# On the Provable Security of an Efficient RSA-Based Pseudorandom Generator

Ron Steinfeld and Josef Pieprzyk and Huaxiong Wang
Dept. of Computing, Macquarie University, Australia
{rons, josef, hwang}@ics.mq.edu.au

September 21, 2006

**Abstract**

Pseudorandom Generators (PRGs) based on the RSA inversion (one-wayness) problem have been extensively studied in the literature over the last 25 years. These generators have the attractive feature of provable pseudorandomness security assuming the hardness of the RSA inversion problem. However, despite extensive study, the most efficient provably secure RSA-based generators output asymptotically only at most $O(\log n)$ bits per multiply modulo an RSA modulus of bitlength $n$, and hence are too slow to be used in many practical applications.

To bring theory closer to practice, we present a simple modification to the proof of security by Fischlin and Schnorr of an RSA-based PRG, which shows that one can obtain an RSA-based PRG which outputs $\Omega(n)$ bits per multiply and has provable pseudorandomness security assuming the hardness of a well-studied variant of the RSA inversion problem, where a constant fraction of the plaintext bits are given. Our result gives a positive answer to an open question posed by Gennaro (J. of Cryptology, 2005) regarding finding a PRG beating the rate $O(\log n)$ bits per multiply at the cost of a reasonable assumption on RSA inversion.

**Key Words:** Pseudorandom generator, RSA, provable security, lattice attack.

## 1 Introduction

*Background.* The RSA Pseudorandom bit generator (RSA PRG) works by iterating the RSA encryption mapping $x \to x^e \bmod N$ (with public RSA modulus $N$ of length $n$ bits and public exponent $e$ coprime to $\phi(N)$) on a secret random initial seed value $x_0 \in \mathbb{Z}_N$ to compute the intermediate state values $x_{i+1} = x_i^e \bmod N$ (for $i = 0, 1, 2, \ldots$) and outputting $r$ least-significant bits of the state value $x_i$ per iteration. The pseudorandomness of the RSA PRG (especially the case $r = 1$) was studied extensively by several researchers [26, 3, 40, 1, 20]. However, even the best security proof so far [20, 39] only applies to the case when only a very small number of bits $r = O(\log n)$ is output per iteration. Consequently, even with small public exponent $e$, these proven RSA PRG variants only output $O(\log n)$ bits per multiply modulo $N$ and hence are too slow for most practical applications. As far as we are aware, these are currently the most efficient RSA-based PRGs with proven pseudorandomness security.

*Our Approach.* Our approach to studying the provable security of efficient variants of the RSA PRG is based on two observations.

First, we observe that existing security proofs of the RSA PRG have always attempted to prove the security assuming the hardness of the classical RSA one-wayness problem (given RSA modulus $N$ and $y = x^e \bmod N$ for random $x \in \mathbb{Z}_N$, find $x$). If we instead make a stronger hardness assumption, we can hope to prove the security of much more efficient and practical variants of the RSA PRG,

with $r = \Omega(n)$. But we must be careful in choosing this stronger hardness assumption to ensure that it is based on substantial evidence – it must be a hard problem which has been undoubtedly studied extensively by experts. This leads to our second observation.

Our second observation is that over the last decade, beginning with the work of Coppersmith [15], the following variant of the RSA one-wayness problem has been studied explicitly:

**$(\delta, e)$-Small Solution RSA ($(\delta, e)$-SSRSA) Problem.** Given a random $n$-bit RSA modulus $N$, the coefficients of a univariate polynomial $f(z) = a_e z^e + a_{e-1} z^{e-1} + \cdots + a_0 \in \mathbb{Z}_N[z]$ of degree $e$ (with $a_e \in \mathbb{Z}_N^*$) and $y = f(\bar{z}) \bmod N$ for a random integer $\bar{z} < N^\delta$ (with $0 < \delta < 1$), find $\bar{z}$ (note that we will only be interested in instances where $f$ is such that $\bar{z}$ is uniquely determined by $(N, f, y)$).

The celebrated lattice-based attack of Coppersmith [15] shows that for small $e$, the $(\delta, e)$-SSRSA problem can be solved in polynomial time (in $n$) whenever $\delta < 1/e$. But when $\delta > 1/e + \epsilon$ for some constant $\epsilon > 0$, the lattice attack fails, and the only known attack (beyond factoring $N$) is to run the lattice attack $O(N^\epsilon)$ times for each guess of the $\epsilon \cdot n$ most-significant bits of $\bar{z}$. Hence, when $\epsilon$ is made sufficiently large to make the above lattice attack slower than factoring $N$ (namely even $\epsilon = O((\log n/n)^{2/3})$ suffices), the best known attack against $(1/e + \epsilon, e)$-SSRSA problem is to factor $N$. Importantly, this hardness assumption is supported by explicit evidence in the literature that the $(1/e + \epsilon, e)$-SSRSA problem has been studied by experts [16, 36, 14], yet these studies have not yielded an efficient algorithm for the $(1/e + \epsilon, e)$-SSRSA problem.

