On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model

Yannick Seurin

ANSSI, Paris, France
yannick.seurin@m4x.org

Abstract. The Schnorr signature scheme has been known to be provably secure in the Random Oracle Model under the Discrete Logarithm (DL) assumption since the work of Pointcheval and Stern (EUROCRYPT '96), at the price of a very loose reduction though: if there is a forger making at most q_h random oracle queries, and forging signatures with probability ε_F, then the Forking Lemma tells that one can compute discrete logarithms with constant probability by rewinding the forger $\mathcal{O}(q_h/\varepsilon_F)$ times. In other words, the security reduction loses a factor $\mathcal{O}(q_h)$ in its time-to-success ratio. This is rather unsatisfactory since q_h may be quite large. Yet Paillier and Vergnaud (ASIACRYPT 2005) later showed that under the One More Discrete Logarithm (OMDL) assumption, any *algebraic* reduction must lose a factor at least $q_h^{1/2}$ in its time-to-success ratio. This was later improved by Garg *et al.* (CRYPTO 2008) to a factor $q_h^{2/3}$. Up to now, the gap between $q_h^{2/3}$ and q_h remained open. In this paper, we show that the security proof using the Forking Lemma is essentially the best possible. Namely, under the OMDL assumption, any algebraic reduction must lose a factor $f(\varepsilon_F)q_h$ in its time-to-success ratio, where $f \leq 1$ is a function that remains close to 1 as long as ε_F is noticeably smaller than 1. Using a formulation in terms of expected-time and queries algorithms, we obtain an optimal loss factor $\Omega(q_h)$, independently of ε_F. These results apply to other signature schemes based on one-way group homomorphisms, such as the Guillou-Quisquater signature scheme.

Keywords: Schnorr signatures, discrete logarithm, Forking Lemma, Random Oracle Model, meta-reduction, one-way group homomorphism.

1 Introduction

Schnorr Signatures. The Schnorr signature scheme [25,26], derived from the Schnorr identification scheme (an honest-verifier zero-knowledge proof of knowledge of a discrete logarithm) through the Fiat-Shamir transform [12], is one of the earliest discrete log-based signature schemes proposed in the literature. Its simplicity and efficiency (short signature length and the possibility of precomputing exponentiations for very quick on-line signature generation) has attracted considerable attention. Its security has been analyzed in the Random Oracle Model (ROM) [2] under the Discrete Logarithm (DL) assumption by

Pointcheval and Stern [23,24]. The main idea of the proof is to have the forger output two distinct forgeries corresponding to the same random oracle query, but for two distinct answers of the random oracle. The so-called Forking Lemma shows that by rewinding the forger $\mathcal{O}(q_h/\varepsilon_F)$ times, where q_h is the maximal number of random oracle queries of the forger and ε_F its success probability, then one finds two such forgeries with constant probability, which enables to compute the discrete logarithm of the public key. Said otherwise, the reduction loses a factor $\mathcal{O}(q_h)$ in its time-to-success ratio. This results in a very loose security assurance since q_h may be quite large (*e.g.* 2^{60}), which implies to increase the problem parameters length in order to achieve an appropriate provable security level.

Previous Negative Results. Whether the loss of this factor q_h is unavoidable remained obscure until Paillier and Vergnaud [22] showed that under the One More Discrete Logarithm (OMDL) assumption[1], any *algebraic*[2] reduction from the DL problem to forging Schnorr signatures in the ROM must lose a factor $\Omega(q_h^{1/2})$ in its time-to-success ratio. Starting from a reduction from the DL problem to forging Schnorr signatures in the ROM, [22] builds a *meta-reduction* that solves the OMDL problem without using any forger (it simulates the forger using the discrete log oracle it can access to solve the OMDL problem). This result was later improved by Garg *et al.* [14] to a factor $\Omega(q_h^{2/3})$, using the same meta-reduction (only the *analysis* of its success probability was improved). Interestingly, [14] also showed that under a simple assumption on the forger (namely that the distribution of the random oracle query index ℓ corresponding to the forged signature is uniformly random in $[1..q_h]$), the factor lost in the time-to-success ratio of the reduction of [24] can be reduced from $\mathcal{O}(q_h)$ to $\mathcal{O}(q_h^{2/3})$. Since the meta-reduction used in [22,14] simulates a forger that obeys this assumption, one cannot hope to improve the analysis of this particular meta-reduction to show that a factor $\Omega(q_h)$ must be lost by any algebraic reduction.

Contributions of this Work. Up to now, the gap between the security reduction of [24] loosing a factor $\mathcal{O}(q_h)$ and the lower bound $\Omega(q_h^{2/3})$ of [14] remained open. Basically two possible directions were conceivable in order to narrow it: either improve the security reduction of [24] for a *general* forger, or find a better meta-reduction enabling to overcome the $q_h^{2/3}$ bound. We essentially close this gap in the second direction by showing that under the OMDL assumption, any algebraic reduction from the DL problem to forging Schnorr signatures in the ROM must lose a factor $f(\varepsilon_F)q_h$ in its time-to-success ratio, where f is a function that remains close to 1 as long as the success probability ε_F of the forger is noticeably smaller than 1. Our meta-reduction is different from the one used in [22,14] (this is unavoidable by the previous considerations). In particular, the random oracle query index ℓ corresponding to the forged signature is

[1] The OMDL problem consists in solving $n+1$ discrete logarithms by making at most n calls to a discrete log oracle (cf. Section 2).

[2] An algebraic reduction is limited to perform group operations when it manipulates group elements (cf. Section 4).

not uniformly distributed in $[1..q_h]$ (it has a truncated geometric distribution), nor is it independent for two distinct executions of the forger (as we argue later, a uniformly distributed forgery index ℓ is in fact quite unnatural). Though the description of our new meta-reduction is slightly more complicated, its analysis is arguably simpler (the analysis of [14] uses advanced results on the statistics of random permutations). Curiously, our bound vanishes when ε_F is negligibly close to 1. We argue however that this shortcoming is due to the formulation in terms of strictly bounded adversaries. By considering definitions using expected-time (and queries) algorithms, we are able to show that any algebraic reduction must lose a factor $\Omega(q_h)$, independently of ε_F, in its expected-time-to-success ratio.

Interpretation of Our Results. Interpreting our results is quite delicate (as is often the case for results in the ROM). The conservative point of view would be to consider that breaking Schnorr signatures in the ROM is strictly easier than solving the DL problem (which our results do not prove), and to increase security parameters adequately. Yet taking into account that no one has been able to find a better forgery attack than by solving the DL problem, another possible interpretation is that they point out the limitations of black-box reduction techniques. For example, consider the (t, q_h, ε)-forger $\mathcal{F}$ obtained as follows: starting from any algorithm that (t, ε)-solves the DL problem, $\mathcal{F}$ first recovers the secret key, and then forges a signature corresponding to one of its $q_h > 1$ random oracle queries (*e.g.* uniformly chosen at random). This adversary is arguably artificial since it could forge a signature for any message *with a single random oracle query.* Yet any black-box reduction will lose a huge factor when using such a forger, whereas a non-black-box one, accessing the DL-subroutine of the forger, would yield back an algorithm solving the DL problem with the same time-to-success ratio as the forger.

Related Work. Techniques similar to the ones of [22,14] and this paper were used to separate one-more computational problems independently by Brown [7] (who termed such results *irreductions*) and Bresson *et al.* [6].

