Digital Signatures from Strong RSA without Prime Generation

David Cash[1], Rafael Dowsley[2], and Eike Kiltz[3]

[1] Department of Computer Science
Rutgers University
[2] Institute of Theoretical Informatics
Karlsruhe Institute of Technology
[3] Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum

Abstract. We construct a signature scheme that is proved secure, without random oracles, under the strong RSA assumption. Unlike other efficient strong-RSA based schemes, the new scheme does not generate large prime numbers during signing. The public key size and signature size are competitive with other strong RSA schemes, but verification is less efficient. The new scheme adapts the prefix signing technique of Hohenberger and Waters (CRYPTO 2009) to work without generating primes.

Keywords. Digital Signatures, Strong RSA.

1 Introduction

Digital signatures are amongst the most widely deployed cryptographic primitives, with several efficient, standardized schemes that are implemented and used in common functionalities like HTTPS. Theoretical constructions study the extent to which digital signatures can be proved secure under mild hardness assumptions, like the existence of one-way functions, giving us good evidence for the possibility of constructing secure schemes.

As is often the case with provable security, however, the proved-secure schemes with the best security guarantees are not nearly as efficient as the (unbroken) schemes that are used in practice, where applications require fast signing and verification along with short public-keys and short signatures. The best provable security evidence (when it is available) for practical schemes comes from security proofs that use the random oracle model [3], where one models a hash function as a random function. Of course, in practice we use a non-random function like SHA-256, a reality that leads some theoretical limitations of these results [7, 12]. From an assurance standpoint it is desirable to have security proofs without random oracles in order to lessen the possibility that a proved-secure scheme will be broken when implemented. From a theoretical standpoint it is interesting to know what is achievable without the random oracle.

Our contribution. In this paper we continue a line of work on designing efficient signature schemes that are proved secure, without a random oracle, under the strong RSA assumption [2]. Unlike other such schemes, ours does not need to generate large prime numbers during signing and avoids this by embedding the strong RSA problem into the scheme in a different way.

Recall that the strong RSA problem requires an adversary, on input (N, y) where $N = pq$ for large random primes p, q, and $y \in \mathbb{Z}_N^*$ is random, to compute (e, x) satisfying $x^e = y \mod N$ and $e > 1$. The structure of the problem suggests a natural approach for embedding it into digital signatures, where the public key is N and a signature will consist of (e, x), where e is computed at signing time and x is an e-th root of a value y that depends on the public key and the message. In order to apply known techniques that prevent an adversary from assembling several signatures into a new signature, the e that is generated is typically required to be a large prime or a product of large primes.

Our construction instead works with e set to be a product of several composite numbers so that the likelihood of one of them being divisible by a large prime factor is large. In order to avoid making e extremely large, we adapt techniques from prior work, including the "prefix signing" of Hohenberger and Waters [19] and analysis techniques for dealing with composite numbers in RSA signatures due to Gennaro, Halevi, and Rabin [14]. A sketch of our approach and the techniques we use is given below in the next sub-section.

It is desirable to avoid prime generation in signing because it is typically an expensive operation and is a step which is not intrinsic for the signing algorithm. While our scheme does this with a relatively simple signing procedure and with public key and signature sizes competitive with prior schemes, it has a much slower verification algorithm. Like all other signature schemes that do not use random oracles in their security proofs, our construction is not competitive with practical schemes and we do not recommend it for consideration in applications. Instead, we aim to have a conceptual contribution towards the goal of practical schemes from conservative hardness assumptions without random oracles.

In order to more precisely describe our contribution, we first need to recall some prior work.

Standard Model RSA Signatures. We focus on signatures whose security is based on the (strong) RSA problem without a random oracle. While there exists a number of different schemes [11, 14, 22, 23, 13, 6, 21, 18, 19, 8, 17, 5], they all have in common that the signing algorithm has to generate one or more primes (of some large size). The prime generation can either be deterministic (via some hash function h from messages to primes) or according to the uniform distribution. In both cases the prime generation remains an expensive step whose elimination is desirable.

Concretely, the strong-RSA based signature schemes from [11, 13, 18, 22, 23, 6] compute a signature on message m as $\sigma(m) = (\mathsf{H}(m)^{1/e} \bmod N, e)$, where e is a random prime and H is some (algebraic) hash function that depends on the specific scheme; the (weakly secure) scheme by Gennaro et al. [14] defines

$\sigma(m) = g^{1/\mathsf{h}(m)} \bmod N$ where h is a hash function that hashes into primes.[4] The signature scheme by Hohenberger and Waters [19] as well as the one by Hofheinz et al. [17] are based on the (weaker) RSA assumption and define $\sigma(m) = g^{1/\prod_{i=1}^{n} \mathsf{h}_i(m)} \bmod N$, where h_i are independent hash functions that hash into primes. Designing a standard-model signature scheme whose signing algorithm does not rely on the generation of prime numbers is an open problem explicitly mentioned in [17].

1.1 Our contributions

Our new scheme is relatively simple, so we describe it right away. Its public key consists of $N = pq$ where p and q are large safe primes[5], a number $h \in \mathbb{Z}_N^*$, and a key of a pseudorandom function $F_K(\cdot)$. For the time being, we assume that $F_K(\cdot)$ takes variable-length inputs, and always outputs odd numbers of some given length. Signatures on a message $m \in \{0,1\}^\ell$ are defined via

$$\mathsf{Sign}(m) = h^{1/e} \bmod N, \qquad \text{where } e = \prod_{i=1}^{\ell} F_K(m[1...i]) \cdot \prod_{i=1}^{d} F_K(m \,\|\, i). \tag{1}$$

Here $m[1..i]$ is the i-bit prefix of m, d is a parameter that factors into the concrete security, and $m \,\|\, i$ means m with an encoding of the number i appended. (Signatures can be computed using the secret key, the factorization of $N = pq$.) We stress that we are using the outputs of the $F_K(\cdot)$, which are random odd numbers that are likely to be composite. This is the main difficulty in proving security. Theorem 5 shows that this scheme achieves a type of weak security under the strong RSA assumption in the standard model. Full security (unforgeability against chosen-message attack) can be achieved by adding a chameleon hash function - see [20] or the full version of [19].

Intuition. Let us sketch how our scheme adapts and differs from prior proof techniques. The notion of weak security for signature schemes means that the adversary gets only one parallel signing query on chosen messages before seeing the public key. Then it is given the public key, along with the requested signatures, and must generate a signature on a new message $\widehat{m}$. See the next section for a formal definition, via the game wCMA.

We start with a very high level explanation of why all of the prefixes of m are processed using F_K and multiplied together. Consider a rooted full binary tree of depth ℓ, with all nodes assigned a label from $\{0,1\}^{\leq \ell}$ according to the left/right steps to that node from the root. The prefixes of a message are exactly the labels on the nodes encountered on the root-to-message path.

Then we can see the requested messages from the adversary's parallel signing query as leaves in the tree, and the union of all root-to-message paths is a subtree.

[4] For GHR signatures, the weaker condition *collision intractability* is sufficient for h. However, the only known way to instantiate h in the standard model is to hash into primes [21, 10, 14].

[5] A safe prime is an odd prime number p such that $p' = (p-1)/2$ is also prime.

Now for any message $\widehat{m}$ not in this subtree, the root-to-$\widehat{m}$ path must have some first node that is not in the subtree. In fact, we can show that *all* paths to messages not in the subtree must pass through one of a small number of "exit nodes".

