $st = m$ and $C = (g, X, m) \in \mathbb{G}^2 \times \{0,1\}^k$, where $X = g^x$ for $x \leftarrow_\$ \mathbb{Z}_p$ and $m \leftarrow_\$ \{0,1\}^k$. $\mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$ always outputs $(st', A) = (st, \emptyset)$. The verification algorithm $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S)$ parses $S$ as $S = (R, y) \in \mathbb{G} \times \mathbb{Z}_p$, sets $c := H(R, st)$, and outputs 1 iff $X^c \cdot R = g^y$.

### 2.3 Representation-Invariant Computational Problems

In our impossibility results given below, we want to rule out the existence of a tight reduction from as large a class of computational problems as possible. Ideally, we want to rule out the existence of a tight reduction from any computational problem that meets Definition 2. However, it is easy to see that this is not achievable in this generality: as Example 6 shows, the problem of forging Schnorr signatures itself is a problem that meets Definition 2. However, of course there exists a trivial tight reduction from the problem of forging Schnorr signatures to the problem of forging Schnorr signatures! Therefore we need to restrict the class of considered computational problems to exclude such trivial, artificial problems.

We introduce the notion of *representation-invariant* computational problems. This class of problems captures virtually any reasonable computational problem defined over an abstract algebraic group, even interactive assumptions, except for a few extremely artificial problems. In particular, the problem of forging Schnorr signatures is *not* contained in this class (see Example 9 below).

Intuitively, a computational problem is *representation-invariant*, if a valid solution to a given problem instance remains valid even if the representation of group elements in challenges, oracle queries, and solutions is converted to a different representation of the same group. More formal is the following definition:

**Definition 7.** Let $\mathbb{G}, \hat{\mathbb{G}}$ be groups such that there exists an isomorphism $\phi : \mathbb{G} \to \hat{\mathbb{G}}$. We say that $\Pi$ is *representation-invariant*, if for all isomorphic groups $\mathbb{G}, \hat{\mathbb{G}}$ and for all generators $g \in \mathbb{G}$, all $C = (C_1, \ldots, C_u, C') \leftarrow_\$ \mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$, all $st = (st_1, \ldots, st_t, st') \in \mathbb{G}^t \times \{0,1\}^*$, and all $S = (S_1, \ldots, S_w, S') \in \mathbb{G}^w \times \{0,1\}^*$ holds that $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S) = 1 \iff \mathcal{V}_\Pi(\mathsf{desc}(\hat{\mathbb{G}}, \hat{g}), \hat{st}, \hat{C}, \hat{S}) = 1$, where $\hat{g} = \phi(g) \in \mathbb{G}'$, $\hat{C} = (\phi(C_1), \ldots, \phi(C_u), C')$, $\hat{st} = (\phi(st_1), \ldots, \phi(st_t), st')$, and $\hat{S} = (\phi(S_1), \ldots, \phi(S_w), S')$.

Observe that this definition only demands the existence of an isomorphism $\phi : \mathbb{G} \to \hat{\mathbb{G}}$ and not that it is efficiently computable.

*Example* 8. The discrete logarithm problem is representation-invariant. Let $C = (g, h) \in \mathbb{G}^2$ be a discrete log challenge, with corresponding solution $S' \in \{0,1\}^*$ such that $S'$ canonically interpreted as an integer $S' \in \mathbb{Z}_p$ satisfies $g^{S'} = h \in \mathbb{G}$. Let $\phi : \mathbb{G} \to \hat{\mathbb{G}}$ be an isomorphism, and let $(\hat{g}, \hat{h}) := (\phi(g), \phi(h))$. Then it clearly holds that $\hat{g}^{\hat{S}'} = \hat{h}$, where $\hat{S}' = S'$.

Virtually all common hardness assumptions in algebraic groups are based on representation-invariant computational problems. Popular examples are, for instance, the discrete log problem (DL), computational Diffie-Hellman (CDH), decisional Diffie-Hellman (DDH), one-more discrete log (OMDL), decision linear (DLIN), and so on.

*Example* 9. The UF-NM-forgery problem for Schnorr signatures with hash function $H$ is *not* representation-invariant for any hash function $H$. Let $C = (g, X, m) \leftarrow_\$ \mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ be a challenge with solution $S = (R, y) \in \mathbb{G} \times \mathbb{Z}_p$ satisfying $X^c \cdot R = g^y$, where $c := H(R, m)$.

Let $\hat{\mathbb{G}}$ be a group isomorphic to $\mathbb{G}$, such that $\mathbb{G} \cap \hat{\mathbb{G}} = \emptyset$ (that is, there exists no element of $\hat{\mathbb{G}}$ having the same representation as some element of $\mathbb{G}$).[2] Let $\mathbb{G} \to \hat{\mathbb{G}}$ denote the isomorphism. If there exists any $R$ such that $H(R, m) \neq H(\phi(R), m)$ in $\mathbb{Z}_p$ (which holds in particular if $H$ is collision resistant), then we have

$$g^y = X^{H(R,m)} \cdot R \qquad \text{but} \qquad \phi(g)^y \neq \phi(X)^{H(\phi(R),m)} \cdot \phi(R).$$

Thus, a solution to this problem is valid only with respect to a particular given representation of group elements.

The UF-NM-forgery problem of Schnorr signatures is not representation-invariant, because a solution to this problem involves the hash value $H(R, m)$ that depends on a concrete representation of group element $R$. We consider such complexity assumptions as rather unnatural, as they are usually very specific to certain constructions of cryptosystems.

## 2.4 Generic Reductions

In this section we recall the notion of *generic groups*, loosely following [27] (cf. also [17, 22], for instance), and define generic (i.e., representation independent) reductions.

**Generic groups.** Let $(\mathbb{G}, \cdot)$ be a group of order $p$ and $E \subseteq \{0,1\}^{\lceil \log p \rceil}$ be a set of size $|E| = |\mathbb{G}|$. If $g, h \in \mathbb{G}$ are two group elements, then we write $g \div h$ for $g \cdot h^{-1}$. Following [27] we define an *encoding function* as a random injective map $\phi : \mathbb{G} \to E$. We say that an element $e \in E$ is the *encoding* assigned to group element $h \in \mathbb{G}$, if $\phi(h) = e$.

A *generic group algorithm* is an algorithm $\mathcal{R}$ which takes as input $\hat{C} = (\phi(C_1), \ldots, \phi(C_u), C')$, where $\phi(C_i) \in E$ is an encoding of group element $C_i$ for all $i \in [u]$, and $C' \in \{0,1\}^*$ is a bit string. The algorithm outputs $\hat{S} = (\phi(S_1), \ldots, \phi(S_w), S')$, where $\phi(S_i) \in E$ is an encoding of group element $S_i$ for all $i \in [w]$, and $S' \in \{0,1\}^*$ is a bit string. In order to perform computations on encoded group elements, algorithm $\mathcal{R} = \mathcal{R}^{\mathcal{O}}$ may query a *generic group oracle* (or "*group oracle*" for short). This oracle $\mathcal{O}$ takes as input two encodings $e = \phi(G), e' = \phi(G')$ and a symbol $\circ \in \{\cdot, \div\}$, and returns $\phi(G \circ G')$. Note that $(E, \cdot_{\mathcal{O}})$, where $\cdot_{\mathcal{O}}$ denotes the group operation on $E$ induced by oracle $\mathcal{O}$, forms a group which is isomorphic to $(\mathbb{G}, \cdot)$.

[2] Such a group $\hat{\mathbb{G}}$ can trivially be obtained for any group $\mathbb{G}$, for instance by modifying the encoding by prepending a suitable fixed string to each group element, and changing the group law accordingly.

| PROC $\mathcal{O}(e, e', \circ)$ | PROC GETIDX$(\vec{e})$ | PROC ENCODE$(G)$ |
|---|---|---|
| $(e, e', \circ) \in E \times E \times \{\cdot, \div\}$ | parse $\vec{e} = (e_1, \ldots, e_w)$ | parse $G = (G_1, \ldots, G_u)$ |
| $(i, j) :=$ GETIDX$(e, e')$ | for $j = 1, \ldots, w$ do | for $j = 1, \ldots, u$ do |
| return ENCODE$(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$ | pick first $i \in [\lvert\mathcal{L}^E\rvert]$ | if $\exists i$ s.t. $\mathcal{L}_i^{\mathbb{G}} = G_j$ |
| | such that $\mathcal{L}_i^E = e_j$ | $e_j := \mathcal{L}_i^E$ |
| | $i_j := i$ | else |
| | return $(i_1, \ldots, i_w)$ | $e_j \leftarrow_{\$} E \setminus \mathcal{L}^E$ |
| | | append $e_j$ to $\mathcal{L}^E$ |
| | | append $G_j$ to $\mathcal{L}^{\mathbb{G}}$ |
| | | return $(e_1, \ldots, e_u)$ |

Figure 1: Procedures implementing the generic group oracle.

It will later be helpful to have a specific implementation of $\mathcal{O}$. We will therefore assume in the sequel that $\mathcal{O}$ internally maintains two lists $\mathcal{L}^{\mathbb{G}} \subseteq \mathbb{G}$ and $\mathcal{L}^E \subseteq E$. These lists define the encoding function $\phi$ as $\mathcal{L}_i^E = \phi(\mathcal{L}_i^{\mathbb{G}})$, where $\mathcal{L}_i^{\mathbb{G}}$ and $\mathcal{L}_i^E$ denote the $i$-th element of $\mathcal{L}^{\mathbb{G}}$ and $\mathcal{L}^E$, respectively, for all $i \in [\lvert\mathcal{L}^{\mathbb{G}}\rvert]$. Note that from the perspective of a generic group algorithm it makes no difference whether the encoding function is fixed at the beginning or lazily evaluated whenever a new group element occurs. We will assume that the oracle uses lazy evaluation to simplify our discussion and avoid unnecessary steps for achieving polynomial runtime of our meta-reductions.

**Procedure ENCODE** takes a list $G = (G_1, \ldots, G_u)$ of group elements as input. It checks for each $G_j \in L$ if an encoding has already been assigned to $G_j$, that is, if there exists an index $i$ such that $\mathcal{L}_i^{\mathbb{G}} = G_j$. If this holds, ENCODE sets $e_j := \mathcal{L}_i^E$. Otherwise (if no encoding has been assigned to $G_j$ so far), it chooses a fresh and random encoding $e_j \leftarrow_{\$} E \setminus \mathcal{L}^E$. In either case $G_j$ and $e_j$ are appended to $\mathcal{L}^{\mathbb{G}}$ and $\mathcal{L}^E$, respectively, which gradually defines the map $\phi$ such that $\phi(G_j) = e_j$. Note also that the same group element and encoding may occur multiple times in the list. Finally, the procedure returns the list $(e_1, \ldots, e_u)$ of encodings.

**Procedure GETIDX** takes a list $(e_1, \ldots, e_w)$ of encodings as input. For each $j \in [w]$ it defines $i_j$ as the smallest[3] index such that $e_j = \mathcal{L}_{i_j}^E$, and returns $(i_1, \ldots, i_w)$.[4]

The lists $\mathcal{L}^{\mathbb{G}}$ and $\mathcal{L}^E$ are initially empty. Then $\mathcal{O}$ calls $(e_1, \ldots, e_u) \leftarrow_{\$}$ ENCODE$(G_1, \ldots, G_u)$ to determine encodings for all group elements $G_1, \ldots, G_u$ and starts the generic group algorithm on input $\mathcal{R}(e_1, \ldots, e_u, C')$.