Coron [10] gave a result close in spirit to ours for the RSA with Full Domain Hash (FDH) signature scheme [3]: he showed that the security of RSA-FDH in the ROM cannot be proved tightly equivalent to the hardness of inverting RSA. This was generalized by Dodis and Reyzin [11] to FDH used with any trapdoor one-way permutation induced by a family of claw-free permutations. There are however two main differences between these results and ours. First, the result of [10,11] is specific to chosen-message attacks (FDH is tightly secure for no-message attacks), whereas in our case the result holds even for no-message attacks. Second, the factor necessarily lost by any reduction for FDH is $\Omega(q_s)$, where q_s is the maximal number of *signature* queries asked by the forger. A security proof matching this $\Omega(q_s)$ bound had been previously given by Coron [9].

The security of the Schnorr signature scheme in the standard model remains elusive (beyond the obvious fact that key-recovery is as hard as the DL problem under no-message attacks).[3] Paillier and Vergnaud [22] showed that under the OMDL assumption, it is immune to key-recovery under chosen-message attacks (whatever the hash function used), but that it cannot be proved universally unforgeable under no-message attacks with respect to an algebraic reduction (again under the OMDL assumption). Neven *et al.* [21] gave necessary conditions on the hash function for the Schnorr signature scheme to be existentially unforgeable under chosen-message attacks, and also showed that these conditions are sufficient in the generic group model. To the best of our knowledge, these are the only results up to now. All practical[4] discrete log-based signature schemes provably secure in the standard model rely on bilinear groups [4,28].

Faced with the apparent impossibility to obtain tight security reductions in the ROM for discrete log-based schemes, two main research options emerged. The first was to rely on weaker assumptions, with proposals such as the EDL scheme [15] and subsequent improvements [8] relying on the Computational Diffie-Hellman assumption, and the proposal by Katz and Wang [18] relying on the Decisional Diffie-Hellman assumption (see also [16]). The second option was to find alternatives to the Fiat-Shamir transform with tighter security reductions, as explored by Micali and Reyzin [20] (but their technique is inapplicable to discrete log-based schemes) and Fischlin [13] (but the resulting scheme is relatively inefficient).

Open Problems. We leave the problem of eliminating the dependency in ε_F for strictly bounded adversaries as an intriguing (though minor) open question. This paper more or less settles the case of algebraic reductions; a natural question is what can be said for arbitrary reductions. More generally, an interesting research subject is to build an efficient signature scheme with a tight reduction in the ROM under the DL assumption (and not under weaker related ones), or to prove a general impossibility result. Another important challenge is to say anything meaningful about the security of Schnorr signatures in the standard model, or to propose a practical scheme based on DL-like assumptions provably secure in the standard model and not relying on bilinear groups.

Organization. In Section 2, we give the necessary background on Schnorr signatures and the DL and OMDL problems. In Section 3, we recall the security proof of [24] for Schnorr signatures through the Forking Lemma. In Section 4, we describe our new meta-reduction and show in Section 5 that it implies a necessary loss of a factor $f(\varepsilon_F)q_h$ for any algebraic reduction. In the full version of the paper [27], we put our results in a more general framework based on one-way group homomorphisms, and extend them to other related signature schemes (such as Modified ElGamal). We also treat the expected-time and queries scenario in the full version.

[3] We note that the Fiat-Shamir transform is known to be intrinsically problematic in the standard model [17].

[4] General constructions of signature schemes from any one-way function are known, but are quite impractical.

2 Preliminaries

$[i..j]$ will denote the set of integers k such that $i \leq k \leq j$. When $\mathcal{X}$ is a non-empty finite set, we write $x \leftarrow_{\$} \mathcal{X}$ to mean that a value is sampled uniformly at random from $\mathcal{X}$ and assigned to x. We denote $\texttt{Ber}_\mu$ the Bernoulli distribution of parameter $\mu \in [0,1]$ (*i.e.* $\delta \leftarrow \texttt{Ber}_\mu$ is such that $\Pr[\delta = 1] = \mu$ and $\Pr[\delta = 0] = 1 - \mu$), and for $\mu \in [0,1]$ and a non-zero positive integer q, we denote $\texttt{Bin}_{\mu,q}$ the binomial distribution of parameters μ and q (*i.e.* $X \leftarrow \texttt{Bin}_{\mu,q}$ is such that $\Pr[X = k] = \binom{q}{k}\mu^k(1-\mu)^{q-k}$). The security parameter will be denoted κ. We will write $f = \texttt{poly}(\cdot)$ to denote a polynomially bounded function and $f = \texttt{negl}(\cdot)$ to denote a negligible function. We assume the existence of an adequate group generation algorithm, which on input 1^κ returns a cyclic group $\mathbb{G}$ of prime order $q \in [2^{\kappa-1}, 2^\kappa[$ and a generator g of $\mathbb{G}$. We will assume that all algorithms are given $(\mathbb{G}, q, g)$ as input and will sometimes not mention it explicitly.

The Schnorr signature scheme is obtained by applying the Fiat-Shamir transform [12] to the Schnorr identification scheme [25,26].

Definition 1 (Schnorr Signature Scheme). *Let $\mathbb{G}$ be a cyclic group of prime order q and g be a generator of $\mathbb{G}$. Let $H : \{0,1\}^* \times \mathbb{G} \to \mathbb{Z}_q$ be a hash function. The Schnorr signature scheme is defined as follows:*

– *Key generation: Let $x \leftarrow_{\$} \mathbb{Z}_q \setminus \{0\}$, and $y = g^x$. The private key is x and the public key is y.*
– *Signature: To sign a message $m \in \{0,1\}^*$, draw $a \leftarrow_{\$} \mathbb{Z}_q$, compute $r = g^a$, $c = H(m, r)$, and $s = a + cx \mod q$. The signature is (s, c).*
– *Verification: Given a message $m \in \{0,1\}^*$, and a claimed signature (s, c), compute $r = g^s y^{-c}$ and check that $c = H(m, r)$.*

From a practical point of view, the Schnorr signature scheme is more usually defined with a hash function mapping its inputs to $\{0,1\}^k$ (interpreted as integers in $[0..(2^k - 1)]$) rather than $\mathbb{Z}_q$. There is no difficulty in extending our results to this case (q must simply be replaced by 2^k in Theorem 2). When we talk of the Schnorr signature scheme in the Random Oracle Model (ROM), we mean the scheme obtained when H is replaced by a random oracle.

In this work we focus on security against universal forgery under no-message attacks (UF-NM-security) in the ROM. This a weak security notion, but this makes our negative result of Section 4 stronger than considering a more constraining notion such as security against existential forgery under chosen-message attacks.

Definition 2 (UF-NM Forger). *A forger $\mathcal{F}$ is said to $(t_F, q_h, \varepsilon_F)$-UF-NM-break Schnorr signatures in the ROM if on input any message $m \in \{0,1\}^*$ and a public key $y \leftarrow_{\$} \mathbb{G}$, $\mathcal{F}$ runs in time at most t_F, makes at most q_h queries to the random oracle, and returns a valid forgery (s, c) for m with probability at least ε_F (where the probability is taken over the random choice of y, the random tape of $\mathcal{F}$, and the answers of the random oracle).*
Moreover, we will say that the forgery (s, c) corresponds to the random oracle query index $\ell \in [1..q_h]$ *if the ℓ-th query/answer of $\mathcal{F}$ to the random oracle was $H(m, g^s y^{-c}) = c$.*

In all the following, we will assume *wlog* the following: when $\mathcal{F}$ returns a forgery (s, c), and made the query $(m, g^s y^{-c})$ to the random oracle, the corresponding answer was c (in other words, the forger never returns a forgery that it knows to be invalid: we assume it returns $\perp$ in this case). For clarity, when the forger returns a forgery corresponding to the random oracle query index ℓ, we will assume it outputs the triplet (ℓ, s, c). Note that the forger may return a random forgery that does not correspond to any of its random oracle queries, in which case it is valid with probability $1/q$. We will denote $(\emptyset, s, c)$ the output of the forger in that case. In all the following, when we say that the forger returns a forgery (ℓ, s, c), we mean $\ell \neq \emptyset$ unless otherwise stated.