The Hohenberger-Waters signature scheme was designed to take advantage of this structure by guessing which exit node would be used by the message on which the adversary forges a message. If the guess is correct, then, using hash functions that output only primes, they can arrange to program in an instance of the (non-strong) RSA problem. Since the number of exit nodes is polynomial, this guess is correct with non-negligible probability, resulting in an acceptable loss in the success probability during the reduction.

We also exploit this structure, but instead we do not guess which exit node will be used. Instead, we arrange so that we can solve the strong RSA problem no matter which exit node is used by the adversary during forging. Examining the proof reveals that this amounts to hoping that several (composite) numbers output by $F_K(\cdot)$ on different inputs will *all* have large prime factors (i.e, they are non-smooth). A naive analysis of this technique in which one hopes that with overwhelming probability all exit nodes are non-smooth gives very bad parameters. So instead of hoping that every exit node helps us solve the problem with overwhelming probability, we show that it is enough for each exit node to help us (i.e., have a large prime factor) with only constant probability. We can show that this is in fact enough, because when a node does not help, we can discard it and look at both its children, recursively repeating this process. Analyzing this behavior is the main difficulty in our proof.

While the idea of using the fact that a number x is not α-smooth is not new in cryptography (see below for related work), it is clear that the straightforward approach of requiring x to be α-smooth with negligible probability would normally result in very bad protocol parameters since the gap between x and α would have to be too big. Our scheme derives its advantage because the reduction can tolerate the random numbers having large prime factors with only constant probability via the recursive tree searching, allowing us to save in parameters (i.e. use smaller numbers), at the expense of the d extra evaluations and multiplications. Consider the set of message queried by the adversary and the subtree formed by all their root-to-message paths. The central idea of the security proof is that for any message $\widehat{m}$ not in this subtree (i.e., all the messages for which a forgery would be acceptable), there should be at least one random number in $\widehat{m}$'s root-to-message path which is not in the subtree of queried messages and has a large prime factor. If all the numbers associated to the exit nodes were such that they had a large prime factor, the proof would be done. But we only require the random numbers to have a large prime factor with constant probability, thus for all exit nodes which do not have a large prime factor, we need to analyze both of its children, and follow the same procedure recursively for the children. Our analysis of this recursive tree searching shows that with overwhelming probability all message $\widehat{m}$ not in the subtree of queried messages will have at least one random number in $\widehat{m}$'s root-to-message path which is not in the subtree and has a large prime factor. The d extra evaluations are due to the exit nodes close to the bottom of the tree. After establishing this fact, the analysis proceeds to show that the existence of this large prime factor can be used to extract solutions to the strong RSA problem from a forged signature. Note that while the Hohenberger-Waters signature scheme is based on the (weaker) RSA assumption, we need to base our scheme on the strong RSA assumption because we cannot simply guess the exit node and program the RSA instance there.

1.2 Efficiency

The public key contains the modulus N, $h \in \mathbb{Z}_N^*$, and a key of a PRF. Recall the definition of a signature from Equation (1). The cost of computing a signature $\sigma(m)$ is dominated by one full exponentiation modulo N to compute $h^{1/e}$. While signing is quite efficient, the cost of signature verification is substantially higher. If the PRF outputs numbers between 1 and $2^n - 1$, the verification has to perform one modular exponentiation of an exponent e which is of an $n(\ell + d)$ bit number. (Note that verification can't reduce e modulo $\varphi(N)$ since that value is only contained in the secret key.) Our security analysis of Theorem 5 and Section 4 give an upper bound in the numbers ℓ, d, and n such that our system is secure. Concretely, for 80 bits security (and assuming that the Dickmann function, $\rho(u)$, is a good approximation for the probability of a random number between 1 and x being $x^{1/u}$-smooth) we can have $\ell = 160$, $n = 200$ and $d = 80$, in which case verification has to perform one exponentiation modulo N with an exponent of size $200(160{+}80) = 48000$ bits. This analysis is for the weakly secure scheme. The fully secure scheme adds one Chameleon Hash and therefore one exponentiation during signing and verification.

Overall, our new signature scheme offers fast and simple signing combined with a small public key, but has relatively slow verification. More importantly, it is the first scheme that does not need to generate primes or run primality tests during the signing process. We believe that this departure from prime generation dependency is a possible direction for future improvements in the quest for more practical signature schemes which are provable secure in the standard model and can also be useful in other contexts.

1.3 Related Work

The key idea that large random numbers are somewhat likely to have large prime factors and that large random numbers can replace large prime numbers are not new in cryptography. In 1999, Gennaro, Halevi, and Rabin [14], in the process of proving the security of their signature scheme, analyzed the probability that a specific random number is smooth and then showed that if such number is non-smooth then the probability that it divides the product (of a polynomial number) of random numbers is negligible, thus establishing an essential step of the security proof of their signature scheme. We adapt their analysis technique in order to extract solutions to the strong RSA problem in our reduction.

Subsequently, in the context of elliptic-curve signatures, Coron, Handschuh and Naccache [9] avoided point counting (i.e., the need of the participants to know the number of points on the curve and a big factor of it) by first using curves over larger underlying fields. As in our case, a naive analysis of the smoothness property, i.e., requiring the number of curve points to be smooth with negligible probability, would result in very bad parameters for the protocol. Hence the authors only increased the size of the underlying field such that the probability that the curve is smooth is low. Next, they iterated the protocol over many independent random curves in order to guarantee that, with overwhelming probability, at least one curve is non-smooth. Hence their modified signature scheme for avoiding point counting consists of many parallel instances of the original signature scheme and therefore had a considerable slowdown around a factor of 500 [9]. If we used the same approach and signed the messages multiple times with Hohenberger-Waters-style signatures in the hope that for all exit nodes, in at least one instance its associated number would be non-smooth, then this would result in a considerable protocol slowdown.

We stress that even though our signatures are syntactically related to the schemes by Hohenberger and Waters [19] and Gennaro et al. [14], the main difference is that in our scheme the hash functions h_i (instantiated via a PRF F_K) do not output primes.

2 Preliminaries

Notation. When convenient, we identify a vector with the set of its entries, i.e. a vector $\mathbf{m}$ with Q entries will be identified with $\{\mathbf{m}[1], \ldots, \mathbf{m}[Q]\}$. We denote by $x \leftarrow_{\$} X$ the action of selecting a random element of a set X and calling it x. Most of our security definitions and proofs will use code-based games in the style of Bellare and Rogaway [4]. These games are algorithms that start by running an Initialize procedure, if present, and giving the output to the adversary. Then the adversary queries the oracles provided by the games, and finally halts, with its output becoming the input to Finalize. The game output is the output of Finalize. We denote by G^A the event that G outputs true when running with A. In the code, all boolean flags are implicitly initialized to false and all tables are initially populated with $\perp$.