$\mathcal{R}^{\mathcal{O}}$ may now submit queries of the form $(e, e', \circ) \in E \times E \times \{\cdot, \div\}$ to the generic group oracle $\mathcal{O}$. In the sequel we will restrict $\mathcal{R}$ to issue only queries $(e, e', \circ)$ to $\mathcal{O}$ such that $e, e' \in \mathcal{L}^E$. It determines the smallest indices $i$ and $j$ with $e = e_i$ and $e' = e_j$ by calling $(i, j) =$ GETIDX$(e, e')$. Then it computes $\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}}$ and returns the encoding ENCODE$(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$. Furthemore, we require that $\mathcal{R}$ only outputs encodings $\phi(S_i)$ such that $\phi(S_i) \in \mathcal{L}^E$.

*Remark* 10. We note that the above restrictions are without loss of generality. To explain this, recall that

[3] Recall that the same encoding may occur multiple times in $\mathcal{L}^E$.

[4] Note that GETIDX may receive only encodings $e_1, \ldots, e_w$ which are already contained in $\mathcal{L}^E$, as otherwise the behavior of GETIDX is undefined. We will make sure that this is always the case.

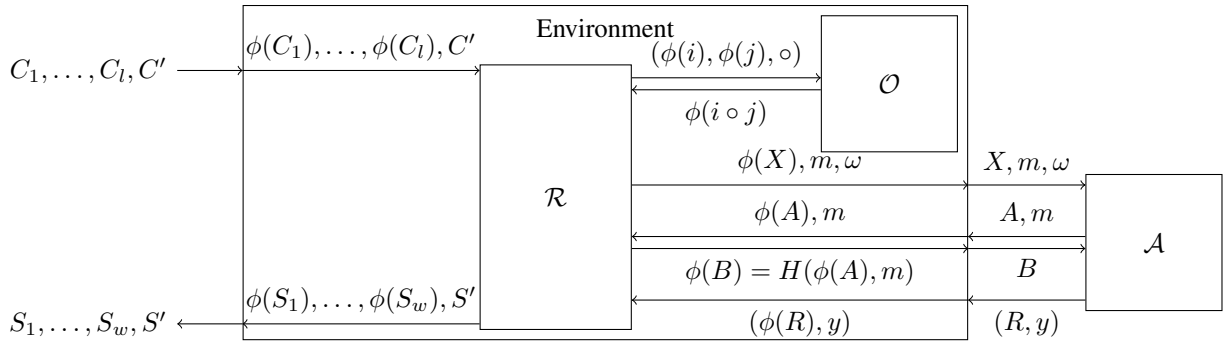


Figure 2: An example of the interaction between a generic reduction $\mathcal{R}$ and a non-generic adversay $\mathcal{A}$ against the unforgeability of Schnorr signatures. All group elements – such as the challenge input, random oracle queries, and the signature output by $\mathcal{A}$ – are encoded by the environment before being passed to $\mathcal{R}$. In the other direction, encodings of group elements output by $\mathcal{R}$ – such as the public key that is the input of $\mathcal{A}$, random oracle responses, and the solution output by $\mathcal{R}$ – are decoded before being passed to the outside world.

the assignment between group elements and encodings is random. An alternative implementation $\mathcal{O}'$ of $\mathcal{O}$ could, given an encoding $e \notin \mathcal{L}^E$, assign a random group element $G \leftarrow_\$ \mathbb{G} \setminus \mathcal{L}^{\mathbb{G}}$ to $e$ by appending $G$ to $\mathcal{L}^{\mathbb{G}}$ and $e$ to $\mathcal{L}^E$, in which case $\mathcal{R}$ would obtain an encoding of an independent, new group element. Of course $\mathcal{R}$ can simulate this behavior easily when interacting with $\mathcal{O}$, too.

**Generic reductions.** Recall that a (fully black-box [21]) reduction from problem $\Pi$ to problem $\Sigma$ is an efficient algorithm $\mathcal{R}$ that solves $\Pi$, having black-box access to an algorithm $\mathcal{A}$ solving $\Sigma$.

In the sequel we consider reductions $\mathcal{R}^{\mathcal{A},\mathcal{O}}$ having black-box access to an algorithm $\mathcal{A}$ as well as to a generic group oracle $\mathcal{O}$. A generic reduction receives as input a challenge $C = (\phi(C_1), \ldots, \phi(C_\ell), C') \in \mathbb{G}^u \times \{0,1\}^*$ consisting of $u$ encoded group elements and a bit-string $C'$. $\mathcal{R}$ may perform computations on encoded group elements, by invoking a generic group oracle $\mathcal{O}$ as described above, and interacts with algorithm $\mathcal{A}$ to compute a solution $S = (\phi(S_1), \ldots, \phi(S_w), S') \in \mathbb{G}^w \times \{0,1\}^*$, which again may consist of encoded group elements $\phi(S_1), \ldots, \phi(S_w)$ and a bit-string $S' \in \{0,1\}^*$. Reductions from an *interactive* computational problem $\Pi$ may additionally have access to an oracle $\mathcal{S}_\Pi$ corresponding to $\Pi$, we write $\mathcal{R}^{\mathcal{A},\mathcal{O},\mathcal{S}_\Pi}$.

We stress that the adversary $\mathcal{A}$ does not necessarily have to be a generic algorithm. It may not be immediately obvious that a generic reduction can make use of a non-generic adversary, considering that $\mathcal{A}$ might expect a particular encoding of the group elements. However, this is indeed possible. In particular, most reductions in security proofs for cryptosystems that are based on algebraic groups (like [20, 6, 28], to name a few well-known examples) are independent of a particular group representation, and thus generic.

Recall that $\mathcal{R}$ is fully blackbox, i.e., $\mathcal{A}$ is external to $\mathcal{R}$. Thus, the environment in which the reduction is run can easily translate between the two encodings. Consider as an example the reduction shown in Figure 2 that interacts with a non-generic adversary $\mathcal{A}$. We stress that the actual algorithm solving the problem $\Pi$, which is a composition of $\mathcal{R}$ and $\mathcal{A}$ is therefore *not* generic.

# 3 Unconditional Tightness Bound for Generic Reductions

In this section, we investigate the possibility of finding a tight *generic* reduction $\mathcal{R}$ that reduces a representation-invariant computational problem $\Pi$ to breaking the UF-NM-security of the Schnorr signature scheme. Our results in this direction are negative, showing that it is impossible to find a generic reduction from any representation-invariant computational problem. This includes even interactive problems.

## 3.1 Single-Instance Reductions

We begin with considering a very simple class of reduction that we call *vanilla reductions*. A vanilla reduction is a reduction that runs the UF-NM forger $\mathcal{A}$ exactly once (without restarting or rewinding) in order to solve the problem $\Pi$. This allows us to explain and analyze the new simulation technique. Later we turn to reductions that may execute $\mathcal{A}$ repeatedly, like for instance the known security proof from [20] based on the Forking Lemma.

**An Inefficient Adversary $\mathcal{A}$** In this section we describe an inefficient adversary $\mathcal{A}$ that breaks the UF-NM-security of the Schnorr signature scheme. Recall that a black-box reduction $\mathcal{R}$ must work for any attacker $\mathcal{A}$. Thus, algorithm $\mathcal{R}^{\mathcal{A}}$ will solve the challenge problem $\Pi$, given black-box access to $\mathcal{A}$. The meta-reduction will be able to simulate this attacker *efficiently* for any generic reduction $\mathcal{R}$. We describe this attacker for comprehensibility, in order to make our meta-reduction more accessible to the reader.

1. The input of $\mathcal{A}$ is a Schnorr public-key $X$, a message $m$, and random coins $\omega \in \{0,1\}^{\kappa}$.
2. The forger $\mathcal{A}$ chooses $q$ uniformly random group elements $R_1, \ldots, R_q \leftarrow_{\$} \mathbb{G}$. (We make the assumption that $q \leq |\mathbb{G}|$.) Subsequently, the forger $\mathcal{A}$ queries the random oracle $\mathcal{H}$ on $(R_i, m)$ for all $i \in [q]$. Let $c_i := \mathcal{H}(R_i, m) \in \mathbb{Z}_p$ be the corresponding answers.
3. Finally, the forger $\mathcal{A}$ chooses an index uniformly at random $\alpha \leftarrow_{\$} [q]$, computes $y \in \mathbb{Z}_p$ which satisfies the equation $g^y = X^{c_\alpha} \cdot R_\alpha$, and outputs $(R_\alpha, y)$. For concreteness, we assume this computation is performed by exhaustive search over all $y \in \mathbb{Z}_p$ (recall that we consider an unbounded attacker here, we show later how to instantiate it efficiently).

Note that $(R_\alpha, y)$ is a valid signature for message $m$ with respect to the public key $X$. Thus, the forger $\mathcal{A}$ breaks the UF-NM-security of the Schnorr signatures with probability 1.

**Main Result for Vanilla Reductions** Now we are ready to prove our main result for vanilla reductions.

**Theorem 11.** *Let $\Pi = (\mathcal{G}_\Pi, \mathcal{S}_\Pi, \mathcal{V}_\Pi)$ be a representation-invariant (possibly interactive) computational problem with a challenge consisting of $u$ group elements and let $p$ be the group order. Suppose there exists a generic vanilla reduction $\mathcal{R}$ that $(\epsilon_\mathcal{R}, t_\mathcal{R})$-solves $\Pi$, having* one-time *black-box access to an attacker $\mathcal{A}$ that $(\epsilon_\mathcal{A}, t_\mathcal{A})$-breaks the* UF-NM*-security of Schnorr signatures with success probability $\epsilon_\mathcal{A} = 1$ by asking $q$ random oracle queries. Then there exists an algorithm $\mathcal{M}$ that $(\epsilon, t)$-solves $\Pi$ with $\epsilon \geq \epsilon_\mathcal{R} - \frac{2(u+q+t_\mathcal{R})^2}{p}$ and $t \approx t_\mathcal{R}$.*

*Remark* 12. Observe that Theorem 11 rules out reductions from nearly arbitrary computational problems (even interactive). At a first glance this might look contradictory, for instance there always exists a trivial reduction from the problem of forging Schnorr signatures to solving the same problem. However, as explained in Example 9, forging Schnorr-signatures is not a representation-invariant computational problem, therefore this is not a contradiction.

PROC $\mathcal{M}_0(C)$

# INITIALIZATION

parse $C = (C_1, \ldots, C_u, C')$

$\mathcal{L}^{\mathbb{G}} := \emptyset \quad ; \quad \mathcal{L}^{E} := \emptyset$

$\vec{R} = (R_1, \ldots, R_q) \leftarrow_{\$} \mathbb{G}^q$

$\mathcal{I} := (C_1, \ldots, C_u, R_1, \ldots, R_q)$

ENCODE$(\mathcal{I})$

$\hat{C} := (\mathcal{L}_1^E, \ldots, \mathcal{L}_u^E, C')$

$\hat{S} \leftarrow_{\$} \mathcal{R}^{\mathcal{O},\mathcal{A}}(\hat{C})$

# FINALIZATION

parse $\hat{S} := (\hat{S}_1, \ldots, \hat{S}_w, S')$

$(i_1, \ldots, i_w) :=$ GETIDX$(\hat{S}_1, \ldots, \hat{S}_w)$

return $(\mathcal{L}_{i_1}^{\mathbb{G}}, \ldots, \mathcal{L}_{i_w}^{\mathbb{G}}, S')$

PROC $\mathcal{A}(\phi(X), m, \omega)$

for all $i \in [q]$

$c_i = \mathcal{R}.\mathcal{H}(\phi(R_i), m)$

$\alpha \leftarrow_{\$} [q]$

$y := \log_g X^{c_\alpha} R_\alpha$

return $(R_\alpha, y)$.