As we will see in Section 3, the security of Schnorr signatures in the ROM can be proved under the assumption that the Discrete Logarithm (DL) problem, that we formalize below, is hard.

Definition 3 (DL Problem). *Let $\mathbb{G}$ be a cyclic group of order q and g be a generator of $\mathbb{G}$. An algorithm $\mathcal{A}$ is said to (t, ε)-solve the DL problem if on input (G, q, g) and $r \leftarrow_{\$} \mathbb{G}$, it runs in time at most t and returns the discrete logarithm of r in base g with probability at least ε (where the probability is taken over the random choice of r and the random tape of $\mathcal{A}$).*

The One-More Discrete Logarithm (OMDL) problem, introduced under the name Known-Target DL problem in [1], is defined as follows. Note that Koblitz and Menezes [19] argue that the ODML problem might be easier than the DL problem for some groups.

Definition 4 (OMDL Problem). *Let $\mathbb{G}$ be a cyclic group of order q and g be a generator of $\mathbb{G}$. Let Θ be an oracle taking no input and returning a random element of $\mathbb{G}$ (named the* challenge *oracle). Let $\mathtt{DLog}_g(\cdot)$ be the oracle returning the discrete logarithm in base g of its input. An algorithm $\mathcal{A}$ is said to (t, n, ε)-solve the OMDL problem if on input $(\mathbb{G}, q, g)$, it runs in time at most t, makes $m \leq n+1$ queries $r_1, \ldots, r_m \leftarrow \Theta$, and returns the discrete logarithm of all r_i's in base g while making* strictly less *than m queries to $\mathtt{DLog}_g(\cdot)$, with probability at least ε (where the probability is taken over the random challenges of Θ and the random tape of $\mathcal{A}$).*

3 Security Proof with the Forking Lemma

In this section, we recall the analysis of the security of the Schnorr signature scheme using the Forking Lemma [23,24]. We focus on UF-NM-security, but there is no difficulty in extending the result to existential forgery and to chosen-message attacks using the honest-verifier zero-knowledge property of the Schnorr identification scheme [24].

The main idea is to obtain from the forger two valid forgeries (ℓ, s, c) and (ℓ, s', c') corresponding to the same random oracle query (m, r), but for distinct answers of the random oracle $c \neq c'$. Indeed this implies $r = g^s y^{-c} = g^{s'} y^{-c'}$, which yields the discrete logarithm of the public key $\mathtt{DLog}_g(y) = (s - s')/(c - c')$ mod q. For this, the reduction runs the forger with input some message m, public key y (the target element of the reduction), and some uniformly chosen random tape ω, answering the random oracle queries of the forger uniformly at random, until it returns a forgery corresponding to some random oracle query index $\ell \in [1..q_h]$. Then, it replays the forger, using the same input (m, y), the same random tape ω and the same answers to random oracle queries up to the $(\ell - 1)$-th one as for the successful execution. Consequently, the ℓ-th random oracle query of the forger is the same as in the successful execution. Starting from the ℓ-th random oracle query, the reduction draws the answers uniformly at random again (using the terminology of Section 4, we will say that such an execution *forks* from the successful one at point ℓ). It repeats this until the forger returns another forgery corresponding to the same random oracle query index $\ell \in [1..q_h]$. The Forking Lemma gives a lower bound on the probability that this strategy succeeds.

The security result for Schnorr signatures can be concretely stated as the following theorem, from which it can easily be seen that the security reduction loses a factor $\mathcal{O}(q_h)$ in its time-to-success ratio t_R/ε_R compared with the one of the forger t_F/ε_F.

Theorem 1 ([24]). *Assume there is a forger which $(t_F, q_h, \varepsilon_F)$-UF-NM-breaks Schnorr signatures in the ROM for some group parameters $(\mathbb{G}, q, g)$. Assume moreover that $\varepsilon_F \geq \max(2/(q+1), 16q_h/q)$. Then there is a reduction $\mathcal{R}$ which (t_R, ε_R)-solves the DL problem (for the same group parameters), where $t_R \simeq (16q_h + 2)t_F/\varepsilon_F$ and $\varepsilon_R > 0.099$.*

Proof. We give a slightly adapted proof in the full version of the paper [27]. □

4 Description of the New Meta-reduction

In the next section we will prove the following result, that we state informally for now.

Theorem (Informal). *Under the OMDL assumption, any algebraic reduction from the DL problem to UF-NM-breaking Schnorr signatures in the ROM must lose a factor $f(\varepsilon_F)q_h$ in its time-to-success ratio, where q_h is the maximal number of random oracle queries of the forger, ε_F its success probability, and $f(\varepsilon_F) = \varepsilon_F / \ln\left((1 - \varepsilon_F)^{-1}\right)$.*

In order to prove this result, we will start from an algebraic reduction $\mathcal{R}$ (the meaning of algebraic will be explained shortly) that turns a UF-NM-forger for Schnorr signatures in the ROM into a solver for the DL problem, and describe a meta-reduction $\mathcal{M}$ that uses the reduction $\mathcal{R}$ to solve the OMDL problem without using any forger (the meta-reduction will actually simulate the forger to the reduction thanks to its discrete log oracle). In order to formalize this, we need a precise definition of a reduction.

Definition 5. *A reduction $\mathcal{R}$ is said to $(t_R, n, \varepsilon_R, q_h, \varepsilon_F)$-reduce the DL problem to UF-NM-breaking Schnorr signatures in the ROM if upon input $r_0 \leftarrow_\$ \mathbb{G}$ and after running at most n times any forger which $(t_F, q_h, \varepsilon_F)$-UF-NM-breaks Schnorr signatures, $\mathcal{R}$ outputs $\texttt{DLog}_g(r_0)$ with probability greater than ε_R, within an additional running time t_R (meaning that the total running time of $\mathcal{R}$ is at most $t_R + nt_F$).*

The probability ε_R is taken as in Definition 3 over the random choice of r_0 and the random tape of $\mathcal{R}$ (the random tape of $\mathcal{F}$ is assumed under control of $\mathcal{R}$). The reduction described in the proof of Theorem 1 is a $(\mathcal{O}(1), (16q_h + 2)/\varepsilon_F, 0.099, q_h, \varepsilon_F)$-reduction.

Similarly to previous work [22,14], we will only consider *algebraic* reductions (originally introduced in [5]). An algorithm $\mathcal{R}$ is algebraic with respect to some group $\mathbb{G}$ if the only operations it can perform on group elements are group operations (see [22] for details). We characterize such reductions by the existence of a procedure `Extract` which, given the group elements $(g_1, \ldots, g_k)$ input to $\mathcal{R}$, other inputs σ to $\mathcal{R}$, $\mathcal{R}$'s code, and any group element y produced by $\mathcal{R}$ during its computation in at most t steps, outputs $\alpha_1, \ldots, \alpha_k \in \mathbb{Z}_q$ such that $y = g_1^{\alpha_1} \ldots g_k^{\alpha_k}$. We require that `Extract` runs in time $\texttt{poly}(t, |\mathcal{R}|, \lfloor \log_2 q \rfloor)$, where $|\mathcal{R}|$ is the code size of $\mathcal{R}$. As will appear clearly later, the need to restrict the reduction to be algebraic arises from the fact that $\mathcal{R}$ can run the forger on arbitrary public keys, and the meta-reduction will need to extract the discrete logarithm of these public keys (assuming $\mathcal{R}$ returns the discrete logarithm of its input r_0). This can also be interpreted as saying that $\mathcal{R}$ runs $\mathcal{F}$ on public keys that are derived from its input r_0 through group operations, which does not seem an overly restrictive assumption. Note in particular that the reduction of [24] using the Forking Lemma is algebraic: it repeatedly runs the forger on the same public key $y = r_0$ (or, in the variant described in the full version of the paper [27], on public keys $y = (r_0)^\alpha$ for α's randomly chosen during the first phase of the reduction).

