Hash Function Requirements for Schnorr Signatures

Gregory Neven[1,2], Nigel P. Smart[3], and Bogdan Warinschi[3]

[1] IBM Research – Zurich, Switzerland
nev@zurich.ibm.com
[2] Katholieke Universiteit Leuven, Belgium
[3] University of Bristol, United Kingdom
{nigel,bogdan}@cs.bris.ac.uk

Abstract. We provide two necessary conditions on hash functions for the Schnorr signature scheme to be secure, assuming compact group representations such as those which occur in elliptic curve groups. We also show, via an argument in the generic group model, that these conditions are sufficient. Our hash function security requirements are variants of the standard notions of preimage and second preimage resistance. One of them is in fact equivalent to the Nostradamus attack by Kelsey and Kohno (Eurocrypt 2006), and, when considering keyed compression functions, both are closely related to the ePre and eSec notions by Rogaway and Shrimpton (FSE 2004).
Our results have a number of interesting implications in practice. First, since security does not rely on the hash function being collision resistant, Schnorr signatures can still be securely instantiated with SHA-1/SHA-256, unlike DSA signatures. Second, we conjecture that our properties require $O(2^n)$ work to solve for a hash function with n-bit output, thereby allowing the use of shorter hashes and saving twenty-five percent in signature size. And third, our analysis does not reveal any significant difference in hardness between forging signatures and computing discrete logarithms, which plays down the importance of the loose reductions in existing random-oracle proofs, and seems to support the use of "normal-size" groups.

1 Introduction

The Schnorr signature scheme [Sch90,Sch91] has been particularly influential in the design of cryptographic protocols. The signature scheme is derived, via the Fiat–Shamir transform [FS87], from an identification scheme that is a three-move honest-verifier zero knowledge proof of knowledge of a discrete logarithm. It is considered highly attractive from an implementor's perspective because of its remarkable efficiency when instantiated in elliptic-curve groups. As for its provable security properties, Pointcheval and Stern [PS00] used the famous forking lemma to prove the scheme secure under the hardness of computing discrete logarithms in the random oracle model. This important result guarantees

that, as long as the hash function behaves "ideally", the only way to break Schnorr signatures is by solving the discrete logarithm problem.

But what happens if hash functions cease to behave ideally? In particular, how do the recent collision attacks on practical hash functions like SHA-1 and MD5 [WY05,WYY05] affect the security of Schnorr signatures? At first sight, being able to find collisions in the underlying hash function does not seem to lead to direct attacks on the signature scheme, but perhaps there are less straightforward attacks? The best way to answer these questions would be to analyse the security of the Schnorr signature scheme in the standard, i.e., non-random-oracle model, as such a proof would surface sufficient real-world properties for the hash function. Unfortunately, not only do we currently lack a proof in the standard model, but Paillier and Vergnaud [PV05] even provided evidence that such a proof is unlikely to ever be found at all.

In this paper, we enhance confidence in the instantiation of Schnorr signatures, or at least its elliptic-curve variant, with hash functions like SHA-1 and MD5 by analysing its security in another popular idealisation, the generic group model [Sho97]. We present two real-world hash function properties, called *random-prefix preimage* (rpp) and *random-prefix second-preimage* (rpsp) resistance, and we show that they are at the same time necessary conditions in the standard model *and* sufficient conditions in the generic group model for the security of Schnorr signatures. Both properties are strictly weaker than collision resistance, meaning that they are implied by collision resistance, but they do not imply it. The rpp property is in fact equivalent to the Nostradamus attack of Kelsey and Kohno [KK06]. When considering a particular implementation of the hash function based on keyed compression functions, our properties become equivalent to the ePre and eSec notions in the framework of Rogaway and Shrimpton [RS04].

What our result means in practice is that, as long as the underlying group behaves "ideally", the only way to break Schnorr signatures is by breaking either the rpp problem or the rpsp problem associated to the hash function. In particular, it warrants the secure use of Schnorr when implemented with SHA-1/SHA-256 or MD5, as long as the rpp and rpsp problems are still believed to be hard for the respective hash functions.

Apart from instantiation candidates for the hash function, our results have a number of other important implications for the efficiency and security of Schnorr signatures. It was already remarked in Schnorr's original papers [Sch90,Sch91] that the hash functions could be chosen to have smaller output sizes, resulting in shorter signatures, since the schemes' security did not appear to be related to finding collisions in the hash function. In fact, a closer look at the concrete bounds in the random oracle model as obtained through the forking lemma [PS00,BN06] shows that these proofs already supported shorter hashes too.

We observe that for a hash function with an output length of n bits, one expects that both the rpp and rpsp problems require an amount of work of $O(2^n)$ to solve, as opposed to the $O(2^{n/2})$ work needed to find collisions. This warrants

using 128-bit hashes instead of the 256-bit hashes in use today (for a security level of 128 bits), and, since Schnorr signatures consist of one group element and one hash value, this immediately cuts down signature size by twenty-five percent to 384 bits instead of 512 bits, compared to DSA signatures.

One oddity about our analysis is that the concrete bound in our generic-group proof is not tight. In principle, this advises *against* using short hashes, as the loss in security needs to be compensated for by increasing the security parameter, i.e., the hash output length. Interestingly, this situation exactly mirrors the debate around the group size for Schnorr signatures, where the random-oracle analysis via the forking lemma yields a notoriously loose security reduction, while our generic-group analysis does not reflect such a loss. In principle the forking-lemma loss should be compensated for by inflating the size of a group element, but this is rarely done in practice. If one continues this reasoning and considers tightness as a second-order issue, then one can safely use short hash values as well. If on the other hand one takes tightness seriously, then one should inflate both the group and the hash output size. A more detailed discussion is provided in Section 7.

It is worth comparing our results for Schnorr signatures with what is known for the highly similar and ubiquitous DSA scheme [Nat94]. Unlike the Schnorr scheme, in DSA finding a collision in the hash function *does* result, via a known-message attack, in a break on the signature scheme. DSA therefore cannot profit from the reduced hash output and signature size that Schnorr enjoys. For DSA no security proof in the random oracle model is known, but Brown [Bro02,Bro05] did provide a proof in the generic group model for the elliptic-curve variant EC-DSA [JMV01]. The proof is quite involved and reduces the security of EC-DSA to a set of non-standard properties of the hash function and the "conversion function". We feel our result for Schnorr is cleaner, and the associated hash function properties are more natural. Combining our result in the generic group model with the advantage of additionally having a security proof in the random oracle model, we feel that Schnorr signatures are to be preferred over EC-DSA.

In [BPVY00] other variants of DSA are presented, of particular interest to us is what they term Type-II DSA signatures, since in these signatures the hash function H is applied to both the message and the "commitment" as is done in Schnorr signatures. However, there are many differences between the work in [BPVY00] and our work. Firstly [BPVY00] is in the random oracle model, i.e. they assume that H is a random oracle. Our purpose is to show what properties are required of H, and so we do not model H as a random oracle. Secondly, and more importantly, they model what we call the conversion function as a hash function G and then show various conditions on G for the resulting signature scheme to be secure. This is important in DSA-like applications where the function $k \longrightarrow \left(g^k \pmod p\right) \pmod q$ is hard to analyse, but for elliptic curve based signatures the equivalent function $k \longrightarrow (x(kP)) \pmod q$ is much easier to understand and so modelling G as a hash function does not make sense. Thirdly, the authors of [BPVY00] mention that one can obtain a saving of 25 percent in the length of the signature since they use the output of G

as a component of the signature, as opposed to using the output of H as a component of the signature, as is done in the Schnorr scheme. Hence, our work can be considered an analogue to the work in [BPVY00] in that we perform an analysis for the function H as opposed to the function G. Indeed in our analysis we require very little of the conversion function at all.