Signature schemes. A signature scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ consists of three algorithms that satisfy the following syntax requirements. Algorithm KeyGen takes the security parameter λ as input and outputs a public/secret key pair, denoted (pk, sk). Sign takes as input a secret key sk, a message $m \in \{0,1\}^\ell$, and outputs a signature σ or $\perp$ (our security definitions will imply that it should only output $\perp$ with very small probability). Verify takes as input a public key pk, a message m, and a signature σ and outputs accept or reject. We require that, for all (pk, sk) output by $\mathsf{KeyGen}(1^\lambda)$, all messages $m \in \{0,1\}^\ell$, and all $\sigma \neq \perp$ output by $\mathsf{Sign}(sk, m)$, $\mathsf{Verify}(pk, m, \sigma)$ accepts.

```
proc Initialize
$M \leftarrow \emptyset$
$(pk, sk) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$
Return $pk$

proc Sign$(m)$
$M \leftarrow M \cup \{m\}$
$\sigma \leftarrow\$ \mathsf{Sign}(sk, m)$
If $\sigma = \perp$ then Win $\leftarrow$ true
Return $\sigma$

proc Finalize$(\widehat{m}, \widehat{\sigma})$
If Win then Return true
If $\widehat{m} \in M$ then Return false
Return $\mathsf{Verify}(pk, \widehat{m}, \widehat{\sigma}) = 1$
```

```
proc Initialize
$(pk, sk) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$
Return $1^\lambda$

proc Sign$(\mathbf{m})$
For $i = 1, \ldots, |\mathbf{m}|$ do
  $\boldsymbol{\sigma}[i] \leftarrow\$ \mathsf{Sign}(sk, \mathbf{m}[i])$
  If $\boldsymbol{\sigma}[i] = \perp$ then Win $\leftarrow$ true
Return $(pk, \boldsymbol{\sigma})$

proc Finalize$(\widehat{m}, \widehat{\sigma})$
If Win then Return true
If $\widehat{m} \in \mathbf{m}$ then Return false
Return $\mathsf{Verify}(pk, \widehat{m}, \widehat{\sigma}) = 1$
```

```
proc Initialize
$K \leftarrow\$ \{0,1\}^\lambda$
$b \leftarrow\$ \{0,1\}$
Return $1^\lambda$

proc Fn$(x)$
If $b = 1$
  $T[x] \leftarrow F_K(x)$
If $(b = 0) \wedge (T[x] = \perp)$
  $T[x] \leftarrow\$ \{0,1\}^n$
Return $T[x]$

proc Finalize$(\widehat{b})$
Return $(\widehat{b} = b)$
```

Fig. 1. Games CMA (left), wCMA (middle), and PRF (right). In wCMA the adversary is only allowed one query to Sign.

Security notions for signature schemes. We will target existential unforgeability under chosen message attacks. We define security using the game CMA in Fig. 1. For a signature scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ and an adversary A, define the *CMA advantage of* A to be $\mathbf{Adv}^{\mathrm{cma}}_{\Pi,A}(\lambda) = \Pr[\mathrm{CMA}^A]$. Note that, in a slight departure from the standard definition, the adversary wins when the signature scheme outputs $\perp$.

While existential unforgeability under chosen message attacks is our ultimate target, we will mostly deal with an intermediate notion called existential unforgeability under *weak* chosen message attacks, which is defined via the game wCMA in Fig. 1. In this game, the adversary is only allowed one Sign query, which is issued before the adversary sees the public key. We define the *wCMA advantage of* A to be $\mathbf{Adv}^{\mathrm{wcma}}_{\Pi,A}(\lambda) = \Pr[\mathrm{wCMA}^A]$.

Chameleon hash functions. A hash function $\mathsf{HF} = (\mathsf{K}, \mathsf{HE})$ is a tuple of polynomial-time algorithms. Algorithm K takes the security parameter λ as input and outputs a key K. The hash evaluation algorithm HE takes a key K and some input x and computes $y \leftarrow \mathsf{HE}(K, x)$. A collision-resistance adversary H, given $K \leftarrow \mathsf{K}(1^\lambda)$ as input, outputs (x, x'). The adversary advantage, $\mathbf{Adv}^{\mathrm{cr}}_{\mathsf{HF},H}(\lambda)$, is given by the probability that the outputted (x, x') satisfy $(x \neq x') \wedge (\mathsf{HE}(K, x) = \mathsf{HE}(K, x'))$. A chameleon hash [20] is a collision-resistant hash function with additional properties.

- The hash evaluation algorithm HE takes a pair consisting of a message m and randomness r as input.
- The algorithm K, in addition to K, also generates a secret trapdoor information, HT. There should be an efficient algorithm that when given messages m_1, m_2, randomness r_1 and HT as input, finds r_2 such that $\mathsf{HE}(K, (m_1, r_1)) = \mathsf{HE}(K, (m_2, r_2))$.

– All the messages m should induce the same probability distribution on the hash output for r chosen uniformly at random.

Fact 1: There is a generic way to transform an wCMA secure signature scheme into an CMA secure signature scheme using a chameleon hash [20, 19].

Fact 2: There is a construction of a chameleon hash based on the RSA assumption [20, 19].

PSEUDORANDOM FUNCTIONS. We define pseudorandom function security using the game PRF in Fig. 1. For a function family F with outputs of length n and an adversary C, we define the *PRF advantage of* C to be $\mathbf{Adv}^{\mathrm{prf}}_{F,C}(\lambda) = 2\Pr[\mathrm{PRF}^C] - 1$.

STRONG RSA ASSUMPTION. An RSA parameter generator is an algorithm that, outputs two random, equal length safe primes (p, q) (a safe prime is a prime p such that $p' = (p-1)/2$ is also prime). Let $\mathsf{RSAGen}(1^\lambda)$ be an RSA parameter generator. We define the advantage of an adversary C against the strong RSA assumption with RSAGen [2, **?**], $\mathbf{Adv}^{\mathrm{srsa}}_{\mathsf{RSAGen},C}(\lambda)$, as the probability that C given (N, y), where $(p, q) \leftarrow \mathsf{RSAGen}(1^\lambda)$ and $N = pq$ and $y \leftarrow_{\$} \mathbb{Z}_N^*$ as input, returns (e, x) such that $e > 1$ and $x^e = y \mod N$.

FACTS FROM NUMBER THEORY. Let α be a positive integer. An integer x is called α-smooth if all prime factors of x are less than or equal to α. We will denote by $\varepsilon(\alpha, n)$ the probability that a random number between 0 and $2^n - 1$ is α-smooth. Define the function

$$L_x[a] = \exp\left((a + o(1))\sqrt{\log x \log\log x}\right).$$

The probability that a random integer between one and x is $L_x[a]$-smooth is $L_x\left[\frac{-1}{2a}\right]$ (see [10]). We will also use the following lemma from [16].

Lemma 1 *Given $x, y \in \mathbb{Z}_N^*$ and $a, b \in \mathbb{Z}$ such that $x^a = y^b$, one can efficiently compute $z \in \mathbb{Z}_N^*$ such that $z = y^{\frac{\gcd(a,b)}{a}}$.*

STRINGS. We will write $\{0,1\}^{\leq \ell}$ for $\cup_{i=0}^{\ell}\{0,1\}^i$. For a string $x \in \{0,1\}^{\leq \ell}$ we write $\mathsf{Pref}(x)$ for the set of all prefixes of x, including the empty string and x. We extend this notation to prefixes of sets in the obvious way.

The following definition formalizes the notion of "exit nodes" from the introduction.

Definition 2 Let $M \subseteq \{0,1\}^\ell$ be non-empty. A *minimal non-prefix of* M is a string $x \in \{0,1\}^{\leq \ell}$ such that $x \notin \mathsf{Pref}(M)$ but $x' \in \mathsf{Pref}(M)$, where x' is x with the last bit deleted. We denote the set of minimal non-prefixes of M by $\mathsf{MNP}(M)$.