We now describe the new meta-reduction $\mathcal{M}$. It has access to an OMDL challenge oracle Θ returning random elements from $\mathbb{G}$, and to an oracle $\texttt{DLog}_g(\cdot)$ returning the discrete logarithm in base g of its input. It also has access[5] to a $(t_R, n, \varepsilon_R, q_h, \varepsilon_F)$-algebraic reduction $\mathcal{R}$, which expects access to a forger $\mathcal{F}$, and offers a random oracle interface that we denote $\mathcal{R}.H$. We assume $t_R, n, q_h = \texttt{poly}(\kappa)$ and $\varepsilon_R, \varepsilon_F = 1/\texttt{poly}(\kappa)$. Recall that the goal of $\mathcal{M}$ is to return the discrete logarithm of all challenge elements it queries to Θ, by making strictly less queries to $\texttt{DLog}_g(\cdot)$. In all the following we assume $0 < \varepsilon_F < 1$, we fix $\alpha \in]0, (1 - \varepsilon_F)^{1/q_h}[$ and we define the quantities μ_0 and $\mu \in]0,1[$ (whose meaning will appear clearer in view of Lemmata 2 and 3) as:

$$\mu_0 = 1 - (1 - \varepsilon_F)^{1/q_h} \quad \text{and} \quad \mu = \frac{\mu_0}{1-\alpha} = \frac{1}{1-\alpha}\left(1 - (1-\varepsilon_F)^{1/q_h}\right) .$$

[5] By access we essentially mean black-box access, but $\mathcal{M}$ also needs the code of $\mathcal{R}$ to run procedure `Extract`.

$\mathcal{M}$ first queries the OMDL challenge oracle Θ, receiving a random element $r_0 \in \mathbb{G}$, and runs $\mathcal{R}$ on input r_0 and some uniformly chosen random tape. Then it simulates (at most) n sequential executions of the forger that we denote $\mathcal{F}_i(m_i, y_i, \omega_i)$, $1 \le i \le n$, where m_i is the input message, y_i the input public key, and ω_i the random tape of the forger received from the reduction.[6] Depending on how $\mathcal{R}$ chooses (m_i, y_i, ω_i) and the answers to queries of $\mathcal{M}$ to $\mathcal{R}.H$, these successive executions may be identical up to some point, that we will call a *forking point.*

Definition 6 (Forking Point). *Consider two distinct simulated executions of the forger $\mathcal{F}_i(m_i, y_i, \omega_i)$ and $\mathcal{F}_j(m_j, y_j, \omega_j)$, $1 \le j < i \le n$. We say that execution $\mathcal{F}_i$ forks from execution $\mathcal{F}_j$ at point $t_{i/j} = 0$ if $(m_i, y_i, \omega_i) \ne (m_j, y_j, \omega_j)$, or at point $t_{i/j} \in [1..q_h]$ if all the following holds:*

- *$(m_i, y_i, \omega_i) = (m_j, y_j, \omega_j)$;*
- *for $k \in [1..(t_{i/j} - 1)]$, the k-th query and answer to $\mathcal{R}.H$ are the same in both executions;*
- *the $t_{i/j}$-th query to $\mathcal{R}.H$ is the same in both executions, but the answers are distinct.*

We also define the point where execution $\mathcal{F}_i$ forks from all previous executions as $t_i = \max\{t_{i/j},\ 1 \le j < i\}$.

We assume *wlog* that all simulated executions are distinct, *i.e.* they fork at some point.

The simulation of the forger works as follows. The meta-reduction will dynamically construct two (initially empty) disjoint sets $\Gamma_{\text{good}}, \Gamma_{\text{bad}} \subset \mathbb{G}$. Γ_{good} will be the set of elements $z \in \mathbb{G}$ whose discrete logarithm is known from $\mathcal{M}$ because it has made the corresponding query to its discrete log oracle (we assume the discrete logarithm of elements in Γ_{good} are adequately stored by $\mathcal{M}$), while Γ_{bad} will be the set of elements $z \in \mathbb{G}$ such that $\mathcal{M}$ will never make the corresponding query to its discrete log oracle. The main idea of the simulation of the forger on input (m, y, ω) is that $\mathcal{M}$ will return a forgery corresponding to the *first* query $\mathcal{R}.H(m, r)$ such that the answer c satisfies $ry^c \in \Gamma_{\text{good}}$. Whether an element $z \in \mathbb{G}$ will be in Γ_{good} or Γ_{bad} will be determined by drawing a random coin $\delta_z \leftarrow \mathtt{Ber}_\mu$ during the simulation. If $\delta_z = 1$ (resp. $\delta_z = 0$), z will be added to Γ_{good} (resp. Γ_{bad}).

We now describe in details the i-th execution of the forger $\mathcal{F}_i(m_i, y_i, \omega_i)$ (see also Figure 1). Before the simulation begins, $\mathcal{M}$ queries a challenge r_i from Θ and initializes a flag $\mathtt{forge} = \mathtt{false}$. Let t_i denote the point where execution $\mathcal{F}_i$ forks from all previous executions. Assume first that $t_i = 0$, meaning that (m_i, y_i, ω_i) is distinct from the input to all previous executions. Then $\mathcal{M}$ proceeds as follows. For $k = 1, \ldots, q_h$, and while $\mathtt{forge} = \mathtt{false}$, it makes queries $(m_i, r_i^{\beta_{ik}})$ to $\mathcal{R}.H$ using arbitrary[7] randomization exponents $\beta_{ik} \in \mathbb{Z}_q \backslash \{0\}$. Denoting c_{ik} the answer received from $\mathcal{R}.H$, $\mathcal{M}$ computes $z_{ik} = r_i^{\beta_{ik}} y_i^{c_{ik}}$. Three distinct cases may occur:

[6] We stress that $\mathcal{F}_i$, $i = 1, \ldots, n$, denote *distinct executions* of the *same* forger $\mathcal{F}$.

[7] The only constraint is that the β_{ik}'s be distinct in order to avoid making twice the same query.

i) If $z_{ik} \in \Gamma_{\text{bad}}$, then $\mathcal{M}$ simply continues with the next query to $\mathcal{R}.H$.
ii) If $z_{ik} \in \Gamma_{\text{good}}$, then by definition $\mathcal{M}$ already requested $\mathtt{DLog}_g(z_{ik})$ to its discrete log oracle. In that case, it sets $\ell_i = k$, $s_i = \mathtt{DLog}_g(z_{ik})$, $c_i = c_{ik}$, and sets the flag `forge` to `true`.
iii) If $z_{ik} \notin \Gamma_{\text{good}} \cup \Gamma_{\text{bad}}$, then $\mathcal{M}$ draws a random coin $\delta_{z_{ik}} \leftarrow \mathtt{Ber}_\mu$. If $\delta_{z_{ik}} = 0$, z_{ik} is added to Γ_{bad} and $\mathcal{M}$ continues with the next query to $\mathcal{R}.H$. If $\delta_{z_{ik}} = 1$, then $\mathcal{M}$ queries $\mathtt{DLog}_g(z_{ik})$ and adds z_{ik} to Γ_{good}. It then proceeds exactly as in case ii), and moreover stores the value of β_{ik} as β_i.