As a word of warning about security proofs in the random oracle and generic group models, we mention that for both idealizations counterexamples have been found that are secure in the idealized model, but completely insecure for any real-world instantiation [CGH98,Den02]. These are clearly contrived counterexamples however, and the fact that no more natural ones have been found until today has led some researchers to conclude that perhaps they are realistic models after all [KM07]. Fischlin [Fis00] pointed out that anomalies can arise in the *combined* generic group and random oracle model, which was previously used to prove the security of Schnorr signatures [Sch90,Sch91]. We stress that these anomalies do not affect our results, as we consider two separate proofs, one in each model, rather than a single proof in a combined model.

Finally, we hope that our security definitions for hash functions provide further motivation for the hash function community to study in more detail new security notions, including those of [RS04,KK06] and ours.

2 The Generic Group Model

Let G be an abstract group of prime order q, which we shall write additively; one can think of G as the group of integers modulo q under addition. In particular we do not make any assumption as to whether discrete logarithms are hard to compute in G. We let $s = \lceil \log_2 q \rceil$.

In a cryptographic scheme elements of G are encoded by bit strings of length ℓ. Solving the discrete logarithm problem is essentially equivalent to discovering the precise encoding used, it is this intuition which sits behind the generic group model.

We let $\mathbb{G}$ be the set of bit strings of length ℓ, and we let $\tau : G \to \mathbb{G}$ be the "natural representation" of G in $\mathbb{G}$. We shall represent the induced group operation on the set $\tau(G) \subset \mathbb{G}$ multiplicatively. Thus we will use additive notation for the representation in which discrete logarithms might be easy (since we think of this representation as the additive group of integers modulo q) and we use multiplicative notation for the representation in which we believe discrete logarithms to be hard.

In a large number of protocols one needs to also map group elements in $\mathbb{G}$ to the smaller set $\{0,1\}^d$ for $d \leq \ell$, often because ℓ is too large for practical use, or for other efficiency reasons. We therefore assume the existence of a "conversion" function $f : \mathbb{G} \to \{0,1\}^d$. The conversion function does not necessarily preserve any properties of the group law in $\tau(G)$, nor is it necessarily invertible (see the examples below). An important quantity in our analysis is the *conversion density*

$$\delta \;=\; \frac{|f(\tau(G))|}{2^d} \;.$$

We call the function f "almost-invertible" if there is an efficient randomized algorithm which given a random bit string $R \xleftarrow{\$} \{0,1\}^d$, with probability δ, computes a preimage $\mathfrak{R} \in \tau(G) \subset \mathbb{G}$ such that $f(\mathfrak{R}) = R$.

The conversion function f plays a crucial role in Brown's analysis of EC-DSA [Bro02,Bro05], and its existence explains the distinct difference between the existence of a proof of security of EC-DSA in the generic group model and the absence of one for normal DSA. In [Bro02,Bro05] a similar definition of "almost-invertible" is given for the conversion function f, however Brown's definition is stricter than what we will need. Our analysis only requires the relatively weak definition given above.

In our analysis the existence of the conversion function and an almost-inverse algorithm will also be crucial, for essentially the same reasons as in Brown's analysis. However, since the output of the conversion function is passed to the hash function in Schnorr signatures one obtains a considerable simplification of the properties required of the conversion function. Recall in EC-DSA it is the output of the conversion function which forms a portion of the signature, but in Schnorr signatures it is the output of the hash function which performs this task.

Before proceeding we present a number of specific instantiations to illustrate the above setup.

Finite Field Based Systems:

In this case we let p denote a prime of ℓ bits in length such that q divides $p-1$. We let $\mathfrak{g} \in \mathbb{F}_p^* \subset \mathbb{G} = \{0,1\}^\ell$ denote a generator of the subgroup $\langle \mathfrak{g} \rangle$ of order q in $\mathbb{F}_p^*$. The function τ is defined as the group homomorphism from G to $\mathbb{F}_p^* \subset \mathbb{G}$ defined by $\tau(g) = \mathfrak{g}$, where g is the generator of the cyclic group $G = (\mathbb{Z}_q, +)$ of order q. Solving discrete logarithms in the group generated by $\mathfrak{g}$ is then equivalent to being able to invert τ.

There are two standard choices for the conversion function in finite field based systems:

1. Either one selects $d = \ell$ and sets f to be the identity function. In which case f is always invertible, and we have
$$\delta \approx 2^{s-\ell} .$$
2. Or one selects $d = s$ and sets f to be the function $f(x) = x \pmod q$, where one interprets the bit strings as integers. This is the traditional case for DSA-like systems, as it helps reduce the signature size for DSA. In this case it is believed that f is hard to invert, and thus our results will not apply.

Elliptic Curve Based Systems:

In this case, assuming we use (as is normally the case) elliptic curves with small cofactor, we therefore have $\ell = 2s$ since points are represented by two elements in the finite field. The map τ is determined by mapping the generator g of G to

the generator of the elliptic curve. Consequently, solving discrete logarithms in the subgroup generated by this generator is equivalent to inverting τ.

There are again two standard choices for the conversion function f used in elliptic curve based systems:

1. Either one selects $d = \ell$ and sets f to be the identity function. In which case f is invertible, and we have
$$\delta \approx 2^{s-\ell} = 2^{-s}.$$
2. Or one selects $d = s$ and sets f to be the function which returns the x coordinate of a point. In this case we have
$$\delta \approx 1/2$$
and there is a randomized algorithm which given an element in D will return an element in $f(\mathbb{G})$ with probability δ, or will return $\perp$, signalling that f is not invertible on this element of the codomain D. Hence, f is almost-invertible in this case with probability δ.

The generic group model captures the idea that an adversary that attacks a primitive based on some group, does so without exploiting the concrete representation of the group elements (that is the values in the set $\mathbb{G}$.) In terms of the setup described above, attacks in the generic group model are captured by providing the adversary $\mathcal{A}$ with indirect access to $\mathbb{G}$. More precisely, once the adversary is fixed, we select the representation function τ at random from the set $[G \hookrightarrow \mathbb{G}]$ of all possible injective maps from G to $\mathbb{G}$. This ensures that the adversary has no a priori knowledge of how group elements are represented. To perform group computations, the adversary is granted access to an oracle parametrized by τ. The oracle allows us to "indirectly" compute subtractions in G (that is, divisions in the multiplicative group $\tau(G) \subset \mathbb{G}$) as follows. On an input of the form $(\mathfrak{g}_1, \mathfrak{g}_2) \in \mathbb{G} \times \mathbb{G}$ the oracle returns $\mathfrak{g}_1/\mathfrak{g}_2 := \tau(\tau^{-1}(\mathfrak{g}_1) - \tau^{-1}(\mathfrak{g}_2))$ if $\mathfrak{g}_1, \mathfrak{g}_2 \in \tau(G)$. The oracle also returns $\mathfrak{g} = \tau(g)$ on request.

Note that from a division oracle and a generator $\mathfrak{g}$ one can compute all group operations as follows:

- The identity can be computed as $\tau(0) = \mathfrak{i} = \mathfrak{g}/\mathfrak{g}$.
- Given an element $\mathfrak{h}$, its inverse can then be computed from $\mathfrak{h}^{-1} = \mathfrak{i}/\mathfrak{h}$.
- Then given two elements $\mathfrak{g}_1$ and $\mathfrak{g}_2$ one can compute their product from $\mathfrak{g}_1 \cdot \mathfrak{g}_2 = \mathfrak{g}_1/\mathfrak{g}_2^{-1}$.