PROC $\mathcal{S}_\Pi{}'(Q)$

parse $Q = (e_1, \ldots, e_v, Q')$

$(i_1, \ldots, i_v) =$ GETIDX$(e_1, \ldots, e_v)$

$(A_1, \ldots, A_\nu, A') = \mathcal{S}_\Pi(\mathcal{L}_{i_1}, \ldots, \mathcal{L}_{i_\nu}, Q')$

$(f_1, \ldots, f_\nu) =$ ENCODE$(A_1, \ldots, A_\nu)$

return $(f_1, \ldots, f_\nu, A')$.

Figure 3: Implementation of $\mathcal{M}_0$.

PROOF. Assume that there exists a generic vanilla reduction $\mathcal{R} := \mathcal{R}^{\mathcal{O},\mathcal{S}'_\Pi,\mathcal{A}}$ that $(\epsilon_\mathcal{R}, t_\mathcal{R})$-solves $\Pi$, when given access to a generic group oracle $\mathcal{O}$, an oracle $\mathcal{S}'_\Pi$, and a forger $\mathcal{A}(\phi(X), m, \omega)$, where the inputs to the forger are chosen by $\mathcal{R}$. Furthermore, the reduction $\mathcal{R}$ simulates the random oracle $\mathcal{R}.\mathcal{H}$ for $\mathcal{A}$. We show how to build a meta-reduction $\mathcal{M}$ that has black-box access to $\mathcal{R}$ and to an oracle $\mathcal{S}_\Pi$ and that solves the representation-invariant problem $\Pi$ directly.

We describe $\mathcal{M}$ in a sequence of games, beginning with an *inefficient* implementation $\mathcal{M}_0$ of $\mathcal{M}$ and we modify it gradually until we obtain an *efficient* implementation $\mathcal{M}_2$ of $\mathcal{M}$. We bound the probability with which any reduction $\mathcal{R}$ can distinguish each implementation $\mathcal{M}_i$ from $\mathcal{M}_{i-1}$ for all $i \in \{1, 2\}$, which yields that $\mathcal{M}_2$ is an efficient algorithm that can use $\mathcal{R}$ to solve $\Pi$ if $\mathcal{R}$ in tight.

In what follows let $X_i$ denote the event that $\mathcal{R}$ outputs a valid solution to the given problem instance $\hat{C}$ of $\Pi$ in Game $i$.

**Game 0.** Our meta-reduction $\mathcal{M}_0 := \mathcal{M}_0^{\mathcal{S}_\Pi}$ is an algorithm for solving a representation-invariant computational problem $\Pi$, as defined in Section 2.3. That is, $\mathcal{M}_0$ takes as input an instance $C = (C_1, \ldots, C_u, C') \in \mathbb{G}^u \times \{0,1\}^*$, of the representation-invariant computational problem $\Pi$, has access to oracle $\mathcal{S}_\Pi$ provided by $\Pi$, and outputs a candidate solution $S$. $\mathcal{R}$ is a generic reduction, i.e., a representation-independent algorithm for $\Pi$ having black-box access to an attacker $\mathcal{A}$. Algorithm $\mathcal{M}_0$ runs reduction $\mathcal{R}$ as a subroutine, by simulating the generic group oracle $\mathcal{O}$, the $\mathcal{S}_\Pi$ oracle, and attacker $\mathcal{A}$ for $\mathcal{R}$. In order to provide the generic group oracle for $\mathcal{R}$, $\mathcal{M}_0$ implements the following procedures (cf. Figure 3).

INITIALIZATION OF $\mathcal{M}_0$: At the beginning of the game, $\mathcal{M}_0$ initializes two lists $\mathcal{L}^{\mathbb{G}} := \emptyset$ and $\mathcal{L}^{E} := \emptyset$, which are used to simulate the generic group oracle $\mathcal{O}$. Furthermore, $\mathcal{M}_0$ chooses $\vec{R} = (R_1, \ldots, R_q) \leftarrow_{\$} \mathbb{G}^q$ at random (these values will later be used by the simulated attacker $\mathcal{A}$), sets $\mathcal{I} := (C_1, \ldots, C_u, R_1, \ldots, R_q)$, and runs ENCODE$(\mathcal{I})$ to assign encodings to these group elements. Then $\mathcal{M}_0$ starts the reduction $\mathcal{R}$ on input $\hat{C} := (\mathcal{L}_1^E, \ldots, \mathcal{L}_u^E, C')$. Note that $\hat{C}$ is an encoded version of the challenge instance of $\Pi$

received by $\mathcal{M}_0$. That is, we have $\hat{C} = (\phi(C_1), \ldots, \phi(C_u), C')$. Oracle queries of $\mathcal{R}$ are answered by $\mathcal{M}_0$ as follows:

GENERIC GROUP ORACLE $\mathcal{O}(e, e', \circ)$: To simulate the generic group oracle, $\mathcal{M}_0$ implements procedures ENCODE and GETIDX as described in Section 2.4. Whenever $\mathcal{R}$ submits a query $(e, e', \circ) \in E \times E \times \{\cdot, \div\}$ to the generic group oracle $\mathcal{O}$, the meta-reduction determines the smallest indices $i$ and $j$ such that $e = e_i$ and $e' = e_j$ by calling $(i, j) = \text{GETIDX}(e, e')$. Then it computes $\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}}$ and returns $\text{ENCODE}(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$.

ORACLE $\mathcal{S}'_\Pi(Q)$: This procedure handles queries issued by $\mathcal{R}$ to $\mathcal{S}'_\Pi$ by forwarding them to oracle $\mathcal{S}_\Pi$ provided by the challenger and returning the response. That is, whenever $\mathcal{R}$ submits a query $Q = (e_1, \ldots, e_v, Q') \in E^v \times \{0,1\}^*$ to $\mathcal{S}'_\Pi$, the meta-reduction runs $(i_1, \ldots, i_v) := \text{GETIDX}(e_1, \ldots, e_v)$ and queries $\mathcal{S}_\Pi$ to compute $(A_1, \ldots, A_\nu, A') := \mathcal{S}_\Pi(\mathcal{L}_{i_1}, \ldots, \mathcal{L}_{i_\nu}, Q')$. Then $\mathcal{M}_0$ determines the corresponding encodings as $(f_1, \ldots, f_\nu) := \text{ENCODE}(A_1, \ldots, A_\nu)$ and returns $(f_1, \ldots, f_\nu, A')$ to $\mathcal{R}$.

THE FORGER $\mathcal{A}(\phi(X), m, \omega)$: This procedure implements a simulation of the inefficient attacker $\mathcal{A}$ described in Section 3.1. It proceeds as follows. When $\mathcal{R}$ outputs $(\phi(X), m, \omega)$ to invoke an instance of $\mathcal{A}$, $\mathcal{A}$ queries the random oracle $\mathcal{R}.\mathcal{H}$ provided by $\mathcal{R}$ on $(\phi(R_i), m)$ for all $i \in [q]$, to determine $c_i = \mathcal{H}(\phi(R_i), m)$. Afterwards, $\mathcal{M}_0$ chooses an index $\alpha \leftarrow_\$ [q]$ uniformly at random, computes the the discrete logarithm $y := \log_g X^{c_\alpha} R_\alpha$ by exhaustive search, and outputs $(R_\alpha, y)$. (This step is not efficient. We show in subsequent games how to implement this attacker efficiently.)

FINALIZATION OF $\mathcal{M}_0$: Eventually, the algorithm $\mathcal{R}$ outputs a solution $\hat{S} := (\hat{S}_1, \ldots, \hat{S}_w, S') \in E^w \times \{0,1\}^*$. The algorithm $\mathcal{M}_0$ runs $(i_1, \ldots, i_w) := \text{GETIDX}(\hat{S}_1, \ldots, \hat{S}_w)$ to determine the indices of group elements $(\mathcal{L}_{i_1}^{\mathbb{G}}, \ldots, \mathcal{L}_{i_w}^{\mathbb{G}})$ corresponding to encodings $(\hat{S}_1, \ldots, \hat{S}_w)$, and outputs $(\mathcal{L}_{i_1}^{\mathbb{G}}, \ldots, \mathcal{L}_{i_w}^{\mathbb{G}}, S')$.

**Analysis of $\mathcal{M}_0$.** Note that $\mathcal{M}_0$ provides a perfect simulation of the oracles $\mathcal{O}$ and $\mathcal{S}_\Pi$ and it also mimics the attacker from Section 3.1 perfectly. In particular, $(R_\alpha, y)$ is a valid forgery for message $m$ and thus, $\mathcal{R}$ outputs a solution $\hat{S} = (\hat{S}_1, \ldots, \hat{S}_w, S')$ to $\hat{C}$ with probability $\Pr[X_0] = \epsilon_{\mathcal{R}}$. Since $\Pi$ is assumed to be representation-invariant, $S := (S_1, \ldots, S_w, S')$ with $\hat{S}_i = \phi(S_i)$ for $i \in [w]$ is therefore a valid solution to $C$. Thus, $\mathcal{M}_0$ outputs a valid solution $S$ to $C$ with probability $\epsilon_{\mathcal{R}}$.

**Game 1.** In this game we introduce a meta-reduction $\mathcal{M}_1$, which essentially extends $\mathcal{M}_0$ with additional bookkeeping to record the sequence of group operations performed by $\mathcal{R}$. The purpose of this intermediate game is to simplify our analysis of the final implementation $\mathcal{M}_2$. Meta-reduction $\mathcal{M}_1$ proceeds identical to $\mathcal{M}_0$, except for a few differences (cf. Figure 4).

INITIALIZATION OF $\mathcal{M}_1$: The initialization is exactly like before, except that $\mathcal{M}_1$ maintains an additional list $\mathcal{L}^V$ of elements of $\mathbb{Z}_p^{u+q}$. Let $\mathcal{L}_i^V$ denote the $i$-th entry of $\mathcal{L}^V$.

List $\mathcal{L}^V$ is initialized with the $u + q$ canonical unit vectors in $\mathbb{Z}_p^{u+q}$. That is, let $\eta_i$ denote the $i$-th canonical unit vector in $\mathbb{Z}_p^{u+q}$, i.e., $\eta_1 = (1, 0, \ldots, 0), \eta_2 = (0, 1, 0, \ldots, 0), \ldots, \eta_{u+q} = (0, \ldots, 0, 1)$. Then $\mathcal{L}^V$ is initialized such that $\mathcal{L}_i^V := \eta_i$ for all $i \in [u + q]$.

GENERIC GROUP ORACLE $\mathcal{O}(e, e', \circ)$: In parallel to computing the group operation, the generic group oracle implemented by $\mathcal{M}_1$ also performs computations on vectors of $\mathcal{L}^V$.

Given a query $(e, e', \circ) \in E \times E \times \{\cdot, \div\}$, the oracle $\mathcal{O}$ determines the smallest indices $i$ and $j$ such that $e = e_i$ and $e' = e_j$ by calling GETIDX. It computes $a := \mathcal{L}_i^V \diamond \mathcal{L}_j^V \in \mathbb{Z}_p^{u+q}$, where $\diamond := +$ if $\circ = \cdot$ and $\diamond := -$ if $\circ = \div$, and appends $a$ to $\mathcal{L}^V$. Finally it returns $\text{ENCODE}(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$.