Once the flag `forge` has been set to `true`, $\mathcal{M}$ completes the sequence of queries to $\mathcal{R}.H$ arbitrarily.[8] When the q_h queries to $\mathcal{R}.H$ have been issued, if `forge` $=$ `false`, then $\mathcal{M}$ returns $\perp$ to $\mathcal{R}$, meaning that execution $\mathcal{F}_i$ fails to forge. Else, `forge` $=$ `true` and $\mathcal{M}$ returns (ℓ_i, s_i, c_i) as set at step ii) as forgery for m_i to $\mathcal{R}$. Moreover, if $\mathcal{M}$ did not query its discrete log oracle during the simulation (either because no forgery was returned or because z_{ik} was already in Γ_{good}), then $\mathcal{M}$ directly queries $\mathtt{DLog}_g(r_i)$ (a more economic strategy could be used, but this simplifies notations).

The simulation for the case $t_i \geq 1$ is quite similar to the case $t_i = 0$, with one important difference though. By definition of the forking point, the t_i first queries to $\mathcal{R}.H$ are determined by previous executions, and $\mathcal{M}$ must simulate the forger accordingly. In particular, it cannot embed the current challenge r_i before the $(t_i + 1)$-th query. If there is some query $\mathcal{R}.H(m_i, r)$ of index $k \in [1..(t_i - 1)]$ such that the answer c satisfies $z = ry_i^c \in \Gamma_{\text{good}}$, then $\mathcal{M}$ sets the flag `forge` to `true` and will return a forgery corresponding to the first such query (without having to query its discrete log oracle since z is already in Γ_{good}). Note that this same forgery was necessarily already returned in at least one previous execution. At the end of the simulation, $\mathcal{M}$ directly queries $\mathtt{DLog}_g(r_i)$.

Assume now that the flag `forge` is still set to `false` when arrived at the t_i-th query. By definition of the forking point, this query was first issued during a previous execution $j < i$, so that $\mathcal{M}$ cannot choose it freshly. The answer of $\mathcal{R}.H$, however, differs from the one received in all previous executions from which $\mathcal{F}_i$ forks exactly at point t_i. Denote $(m_i, \hat{r})$ this t_i-th query to $\mathcal{R}.H$ ($\hat{r} = r_j^{\beta_{jt_i}}$, where r_j was the challenge used during the j-th execution), $\hat{c}$ the corresponding new answer, and $\hat{z} = \hat{r}y_i^{\hat{c}}$. If $\hat{z} \in \Gamma_{\text{bad}}$, then $\mathcal{M}$ can resume the simulation as described for $t_i = 0$, starting from the $(t_i + 1)$-th query to $\mathcal{R}.H$. If $\hat{z} \in \Gamma_{\text{good}}$, then $\mathcal{M}$ can forge a signature for this query without calling its discrete log oracle (and hence will be able to query directly $\mathtt{DLog}_g(r_i)$ at the end of the simulation). If $\hat{z} \notin \Gamma_{\text{good}} \cup \Gamma_{\text{bad}}$, then $\mathcal{M}$ draws a fresh coin $\delta_{\hat{z}} \leftarrow \mathtt{Ber}_\mu$. If $\delta_{\hat{z}} = 0$, then $\mathcal{M}$ can also resume the simulation as described for $t_i = 0$, starting from the $(t_i + 1)$-th query to $\mathcal{R}.H$. The problematic case arises if $\delta_{\hat{z}} = 1$, since $\mathcal{M}$ must return a forgery for the t_i-th query but does not know the discrete logarithm of $\hat{z}$ yet. Hence, $\mathcal{M}$ queries $\hat{s} = \mathtt{DLog}_g(\hat{z})$, completes the sequence of queries to $\mathcal{R}.H$ arbitrarily for $k = t_i + 1$ to q_h, and outputs $(\ell_i = t_i, \hat{s}, \hat{c})$ as forgery for message m_i. After the simulation of $\mathcal{F}_i$, $\mathcal{M}$ makes the additional query $\mathtt{DLog}_g(r_i)$. For

[8] Alternatively, we could let $\mathcal{M}$ stop its queries here since queries after the forgery point are irrelevant.

the sake of the discussion in Section 5, we will say that event Bad happens if this last case occurs during one of the n simulations. As we will see shortly, event Bad makes $\mathcal{M}$ fail since in total $\mathcal{M}$ makes two calls to $\mathtt{DLog}_g(\cdot)$ related to the same challenge r_j.[9]

Once the n calls to the forger have been simulated, the reduction $\mathcal{R}$ returns either $\perp$ (in which case $\mathcal{M}$ returns $\perp$ as well), or the discrete logarithm a_0 of r_0. In the latter case, $\mathcal{M}$ uses the procedure Extract to retrieve[10] $x_i = \mathtt{DLog}_g(y_i)$ for $i = 1$ to n. For each challenge r_i received from Θ, either $\mathcal{M}$ queried directly $a_i = \mathtt{DLog}_g(r_i)$, or during the simulation of $\mathcal{F}_i$, $\mathcal{M}$ returned (ℓ_i, s_i, c_i) as forgery, with $s_i = \mathtt{DLog}_g(r_i^{\beta_i} y_i^{c_i})$. Hence $\mathcal{M}$ can compute the discrete logarithm of r_i as $a_i = (s_i - c_i x_i)/\beta_i \mod q$. Finally, $\mathcal{M}$ returns a_0 and $(a_i)_{i=1..n}$. This concludes the description of the meta-reduction.

Differences with the Previous Meta-reduction. In [22,14], the distribution of the indexes ℓ_i returned by the meta-reduction was uniform in $[1..q_h]$ and independent for each execution. On the contrary, for our meta-reduction, it is not difficult to see that for an execution such that all $z_{ik} = r_i^{\beta_{ik}} y_i^{c_{ik}}$ are fresh, ℓ_i is distributed according to a *truncated geometric distribution*:

$$\Pr[\ell_i = k] = \mu(1-\mu)^{k-1} \text{ for } k \in [1..q_h] \text{ and } \Pr[\ell_i = \perp] = 1 - \sum_{k=1}^{q_h} \mu(1-\mu)^{1-k} \ .$$

Moreover, when an execution forks from previous ones at $t_i > 0$, the distribution of ℓ_i is obviously not independent from the previous forgery indexes ℓ_j. In fact, returning a forgery for independently and uniformly chosen ℓ_i's leads to counter-intuitive behaviors. Consider two distinct executions of a forger $\mathcal{F}$. Assume that some execution $\mathcal{F}_1$ returns a forgery corresponding to some random oracle query index ℓ_1. Then, if another execution $\mathcal{F}_2$ forks from the first one at $t_{2/1} > \ell_1$, it seems more natural for $\mathcal{F}_2$ to return the same forgery as $\mathcal{F}_1$ rather than a new one since the forger "knows" the corresponding signature. Such events cannot happen with our meta-reduction because it simulates a forger that has a natural interpretation: when run on input (m, y), it returns a forgery for the first query $H(m, r)$ such that the answer c satisfies $ry^c \in \Gamma_{\text{good}}$, where Γ_{good} is a set of size $\sim \mu q$ such that the forger can compute the discrete logarithm of elements of Γ_{good} efficiently.

5 Proof of the Main Theorem

We will now prove a sequence of lemmata from which our main result will easily follow. The following lemma will be useful. It results from a simple function analysis and is stated without proof.

[9] We could simply let $\mathcal{M}$ abort in that case, but for simplicity of the analysis we prefer to let it make an additional call to $\mathtt{DLog}_g(\cdot)$.