Note that the empty string is never in $\mathsf{MNP}(M)$ because it is always in $\mathsf{Pref}(M)$. The following lemma is implicit in [19].

Lemma 3 *Let $M \subseteq \{0,1\}^\ell$ be non-empty. Then we have:*

– *For all $i = 1, \dots, \ell$, $|\mathsf{MNP}(M) \cap \{0,1\}^i| \le |M|$, so $|\mathsf{MNP}(M)| \le \ell|M|$. Moreover, $\mathsf{MNP}(M)$ can be computed in time linear in $\ell|M|$.*
– *For any $y \notin M$, $\mathsf{Pref}(y) \cap \mathsf{MNP}(M)$ consists of exactly one string.*

Chernoff bound. We will use the following standard multiplicative Chernoff bound.

Lemma 4 *Let $Y_1, \dots, Y_\ell$ be independent Bernoulli random variables such that $\Pr[Y_i = 1] = \varepsilon$ for all i, and let $Y = Y_1 + \cdots Y_\ell$. Then for all $\delta > 0$,*

$$\Pr[Y > (1+\delta)\varepsilon\ell] < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{\varepsilon\ell}.$$

3 Signature Scheme

The signature scheme works as follows. Let λ be the security parameter and let $n = n(\lambda)$, $\ell = \ell(\lambda)$ and $d = d(\lambda)$ be functions of the security parameter. The scheme signs messages from $\{0,1\}^\ell$, but this can be extended using a collision resistant hash function.

Fix an RSA parameter generator RSAGen and a function family F that maps $\{0,1\}^{\le \ell'}$ to odd numbers between 0 and $2^n - 1$ (i.e., bitstrings with the last bit set), where $\ell' = \ell + \lceil \log d \rceil$. We associate with each message $m \in \{0,1\}^\ell$ a set of strings $S(m) \subseteq \{0,1\}^{\le \ell'}$ given by

$$S(m) = \mathsf{Pref}(m) \cup \{m \,\|\, i : i \in [d]\}. \tag{2}$$

That is, $S(m)$ consists of all of the prefixes of m, including the empty string and m itself, along with d strings that are formed by appending to m (an encoding of) an integer between 1 and d.

$\mathsf{KeyGen}(1^n)$**:** Run $(p,q) \leftarrow_\$ \mathsf{RSAGen}(1^\lambda)$, then set $N = pq$. Select $h \leftarrow_\$ \mathbb{Z}_N^*$ and a key $K \leftarrow_\$ \{0,1\}^\lambda$ for the function family F. The public key is $pk = (N, h, K)$ and the secret key is $sk = (pk, p, q)$.

$\mathsf{Sign}(sk, m)$**:** Compute the set $S(m)$. For each $s \in S(m)$, let $e_s \leftarrow F_K(s)$ (recall that F outputs odd numbers), and set $e \leftarrow \prod_{s \in S(m)} e_s$. If e is not coprime with $\phi(N)$, then output $\perp$. Otherwise, solve the equation

$$\sigma = h^{1/e} \mod N,$$

for σ using the factorization of N. Then output the signature σ. Note that the probability of outputting $\perp$ is negligible in the security parameter.

$\mathsf{Verify}(pk, m, \sigma)$**:** Compute the set $S(m)$, and for each $s \in S(m)$ let $e_s \leftarrow F_K(s)$. Then let $e \leftarrow \prod_{s \in S(m)} e_s$, and finally accept if

$$\sigma^e = h \mod N$$

and otherwise reject.

```
proc Sign(m)   // G_0, \boxed{G_1}
$(p,q) \leftarrow_{\$} \mathsf{RSAGen}(1^\lambda)$ ; $N \leftarrow pq$
$p' \leftarrow (p-1)/2$ ; $q' \leftarrow (q-1)/2$
$h \leftarrow_{\$} \mathbb{Z}_N^*$ ; $K \leftarrow_{\$} \{0,1\}^\lambda$
$pk \leftarrow (N,h,K)$
If $\mathsf{BadSet}^{F_K(\cdot)}(\mathbf{m})$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$ ; $\boxed{\text{Lose} \leftarrow \mathsf{true}}$
For each $j = 1,\ldots,Q$ do
    $e_j \leftarrow \prod_{s \in S(\mathbf{m}[j])} F_K(s)$
    If $\gcd(e_j, p'q') > 1$ then
        $\boldsymbol{\sigma}[j] \leftarrow \perp$
    Else $\boldsymbol{\sigma}[j] \leftarrow h^{1/e_j} \bmod N$
Return $(pk, \boldsymbol{\sigma})$

proc Finalize($\widehat{m}, \widehat{\sigma}$)   // G_0, G_1, G_2
$\widehat{e} \leftarrow \prod_{s \in S(\widehat{m})} F_K(s)$
If $\gcd(\widehat{e}, p'q') > 1$ then return false
If Lose then return false
Return $\perp \in \boldsymbol{\sigma} \vee ((\widehat{\sigma}^{\widehat{e}} = h) \wedge (\widehat{m} \notin \mathbf{m}))$
```

```
proc Sign(m)   // G_2
$(p,q) \leftarrow_{\$} \mathsf{RSAGen}(1^\lambda)$ ; $N \leftarrow pq$
$p' \leftarrow (p-1)/2$ ; $q' \leftarrow (q-1)/2$
$y \leftarrow_{\$} \mathbb{Z}_N^*$ ; $K \leftarrow_{\$} \{0,1\}^\lambda$
If $\mathsf{BadSet}^{F_K(\cdot)}(\mathbf{m})$ then Lose $\leftarrow$ true
For each $j = 1,\ldots,Q$ do
    $e_j \leftarrow \prod_{s \in S(\mathbf{m}[j])} F_K(s)$
$e^* \leftarrow \prod_{j=1}^{Q} e_j$
$h \leftarrow y^{e^*} \bmod N$ ; $pk \leftarrow (N,h,K)$
For each $j = 1,\ldots,Q$ do
    If $\gcd(e_j, p'q') > 1$ then
        $\boldsymbol{\sigma}[j] \leftarrow \perp$
    Else
        $e'_j \leftarrow e^*/e_j$ ; $\boldsymbol{\sigma}[j] \leftarrow y^{e'_j} \bmod N$
Return $(pk, \boldsymbol{\sigma})$

proc Sign(m)   // G_3
Choose a random function $\pi$
If $\mathsf{BadSet}^{\pi}(\mathbf{m})$ then
    bad $\leftarrow$ true

proc Finalize($\widehat{m}, \widehat{\sigma}$)   // G_3
Return bad
```

Fig. 2. Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2, \mathrm{G}_3$ for the proof of Theorem 5. G_1 includes the boxed code and G_0 does not. BadSet is described below.

3.1 Security Proof

Theorem 5 *Let F be a function family with outputs of length n,* RSAGen *be a RSA parameter generator, and Π be the signature scheme associated to F and* RSAGen *via the construction above. Let α be such that $\varepsilon(\alpha, n) \leq 1/4$. Then for all adversaries A that request Q signatures, there exist efficient adversaries B, C such that*

$$\mathbf{Adv}^{\mathrm{wcma}}_{\Pi,A}(\lambda) \leq \mathbf{Adv}^{\mathrm{prf}}_{F,C}(\lambda) + \mathbf{Adv}^{\mathrm{srsa}}_{\mathsf{RSAGen},B}(\lambda) + Q\left(2^{-2d+1} + 2\ell^3(\ell+d)Q/\alpha + \ell^2(e/4)^\ell\right). \quad (3)$$

where e is the base of the natural logarithm.

