Short Signatures From Weaker Assumptions

Dennis Hofheinz[1], Tibor Jager[2], and Eike Kiltz*[2]

[1]Institut für Kryptographie und Sicherheit, Karlsruhe Institute of Technology, Germany,
Dennis.Hofheinz@kit.edu
[2]Horst-Görtz Institute for IT Security, Ruhr-Universität Bochum, Germany,
{tibor.jager,eike.kiltz}@rub.de

Abstract

We provide constructions of $(m, 1)$-programmable hash functions (PHFs) for $m \geq 2$. Mimicking certain programmability properties of random oracles, PHFs can, e.g., be plugged into the generic constructions by Hofheinz and Kiltz (J. Cryptol. 2011) to yield digital signatures schemes from the strong RSA and strong q-Diffie-Hellman assumptions. As another application of PHFs, we propose new and efficient constructions of digital signature schemes from weaker assumptions, i.e., from the (standard, non-strong) RSA and the (standard, non-strong) q-Diffie-Hellman assumptions.

The resulting signature schemes offer interesting tradeoffs between efficiency/signature length and the size of the public-keys. For example, our q-Diffie-Hellman signatures can be as short as 200 bits; the signing algorithm of our Strong RSA signature scheme can be as efficient as the one in RSA full domain hash; compared to previous constructions, our RSA signatures are shorter (by a factor of roughly 2) and we obtain a considerable efficiency improvement (by an even larger factor). All our constructions are in the standard model, i.e., without random oracles.

Keywords: digital signatures, RSA assumption, q-DH assumption, programmable hash functions.

1 Introduction

Digital Signatures are one of the most fundamental cryptographic primitives. They are used as a building block in numerous high-level cryptographic protocols. Practical signature schemes are known whose security is based on relatively mild intractability assumptions such as the RSA [7] or the (bilinear) Computational Diffie-Hellman (CDH) assumption [14]. However, their security can only be proved in the random oracle model [6] with all its limitations (e.g., [18, 27]).

Standard Model Signatures. Signature schemes in the standard model (i.e., without using random oracles) are often considerably less efficient or based on much stronger assumptions. While tree-based signature schemes can be built from any one-way function [50], these constructions are far from practical. On the other hand, "Hash-and-sign" signatures are considerably more efficient, but the most efficient of these schemes rely on specific "strong" number theoretic hardness assumptions

*Funded by a Sofja Kovalevskaja Award of the Alexander von Humboldt Foundation and the German Federal Ministry for Education and Research.

which we call Strong q-assumptions.[1] In Strong q-assumptions, an adversary is provided with a polynomial number of random "solved instances" and has to compute a new solved instance *of its choice*. For example, the schemes in [24, 31, 30, 37, 39] are based on the Strong (or, Flexible) RSA assumption and the schemes in [12, 39] are based on the Strong q-Diffie-Hellman assumption. Both assumptions are considerably stronger than their "non-strong" counterparts (i.e., the q-Diffie-Hellman and the RSA assumptions, respectively), in which an adversary has to solve a *given, fixed instance*. (The exact difference between strong and non-strong assumptions will be discussed in Appendix B.)

Programmable Hash Functions. In order to mimic certain "programmability properties" of random oracles, Hofheinz and Kiltz [39] introduced the combinatorial concept of programmable hash functions (PHF). (See Section 3 for a formal definition.) Among a number of other applications, they used PHFs as a building block for efficient and short hash-and-sign signatures based on the Strong RSA and the Strong q-Diffie-Hellman assumptions. Concretely, signatures in the Strong RSA based HK signature scheme $\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}]$ are of the form $\mathsf{sig}(M) = (\mathsf{H}(M)^{1/e} \bmod N, e)$, where $N = pq$ is a public RSA modulus, $\mathsf{H}(\cdot)$ is a $(m, 1)$-PHF, and e is a *short* prime (chosen at random during the signing process). A given HK signature (σ, e) is verified by checking if $\sigma^e = \mathsf{H}(M) \bmod N$. The efficiency of the HK signature scheme is dominated by the time needed to generate the prime e, which (as shown in [39]) depends on the parameter m of the PHF: the bigger m, the smaller e and consequently the more efficient is the signing process. Over bilinear groups there exists a similar construction, $\mathsf{Sig}_{\mathsf{S}\text{-}q\text{-}\mathsf{DH}}[\mathsf{H}]$, whose security is based on the Strong q-DH assumption. The main disadvantages of HK signatures is that their security relies on Strong assumptions, i.e., on the Strong RSA (Strong q-DH) and not on the standard RSA (q-DH) assumption.

RSA signatures. As a step towards practical signatures from the (standard) RSA assumption, Hohenberger and Waters [41, 40] proposed the first hash-and-sign signature scheme (HW signatures) whose security is based on the RSA assumption. HW signatures are computed as $\mathsf{sig}(M) = g^{1/\mathsf{P}(M)} \bmod N$, where $g \in \mathbb{Z}_N^*$ is a public element and $\mathsf{P}(M) = e_1 \cdot \ldots \cdot e_{|M|}$ is the product of $|M|$ distinct primes. Here each prime e_i is uniquely determined by the i-bit prefix $M_{|i}$ of the message M, and for each generation of e_i a number of primality tests have to be executed which is the dominant running time of signing (and verifying). The above signature scheme is only weakly secure under the RSA assumption, and a chameleon hash has to be used to make it fully secure, thereby doubling the signature size to two elements from $\mathbb{Z}_N$ and adding $\approx$ 2kbit to the public-key size [40]. The main disadvantage of HW signatures is, however, the generation and testing of the $|M|$ primes $e_1, \ldots, e_{|M|}$ necessary to compute the hash function $\mathsf{P}(M)$. Concretely, for $k = 80$ bits security, HW signatures need to generate $|M| = 160$ random primes for the signing process.

1.1 Summary of our Contributions

As the main technical contribution we propose several new constructions of $(m, 1)$-PHFs for any $m \geq 1$. In particular, we solve the open problem posed in [39] of constructing deterministic $(m, 1)$-PHFs for $m > 2$. Even though our main applications are digital signatures we remark that PHFs are a very general framework for designing and analyzing cryptographic protocols in the Diffie-Hellman and RSA setting. For example, in [39], it was shown that PHFs imply collision-resistant

[1] There are exceptions, e.g., by Waters [54] (CDH assumption in bilinear groups), Hohenberger and Waters [41], and the lattice-based schemes [19, 15] (SIS assumption). However, these are not among the most efficient "Hash-and-sign"-type schemes.

hash functions and lead to elegant and simple proofs of Waters' IBE and signature schemes [54] and its countless variants (e.g., [16, 8]). More importantly, a large body of cryptographic protocols with security in the standard model are using — implicitly or explicitly — the partitioning trick that is formalized in PHFs. To mention only a few examples, this ranges from collision-resistant hashing [21, 4], digital signature schemes [13, 54] (also in various flavors [49, 52, 9]), chosen-ciphertext secure encryption [16, 42, 36, 38, 15], identity-based encryption [10, 11, 43, 19, 1], attribute-based encryption [51] to symmetric authentication [44]. We expect that our new PHF constructions can also be applied to some of the mentioned applications.

We also show how to use our new $(m, 1)$-PHFs for generic constructions of short yet efficient hash-and-sign signatures whose security is based on weaker hardness assumptions: the q-DH and the RSA assumption. Whereas our q-DH schemes $\mathsf{Sig}_{q\text{-}\mathsf{DH}}[\mathsf{H}]$ are (to the best of our knowledge) the first hash-and-sign schemes from this assumption, our RSA schemes $\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}]$ and $\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}]$ are conceptually different from HW signatures and we obtain a considerable efficiency improvement. A large number of new signature schemes with different tradeoffs can be derived by combining the generic signature schemes with PRFs. An overview of the efficiency of some resulting schemes and a comparison with existing schemes from [24, 31, 12, 39, 41] is provided in Table 1. Our new schemes offer different tradeoffs between signature size, efficiency, and public-key size. The bigger the parameter m in the $(m, 1)$-PHF, the larger the public-key size, the shorter the signatures. To obtain extremely short and/or efficient signatures, the size of the public key can get quite large. Concretely, with a public-key of size 26mbit we obtain 200 bit signatures from the (Strong) q-DH assumption. These are the shortest knwon standard-model digital signatures in bilinear groups. Remarkably, $\mathsf{Sig}_{\mathsf{SRSA}}[\mathsf{H}_{\mathsf{cfs}}]$ which instatiates the Strong RSA signatures from [39] with our new $(m, 1)$-PHF $\mathsf{H}_{\mathsf{cfs}}$ for $m \geq 6$, results in a hash-and-sign signature scheme where the signing procedure is dominated by one single modular exponentiation. This is the first RSA-based signature scheme whose signing complexity is not dominated by generating random primes.[2] Hence signing is essentially as efficient as RSA full-domain-hash [7] with the drawback of a huge public-key.

While these short signatures are mostly of theoretical interest and contribute to the problem of determining concrete bounds on the size of standard-model signatures, we think that in certain applications even a large public-key is tolerable. In particular, our public key sizes are still comparable to the ones of recently proposed lattice-based signatures [47, 32, 19, 15].

We note furthermore, that it is possible to apply efficiency improvements from [41] to our RSA-based schemes as well. This allows us to reduce the number of primality tests required for signing and verification sigificantly. More precisely, it is possible to transform each signature scheme requiring λ primality tests into a scheme which requires only λ/c primality tests, at the cost of loosing a factor of 2^{-c} in the security reduction. For example, $\mathsf{Sig}^{*}_{\mathsf{RSA}}[\mathsf{H}_{\mathsf{Weak}}]^{\S}$ with $m = 11$ and $c = 40$ is a RSA-based signature scheme which requires only a *single* primality test for signing and verification, at the cost of loosing a factor of 2^{-40} in the security reduction.

1.2 Details of our Contributions

New Programmable Hash Functions. Our main technical contribution to obtain shorter signatures are several new constructions of $(m, 1)$-PHFs for $m \geq 2$ (cf. Table 2 in Section 3). Using

[2] Since the complexity of finding a random μ-bit prime with error 2^{-k} is $O(k\mu^4)$, we expect that for $\mu \approx 60$ (or, equivalently, using $\mathsf{H}_{\mathsf{cfs}}$ with $m \geq 6$) a full exponentiation modulo a 1024-bit integer become roughly as expensive as generating a random μ-bit prime.

Signature scheme	Assumption	Signature Size (bits)	Efficiency	Public key size (bits)
Waters [54]	CDH	$2 \times \lvert\mathbb{G}\rvert = 320$	$2 \times \mathsf{Exp}$	$l \times \lvert\mathbb{G}_1\rvert = 26\text{k}$
Boneh-Boyen [12]	Strong q-DH	$\lvert\mathbb{G}_1\rvert + \lvert\mathbb{Z}_p\rvert = 320$	$1 \times \mathsf{Exp}$	$2 \times \lvert\mathbb{G}_2\rvert = 640$
$\mathsf{Sig}_{\mathsf{S}\text{-}q\text{-}\mathsf{DH}}[\mathsf{H}_{\mathsf{Wat}}]$ [39]	Strong q-DH	$\lvert\mathbb{G}\rvert + \lvert s\rvert = 230$	$1 \times \mathsf{Exp}$	$l \times \lvert\mathbb{G}_1\rvert = 26\text{k}$
$\mathsf{Sig}_{\mathsf{S}\text{-}q\text{-}\mathsf{DH}}[\mathsf{H}_{\mathsf{cfs}}]$ $(m{=}8)$	Strong q-DH	$\lvert\mathbb{G}_1\rvert + \lvert s\rvert = 200$	$1 \times \mathsf{Exp}$	$16m^2 l \times \lvert\mathbb{G}_1\rvert = 26\text{m}$
$\mathsf{Sig}_{q\text{-}\mathsf{DH}}[\mathsf{H}_{\mathsf{Wat}}, \mathsf{H}_{\mathsf{Wat}}]$ $(m{=}2)$	q-DH	$\lvert\mathbb{G}_1\rvert + \lvert s\rvert = 230$	$1 \times \mathsf{Exp}$	$l \times \lvert\mathbb{G}_1\rvert + \lvert s\rvert \times \lvert\mathbb{G}_2\rvert = 48\text{k}$
$\mathsf{Sig}_{q\text{-}\mathsf{DH}}[\mathsf{H}_{\mathsf{cfs}}, \mathsf{H}_{\mathsf{Wat}}]$ $(m{=}8)$	q-DH	$\lvert\mathbb{G}_1\rvert + \lvert s\rvert = 200$	$1 \times \mathsf{Exp}$	$16m^2 l \times \lvert\mathbb{G}_1\rvert + \lvert s\rvert \times \lvert\mathbb{G}_2\rvert = 26\text{m}$
Cramer-Shoup [24]	Strong RSA	$2 \times \lvert\mathbb{Z}_N\rvert + \lvert e\rvert = 2208$	$1 \times \mathsf{P}_{160}$	$3 \times \lvert\mathbb{Z}_N\rvert + \lvert e\rvert = 3\text{k}$
Gennaro et. al.* [31]	Strong RSA	$2 \times \lvert\mathbb{Z}_N\rvert = 2048$	$1 \times \mathsf{P}_{160}$	$\lvert\mathbb{Z}_N\rvert + \lvert pk_{\mathsf{CH}}\rvert = 3\text{k}$
$\mathsf{Sig}_{\mathsf{SRSA}}[\mathsf{H}_{\mathsf{Wat}}]$ [39]	Strong RSA	$\lvert\mathbb{Z}_N\rvert + \lvert e\rvert = 1104$	$1 \times \mathsf{P}_{80}$	$l \times \lvert\mathbb{Z}_N\rvert = 128\text{k}$
$\mathsf{Sig}_{\mathsf{SRSA}}[\mathsf{H}_{\mathsf{cfs}}]$ $(m{=}6)$	Strong RSA	$\lvert\mathbb{Z}_N\rvert + \lvert s\rvert = 1068$	$\approx 1 \times \mathsf{Exp}$	$16m^2 l \times \lvert\mathbb{Z}_N\rvert = 94\text{m}$
$\mathsf{Sig}^*_{\mathsf{SRSA}}[\mathsf{H}_{\mathsf{Weak}}]$ $(m{=}6)$	Strong RSA	$2 \times \lvert\mathbb{Z}_N\rvert = 2048$	$\approx 2 \times \mathsf{Exp}$	$7 \times \lvert\mathbb{Z}_N\rvert + \lvert pk_{\mathsf{CH}}\rvert = 9\text{k}$
Hohenberger-Waters* [41]	RSA	$2 \times \lvert\mathbb{Z}_N\rvert = 2048$	$160 \times \mathsf{P}_{1024}$	$\lvert\mathbb{Z}_N\rvert + \lvert pk_{\mathsf{CH}}\rvert = 3\text{k}$
$\mathsf{Sig}^*_{\mathsf{RSA}}[\mathsf{H}_{\mathsf{Weak}}]$ $(m{=}2)$	RSA	$2 \times \lvert\mathbb{Z}_N\rvert = 2048$	$70 \times \mathsf{P}_{1024}$	$3 \times \lvert\mathbb{Z}_N\rvert + \lvert pk_{\mathsf{CH}}\rvert = 5\text{k}$
$\mathsf{Sig}^*_{\mathsf{RSA}}[\mathsf{H}_{\mathsf{Weak}}]$ $(m{=}4)$	RSA	$2 \times \lvert\mathbb{Z}_N\rvert = 2048$	$50 \times \mathsf{P}_{1024}$	$5 \times \lvert\mathbb{Z}_N\rvert + \lvert pk_{\mathsf{CH}}\rvert = 7\text{k}$
$\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}_{\mathsf{Wat}}]$ $(m{=}2)$	RSA	$\lvert\mathbb{Z}_N\rvert + \lvert s\rvert = 1094$	$70 \times \mathsf{P}_{1024}$	$l \times \lvert\mathbb{Z}_N\rvert = 128\text{k}$
$\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}_{\mathsf{rand}}]$ $(m{=}4)$	RSA	$\lvert\mathbb{Z}_N\rvert + \lvert s\rvert + \lvert r\rvert = 1214$	$50 \times \mathsf{P}_{1024}$	$2m^2 \times \lvert\mathbb{Z}_N\rvert = 32\text{k}$
$\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}_{\mathsf{cfs}}]$ $(m{=}4)$	RSA	$\lvert\mathbb{Z}_N\rvert + \lvert s\rvert = 1074$	$50 \times \mathsf{P}_{1024}$	$16m^2 l \times \lvert\mathbb{Z}_N\rvert = 40\text{m}$
$\mathsf{Sig}^*_{\mathsf{RSA}}[\mathsf{H}_{\mathsf{Weak}}]^{\S}$ $(m{=}11)$	RSA	$2 \times \lvert\mathbb{Z}_N\rvert = 2048$	$1 \times \mathsf{P}_{1024}$	$12 \times \lvert\mathbb{Z}_N\rvert + \lvert pk_{\mathsf{CH}}\rvert = 14\text{k}$

*The RSA-based chameleon hash function from [40] was used (adding $1 \times \lvert\mathbb{Z}_N\rvert$ to signature size).
§Security reduction loses an additional factor of 2^{40}.