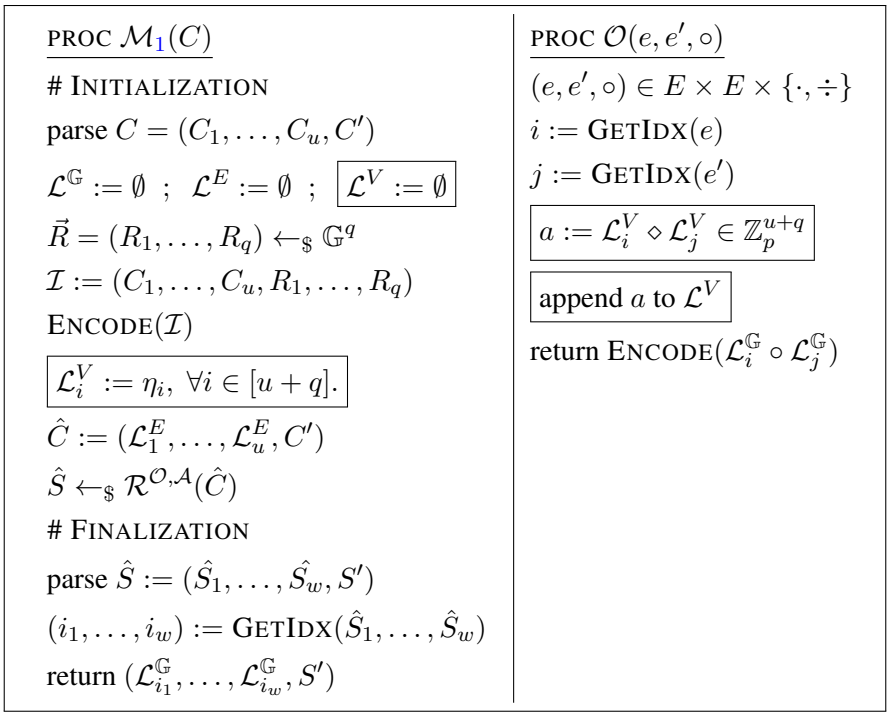


Figure 4: Meta-Reduction $\mathcal{M}_1$. Boxed elements show the differences to $\mathcal{M}_0$. All other procedures are identical to $\mathcal{M}_0$ and thus omitted.

**Analysis of $\mathcal{M}_1$.** Recall that the initial content $\mathcal{I}$ of $\mathcal{L}^{\mathbb{G}}$ is $\mathcal{I} = (C_1, \ldots, C_u, R_1, \ldots, R_q)$, and that $\mathcal{R}$ performs only group operations on $\mathcal{I}$. Thus, any group element $h \in \mathcal{L}^{\mathbb{G}}$ can be written as $h = \prod_{i=1}^{u} C_i^{a_i} \cdot \prod_{i=1}^{q} R_i^{a_{u+i}}$ where the vector $a = (a_1, \ldots, a_{u+q}) \in \mathbb{Z}_p^{u+q}$ is (essentially) determined by the sequence of queries issued by $\mathcal{R}$ to $\mathcal{O}$. For a vector $a \in \mathbb{Z}_p^{u+q}$ and a vector of group elements $V = (v_1, \ldots, v_{u+q}) \in \mathbb{G}^{u+q}$ let us write $\mathsf{Eval}(V, a)$ shorthand for $\mathsf{Eval}(V, a) := \prod_{i=1}^{u+q} v_i^{a_i}$ in the sequel. In particular, it holds that $\mathsf{Eval}(\mathcal{I}, a) = \prod_{i=1}^{u} C_i^{a_i} \cdot \prod_{i=1}^{q} R_i^{a_{u+i}}$. The key motivation for the changes introduced in Game 1 is that now (by construction of $\mathcal{M}_1$) it holds that $\mathcal{L}_i^{\mathbb{G}} = \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V)$ for all $i \in [|\mathcal{L}^{\mathbb{G}}|]$. Thus, at any point in time during the execution of $\mathcal{R}$, the entire list $\mathcal{L}^{\mathbb{G}}$ of group elements can be recomputed from $\mathcal{L}^V$ and $\mathcal{I}$ by setting $\mathcal{L}_i^{\mathbb{G}} := \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V)$ for $i \in [|\mathcal{L}^V|]$. The reduction $\mathcal{R}$ is completely oblivious to this additional bookkeeping performed by $\mathcal{M}_1$, thus we have $\Pr[X_1] = \Pr[X_0]$.

**Game 2.** Note that the meta-reductions described in previous games were not efficient, because the simulation of the attacker in procedure $\mathcal{A}$ needed to compute a discrete logarithm by exhaustive search. In this final game, we construct a meta-reduction $\mathcal{M}_2$ that simulates $\mathcal{A}$ efficiently. $\mathcal{M}_2$ proceeds exactly like $\mathcal{M}_1$, except for the following (cf. Figure 5).

THE FORGER $\mathcal{A}(\phi(X), m, \omega)$: When $\mathcal{R}$ outputs $(\phi(X), m, \omega)$ to invoke an instance of $\mathcal{A}$, $\mathcal{A}$ queries the random oracle $\mathcal{R}.\mathcal{H}$ provided by $\mathcal{R}$ on $(\phi(R_i), m)$ for all $i \in [q]$, to determine $c_i = \mathcal{H}(\phi(R_i), m)$. Then it chooses an index $\alpha \leftarrow_{\$} [q]$ uniformly at random, samples an element $y$ uniformly at random from $\mathbb{Z}_p$, computes $R_\alpha^* := g^y X^{-c_\alpha}$, and re-computes the entire list $\mathcal{L}^{\mathbb{G}}$ using $R_\alpha^*$ instead of $R_\alpha$.

More precisely, let $\mathcal{I}^* := (C_1, \ldots, C_u, R_1, \ldots, R_{\alpha-1}, R_\alpha^*, R_{\alpha+1}, \ldots, R_q)$. Observe that the vector $\mathcal{I}^*$ is identical to the initial contents $\mathcal{I}$ of $\mathcal{L}^{\mathbb{G}}$, with the difference that $R_\alpha$ is replaced by $R_\alpha^*$. The list $\mathcal{L}^{\mathbb{G}}$ is

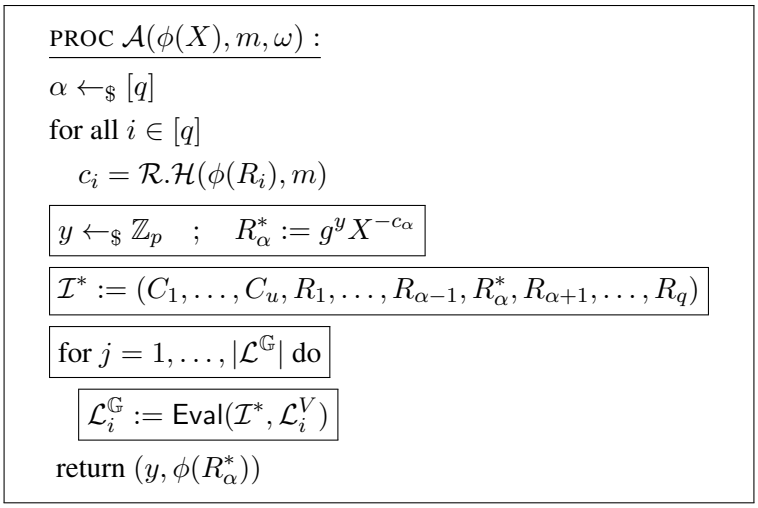


Figure 5: Efficient simulation of attacker $\mathcal{A}$ by $\mathcal{M}_2$.

now recomputed from $\mathcal{L}^V$ and $\mathcal{I}^*$ by setting $\mathcal{L}^{\mathbb{G}}_i := \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i)$ for all $i \in [|\mathcal{L}^V|]$. Finally, $\mathcal{M}_2$ returns $(\phi(R^*_\alpha), y)$ to $\mathcal{R}$ as the forgery.

**Analysis of $\mathcal{M}_2$.** First note that $(\phi(R^*_\alpha), y)$ is a valid signature, since $\phi(R^*_\alpha)$ is the encoding of group element $R^*_\alpha$ satisfying the verification equation $g^y = X^{c_\alpha} \cdot R^*_\alpha$, where $c_\alpha = \mathcal{H}(\phi(R^*_\alpha), m)$. Next we claim that $\mathcal{R}$ is not able to distinguish $\mathcal{M}_2$ from $\mathcal{M}_1$, except for a negligibly small probability. To show this, observe that Game 2 and Game 1 are perfectly indistinguishable, if for all pairs of vectors $\mathcal{L}^V_i, \mathcal{L}^V_j \in \mathcal{L}^V$ it holds that $\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_j) \iff \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_j)$, because in this case $\mathcal{M}_2$ chooses identical encodings for two group elements $\mathcal{L}^{\mathbb{G}}_i, \mathcal{L}^{\mathbb{G}}_j \in \mathcal{L}^{\mathbb{G}}$ *if and only if* $\mathcal{M}_1$ chooses identical encodings.

**Lemma 13.** *Let $F$ denote the event that $\mathcal{R}$ computes vectors $\mathcal{L}^V_i, \mathcal{L}^V_j \in \mathcal{L}^V$ such that*

$$\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_j) \quad \wedge \quad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i) \neq \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_j) \tag{1}$$

*or*

$$\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_i) \neq \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_j) \quad \wedge \quad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_j). \tag{2}$$

*Then*

$$\Pr[F] \leq 2(u + q + t_{\mathcal{R}})^2/p.$$

The proof of Lemma 13 is deferred to Section 3.2. We apply it to finish the proof of Theorem 11. By Lemma 13, algorithm $\mathcal{M}_2$ fails to simulate $\mathcal{M}_1$ with probability at most $2(u + q + t_{\mathcal{R}})^2/p$. Thus, we have $\Pr[X_2] \geq \Pr[X_1] - 2(u + q + t_{\mathcal{R}})^2/p$.

Note also that $\mathcal{M}_2$ provides an efficient simulation of adversary $\mathcal{A}$. The total running time of $\mathcal{M}_2$ is essentially of the running time of $\mathcal{R}$ plus some minor additional computations and bookkeeping. Furthermore, if $\mathcal{R}$ is able to $(\epsilon_{\mathcal{R}}, t_{\mathcal{R}})$ solve $\Pi$, then $\mathcal{M}_2$ is able to $(\epsilon, t)$-solve $\Pi$ with probability at least

$$\epsilon \geq \Pr[X_2] \geq \epsilon_{\mathcal{R}} - \frac{2(u + q + t_{\mathcal{R}})^2}{p}.$$

□

### 3.2 Proof of Lemma 13

The proof of this lemma is based on the observation that an algorithm that performs only a (polynomially) limited number of group operations in an (exponential-size) generic group is very unlikely to find any "non-trivial relation" among random group elements. This technique was introduced in [27] in a different setting, to analyze the complexity of algorithms for the discrete logarithm problem.