Recall that $\varepsilon(\alpha, n)$ is the probability that a random number between 0 and $2^n - 1$ is α-smooth. In Section 4, we will provide example instantitations of the parameters n, ℓ, and d.

Proof. We use the games in Fig. 2. These games use as subroutine an oracle algorithm BadSet, which we describe now. Algorithm BadSet takes as input a vector of messages $\mathbf{m} \subseteq \{0,1\}^\ell$, expects access to an oracle $\mathcal{O}$ mapping $\{0,1\}^*$ to $\mathbb{Z}_N$, and outputs true or false. It works as follows.

$\mathsf{BadSet}^{\mathcal{O}}(\mathbf{m})$ first computes $e^* \leftarrow \prod_{m\in\mathbf{m}} \prod_{t\in S(m)} \mathcal{O}(t)$, and then computes the set $\mathsf{MNP}(\mathbf{m})$. Then for each $x \in \mathsf{MNP}(\mathbf{m})$, it runs the recursive subroutine $\mathsf{Check}^{\mathcal{O}}(x)$. If any of these runs returns true, then BadSet returns true, and otherwise it returns false.

```
Check^O(x)
    If $\mathcal{O}(x) \nmid e^*$ then
        Return false
    If $|x| = \ell$ then
        If $\mathcal{O}(x \,\|\, i) \nmid e^*$ for some $i = 1, \dots, d$ then
            Return false
        Else
            Return true
    If $|x| < \ell$ then
        Return $\mathsf{Check}^{\mathcal{O}}(x \,\|\, 0) \vee \mathsf{Check}^{\mathcal{O}}(x \,\|\, 1)$
```

We add the rule that, if any call to $\mathsf{Check}^{\mathcal{O}}(x)$ results in more than $2\ell^2$ recursive calls, then $\mathsf{BadSet}^{\mathcal{O}}(\mathbf{m})$ halts the computation and returns true.

We now turn to relating the games. Game G_0 is just an implementation of the game wCMA with an extra bad flag that is set when BadSet returns true. The purpose of this flag will become clear later - it is meant to catch cases where, after issuing its signing query, the adversary could find a message for which a forgery is easily created by assembling the given signatures.

Game G_1 is the same as G_0, except that when the bad flag is set it returns false to the adversary. We have

$$\mathbf{Adv}^{\mathrm{wcma}}_{\Pi,A}(\lambda) \;=\; \Pr[\mathrm{G}_0^A] \tag{4}$$

$$\Pr[\mathrm{G}_0^A] \;\le\; \Pr[\mathrm{G}_1^A] + \Pr[\mathrm{B{\scriptsize AD}}(\mathrm{G}_0^A)]. \tag{5}$$

The inequality follows by the fundamental lemma of game playing since G_0 and G_1 are identical-until-bad.

We will bound the summands in (5) individually. We first deal with $\Pr[\mathrm{G}_1^A]$. To this end, we use G_2, which employs the now-standard technique of computing the public key and signatures “backwards” without changing their distribution - instead, the changes are meant to help the reduction to the SRSA problem. G_2 computes h by choosing a random element of $\mathbb{Z}_N^*$, y, and raising it to the product of all of the exponents used during signing. Since the adversary wins if this product is not relatively prime to $p'q'$, the resulting h is a uniformly random element of $\mathbb{Z}_N^*$ (or both games output true). Subsequently, signatures can be computed using y and the exponents using the identity

$$h^{1/e_j} = y^{(\prod_{i=1}^{n} e_i)/e_j} = y^{\prod_{i\neq j} e_i}.$$

We have

$$\Pr[\mathrm{G}_1^A] = \Pr[\mathrm{G}_2^A].$$

We claim there exists an efficient adversary B such that

$$\Pr[\mathrm{G}_2^A] \;\le\; \mathbf{Adv}^{\mathrm{srsa}}_{\mathsf{RSAGen},B}(\lambda).$$

Naturally, B is designed to take advantage of the changes made in G_2. B takes as input (N, y) and attempts to compute some $x \in \mathbb{Z}_N^*$ and $e > 1$ such that $x^e = y \mod N$. It gives the adversary the security parameter, and then the adversary queries Sign with a vector of messages $\mathbf{m}$. B then computes

```
$K \leftarrow_{\$} \{0,1\}^\lambda$
If $\mathsf{BadSet}^{F_K(\cdot)}(\mathbf{m})$ then halt with output $\perp$
For each $j = 1, \ldots, Q$ do
    $e_j \leftarrow \prod_{s \in S(\mathbf{m}[j])} F_K(s)$
$e^* \leftarrow \prod_{j=1}^{Q} e_j$
$h \leftarrow y^{e^*} \mod N$ ; $pk \leftarrow (N, h, K)$
For each $j = 1, \ldots, Q$ do
    If $N > \gcd(2e_j + 1, N) > 1$ then $\boldsymbol{\sigma}[j] \leftarrow \perp$ ;  use $e_j$ to factor $N$
    Else $e'_j \leftarrow e^*/e_j$ ; $\boldsymbol{\sigma}[j] \leftarrow y^{e'_j} \mod N$
Return $(pk, \boldsymbol{\sigma})$
```

B runs A until it outputs $(\widehat{m}, \widehat{\sigma})$. We claim that whenever A would have won G_2, B will solve its SRSA instance. First, if it occurs that $\perp \in \boldsymbol{\sigma}$ in G_2, then B will have factored N in the computation above, thus B can use the factorization to solve the SRSA problem. Note that $\gcd(2e_j + 1, N) > 1$ if and only if $\gcd(e_j, p'q') > 1$.

If instead the output of A satisfies $\widehat{\sigma}^{\widehat{e}} = h \mod N$ and $\widehat{m} \notin \mathbf{m}$, then B uses the output of A to solve the SRSA instance. Now the fact that $\mathsf{BadSet}^{F_K(\cdot)}(\mathbf{m})$ must have returned false become relevant: We claim it implies that $\widehat{e} = \prod_{s \in S(\widehat{m})} F_K(s)$ does not divide e^*. To see this, let x be the unique prefix of $\widehat{m}$ in $\mathsf{MNP}(\mathbf{m})$. Then $\mathsf{Check}^{F_K}(x)$ must have returned false, meaning that, for one of the $s \in S(\widehat{m})$, $F_K(s)$ did not divide e^*. If one of these did not divide e^*, then certainly product did not, as desired.

Given that $\widehat{e}$ does not divide e^*, we can apply Lemma 1. More specifically, $\widehat{\sigma}, \widehat{e}, y, e^*$ satisfy $\widehat{\sigma}^{\widehat{e}} = y^{e^*} \mod N$, and we can compute $z \in \mathbb{Z}_N^*$ satisfying $z^{\widehat{e}/\gcd(\widehat{e}, e^*)} = y \mod N$. We have $\widehat{e}/\gcd(\widehat{e}, e^*) > 1$, so $(z, \widehat{e}/\gcd(\widehat{e}, e^*))$ is a valid solution to the SRSA instance. This establishes the claim.