Table 1: Signature sizes of different schemes. Rows with grey background indicate new results from this paper. The chosen parameters provide unforgeability with $k = 80$ bits of security after revealing maximally $q = 2^{30}$ signatures. RSA signatures are instantiated with a modulus of $|N| = 1024$ bits, Bilinear signatures in asymmetric pairings using a BN curve [3] with $\log p = 160$ bits. We assume that elements in $\mathbb{G}_1$ can be represented by $|\mathbb{G}_1| = 160$ bits, while an element $\mathbb{G}_2$ by $|\mathbb{G}_2| = 320$ bits. The description of the bilinear group/modulus N is not counted in the public key. We assume l-bit messages with $l = 2k = 160$ in order to provide k bits of security (to sign longer messages, we can apply a collision-resistant hash function first). The efficiency column counts the dominant operations for signing. For Bilinear and RSA signatures this counts the number of modular exponentiations, for RSA signatures $k \times \mathsf{P}_\mu$ counts the number of random μ-bit primes that need to be generated to evaluate function $\mathsf{P}(\cdot)$. (For $\mu \gg 60$, $1 \times \mathsf{P}_\mu$ takes more time than $1 \times \mathsf{Exp}$.)

cover-free sets, we construct a deterministic $(m, 1)$-PHF $\mathsf{H}_{\mathsf{cfs}}$ with public parameters of $O(km^2)$ group elements. This solves the problem from [39] of constructing deterministic $(m, 1)$-PHFs for $m > 2$. We remark that cover-free sets were already used in [26, 35, 23] to construct identity-based encryption schemes. Furthermore, we propose a randomized $(m, 1)$-PHF $\mathsf{H}_{\mathsf{rand}}$ with public parameters of $O(m^2)$ group elements and small randomness space. Finally, we construct a weakly secure deterministic $(m, 1)$-PHF $\mathsf{H}_{\mathsf{Weak}}$ with public parameters of m group elements. The latter PHF already appeared implicitly in the context of identity/attribute-based encryption [20, 51] (generalizing [10]). Weakly secure PHFs only yield weakly secure signature schemes that need to be "upgraded" to fully secure schemes using a chameleon hash function.

RSA Signatures. Our new RSA signatures $\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}]$ are of the form

$$\mathsf{sig}(M) = (\mathsf{H}(M)^{1/\mathsf{P}(s)} \bmod N, s), \tag{1}$$

where s is a short random bitstring, $\mathsf{H}(\cdot)$ is a $(m, 1)$-PHF, and $\mathsf{P}(s) := e_1 \cdot \ldots \cdot e_{|s|}$ is the product of $|s|$ primes $e_1, \ldots, e_{|s|}$, where the ith prime is uniquely determined by the ith prefix $s_{|i}$ of the randomness s. (If the PHF H is probabilistic, sig additionally contains a small random bitstring r.) Our security proof is along the lines of [39], but using P enables a reduction to the RSA assumption (Theorem 4.1) in the standard model. The main conceptual novelty is that we apply P to the

randomness s rather than the message M as in HW signatures. Because the values s are relatively small, our scheme is considerably more efficient than that of [41].

Concretely, the length of s is controlled by the PHF parameter m as $|s| = \log q + k/m$, where q is an upper bound on the number of signatures the scheme supports. (See Appendix A for a formal argument.) For $k = 80$ bits security and $q = 2^{30}$ (as recommended in [7]) we can make use of our new constructions of $(m, 1)$-PHFs with $m \geq 2$. For example, with a $(4, 1)$-PHF, the bitstring s can be as small as 50 bits which leads to very small signatures. More importantly, since the function $\mathsf{P}(s)$ only has to generate $|s|$ distinct primes $e_1, \ldots, e_{|s|}$ (compared to $|M| \gg |s|$ primes in HW signatures), the signing and verification algorithms are considerably faster. The drawback of our new signature scheme is that the system parameters of H grow with m.

Bilinear Signatures. Our new q-DH signatures $\mathsf{Sig}_{q\text{-}\mathsf{DH}}[\mathsf{H}]$ are of the form

$$\mathsf{sig}(M) = (\mathsf{H}(M)^{1/\mathsf{d}(s)}, s), \tag{2}$$

where again s is a short random bitstring, H is a $(m, 1)$ programmable hash function, and $\mathsf{d}(\cdot)$ is a special (secret) function mapping bitstrings to $\mathbb{Z}_p$. Since $\mathsf{D}(s) := g^{\mathsf{d}(s)}$ can be computed publicly, verification is done by using the properties of the bilinear group. Security is proved under the q-DH assumption in the standard model (Theorem 5.1). Similar to our RSA-based signatures the length of s is controlled by the PHF parameter m. For example, for $m = 8$ we obtain standard-model signatures of size $|\mathbb{G}| + |s| = 160 + 40 = 200$ bits.

Full Domain Signatures. We remark that full-domain hash signature schemes over a homomorphic domain (e.g., RSA-FDH [7] and BLS signatures [14]) instantiated with $(m, 1)$-PHFs provide efficient m-time signature schemes without random oracles. This nicely complements the impossibility results from [27] who show that without the homomorphic property this is not possible. We remark that an instantiation of RSA-FDH as a m-time signature scheme was independently observed in [25].

Proof Techniques and Related Work. Our RSA-based signature scheme represents a combination of techniques from [39] and [41]. Namely, in the basic RSA-based signature scheme from [39], a signature is of the form $(\mathsf{H}(M)^{1/s} \bmod N, s)$ for a prime s. The use of a *programmable* hash function H enables very efficient schemes, whose security however cannot be reduced to the standard (non-strong) RSA problem, since a forged signature $(\mathsf{H}(M)^{1/s^*}, s^*)$ corresponds to an RSA inversion with adversarially chosen exponent s^*. On the other hand, the (basic, weakly secure) signature scheme from [41] is of the form $g^{1/\mathsf{P}(M)} \bmod N$. The special structure of P (which maps a message M to the product of $|M|$ primes) makes it possible to prove security under the standard RSA assumption. However, since P is applied to messages (i.e., 160-bit strings), evaluation of P requires a large number of primality tests. We combine the best of both worlds with signatures of the form $(\mathsf{H}(M)^{1/\mathsf{P}(s)} \bmod N, s)$ for *short* (e.g., 40-bit) random strings s. In contrast to the scheme of [41], this directly yields a fully secure signature scheme, so we do not need a chameleon hash function.

In the security proof of our RSA signatures we distinguish between two types of forgers: type I forgers recycle a value from $\{s_1, \ldots, s_q\}$ for the forgery, where the s_i's are the random bitstrings used for the simulated signatures; type II forgers use a new value $s^* \notin \{s_1, \ldots, s_q\}$ for the forgery and therefore are more difficult to reduce to the RSA assumption. For the reduction of type II forgers to the RSA assumption we can use a clever "prefix-guessing" technique from [41] to embed

the prime e from the RSA challenge in the function $\mathsf{P}(\cdot)$ such that the product $\mathsf{P}(s^*)$ contains e.[3] Similar to the proof of HK signatures [39], the reduction for Type I forgers makes use of the $(m, 1)$ programmability of $\mathsf{H}(\cdot)$.

Strong q-DH signatures from [39] can actually be viewed as our q-DH signatures from (2) instantiated with the special function $\mathsf{d}(s) = x+s$ (where x is part of the secret-key). In our scheme, the leverage to obtain security from q-DH is that the function $\mathsf{D}(s) := g^{\mathsf{d}(s)}$ acts as a $(\mathsf{poly}, 1)$-PHF. That is, $\mathsf{d}(\cdot)$ can be setup such that (with non-negligible probability) $\mathsf{d}(s_i) = x + a(s_i)$ for $a(s_i) \neq 0$ but $\mathsf{d}(s^*) = x$, where $s_1, \ldots, s_q$ is the randomness used for the generated signatures and s^* is the randomness used for the forgery.

1.3 Open Problems

A number of interesting open problems remain. We ask how to construct (deterministic) $(m, 1)$-PHFs for $m \geq 1$ with smaller parameters than the ones from Table 2. Since the constructions of cover free sets are known to be optimal up to a log factor, a new method will be required. Furthermore, obtaining truely practical signatures from the RSA or factoring assumption is still an open problem. In particular, we ask for a construction of hash-and-sign (strong) RSA signatures that do not require the generation of primes at signing.

2 Preliminaries

For $k \in \mathbb{N}$, we write 1^k for the string of k ones, and $[k]$ for $\{1, \ldots, k\}$. Moreover, $|x|$ denotes the length of a bitstring x, while $|S|$ denotes the size of a set S. Further, $s \xleftarrow{\$} S$ denotes the sampling a uniformly random element s of S. For an algorithm $\mathcal{A}$, we write $z \xleftarrow{\$} \mathcal{A}(x, y, \ldots)$ to indicate that $\mathcal{A}$ is a (probabilistic) algorithm that outputs z on input $(x, y, \ldots)$.

2.1 Digital signatures

A digital signature scheme $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ consists of three algorithms. Key generation Gen generates a keypair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$ for a secret signing key sk and a public verification key pk. The signing algorithm Sign inputs a message and the secret signing key, and returns a signature $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$ of the message. The verification algorithm Vfy takes a verification key and a message with corresponding signature as input, and returns $b \leftarrow \mathsf{Vfy}(pk, m, \sigma)$ where $b \in \{\texttt{accept}, \texttt{reject}\}$. We require the usual correctness properties.

Let us recall the *existential unforgeability against chosen message attacks* (EUF-CMA) security experiment [33], played between a challenger and a forger $\mathcal{F}$.

1. The challenger runs Gen to generate a keypair (pk, sk). The forger receives pk as input.
2. The forger may ask the challenger to sign a number of messages. To query the i-th signature, $\mathcal{F}$ submits a message m_i to the challenger. The challenger returns a signature σ_i under sk for this message.
3. The forger outputs a message m^* and signature σ^*.

[3] More precisely, when simulating a type II forger, the values $s_1, \ldots, s_q$ are known in advance to the simulator. Since $s^* \notin \{s_1, \ldots, s_q\}$ there is some prefix $s^*_{|i}$ of s^* that is different from all prefixes of $s_1, \ldots, s_q$. We can guess the smallest such prefix such that the simulator knows $s^*_{|i}$ from the forgery at the beginning. This knowledge can be used to embed e from the RSA challenge in the function $\mathsf{P}(\cdot)$ such that the product $\mathsf{P}(s^*)$ contains e.

$\mathcal{F}$ wins the game, if $\texttt{accept} \leftarrow \mathsf{Vfy}(pk, m^*, \sigma^*)$, that is, σ^* is a valid signature for m^*, and $m^* \neq m_i$ for all i. We say that $\mathcal{F}$ (t, q, ϵ)-breaks the EUF-CMA security of Sig, if $\mathcal{F}$ runs in time t, makes at most q signing queries, and has success probability ϵ. We say that Sig is EUF-CMA secure, or Sig is *fully* secure, if ϵ is negligible for any probabilistic polynomial-time algorithm $\mathcal{F}$.

We also say, that a scheme is *weakly* secure, if it meets the above security definition, but the adversary can not choose the messages to be signed adaptively. Instead it has to commit to a list $m_1, \ldots, m_q$ before seeing the public key. There exist efficient generic techniques to convert a weakly secure signature scheme into a fully secure one, e.g., using chameleon hashes [45].

2.2 Prime Numbers, Factoring, and the RSA Assumption

For $x \in \mathbb{N}$ let $\pi(x)$ denote the number of primes between 0 and x. The following lemma is a direct consequence of Chebyshev's bounds on $\pi(x)$ (see [34], for instance).

Lemma 2.1. $\frac{x}{\log_2 x} < \pi(x) < \frac{2x}{\log_2 x}$

We say that a prime p is a *safe* prime, if $p = 2p' + 1$ and p' is also prime. Let p and q be two randomly chosen $k/2$-bit safe primes, and let $N = pq$. Let $e \in \mathbb{Z}_{\phi(n)}$ be a random integer, relatively prime to $\phi(N)$. We say that an algorithm $\mathcal{A}$ (t, ϵ)-breaks the RSA assumption, if $\mathcal{A}$ runs in time t and

$$\Pr[y^{1/e} \xleftarrow{\$} \mathcal{A}(N, e, y)] \geq \epsilon.$$

We assume that there exists no algorithm that (t, ϵ)-breaks the RSA assumption with polynomial t and non-negligible ϵ.

We denote with QR_N the group of quadratic residues modulo N. The following lemma, which is due to Shamir [53], is useful for the security proof of the generic RSA-based signature scheme described in Section 4.

Lemma 2.2. *There is an efficient algorithm that, on input $y, z \in \mathbb{Z}_N$ and integers $e, f \in \mathbb{Z}$ such that $\gcd(e, f) = 1$ and $z^e \equiv y^f \bmod n$, computes $x \in \mathbb{Z}_N$ satisfying $x^e \equiv y \bmod N$.*

2.3 Generalized Birthday Bound

Although not explicitly stated, the following lemma is implicit in [37]. We will apply it several times in the security proofs for our generic signature schemes.

Lemma 2.3. *Let A be a set with $|A| = a$. Let $X_1, \ldots, X_q$ be q independent random variables, taking uniformly random values from A. Then the probability that there exist $m+1$ pairwise distinct indices $i_1, \ldots, i_{m+1}$ such that $X_{i_1} = \cdots = X_{i_{m+1}}$ is upper bounded by $\frac{q^{m+1}}{a^m}$.*

2.4 Pairing groups and q-Diffie-Hellman.

The generic signature scheme we describe in Section 5 is defined on families of groups with bilinear pairing. A pairing group $\mathbb{PG} = \mathbb{PG}_k = (\mathbb{G}, \mathbb{G}_T, p, \hat{e}, g)$ consists of groups $\mathbb{G}$ and $\mathbb{G}_T$ with prime order p, where p is a $2k$-bit prime, a generator $g \in \mathbb{G}$, and a non-degenerate bilinear pairing map $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We say that an adversary $\mathcal{A}$ (t, ϵ)-breaks the q-Diffie-Hellman (q-DH) assumption, if $\mathcal{A}$ runs in time t and

$$\Pr[g^{1/x} \xleftarrow{\$} \mathcal{A}(g, g^x, \ldots, g^{x^q})] \geq \epsilon$$

for $x \xleftarrow{\$} \mathbb{Z}_p$. We assume that there exists no algorithm that (t, ϵ)-breaks the q-Diffie-Hellman Inversion assumption with polynomial t and non-negligible ϵ.

3 Programmable Hash Functions

3.1 Definitions

Let $G = (\mathbb{G}_k)$ be a family of groups, indexed by security parameter $k \in \mathbb{N}$. We omit the subscript when the reference to the security parameter is clear, thus write $\mathbb{G}$ for $\mathbb{G}_k$.

A *group hash function* H over $\mathbb{G}$ with input length $l = l(k)$ consists of two efficient algorithms $\mathsf{PHF.Gen}$ and $\mathsf{PHF.Eval}$. The probabilistic algorithm $\kappa \xleftarrow{\$} \mathsf{PHF.Gen}(1^k)$ generates a hash key κ for security parameter k. Algorithm $\mathsf{PHF.Eval}$ is a deterministic algorithm, taking as input a hash function key κ and $X \in \{0,1\}^l$, and returning $\mathsf{PHF.Eval}(\kappa, X) \in \mathbb{G}$.

Definition 3.1. A group hash function $H = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ is (m, n, γ, δ)-*programmable*, if there is an efficient trapdoor key generation algorithm $\mathsf{PHF.TrapGen}$ and an efficient trapdoor evaluation algorithm $\mathsf{PHF.TrapEval}$ with the following properties.

1. The probabilistic trapdoor generation algorithm $(\kappa, \tau) \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$ takes as input group elements $g, h \in \mathbb{G}$, and produces a hash function key κ together with trapdoor information τ.
2. For all generators $g, h \in \mathbb{G}$, the keys $\kappa \xleftarrow{\$} \mathsf{PHF.Gen}(1^k)$ and $\kappa' \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$ are statistically γ-close.
3. On input $X \in \{0,1\}^l$ and trapdoor information τ, the deterministic trapdoor evaluation algorithm $(a_X, b_X) \leftarrow \mathsf{PHF.TrapEval}(\tau, X)$ produces $a_X, b_X \in \mathbb{Z}$ so that for all $X \in \{0,1\}^l$,
$$\mathsf{PHF.Eval}(\kappa, X) = g^{a_X} h^{b_X}$$
4. For all $g, h \in \mathbb{G}$, all κ generated by $\kappa \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$, and all $X_1, \ldots, X_m \in \{0,1\}^l$ and $Z_1, \ldots, Z_n \in \{0,1\}^l$ such that $X_i \neq Z_j$ for all i, j, we have
$$\Pr[a_{X_1} = \cdots = a_{X_m} = 0 \text{ and } a_{Z_1}, \ldots, a_{Z_n} \neq 0] \geq \delta,$$
where $(a_{X_i}, b_{X_i}) = \mathsf{PHF.TrapEval}(\tau, X_i)$ and $(a_{Z_j}, b_{Z_j}) = \mathsf{PHF.TrapEval}(\tau, Z_j)$, and the probability is taken over the trapdoor τ produced along with κ.