Shoup's result [Sho97] on the lower bound of $2^{-s/2}$ for solving discrete logarithms in the generic group model can then be intuitively interpreted as follows. The adversary against discrete logarithms essentially only obtains information about τ when it calls the generic group oracle. As this oracle returns random values, information about the underlying group is only determined when the same value is returned twice, and the lower bound then follows from the birthday paradox. In the context of Maurer's model [MW98] our function τ represents the handles on the underlying group elements.

3 The Schnorr Signature Scheme

A digital signature scheme is a tuple of algorithms $\mathsf{DS} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$, where Kg generates a public key pk and corresponding secret key sk for security parameter k; $\mathsf{Sign}(sk, m)$ generates a signature σ on message $m \in \{0,1\}^*$; and $\mathsf{Vfy}(pk, m, \sigma)$ outputs 1 if σ is a valid signature for m under pk and 0 otherwise. Correctness requires that $\mathsf{Vfy}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

We recall the standard security notion from [GMR88] of existential unforgeability under chosen-message attack ($\mathsf{uf\text{-}cma}$). The advantage of an adversary $\mathcal{A}$ in breaking signature scheme $\mathsf{DS} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ is given by

$$\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{DS}}(\mathcal{A}) \;=\; \Pr\left[\begin{array}{c}\mathsf{Vfy}(pk, m, \sigma) = 1 \;\text{ and}\\ \mathcal{A} \text{ did not query } \mathsf{Sign}(sk, m)\end{array}\;\middle|\;\begin{array}{c}(pk, sk) \xleftarrow{\$} \mathsf{Kg}\;;\\ (m, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk)\end{array}\right].$$

We say that DS is $(t, q_{\mathrm{S}}, \epsilon)$ secure if no adversary $\mathcal{A}$ running in time at most t and making at most q_{S} queries to its $\mathsf{Sign}(sk, \cdot)$ oracle has advantage greater than ϵ. In the random oracle model [BR93], the adversary additionally has access to a random oracle that it can query up to q_{H} times.

To a generic group $\mathbb{G}$ as described in Section 2 and a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ (where we interpret the output as an integer in $[0, \ldots, 2^n - 1]$), we associate the Schnorr signature scheme $\mathsf{Sch}[\mathsf{H}]$ as follows:

Kg:	$\mathsf{Sign}(sk, m)$:	$\mathsf{Vfy}(pk, m, (s, h))$:
$sk \xleftarrow{\$} \mathbb{Z}_q\;;\; pk \leftarrow \mathfrak{g}^{sk}$	$r \xleftarrow{\$} \mathbb{Z}_q\;;\; R \leftarrow f(\mathfrak{g}^r)$	$R \leftarrow f(\mathfrak{g}^s \cdot pk^{-h})$
Return (pk, sk)	$h \leftarrow \mathsf{H}(R \Vert m)$	If $\mathsf{H}(R \Vert m) = h$
	$s \leftarrow r + sk \cdot h \pmod q$	Then return 1
	Return (s, h)	Else return 0.

We do not explicitly show the dependency of $\mathsf{Sch}[\mathsf{H}]$ on $\mathbb{G}$, as the group will be clear from the context.

The Schnorr signature scheme was proved secure in the random oracle model using the forking lemma by Pointcheval and Stern [PS00]. It is hard however to extract from their proof any guidance on the output length of H, because it only considers hash functions mapping into $\mathbb{Z}_q$. The same is true for the concrete treatment by Ohta and Okamoto [OO98], and the generalization by Abdalla et al. [AABN02] hides the output length in the security of an underlying identification scheme. Using the general forking lemma of Bellare and Neven [BN06] however, one can obtain the following concrete security bounds for the Schnorr signature scheme:

Theorem 1. *If the discrete logarithm problem in $\mathbb{G}$ is $(t_{\mathsf{dlog}}, \epsilon_{\mathsf{dlog}})$-hard, then the Schnorr signature scheme is $(t_{\mathsf{uf\text{-}cma}}, q_{\mathrm{S}}, q_{\mathrm{H}}, \epsilon_{\mathsf{uf\text{-}cma}})$-secure for*

$$\epsilon_{\mathsf{uf\text{-}cma}} \;=\; \sqrt{(q_{\mathrm{H}} + q_{\mathrm{S}} + 1) \cdot \epsilon_{\mathsf{dlog}}} + \frac{q_{\mathrm{H}} + q_{\mathrm{S}} + 1}{2^n} + \frac{q_{\mathrm{S}}(q_{\mathrm{H}} + q_{\mathrm{S}} + 1)}{q} \qquad (1)$$

and $t_{\mathsf{uf\text{-}cma}} = t_{\mathsf{dlog}}/2 - q_{\mathrm{S}} t_{\mathrm{exp}} + O(q_{\mathrm{H}} + q_{\mathrm{S}} + 1)$, where t_{exp} is the cost of an exponentiation in the group $\mathbb{G}$.

This bound clearly indicates that a hash function with $n = s/2$ output bits should be sufficient to obtain a security level of $s/2$ bits, conforming to our result that H need only be rpp and rpsp-secure, and not collision resistant. (A term of the form $q_{\mathrm{H}}^2/2^n$ would have advised for an s-bit hash function.) We do not claim the above bound as a new result of this paper, but its implication to the hash output length, and hence signature length, seems to have gone mostly unnoticed until now.

4 Hash Function Requirements

In this section we define the two properties of the hash function that we show in this paper to be necessary and sufficient for the security of Schnorr signatures in the generic group model. The properties are variants of the preimage and second preimage problems, where a random prefix is imposed by the experiment. The following definition formally captures the experiments that define these two security notions.

Definition 1 (Random-prefix (second-)preimage problem). *The advantage of an adversary $\mathcal{A}$ in solving the random-prefix preimage (rpp) problem, (respectively the random-prefix second-preimage (rpsp) problem) for prefix in a domain of bitstrings D and hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ is given by*

$$\mathsf{Adv}_{\mathsf{H}}^{\mathsf{rpp}[D]}(\mathcal{A}) \;=\; \Pr\left[\mathsf{H}(R\|m) = h \;\middle|\; (h, St) \xleftarrow{\$} \mathcal{A}\,;\; R \xleftarrow{\$} D\,;\; m \xleftarrow{\$} \mathcal{A}(R, St)\right] \;,$$

$$\mathsf{Adv}_{\mathsf{H}}^{\mathsf{rpsp}[D]}(\mathcal{A}) \;=\; \Pr\left[\mathsf{H}(R\|m) = \mathsf{H}(R\|m') \;\middle|\; \begin{array}{c}(m, St) \xleftarrow{\$} \mathcal{A}\,;\; R \xleftarrow{\$} D \\ m' \xleftarrow{\$} \mathcal{A}(R, St)\end{array}\right] \;,$$

where the probability is taken over the coins of $\mathcal{A}$ and the choice of R. We say that the random-prefix (second-)preimage problem for H is (t, ϵ) hard if no adversary $\mathcal{A}$ running in time t has advantage greater than ϵ in solving it.

Both of the above assumptions are directly implied by the collision resistance of H, a result which we leave to the reader. The fact that collisions can be found in time $O(2^{n/2})$ using a birthday attack does not mandate shorter hashes. For an ideal hash function and sufficiently large d though, one expects both of the above problems to take time $O(2^n)$ to solve, so that an n-bit output hash should be sufficient to provide n bits of security. The rpp problem appeared earlier in work by Kelsey and Kohno [KK06] as the *chosen target forced prefix preimage* problem. They present the so-called "herding" attack that essentially solves it in time $O(2^{3n/4})$, for a diamond structure of width $2^{n/4}$, if the hash function follows the Merkle-Damgård iteration [Mer90,Dam90]. The attack is easily adapted to break rpsp resistance as well. What this means for practice is that one should *not* instantiate the hash function with a Merkle-Damgård iteration of an n-bit compression function. Instead, one should probably simply truncate the output of a $2n$-bit hash function to n bits. (Such a method would in our situation be reminiscent of Lucks' wide-pipe hash [Luc05].) Therefore, using for example the

first 128 bits of the SHA-256 hash should in practice provide a security level of 128 bits.