Returning to our bound on the advantage of A and now considering Game G_3, substitutions give

$$\begin{aligned} \mathbf{Adv}^{\mathrm{wcma}}_{\Pi,A}(\lambda) &\le \mathbf{Adv}^{\mathrm{srsa}}_{\mathsf{RSAGen},B}(\lambda) + \Pr[\mathrm{Bad}(\mathrm{G}_0^A)] && (6)\\ &= \mathbf{Adv}^{\mathrm{srsa}}_{\mathsf{RSAGen},B}(\lambda) + \Pr[\mathrm{G}_3^A] + (\Pr[\mathrm{Bad}(\mathrm{G}_0^A)] - \Pr[\mathrm{G}_3^A]). && (7) \end{aligned}$$

We complete the proof by showing

$$\Pr[\mathrm{Bad}(\mathrm{G}_0^A)] - \Pr[\mathrm{G}_3^A] \;\le\; \mathbf{Adv}^{\mathrm{prf}}_{F,C}(\lambda) \qquad (8)$$

and

$$\Pr[\mathrm{G}_3^A] \;\leq\; Q\left(2^{-2d+1} + 2\ell^3(\ell+d)Q/\alpha + \ell^2(e/4)^\ell\right). \tag{9}$$

We first prove (8). The adversary C works as follows. It has access to an oracle $\textsc{Fn}$ which returns "real" evaluations of F or random samples. It runs A on input 1^λ. When A queries $\textsc{Sign}$ with a message vector $\mathbf{m}$, C evaluates $\mathsf{BadSet}^{\mathcal{O}}(\mathbf{m})$, where all of the calls to $\mathcal{O}$ are answered using the $\textsc{Fn}$ oracle provided to C. Finally, C outputs the value returned by BadSet. It is easy to see that C satisfies (8).

Proving (9) requires more work. It follows from the next lemma, which will complete the proof of the theorem.

Lemma 6 *Let $\ell' = \ell + \lceil \log d \rceil$ and π be a random function from $\{0,1\}^{\leq \ell'}$ to $\{0,1\}^n$. Let BadSet^π be defined as above. For $\alpha > 0$, let $\varepsilon = \varepsilon(\alpha, n)$ be the probability that a random number between 0 and $2^n - 1$ is α-smooth. Then for any $\mathbf{m} \subseteq \{0,1\}^\ell$ and $\alpha, n > 0$ such that $\varepsilon(\alpha, n) \leq 1/4$,*

$$\Pr[\mathsf{BadSet}^\pi(\mathbf{m})] \leq Q\left(2^{-2d+1} + 2\ell^3(\ell+d)Q/\alpha + \ell^2(e/4)^\ell\right).$$

For the proof, we define an alternative, more restrictive version of BadSet, called $\mathsf{SmoothSet}$. $\mathsf{SmoothSet}$ will not be efficiently computable, but we stress that this is inconsequential for our claims below. $\mathsf{SmoothSet}$ takes the same input and has access to the same oracle. On input $\mathbf{m}$, $\mathsf{SmoothSet}^\pi$ first computes $e^* \leftarrow \prod_{m \in \mathbf{m}} \prod_{t \in S(m)} \pi(t)$, and then computes the set $\mathsf{MNP}(\mathbf{m})$. Then it runs $\mathsf{FindPrimes}^\pi(x)$ for each $x \in \mathsf{MNP}(\mathbf{m})$, which is the following algorithm that returns a set of prime numbers.

```
$\mathsf{FindPrimes}^\pi(x)$
    If $\exists$ prime $p > \alpha$ s.t. $p \mid \pi(x)$ then
        Choose one $p$ meeting such criteria and return $\{p\}$
    If $|x| = \ell$ then
        If $\exists$ prime $p > \alpha, i \in [d]$ s.t. $p \mid \pi(x \,\|\, i)$ then
            Choose one $p$ meeting such criteria and return $\{p\}$
        Else
            Return $\{\perp\}$
    If $|x| < \ell$ then Return $\mathsf{FindPrimes}^\pi(x \,\|\, 0) \cup \mathsf{FindPrimes}^\pi(x \,\|\, 1)$
```

We add the rule that, if any call to $\mathsf{FindPrimes}^\pi(x)$ results in more than $2\ell^2$ recursive calls, then $\mathsf{SmoothSet}^\pi(\mathbf{m})$ halts the computation and returns true. $\mathsf{SmoothSet}^\pi(\mathbf{m})$ takes the union of all the sets returned by calls to $\mathsf{FindPrimes}^\pi(x)$. If $\perp$ was returned at any point, or if any of the returned primes divide e^*, then it returns true. Otherwise, it returns false.

We first argue that whenever $\mathsf{BadSet}^\pi(\mathbf{m})$ returns true due to the halting condition (i.e., if some call $\mathsf{Check}^\pi(x)$ results in more than $2\ell^2$ recursive calls) then $\mathsf{SmoothSet}^\pi(\mathbf{m})$ also returns true. Note that in such cases either $\mathsf{SmoothSet}^\pi(\mathbf{m})$ also returns true due to the halting condition or for at least one called value x

there were less recursive calls in $\mathsf{FindPrimes}^{\pi}(x)$ than in $\mathsf{Check}^{\pi}(x)$. But then there exists some value $u = x \,\|\, v$ such that: (i) $\mathsf{FindPrimes}^{\pi}(u)$ was called and did not generate any recursive call but (ii) $\mathsf{Check}^{\pi}(u)$ generated recursive calls. (i) implies that $\mathsf{FindPrimes}^{\pi}(u)$ outputted some prime $p > \alpha$ such that $p \mid \pi(u)$ and (ii) implies that $\pi(u) \mid e^*$. Therefore these two facts together imply that $p \mid e^*$ and so $\mathsf{SmoothSet}^{\pi}(\mathbf{m})$ returns true by definition.

The other case in which $\mathsf{BadSet}^{\pi}(\mathbf{m})$ returns true is if there is an $x \in \mathsf{MNP}(\mathbf{m})$ and $y \in \{0,1\}^{\ell-|x|}$ such that

$$\forall s \in \mathsf{Pref}(y), \quad \pi(x \,\|\, s) \mid e^*$$

and

$$\forall i = 1, \dots, d, \quad \pi(x \,\|\, y \,\|\, i) \mid e^*.$$

But then it also holds that $\forall s \in \mathsf{Pref}(y)$ all prime factors of $\pi(x \,\|\, s)$ divide e^* and $\forall i = 1, \dots, d$ all prime factors of $\pi(x \,\|\, y \,\|\, i)$ divide e^* and hence $\mathsf{SmoothSet}^{\pi}(\mathbf{m})$ also returns true. So we have

$$\Pr[\mathsf{BadSet}^{\pi}(\mathbf{m})] \leq \Pr[\mathsf{SmoothSet}^{\pi}(\mathbf{m})].$$

Thus it suffices to bound the latter probability. We recall that $\mathsf{SmoothSet}^{\pi}(\mathbf{m})$ returns true if, and only if, for some $x \in \mathsf{MNP}(\mathbf{m})$, $\mathsf{FindPrimes}^{\pi}(x)$ returns $\perp$ or performs more than $2\ell^2$ recursions or one of the returned primes divides e^*.