AN ALTERNATIVE FORMULATION OF EVENT $F$. Recall that the vectors $\mathcal{I}$ and $\mathcal{I}^*$ differ only in their $\alpha$-th component. In the sequel let us write $\mathcal{I}_\alpha$ to denote the vector $\mathcal{I}$, but with its $\alpha$-th component $R_\alpha$ set equal to $1 \in \mathbb{G}$. That is,

$$\mathcal{I}_\alpha := (R_1, \ldots, R_{\alpha-1}, 1, R_{\alpha+1}, \ldots, R_q, g_1, \ldots, g_u).$$

Then we have

$$\mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V) = \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V) \cdot R_\alpha^{\mathcal{L}_{i,\alpha}^V} \qquad \text{and} \qquad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}_i^V) = \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V) \cdot (R_\alpha^*)^{\mathcal{L}_{i,\alpha}^V}$$

where $\mathcal{L}_{i,\alpha}^V$ denotes the $\alpha$-th component of vector $\mathcal{L}_i^V$. In particular, for any two vectors $\mathcal{L}_i^V, \mathcal{L}_j^V$ we have

$$\begin{aligned} \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V) = \mathsf{Eval}(\mathcal{I}, \mathcal{L}_j^V) &\iff \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V) \cdot R_\alpha^{\mathcal{L}_{i,\alpha}^V} = \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_j^V) \cdot R_\alpha^{\mathcal{L}_{j,\alpha}^V} \\ &\iff \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V - \mathcal{L}_j^V) \cdot R_\alpha^{\mathcal{L}_{i,\alpha}^V - \mathcal{L}_{j,\alpha}^V} = 1 \end{aligned}$$

Thus, (1) is equivalent to

$$\mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V - \mathcal{L}_j^V) \cdot R_\alpha^{\mathcal{L}_{i,\alpha}^V - \mathcal{L}_{j,\alpha}^V} = 1 \quad \wedge \quad \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V - \mathcal{L}_j^V) \cdot (R_\alpha^*)^{\mathcal{L}_{i,\alpha}^V - \mathcal{L}_{j,\alpha}^V} \neq 1 \tag{3}$$

If we take discrete logarithms to base $\gamma \in \mathbb{G}$, where $\gamma$ is an arbitrary generator of $\mathbb{G}$, and define the degree-one polynomial $\Delta_{i,j,\alpha} \in \mathbb{Z}_p[X]$ as

$$\Delta_{i,j} := \log \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V - \mathcal{L}_j^V) + X \cdot \left(\mathcal{L}_{i,\alpha}^V - \mathcal{L}_{j,\alpha}^V\right),$$

then (3) (and therefore also (1)) is in turn equivalent to

$$\Delta_{i,j}(\log R_\alpha) \equiv 0 \bmod p \qquad \wedge \qquad \Delta_{i,j}(\log R_\alpha^*) \not\equiv 0 \bmod p. \tag{4}$$

Similarly, (2) is equivalent to

$$\Delta_{i,j}(\log R_\alpha) \not\equiv 0 \bmod p \qquad \wedge \qquad \Delta_{i,j}(\log R_\alpha^*) \equiv 0 \bmod p. \tag{5}$$

Thus, event $F$ occurs if $\mathcal{R}$ computes vectors $\mathcal{L}_i^V, \mathcal{L}_j^V$ such that either (4) or (5) holds.

FAILURE EVENT $F_1$. Let $F_1$ denote the event that (4) holds. Note that this can only happen if $\mathcal{R}$ performs a sequence of computations, such that there exist a pair $(i,j) \in [|\mathcal{L}^V|] \times [|\mathcal{L}^V|]$ such that the polynomial $\Delta_{i,j}$ is not the zero-polynomial in $\mathbb{Z}_p[X]$, but it holds that $\Delta_{i,j}(R_\alpha) \equiv 0 \bmod p$.

At the beginning of the game $\mathcal{R}$ receives only a random encoding $\phi(R_\alpha)$ of group element $R_\alpha$. The only further information that $\mathcal{R}$ learns about $R_\alpha$ throughout the game is through equality or inequality of encodings. Since $\mathcal{R}$ runs in time $t_\mathcal{R}$, it can issue at most $t_\mathcal{R}$ oracle queries. Thus, at the end of the game the list $\mathcal{L}^V$ contains at most $|\mathcal{L}^V| \leq t_\mathcal{R} + q + u$ entries. Each pair $(i,j) \in [|\mathcal{L}^V|]$ with $i \neq j$ defines a (possibly non-zero) polynomial $\Delta_{i,j}$. In total there are at most $(t_\mathcal{R} + q + u) \cdot (t_\mathcal{R} + q + u - 1) \leq (t_\mathcal{R} + q + u)^2$ such polynomials.

Since all polynomials have degree one, and $\log R_\alpha$ is uniformly distributed over $\mathbb{Z}_p$ (because $R_\alpha$ is uniformly random over $\mathbb{G}$), the probability that $\log R_\alpha$ is a root of any of these polynomials is upper bounded by

$$\Pr[F_1] \leq \frac{(u + q + t_\mathcal{R})^2}{p}.$$

FAILURE EVENT $F_2$. Let $F_2$ denote the event that (5) holds. Since $\log R^*_\alpha$ is uniformly distributed over $\mathbb{Z}_p$ (because we have defined $R^*_\alpha := g^y X^{-c}$ for uniformly $y \leftarrow_\$ \mathbb{Z}_p$), with similar arguments as before we have

$$\Pr[F_2] \leq \frac{(u + q + t_\mathcal{R})^2}{p}.$$

BOUNDING $\Pr[F]$. Since $F = F_1 \cup F_2$ we have

$$\Pr[F] \leq \Pr[F_1] + \Pr[F_2] \leq \frac{2(u + q + t_\mathcal{R})^2}{p}.$$

# 4 Multi-Instance Reductions

Now we turn to considering multi-instance reductions, which may run multiple sequential executions of the signature forger $\mathcal{A}$. This is the interesting case, in particular because the Forking-Lemma based security proof for Schnorr signatures by Pointcheval and Stern [20] is of this type.

Again we construct a meta-reduction with simulated adversary. The main difference to our single-instance adversary is that it does not succeed with probability 1, but tosses a biased coin that decides if it forges for the message or not. On the first glance this approach might seem to be of little value, because an adversary with a higher success probability should improve the success probability of the reduction. However, it was shown in [26] that, once we consider a reduction that runs multiple sequential executions of this adversary, this approach allows to derive an optimal tightness bound.

In the following we assume that the reduction $\mathcal{R}$ executes $n$ sequential instances of the same adversary $\mathcal{A}(\phi(X), m, \omega)$, where the public key $\phi(X)$, the message $m$, and the randomness $\omega$ of each instance are chosen by $\mathcal{R}$. Observe that the input to the adversary and the random oracle query/answers completely determine the behaviour of the adversary. Thus, any successive execution of an instance of $\mathcal{A}$ may be identical to a previous execution up to a certain point, where the response $c = \mathcal{H}(R, m)$ of the random oracle differs from a response $c' = \mathcal{H}(R, m)$ received by $\mathcal{A}$ in a previous execution. This point is called the *forking point* [26].

### 4.1 A Family of Inefficient Adversaries $\mathcal{A}_{F,f}$

In this section, we describe a different inefficient adversary $\mathcal{A}$ against the UF-NM-security of the Schnorr signature scheme. In fact, we do not describe a single adversary but a family of adversaries from which the meta-reduction will choose one to simulate at random.

To define this family, we fix the following notations. The Bernoulli distribution of a parameter $\mu \in [0,1]$ is defined by $\mathtt{Ber}_\mu$, i.e., $\Pr[\delta = 1] = \mu$ and $\Pr[\delta = 0] = 1 - \mu$. Let $\mathcal{Q} = \mathbb{G} \times \mathbb{Z}_p$ be the set of possible random oracle queries and answers. By $\mathbb{S}_i = \mathcal{Q}^i$ we denote the set of random oracle query sequences of length $i$ and the set of all possible sequences is defined as $\mathbb{S} = \bigcup_{i=1}^{q} \mathbb{S}_i$. Consider now the set $\mathbb{F}$ of all functions $F : \{0,1\}^k \times \mathbb{G} \times \{0,1\}^\kappa \times \mathbb{S} \to \mathbb{G}$. And the set $\mathbb{E}$ of functions $f : \mathbb{G} \to \{0,1\}$ for which the following holds $\Pr[f(g) = 1 | g \leftarrow_\$ \mathbb{G}] = \Pr[b = 1 | b \leftarrow_\$ \mathtt{Ber}_\mu]$. For each pair $(F, f) \in \mathbb{F} \times \mathbb{E}$ we define the adversary $\mathcal{A}_{F,f}$ as follows:

1. The input of $\mathcal{A}$ is a Schnorr public-key $X$, a message $m$, and random coins $\omega \in \{0,1\}^\kappa$.
2. The forger $\mathcal{A}$ sets $\sigma := \bot$ and performs the following computations. For $i = 1, \ldots, q$ it computes $R_i := F(m, X, \omega, (R_1, c_1), \ldots, (R_{i-1}, c_{i-1}))$ and queries the random oracle $\mathcal{H}$ on $(R_i, m)$, where $c_i := \mathcal{H}(R_i, m) \in \mathbb{Z}_p$ is the corresponding answers. If $\sigma = \bot$, then $\mathcal{A}_{F,f}$ sets $Z_i := X^{c_i} R_i$ and checks if $f(Z_i) = 1$. If this is the case, then $\mathcal{A}_{F,f}$ computes $y_i \in \mathbb{Z}_p$ satisfying the equation $g^{y_i} = X^{c_i} \cdot R_i$ by exhaustive search and sets $\sigma := (R_i, y_i)$. Otherwise, if $f(Z_i) = 0$, then it continues with the loop.
3. Finally, the forger $\mathcal{A}_{F,f}$ returns $\sigma$.

Note that $(R_i, y_i)$ is a valid signature for message $m$ with respect to the public key $X$. Thus, the forger $\mathcal{A}_{F,f}$ breaks the UF-NM-security of the Schnorr signatures whenever $f(Z_i) = 1$ for at least one $i \in [q]$. This translates to a success probability of $\epsilon_{\mathcal{A}} = 1 - (1 - \mu)^q$.

Observe that defining the adversaries as above ensures that, while different instances of the same adversary will behave identically as long as their input and the answers of the random oracle are the same, as soon as one of the inputs or one of the random oracle answers differ the behavior of two instances will be independent of one another from that point onwards. As such, the behavior of these adversaries mimics closely the idea behind the forking lemma and it allows us to easily simulate the adversary in our meta-reduction below.

### 4.2 Main Result for Multi-Instance Reductions

In this section, we combine the approach of Seurin [26] with our simulation of signature forgeries based on re-programming of the group representation, as introduced in Section 3.1. This allows to prove a nearly optimal unconditional tightness bound for all generic reductions and any representation-invariant computational problem $\Pi$.

Unfortunately, the combination of the elaborate techniques of Seurin [26] with our approach yields a rather complex meta-reduction. We stress that we follow Seurin's work as closely as possible. The main difference lies in the way how signature forgeries are computed, namely in our case by exploiting the properties of the generic group representation, instead of using an OMDL-oracle as in [26].

The main difference between the meta-reduction described in this section and the one presented in Section 3.1 lies in the simulation of the Random Oracle queries issued by the adversary in different sequential executions. In particular, the meta-reduction $\mathcal{M} := \mathcal{M}^{\mathcal{S}_\Pi}$ simulates the oracles procedures (resp. oracle) ENCODE, GETIDX, $\mathcal{O}$, and $\mathcal{S}'_\Pi$ exactly as before.