A connection arises between our new notions and the seven-notion framework of Rogaway-Shrimpton [RS04] when H is the particular instantiation in the dedicated key setting [BR07], of $\mathsf{H}(R\|m) = \mathsf{G}(R, m)$ where $\mathsf{G} : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^n$ and the key R of the hash function G comes from the codomain of f. Namely, it is easy to see that H is rpp resistant if and only if G is everywhere preimage (ePre) resistant, and H is rpsp resistant if and only if G is everywhere second-preimage (eSec) resistant, a notion that is sometimes better known as universal one-way hashing [NY89] or target-collision resistance [BR97]. Unfortunately, to support arbitrary-length messages, we would need a Merkle-Damgård like iteration that preserves ePre and eSec resistance. Such iterations typically require much more random key material [Mir01] than the 256 bits provided by R, and this key material would have to be included in the signatures, as in [HK06], thereby blowing up the signature size. Alternatively, one could derive the keys from R using a small-input random oracle as done in [ANPS07].

5 Necessary Security Properties on the Hash Function

We now show that if either of our two assumptions on the hash function does not hold, then an algorithm can be constructed which breaks the Schnorr signature scheme. Our proofs only work for values of δ which are not negligible small, and so our necessary conditions essentially only have an effect in the real world for specific choices of f. Our necessary conditions however do not require f to be almost-invertible.

Proposition 1. *Let $\mathcal{A}$ be an adversary against the $\mathsf{rpp}[D]$ problem for the hash function H, with domain $D = f(\tau(G)) \subset \{0,1\}^d$. Then there exists an adversary $\mathcal{B}$ against the Schnorr signature scheme such that*

$$\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{Sch}[\mathsf{H}]}(\mathcal{B}) \;=\; \mathsf{Adv}^{\mathsf{rpp}[D]}_{\mathsf{H}}(\mathcal{A}).$$

Proof. Let $\mathcal{A}$ be an adversary against the $\mathsf{rpp}[D]$ problem. We construct algorithm $\mathcal{B}$ that on input public key pk runs $\mathcal{A}$'s first stage to obtain (h, St), chooses $s \xleftarrow{\$} \mathbb{Z}_q$, computes $R = f(\mathfrak{g}^s \cdot pk^{-h})$, and then runs $\mathcal{A}$'s second stage to obtain $m \xleftarrow{\$} \mathcal{A}(R, St)$. Algorithm $\mathcal{B}$ outputs (s, h) as a forgery for message m. If $\mathcal{A}$ is successful then $\mathcal{B}$'s attempted forgery satisfies the verification equation since:

$$\mathsf{H}(R\|m) = h \text{ and } R = f(\mathfrak{g}^s \cdot pk^{-h}).$$

The desired result follows.

A similar result holds for the random-prefix second-preimage resistance property, except now the algorithm $\mathcal{B}$ is no longer a passive adversary against the signature scheme.

Proposition 2. *Let $\mathcal{A}$ be an adversary against the* $\mathsf{rpsp}[D]$ *problem for the hash function* H*, with domain* $D = f(\tau(G)) \subset \{0,1\}^d$*. Then there exists an adversary* $\mathcal{B}$ *against the Schnorr signature scheme such that*

$$\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{Sch}[\mathsf{H}]}(\mathcal{B}) \;=\; \delta \cdot \mathsf{Adv}^{\mathsf{rpsp}[D]}_{\mathsf{H}}(\mathcal{A}) \;.$$

Proof. Let $\mathcal{A}$ be an adversary for the $\mathsf{rpsp}[D]$ problem. Consider the algorithm $\mathcal{B}$ that on input pk runs $\mathcal{A}$'s first stage to obtain (m, St). Algorithm $\mathcal{B}$ then makes a signature query $(s,h) \leftarrow \mathsf{Sign}(sk, m)$. It then computes $R = f(\mathfrak{g}^s \cdot pk^{-h})$ and runs $\mathcal{A}$'s second stage to obtain $m' \xleftarrow{\$} \mathcal{A}(R, St)$. Algorithm $\mathcal{B}$ outputs (s,h) as its forgery on message m'. If $\mathcal{A}$ is successful then $\mathcal{B}$'s attempted forgery satisfies the verification equation since:

$$\mathsf{H}(R\|m) = \mathsf{H}(R\|m') \text{ and } R = f(\mathfrak{g}^s \cdot pk^{-h}).$$

The desired result follows.

6 Sufficient Security Properties in the Generic Group Model

We now adapt the security definition for signature schemes to take into account the generic group model. In the particular case of the Schnorr signature scheme we define its security via the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In the game, both parties have access to a generic group oracle G_τ with τ selected at random (as described earlier in the paper). The challenger generates a signature key $sk = x \xleftarrow{\$} \mathbb{Z}_q$ and computes, using its access to the generic group oracle, the corresponding verification key $pk = \mathfrak{g}^x = \tau(x \cdot g)$ which it passes as input to A.

Besides queries to the generic group oracle, the adversary $\mathcal{A}$ can also request signatures from $\mathcal{C}$: the adversary sends m to $\mathcal{C}$ and obtains $(s,h) \xleftarrow{\$} \mathsf{Sign}(sk, m)$, computed by $\mathcal{C}$ using the generic group oracle.

At some point the adversary outputs a tentative forgery $(m^*, (s^*, h^*))$. Let $\mathcal{A}$ wins be the event that (s^*, h^*) is a valid signature on m^*, and m^* had not been queried prior to the challenger. We define the advantage of A in breaking the security of signature scheme $\mathsf{Sch}[\mathsf{H}]$ in the generic group model by:

$$\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{Sch}[\mathsf{H}]}(\mathcal{A}) = \Pr_{\mathcal{A},\mathcal{C},\tau}[\mathcal{A} \text{ wins}] \,,$$

where the probability is taken over the coins of $\mathcal{A}$ and $\mathcal{C}$, as well as over the choice of τ. We say that the Schnorr signature scheme is (t, ϵ) secure (in the generic group model) if for any adversary that runs in time t we have $\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{Sch}[\mathsf{H}]}(\mathcal{A}) < \epsilon$.

Notice that both the challenger and the adversary access the group via the generic group oracle. In our proofs, the challenger has control over the generic group oracle (in particular the selection of τ), in much the same way that a challenger in the random oracle model may have control over the random oracle

(the so-called "programmable random oracle"). The following theorem, which formalizes our main result, says that the Schnorr signature scheme is secure as long as the hash function used in its construction satisfies the security notions that we put forth in this paper.