If γ is negligible and δ is noticeable, then we also say that H is (m, n)-*programmable* for short. If H is $(1, q)$-programmable for every polynomial $q = q(k)$, then we say that H is $(1, \mathsf{poly})$-programmable.

In settings in which the group order is hidden, we will use a refinement of the PHF definition:

Definition 3.2. A group hash function $H = (\mathsf{RPHF.Gen}, \mathsf{RPHF.Eval})$ is *evasively* (m, n, γ, δ)-*programmable*, if it is (m, n, γ, δ)-programmable as in Definition 3.1, but with the strengthened requirement

4'. For *all prime numbers* e *with* $2^l < e \leq |\mathbb{G}|$, all $g, h \in \mathbb{G}$, and all κ generated by $\kappa \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$, and all $X_1, \ldots, X_m \in \{0,1\}^l$ and $Z_1, \ldots, Z_n \in \{0,1\}^l$ such that $X_i \neq Z_j$ for all i, j, we have
$$\Pr[a_{X_1} = \cdots = a_{X_m} = 0 \text{ and } \gcd(a_{Z_1}, e) = \cdots = \gcd(a_{Z_n}, e) = 1] \geq \delta,$$
where $(a_{X_i}, b_{X_i}) = \mathsf{PHF.TrapEval}(\tau, X_i)$ and $(a_{Z_j}, b_{Z_j}) = \mathsf{PHF.TrapEval}(\tau, Z_j)$, and the probability is taken over the trapdoor τ produced along with κ.

Hofheinz and Kiltz [37] have also introduced the notion of *randomized* programmable hash functions. A *randomized* group hash function H with input length $l = l(k)$ and randomness space $R = (\mathcal{R}_k)$ consists of two efficient algorithms $\mathsf{RPHF.Gen}$ and $\mathsf{RPHF.Eval}$. Algorithm $\mathsf{RPHF.Gen}$ is probabilistic, and generates a hash key $\kappa \xleftarrow{\$} \mathsf{RPHF.Gen}(1^k)$ for security parameter k. The deterministic algorithm $\mathsf{RPHF.Eval}$ takes randomness $r \in \mathcal{R}_k$ and $X \in \{0,1\}^l$ as input, and returns a group element $\mathsf{RPHF.Eval}(\kappa, X) \in \mathbb{G}$.

Definition 3.3. A randomized group hash function $H = (\mathsf{RPHF.Gen}, \mathsf{RPHF.Eval})$ is called (m, n, γ, δ)-programmable, if there are efficient algorithms $\mathsf{RPHF.TrapGen}$, $\mathsf{RPHF.TrapEval}$, and $\mathsf{RPHF.TrapRand}$ such that:

1. The probabilistic algorithm $\mathsf{PHF.TrapGen}(1^k, g, h)$ takes as input group elements $g, h \in \mathbb{G}$, and produces a key κ and trapdoor τ. For all generators $g, h \in \mathbb{G}$, the keys $\kappa \xleftarrow{\$} \mathsf{PHF.Gen}(1^k)$ and $\kappa' \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$ are statistically γ-close.
2. The deterministic trapdoor evaluation algorithm takes as input $X \in \{0,1\}^l$ and $r \in \mathcal{R}_k$, and produces two functions $(a_X(\cdot), b_X(\cdot)) \leftarrow \mathsf{PHF.TrapEval}(\tau, X, r)$ such that for all $X \in \{0,1\}^l$,
$$\mathsf{PHF.Eval}(\kappa, X, r) = g^{a_X(r)} h^{b_X(r)}.$$
3. On input of trapdoor τ, $X \in \{0,1\}^l$, and index $i \in [m]$, the $\mathsf{RPHF.TrapRand}$ algorithm produces $r \leftarrow \mathsf{RPHF.TrapRand}(\tau, X, i)$ with $r \in \mathcal{R}_k$. For all $g, h \in \mathbb{G}$, all κ generated by $(\kappa, \tau) \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$, all $X_1, \ldots, X_m$, and $r_{X_i} = \mathsf{RPHF.TrapRand}(\tau, X_i, i)$, we require that the r_{X_i} are independent and uniformly distributed random variables over $\mathcal{R}_k$.
4. For all $g, h \in \mathbb{G}$ and all κ generated by $(\kappa, \tau) \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$, all $X_1, \ldots, X_m \in \{0,1\}^l$ and $Z_1, \ldots, Z_n \in \{0,1\}^l$ such that $X_i \neq Z_j$, and for all $\tilde{r}_1, \ldots, \tilde{r}_n \in \mathcal{R}_k$ and $r_{X_i} \leftarrow \mathsf{RPHF.TrapRand}(\tau, X_i, i)$, we have
$$\Pr[a_{X_1}(r_{X_1}) = \cdots = a_{X_m}(r_{X_m}) = 0 \text{ and } a_{Z_1}(\tilde{r_1}), \ldots, a_{Z_n}(\tilde{r_n}) \neq 0] \geq \delta,$$
where $(a_{X_i}, b_{X_i}) = \mathsf{RPHF.TrapEval}(\tau, X_i, r_{X_i})$ and $(a_{Z_j}, b_{Z_j}) = \mathsf{PHF.TrapEval}(\tau, Z_j, \tilde{r}_j)$, and the probability is taken over the trapdoor τ produced along with κ. Here X_i may depend on X_j and r_{X_j} for $j < i$, and the $Z_1, \ldots, Z_n$ may depend on all X_i and r_i.

Again we omit γ and δ, if γ is negligible and δ is noticeable. *Randomized evasively programmable hash functions* are defined as in Definition 3.2.

In the remainder of this Section we propose a number of new PHFs offering different trade-offs. Our results are summarized in Table 2.

Name	**Type**	**Parameters** (m, n)	**Size of** κ	**Randomness space**
$\mathsf{H}_{\mathsf{Wat}}$ [54, 37] (§ 3.2)	PHF	$(1, \mathsf{poly})$ and $(2, 1)$	$(l+1) \times \lvert\mathbb{G}\rvert$	—
$\mathsf{H}_{\mathsf{cfs}}$ (§ 3.3)	PHF	$(m, 1)$	$(16m^2 l + 1) \times \lvert\mathbb{G}\rvert$	—
$\mathsf{H}_{\mathsf{rand}}$ (§ 3.4)	RPHF	$(m, 1)$	$(2m^2 + 1) \times \lvert\mathbb{G}\rvert$	$\mathcal{R} = \{0,1\}^l$
$\mathsf{H}_{\mathsf{Weak}}$ (§ 3.5)	weak PHF	$(m, 1)$	$(m+1) \times \lvert\mathbb{G}\rvert$	—

Table 2: Overview of our constructions of (randomized/weak) programmable hash functions. Rows with grey background are new constructions from this paper.

3.2 Multi-generator programmable hash function

The programmable hash function described in Definition 3.4 below was (implicitly) introduced in [54]. An explicit analysis can be found in [37].

Definition 3.4. Let $G = (\mathbb{G}_k)$ be a group family, and $l = l(k)$ be a polynomial. Let $\mathsf{H}_{\mathsf{Wat}} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ be defined as follows.

- $\mathsf{PHF.Gen}(1^k)$ returns $\kappa = (h_0, \ldots, h_l)$, where $h_i \xleftarrow{\$} \mathbb{G}_k$ for $i \in [l]$.
- On input $X = (x_1, \ldots, x_l) \in \{0,1\}^l$ and $\kappa = (h_0, \ldots, h_l)$, $\mathsf{PHF.Eval}(\kappa, X)$ returns

$$\mathsf{PHF.Eval}(\kappa, X) = h_0 \prod_{i=1}^{l} h_i^{x_i}.$$

Theorem 3.5 (Theorem 3.6 of [37])**.** *For any fixed polynomial $q = q(k)$ and any group with known order, $\mathsf{H}_{\mathsf{Wat}}$ is an evasively $(1, q, 0, O(1/(q\sqrt{l})))$-programmable and $(2, 1, 0, O(1/l))$-programmable hash function.*

Although evasive programmability was not introduced in [37], it follows from their proof, since the values of a_{Z_j} that occur there are bounded in the sense $|a_{Z_j}| < 2^l$. We remark that Theorem 3.5 also carries over to groups of unknown order.

3.3 A new deterministic programmable hash function

Let S, T be sets. We say that S does not cover T, if $T \not\subseteq S$. Let d, m, s be integers, and let $F = (F_i)_{i \in [s]}$ be a family of s subsets of $[d]$. We say that F is m-cover free, if for any set I containing (up to) m indices $I = \{i_1, \ldots, i_m\} \subseteq [s]$, it holds that $F_j \not\subseteq \bigcup_{i \in I} F_i$ for any j which is *not* contained in I. In other words, if $|I| \leq m$, then the union $\bigcup_{i \in I} F_i$ is not covering F_j for all $j \in [s] \setminus I$. We say that F is w-uniform, if $|F_i| = w$ for all $i \in [s]$.

Lemma 3.6 ([28, 46])**.** *There is a deterministic polynomial-time algorithm that, on input of integers $m, s = 2^l$, returns $d \in \mathbb{N}$ and set family $F = (F_i)_{i \in [s]}$ such that F is m-cover free over $[d]$ and w-uniform, where $d \leq 16m^2 l$ and $w = d/4m$.*

In the following we will associate $X \in \{0,1\}^l$ to a subset F_i, $i \in [s]$, by interpreting X as an integer in the range $[0, 2^l - 1]$, and setting $i = X + 1$. We will write F_X to denote the subset associated to X.

Definition 3.7. Let $G = (\mathbb{G}_k)$ be a group family, and $l = l(k)$ and $m = m(k)$ be polynomials. Define $s = 2^l$, $d = 16m^2 l$, and $w = d/4m$. Let $\mathsf{H}_{\mathsf{cfs}} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ be as follows.

- $\mathsf{PHF.Gen}(1^k)$ returns $\kappa = (h_1, \ldots, h_d)$, where $h_i \xleftarrow{\$} \mathbb{G}_k$ for $1 \leq i \leq d$.
- Let $F_X \subseteq [d]$ be the subset associated to $X \in [0, 2^l - 1]$. On input X and $\kappa = (h_1, \ldots, h_d)$, $\mathsf{PHF.Eval}(\kappa, X)$ returns

$$\mathsf{PHF.Eval}(\kappa, X) = \prod_{i \in F_X} h_i.$$

Theorem 3.8. *Let $\mathbb{G} = \mathbb{G}_k$ be a group of known order p. $\mathsf{H}_{\mathsf{cfs}}$ is an evasively $(m, 1, \gamma, \delta)$-programmable hash function with $\gamma = 0$ and $\delta = 1/(16m^2 l)$.*

Proof. Consider the following algorithms.

- $\mathsf{PHF.TrapGen}(1^k, g, h)$ samples d uniformly random integers $b_1, \ldots, b_d \xleftarrow{\$} \mathbb{Z}_p$ and an index $t \xleftarrow{\$} [d]$. Then it sets $h_t = gh^{b_t}$, and $h_i = h^{b_i}$ for all $i \in [1, d]$ with $i \neq t$. $\mathsf{PHF.TrapGen}$ returns (κ, τ) with $\tau = (t, b_1, \ldots, b_d)$ and $\kappa = (h_1, \ldots, h_d)$.
- On input (τ, X), $\mathsf{PHF.TrapEval}$ sets $b_X = \sum_{i \in F_X} b_i$, and $a_X = 1$ if $t \in F_X$, and $a_X = 0$ if $t \notin F_X$, and returns (a_X, b_X).

$\mathsf{PHF.TrapGen}$ outputs a vector of independent and uniformly distributed group elements, thus we have $\gamma = 0$. Fix $X_1, \ldots, X_m, Z \in [0, 2^l - 1]$. Since F is a m-cover free set family, there must be an index t' such that $t' \notin \bigcup_{i=1}^m F_{X_i}$, but $t' \in F_Z$. Since t is picked uniformly random among $16m^2l$ possibilities, we have $t = t'$, and thus $a_{X_i} = 0$ and $a_Z = 1$, with probability $\delta = 1/(16m^2l)$. Finally, $a_Z = 1$ implies $\gcd(a_Z, e) = 1$ for all primes e, thus $\mathsf{H_{cfs}}$ is evasively programmable. □

Theorem 3.8 can be generalized to groups of hidden order. The proof proceeds exactly like the proof of Theorem 3.8, except that we have to approximate the group order. E.g., for the group of quadratic residues QR_n, we can sample random exponents $b_i \xleftarrow{\$} \mathbb{Z}_{n^2}$. This way, we can sample nearly uniform ($1/\sqrt{n}$-close) group elements $h_i = h^{b_i}$, which yields the following theorem.

Theorem 3.9. *Let $\mathbb{G} = \mathsf{QR}_n$ be the group of quadratic residues modulo $n = pq$, where p and q are safe distinct primes. $\mathsf{H_{cfs}}$ is a $(m, 1, \gamma, \delta)$-evasively programmable hash function over $\mathbb{G}$ with $\gamma = d/\sqrt{n}$ and $\delta = 1/(16m^2l)$.*

3.4 A randomized programmable hash function

In [39] a randomized $(2, 1)$-PHF was described which we now generalize to a randomzied $(m, 1)$-PRF, for any $m \geq 1$.

Definition 3.10. Let $G = (\mathbb{G}_k)$ be a group family, and $m = m(k)$ be a polynomial. In the following, let $[X]_{2^l} \in \mathbb{Z}$ denote a canonical interpretation of a field element $X \in \mathbb{F}_{2^l}$ as an integer between 0 and $2^l - 1$. We assume that X and $[X]_{2^l}$ are efficiently computable from one another. Let $\mathsf{H_{rand}} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ be defined as follows.

- $\mathsf{RPHF.Gen}(1^k)$ returns a uniformly sampled $\kappa = (h_0, (h_{i,j})_{(i,j) \in [2m] \times [m]}) \in \mathbb{G}^{2m^2+1}$.
- $\mathsf{RPHF.Eval}(\kappa, X; r)$ parses $X, r \in \mathbb{F}_{2^l}$, and computes and returns

$$\mathsf{RPHF.Eval}_\kappa(X; r) = h_0 \prod_{i,j=1}^{m} h_{i,j}^{([iX+r]_{2^l})^j}.$$

Theorem 3.11. *For any group $\mathbb{G}$ of known order, $\mathsf{H_{rand}}$ is evasively $(m, 1, 0, 1/2)$-programmable. For the group $\mathbb{G} = \mathsf{QR}_N$ of quadratic residues modulo $N = pq$ for safe distinct primes p and q, the function $\mathsf{H_{rand}}$ is evasively $(m, 1, (2m^2 + 1)/\sqrt{N}, 1/2)$-programmable.*

Proof. We describe suitable algorithms $\mathsf{RPHF.TrapGen}$ and $\mathsf{RPHF.TrapEval}$. First assume a group $\mathbb{G}$ with known order $|\mathbb{G}|$.

- $\mathsf{RPHF.TrapGen}(1^k, g, h)$ uniformly picks $i^* \in [2m]$, as well as $\zeta_j \in \mathbb{F}_{2^l}$ and $\beta_0, \beta_{i,j} \in \mathbb{Z}_{|\mathbb{G}|}$ for $(i, j) \in [2m] \times [m]$, sets

$$\alpha(t) := \prod_{j \in [m]} (t - [\zeta_j]_{2^l}) =: \sum_{j=0}^{m} \alpha_j t^j \in \mathbb{Z}[t], \tag{3}$$

and so defines

$$h_0 := g^{\alpha_0} h^{\beta_0} \qquad h_{i,j} := \begin{cases} g^{\alpha_j} h^{\beta_{i,j}} & \text{for } i = i^*, \\ h^{\beta_{i,j}} & \text{for } i \neq i^*. \end{cases}$$

Output is $\kappa = (h_0, (h_{i,j})_{(i,j)\in[2m]\times[m]})$ and $\tau = (i^*, (\zeta_j)_{j\in[m]}, (\beta_{i,j})_{(i,j)\in[2m]\times[m]})$. Since the $\beta_{i,j}$ are chosen independently and uniformly, this implies that κ is distributed exactly like the output of $\mathsf{RPHF.Gen}$.