**Theorem ??.** *Let $\Pi$ be a representation-invariant computational problem. Suppose there exists a generic*

*reduction* $\mathcal{R}^{\mathcal{O},\mathcal{S}'_\Pi,\mathcal{A}_{F,f}}$ *that* $(\epsilon_\mathcal{R}, t_\mathcal{R})$*-solves* $\Pi$*, having* $n$-time *black-box access to an attacker* $\mathcal{A}_{F,f}$ *that* $(\epsilon_\mathcal{A}, t_\mathcal{A}, q)$*-breaks the* UF-NM*-security of Schnorr signatures with success probability* $\epsilon_\mathcal{A} = 1-(1-\mu)^q$ *in time* $t_\mathcal{A} \approx q$*. Then there exists an algorithm* $\mathcal{M}$ *that* $(\epsilon, t)$*-solves* $\Pi$ *with*

$$\epsilon \geq \epsilon_\mathcal{R} - \frac{2n(u+nq+t_\mathcal{R})}{p} - \frac{n\ln\left((1-\epsilon_\mathcal{A})^{-1}\right)}{q(1-p^{-1/4})}.$$

PROOF. Suppose that there exists a generic reduction $\mathcal{R} := \mathcal{R}^{\mathcal{O},\mathcal{S}'_\Pi,\mathcal{A}}$ that $(\epsilon_\mathcal{R}, t_\mathcal{R})$-solves $\Pi$, when given access to a generic group oracle $\mathcal{O}$, an oracle $\mathcal{S}'_\Pi$, and to $n$ instances of the same forger $\mathcal{A}(\phi(X), m, \omega)$, where the inputs to each instance of the forger are chosen by $\mathcal{R}$. As before, the random oracle $\mathcal{R}.\mathcal{H}$ for $\mathcal{A}$ is provided by $\mathcal{R}$. We show how to build a meta-reduction $\mathcal{M}$ that has black-box access to $\mathcal{R}$ and to an oracle $\mathcal{S}_\Pi$ and that solves the representation-invariant problem $\Pi$ directly. Again we proceed in a sequence of games, and denote with $\mathcal{M}_i$ the implementation of algorithm $\mathcal{M}$ in Game $i$, and with $X_i$ the event that $\mathcal{R}$ outputs a valid solution $S$ to $C$ in Game $i$. As in Section 3.1, we will bound the probability with which any efficient reduction $\mathcal{R}$ can distinguish each implementation $\mathcal{M}_i$ from $\mathcal{M}_{i-1}$ for all $i \in \{1,2,3\}$. We start with an inefficient implementation $\mathcal{M}_0$ of $\mathcal{M}$, and modify this implementation gradually until we obtain an efficient algorithm $\mathcal{M}_3$ that uses $\mathcal{R}$ to solve $\Pi$.

**Game 0.** $\mathcal{M}_0 := \mathcal{M}_0^{\mathcal{S}_\Pi}$ (cf. Figure 6) takes as input an instance $C = (C_1, \ldots, C_u, C') \in \mathbb{G}^u \times \{0,1\}^*$ of the representation-invariant computational problem $\Pi$, it has access to oracle $\mathcal{S}_\Pi$ provided by $\Pi$, and outputs a candidate solution $S$. It also maintains the encoding of the group using two lists $\mathcal{L}^\mathbb{G} \subseteq \mathbb{G}$ and $\mathcal{L}^E \subseteq E$. Our first instance $\mathcal{M}_0$ perfectly simulates one adversary chosen from the family of adversaries described above chosen at random. The only difference between the real and the simulated adversary is that the meta-reduction does not fix the functions $F, f$ at the beginning but instead defines them on the fly.

INITIALIZATION OF $\mathcal{M}_0$: At the beginning of the game, $\mathcal{M}_0$ chooses $\vec{R} = (R_{1,1}, \ldots, R_{n,q}) \leftarrow_\$ \mathbb{G}^{nq}$ at random (these are the values the function $F$ will be lazily programmed to evaluate to), sets $\mathcal{I} := (C_1, \ldots, C_u, R_{1,1}, \ldots, R_{n,q})$, and runs ENCODE$(\mathcal{I})$ to assign encodings to these group elements. Furthermore, $\mathcal{M}_0$ initializes lists $\mathcal{T}$, $\Gamma_{\text{good}}$, $\Gamma_{\text{bad}}$, and $\mathcal{D}$ as empty lists. Recall that $\mathcal{R}$ executes $n$ sequential instances of the simulated adversary $\mathcal{A}$ and that depending on the input and the query/answer pairs to $\mathcal{R}.\mathcal{H}$, the successive execution might be identical to a certain point. The list $\mathcal{T}$ will be used to store the inputs and query answer pairs of each adversary to ensure consistency of $F$ across adversary instances. Note further that the simulated adversary tosses a biased coin and decides whether it forges a signature or not. The lists $\Gamma_{\text{good}}$ and $\Gamma_{\text{bad}}$ are used to store these decisions whether for a given $Z$, the simulated adversary $\mathcal{A}_{F,f}$ will forge a signature or not. Again, they are used to ensure consistency of $f$ across adversary instances. Accordingly $\Gamma_{\text{good}}$ contains exactly those elements for which $\mathcal{M}_0$ knows the discrete logarithms and $\Gamma_{\text{bad}}$ contains exactly those elements for which it will never compute the discrete logarithms. Finally, $\mathcal{D}$ is used to store known discrete logarithms. Then, $\mathcal{M}_0$ runs a black-box simulation of the reduction $\mathcal{R}$ on input $\hat{C} := (\mathcal{L}_1^E, \ldots, \mathcal{L}_u^E, C')$. Note that $\hat{C}$ is an encoded version of the challenge instance of $\Pi$ received by $\mathcal{M}_0$. That is, we have $\hat{C} = (\phi(C_1), \ldots, \phi(C_u), C')$. Oracle queries of $\mathcal{R} = \mathcal{R}^{\mathcal{O},\mathcal{S}'_\Pi,\mathcal{A}}$ are answered exactly as described in Section 3.1, with the difference being the forger that we describe in the following.

THE FORGER $\mathcal{A}(\phi(X), m, \omega)$: The simulation of the forger $\mathcal{A}$ is rather technical, because $\mathcal{M}_0$ has to provide a consistent simulation of the $n$ sequential executions of $\mathcal{A}$. As already discussed at the beginning of this chapter, $\mathcal{M}_0$ has to emulate an identical behavior of $\mathcal{A}$ up to the forking point, or the reduction might loose its advantage. We split this algorithm up into several sup-procedures (see Figure 6). The main subprocedures are BEFOREFORK and AFTERFORK, with the idea that $\mathcal{A}$ runs the code of BEFOREFORK if the

forking point has not been reached yet and the simulation must be consistent with a previous execution. The second procedure, AFTERFORK describes how $\mathcal{M}_0$ simulates $\mathcal{A}$ after the forking point.

Now we proceed with the technical description of the main procedure of $\mathcal{A}$ and explain the sub-procedures in the following. When $\mathcal{R}$ outputs $(\phi(X), m, \omega)$ to invoke an instance of $\mathcal{A}_{F,f}$, then $\mathcal{M}_0$'s simulation of $\mathcal{A}_{F,f}$ initializes the list $\tau$ with its input $(\phi(X), m, \omega)$ and the forgery $\sigma$ it will output with $\perp$. These inputs are part of the function $F$ and we need to store them in oder to ensure consistency with previous adversary instances.

THE FORGER'S FIRST STAGE: BEFOREFORK$(X, m)$: In this stage, the forger first tries to evaluate the function $F$ on its input using EVALF. If no previous instance with the same input exists, the instance has already forked and BEFOREFORK immediately returns. If the instance has not yet forked from all other instances, i.e., if there exists a previous instance with the same input, it receives back the index $k$ of the $R$ to which $F$ evaluates. In this case it proceeds to ask query $c_i = \mathcal{R}.\mathcal{H}(\phi(R_k), m)$ and appends $(k, c_i)$ to $\tau$. If it has not already forged a signature it then computes $Z_i := R_k X^{c_i}$. If the forking point has been reached, the adversary now forks from the previous instances as described in FORK. Otherwise, if $Z_i \in \Gamma_{\text{good}}$, then $\mathcal{A}_{F,f}$ forges a signature by calling FORGE$(R_k, Z_i)$. The algorithm will repeat the described process until the forking point is reached.

THE FORGER'S SECOND STAGE AFTERFORK$(X, m)$: After the current instance has forked from all previous instances it proceeds as follows. Until exactly $q$ random oracle queries have been asked, $\mathcal{A}_{F,f}$ queries $c_i := \mathcal{R}.\mathcal{H}(\phi(R_{j,i}), m)$ and appends $((j, i), c_i)$ to $\tau$. If the adversary has not already forged a signature, it continues to compute $Z_i := R_{j,i} X^{c_i}$. If $\phi(Z_i)$ is neither in $\Gamma_{\text{good}}$ nor in $\Gamma_{\text{bad}}$, the adversary decides in which set to put it by invoking DECIDE. If afterwards $\phi(Z_i)$ is in $\Gamma_{\text{good}}$, a signature is forged. The algorithm continues in this fashion until exactly $q$ random oracle queries have been asked.

HANDLING THE FORKING POINT FORK$(Z, k, c)$: When the simulation of $\mathcal{A}_{F,f}$ reaches the forking point, it first checks if $\phi(Z)$ is already in $\Gamma_{\text{good}}$, i.e., if it already knows the discrete logarithm. If that is the case it produces a forgery. Otherwise, it checks whether $\phi(Z)$ is also not contained in $\Gamma_{\text{bad}}$ and if $\phi(Z)$ is indeed neither contained in $\Gamma_{\text{good}}$ nor in $\Gamma_{\text{bad}}$, the simulation again decides in which set to put it by invoking DECIDE. If afterwards $\phi(Z)$ is contained in $\Gamma_{\text{good}}$ the simulation also produces a forgery.

DECIDING WHETHER TO FORGE DECIDE$(Z, k, c)$: To decide whether $Z$ belongs in $\Gamma_{\text{good}}$ or $\Gamma_{\text{bad}}$, the simulation tosses a biased coin $\delta_z \leftarrow_{\$} \texttt{Ber}_{\mu}$. If $\delta_z = 0$ then $Z$ is added to $\Gamma_{\text{bad}}$. If $\delta_z = 1$ then $Z$ is added to $\Gamma_{\text{good}}$, its discrete logarithm $y$ is computed using DLOG and $(Z, y)$ is appended to $\mathcal{D}$.

COMPUTING THE DISCRETE LOGARITHM DLOG$(Z, k, c)$: Computation of the discrete logarithm is performed by exhaustively searching for a $y \in \mathbb{Z}_p$ satisfying $g^y = Z$.

PRODUCING A FORGERY FORGE$(R, Z)$: Actually producing a forgery is trivial, because forgeries will only be produced for $Z \in \Gamma_{\text{good}}$ and for each such $Z$, $\mathcal{D}$ already contains the discrete logarithm. Accordingly, a forgery is produced by finding the entry $(Z', y') \in \mathcal{D}$ such that $Z' = Z$ and returning $(R, y')$

FINALIZATION OF $\mathcal{M}_0$: Eventually, $\mathcal{R}$ outputs a solution $\hat{S} := (\hat{S}_1, \ldots, \hat{S}_w, S') \in \hat{G}^w \times \{0,1\}^*$. Then $\mathcal{M}_0$ runs $(i_1, \ldots, i_w) :=$ GETIDX$(\hat{S}_1, \ldots, \hat{S}_w)$ to determine the indices of group elements $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w})$ corresponding to encodings $(\hat{S}_1, \ldots, \hat{S}_w)$, and outputs $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w}, S')$.