Theorem 2. *Let* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n \hookrightarrow \mathbb{Z}_q$ *be some hash function, and let* G *be some fixed group modelled as a generic group over the set of bit strings* $\mathbb{G}$*. Let* $f : \mathbb{G} \to \{0,1\}^d$ *be an almost-invertible conversion function. If both the* $\mathsf{rpp}[D]$ *and the* $\mathsf{rpsp}[D]$ *are hard for* H*, with respect to the domain* $D = \{0,1\}^d$*, then the Schnorr signature scheme* $\mathsf{Sch}[\mathsf{H}]$ *is secure in the generic group model.*

Proof. Fix some adversary $\mathcal{A}$ against the Schnorr signature scheme, and let q_G and q_S be upper bounds on the number of queries that $\mathcal{A}$ makes to its generic group oracle, and to its signing oracle, respectively. Assume that the rpp and rpsp problems are, respectively, $(\epsilon_{\mathsf{rpp}}, t_{\mathsf{rpp}})$ and $(\epsilon_{\mathsf{rpsp}}, t_{\mathsf{rpsp}})$-hard over $\{0,1\}^d$. We show that

$$\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{Sch}[\mathsf{H}]}(\mathcal{A}) \leq \frac{q_G}{\delta} \cdot \epsilon_{\mathsf{rpp}} + \frac{q_S + 2}{\delta} \cdot \epsilon_{\mathsf{rpsp}} + O\left(\frac{(q_S + q_G)^2}{q}\right) \tag{2}$$

for all adversaries $\mathcal{A}$ running in time $t_{\mathcal{A}}$. We will determine the relation between $t_{\mathcal{A}}$, t_{rpp} and t_{rpsp} as we progress.

A particularly useful way of describing the execution of some generic group adversary $\mathcal{A}$ against the Schnorr signature scheme is through a list $\mathcal{L}$ that records what the adversary learns during the interaction. Each entry in the list is a tuple of the form $(\mathfrak{g}, k, l) \in \mathbb{G} \times \mathbb{Z}_q \times \mathbb{Z}_q$ which indicates that the adversary had learned the representation $\mathfrak{g}$ of the group element $(k - x \cdot l) \cdot g$. In particular $\mathcal{L}$ contains $(\mathfrak{g}_g, 1, 0)$ and $(\mathfrak{g}_{g_x}, 0, q-1)$ (the group generator, and the verification key of the scheme), where $\mathfrak{g}_g, \mathfrak{g}_{g_x} \in \mathbb{G}$ are randomly chosen, and are deemed to represent $\mathfrak{g}_g = \tau(g)$, $\mathfrak{g}_{g_x} = \tau(x \cdot g)$. New tuples are added to the list, following the queries of the adversary. When $\mathcal{A}$ makes a query $(\mathfrak{g}_1, \mathfrak{g}_2)$ to its generic group oracle $\mathcal{L}$ is modified as follows. If there exists tuples $(\mathfrak{g}_1, k_i, l_i)$ and $(\mathfrak{g}_2, k_j, l_j)$ in $\mathcal{L}$, but $\mathcal{L}$ does not contain a tuple of the form $(\mathfrak{g}, k_i - k_j, l_i - l_j)$ then the tuple $(\mathfrak{g}, k_i - k_j, l_i - l_j)$ is added to the list. Here, $\mathfrak{g}$ is the representation, under specific map $G \to \mathbb{G}$ chosen in the generic group oracle, of the group element obtained from the $((k_i - k_j) - (l_i - l_j) \cdot x \bmod q) \cdot g$. If either of the tuples $(\mathfrak{g}_1, k_i, l_i)$ or $(\mathfrak{g}_2, k_j, l_j)$ does not exist in $\mathcal{L}$ then we abort. We remark that it is possible to make modifications to the model, and to what follows, so that the queries above can always be answered. However, the changes would only complicate the discussion while adding no additional insight. For each signature query m that the adversary makes to its signing oracle, the tuple $(\mathfrak{g}_R, s, h)$ is added to the list where: $\mathfrak{g}_R$ is the representation of the group element $R = r \cdot g$ (for some randomly chosen $r \in \mathbb{Z}_q$), $h = \mathsf{H}(f(\mathfrak{g}_R)||m)$, and $s = r + x \cdot h \bmod q$.

Conversely (and this is one key technique in our proof, as is in other works on the generic group model), one can simulate the environment of the adversary by constructing and maintaining the function $\tau : G \to \mathbb{G}$ on the fly: each time a new tuple $(\cdot, k, l)$ needs to be inserted in the list, the adversary selects a random

value $\mathfrak{g} \in \mathbb{G}$ and inserts $(\mathfrak{g}, k, l)$ in $\mathcal{L}$, such that $\mathfrak{g}$ is distinct from the first component of all other tuples in $\mathcal{L}$. Provided that during the real execution of the algorithm no two entries in $\mathcal{L}$ correspond to the same group element, that is, $k_i - l_i \cdot x \neq k_j - l_j \cdot x \bmod p$, the simulation carried out this way is perfect (otherwise the first component of entries i and j in $\mathcal{L}$ should be equal, whereas in the simulation they are chosen independently at random). We write $\mathsf{NoEqual}$ for the event that during the execution of $\mathcal{A}$ no two entries i, j in $\mathcal{L}$ satisfy the property that $k_i - x \cdot l_i = k_j - x \cdot l_j \bmod p$.

We have:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{Sch[H]}}(\mathcal{A}) &= \Pr\left[\mathcal{A} \text{ wins} \wedge \mathsf{NoEqual}\right] + \Pr\left[\mathcal{A} \text{ wins} \wedge \overline{\mathsf{NoEqual}}\right] \\ &\leq \Pr\left[\mathcal{A} \text{ wins} \mid \mathsf{NoEqual}\right] + \Pr\left[\overline{\mathsf{NoEqual}}\right]\end{aligned}$$

We obtain the desired result by the two terms on the right hand side. Via the union bound we get:

$$\begin{aligned}\Pr\left[\overline{\mathsf{NoEqual}}\right] &\leq \sum_{i,j \leq |\mathcal{L}|, i \neq j} \Pr_x\left[k_i - x \cdot l_i = k_j - x \cdot l_j \bmod p\right] \\ &= \sum_{i,j \leq |\mathcal{L}|, i \neq j} \Pr_x\left[x = (k_i - k_j)(l_j - l_i)^{-1} \bmod p\right] \\ &= O(\frac{|\mathcal{L}|^2}{q})\end{aligned}$$

To upper bound $\Pr\left[\mathcal{A} \text{ wins} \wedge \mathsf{NoEqual}\right]$ we proceed as follows. We assume that prior to outputting its attempted forgery, the adversary actually verifies its validity using the verification algorithm of the signature scheme. Our assumption is without loss of generality. For any adversary $\mathcal{A}$ who does not fulfill this property, one can easily construct an adversary $\mathcal{B}$ which does, and which has the same advantage in breaking the signature scheme. The assumption implies that for a valid forgery $(m^*, (s^*, h^*))$ output by $\mathcal{A}$ a tuple of the form $(\mathfrak{g}, s^*, h^*)$ necessarily occurs in $\mathcal{L}$. We distinguish several possible relations between the forgery $(m^*, (s^*, h^*))$ output by adversary $\mathcal{A}$ and the entries in list $\mathcal{L}$ that describes its execution. For each of the possibilities we describe an event whose probability we then bound. There are four types of entries in $\mathcal{L}$:

1. The entries triggered by generic group oracle queries.
2. The entries triggered by signature queries.
3. The entry used to initialise the generator (i.e. $(\mathfrak{g}_g, 1, 0)$).
4. The entry used to initialise the public key (i.e. $(\mathfrak{g}_{g_x}, 0, q-1)$).

1. We define $\mathsf{MatchGG}$ to be the event that the forgery output by $\mathcal{A}$ is valid and a tuple of the form $(\mathfrak{g}_{i_0^*}, s^*, h^*)$ has been added to $\mathcal{L}$ following the i_0^* query of $\mathcal{A}$ to its generic group oracle.
2. We define $\mathsf{MatchSig}$ to be the event that the forgery output by $\mathcal{A}$ is valid and that a tuple of the form $(\mathfrak{g}_{i_0^*}, s^*, h^*)$ has been added to $\mathcal{L}$ following the i_0^* signature request of $\mathcal{A}$ (on message $m_{i_0^*}$).