- $\mathsf{RPHF.TrapEval}(\tau, X, r)$ computes and outputs

$$a := \alpha([i^*X + r]_{2^l}) = \sum_{j=0}^{m} \alpha_j \cdot ([i^*X + r]_{2^l})^j \qquad b := \sum_{(i,j)\in[2m]\times[m]} \beta_{i,j} \cdot ([iX + r]_{2^l})^j$$

  for $\alpha(t) \in \mathbb{Z}[t]$ as defined by the ζ_j through (3). By construction, $\mathsf{RPHF.Eval}(\kappa, X, r) = h_0 \prod_{i,j=1}^{m} h_{i,j}^{([iX+r]_{2^l})^j} = g^a h^b$.
- $\mathsf{RPHF.TrapRand}(\tau, X_j, j)$ outputs $r_j := \zeta_j - i^*X \in \mathbb{F}_{2^l}$, so $\mathsf{RPHF.TrapEval}(\tau, X_j, r_j)$ yields $a = \alpha([i^*X_j + r_j]_{2^l}) = \alpha([\zeta_j]_{2^l}) = 0$. As the ζ_j are uniform and independent of κ, so is r_j.

It remains to prove that for all prime e with $2^l < e \leq |\mathbb{G}|$, all κ, $X_1, \ldots, X_m$, Z, for $r_j \xleftarrow{\$} \mathsf{RPHF.TrapRand}(\tau, X_j, j)$, and all $\tilde{r}$ (with dependencies as in Definition 3.3), we have $a_{X_j}(r_j) = 0$ for all j, but $\gcd(e, a_Z(\tilde{r})) = 1$ with probability at least $1/2$ over τ. So fix e, κ, X_i, and Z. First, we have $a_{X_j}(r_j) = \alpha([i^*X_j + r_j]_{2^l}) = 0$ always, by definition of $\mathsf{RPHF.TrapRand}$. Second,

$$\gcd(e, a_Z(r)) \neq 1 \quad \Leftrightarrow \quad a_Z(r) \equiv \alpha([i^*Z + r]_{2^l}) \equiv 0 \bmod e \quad \Leftrightarrow \quad i^*Z + r \in \{\zeta_j\}_{j\in[m]},$$

since $e > 2^l$. Hence, we only have to upper bound the probability for $i^*Z + r \in \{\zeta_j\}_{j\in[m]}$.

Now $i^*Z + r \in \{\zeta_j\}_{j\in[m]}$ iff $i^* \in \{(\zeta_j - r)/Z\}_{j\in[m]}$. Now $i^* \in [2m]$ is uniform and independent of Z and the ζ_j, and the set $\{(\zeta_j - r)/Z\}_{j\in[m]}$ has cardinality at most m. Hence, $i^* \in \{(\zeta_j - r)/Z\}_{j\in[m]}$ with probability at most $1/2$ as desired.

This proves the statement for $\mathbb{G}$ with known group order. Now assume $\mathbb{G} = \mathsf{QR}_N$. We define $\mathsf{RPHF.TrapGen}$, $\mathsf{RPHF.TrapEval}$, and $\mathsf{RPHF.TrapRand}$ as above, with the difference that we sample $\beta_0, \beta_{i,j} \in \{0, \ldots, \lfloor N/4 \rfloor\}$ because the group order $|\mathbb{G}| = |\mathsf{QR}_N| = \varphi(N)$ is unknown. This approximate sampling of a random exponent yields a statistical error of at most $(2m^2 + 1)/\sqrt{N}$ in the distribution of κ, as sampled by $\mathsf{RPHF.TrapGen}$. The remaining part of the proof remains unchanged. □

3.5 A Weak Programmable Hash Function

Essentially, a *weak* programmable hash function is a programmable hash function according to Definition 3.1, except that the trapdoor generation algorithm receives a list $X_1, \ldots, X_m \in \{0,1\}^l$ as additional input. On the one hand this allows us to construct significantly more efficient deterministic programmable hash functions, while on the other hand our generic signatures schemes described in Sections 5 and 4 are only weakly secure when instantiated with weak programmable hash functions. Fully secure signature schemes can be obtained by applying a generic conversion from weak to full security, for instance using chameleon hashes [45] which can be constructed based on standard assumptions like discrete logarithms [45], RSA [2, 40], or factoring [45].

Definition 3.12. A group hash function is a *weak* (m, n, γ, δ)-programmable hash function, if there is a (probabilistic) algorithm $\mathsf{PHF.TrapGen}$ and a (deterministic) algorithm $\mathsf{PHF.TrapEval}$ such that:

1. $(\kappa, \tau) \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h, X_1, \ldots, X_m)$ takes as input group elements $g, h \in \mathbb{G}$ and $X_1, \ldots, X_m \in \{0,1\}^l$, and produces a hash function key κ together with trapdoor information τ.

2.-4. Like in Definition 3.1.

As before, we may omit γ and δ, if γ is negligible and δ is noticeable. *Weak evasively programmable hash functions* are defined as in Definition 3.2.

Interestingly, there is a very simple way to construct a randomized programmable hash function according to Definition 3.3 from any weak programmable hash function. Let us now describe our instantiation of a weak (evasively) programmable hash function. This PHF already appeared implicitly in [20, 51] and [10] for $m = 1$.

Definition 3.13. Let $G = (\mathbb{G}_k)$ be a group family, and $l = l(k)$ and $m = m(k)$ be polynomials. Let $\mathsf{H_{Weak}} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ be defined as follows.

- $\mathsf{PHF.Gen}(1^k)$ returns $\kappa = (h_0, \ldots, h_m)$, where $h_i \xleftarrow{\$} \mathbb{G}_k$ for $i \in \{0, \ldots, m\}$.
- On input $X \in \{0,1\}^l$ and $\kappa = (h_0, \ldots, h_m)$, $\mathsf{PHF.Eval}(\kappa, X)$ returns

$$\mathsf{PHF.Eval}(\kappa, X) = \prod_{i=0}^{m} h_i^{(X^i)}.$$

Here we interpret the l-bit strings X_i, $i \in [m]$, as integers in the canonical way.

Theorem 3.14. *Let $\mathbb{G} = \mathbb{G}_k$ be a group of known order p. $\mathsf{H_{Weak}}$ is a weak evasively $(m, 1, \gamma, \delta)$-programmable hash function with $\gamma = 0$ and $\delta = 1$.*

Proof. Consider the following algorithms.

- $\mathsf{PHF.TrapGen}(1^k, g, h, X_1, \ldots, X_m)$ samples $m+1$ random integers $\beta_0, \ldots, \beta_m \xleftarrow{\$} \mathbb{Z}_p$ and $X_0 \xleftarrow{\$} \{0,1\}^l$. Then it computes the coefficients $(\alpha_0, \ldots, \alpha_m)$ of the polynomial

$$\alpha(t) := \sum_{i=0}^{m} \alpha_i t^i = \prod_{i=0}^{m+1} (t - X_i) \in \mathbb{Z}[t].$$

The algorithm sets $\tau = (\alpha_0, \ldots, \alpha_m, \beta_0, \ldots, \beta_m)$, and $\kappa = (h_0, \ldots, h_m)$, where $h_i = g^{\alpha_i} h^{\beta_i}$ for $i \in [0, m]$.

- $\mathsf{PHF.TrapEval}$ returns (a_X, b_X) on input (τ, X), where $a_X = \alpha(X) = \sum_{i=0}^{m} \alpha_i X^i$ and $b_X = \sum_{i=0}^{m} \beta_i X^i$.

Just like the $\mathsf{PHF.Gen}$ algorithm, $\mathsf{PHF.TrapGen}$ returns a vector κ of uniformly distributed group elements, which implies $\gamma = 0$. Note also that we have $a_X = \alpha(X) = 0$ for all $X \in \{X_1, \ldots, X_m\}$. It is left to observe that

$$\gcd(e, a_Z) \neq 1 \quad \Leftrightarrow \quad \alpha(Z) \equiv 0 \bmod e \quad \Leftrightarrow \quad Z \in \{X_1, \ldots, X_m\},$$

since $e > 2^l$ is prime and $0 \leq X_i, Z < 2^l$. □

Again we can generalize Theorem 3.14 to groups of hidden order. The proof proceeds exactly like the proof of Theorem 3.14, except that we have to approximate the group order. For the group of quadratic residues QR_n, we can sample the random exponents b_i from $\mathbb{Z}_{n^2}$ for $i \in [0, m]$, which yields the following theorem.

Theorem 3.15. *Let $\mathbb{G} = \mathsf{QR}_N$ be the group of quadratic residues modulo $N = pq$, where p and q are safe distinct primes. $\mathsf{H}_{\mathsf{Weak}}$ is a $(m, 1, \gamma, \delta)$-programmable hash function over $\mathbb{G}$ with $\gamma = (m+1)/\sqrt{N}$ and $\delta = 0$.*

4 Signatures from the RSA Problem

4.1 Construction

Let $l = l(k)$ and $\lambda = \lambda(k)$ be polynomials. Let $\mathsf{H} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ be group hash functions over $\mathbb{G} = \mathsf{QR}_N$ with input length l. We define the signature scheme $\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}] = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ as follows.

$\mathsf{Gen}(1^k)$**:** The key generation algorithm picks two large safe $k/2$-bit primes p and q, and sets $N = pq$. Then it generates a group hash function key $\kappa \xleftarrow{\$} \mathsf{PHF.Gen}(1^k)$ for the group QR_N. Finally it chooses a random key K for the pseudorandom function $\mathsf{PRF} : \{0,1\}^* \to \{0,1\}^r$ and picks $c \xleftarrow{\$} \{0,1\}^r$, where $r = \lceil \log N \rceil$. These values define a function F as

$$\mathsf{F}(z) = \mathsf{PRF}_K(\mu || z) \oplus c,$$

where μ, called the *resolving index* of z, denotes the smallest positive integer such that $\mathsf{PRF}_K(\mu||z) \oplus c$ is an odd prime. Here $\oplus$ denotes the bit-wise XOR operation, and we interpret the r-bit string returned by F as an integer in the obvious way. (The definition of F is the same as in [41]. It is possible to replace the PRF with an $2k^2$-wise independent hash function [17].) The public key is $pk = (n, \kappa, K, c)$, the secret key is $sk = (pk, p, q)$.

In the following we will write $\mathsf{H}(M)$ shorthand for $\mathsf{PHF.Eval}(\kappa, M)$, and define $\mathsf{P} : \{0,1\}^\lambda \to \mathbb{N}$ as $\mathsf{P}(s) = \prod_{i=1}^{\lambda} \mathsf{F}(s_{|i})$, where $s_{|i}$ is the i-th prefix of s, i.e., the bit string consisting of the first i bits of s. We also define $s_{|0} = \emptyset$, where $\emptyset$ is the empty string, for technical reasons.

$\mathsf{Sign}(sk, M)$**:** On input of secret key sk and message $M \in \{0,1\}^l$, the signing algorithm picks $s \xleftarrow{\$} \{0,1\}^\lambda$ uniformly random and computes

$$\sigma = \mathsf{H}(M)^{1/\mathsf{P}(s)} \bmod N,$$

where the inverse of $\mathsf{P}(s)$ is computed modulo the order $\phi(n) = (p-1)(q-1)$ of the multiplicative group $\mathbb{Z}_N^*$. The signature is $(\sigma, s) \in \mathbb{Z}_N \times \{0,1\}^\lambda$.

$\mathsf{Vfy}(pk, M, (\sigma, s))$**:** On input of pk, message M, and signature (σ, s), return `accept` if

$$\mathsf{H}(M) = \sigma^{\mathsf{P}(s)} \bmod N.$$

Otherwise return `reject`.

Correctness. If $\sigma = \mathsf{H}(M)^{1/\mathsf{P}(s)}$, then we have $\sigma^{\mathsf{P}(s)} = \mathsf{H}(M)^{\mathsf{P}(s)/\mathsf{P}(s)} = \mathsf{H}(M)$.

4.2 Security

Theorem 4.1. *Let* PRF *be a* (ϵ'', t'')*-secure pseudo-random function and* H *be a* $(m, 1, \gamma, \delta)$*-evasively programmable hash function. Suppose there exists a* (t, q, ϵ)*-forger* $\mathcal{F}$ *breaking the existential forgery under adaptive chosen message attacks of* $\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}]$*. Then there exists an adversary that* (t', ϵ')*-breaks the RSA assumption with* $t' \approx t$ *and*

$$\epsilon \leq (q+1)\lambda \left(\frac{4r^2}{\delta} \left(\epsilon' + \frac{r}{l \cdot 2^{r-l-1}} \right) + 3\epsilon'' + \frac{r(q+1)^2\lambda^2 + 2r + 1}{2^r} + \gamma + \frac{1}{2^{r-l}} \right) + \frac{q^{m+1}}{2^{m\lambda}}$$

We postpone a full proof to Appendix C.1, and only give a brief outline here. As customary in proofs for similar signature schemes (e.g., [24, 30, 37]), we distinguish between *Type I* and *Type II* forgers. A Type I forger forges a signature of the form (M^*, σ^*, s^*) with $s^* = s_i$ for some $i \in [q]$. (That is, a Type I forger reuses some s_i from a signature query.) A *Type II* forger returns a signature with a fresh s^*.

It will be easiest to first describe how to treat a Type II forger $\mathcal{F}$. Recall that we need to put up a simulation that is able to generate q signatures $(M_i, \sigma_i, s_i)_{i \in [q]}$ for adversarially chosen messages M_i. To do this, we choose all s_i in advance. We then prepare the PHF H using $\mathsf{PHF.TrapGen}$, but relative to generators g and h for which we know $\mathsf{P}(s_i)$-th roots. (That is, we set $g := \hat{g}^E$ and $h = \hat{h}^E$ for $E := \prod_i \mathsf{P}(s_i)$.) This allows to generate signatures for $\mathcal{F}$; also, by the security of the PHF H, this change goes unnoticed by $\mathcal{F}$. However, each time $\mathcal{F}$ outputs a new signature, it essentially outputs a fresh root $g^{1/\mathsf{P}(s^*)}$ of g, from which we can derive a $\mathsf{P}(s^*)$-th root of $\hat{g}$. To construct an RSA adversary from this experiment, we have to embed an auxiliary given exponent e into the definition of P, such that $\hat{g}^{1/\mathsf{P}(s^*)}$ allows to derive $\hat{g}^{1/e}$. This can be done along the lines of the proof of the Hohenberger-Waters scheme [41]. Concretely, for *initially given* values s_i and e, we can set up P such that (a) e does not divide any $\mathsf{P}(s_i)$, but (b) for any other fixed s^*, the probability that e divides $\mathsf{P}(s^*)$ is significant. Note that in our scheme, the s_i are chosen by the signer, and thus our simulation can select them in advance. In contrast to that, the HW scheme uses the signed messages M_i as arguments to P, and thus their argument achieves only a weaker form of security in which the forger has to commit to all signature queries beforehand.

Now the proof for Type I forgers proceeds similarly, but with the additional complication that we have to prepare one or more signatures of the form $\mathsf{H}(M_i)^{1/\mathsf{P}(s_i)}$ for the same $s_i = s^*$ that $\mathcal{F}$ eventually uses in his forgery. We resolve this complication by relying on the PHF properties of H. Namely, we first choose all s_i and guess i (i.e., the index of the s_i with $s_i = s^*$). We then prepare H with generators g, h such that we know all $\mathsf{P}(s_j)$th roots of h (for all j), and all $\mathsf{P}(s_j)$th roots of g *for all* $s_j \neq s_i$. Our hope is that whenever $\mathcal{F}$ asks for the signature of some M_j with $s_j = s_i$, we have $\mathsf{H}(M_i) \in \langle h \rangle$, so we can compute $\mathsf{H}(M_j)^{1/\mathsf{P}(s_j)}$. At the same time, we hope that $\mathsf{H}(M^*) \notin \langle h \rangle$ has a nontrivial g-factor, so we can build an RSA adversary as for Type II forgers. The PHF property of H guarantees a significant probability that this works out, provided that there are no more than m indices j with $s_j = s_i$ (i.e., provided that there are no $(m+1)$-collisions). However, using a birthday bound, we can reasonably upper bound the probability of $(m+1)$-collisions.

4.3 Deterministic signatures with Weak Security

We now give a variant of our scheme which is slightly more efficient but only offers weak security. A weakly secure signature scheme can be updated to a fully secure one by using a (randomized) Chameleon Hash Function. The weakly secure signature scheme $\mathsf{Sig}^*_{\mathsf{RSA}}[\mathsf{H}] = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ uses

the key generation algorithm with $\mathsf{Sig}_{\mathsf{RSA}}$ and signing and verification algorithms are defined as follows.

$\mathsf{Sign}(sk, M)$**:** On input of secret key sk and message $M \in \{0,1\}^l$, the signing algorithm computes

$$\sigma = \mathsf{H}(M)^{1/\mathsf{P}(M)} \bmod N,$$

where $\mathsf{P}(M) = \prod_{i=1}^{\lambda} \mathsf{F}(\mathsf{PRF}_K(M)_{|i})$, i.e., the product is over the first λ prefixes of $\mathsf{PRF}_K(M)$. The signature is $\sigma \in \mathbb{Z}_N$.

$\mathsf{Vfy}(pk, M, \sigma)$**:** On input of pk, message M, and signature σ, return `accept` if

$$\mathsf{H}(M) = \sigma^{\mathsf{P}(M)} \bmod N.$$

Otherwise return `reject`.