[10] More precisely, for each $i \in [1..n]$, Extract returns γ_i and γ'_i such that $y_i = g^{\gamma_i} r_0^{\gamma'_i} = g^{\gamma_i + a_0 \gamma'_i}$.

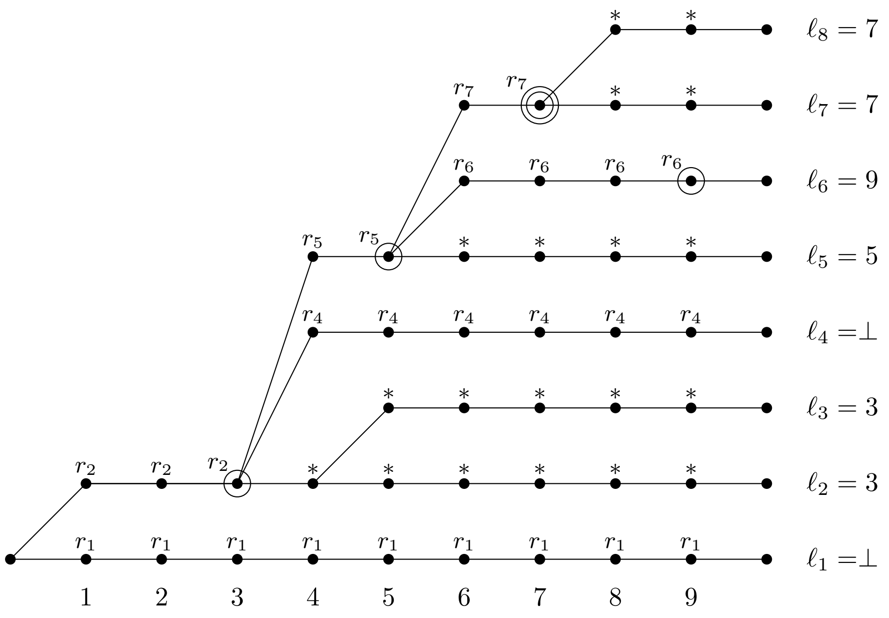


Fig. 1. A possible execution tree of the simulated forger for $q_h = 9$ and $n = 8$. Execution paths go from the root to the leaves. The root symbolizes the beginning of each simulation of the forger. Vertices originating from the root symbolizes the input (m, y, ω) received from $\mathcal{R}$: execution paths sharing the same vertex correspond to the same input. Then, each internal node symbolizes a query to the random oracle $\mathcal{R}.H$, and the vertex originating from this node symbolizes the corresponding answer. Again, execution paths sharing a node, resp. a vertex, share the same query, resp. answer. The label above each query node represents the challenge r_i from Θ used by $\mathcal{M}$ to construct the query (we do not indicate the randomization exponent β_{ik}). Stars indicate that the query is arbitrary since it comes after the forgery point for the execution. Finally, leaves symbolize the output of the forger (a forgery or $\perp$). Here, we simply label leaves with the index ℓ_i of the random oracle query corresponding to the forgery (with the convention that $\ell_i = \perp$ in case the simulated forger returns $\perp$) and we circle the corresponding random oracle query in the execution path. The first execution is run on some input (m_1, y_1, ω_1) and returns no forgery. All subsequent executions are run on the same input $(m_2, y_2, \omega_2) \neq (m_1, y_1, \omega_1)$. The second execution returns some forgery for $\ell_2 = 3$. The third execution forks from the second one at $t_3 = 4 > \ell_2$ so that it returns the same forgery as the second execution. The fourth and fifth executions both fork from previous ones at $t_4 = t_5 = 3$. The fourth one returns no forgery while the fifth one returns a forgery for $l_5 = 5$. The sixth and seventh executions both fork from previous ones at $t_6 = t_7 = 5$, both returning a forgery for resp. $l_6 = 9$ and $l_7 = 7$. Finally, execution 8 forks from previous ones at $t_8 = 7$, and returns a forgery for $l_8 = 7$: since two forgeries related to the same challenge r_7 are returned, event `Bad` happens (assuming $\mathcal{M}$ has to make two queries to its discrete log oracle to forge the signatures).

Lemma 1. *Let $\varepsilon_F \in]0,1[$, and $\mu_0 = 1-(1-\varepsilon_F)^{1/q_h}$. Then for any $q_h \geq 1$, one has:*

$$\varepsilon_F \leq q_h\mu_0 \leq \ln\left((1-\varepsilon_F)^{-1}\right) \ .$$

5.1 Successful Simulation of the Forger

The first thing to do is to lower bound the probability that $\mathcal{R}$ succeeds in returning $\mathtt{DLog}_g(r_0)$. For this, we will show that with sufficiently high probability, $\mathcal{M}$ simulates a "good" forger, *i.e.* a forger that would succeed with probability greater than ε_F when interacting with a real random oracle (rather than $\mathcal{R}.H$).

Definition 7 (Good Forger). *We say that a forger $\mathcal{F}$ making q_h random oracle queries is μ_0-*good *if for any input (m, y, ω), the distribution over uniform sequences of random oracle answers $(c_1, \ldots, c_{q_h})$ of the forgery index ℓ follows a truncated geometric law of parameter $\tilde{\mu} \geq \mu_0$,* i.e. $\Pr[\ell = k] = \tilde{\mu}(1-\tilde{\mu})^{k-1}$ *for $k \in [1..q_h]$.*

Lemma 2. *Let $\mu_0 = 1-(1-\varepsilon_F)^{1/q_h}$. Then a μ_0-good forger making q_h random oracle queries $(t_F, q_h, \varepsilon_F)$-UF-NM-breaks Schnorr signatures in the ROM (for some t_F).*

Proof. Fix any message m. Then for any (y, ω), the probability over the answers $(c_1, \ldots, c_{q_h})$ of the random oracle that $\mathcal{F}$ returns a valid forgery is

$$\sum_{k=1}^{q_h} \tilde{\mu}(1-\tilde{\mu})^{k-1} = 1-(1-\tilde{\mu})^{q_h} \geq 1-(1-\mu_0)^{q_h} = \varepsilon_F \ .$$

This remains true for the probability over (y, ω) and the answers of the random oracle. □

The success probability of the forger simulated by $\mathcal{M}$ when interacting with a real random oracle depends on the random tape of $\mathcal{M}$ through the draws of the coins δ_z. We will now show that with overwhelming probability, $\mathcal{M}$ simulates a μ_0-good forger. Note that the oracle answers c of $\mathcal{R}.H$ may be determined by the random tape of $\mathcal{R}$, which is set uniformly at random by $\mathcal{M}$. Hence elements $z = ry^c$ may range over all $\mathbb{G}$, and $\mathcal{M}$ must be able to draw δ_z independently for *any* $z \in \mathbb{G}$. In order to avoid using an exponential amount of randomness, $\mathcal{M}$ should derive the coins δ_z from a secure pseudorandom number generator. In all the following, we will assume that the coins δ_z are truly random. By a standard hybrid argument, this assumption cannot affect the success probability of $\mathcal{M}$ by more than a negligible quantity (since otherwise $\mathcal{M}$ would constitute a distinguisher for the pseudorandom number generator).