**Analysis of $\mathcal{M}_0$** Note that $\mathcal{M}_0$ provides a perfect simulation of the oracles $\mathcal{O}$ and $\mathcal{S}_\Pi$ and it also mimics the inefficient attacker from Section 4.1 perfectly, the only difference being that $F$ is chosen lazily. In particular, $(R, y')$ is a valid forgery for message $m$ and thus, $\mathcal{R}^{\mathcal{O}, \mathcal{S}'_\Pi, \mathcal{A}_{F,f}}$ outputs a solution $\hat{S} = (\hat{S}_1, \ldots, \hat{S}_w, S')$ to

PROC $\mathcal{M}_0(C)$

\# INITIALIZE

parse $C = (C_1, \ldots, C_u, C')$

$\mathcal{L}^{\mathbb{G}} := \emptyset \quad ; \quad \mathcal{L}^{E} := \emptyset$

$\vec{R} = (R_{1,1}, \ldots, R_{n,q}) \leftarrow_{\$} \mathbb{G}^{nq}$

$\mathcal{I} := (C_1, \ldots, C_u, R_{1,1}, \ldots, R_{n,q})$

ENCODE($\mathcal{I}$)

$\mathcal{T} := \emptyset \; ; \; \Gamma_{\text{good}} := \emptyset \; ; \; \Gamma_{\text{bad}} := \emptyset$

$\mathcal{D} := \emptyset \quad ; \quad j := 0$

$\hat{C} := (\mathcal{L}_1^E, \ldots, \mathcal{L}_u^E, C')$

$\hat{S} \leftarrow_{\$} \mathcal{R}^{\mathcal{O}, \mathcal{S}'_{\Pi}, \mathcal{A}}(\hat{C})$

\# FINALIZATION

parse $\hat{S} := (\hat{S}_1, \ldots, \hat{S}_w, S')$

$(i_1, \ldots, i_w) :=$ GETIDX$(\hat{S}_1, \ldots, \hat{S}_w)$

return $(\mathcal{L}_{i_1}^{\mathbb{G}}, \ldots, \mathcal{L}_{i_w}^{\mathbb{G}}, S')$

PROC FORGE$(R, Z)$ :

Find $(Z', y') \in \mathcal{D}$ s.t. $Z' = Z$

$\sigma := (\phi(R), y')$

PROC EVALF$(\tau)$ :

if $\nexists \tau \in \mathcal{T}$ s.t. $(m, \phi(X), \omega) \prec \tau$

  return $\perp$

pick first $i \in [|\mathcal{T}|]$

  such that $\tau \prec \mathcal{T}_i$

$(k, c) := \mathcal{T}_{i,|\tau|+1}$

return $k$

PROC AFTERFORK$(m, X)$ :

while $i \leq q$

  $c_i := \mathcal{R}.\mathcal{H}(\phi(R_{j,i}), m)$

  append $((j, i), c_i)$ to $\tau$

  if $\sigma = \perp$

    $Z_i := R_{j,i} X^{c_i}$

    ENCODE$(Z_i)$

    if $\phi(Z_i) \notin \Gamma_{\text{good}} \cup \Gamma_{\text{bad}}$

      DECIDE$(Z_i, (j, i), c_i)$

    if $\phi(Z_i) \in \Gamma_{\text{good}}$

      FORGE$(R_{j,i}, Z_i)$

    $i := i + 1$

PROC $\mathcal{A}(\phi(X), m, \omega)$ :

$j := j + 1$

$i_X :=$ GETIDX$(\phi(X))$

$\tau := (\phi(X), m, \omega)$

$\sigma := \perp \quad ; \quad i := 1$

BEFOREFORK$(m, \mathcal{L}_{i_X}^{\mathbb{G}})$

AFTERFORK$(m, \mathcal{L}_{i_X}^{\mathbb{G}})$

append $\tau$ to $\mathcal{T}$

return $\sigma$

PROC FORK$(Z, k, c)$ :

if $\phi(Z) \in \Gamma_{\text{good}}$

  FORGE$(R_k, Z)$

else

  if $\phi(Z) \notin \Gamma_{\text{good}} \cup \Gamma_{\text{bad}}$

    DECIDE$(Z, k, c)$

  if $\phi(Z) \in \Gamma_{\text{good}}$

    FORGE$(R_k, Z)$

PROC BEFOREFORK$(m, X)$

$k :=$ EVALF$(\tau)$

while $k \neq \perp$

  $c_i := \mathcal{R}.\mathcal{H}(R_k, m)$

  append $(k, c_i)$ to $\tau$

  $k' :=$ EVALF$(\tau)$

  if $\sigma = \perp$

    $Z_i := X^{c_i} R_k$

    ENCODE$(Z_i)$

    if $k' = \perp$

      FORK$(Z_i, k, c_i)$

    else

      if $\phi(Z_i) \in \Gamma_{\text{good}}$

        FORGE$(R_k, Z_i)$

  $i := i + 1 \quad ; \quad k := k'$

PROC DECIDE$(Z, k, c)$ :

$\delta_z \leftarrow_{\$} \mathtt{Ber}_{\mu}$

if $\delta_z = 0$

  $\Gamma_{\text{bad}} = \Gamma_{\text{bad}} \cup \{\phi(Z)\}$

else

  $\Gamma_{\text{good}} = \Gamma_{\text{good}} \cup \{\phi(Z)\}$

  $y :=$ DLOG$(Z, k, c)$

  append $(Z, y)$ to $\mathcal{D}$

PROC DLOG$(Z, k, c)$ :

for each $y \in \mathbb{Z}_p$

  if $g^y = Z$ return $y$

Figure 6: Meta-Reduction $\mathcal{M}_0$.

$\hat{C}$ with probability $\Pr[X_0] = \epsilon_{\mathcal{R}}$. Since $\Pi$ is assumed to be representation-invariant, $S := (S_1, \ldots, S_w, S')$ is therefore a valid solution to $C$, where $\hat{S}_i = \phi(S_i)$ for $i \in [w]$. Thus $\mathcal{M}_0$ outputs a valid solution $S$ to $C$ with probability $\epsilon_{\mathcal{R}}$.

**Game 1.** In this game we introduce an implementation $\mathcal{M}_1$ which extends $\mathcal{M}_0$ with bookkeeping, exactly like in Game 1 from the proof of Theorem 11. See Figure 7. Briefly summarized, we introduce an additional list $\mathcal{L}^V \subseteq \mathbb{Z}_p^{u+nq}$ to record the sequence of operations performed by $\mathcal{A}$. Let $\eta_i$ denote the $i$-th canonical unit vector in $\mathbb{Z}_p^{u+nq}$. Then this list is initialized as $\mathcal{L}_i^V = \eta_i$ for $i \in [u + nq]$. Whenever $\mathcal{R}$ asks to perform a computation $(\mathcal{L}_i^E, \mathcal{L}_j^E, \circ)$, then $\mathcal{M}_1$ proceeds as before, but additionally appends $a := \mathcal{L}_i^V + \mathcal{L}_j^V \in \mathbb{Z}_p^{u+nq}$ (if $\circ = \cdot$) or $\mathcal{L}_i^V - \mathcal{L}_j^V \in \mathbb{Z}_p^{u+nq}$ (if $\circ = \div$) to $\mathcal{L}^V$.

Furthermore, in order to keep list $\mathcal{L}^V$ consistent with $\mathcal{L}^{\mathbb{G}}$ (exactly like in the proof of Theorem 11), we replace the generic group oracle $\mathcal{O}$ of $\mathcal{M}_0$ with the following procedure.

GENERIC GROUP ORACLE $\mathcal{O}(e, e', \circ)$: Given a query $(e, e', \circ) \in E \times E \times \{\cdot, \div\}$, the oracle $\mathcal{O}$ determines the smallest indices $i$ and $j$ such that $e = e_i$ and $e' = e_j$ by calling GETIDX. It computes $a := \mathcal{L}_i^V \diamond \mathcal{L}_j^V \in \mathbb{Z}_p^{u+nq}$, where $\diamond := +$ if $\circ = \cdot$ and $\diamond := -$ if $\circ = \div$, and appends $a$ to $\mathcal{L}^V$. Finally it returns ENCODE$(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$.

Recall that the initial content $\mathcal{I}$ of $\mathcal{L}^{\mathbb{G}}$ is $\mathcal{I} = (C_1, \ldots, C_u, R_{1,1}, \ldots, R_{n,q})$, and that $\mathcal{R}$ performs only group operations on $\mathcal{I}$. Now, by construction of $\mathcal{M}_1$, it holds that

$$\mathcal{L}_i^{\mathbb{G}} = \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V) \qquad \text{for all} \qquad i \in [|\mathcal{L}^{\mathbb{G}}|].$$

Thus, at any point in time during the execution of $\mathcal{R}$, the entire list $\mathcal{L}^{\mathbb{G}}$ of group elements can be recomputed from $\mathcal{L}^V$ and $\mathcal{I}$ by setting $\mathcal{L}_i^{\mathbb{G}} := \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V)$ for $i \in [|\mathcal{L}^V|]$.

Again this change is made to keep list $\mathcal{L}^V$ consistent with $\mathcal{L}^{\mathbb{G}}$, i.e., to ensure that $\mathcal{L}_i^{\mathbb{G}} = \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V)$ for all $i \in [|\mathcal{L}^{\mathbb{G}}|]$, where $\mathcal{I} := (C_1, \ldots, C_u, R_{1,1}, \ldots, R_{n,q})$. Clearly $\mathcal{R}$ is completely oblivious to this change, thus

$$\Pr[X_1] = \Pr[X_0]$$

**Game 2.** In this game we introduce an implementation $\mathcal{M}_2$ (cf. Figure 8) which works exactly as $\mathcal{M}_1$, except that it aborts when it would have to compute a new forgery at a forking point. That is, $\mathcal{M}_2$ aborts when it would have to forge in the case where it queried an $R_i$ already asked by a previous instance of the adversary but received a different answer $c_i$. This step is important, because in the final implementation $\mathcal{M}_3$ we will not be able to simulate valid signatures if this happens.

FORK$(Z, k, c)$: If FORK is called on input $\phi(Z)$, such that $\phi(Z)$ is neither in $\Gamma_{\text{good}}$ nor in $\Gamma_{\text{bad}}$, and the DECIDE places it in $\Gamma_{\text{good}}$, then $\mathcal{M}_2$ aborts.

**Analysis of $\mathcal{M}_2$.** We claim that $\mathcal{R}$ is not able to distinguish $\mathcal{M}_2$ from $\mathcal{M}_1$ with probability greater than $(n \ln\left((1 - \epsilon_{\mathcal{A}})^{-1}\right))/q(1 - p^{-1/4})$. To show this, observe that Game 2 and Game 1 are perfectly indistinguishable, as long as $\mathcal{M}_2$ does not abort in FORK. We use Lemma 4 of [26] to bound the probability of an abort.