The proof of security is analog to the one of Theorem 4.1 and therefore omitted.

4.4 Efficiency

Given $\mathsf{P}(s)$ and $\phi(N)$, computing $\sigma = \mathsf{H}(M)^{1/\mathsf{P}(s)}$ can also be carried out by one single exponentiation. Since one single evaluation of $\mathsf{P}(\cdot)$ has to perform (expected) λr many primality tests (for r-bit primes), the dominiant part of signing and verification is to compute $\mathsf{P}(s)$, for $s \in \{0,1\}^{\lambda}$. Theorem 4.1 tells us that if H is a $(m,1)$-PHF we can set $\lambda = \log q + k/m$, see Appendix A for more details.

Hohenberger and Waters [41] proposed several ways to improve the efficiency of their RSA-based signature scheme. These improvements apply to our RSA-based schemes as well.

Using a larger alphabet. Instead of applying the function F to each prefix of s, one could break s into larger, say u-bit, chunks. This would result in a decrease of security by a factor of about $1/(2^u - 1)$, since the simulator in the Type II forger game has to guess which of the $1/(2^u - 1)$ values the forger will use. However, for small values of u, this reduces the cost of signing and verifying significantly (about $1/2$ for $u = 2$), while imposing only a moderate decrease in security (about $1/3$ for $u = 2$). In particular of interest is a variant of our scheme using $\mathsf{P}(s) := \mathsf{F}(s)$. In this case the security reduction of Theorem 4.1 looses an additional factor of $2^{\lambda} = q \cdot 2^{k/m} \approx q$ (for $m \approx k$) but the scheme only has to generate one single prime for signing and verifying.

Include the resolving index in the signature. In order to reduce the number of primality tests to be performed during signature verification, it is possible to include the resolving indices in the signatures. This increases the size of signatures by a factor of $\lambda \cdot \log(\lfloor \log n \rfloor^2)$, and we have to modify the verification algorithm such that a signature is rejected if a resolving index exceeds $\log(\lfloor \log n \rfloor^2)$, which imposes a negligible correctness error.

Using smaller prime exponents. Defining the scheme such that instead of $\log N$-bit primes only small prime exponents are used, say r-bit for some $2k \leq r < \log N$, yields a considerable efficiency improvement, since the signing and verifying algorithms need only find small primes. However, in this case we can reduce the security of our schemes only to some low-exponent RSA assumption, that is, the RSA assumption with exponents smaller than 2^r.

Moreover, when instantiated with a weak programmable hash function plus chameleon hash, then our schemes can also be used as online-offline signature schemes as introduced by Even, Goldreich, and Micali [29]

5 Signatures from the q-Diffie-Hellman Problem

5.1 Construction

Let $\mathbb{G}, \mathbb{G}_T$ be groups of prime order p with bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let $l = l(k)$ and $\lambda = \lambda(k)$ be polynomials. Let $\mathsf{H} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ and $\mathsf{D} = (\mathsf{PHF.Gen}', \mathsf{PHF.Eval}')$ be group hash functions over $\mathbb{G}$ with input length l, such that D is programmable using algorithms $(\mathsf{PHF.TrapGen}', \mathsf{PHF.TrapEval}')$. We define $\mathsf{Sig}_{q\text{-}\mathsf{DH}}[\mathsf{H}, \mathsf{D}] = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ as follows.

$\mathsf{Gen}(1^k)$**:** The key generation algorithm generates hash function keys through $\kappa \xleftarrow{\$} \mathsf{PHF.Gen}(1^k)$ and $(\kappa', \tau') \xleftarrow{\$} \mathsf{PHF.TrapGen}'(1^k, \hat{g}, \hat{g}^y)$ for $\hat{g} \xleftarrow{\$} \mathbb{G}$ and $y \xleftarrow{\$} \mathbb{Z}_p$. Note that κ' is computed using the *trapdoor* key generation procedure. The public key is defined as $pk = (\hat{g}, \kappa, \kappa')$, the secret key is $sk = (pk, y, \tau')$.

In the following we will write $\mathsf{H}(M)$ shorthand for $\mathsf{PHF.Eval}(\kappa, M)$ and $\mathsf{D}(s)$ for $\mathsf{PHF.Eval}(\kappa', s)$. We write $\mathsf{d}(s)$ shorthand for the function computing $(a, b) \leftarrow \mathsf{PHF.TrapEval}'(\tau', s)$ and returning $a + yb$. Note that $\mathsf{D}(s) = \hat{g}^{\mathsf{d}(s)}$, and that the functions H and D can be computed given the public key, while d can be computed given the secret key.

$\mathsf{Sign}(sk, M)$**:** On input of secret key sk and message $M \in \{0,1\}^l$, the signing algorithm samples $s \xleftarrow{\$} \{0,1\}^\lambda$ uniformly random until[4] $\mathsf{d}(s) \not\equiv 0 \bmod p$, and computes

$$\sigma = \mathsf{H}(M)^{1/\mathsf{d}(s)}.$$

The signature is $(\sigma, s) \in \mathbb{G} \times \{0,1\}^\lambda$.

$\mathsf{Vfy}(pk, M, (\sigma, s))$**:** On input of public key pk, message M and signature (σ, s), return `accept` if

$$\hat{e}(\mathsf{D}(s), \sigma) = \hat{e}(\hat{g}, \mathsf{H}(M)).$$

Otherwise return `reject`.

Correctness. If $\mathsf{d}(s) \not\equiv 0 \bmod p$, the correctness of $\mathsf{Sig}_{q\text{-}\mathsf{DH}}[\mathsf{H}, \mathsf{D}]$ can be verified by checking

$$\hat{e}(\mathsf{D}(s), \sigma) = \hat{e}\left(\hat{g}^{\mathsf{d}(s)}, \mathsf{H}(M)^{1/\mathsf{d}(s)}\right) = \hat{e}\left(\hat{g}, \mathsf{H}(M)\right).$$

We have $\mathsf{D}(s) \neq 1 \in \mathbb{G}$, and therefore $\mathsf{d}(s) \not\equiv 0 \bmod p$, with overwhelming probability, if the discrete logarithm assumption holds in $\mathbb{G}$. To see this, observe that $\mathsf{d}(s) = 0$ is equivalent to $a + yb = 0$, thus the discrete logarithm of $h = g^y$ to base g is determined by running $(a, b) \leftarrow \mathsf{PHF.TrapEval}'(\tau', s)$ and computing $y = -ab^{-1}$.

5.2 Security

Theorem 5.1. *Let* H *be* $(m, 1, \gamma, \delta)$*-programmable and* D *be* $(1, \mathsf{poly}, \gamma', \delta')$*-programmable. Suppose there exists a* (t, q, ϵ)*-forger* $\mathcal{F}$ *breaking the security of* $\mathsf{Sig}_{q\text{-}\mathsf{DH}}[\mathsf{H}, \mathsf{D}]$ *against existential forgery under*

[4] Technically, this makes Sign expected polynomial-time. At the price of an exponentially small loss in security, this can be improved to strict polynomial-time by, say, outputting sk as soon as $\mathsf{d}(s) \equiv 0 \bmod p$ for k tries.

adaptive chosen message attacks. Then there exists an adversary that (t', ϵ')-breaks the q-Diffie-Hellman assumption with $t' \approx t$ and

$$\epsilon \leq q \left(\frac{\epsilon'}{\delta \delta'} + \gamma \right) + \frac{q^{m+1}}{2^{m\lambda}}.$$

The proof is conceptually similar to the one for the RSA case (Theorem C.1), and we postpone it to Appendix C.2.

5.3 Efficiency

To compute a signature, the signer must compute $\mathsf{H}(M) = \mathsf{PHF.Eval}(\kappa, M)$ first, and then $\mathsf{H}(M)^{1/\mathsf{d}(s)}$. One could also compute κ via trapdoor generation as $(\kappa, \tau) \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, g^x)$ during key generation, and then compute a signature by first running $(a, b) \leftarrow \mathsf{PHF.TrapEval}(\tau, M)$ and then computing

$$\sigma = g^{(a+xb)/\mathsf{d}(s)} \quad (= \mathsf{H}(M)^{1/\mathsf{d}(s)}).$$

In this way, a signature can be computed by performing one single exponentiation in $\mathbb{G}$. As we will show in Appendix A, the parameter λ can be set to $\lambda = q + k/m$ when the scheme is instantiated with a $(m, 1)$-PHF.

We remark that the scheme can also be instantiated in asymmetric pairing groups where the pairing is given by $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and $\mathbb{G}_1 \neq \mathbb{G}_2$. In that case we let the element σ from the signature be in $\mathbb{G}_1$ such that σ can be represented in 160 bits, as well as the group elements decribing hash function H. The elements decribing hash function D in the public key are from $\mathbb{G}_2$. It can be verified that the following proof also holds in asymmetric pairing groups. In this case, we rely on the assumption that computing $g_1^{1/x} \in \mathbb{G}_1$ is hard, given $g_1, g_1^x, \ldots, g_1^{x^q} \in \mathbb{G}_1$ and $g_2, g_2^x \in \mathbb{G}_2$.

References

[1] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer, Berlin, Germany.

[2] Giuseppe Ateniese and Breno de Medeiros. Identity-based chameleon hash and applications. In Ari Juels, editor, *FC 2004: 8th International Conference on Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 164–180, Key West, USA, February 9–12, 2004. Springer, Berlin, Germany.

[3] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331, Kingston, Ontario, Canada, August 11–12, 2005. Springer, Berlin, Germany.

[4] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*,

volume 839 of *Lecture Notes in Computer Science*, pages 216–233, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Berlin, Germany.

[5] Mihir Bellare and Thomas Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 407–424, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Germany.

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

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

[8] Kamel Bentahar, Pooya Farshim, John Malone-Lee, and Nigel P. Smart. Generic constructions of identity-based and certificateless KEMs. *Journal of Cryptology*, 21(2):178–199, April 2008.

[9] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In *Public Key Cryptography*, pages ???–???, 2011.

[10] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.

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

[12] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.

[13] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

[14] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Berlin, Germany.

[15] Xavier Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th*

International Conference on Theory and Practice of Public Key Cryptography, volume 6056 of *Lecture Notes in Computer Science*, pages 499–517, Paris, France, May 26–28, 2010. Springer, Berlin, Germany.

[16] Xavier Boyen, Qixiang Mei, and Brent Waters. Direct chosen ciphertext security from identity-based techniques. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05: 12th Conference on Computer and Communications Security*, pages 320–329, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press.

[17] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414, Prague, Czech Republic, May 2–6, 1999. Springer, Berlin, Germany.

[18] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, Texas, USA, May 23–26, 1998. ACM Press.

[19] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 523–552, French Riviera, May 30 – June 3, 2010. Springer, Berlin, Germany.

[20] Sanjit Chatterjee and Palash Sarkar. Generalization of the selective-ID security model for HIBE protocols. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 241–256, New York, NY, USA, April 24–26, 2006. Springer, Berlin, Germany.

[21] David Chaum, Jan-Hendrik Evertse, and Jeroen van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology – EUROCRYPT'87*, volume 304 of *Lecture Notes in Computer Science*, pages 127–141, Amsterdam, The Netherlands, April 13–15, 1988. Springer, Berlin, Germany.

[22] Jung Hee Cheon. Security analysis of the strong Diffie-Hellman problem. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 1–11, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Berlin, Germany.

[23] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Bounded CCA2-secure encryption. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 502–518, Kuching, Malaysia, December 2–6, 2007. Springer, Berlin, Germany.

[24] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In *ACM CCS 99: 6th Conference on Computer and Communications Security*, pages 46–51, Kent Ridge Digital Labs, Singapore, November 1–4, 1999. ACM Press.

[25] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the (in)security of rsa signatures. Cryptology ePrint Archive, Report 2011/087, 2011. `http://eprint.iacr.org/`.

[26] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 65–82, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.

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

[28] P. Erdös, P. Frankel, and Z. Furedi. Families of finite sets in which no set is covered by the union of r others. *Israeli Journal of Mathematics*, 51:79–89, 1985.

[29] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

[30] Marc Fischlin. The Cramer-Shoup strong-RSA signature scheme revisited. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 116–129, Miami, USA, January 6–8, 2003. Springer, Berlin, Germany.

[31] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139, Prague, Czech Republic, May 2–6, 1999. Springer, Berlin, Germany.

[32] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

[33] 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.

[34] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, fifth edition, 1979.

[35] Swee-Huay Heng and Kaoru Kurosawa. k-resilient identity-based encryption in the standard model. In Tatsuaki Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 67–80, San Francisco, CA, USA, February 23–27, 2004. Springer, Berlin, Germany.

[36] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 553–571, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Berlin, Germany.

[37] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 21–38, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Berlin, Germany.

[38] Dennis Hofheinz and Eike Kiltz. Practical chosen ciphertext secure encryption from factoring. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 313–332, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Germany.

[39] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. *Journal of Cryptology*, pages ???–???, 2011.

[40] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 333–350, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Germany.

[41] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 654–670, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Berlin, Germany.

[42] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 581–600, New York, NY, USA, March 4–7, 2006. Springer, Berlin, Germany.

[43] Eike Kiltz and David Galindo. Direct chosen-ciphertext secure identity-based key encapsulation without random oracles. *Theor. Comput. Sci.*, 410(47-49):5093–5111, 2009.

[44] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient authentication from hard learning problems. In *EUROCRYPT*, pages ???–???, 2011.

[45] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *ISOC Network and Distributed System Security Symposium – NDSS 2000*, San Diego, California, USA, February 2–4, 2000. The Internet Society.

[46] Ravi Kumar, Sridhar Rajagopalan, and Amit Sahai. Coding constructions for blacklisting problems without computational assumptions. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 609–623, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Germany.

[47] Vadim Lyubashevsky and Daniele Micciancio. Asymptotically efficient lattice-based digital signatures. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 37–54, San Francisco, CA, USA, March 19–21, 2008. Springer, Berlin, Germany.

[48] Shigeo Mitsunari, Ryuichi Saka, and Masao Kasahara. A new traitor tracing. *IEICE Transactions*, E85-A(2):481–484, February 2002.

[49] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 80–99, New York, NY, USA, March 4–7, 2006. Springer, Berlin, Germany.

[50] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, Maryland, USA, May 14–16, 1990. ACM Press.

[51] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473, Aarhus, Denmark, May 22–26, 2005. Springer, Berlin, Germany.

[52] Sven Schäge and Jörg Schwenk. A CDH-based ring signature scheme with short signatures and public keys. In Radu Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 129–142, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Berlin, Germany.

[53] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.

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

A A bound on the size of the randomness

The efficiency of our two signature schemes from Section 5 and 4 depends on the randomness $s \in \{0,1\}^\lambda$. Following [37] we now show that it suffices to choose $\lambda = \log q + k/m$, where q is the number of allowed signature queries and m is the parameter from the $(m,1)$-PHF.