We first bound the probability that $\perp$ is in the set returned by a call to $\mathsf{FindPrimes}^{\pi}(x)$. This will happen if there is an $x \in \mathsf{MNP}(\mathbf{m})$ and $y \in \{0,1\}^{\ell-|x|}$ such that

$$\forall s \in \mathsf{Pref}(y), \quad \pi(x \,\|\, s) \text{ is } \alpha\text{-smooth}$$

and

$$\forall i = 1, \dots, d, \quad \pi(x \,\|\, y \,\|\, i) \text{ is } \alpha\text{-smooth}.$$

For a particular x and y this happens with probability $\varepsilon^{|y|+d} = \varepsilon^{\ell-|x|+d}$. A union bound over all x and y show that $\perp$ is in a set with probability at most

$$\begin{aligned}
\sum_{x \in \mathsf{MNP}(\mathbf{m})} \sum_{y \in \{0,1\}^{\ell-|x|}} \varepsilon^{\ell-|x|+d} &= \sum_{x \in \mathsf{MNP}(\mathbf{m})} 2^{\ell-|x|} \varepsilon^{\ell-|x|+d} \\
&\leq \sum_{i=1}^{\ell} Q 2^{\ell-i} \varepsilon^{\ell-i+d} \\
&= Q\varepsilon^d \sum_{i=0}^{\ell-1} (2\varepsilon)^i < 2Q\varepsilon^d \leq Q2^{-2d+1}
\end{aligned}$$

For the first inequality we used Lemma 3, and for the second we used the assumption that $\varepsilon \leq 1/4$ and applied the formula for summing a geometric series.

Next we need the following lemma that gives an upper bound on the number of recursive calls generated by $\mathsf{FindPrimes}^{\pi}(x)$ (i.e., this lemma bounds the probability that the halting condition makes $\mathsf{SmoothSet}$ return true due to this x). Note that since each call adds at most one element to the returned set, this will also bound the size of the returned set.

Lemma 7 *Let π be a random function from $\{0,1\}^*$ to $\{0,1\}^n$, $\alpha > 0$, and $\varepsilon = \varepsilon(\alpha, n) \leq 1/4$ be the probability that a random number between 0 and $2^n - 1$ is α-smooth. Then, for any $x \in \mathsf{MNP}(\mathbf{m})$, the probability that $\mathsf{FindPrimes}^\pi(x)$ generates more than $2\ell^2$ recursive calls is at most $\ell(e/4)^\ell$.*

Using Lemma 7, it is possible to complete the proof of Lemma 6. Suppose that $\mathsf{FindPrimes}^\pi(x)$ returns a set of at most $2\ell^2$ primes without ever returning $\perp$. The probability that each of the primes p divides e^* is at most the probability that p divides one of the $Q(\ell + d)$ random factors used in the product defining e^*. This probability is $1/p < 1/\alpha$, because both numbers are random and independent. A union bound over the factors shows that a given prime divides e^* with probability at most $Q(\ell + d)/\alpha$; another union bound over the (at most) $2\ell^2$ primes gives a bound of $2Q\ell^2(\ell + d)/\alpha$.

Finally, we sum the probability that $\mathsf{FindPrimes}^\pi(x)$ returns $\perp$ and the probabilities that $\mathsf{FindPrimes}^\pi(x)$ performs more than $2\ell^2$ recursions or one of the returned primes divides e^* for an $x \in \mathsf{MNP}(\mathbf{m})$ (there are at most $Q\ell$ by Lemma 3) in order to conclude the proof of Lemma 6 and hence of Theorem 5.

Proof of Lemma 7: Fix any $x \in \mathsf{MNP}(\mathbf{m})$. We bound the number of recursive calls in the computation of $\mathsf{FindPrimes}^\pi(x)$: Since each call adds at most one element to the returned set, this will also bound the size of that set. We consider the number of calls to $\mathsf{FindPrimes}$ on inputs of each length. Let X_i the number of calls on inputs of length i that are generated due to the computation of $\mathsf{FindPrimes}^\pi(x)$. We will show that with high probability, $X_i < 2\ell$ for all i, which gives $\sum_{i=1}^{\ell} X_i < 2\ell^2$.

$$\begin{aligned}\Pr[X_1, \ldots, X_\ell \leq 2\ell] &= 1 - \Pr[\exists X_i \ : \ X_i > 2\ell] \\ &= 1 - \sum_{i=1}^{\ell} \Pr[X_i > 2\ell | X_1, \ldots, X_{i-1} \leq 2\ell]\end{aligned}$$

We proceed to prove that for all i

$$\Pr[X_i > 2\ell | X_1, \ldots, X_{i-1} \leq 2\ell] < (e/4)^\ell. \tag{10}$$

Let j be the length of x. Then we have that $X_1, \ldots, X_j \leq 1$ with probability 1. Now consider the other X_i. Since each call results in at most two calls at the input with one bit appended, we have $X_i \leq 2X_{i-1}$ for all $i = j+1, \ldots, \ell$. We have that

$$\begin{aligned}\Pr[X_i > 2\ell | X_1, \ldots, X_{i-1} \leq 2\ell] = \Pr&\left[X_i > 2\ell \,\middle|\, \begin{matrix} X_1, \ldots, X_{i-2} \leq 2\ell \\ \wedge\, X_{i-1} < \ell \end{matrix}\right] \\ + \Pr&\left[X_i > 2\ell \,\middle|\, \begin{matrix} X_1, \ldots, X_{i-2} \leq 2\ell \\ \wedge\, \ell \leq X_{i-1} \leq 2\ell \end{matrix}\right].\end{aligned}$$

In the right-hand side the first probability is 0 because $X_i \leq 2X_{i-1} < 2\ell$. We are left to bound the the second term. Now we use the observation that X_i is the sum of $\ell' = X_{i-1}$ (with $2\ell \geq \ell' \geq \ell$) Bernoulli random variables with expectation 2ε, allowing us to apply the Chernoff bound in Lemma 4 giving

$$\begin{aligned}
\Pr[X_i > 2\ell | X_1, \ldots, X_{i-1} \leq 2\ell] &= \Pr[X_i > 2\ell | X_1, \ldots, X_{i-1} \leq 2\ell \ \wedge \ X_{i-1} \geq \ell] \\
&= \Pr\left[X_i > \left(\frac{\ell}{\varepsilon\ell'}\right) 2\varepsilon\ell' \middle| \begin{matrix} X_1, \ldots, X_{i-1} \leq 2\ell \\ \wedge\ X_{i-1} \geq \ell \end{matrix} \right] \\
&< \left(\frac{(e)^{\frac{\ell}{\varepsilon\ell'}-1}}{\left(\frac{\ell}{\varepsilon\ell'}\right)^{\frac{\ell}{\varepsilon\ell'}}} \right)^{2\varepsilon\ell'} \\
&= \frac{e^{2\ell - 2\varepsilon\ell'}}{\left(\frac{\ell}{\varepsilon\ell'}\right)^{2\ell}} \\
&= \frac{e^{2\ell - 2\ell(\varepsilon\ell'/\ell)}}{\left(\frac{\ell}{\varepsilon\ell'}\right)^{2\ell}}
\end{aligned}$$

where we used the fact that $\frac{\ell}{\varepsilon\ell'} > 1$ for $\varepsilon \leq 1/4$ in order to apply the Chernoff bound. Now letting $\beta = \frac{\varepsilon\ell'}{\ell}$

$$\frac{e^{2\ell - 2\ell(\varepsilon\ell'/\ell)}}{\left(\frac{\ell}{\varepsilon\ell'}\right)^{2\ell}} = \left(e^{1-\beta}\beta\right)^{2\ell}$$

We note that $0 < \beta < 1/2$ since $2\ell \geq \ell' \geq \ell$ and $0 < \varepsilon \leq 1/4$, and that this is an increasing function of β for the range $0 < \beta < 1/2$. Therefore the worst case is

$$\begin{aligned}
\Pr[X_i > 2\ell | X_1, \ldots, X_{i-1} \leq 2\ell] &< \left(\frac{\sqrt{e}}{2}\right)^{2\ell} \\
&= \left(\frac{e}{4}\right)^{\ell}
\end{aligned}$$

Note that this bound can be made stronger if the upper bound of ε is decreased. This proves Inequality 10 and then summing the probabilities over the l different lengths we conclude the proof of the lemma.