**Lemma 14** (Based on Lemma 4 of [26]). *The probability that $\mathcal{M}_2$ aborts in* FORK *is at most*

$$\frac{n \ln\left((1 - \epsilon_{\mathcal{A}})^{-1}\right)}{q(1 - p^{-1/4})}$$

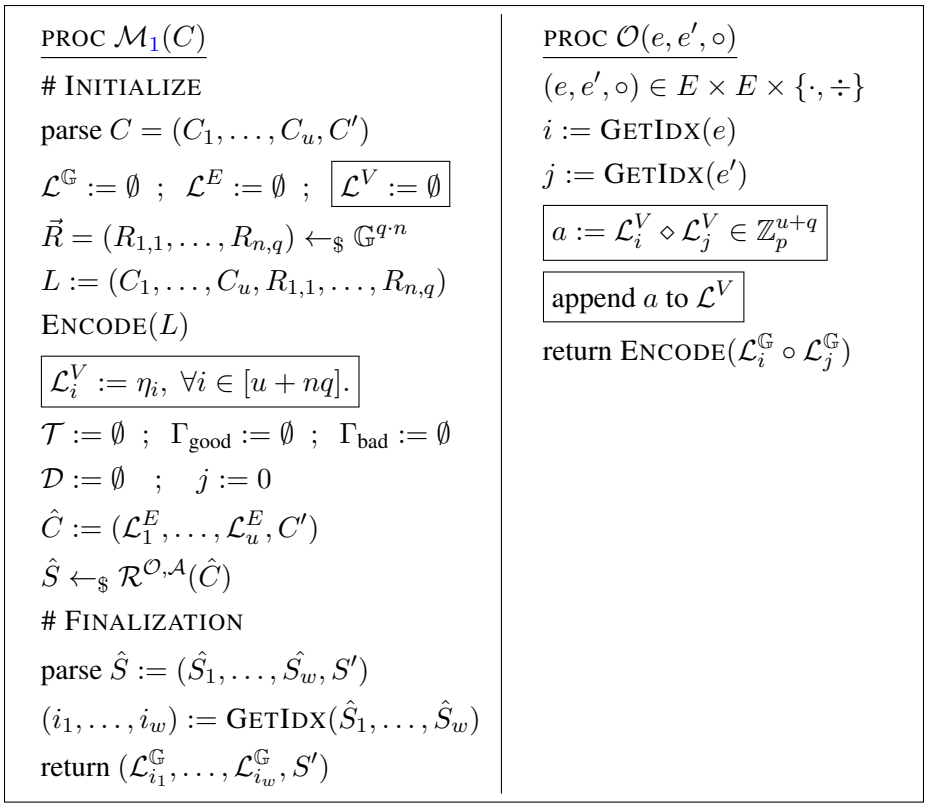


Figure 7: Extending $\mathcal{M}_0$ with additional bookkeeping yields $\mathcal{M}_1$. The boxed elements show the difference to $\mathcal{M}_0$. All procedures not shown are not changed.

We thus have

$$\Pr[X_2] \geq \Pr[X_1] - \Pr[F_1] \geq \Pr[X_1] - \frac{n \ln\left((1-\epsilon_{\mathcal{A}})^{-1}\right)}{q(1-p^{-1/4})}.$$

**Game 3.** Note that the meta-reductions described in previous games were not efficient, because the simulation of the attacker in procedure $\mathcal{A}$ needed to compute a discrete logarithm by exhaustive search. In this final game, we construct an efficient meta-reduction $\mathcal{M}_3$ that it identical to $\mathcal{M}_2$, with the difference that it simulates $\mathcal{A}$ efficiently. $\mathcal{M}_3$ proceeds exactly like $\mathcal{M}_2$, except for the following (cf. Figure 9).
DLOG$(Z, k, c)$: The DLOG procedure chooses $y \leftarrow_{\$} \mathbb{Z}_p$ uniformly random and computes

$$R^*_{j,i} := g^y \cdot X^{-c} \tag{6}$$

Then it reads the first $u + qn$ entries from $\mathcal{L}^{\mathbb{G}}$ as

$$(C_1, ..., C_u, R'_{1,1}, ..., R'_{q,n}) := (\mathcal{L}_1^{\mathbb{G}}, \ldots, \mathcal{L}_{u+qn}^{\mathbb{G}}),$$

replaces $R_{j,i}$ with $R^*_{j,i}$ by setting

$$\mathcal{I}^* := (C_1, \ldots, C_u, R'_{1,1}, \ldots, R'_{j,i-1}, R^*_{i,j}, R'_{j,i+1}, \ldots, R'_{q,n}),$$

and finally re-computes the entire list $\mathcal{L}^{\mathbb{G}}$ from $\mathcal{L}^{V}$ by setting $\mathcal{L}_a^{\mathbb{G}} := \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}_a^{V})$ for all $a \in [|\mathcal{L}^{V}|]$. Note that this implicitly defines $Z$ as $Z := g^y$, due to (6).

PROC FORK$(Z, k, c)$ :

if $\phi(Z) \in \Gamma_{\text{good}}$

  FORGE$(R_k, Z)$

else

  if $\phi(Z) \notin \Gamma_{\text{good}} \cup \Gamma_{\text{bad}}$

    DECIDE$(Z, k, c)$

  if $\phi(Z) \in \Gamma_{\text{good}}$

    Abort simulation

Figure 8: The difference between $\mathcal{M}_1$ and $\mathcal{M}_2$.

PROC DLOG$(Z, (j, i), c)$ :

$y \leftarrow_{\$} \mathbb{Z}_p$

$R^*_{j,i} := g^y \cdot X^{-c}$

$(C_1, ..., C_u, R'_{1,1}, ..., R'_{q,n}) := (\mathcal{L}^{\mathbb{G}}_1, \ldots, \mathcal{L}^{\mathbb{G}}_{u+qn})$

$\mathcal{I}^* := (C_1, \ldots, C_u, R'_{1,1}, \ldots, R'_{j,i-1}, R^*_{j,i}, R'_{j,i+1}, \ldots, R'_{n,q})$

for $k = 1, \ldots, |\mathcal{L}^{\mathbb{G}}|$ do

  $\mathcal{L}^{\mathbb{G}}_k := \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_k)$

return $y$

Figure 9: The difference between $\mathcal{M}_2$ and $\mathcal{M}_3$.

Note that meta-reduction $\mathcal{M}_3$ can be implemented efficiently, as it does not have to compute discrete logarithms. It remains to show that it is indistinguishable from $\mathcal{M}_2$ for $\mathcal{R}$ with all but negligible probablility.

**Analysis of $\mathcal{M}_3$.** First note that each $\sigma$ with $\sigma \neq \perp$ output by $\mathcal{A}$ is a valid signature. Moreover, we claim that $\mathcal{R}$ is not able to distinguish $\mathcal{M}_3$ from $\mathcal{M}_2$, except for a negligibly small probability. To this end, we apply a lemma which is very similar to Lemma 13 from the proof of Theorem 11.

**Lemma 15.** *Let $F_2$ denote the event that $\mathcal{R}$ computes vectors $\mathcal{L}^V_a, \mathcal{L}^V_b \in \mathcal{L}^V$ such that*

$$\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_a) = \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_b) \quad \wedge \quad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_a) \neq \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_b)$$

$$\textit{or}$$

$$\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_a) \neq \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_b) \quad \wedge \quad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_a) = \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_b).$$

*Then*

$$\Pr[F_2] \leq \frac{2n(u + nq + t_{\mathcal{R}})}{p}.$$

Before we sketch the proof of this lemma (which is very similar to the proof of Lemma 13), let us finish the proof of Theorem 11. Note that $\mathcal{M}_3$ is perfectly indistinguishable from $\mathcal{M}_2$, unless Event $F$ occurs. Applying the above lemma, we thus obtain

$$\Pr[X_3] \geq \Pr[X_2] - \Pr[F_2] \geq \Pr[X_2] - \frac{2n(u + nq + t_{\mathcal{R}})}{p}.$$

Summing up, we thus obtain that

$$\epsilon \geq \epsilon_{\mathcal{R}} - \frac{2n(u + nq + t_{\mathcal{R}})}{p} - \frac{n \ln\left((1 - \epsilon_{\mathcal{A}})^{-1}\right)}{q(1 - p^{-1/4})}.$$

□

**Proof sketch for Lemma 15.** The proof of Lemma 15 is almost identical to the proof of Lemma 13. The main difference is that we need to simulate many (up to $n$) signatures in the multi-instance case. This works well, with the same arguments as in the proof of Lemma 13, as long as we make sure that we do not need to re-assign the same encoding twice. (In particular because this would invalidate a signature previously computed by $\mathcal{A}$, and thus be easily noticeable for $\mathcal{R}$.)

By construction of $\mathcal{M}_3$, this can happen only if FORK receives as input a group element $Z$ such that $\phi(Z) \in \Gamma_{\text{good}}$. Note that this is exactly when event $F_1$ occurs, in which case the game is aborted anyway, due to the changes introduced in Game 2.

Suppose that event $F_1$ does not occur. In this case we re-assign each encoding at most once, by replacing in list $\mathcal{L}^{\mathbb{G}}$ a uniformly distributed group element $R_{i,j}$ with another uniform group element $R^*_{i,j}$, and re-computing all group elements contained in $\mathcal{L}^{\mathbb{G}}$. Following Lemma 13, each replacement can be noticed by $\mathcal{R}$ with probability at most

$$\frac{2(u + nq + t_{\mathcal{R}})}{p},$$

where the term $u + nq$ (instead of $u + q$ as before) is due to the fact that in the multi-instance case $\mathcal{L}^{\mathbb{G}}$ is now initialized with $u + nq$ group elements. Since in total at most $n$ encodings are re-assigned throughout the game, a union bound yields

$$\Pr[F_2] \leq \frac{2n(u + nq + t_{\mathcal{R}})}{p}.$$

## Acknowledgments

Nils Fleischhacker and Dominique Schröder were supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA – www.cispa-security.org).

## References

[1] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003.

[2] Mihir Bellare and Adriana Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 162–177, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Berlin, Germany.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[4] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416, Saragossa, Spain, May 12–16, 1996. Springer, Berlin, Germany.

[5] Daniel J. Bernstein. Proving tight security for Rabin-Williams signatures. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 70–87, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany.

[6] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 443–459, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Berlin, Germany.

[7] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Berlin, Germany.

[8] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.

[9] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign RSA signatures. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 112–132, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Berlin, Germany.

[10] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 449–466, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany.

[11] Yevgeniy Dodis and Leonid Reyzin. On the power of claw-free permutations. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 55–73, Amalfi, Italy, September 12–13, 2002. Springer, Berlin, Germany.

[12] Marc Fischlin and Nils Fleischhacker. Limitations of the meta-reduction technique: The case of Schnorr signatures. *EUROCRYPT 2013*, 2013.

[13] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 303–320, Singapore, December 5–9, 2010. Springer, Berlin, Germany.

[14] Sanjam Garg, Raghav Bhaskar, and Satyanarayana V. Lokam. Improved bounds on security reductions for discrete log based signatures. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 93–107, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Berlin, Germany.

[15] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[16] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 537–553, Cambridge, UK, April 15–19, 2012. Springer, Berlin, Germany.

[17] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12, Cirencester, UK, December 19–21, 2005. Springer, Berlin, Germany.

[18] Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for schnorr signatures. *J. Mathematical Cryptology*, 3(1):69–87, 2009.

[19] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 1–20, Chennai, India, December 4–8, 2005. Springer, Berlin, Germany.

[20] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Saragossa, Spain, May 12–16, 1996. Springer, Berlin, Germany.

[21] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20, Cambridge, MA, USA, February 19–21, 2004. Springer, Berlin, Germany.

[22] Andy Rupp, Gregor Leander, Endre Bangerter, Alexander W. Dent, and Ahmad-Reza Sadeghi. Sufficient conditions for intractability over black-box groups: Generic lower bounds for generalized DL and DH problems. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 489–505, Melbourne, Australia, December 7–11, 2008. Springer, Berlin, Germany.

[23] Sven Schäge. Tight proofs for signature schemes without random oracles. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 189–206, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.

[24] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Berlin, Germany.

[25] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[26] Yannick Seurin. On the exact security of schnorr-type signatures in the random oracle model. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 554–571, Cambridge, UK, April 15–19, 2012. Springer, Berlin, Germany.

[27] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Berlin, Germany.

[28] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127, Aarhus, Denmark, May 22–26, 2005. Springer, Berlin, Germany.