Following the concrete security approach of Bellare and Ristenpart [5], we define the *success ratio* of an adversary $\mathcal{A}$ running in time t and having success probability ϵ as $\mathbf{SR}(\mathcal{A}) = \epsilon/t$. We require that a cryptosystem should be implemented with security parameter ("bits of security") k such that $\mathbf{SR}(\mathcal{A}) \leq 2^{-k+1}$ for any adversary $\mathcal{A}$.

q-DH BASED SIGNATURES. For an $(t_\mathcal{A}, \epsilon_\mathcal{A})$-adversary $\mathcal{A}$ against the EUF-CMA security of $\mathsf{Sig}_{q\text{-}\mathsf{DH}}$, we relate the success ratio of $\mathcal{A}$ to the success ratio of an $(t_\mathcal{B}, \epsilon_\mathcal{B})$-adversary $\mathcal{B}$ against the q-DH problem. Assuming $t_\mathcal{A} = t_\mathcal{B}$ for simplicity (in Theorem 5.1 we have $t_\mathcal{A} = t \approx t' = t_\mathcal{B}$) and using that $\gamma = 0$ for all our programmable hash functions over known-order groups, we apply the bound from Theorem 5.1 to obtain

$$\mathbf{SR}(\mathcal{A}) \leq \frac{1}{t_\mathcal{B}} \cdot \left(q\frac{\epsilon_\mathcal{B}}{\delta\delta'} + \frac{q^{m+1}}{2^{m\lambda}} \right) = \frac{q}{\delta\delta'} \cdot \mathbf{SR}(\mathcal{B}) + \frac{q^{m+1}}{2^{m\lambda}} \cdot \frac{1}{t_\mathcal{B}} \leq \frac{q}{\delta\delta'} \cdot \mathbf{SR}(\mathcal{B}) + \frac{q^m}{2^{m\lambda}} \; .$$

Now, clearly we have $\mathbf{SR}(\mathcal{A}) \leq 2^{-k+1}$, if both $\frac{q}{\delta\delta'} \cdot \mathbf{SR}(\mathcal{B}) \leq 2^{-k}$ and $\frac{q^{m+1}}{2^{m\lambda}} \leq 2^{-k}$. The relevant bound to determine λ is the second equation which yields to $\lambda \geq \log q + k/m$, as claimed.[5]

[5]Strictly speaking, one would also have to take the first bound into account which leads to an incease of the group

RSA-based Signatures. We follow a similar approach to derive the parameter sizes for our RSA-based constructions. For an $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}})$-adversary $\mathcal{A}$ against the EUF-CMA security of $\mathsf{Sig}_{\mathsf{RSA}}$, we relate the success ration of $\mathcal{A}$ to the success ratio of an $(t_{\mathcal{B}}, \epsilon_{\mathcal{B}})$-adversary $\mathcal{B}$ against the RSA problem. Let us first simplify the bound of Theorem 4.1 a little. We assume that

$$2^{-k-1} \geq (q+1)\lambda \left(3\epsilon'' + \frac{r(q+1)^2\lambda^2 + 2r + 1}{2^r} + \gamma + \frac{1}{2^{r-l}}\right) .$$

For instance, consider $k = 80$, then usual choices are $r = \lfloor \log n \rfloor = 1024$, $l = 160$, $q < 2^{80}$, and we have $\gamma \leq 16m^2 l/\sqrt{n}$ for our constructions in Section 3. If we also assume that the pseudorandom function is secure, then the above assumption seems reasonable. Applying Theorem 4.1 we thus have

$$\begin{aligned}
\mathbf{SR}(\mathcal{A}) &\leq \frac{1}{t_{\mathcal{B}}} \cdot \left(\frac{4r^2}{\delta}\left((q+1)\lambda\epsilon_{\mathcal{B}} + \frac{r}{l \cdot 2^{r-l-1}}\right) + \frac{1}{2^{k+1}} + \frac{q^{m+1}}{2^{m\lambda}}\right) \\
&= \frac{4r^2}{\delta}\left((q+1)\lambda \cdot \mathbf{SR}(\mathcal{B}) + \frac{r}{l \cdot 2^{r-l-1} \cdot t_{\mathcal{B}}}\right) + \frac{1}{2^{k+1} \cdot t_{\mathcal{B}}} + \frac{q^{m+1}}{2^{m\lambda} \cdot t_{\mathcal{B}}} \\
&\leq \frac{4r^2}{\delta}\left((q+1)\lambda \cdot \mathbf{SR}(\mathcal{B}) + \frac{r}{l \cdot 2^{r-l-1}}\right) + \frac{1}{2^{k+1}} + \frac{q^m}{2^{m\lambda}} .
\end{aligned}$$

Assuming that the best way of solving an RSA instance is factoring the modulus N, we require that N is chosen such that $2^{-k-1} \geq \frac{4r^2}{\delta}\left((q+1)\lambda \cdot \mathbf{SR}(\mathcal{B}) + \frac{r}{l \cdot 2^{r-l-1}}\right)$ for all $\mathcal{B}$. Then again we have $\mathbf{SR}(\mathcal{A}) \leq 2^{-k+1}$, if $\frac{q^m}{2^{m\lambda}} \leq 2^{-k}$, thus it suffices to set $\lambda \geq \log q + k/m$.

B Strong q-problems

Let $S(c)$ be the solution to a problem instance, such that $S(c)$ is hard to compute from c. In a *q-problem* an adversary is given q (polynomially many) "solved problem instances" $(c_i, S(c_i))$ (for random c_i) and some challenge instance $c \notin \{c_1, \ldots, c_q\}$ and has to compute $S(c)$. In Strong q-problems an adversary only has to come up with a fresh solved instance $(c, S(c))$ for an *arbitrary* $c \notin \{c_1, \ldots, c_q\}$ of its choice. *Strong* q-problems are naturally well-suited for building (weakly secure) signature schemes by defining the signature on M to be $S(M)$, whereas it seems more difficult from (standard) q-problems. To understand the difference between strong and non-strong problems, let us verify that the (Strong) q-DH and the (Strong) RSA problems can be naturally interpreted as (Strong) q-problems.

For the q-DH case, let g and $h = g^x$ be two generators of a prime order group $\mathbb{G}$. A problem instance for $c \in \mathbb{Z}_{|\mathbb{G}|}$ is given by $S(c) := g^{1/(x+c)}$. It is well-known [48] that the q-problem is equivalent to the q-DH problem which is given $g, g^x, \ldots, g^{x^q}$, compute $g^{1/x}$. The same related also holds between the Strong q-problem and the Strong q-DH problem (in which the adversary has to compute $(c, g^{1/(x+c)})$ for any $c \in \mathbb{Z}_{|\mathbb{G}|}$).

In the RSA case let $N = pq$ be an RSA modulus and let $y \in \mathbb{Z}_N$. A problem instance for a prime e is given by $S(e) = y^{1/e} \bmod N$. It is implicit in many prior works (e.g., [24]) that the corresponding q-problem is equivialent to the RSA problem which is to compute $y^{1/e} \bmod N$ for a

$\mathbb{G}$ that is mostly ignored in the literature. Assuming Cheon's attack [22] is the optimal success ratio for attacks against the q-DH problem (i.e., $\mathbf{SR}(\mathcal{B}) \leq \sqrt{q/p}$ for all adversaries $\mathcal{B}$) we obtain $\log p \geq 3\log q - 2\log \delta\delta' + 2k$.

given e.[6] The same relation also hold between the Strong q-problem and the Strong RSA problem which is to compute $(e, y^{1/e} \bmod N)$ for any $e > 2$.

C Omitted Proofs

C.1 Proof of Theorem 4.1

In the following let M_i denote the i-th query to the signing oracle, and let (σ_i, s_i) denote the reply. Let (M^*, σ^*, s^*) be the forgery output by $\mathcal{F}$. We distinguish between two types of forgers. A *Type I* forger returns (M^*, σ^*, s^*) with $s^* = s_i$ for some $i \in [q]$. A *Type II* forger returns (M^*, σ^*, s^*) with $s^* \neq s_i$ for all $i \in [q]$.

C.1.1 Type I forgers.

Lemma C.1. *Let $\mathcal{F}$ be a type I forger that (t, q, ϵ)-breaks the existential unforgeability of $\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}]$. Then there exists an adversary $\mathcal{A}$ that (t', ϵ')-breaks the RSA assumption with $t' \approx t$ and*

$$\epsilon' \geq \frac{1}{4r^2}\left(\frac{\delta}{q}\left(\epsilon - \frac{q^{m+1}}{2^{m\lambda}}\right) - \gamma - \frac{1}{2^{r-l}} - 3\epsilon'' - \frac{r(q^2\lambda^2 + 2q\lambda) + 1}{2^r}\right) - \frac{r}{2^{r-l-1}}.$$

We proceed in games. Let X_i denote the probability that $\mathcal{F}$ is successful in Game i.

Game 0. We define Game 0 as the existential unforgeability experiment with forger $\mathcal{F}$. By definition, we have

$$\Pr[X_0] = \epsilon.$$

Game 1. Now the simulator aborts if there exist (at least) $m+1$ indices $i_1, \ldots, i_{m+1}$, such that $s_j = s_{j'}$ for all $j, j' \in \{i_1, \ldots, i_{m+1}\}$. By Lemma 2.3 we have

$$\Pr[X_1] \geq \Pr[X_0] - \frac{q^{m+1}}{2^{m\lambda}}.$$

Game 2. In this game the simulator chooses the randomness $s_1, \ldots, s_q$ in advance, guesses the index i such that $s^* = s_i$ is used by $\mathcal{F}$ in the forged signature, and aborts if $\mathcal{F}$ outputs a forgery (M', σ', s') with $s' \neq s^*$. Since $s^* \in \{s_1, \ldots, s_q\}$ we have

$$\Pr[X_2] \geq \frac{1}{q}\Pr[X_1].$$

Game 3. In this game we run the trapdoor key generation algorithm $\mathsf{PHF.TrapGen}$ to generate the group hash function H. Let $E = \bigcup_{i=1}^{q}\{s_i\}$ and $E^* = E \backslash \{s^*\}$. The simulator picks $\hat{g} \xleftarrow{\$} \mathbb{Z}_n^*$ and $\hat{h} \xleftarrow{\$} \mathsf{QR}_N$ and sets

$$g = \hat{g}^{2\prod_{t \in E^*} \mathsf{P}(t)} \text{ and } h = \hat{h}^{\prod_{t \in E} \mathsf{P}(t)}.$$

Then it runs $(\kappa, \tau) \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$ to generate hash key κ together with trapdoor τ. Since H is $(m, 1, \gamma, \delta)$-programmable, we have

$$\Pr[X_3] \geq \Pr[X_2] - \gamma.$$

[6] This is since given y_0 and c, $y = y_0^{\prod e_i}$ can be computed that allows to establish q random solved instaces $(e_i, S(e_i) = y^{1/e_i})$ and use $S(e) = y^{1/e}$ to compute $y_0^{1/e}$.

Game 4. In this game the simulator picks a prime e uniformly from $\mathbb{Z}_{2^r}$. Moreover, it computes $(a_i, b_i) \leftarrow \mathsf{PHF.TrapEval}(\tau, M_i)$ for each queried message M_i, $i \in [q]$, and $(a^*, b^*) \leftarrow \mathsf{PHF.TrapEval}(\tau, M^*)$ for the message M^* on which $\mathcal{F}$ forges a signature. The simulator aborts, if $a_i \neq 0$ for some $i \in [q]$ with $s_i = s^*$, or if $\gcd(a^*, e) \neq 1$, we denote this event with $\mathsf{abort}_{\mathsf{PHF}}$. Recall that there are at most m values s_i such that $s_i = s^*$ by Game 1. Furthermore, $e \geq 2^l$ except with probability $1/2^{r-l}$. Thus, using that H is $(m, 1, \gamma, \delta)$-evasively programmable, we have

$$\Pr[X_4] = \Pr[X_3 \wedge \neg \mathsf{abort}_{\mathsf{PHF}}] \geq \delta \cdot (\Pr[X_3] - \frac{1}{2^{r-l}}).$$

Game 5. Now the simulator computes a signature on some chosen-message M_i by running $(a_i, b_i) \leftarrow \mathsf{PHF.TrapEval}(t, M_i)$ first to determine (a_i, b_i), and then computing σ_i as

$$\sigma_i = \hat{g}^{2a_i \prod_{t \in E_i^*} \mathsf{P}(t)} \hat{h}^{b_i \prod_{t \in E_i} \mathsf{P}(t)} = \mathsf{H}(M)^{1/\mathsf{P}(s_i)}, \tag{4}$$

where $E_i = E \backslash \{s_i\}$ and $E_i^* = E^* \backslash \{s_i\}$. The latter equality uses that $a_i = 0$ for all M_i with $s_i = s^*$ by Game 4. This change is only conceptual, and thus

$$\Pr[X_5] = \Pr[X_4].$$

Game 6. The simulator in Game 6 aborts if two different prefixes are mapped to the same prime. That is, we abort if there exist $s, s' \in \{s_1, \ldots, s_q\}$ and $i, j \in [\lambda]$ such that $\mathsf{F}(s_{|i}) = \mathsf{F}(s'_{|j})$ and $s_{|i} \neq s'_{|j}$. This event is denoted with $\mathsf{abort}_{\mathsf{coll}}$.

Recall that $\mathsf{F}(z) = \mathsf{PRF}_K(\mu || z) \oplus c$, where μ is incremented until $\mathsf{PRF}_K(\mu || z) \oplus c$ is prime. Let us assume PRF_K is replaced with a truly random function. Then evaluating F is equivalent to sampling a uniformly random r-bit prime. There are at least $2^r/r$ such primes by Lemma 2.1, and at most $q\lambda$ primes are sampled. Applying Lemma 2.3, we conclude that the collision probability for a truly random function is at most $rq^2\lambda^2 \cdot 2^{-r}$.

Now consider the case where the truly random function is instantiated with PRF_K, and suppose that a collision occurs with probability $\Pr[\mathsf{abort}_{\mathsf{coll}}]$. Then this would allow an attack distinguishing PRF_K from a random function with probability at least $\epsilon_{\mathsf{coll}} \geq \Pr[\mathsf{abort}_{\mathsf{coll}}] - \frac{rq^2\lambda^2}{2^r}$. Since we have $\epsilon_{\mathsf{coll}} \leq \epsilon''$ by assumption, this implies $\Pr[\mathsf{abort}_{\mathsf{coll}}] \leq \epsilon'' + \frac{rq^2\lambda^2}{2^r}$, and thus

$$\Pr[X_6] \geq \Pr[X_5] - \Pr[\mathsf{abort}_{\mathsf{coll}}] \geq \Pr[X_5] - \epsilon'' - \frac{rq^2\lambda^2}{2^r}.$$

Game 7. Now we abort if there exists $(i, j) \in [q] \times [\lambda]$ such that $\mathsf{F}(s_{i|j})$ divides $\phi(n)$; we denote this event with $\mathsf{abort}_{\mathsf{div}}$. Recall that $\phi(n) = 4p'q'$ and that F returns only odd primes. Again replacing PRF_K with a truly random function, the probability that one out of at most $q\lambda$ randomly chosen odd r-bit primes equals one of the two odd primes dividing $\phi(n)$ is at most $(q\lambda 2r)/2^r$.

Similar to Game 6, we can mount a distinguishing attack against PRF_K with success probability at least $\epsilon_{\mathsf{div}} \geq \Pr[\mathsf{abort}_{\mathsf{div}}] - (q\lambda 2r)/2^r$. By assumption we have $\epsilon_{\mathsf{div}} \leq \epsilon''$, and thus

$$\Pr[X_7] \geq \Pr[X_6] - \Pr[\mathsf{abort}_{\mathsf{div}}] \geq \Pr[X_6] - \epsilon'' - \frac{q\lambda 2r}{2^r}.$$

Game 8. The simulator in this game proceeds just like the simulator in Game 7, except that we add an abort condition. The simulator aborts if for some $s_{i|j}$, $(i,j) \in [q] \times [\lambda]$, the resolving index μ is greater than r^2. We denote this event with abort_μ.

Let us again assume PRF_K is replaced with a truly random function, and let us consider the probability of not finding a prime by evaluating the random function r^2 times and computing the exclusive or with c. This is equivalent to sampling r^2 uniform r-bit strings. Lemma 2.1 tells us that the probability of finding a prime by sampling r random bits is at least $1/r$, thus the probability of not finding a prime in r^2 trials is at most $(1-1/r)^{r^2}$.

We can therefore construct an adversary distinguishing PRF_K from a random permutation with probability at least $\epsilon_\mu \geq \mathsf{abort}_\mu - (1-1/r)^{r^2} \geq \mathsf{abort}_\mu - 1/2^r$, where the latter inequality uses that $(1-1/r)^r \leq 1/2$ for all $r \in \mathbb{N}$. [7] Since we must have $\epsilon_\mu \leq \epsilon''$, this implies

$$\Pr[X_8] \geq \Pr[X_7] - \epsilon'' - \frac{1}{2^r}.$$

Game 9. In this game the simulator picks $\mu^* \xleftarrow{\$} [r^2]$ uniformly random, and aborts if μ^* is not the resolving index of s^*. We denote this event $\mathsf{abort}_{\mathsf{guess}\mu^*}$. Due to the changes introduced in Game 8 we know that the resolving index of s^* lies in the interval $[1, r^2]$. Thus we have $\Pr[\mathsf{abort}_{\mathsf{guess}\mu}] = 1 - 1/r^2$, and therefore

$$\Pr[X_9] \geq \Pr[X_8 \wedge \neg \mathsf{abort}_{\mathsf{guess}\mu}] = \frac{1}{r^2} \cdot \Pr[X_8].$$

Note that the resolving index μ^* is now uniformly distributed, and has the property that $e = \mathsf{PRF}(\mu || s^*) \oplus c$ is prime.

Game 10. Recall that c is uniformly distributed, and that we abort if μ^* is not the resolving index of s^* (Game 9). The latter implies that $\mathsf{PRF}(\mu^* || s^*) \oplus c$ is prime, thus e has the distribution of a uniformly random prime.

In this game the simulator determines c differently. Instead of sampling c at random, the simulator sets $c = \mathsf{PRF}(\mu^* || s^*) \oplus e$, where e is the random prime the simulator chooses starting from Game 4. Observe that this defines $\mathsf{F}(s^*) = e$. The distribution of μ^*, c, and e is not altered, thus we have

$$\Pr[X_{10}] = \Pr[X_9].$$

The RSA Adversary. The adversary receives a RSA-challenge (N', e', y) as input, and aborts if e is not a prime, $e > 2^r$, or $e < 2^l$. Otherwise the adversary sets $N = N'$, $\hat{g} = y$, $e = e'$, and proceeds like the simulator in Game 10. Recall that $e = e'$ defines $\mathsf{F}(s^*) = e$.