4 Setting the Parameters

From Theorem 5 we have

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{wcma}}_{\Pi,A}(\lambda) \ &\leq \mathbf{Adv}^{\mathrm{prf}}_{F,C}(\lambda) + \mathbf{Adv}^{\mathrm{srsa}}_{\mathsf{RSAGen},B}(\lambda) \\
&\quad + Q\left(2^{-2d+1} + 2\ell^3(\ell+d)Q/\alpha + \ell^2(e/4)^{\ell}\right).
\end{aligned}$$

Now we consider the case in which we want to give concrete upper bounds on the advantage of any wCMA adversary A. Using the previous equation we would like to have

$$\begin{aligned}2^{-\lambda} &\geq \mathbf{Adv}^{\mathrm{prf}}_{F,C}(\lambda) + \mathbf{Adv}^{\mathrm{srsa}}_{\mathsf{RSAGen},B}(\lambda) + Q\left(2^{-2d+1} + 2\ell^3(\ell+d)Q/\alpha + \ell^2(e/4)^\ell\right)\\ &= \mathbf{Adv}^{\mathrm{prf}}_{F,C}(\lambda) + \mathbf{Adv}^{\mathrm{srsa}}_{\mathsf{RSAGen},B}(\lambda) + Q2^{-2d+1} + 2Q^2\ell^3(\ell+d)/\alpha + Q\ell^2(e/4)^\ell\end{aligned}$$

for some security parameter λ. To obtain such bound, we will upper bound each of the five terms by $2^{-\lambda}/8$. For the first two terms, we only need to setup the function family F and the RSA parameter generator RSAGen in such a way that for any polynomial-time adversaries C and B we have

$$\mathbf{Adv}^{\mathrm{prf}}_{F,C}(\lambda) < 2^{-\lambda}/8$$

and

$$\mathbf{Adv}^{\mathrm{srsa}}_{\mathsf{RSAGen},B}(\lambda) < 2^{-\lambda}/8$$

For bounding

$$Q2^{-2d+1} < 2^{-\lambda}/8$$

we only need to set

$$d > \frac{\lambda + 4 + \log(Q)}{2}$$

Having fixed the value of d, we can now bound

$$2Q^2\ell^3(\ell+d)/\alpha < 2^{-\lambda}/8$$

by setting

$$\alpha > 2^{\lambda+4}Q^2\ell^3(\ell+d).$$

Now the value of α will determine the value of n, since we need $\varepsilon(\alpha, n) \leq 1/4$. To set the value of n we will use the Dickman function. The Dickman function $\rho(u)$ is an asymptotical approximation for the probability of a random number between 1 and x being $x^{1/u}$-smooth. Assuming that the Dickman function gives a good approximation in the range of interest, we can use the fact that $\rho(2.2) < 0.221$ [15] and set $n = \log(\alpha^{2.2})$ (i.e., we are choosing numbers up to $\alpha^{2.2}$) in order to obtain $\varepsilon(\alpha, \log(\alpha^{2.2})) < 1/4$ as required by Theorem 5.

Soundness of using Dickman approximation. As mentioned by Bach and Peralta [1] no discrepancy has been observed between the values predicted by the ρ function and the real smoothness probabilities, in the range of interest to algorithm designers. In addition, for small values of u (the case that we are interested), counts of smooth number have shown that the error of the approximation is as low as 2% even for values of x as low as 10^{15} (i.e., for numbers between 1 and 10^{15}, considering $10^{15/u}$-smoothness). Tables available in [1].

Acknowledgments

The third author was supported by a Sofja Kovalevskaja Award of the Alexander von Humboldt Foundation and the German Federal Ministry for Education and Research. Part of this work was done while the first two authors were visiting Ruhr-University Bochum, supported by the Sofja Kovalevskaja Award.

References

1. E. Bach and R. Peralta. Asymptotic semismoothness probabilities. *Math. Comput.*, 65(216):1701–1715, 1996.
2. N. Bari and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 480–494. Springer, May 1997.
3. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.
4. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, May / June 2006.
5. F. Böhl, D. Hofheinz, T. Jager, J. Koch, J. H. Seo, and C. Striecks. Practical signatures from standard assumptions. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 461–485. Springer, 2013.
6. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Aug. 2004.
7. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.
8. D. Catalano, D. Fiore, and B. Warinschi. Adaptive pseudo-free groups and applications. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 207–223. Springer, May 2011.
9. J.-S. Coron, H. Handschuh, and D. Naccache. ECC: Do we need to count? In K.-Y. Lam, E. Okamoto, and C. Xing, editors, *ASIACRYPT'99*, volume 1716 of *LNCS*, pages 122–134. Springer, Nov. 1999.
10. J.-S. Coron and D. Naccache. Security analysis of the Gennaro-Halevi-Rabin signature scheme. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 91–101. Springer, May 2000.
11. R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *ACM CCS 99*, pages 46–51. ACM Press, Nov. 1999.
12. Y. Dodis, R. Oliveira, and K. Pietrzak. On the generic insecurity of the full domain hash. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 449–466. Springer, Aug. 2005.
13. M. Fischlin. The Cramer-Shoup strong-RSA signature scheme revisited. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 116–129. Springer, Jan. 2003.
14. R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 123–139. Springer, May 1999.
15. A. Granville. Smooth numbers: computational number theory and beyond. *Algorithmic Number Theory*, 44:267–323, 2008.
16. L. C. Guillou and J.-J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 123–128. Springer, May 1988.
17. D. Hofheinz, T. Jager, and E. Kiltz. Short signatures from weaker assumptions. In *ASIACRYPT 2011*, LNCS, pages 1–12. Springer, 2011.

18. D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 21–38. Springer, Aug. 2008.
19. S. Hohenberger and B. Waters. Short and stateless signatures from the RSA assumption. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 654–670. Springer, Aug. 2009.
20. H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS 2000*. The Internet Society, Feb. 2000.
21. K. Kurosawa and K. Schmidt-Samoa. New online/offline signature schemes without random oracles. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 330–346. Springer, Apr. 2006.
22. H. Zhu. New digital signature scheme attaining immunity to adaptive chosen-message attack. *Chinese Journal of Electronics*, 10(4):484–486, Oct 2001.
23. H. Zhu. A formal proof of zhus signature scheme. Cryptology ePrint Archive, Report 2003/155, 2003. `http://eprint.iacr.org/`.