Lemma 3. *Set $\alpha = q^{-1/4}$. Then there is a negligible function ν such that for any challenges $(r_1, \ldots, r_n)$ received from Θ and any randomization exponents β_{ik}, $\mathcal{M}$ simulates a μ_0-good forger with probability greater that $(1-\nu)$ over its random tape.*

Proof. Assume that all coins δ_z for $z \in \mathbb{G}$ are drawn before the simulation starts rather than by lazy sampling (this does not change the success probability of the simulated forger). By definition, $\Gamma_{\text{good}} = \{z \in \mathbb{G} : \delta_z = 1\}$. Clearly, the size of Γ_{good} is distributed according to the binomial distribution $\mathtt{Bin}_{\mu,q}$. A Chernoff bound hence gives:

$$\nu \overset{\text{def}}{=} \Pr_{\delta_z} [|\Gamma_{\text{good}}| \leq (1-\alpha)\mu q] \leq e^{-\mu q \alpha^2/2} \ .$$

Fix an arbitrary input (m, y, ω). For any $r \in \mathbb{G}$, the probability over $c \leftarrow_\$ \mathbb{Z}_q$ that $ry^c \in \Gamma_{\text{good}}$ is equal to $\tilde{\mu} = |\Gamma_{\text{good}}|/q$. Recall that the simulated forger returns a forgery corresponding to the first random oracle query $H(m, r)$ such that the answer c satisfies $ry^c \in \Gamma_{\text{good}}$. Hence, independently of the sequence of queries of the simulated forger, the distribution over uniform sequences of random oracle answers $(c_1, \ldots, c_{q_h})$ of the forgery index ℓ follows a truncated geometric law of parameter $\tilde{\mu}$. When $|\Gamma_{\text{good}}| > (1-\alpha)\mu q = \mu_0 q$, then $\tilde{\mu} > \mu_0$. This holds for any input (m, y, ω) and any sequence of queries of the simulated forger, so that for any challenges $(r_1, \ldots, r_n)$ received from Θ and any randomization exponents β_{ik}, with probability greater than $(1-\nu)$ over the draws of the coins δ_z, $\mathcal{M}$ simulates a μ_0-good forger. Moreover, we have:

$$e^{-\mu q \alpha^2/2} = e^{-\frac{q_h \mu_0 q \alpha^2}{2q_h(1-\alpha)}} \leq e^{-\frac{q_h \mu_0 q \alpha^2}{2q_h}} \leq e^{-\frac{\varepsilon_F \sqrt{q}}{2q_h}} \ ,$$

where for the last inequality we used Lemma 1 and $\alpha = q^{-1/4}$. Since by assumption $q_h = \mathtt{poly}(\kappa)$ and $\varepsilon_F = 1/\mathtt{poly}(\kappa)$, we see that ν is negligible, hence the result. □

5.2 Success of the Meta-reduction

The next step is to analyze the probability that $\mathcal{M}$ succeeds given that $\mathcal{R}$ does. It is straightforward to verify that the computation of the discrete logarithm of all challenges $(r_1, \ldots, r_n)$ received from Θ by $\mathcal{M}$ is correct. Consequently, given that $\mathcal{R}$ returns the discrete logarithm of r_0, $\mathcal{M}$ may only fail because it did not make strictly less queries to $\mathtt{DLog}_g(\cdot)$ than to Θ. However, it is not hard to see from the description of $\mathcal{M}$ that if event $\mathtt{Bad}$ does not happen, then $\mathcal{M}$ makes *exactly* one query to its discrete log oracle per simulation of the forger, and hence returns the discrete logarithm of $n+1$ challenges while making n queries to $\mathtt{DLog}_g(\cdot)$. Hence, given that $\mathcal{R}$ returns $a_0 = \mathtt{DLog}_g(r_0)$, and that event $\mathtt{Bad}$ does not happen, then $\mathcal{M}$ is successful.

The last step towards proving our main theorem is to bound the probability of event $\mathtt{Bad}$.

Lemma 4. *Event* $\mathtt{Bad}$ *happens with probability less than*

$$n\mu \leq \frac{n \ln\left((1-\varepsilon_F)^{-1}\right)}{(1-\alpha)q_h} \ .$$

Proof. Consider the i-th simulation of the forger by $\mathcal{M}$. Let t_i be the point where this execution forks from all previous executions. By construction of $\mathcal{M}$, `Bad` can only happen if $t_i \geq 1$, and the output of the fresh coin $\delta_{\hat{z}}$ (we refer to notations of Section 4) drawn to decide whether a signature must be forged for the t_i-th query is 1, which happens with probability μ. An union bound on the n simulated executions and Lemma 1 give the result. □

5.3 Main Theorem and Discussion

We are now ready to state and prove the main theorem of this paper.

Theorem 2. *Assume there is an algebraic reduction $\mathcal{R}$ that $(t_R, n, \varepsilon_R, q_h, \varepsilon_F)$-reduces the DL problem to UF-NM-breaking Schnorr signatures in the ROM, with $\varepsilon_F < 1$. Set $\alpha = q^{-1/4}$. Then there is a negligible function ν such that the meta-reduction $\mathcal{M}$ (t_M, n, ε_M)-solves the OMDL problem, where:*

$$\varepsilon_M \geq \varepsilon_R \left(1 - \nu - \frac{n \ln\left((1-\varepsilon_F)^{-1}\right)}{(1-\alpha)q_h}\right)$$
$$t_M \leq \texttt{poly}(t_R, |\mathcal{R}|, n, q_h, \lfloor \log_2(q) \rfloor) \ .$$

Proof. Denote `Sim` the event that $\mathcal{M}$ simulates a μ_0-good forger. By Lemma 2 and by definition of a $(t_R, n, \varepsilon_R, q_h, \varepsilon_F)$-reduction, when `Sim` happens, $\mathcal{R}$ returns $\texttt{DLog}_g(r_0)$ with probability greater than ε_R (over r_0 and its own random tape). Provided that $\mathcal{R}$ returns the discrete logarithm of r_0 and that `Bad` does not happen, the meta-reduction is successful. Hence, one has $\varepsilon_M \geq \varepsilon_R(1 - \Pr[\overline{\texttt{Sim}}] - \Pr[\texttt{Bad}])$. Combining Lemmata 3 and 4 yields the lower bound on ε_M. Taking into account the fact that $\mathcal{M}$ uses a secure pseudorandom number generator rather than truly random coins cannot modify ε_M by more than a negligible amount (otherwise $\mathcal{M}$ would constitute a distinguisher), that we can incorporate in ν. The running time of $\mathcal{M}$ is upper bounded by the sum of the time needed to simulate the n executions of the forger which is $\texttt{poly}(n, q_h, \lfloor \log_2 q \rfloor)$, the additional running time t_R of $\mathcal{R}$, and the time to run `Extract` which is $\texttt{poly}(t_R, |\mathcal{R}|, \lfloor \log_2 q \rfloor)$, hence the result. □

Remark 1. As already noted by [22] for their meta-reduction, the above proof can be straightforwardly extended to reductions of the OMDL problem to forging Schnorr signatures in the ROM. Hence the security of Schnorr signatures cannot be proved tightly equivalent to the OMDL problem either (under the OMDL assumption).