Since we have set $r = \lfloor \log_2 N \rfloor - 1$, the probability that $e \geq 2^r$ is at most $1/4$. Moreover, among the at least $2^r/r$ primes from which e is chosen uniformly, there are at most $2^{l+1}/l$ primes smaller than 2^l (Lemma 2.1). Thus, the success probability of the adversary is at least

$$\Pr[X_{10} \wedge e < 2^r] - \Pr[e < 2^l] \geq \frac{1}{4} \cdot \Pr[X_{10}] - \frac{r}{l 2^{r-l-1}}$$

Answering Signing Queries. Due to the changes introduced in Games 2 to 5, the adversary does not need to know the factorization of N, since it uses the $\mathsf{PHF.TrapEval}$ algorithm and Equation (4) to answer signing queries.

[7] This holds since $(1-1/r_0)^{r_0} < (1-1/r_1)^{r_1}$ for $r_0, r_1 \in \mathbb{N}$ with $r_0 < r_1$, and $\lim_{r\to\infty}(1-1/r)^r = 1/\tilde{e}$, where $\tilde{e} = 2.71828...$ is Euler's number.

Extracting the Solution to the RSA Challenge. Eventually, the forger returns a forgery (M^*, σ^*, s^*), from which the adversary extracts the solution to the RSA challenge as follows. First it computes $z' = \frac{\sigma^*}{\hat{h}^{b^* \prod_{t \in E^*} \mathsf{P}(t)}}$. Observe here that

$$\begin{aligned} z' &= \frac{\sigma^*}{\hat{h}^{b^* \prod_{t \in E^*} \mathsf{P}(t)}} = \frac{\mathsf{H}(M^*)^{1/\mathsf{P}(s^*)}}{\hat{h}^{b^* \prod_{t \in E^*} \mathsf{P}(t)}} = \frac{\left(g^{a^*} h^{b^*}\right)^{1/\mathsf{P}(s^*)}}{\hat{h}^{b^* \prod_{t \in E^*} \mathsf{P}(t)}} \\ &= \frac{\hat{g}^{\frac{2a^* \prod_{t \in E^*} \mathsf{P}(t)}{\mathsf{P}(s^*)}} \hat{h}^{b^* \prod_{t \in E^*} \mathsf{P}(t)}}{\hat{h}^{b^* \prod_{t \in E^*} \mathsf{P}(t)}} = \hat{g}^{\frac{2a^* \prod_{t \in E^*} \mathsf{P}(t)}{\mathsf{P}(s^*)}} \end{aligned}$$

From this it computes $y' = z'^{\prod_{i=1}^{\lambda-1} \mathsf{F}(s^*_{|i})}$, which equals

$$y' = z'^{\prod_{i=1}^{\lambda-1} \mathsf{F}(s^*_{|i})} = \left(\hat{g}^{\frac{2a^* \prod_{t \in E^*} \mathsf{P}(t)}{\mathsf{P}(s^*)}}\right)^{\prod_{i=1}^{\lambda-1} \mathsf{F}(s^*_{|i})} = \hat{g}^{\frac{2a^* \prod_{t \in E^*} \mathsf{P}(t)}{e}} = y^{\frac{2a^* \prod_{t \in E^*} \mathsf{P}(t)}{e}}$$

We have $\gcd(2a^* \prod_{t \in E^*} \mathsf{P}(t), e) = 1$ because e is odd, $e \nmid \prod_{t \in E^*} \mathsf{P}(t)$ by Game 6, and $\gcd(e, a^*) = 1$ by Game 4. Thus, x with $x^e \equiv y \bmod N$ can be extracted from y' using Lemma 2.2.

C.1.2 Type II forgers.

Lemma C.2. *Let $\mathcal{F}$ be a type II forger that (t, q, ϵ)-breaks the existential unforgeability of $\mathsf{Sig}_{\mathsf{RSA}}[\mathsf{H}]$. Then there exists an adversary $\mathcal{A}$ that (t', ϵ')-breaks the RSA assumption with $t' \approx t$ and*

$$\epsilon' \geq \frac{\delta}{4r^2} \left(\frac{\epsilon - \gamma}{(q+1)\lambda} - 3\epsilon'' - \frac{r(q+1)^2\lambda^2 + 2r + 1}{2^r} \right) - \frac{r}{l 2^{r-l-1}}.$$

Let X_i denote the probability that $\mathcal{F}$ is successful in Game i.

Game 0. Game 0 is the existential unforgeability experiment with forger $\mathcal{F}$, thus we have

$$\Pr[X_0] = \epsilon.$$

Game 1. In this game the simulator chooses the randomness $s_1, \ldots, s_q$ in advance, and runs the trapdoor key generation algorithm $\mathsf{PHF.TrapGen}$ to generate the group hash function H. Again let $E = \bigcup_{i=1}^{q} \{s_i\}$. The simulator picks $\hat{g} \xleftarrow{\$} \mathbb{Z}_n^*$, $\hat{h} \xleftarrow{\$} \mathsf{QR}_N$, and a uniformly random prime $e \xleftarrow{\$} [0, 2^r - 1]$. Then it sets

$$g = \hat{g}^{2 \prod_{t \in E} \mathsf{P}(t)} \text{ and } h = \hat{h}^{e \prod_{t \in E} \mathsf{P}(t)}.$$

Then it runs $(\kappa, \tau) \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$. By the $(m, 1, \gamma, \delta)$-programmability of H, we have

$$\Pr[X_1] \geq \Pr[X_0] - \gamma.$$

Game 2. Again we modify the simulator such that chosen-message queries are answered without inverting exponents. Again let $E_i = E \backslash \{s_i\}$, and let

$$g_i = \hat{g}^{2 \prod_{t \in E_i} \mathsf{P}(t)} \text{ and } h_i = \hat{h}^{e \prod_{t \in E_i} \mathsf{P}(t)}.$$

The simulator computes a signature on M_i by running $(a_i, b_i) \leftarrow \mathsf{PHF.TrapEval}(t, M_i)$ first to determine (a_i, b_i), and then computing σ_i as

$$\sigma_i = g_i^{a_i} h_i^{b_i} = \mathsf{H}(M)^{1/\mathsf{P}(s_i)}. \tag{5}$$

Game 2 is perfectly indistinguishable from Game 1. Thus,

$$\Pr[X_2] = \Pr[X_1].$$

Game 3. In this game the simulator guesses the shortest prefix ψ of s^* that differs from all prefixes of $s_1, \ldots, s_q$. Note that this prefix must exist, because the Type II-forger will return a forgery (M^*, σ^*, s^*) with $s^* \notin \{s_1, \ldots, s_q\}$.

To this end, the simulator proceeds as follows. If $q = 0$, it samples a bit $\psi \xleftarrow{\$} \{0,1\}$ at random, and aborts if the forger returns s^* with $s^*_{|1} \neq \psi$. If $q \geq 0$, the simulator picks $i \in [q]$ and $j \in [\lambda]$, and sets $\psi = s_{i|j-1}||b$, where b is the complement of the j-th bit of s_i. (Recall that we defined the 0-th prefix as the empty string $\emptyset$, thus $s_{i|0} = \emptyset$). The simulator aborts if either

- ψ is a prefix of some $s_i \in \{s_1, \ldots, s_q\}$, that is, there exists (i', j') such that $\psi = s_{i'|j'}$, or if
- the forger returns (M^*, σ^*, s^*) such that ψ is not a prefix of s^*.

We denote this event with $\mathsf{abort}_{\mathsf{prefix}}$. If $q = 0$, then the simulator aborts with probability $1/2$. Otherwise there are $q\lambda$ possible choices for $(i, j) \in [q] \times [\lambda]$. Thus we have $\Pr[\mathsf{abort}_{\mathsf{prefix}}] \leq 1 - 1/((q+1)\lambda)$, and therefore

$$\Pr[X_3] \geq \Pr[X_2 \wedge \neg \mathsf{abort}_{\mathsf{prefix}}] \geq \frac{1}{(q+1)\lambda} \cdot \Pr[X_2].$$

Game 4. We add an abort condition. The simulator aborts, if $\mathsf{F}(\psi) \mid \prod_{t \in E} \mathsf{P}(t)$, or (equivalently) $\mathsf{F}(\psi) = \mathsf{F}(s_{i|j})$ for some $(i, j) \in [q] \times [\lambda]$. We denote this event with $\mathsf{abort}_{\psi\mathsf{coll}}$.

Note that $\psi \neq s_{i|j}$ for all $(i, j) \in [q] \times [\lambda]$. Thus, if we replace PRF_K with a truly random function, then according to Lemma 2.3 the probability of a collision among (at most) $(q+1)\lambda$ uniformly random primes from $[0, 2^r - 1]$ is bounded by $r(q+1)^2\lambda^2 \cdot 2^{-r}$.

We can therefore construct an adversary distinguishing PRF_K from a random function with probability at least $\epsilon_{\psi\mathsf{coll}} \geq \Pr[\mathsf{abort}_{\psi\mathsf{coll}}] - r(q+1)^2\lambda^2 \cdot 2^{-r}$. Since $\epsilon_{\psi\mathsf{coll}} \leq \epsilon''$ by assumption, we have $\Pr[\mathsf{abort}_{\mathsf{coll}}] \leq \epsilon'' + r(q+1)^2\lambda^2 \cdot 2^{-r}$, and thus

$$\Pr[X_4] \geq \Pr[X_3] - \Pr[\mathsf{abort}_{\psi\mathsf{coll}}] \geq \Pr[X_3] - \epsilon'' - \frac{r(q+1)^2\lambda^2}{2^r}.$$

Game 5. We introduce a number of changes to the simulator, which equal the modifications introduced in Games 7 to 10 in the proof of Lemma C.1.

- We abort if there exists $i, j \in [q] \times [\lambda]$ such that $\mathsf{F}(s_{i|j})$ divides $\phi(n)$. (Game 7)
- We abort if the resolving index μ is greater than r^2 for some $s_{i|j}$, $(i, j) \in [q] \times [\lambda]$. (Game 8)
- We pick $\mu^* \xleftarrow{\$} [r^2]$ uniformly random, and abort if μ^* is not the resolving index of ψ. (Game 9)

- Instead of sampling c at random, we set $c = \mathsf{PRF}(\mu^*||\psi) \oplus e$, where e is the uniformly random prime that the simulators pick starting from Game 1. Observe that this defines $\mathsf{F}(\psi) = e$. (Game 10)

With the same arguments as in the proof of Lemma C.1 we have

$$\Pr[X_5] \geq 1/r^2 \left(\Pr[X_4] - 2\epsilon'' - \frac{2r+1}{2^r} \right).$$

Game 6. In this game the simulator picks a prime e uniformly from the interval $[2^l, 2^r - 1]$, computes $(a^*, b^*) \leftarrow \mathsf{PHF.TrapEval}(\tau, M^*)$ for the message M^* on which $\mathcal{F}$ forges a signature, and if $\gcd(a^*, e) \neq 1$, we denote this event with $\mathsf{abort}_{\mathsf{PHF}}$. Using that H is $(m, 1, \gamma, \delta)$-evasively programmable, we have

$$\Pr[X_6] = \Pr[X_5 \wedge \neg \mathsf{abort}_{\mathsf{PHF}}] \geq \delta \cdot \Pr[X_5].$$

The RSA Adversary. On input a RSA-challenge (N', e', y), the adversary aborts if e is not a prime in the interval $[2^l, 2^r]$, and otherwise sets $N = N'$, $\hat{g} = y$, $e = e'$. The latter now defines $\mathsf{F}(\psi) = e$. Otherwise it proceeds like the simulator in Game 6.

As in the proof of Lemma C.1, the success probability of the RSA adversary is at least

$$\frac{1}{4} \cdot \Pr[X_6] - \frac{r}{l 2^{r-l-1}}.$$

Answering Signing Queries. Again the adversary can answer signing queries without computing inverses modulo $\phi(n)$ by using the $\mathsf{PHF.TrapEval}$ algorithm and Equation (5).

Extracting the Solution to the RSA Challenge. When the forger returns (M^*, σ^*, s^*), the adversary computes $w = \sigma^{*z}$, where $z = \prod_{\{i \in [\lambda] | s^*_{|i} \neq \psi\}} \mathsf{F}(s^*_{|i})$. Note that $z = \mathsf{P}(s^*)/e$, since $\mathsf{F}(\psi) = e$. Thus we have

$$w = \left(\mathsf{H}(M^*)^{1/\mathsf{P}(s^*)} \right)^z = \mathsf{H}(M^*)^{1/e} = \left(\hat{g}^{2a^* \prod_{t \in E} \mathsf{P}(t)} \hat{h}^{eb^* \prod_{t \in E} \mathsf{P}(t)} \right)^{1/e} = \hat{g}^{\frac{2a^* \prod_{t \in E} \mathsf{P}(t)}{e}} \hat{h}^{b^* \prod_{t \in E} \mathsf{P}(t)}.$$

From this the adversary computes y' as

$$y' = w \cdot \hat{h}^{-b^* \prod_{t \in E} \mathsf{P}(t)} = \hat{g}^{\frac{2a^* \prod_{t \in E} \mathsf{P}(t)}{e}}.$$

Again, if we have $\gcd(2a^* \prod_{t \in E} \mathsf{P}(t), e) = 1$, then the solution to the given RSA challenge can be extracted from y' using Lemma 2.2. We have $\gcd(2a^* \prod_{t \in E^*} \mathsf{P}(t), e) = 1$, since e is odd, $e \nmid \prod_{t \in E} \mathsf{P}(t)$ by Game 4, and $\gcd(e, a^*) = 1$ by Game 6.

C.2 Proof of Theorem 5.1

Again let M_i denote the i-th query to the signing oracle, and let (σ_i, s_i) denote the reply. Let (M^*, σ^*, s^*) be the forgery output by $\mathcal{F}$.

We again distinguish between a *Type I* forger, returning (M^*, σ^*, s^*) with $s^* = s_i$ for some $i \in [q]$, and a *Type II* forger returning (M^*, σ^*, s^*) with $s^* \neq s_i$ for all $i \in [q]$.

C.2.1 Type I forgers

The following Lemma proves security against Type I forgers.

Lemma C.3. *Let $\mathcal{F}$ be a forger of type I that (t, q, ϵ)-breaks the existential unforgeability of $\mathsf{Sig}_{q\text{-}\mathsf{DH}}[\mathsf{H}, \mathsf{D}]$. Then there exists an adversary $\mathcal{A}$ that (t', ϵ')-breaks the q-DH assumption with $t' \approx t$ and*

$$\epsilon' \geq \delta\delta' \left(\frac{1}{q} \left(\epsilon - \frac{q^{m+1}}{2^{m\lambda}} \right) - \gamma \right).$$

Proof. We prove Lemma C.3 by a sequence of games. In the following let X_i denote the probability that $\mathcal{F}$ is successful in Game i.

Game 0. We define Game 0 as the existential unforgeability experiment with forger $\mathcal{F}$. By definition, we have

$$\Pr[X_0] = \epsilon.$$

Game 1. Now the simulator aborts if there exist (at least) $m+1$ indices $i_1, \ldots, i_{m+1}$, such that $s_j = s_{j'}$ for all $j, j' \in \{i_1, \ldots, i_{m+1}\}$. We denote this event with $\mathsf{abort}_{\mathsf{mColl}}$. The s_i are picked uniformly from $\{0,1\}^\lambda$, thus by Lemma 2.3 the probability of an $m+1$-wise collision is at most $\Pr[\mathsf{abort}_{\mathsf{mColl}}] \leq q^{m+1}/2^{m\lambda}$, which implies

$$\Pr[X_1] \geq \Pr[X_0] - \Pr[\mathsf{abort}_{\mathsf{mColl}}] \geq \Pr[X_0] - \frac{q^{m+1}}{2^{m\lambda}}.$$

Game 2. In this game the simulator chooses $s_1, \ldots, s_q \xleftarrow{\$} \{0,1\}^\lambda$ at the beginning of the game, and uses s_i as randomness for the signature of M_i. This change is purely conceptual and oblivious to $\mathcal{F}$, thus does not affect the success probability of $\mathcal{F}$.