3. Let Match_g be the event that the forgery output by $\mathcal{A}$ is successful, and is of the form $(m^*, (1, 0))$.
4. Let Match_{g_x} be the event that the forgery output by $\mathcal{A}$ is successful, and is of the form $(m^*, (0, q-1))$.

Next we bound the probability of the above events, conditioned on the event $\mathsf{NoEqual}$, and use the bounds to also bound the advantage of the adversary. More precisely, we construct adversaries $\mathcal{B}$ and $\mathcal{C}$ such that:

$$\frac{\delta}{q_G} \cdot \Pr\left[\mathsf{MatchGG} \mid \mathsf{NoEqual}\right] \leq \mathsf{Adv}^{\mathsf{rpp}[D]}_{\mathcal{B},G}(\mathcal{B}) \tag{3}$$

and

$$\frac{\delta}{(q_S+2)} \cdot \Pr\left[\mathsf{MatchSig} \vee \mathsf{Match}_g \vee \mathsf{Match}_{g_x} \mid \mathsf{NoEqual}\right] \leq \mathsf{Adv}^{\mathsf{rpsp}}_{\mathsf{H}}(\mathcal{C}). \tag{4}$$

Bounding the probability of event $\mathsf{MatchGG}$. We construct an adversary $\mathcal{B}$ which runs $\mathcal{A}$ as a subroutine and simulates its environment. The adversary is such that whenever event $\mathsf{MatchGG}$ occurs (and $\mathcal{B}$ does not abort), then adversary $\mathcal{B}$ wins in the game that defines the rpp game.

Adversary $\mathcal{B}$ works as follows. It selects uniformly at random an integer $i_0 \in \{1, 2, \ldots, q_G\}$ (representing the index of one of the generic group queries that $\mathcal{A}$ makes), then selects $x \xleftarrow{\$} \mathbb{Z}_q$, and initialises the list $\mathcal{L}$ with $(\mathfrak{g}_g, 1, 0)$ and $(\mathfrak{g}_{g_x}, 0, q-1)$, with $\mathfrak{g}_g, \mathfrak{g}_{g_x}$ selected uniformly at random from $\mathbb{G}$. Then $\mathcal{B}$ passes $\mathfrak{g}_{g_x}$ to $\mathcal{A}$ and then answers its queries as follows:

- For each query $(\mathfrak{g}_1, \mathfrak{g}_2)$ (except the i_0th query) that $\mathcal{A}$ makes to its generic group oracle, adversary $\mathcal{B}$ proceeds as follows: it looks up in the list $\mathcal{L}$ two tuples of the form $(\mathfrak{g}_1, k_1, l_1)$ and $(\mathfrak{g}_2, k_2, l_2)$. If such tuples cannot be found, then $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ computes $(k, h) = (k_1 - k_2 \bmod q, l_1 - l_2 \bmod q)$ and checks if $k - x \cdot l = k_i - x \cdot l_i \bmod q$ for some tuple $(\mathfrak{g}_i, s_i, h_i) \in \mathcal{L}$. If such a tuple can be found, then $\mathcal{B}$ returns $\mathfrak{g}_i$ to $\mathcal{A}$. Else, $\mathcal{B}$ selects random $\mathfrak{g} \xleftarrow{\$} \mathbb{G}$, adds the tuple $(\mathfrak{g}, s, h)$ to $\mathcal{L}$, and returns $\mathfrak{g}$ to $\mathcal{A}$.
- When $\mathcal{A}$ makes its i_0's query to its generic group oracle, $(\mathfrak{g}_1, \mathfrak{g}_2)$, adversary $\mathcal{B}$ searches $\mathcal{L}$ for two tuples $(\mathfrak{g}_1, k_1, l_1)$ and $(\mathfrak{g}_2, k_2, l_2)$. If these are not found, the $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ computes $(k_{i_0}, l_{i_0}) = (k_1 - k_2 \bmod q, l_1 - l_2 \bmod q)$ and sends l_{i_0} to its environment (recall that $\mathcal{B}$ is against the rpp of H, and so we are using $l_{i_0} = l_1 - l_2 \bmod q$ as the h required as the challenge to the simulator in the rpp game). It then receives an element ξ_{i_0} selected at random from $\{0,1\}^d$. Adversary $\mathcal{B}$ now tries to invert f on the element ξ_{i_0} to obtain $\mathfrak{g}_{i_0} \in \mathbb{G}$, which it can do with probability δ. If this does not work then $\mathcal{B}$ aborts, otherwise $\mathcal{B}$ adds $(\mathfrak{g}, k_{i_0}, l_{i_0})$ to $\mathcal{L}$, and returns $\mathfrak{g}$ to $\mathcal{A}$.
- For each query m that $\mathcal{A}$ makes to its signing oracle adversary $\mathcal{B}$ computes a signature on the m as follows. First $\mathcal{B}$ selects random $r \in \mathbb{Z}_q$, computes $R = r \cdot g$, selects $\mathfrak{g}_R \xleftarrow{\$} \mathbb{G}$, computes $h = \mathsf{H}(f(\mathfrak{g}_R)||m)$, and $s = r + x \cdot h \bmod q$. It then returns (s, h) to $\mathcal{A}$, and adds $(\mathfrak{g}_R, s, h)$ to $\mathcal{L}$.

When $\mathcal{A}$ outputs its attempted forgery $(m^*, (s^*, h^*))$, adversary $\mathcal{B}$ checks whether $h^* = k_{i_0}$. If this is not the case, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ outputs m^* to its environment. Notice that if $(m^*, (s^*, h^*))$ is a valid forgery, then we have that $\mathsf{H}(f(\mathfrak{g}_{R_{i_0}})||m^*) = k_{i_0}$, and therefore $\mathcal{B}$ wins in the rpp game.

It is immediate that the simulation that $\mathcal{B}$ offers to $\mathcal{A}$ is perfect, provided that event $\mathsf{NoEqual}$ occurs.

Then whenever event $\mathsf{MatchGG}$ occurs, $\mathcal{B}$ guesses successfully i_0, and $\mathcal{B}$ does not abort its execution, (which happens with probability δ, the probability that $\mathcal{B}$ can invert f on ξ_{i_0}), adversary $\mathcal{B}$ wins in the game for random-prefix preimage resistance. That is:

$$\frac{\delta}{q_G} \cdot \Pr\left[\mathsf{MatchGG} \mid \mathsf{NoEqual}\right] \leq \mathsf{Adv}_{\mathsf{H}}^{\mathsf{rpp}[D]}(\mathcal{B}) \tag{5}$$

Furthermore, if $\mathcal{A}$ runs in time $t_{\mathcal{A}}$ then $\mathcal{B}$ runs in time $t_{\mathcal{A}} + t'$, where t' is the time spent in maintaining the environment of $\mathcal{A}$.

Bounding the probability of event $\mathsf{MatchSig} \vee \mathsf{Match}_g \vee \mathsf{Match}_{g_x}$. We construct an adversary $\mathcal{C}$ which runs $\mathcal{A}$ as a subroutine and simulates its environment. The adversary is such that whenever one of the events $\mathsf{MatchSig}, \mathsf{Match}_g$, or Match_{g_x} occurs in the simulated execution of $\mathcal{A}$, then $\mathcal{C}$ wins in the rpsp game.