Interpretation. Recall that the total running time of the reduction is at most $t_R + nt_F$. Denote $\rho_F = t_F/\varepsilon_F$ and $\rho_R = (t_R + nt_F)/\varepsilon_R \geq nt_F/\varepsilon_R$ the time-to-success ratio of resp. the forger and the reduction. Then some computation gives:

$$\frac{n \ln\left((1-\varepsilon_F)^{-1}\right)}{(1-\alpha)q_h} \leq \frac{\varepsilon_R \rho_R}{(1-\alpha) f(\varepsilon_F) q_h \rho_F} \leq \frac{\rho_R}{(1-\alpha) f(\varepsilon_F) q_h \rho_F} \ ,$$

where $f(\varepsilon_F) = \varepsilon_F / \ln\left((1-\varepsilon_F)^{-1}\right)$. Hence one has:

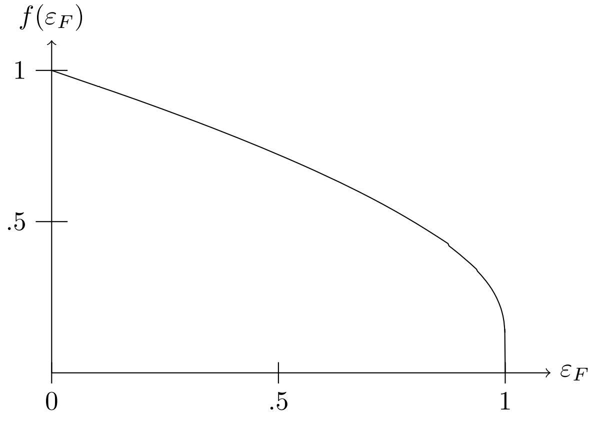
$$\varepsilon_M \geq \varepsilon_R \left(1 - \nu - \frac{\rho_R}{(1-\alpha) f(\varepsilon_F) q_h \rho_F}\right) \ .$$

Since $t_R, |\mathcal{R}|, n, q_h, \lfloor \log_2(q) \rfloor = \texttt{poly}(\kappa)$, $t_M = \texttt{poly}(\kappa)$, so that under the OMDL assumption, one must have ε_M negligible. Then the inequality above yields (using $\varepsilon_R = 1/\texttt{poly}(\kappa)$ and $\nu, \alpha = \texttt{negl}(\kappa)$):

$$\rho_R \geq f(\varepsilon_F) q_h \rho_F - \texttt{negl}(\kappa) \ .$$

Hence one must have that ρ_R is negligibly close to $f(\varepsilon_F) q_h \rho_F$: the reduction essentially loses a factor $f(\varepsilon_F) q_h$ in its time-to-success ratio.

The function $f(\varepsilon_F)$ is depicted below. For small ε_F, one has $f(\varepsilon_F) \simeq 1 - \varepsilon_F/2$ (which is a good approximation up to $\varepsilon_F \simeq 0.5$). For ε_F close to 1, writing $\varepsilon_F = 1 - u$, one has $f(\varepsilon_F) \simeq -1/\ln(u)$. In particular, for $\varepsilon_F = 1 - 1/\texttt{poly}(\kappa)$, $f(\varepsilon_F) \simeq C/\ln(\kappa)$ for some constant C, which shows that f approaches 0 very slowly. For $f(\varepsilon_F) \leq q_h^{-1/3}$, our bound becomes worse than the one by Garg *et al.* [14]. However, for large q_h (which is the case of interest), this implies that ε_F is very close to 1 (*e.g.* for $q_h = 2^{60}$, a rough estimation shows that our bound is not worse than $q_h^{2/3}$ before $\varepsilon_F > 1 - e^{-2^{19}}$).



It is interesting to consider what happens when $\varepsilon_F = 1$ since our bound vanishes in that case, while both the security reduction of [24] and the necessary loss $\Omega(q_h^{2/3})$ of [14] hold. In that case one has by definition $\mu = 1$, which means that the meta-reduction simulates an adversary which always returns a forgery corresponding to its *first* random oracle query (in which case there is a reduction which succeeds by running the forger only twice). However, this singularity seems to be an artifact due to definitions in terms of strictly bounded-time and queries algorithms and we can escape it by considering expected-time and queries algorithms. This is developed in the full version of the paper [27]. The main idea is that when simulating a forger making an expected number of random oracle queries q_h, one can choose the distribution of the forgery index ℓ to be a geometric distribution of parameter $\mu \simeq 1/q_h$. This is not possible when the number of oracle queries must be strictly less than q_h, in which case we had to appeal to a truncated geometric distribution. It remains nevertheless that in the special case of a forger making strictly less than q_h random oracle queries and forging with probability $\varepsilon_F = 1$, we do not know of any better simulation strategy than choosing the forgery index uniformly at random in $[1..q_h]$ as was done in the meta-reduction of [22,14], in which case one gets a loss factor $\Omega(q_h^{2/3})$ at best.

References

1. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme. Journal of Cryptology 16(3), 185–215 (2003)
2. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: ACM Conference on Computer and Communications Security, pp. 62–73 (1993)
3. Bellare, M., Rogaway, P.: The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
4. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
5. Boneh, D., Venkatesan, R.: Breaking RSA May Not Be Equivalent to Factoring. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 59–71. Springer, Heidelberg (1998)
6. Bresson, E., Monnerat, J., Vergnaud, D.: Separation Results on the "One-More" Computational Problems. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 71–87. Springer, Heidelberg (2008)
7. Brown, D.R.L.: Irreducibility to the One-More Evaluation Problems: More May Be Less. ePrint Archive Report 2007/435 (2007), `http://eprint.iacr.org/2007/435.pdf`
8. Chevallier-Mames, B.: An Efficient CDH-Based Signature Scheme with a Tight Security Reduction. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 511–526. Springer, Heidelberg (2005)
9. Coron, J.-S.: On the Exact Security of Full Domain Hash. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 229–235. Springer, Heidelberg (2000)
10. Coron, J.-S.: Optimal Security Proofs for PSS and Other Signature Schemes. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 272–287. Springer, Heidelberg (2002)
11. Dodis, Y., Reyzin, L.: On the Power of Claw-Free Permutations. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 55–73. Springer, Heidelberg (2003)
12. Fiat, A., Shamir, A.: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
13. Fischlin, M.: Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152–168. Springer, Heidelberg (2005)

14. Garg, S., Bhaskar, R., Lokam, S.V.: Improved Bounds on Security Reductions for Discrete Log Based Signatures. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 93–107. Springer, Heidelberg (2008)
15. Goh, E.-J., Jarecki, S.: A Signature Scheme as Secure as the Diffie-Hellman Problem. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 401–415. Springer, Heidelberg (2003)
16. Goh, E.-J., Jarecki, S., Katz, J., Wang, N.: Efficient Signature Schemes with Tight Reductions to the Diffie-Hellman Problems. Journal of Cryptology 20(4), 493–514 (2007)
17. Goldwasser, S., Kalai, Y.T.: On the (In)security of the Fiat-Shamir Paradigm. In: Symposium on Foundations of Computer Science, FOCS 2003, pp. 102–115. IEEE Computer Society (2003)
18. Katz, J., Wang, N.: Efficiency improvements for signature schemes with tight security reductions. In: Jajodia, S., Atluri, V., Jaeger, T. (eds.) ACM Conference on Computer and Communications Security, pp. 155–164. ACM (2003)
19. Koblitz, N., Menezes, A.: Another Look at Non-Standard Discrete Log and Diffie-Hellman Problems. ePrint Archive Report 2007/442 (2007), `http://eprint.iacr.org/2007/442.pdf`
20. Micali, S., Reyzin, L.: Improving the Exact Security of Digital Signature Schemes. Journal of Cryptology 15(1), 1–18 (2002)
21. Neven, G., Smart, N.P., Warinschi, B.: Hash Function Requirements for Schnorr Signatures. J. Math. Crypt. 3(1), 69–87 (2009)
22. Paillier, P., Vergnaud, D.: Discrete-Log-Based Signatures May Not Be Equivalent to Discrete Log. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 1–20. Springer, Heidelberg (2005)
23. Pointcheval, D., Stern, J.: Security Proofs for Signature Schemes. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)
24. Pointcheval, D., Stern, J.: Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology 13(3), 361–396 (2000)
25. Schnorr, C.-P.: Efficient Identification and Signatures for Smart Cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (1990)
26. Schnorr, C.-P.: Efficient Signature Generation by Smart Cards. Journal of Cryptology 4(3), 161–174 (1991)
27. Seurin, Y.: On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model. Full version of this paper, `http://eprint.iacr.org`
28. Waters, B.: Efficient Identity-Based Encryption Without Random Oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)