Moreover, the simulator guesses the value s^* that will be used by $\mathcal{F}$ in the forged signature, and aborts if $\mathcal{F}$ outputs a forgery (M', σ', s') with $s' \neq s^*$. We denote this event with $\mathsf{abort}_{\mathsf{guess}}$. Since we assume that $s^* \in \bigcup_{i=1}^q \{s_i\}$, we have $\Pr[\mathsf{abort}_{\mathsf{guess}}] \leq 1 - 1/q$, and thus

$$\Pr[X_2] = \Pr[X_1 \wedge \neg \mathsf{abort}_{\mathsf{guess}}] \geq \frac{1}{q} \Pr[X_1].$$

Game 3. We define Game 3 like the previous game, except that we run the trapdoor key generation algorithm $\mathsf{PHF.TrapGen}$ to generate the hash function H. In the following let $E = \bigcup_{i=1}^q \{s_i\}$ and $E^* = E \backslash \{s^*\}$. The simulator sets

$$g = \hat{g}^{\prod_{t \in E^*} \mathsf{d}(t)} \text{ and } h = \hat{g}^{\prod_{t \in E} \mathsf{d}(t)}.$$

Recall that $\hat{g}$ denotes the generator chosen by the Gen procedure. Then the simulator runs $(\kappa, \tau) \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$ to generate hash key κ together with trapdoor τ. Since H is $(m, 1, \gamma, \delta)$-programmable, we have

$$\Pr[X_3] \geq \Pr[X_2] - \gamma.$$

Game 4. In this game, the simulator computes $(a_i, b_i) \leftarrow \mathsf{PHF.TrapEval}(\tau, M_i)$ for each queried message M_i, and $(a^*, b^*) \leftarrow \mathsf{PHF.TrapEval}(\tau, M^*)$ for the message M^* on which $\mathcal{F}$ forges a signature. The simulator aborts, if $a_i \neq 0$ for some $i \in \{i \mid s_i = s^*\}$, or if $a^* = 0$. We denote this event with

$\mathsf{abort}_{\mathsf{PHF}}$. Recall that $|\{i \mid s_i = s^*\}| \leq m$ due to the modification introduced in Game 1. Thus, using that H is $(m, 1, \gamma, \delta)$-programmable, we have $\Pr[\mathsf{abort}_{\mathsf{PHF}}] \leq 1 - \delta$. This implies

$$\Pr[X_4] = \Pr[X_3 \wedge \neg\mathsf{abort}_{\mathsf{PHF}}] \geq \delta \cdot \Pr[X_3].$$

Game 5. Now we change the way chosen-message queries are answered by the simulator. Note that, by the setup of g and h introduced in Game 3, we have

$$\mathsf{H}(M_i) = g^{a_i} h^{b_i} = \hat{g}^{a_i \prod_{t \in E^*} \mathsf{d}(t)} \hat{g}^{b_i \prod_{t \in E} \mathsf{d}(t)}.$$

Let E_i^* and E_i denote the sets $E_i^* = E^* \backslash \{s_i\}$ and $E_i = E \backslash \{s_i\}$, and let

$$g_i := \hat{g}^{\prod_{t \in E_i^*} \mathsf{d}(t)} \text{ and } h_i := \hat{g}^{\prod_{t \in E_i} \mathsf{d}(t)}.$$

The simulator computes a signature on M_i by running $(a_i, b_i) \leftarrow \mathsf{PHF.TrapEval}(\tau, M_i)$ first to determine (a_i, b_i), and then computing σ_i as

$$\sigma_i = g_i^{a_i} h_i^{b_i} = \mathsf{H}(M_i)^{1/\mathsf{d}(s_i)}. \tag{6}$$

The last equation of (6) uses that $a_i = 0$ for all i with $s_i = s^*$ (cf. Game 4). Game 5 is perfectly indistinguishable from Game 4 from the adversary's perspective. Thus we have

$$\Pr[X_5] = \Pr[X_4].$$

Game 6. Now the simulator computes $(e_i, f_i) \leftarrow \mathsf{PHF.TrapEval}'(\tau', s_i)$ for each s_i at the beginning of the game. When $\mathcal{F}$ outputs a forged signature (M^*, s^*, σ^*), the simulator computes $(e^*, f^*) \leftarrow \mathsf{PHF.TrapEval}'(\tau', s^*)$. We abort the game, if $e_i \equiv 0 \bmod p$ for some $i \in [q]$ or if $e^* \neq 0$, and denote this event with $\mathsf{abort}_{\mathsf{PHF}}$. By the $(1, \mathsf{poly}, \gamma', \delta')$-programmability of D we have

$$\Pr[X_6] = \Pr[X_5 \wedge \neg\mathsf{abort}_{\mathsf{PHF}}] \geq \delta' \cdot \Pr[X_5].$$

The q-DH Adversary. We can now replace the simulator with adversary $\mathcal{A}$. The adversary receives a q-DH challenge $(\hat{g}, \hat{g}^y, \ldots, \hat{g}^{y^q})$ as input, and proceeds like the simulator from Game 6, but without knowing y explicitly.

Set-up of the Public Key. The adversary runs $(\kappa', \tau') \xleftarrow{\$} \mathsf{PHF.TrapGen}'(1^k, \hat{g}, \hat{g}^y)$ just like the original key generation algorithm. To see that the adversary can compute the input values g and h to $\mathsf{PHF.TrapGen}$ as required, recall that

$$g = \hat{g}^{\prod_{t \in E^*} \mathsf{d}(t)} = \hat{g}^{\prod_{t \in E^*} (e_t + y f_t)}.$$

Considering the term $\prod_{t \in E^*}(e_t + y f_t)$ as a polynomial in y, the adversary first computes the coefficients α_i of the expansion of the polynomial $\alpha(y) = \prod_{t \in E^*}(e_t + y f_t) = \sum_{i=1}^{q} \alpha_i y^i$. Then it sets

$$g = \prod_{i=1}^{q} \left(\hat{g}^{y^i}\right)^{\alpha_i} \left(= \hat{g}^{\sum_{i=1}^{q} \alpha_i y^i} = \hat{g}^{\prod_{t \in E^*}(e_t + y f_t)}\right).$$

This is possible, since (i) the adversary can compute the coefficients of $\alpha(y)$ from the (e_t, f_t), (ii) the adversary has received $(\hat{g}, \hat{g}^y, \ldots, \hat{g}^{y^q})$ as input, and (iii) the polynomial $\alpha(y)$ has degree at most $q - 1$.

Clearly h can be computed similarly, since (i) and (ii) apply as well, and the polynomial $\prod_{t\in E}(e_t + yf_t)$ has degree at most q.

Thus, the adversary can generate $\kappa \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$, just like the simulator in Game 6. The public key is set to $pk = (\hat{g}, \kappa, \kappa')$ as before. Note that the adversary knows τ', but not y (otherwise the q-DH challenge is solved immediately).

Answering Signing Queries. The adversary $\mathcal{A}$ can answer signature queries, even without explicit knowledge of y, by proceeding as follows. Note that the adversary can compute $(a_i, b_i) \leftarrow \mathsf{PHF.TrapEval}(\tau, M_i)$, since it knows the trapdoor τ. Thus, in order to compute a signature using Equation (6), it suffices to determine

$$g_i = \hat{g}^{\prod_{t\in E_i^*} \mathsf{d}(t)} = \hat{g}^{\prod_{t\in E_i^*}(e_t+yf_t)} \text{ and } h_i = \hat{g}^{\prod_{t\in E_i} \mathsf{d}(t)} = \hat{g}^{\prod_{t\in E_i}(e_t+yf_t)}.$$

This is possible by using $\hat{g}^y, \ldots, \hat{g}^{y^q}$ and applying the same technique that the adversary has used to compute g and h. The adversary simulates the challenger of Game 6 perfectly, thus we have

$$\Pr[X_6] = \Pr[X_5].$$

Extracting the Solution to the q-DH Challenge. It remains to show how the adversary can extract the solution to the q-DH challenge from a forged signature (M^*, σ^*, s^*). First, using that $a^* \neq 0$ by Game 4, the adversary computes z as

$$z = \left(\sigma^* / \hat{g}^{b^* \prod_{t\in E^*}(e_t+yf_t)}\right)^{f^*/a^*},$$

where the elements of the q-DH challenge are used to compute the term $\hat{g}^{b^* \prod_{t\in E^*}(e_t+yf_t)}$. Note that

$$\begin{aligned}
z &= \left(\sigma^* / \hat{g}^{b^* \prod_{t\in E^*} \mathsf{d}(t)}\right)^{f^*/a^*} = \left(\left(\mathsf{H}(M^*)^{1/\mathsf{d}(s^*)}\right) / \hat{g}^{b^* \prod_{t\in E^*} \mathsf{d}(t)}\right)^{f^*/a^*} \\
&= \left(\left((g^{a^*} h^{b^*})^{1/\mathsf{d}(s^*)}\right) / \hat{g}^{b^* \prod_{t\in E^*} \mathsf{d}(t)}\right)^{f^*/a^*} = \left(\left(g^{a^*/\mathsf{d}(s^*)} \hat{g}^{b^* \prod_{t\in E^*} \mathsf{d}(t)}\right) / \hat{g}^{b^* \prod_{t\in E^*} \mathsf{d}(t)}\right)^{f^*/a^*} \\
&= \left(g^{a^*/\mathsf{d}(s^*)}\right)^{f^*/a^*} = \left(g^{a^*/(e^*+yf^*)}\right)^{f^*/a^*} \\
&\overset{(*)}{=} \left(g^{a^*/yf^*}\right)^{f^*/a^*} = \left(\hat{g}^{\frac{a^* \prod_{t\in E^*}(e_t+yf_t)}{yf^*}}\right)^{f^*/a^*} = \hat{g}^{\frac{\prod_{t\in E^*}(e_t+yf_t)}{y}}.
\end{aligned}$$

Here, $(*)$ uses that $e^* = 0$ (by Game 6). Then $\mathcal{A}$ determines the coefficients β_i of the polynomial $\beta(y) = \prod_{t\in E^*}(e_t + yf_t) = \sum_{i=1}^{q} \beta_i y^i$. Recall that $e_i \neq 0$ for all i with $s_i \neq s^*$ (Game 6), which implies $\beta_0 = \prod_i e_i \neq 0$. Therefore the adversary can compute $\hat{g}^{1/y}$ as

$$\left(\frac{z}{\prod_{i=1}^{q} \left(\hat{g}^{y^{i-1}}\right)^{\beta_i}}\right)^{1/\beta_0} = \left(\frac{\hat{g}^{\frac{\sum_i \beta_i y^i}{y}}}{\hat{g}^{\sum_i \beta_i y^{i-1}}}\right)^{1/\beta_0} = \left(\frac{\hat{g}^{\beta_0/y} \hat{g}^{\sum_i \beta_i y^{i-1}}}{\hat{g}^{\sum_i \beta_i y^{i-1}}}\right)^{1/\beta_0} = \left(\hat{g}^{\beta_0/y}\right)^{1/\beta_0} = \hat{g}^{1/y}.$$

This completes the proof for Type I forgers. □

C.2.2 Type II forgers

Lemma C.4. *Let $\mathcal{F}$ be a forger of type II that (t, q, ϵ)-breaks the existential unforgeability of $\mathsf{Sig}_{q\text{-}\mathsf{DH}}[\mathsf{H}, \mathsf{D}]$. Then there exists an adversary $\mathcal{A}$ that (t', ϵ')-breaks the q-DH assumption and an adversary that (t'', ϵ'')-breaks the discrete logarithm assumption with $t' \approx t'' \approx t$ and*

$$\epsilon' + \delta\epsilon'' \geq \delta(\epsilon - \gamma)$$

Proof. Again we let X_i denote the probability that $\mathcal{F}$ is successful in Game i.

Game 0. We define Game 0 as the existential unforgeability experiment with forger $\mathcal{F}$. By definition we have

$$\Pr[X_0] = \epsilon.$$

Game 1. In this game the simulator chooses the randomness $s_1, \ldots, s_q \xleftarrow{\$} \{0,1\}^\lambda$ used to answer the chosen message queries of the forger at the beginning of the game. This does not affect the success probability of $\mathcal{F}$, thus

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

Game 2. Now we run the trapdoor key generation algorithm $\mathsf{PHF.TrapGen}$ to generate the hash function H. Again let $E = \bigcup_{i=1}^{q} \{s_i\}$. The simulator picks $\theta \xleftarrow{\$} \mathbb{Z}_p$ and sets

$$g = \hat{g}^{\prod_{t \in E} \mathsf{d}(t)} \text{ and } h = g^\theta.$$

Then it runs $(\kappa, \tau) \xleftarrow{\$} \mathsf{PHF.TrapGen}(1^k, g, h)$ to generate hash key κ together with trapdoor τ. By the $(m, 1, \gamma, \delta)$-programmability of H we have

$$\Pr[X_2] \geq \Pr[X_1] - \gamma.$$

Game 3. We change the way chosen-message queries are answered by the simulator. By the setup of g and h introduced in Game 2, we have

$$\mathsf{H}(M_i) = g^{a_i} h^{b_i} = \hat{g}^{a_i \prod_{t \in E} \mathsf{d}(t)} \hat{g}^{b_i \theta \prod_{t \in E} \mathsf{d}(t)}.$$

Let E_i denote the set $E_i = E \backslash \{s_i\}$, and let

$$g_i := \hat{g}^{a_i \prod_{t \in E_i} \mathsf{d}(t)} \text{ and } h_i := g_i^\theta.$$

Just like in Game 5 of the proof of Lemma C.3, the simulator can compute a signature on M_i by running $(a_i, b_i) \leftarrow \mathsf{PHF.TrapEval}(\tau, M_i)$ first to determine (a_i, b_i), and then computing σ_i as

$$\sigma_i = g_i^{a_i} h_i^{b_i} = (g^{a_i} h^{b_i})^{1/\mathsf{d}(s_i)} = \mathsf{H}(M)^{1/\mathsf{d}(s_i)}.$$

Note that Game 3 is perfectly indistinguishable from Game 2 from the adversary's perspective. Thus we have

$$\Pr[X_3] = \Pr[X_2].$$

Game 4. Now the simulator aborts, if $a^* + \theta b^* \equiv 0 \bmod p$. We denote this event with $\mathsf{abort}_{\mathsf{log}}$. Note that explicit knowledge of θ is not used anywhere in the game. Thus we can construct an

adversary against the discrete logarithm problem that takes $(\hat{g}, \hat{g}^\theta)$ as input, and uses these values to simulate Game 4 perfectly. If $a^* + \theta b^* \equiv 0 \bmod p$, then the adversary can compute $\theta = \log_{\hat{g}} \hat{g}^\theta$ from (a^*, b^*). Therefore we get

$$\Pr[X_4] \geq \Pr[X_3] - \epsilon''$$

for a suitable (t'', ϵ'')-attacker against the discrete logarithm problem with $t'' \approx t$.

Game 5. Now the simulator computes $(e_i, f_i) \leftarrow \mathsf{PHF.TrapEval}'(\tau', s_i)$ for each s_i at the beginning of the game. When $\mathcal{F}$ outputs a forged signature (M^*, s^*, σ^*), the simulator computes $(e^*, f^*) \leftarrow \mathsf{PHF.TrapEval}'(\tau', s^*)$. We abort the game, if $e_i = 0$ for some $i \in [q]$ or if $e^* \neq 0$, and denote this event with $\mathsf{abort}_{\mathsf{PHF}}$. By the $(1, \mathsf{poly}, \gamma', \delta')$-programmability of D we have

$$\Pr[X_5] = \Pr[X_4 \wedge \neg \mathsf{abort}_{\mathsf{PHF}}] \geq \delta' \cdot \Pr[X_4].$$

The q-DH Adversary. We can now replace the simulator with adversary $\mathcal{A}$. The adversary receives a q-DH challenge $(\hat{g}, \hat{g}^y, \ldots, \hat{g}^{y^q})$ as input, and proceeds like the simulator from Game 4, but without knowing y explicitly. The adversary can compute the public key and answer all signature queries using $(\hat{g}, \hat{g}^y, \ldots, \hat{g}^{y^q})$ using the same techniques as the adversary in the proof of Lemma C.3. Thus we only need to show how the adversary can extract the solution to the q-DH challenge from a forged signature (M^*, σ^*, s^*) with $s^* \notin \bigcup_{i=1}^q \{s_i\}$.

Extracting the Solution to the q-DH Challenge. Given σ^*, the adversary computes $z = (\sigma^*)^{f_{s^*}/(a^* + \theta b^*)}$. Note that $a^* + \theta b^* \not\equiv 0 \bmod p$ by Game 4, that is $(a^* + \theta b^*)$ is invertible mod p. Observe that

$$\begin{aligned} z = (\sigma^*)^{f_{s^*}/(a^*+\theta b^*)} &= \left(g^{a^*} h^{b^*}\right)^{\frac{f_{s^*}/(a^*+\theta b^*)}{e^*+yf^*}} = \left(g^{a^*+\theta b^*}\right)^{\frac{f_{s^*}/(a^*+\theta b^*)}{e^*+yf^*}} \overset{(*)}{=} \left(g^{a^*+\theta b^*}\right)^{\frac{f_{s^*}/(a^*+\theta b^*)}{yf^*}} \\ &= \left(\hat{g}^{(a^*+\theta b^*)\prod_{s_i \in E}(e_i+yf_i)}\right)^{\frac{f_{s^*}/(a^*+\theta b^*)}{yf^*}} = \left(\hat{g}^{\prod_{s_i \in E}(e_i+yf_i)}\right)^{\frac{1}{y}} = \hat{g}^{\frac{\prod_{s_i \in E}(e_i+yf_i)}{y}} \end{aligned}$$

where again $(*)$ uses that $e^* = 0$ by Game 5. Since $\prod_i e_{s_i} \not\equiv 0 \bmod p$ (also Game 5), the adversary can extract $\hat{g}^{1/y}$ from z just like in the proof of Lemma C.3.

□