The precise details of how $\mathcal{C}$ works are as follows. Adversary $\mathcal{C}$ selects uniformly at random

$$i_0 \xleftarrow{\$} \{-1, 0, 1, 2, \ldots, q_S\}, g \xleftarrow{\$} G, \text{ and } x \xleftarrow{\$} \mathbb{Z}_q.$$

The adversary maintains the list $\mathcal{L}$ associated to the execution of adversary $\mathcal{A}$. The list is initialised as follows:

- If $i_0 = -1$ then $\mathcal{C}$ sends 0 to its environment and receives a string ξ selected uniformly at random from $\{0,1\}^d$. It computes $\mathfrak{g} = f^{-1}(\xi)$ and initialises the list $\mathcal{L}$ with tuples $(\mathfrak{g}_g = \mathfrak{g}, 1, 0)$, $(\mathfrak{g}_{g_x}, 0, q-1)$, where $\mathfrak{g}_{g_x}$ is selected at random from $\mathbb{G}$.
- If $i_0 = 0$ then $\mathcal{C}$ sends $q-1$ to its environment and receives a string ξ selected uniformly at random from $\{0,1\}^d$. It computes $\mathfrak{g} = f^{-1}(\xi)$. The list $\mathcal{L}$ is initialised with $(\mathfrak{g}_g, 1, 0)$, $(\mathfrak{g}_{g_x} = \mathfrak{g}, 0, q-1)$, where $\mathfrak{g}_g$ is selected at random from $\mathbb{G}$.
- If $i_0 \in \{1, 2, \ldots, q_S\}$ then $\mathcal{L}$ is initialised with $(\mathfrak{g}_g, 1, 0)$ and $(\mathfrak{g}_{g_x}, 0, q-1)$, with $\mathfrak{g}_g$ and $\mathfrak{g}_{g_x}$ selected at random from $\mathbb{G}$.

Then $\mathcal{C}$ executes $\mathcal{A}$ as a subroutine, and answers its queries as follows:

- For each query $(\mathfrak{g}_1, \mathfrak{g}_2)$ to the generic group oracle, $\mathcal{C}$ searches $\mathcal{L}$ for two tuples $(\mathfrak{g}_1, l_1, k_1)$ and $(\mathfrak{g}_2, l_2, k_2)$. If such pairs are not found, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ computes $(l, k) = (l_1 - l_2 \bmod q, k_1 - k_2 \bmod q)$ and for each entry $(\mathfrak{g}_i, l_i, k_i)$ in $\mathcal{L}$ algorithm $\mathcal{C}$ verifies if $k_i - x \cdot l_i = k - kl \bmod q$. If this is the case, then it returns $\mathfrak{g}_i$ to $\mathcal{A}$. Else, it selects a random $\mathfrak{g} \in \mathbb{G}$, adds $(\mathfrak{g}, k, l)$ to $\mathcal{L}$ and returns $\mathfrak{g}$ to $\mathcal{A}$.

– For all $i \neq i_0$, when $\mathcal{A}$ makes the i'th signing query m_i, adversary $\mathcal{C}$ responds as follows. It selects random $r \xleftarrow{\$} \mathbb{Z}_q$, computes $R \leftarrow r \cdot g$, $\mathfrak{g}_R \xleftarrow{\$} \mathbb{G}$, $\xi_R = f(\mathfrak{g}_R)$, $h = \mathsf{H}(\xi_R||m_i)$, and $s \leftarrow r + x \cdot h \bmod q$. Adversary $\mathcal{C}$ then adds $(\mathfrak{g}_R, s, h)$ to $\mathcal{L}$ and returns (s, h) to the adversary.
– For the i_0'th signing query of $\mathcal{A}$, m_{i_0} (here $i_0 \in \{1, 2, \ldots, q_S\}$), $\mathcal{C}$ outputs m_{i_0} to its own environment, and receives a random $\xi \in \{0,1\}^d$ (recall that $\mathcal{C}$ is against rpsp). Algorithm $\mathcal{C}$ computes $\mathfrak{g} = f^{-1}(\xi)$, if $\mathfrak{g} = \perp$ then algorithm $\mathcal{C}$ terminates. Then, $\mathcal{C}$ selects random a $r_{i_0} \in \mathbb{Z}_q$, and computes $R_{i_0} = r_{i_0} \cdot g$, $h_{i_0} \leftarrow \mathsf{H}(\xi||m)$ and $s_{i_0} = r_{i_0} + x \cdot h \bmod q$. It adds $(\mathfrak{g}, s_{i_0}, h_{i_0})$ to $\mathcal{L}$, and returns (s, h) to $\mathcal{A}$.

At some point $\mathcal{A}$ outputs its attempted forgery $(m^*, (s^*, h^*))$. If at this point we have $(s^*, h^*) \notin \{(1,0), (0, q-1), (s_{i_0}, h_{i_0})\}$ then $\mathcal{C}$ aborts its execution. Otherwise, $\mathcal{C}$ outputs m^*.

Analysis of adversary $\mathcal{C}$. It is immediate that the simulation that $\mathcal{C}$ offers to $\mathcal{A}$ is perfect, provided that event $\mathsf{NoEqual}$ occurs. If event Match_g occurs (i.e. the forgery output by $\mathcal{A}$ is such that $(s^*, h^*) = (1, 0)$), and $i_0 = -1$ (this happens with probability $\frac{1}{q_S+2}$), then we have that $\mathsf{H}(\xi||m^*) = 0$, so $\mathcal{C}$ wins in the game against rpsp. Similarly, if Match_{g_x} occurs, and $i_0 = 0$ (this happens with probability $\frac{1}{q_S+2}$), then $\mathsf{H}(\xi||m^*) = q - 1$, so $\mathcal{C}$ wins in the game against rpsp. Finally, if $\mathsf{MatchSig}$ occurs, (i.e. the forgery output by $\mathcal{A}$ is such that $(s^*, h^*) = (s_{i_0^*}, h_{i_0^*})$ for some $i_0^* \in \{1, 2, \ldots, q_S\}$ and $i_0 = i_0^*$ (this happens with probability $\frac{1}{q_S+2}$), then $\mathsf{H}(\xi_R||m^*) = h^* = h$. Provided that $\mathcal{C}$ does not abort its execution (which happens with probability δ, the probability that $\mathcal{C}$ can invert f on the input it receives from its environment), then $\mathcal{C}$ wins in its own game. Formally, we have that:

$$\frac{\delta}{q_S + 2} \cdot \Pr\left[\mathsf{Match}_g \vee \mathsf{Match} \vee \mathsf{MatchSig} \mid \mathsf{NoEqual}\right] \leq \mathsf{Adv}_{\mathsf{H}}^{\mathsf{rpsp}[D]}(\mathcal{C}) \qquad (6)$$

Furthermore, if $\mathcal{A}$ runs in time $t_{\mathcal{A}}$, then $\mathcal{C}$ runs in time $t_{\mathcal{A}} + t''$, where t'' is the time taken to maintain the environment of $\mathcal{A}$.

Putting the results together. Since the events Match_g, Match_{g_x}, $\mathsf{MatchSig}$, and $\mathsf{MatchGG}$ form a partition of the event $\mathcal{A}$ wins, from Equations (5), and (6) we have that:

$$\begin{aligned}\Pr\left[\mathcal{A}\ \mathsf{wins} \mid \mathsf{NoEqual}\right] &= \Pr\left[\mathsf{Match}_g \vee \mathsf{Match}_{g_x} \vee \mathsf{MatchSig} \mid \mathsf{NoEqual}\right] \\ &\quad + \Pr\left[\mathsf{MatchGG} \mid \mathsf{NoEqual}\right] \\ &\leq \frac{q_G}{\delta} \cdot \mathsf{Adv}_{\mathsf{H}}^{\mathsf{rpp}[D]}(\mathcal{B}) + \frac{q_S + 2}{\delta} \cdot \mathsf{Adv}_{\mathsf{H}}^{\mathsf{rpsp}[D]}(\mathcal{C}).\end{aligned}$$

and therefore:

$$\mathsf{Adv}_{\mathsf{Sch}[\mathsf{H}]}^{\mathsf{uf\text{-}cma}}(\mathcal{A}) \leq \frac{q_G}{\delta} \cdot \epsilon_{\mathsf{rpp}} + \frac{q_S + 2}{\delta} \cdot \epsilon_{\mathsf{rpsp}} + O\left(\frac{(q_S + q_G)^2}{q}\right)$$

provided that $t_{\mathcal{A}} + t' \leq t_{\mathsf{rpsp}}$, and $t_{\mathcal{A}} + t'' \leq t_{\mathsf{rpp}}$.

7 About the Tightness of the Reduction

We end by discussing the tightness of our security reduction and the implications to the choice of security parameters. First, we note that the factors $1/\delta$ in Equation (2) disappear when security is proved under the $\mathsf{rpp}[D]$ and $\mathsf{rpsp}[D]$ assumptions for $D = f(\tau(G))$. More importantly however, Equation (2) loses a factor of q_G and q_S in the reduction to the rpp and rpsp properties, respectively. If we assume that an evaluation of H costs one time unit, then $\epsilon_{\mathsf{rpp}} \leq t/2^n$ and $\epsilon_{\mathsf{rpsp}} \leq t/2^n$ are reasonable bounds for hash functions with n output bits. If we additionally assume that $q_G \approx q_S \approx t$, then we see that Equation (2) takes the form

$$\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{Sch}[\mathsf{H}]}(\mathcal{A}) \;\leq\; O\left(\frac{t^2}{2^n} + \frac{t^2}{q}\right).$$

This implies that to obtain b bits of security, one needs to set $n = 2b$ and $s = \log_2 q = 2b$. This is very tight with respect to the size of q, but not with respect to our desired value of n. The random-oracle analysis via the forking lemma, on the other hand, comes to the exact opposite conclusion. Taking $\epsilon_{\mathsf{dlog}} \leq t^2/q$ and $q_{\mathrm{H}} \approx q_{\mathrm{S}} \approx t$, Equation (1) yields an inequality of the form

$$\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{Sch}[\mathsf{H}]}(\mathcal{A}) \;\leq\; O\left(\sqrt{\frac{t^3}{q}} + \frac{t}{2^n} + \frac{t^2}{q}\right),$$

which advocates using $n = b$ and $s = \log_2 q = 3b$. This is tight with respect to n, but not with respect to q.

So which values to choose? Well, it depends on how much one cares about tightness. The fact that in the real world Schnorr signatures rarely come in inflated group sizes seems to indicate that practitioners see tightness as a second-order problem. In support of their view, no attacks have been found that match the poor bounds of the random-oracle proof. Moreover, the hardness gap between forging signatures and computing discrete logarithms does *not* arise in our generic-group analysis, indicating that the loose reduction in the random-oracle model is just an anomaly of the proof in this setting, rather than an intrinsic difference in hardness.

The situation for the hash function is the exact mirror image of that for the group size: it seems hard to come up with an attack that matches the bounds of the generic-group proof, and the random-oracle proof supports short hashes without any problem. So if one has been using Schnorr signatures with normal group sizes, then there is no reason why one shouldn't use short hash sizes as well.

A more conservative approach would be to take the loose reductions into account and stick to the worst-case bounds from both proofs, meaning $3b$-bit groups in combination with $2b$-bit hashes. We note though that even in this case the Schnorr scheme is to be preferred over EC-DSA, as the latter lacks a security proof in the random-oracle model with respect to a standard assumption. There is a proof of EC-DSA in the random oracle model with respect to the semi-logarithm assumption [Bro05][Theorem II.10] however. Whilst in the random

oracle model Schnorr signatures are secure with respect to the discrete logarithm assumption [PS00].

Another issue with taking short hashes is the vulnerability to so called "duplicate signatures" [SPMLS02]. For EC-DSA, duplicate signatures (i.e. one signature which can sign two messages) can be created by the signer at the point of creation of his public/private key pair (i.e. the two target messages need to be known at the key generation stage). When revealing the resulting duplicate signatures the private key is revealed. A similar result holds for Schnorr signatures.

However, if one uses short hashes in Schnorr signatures then a dishonest signer can produce duplicate signatures after the key generation stage by creating $O(2^{n/2})$ signatures, by essentially finding a collision in the hash function. The revelation of the two duplicate signatures will not reveal the underlying private key. Such a result does not contradict the security results above, since the GMR security definition assumes an attackers against an honest signer, and makes no statement about security in the presence of dishonest signers.

Acknowledgments

The authors would like to thank Steven Galbraith and Eike Kiltz for various discussions and comments. The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 418–433. Springer-Verlag, 2002.

[ANPS07] Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-property-preserving iterated hashing: ROX. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 130–146. Springer-Verlag, 2007.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06*, pages 390–399. ACM Press, 2006.

[BPVY00] Ernest F. Brickell, David Pointcheval, Serge Vaudenay, and Moti Yung. Design validations for discrete logarithm based signature schemes. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000*, volume 1751 of *LNCS*, pages 276–292. Springer-Verlag, 2000.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, 1993.

[BR97] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 470–484. Springer-Verlag, 1997.

[BR07] Mihir Bellare and Thomas Ristenpart. Hash functions in the dedicated-key setting: Design choices and MPP transforms. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007*, volume 4596 of *LNCS*, pages 399–410. Springer-Verlag, 2007.

[Bro02] Daniel R. L. Brown. Generic groups, collision resistance, and ECDSA. Cryptology ePrint Archive, Report 2002/026, 2002.

[Bro05] Daniel R. L. Brown. On the provable security of ECDSA. In *Advances in Elliptic Curve Cryptography*, pages 21–40. Cambridge University Press, 2005.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *30th ACM STOC*, pages 209–218. ACM Press, 1998.

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 416–427. Springer-Verlag, 1990.

[Den02] Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 100–109. Springer-Verlag, 2002.

[Fis00] Marc Fischlin. A note on security proofs in the generic model. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 458–469. Springer-Verlag, 2000.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, 1987.

[GMR88] 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.

[HK06] Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 41–59. Springer-Verlag, 2006.

[JMV01] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[KK06] John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 183–200. Springer-Verlag, 2006.

[KM07] Neal Koblitz and Alfred J. Menezes. Another look at "provable security". *Journal of Cryptology*, 20(1):3–37, January 2007.

[Luc05] Stefan Lucks. A failure-friendly design principle for hash functions. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 474–494. Springer-Verlag, 2005.

[Mer90] Ralph C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 428–446. Springer-Verlag, 1990.

[Mir01] Ilya Mironov. Hash functions: From Merkle-Damgård to Shoup. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 166–181. Springer-Verlag, 2001.

[MW98] Ueli M. Maurer and Stefan Wolf. Lower bounds on generic algorithms in groups. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 72–84. Springer-Verlag, 1998.

[Nat94] National Institute for Standards and Technology. Digital signature standard (DSS). Federal Information Processing Standards Publication 186, November 1994.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *21st ACM STOC*, pages 33–43. ACM Press, 1989.

[OO98] Kazuo Ohta and Tatsuaki Okamoto. On concrete security treatment of signatures derived from identification. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 354–369. Springer-Verlag, 1998.

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

[PV05] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 1–20. Springer-Verlag, 2005.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 371–388. Springer-Verlag, 2004.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer-Verlag, 1990.

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

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer-Verlag, 1997.

[SPMLS02] Jacques Stern, David Pointcheval, John Malone-Lee, and Nigel P. Smart. Flaws in applying proof methodologies to signature schemes. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 93–110. Springer-Verlag, 2002.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 19–35. Springer-Verlag, 2005.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 17–36. Springer-Verlag, 2005.