*Our Result.* We present a simple modification to the proof of security of the RSA PRG by Fischlin and Schnorr [20] which shows that assuming the hardness of a certain specific $(1/e + \epsilon, e)$-SSRSA one-wayness problem suffices to prove the pseudorandomness of the RSA PRG outputting $r = (1/2 - 1/e - \epsilon - o(1)) \cdot n$ LS bits per iteration. Our specific $(1/e + \epsilon, e)$-SSRSA one-wayness problem can be posed as RSA inversion with some known plaintext bits, namely: Given $N$, $y = [x^e]_N$, $r$ LS bits of $x$ and $w \approx n/2$ MS bits of $x$, for $x \in_R \mathbb{Z}_N$, find $x$. For small (constant) $e \geq 3$ we therefore obtain a throughput of $\Omega(n)$ output pseudorandom bits per multiply modulo the RSA modulus $N$, which is a significant improvement over the $O(\log n)$ bits per multiply throughput obtained using previous proof of security relative to the RSA assumption. We believe this answers in the positive an open question raised by Gennaro [21], who asked whether one can obtain a PRG which beats the rate $O(\log n)$ bits per multiply at the cost of a stronger but reasonable assumption on RSA inversion.

*Organization.* In Section 1.1 we discuss additional related work. Section 2 contains definitions and notations. In Section 3, we review the RSA PRG construction and its proof of security by Fischlin and Schnorr [20]. Section 4 presents our modified security proof for the RSA PRG assuming the hardness of a $(1/e + \epsilon, e)$-SSRSA problem. In Section 5, we estimate concrete parameters and associated PRG performance for given proven security level and security assumptions. In Section 6 we investigate the potential for performance improvements using a stronger hardness assumption. Section 7 discusses some applications of our result. Finally, Section 8 concludes the paper with some open problems.

### 1.1 Additional Related Work

Related PRG constructions can be divided in two classes.

The first class contains PRGs based on related hardness assumptions. The well known Blum-Blum-Shub (BBS) generator [9] has the same structure as the RSA PRG, but uses the Rabin squaring iteration function instead. Similar security results as for the RSA PRG are known for this generator [20], but we need a less known assumption to prove the security of efficient variants of this generator (see Section 6). The construction by Goldreich and Rosen [24] (improving on earlier work by Håstad et al [27]) uses an exponentiation iteration function and its security is proven assuming the hardness of factoring an RSA modulus, but its throughput is only $O(1)$ bits per multiply modulo the $n$ bit modulus, compared to $\Omega(n)$ bits per multiply for our construction. The Micali-Schnorr RSA-

based constructions [34] have a throughput of $\Omega(n)$ bits per multiply, but their pseudorandomness security is only proven assuming the *pseudorandomness* of the RSA function with small inputs (using our notation, it is actually a *decisional* version of the $(2/e, e)$-SSRSA problem), whereas for our construction we can prove pseudorandomness assuming only a much weaker assumption of *one-wayness* of RSA with small inputs (i.e. hardness of $(1/e+\epsilon, e)$-SSRSA inversion problem). The PRG of Boneh et al [13] also achieves a throughput of $\Omega(n)$ bits per multiply (and in fact may use a smaller *prime* modulus), but its provable pseudorandomness security also relies on a pseudorandomness assumption rather than a one-wayness assumption. However, similar to our SSRSA assumption, the conjectured hardness of the associated 'Modular Inverse Hidden Number Problem' is also based on the failure of lattice attacks against a system of non-linear equations with sufficiently large solutions.

The second class of PRGs achieve provable pseudorandomness based on different one-wayness assumptions. The construction by Impagliazzo and Naor [29] is based on the hardness of the Subset Sum problem. Although this construction is potentially very efficient, its concrete security against lattice-based subset sum attacks is difficult to estimate and requires carefully chosen large parameters with a small number of bits output per function evaluation. Very recently, a more practical 'QUAD' construction by Berbain et al [4] was proposed, using similar ideas to [29] in its security proof, but based on the hardness of solving a random system of multivariate quadratic equations over a finite field ('MQ' problem). We compare the practical performance of our construction with QUAD in Section 5. Finally, we remark that the best PRG based on the hardness of Discrete-Log (DL) problem is due to Gennaro [21] (improving on earlier work by Patel and Sundaram [37]). Its throughput is up to about $2n/c$ bits per multiply for a safe prime modulus $p$ of length $n$ bit, assuming that the discrete-log problem modulo $p$ is hard with $c$ bit exponents. Since factoring an $n$ bit RSA modulus and DL modulo an $n$ bit prime both can be solved in subexponential time $2^{O(n^{1/3}\log(n)^{2/3})}$ [32], to achieve security against 'square-root' DL attacks comparable to the difficulty of factoring $n$-bit RSA moduli, we must have $c = \Omega(n^{1/3}\log(n)^{2/3})$, so the throughput of Gennaro's construction is asymptotically only $O(n^{2/3}/\log(n)^{2/3}) = o(n)$ bits per multiply, compared to $\Omega(n)$ bits per multiply for our construction with same modulus length $n$.

Finally, we also wish to mention the lattice-based attacks of Blackburn et al [6, 5] on a class of PRGs having the same iterative structure as our RSA PRG. These attacks show that the RSA PRG is insecure when the number of bits output per iteration $r$ is larger than about $\frac{2}{3}n$ [6] for $e = 2$, and about $(1 - \frac{1}{e(e+1)/2+2})n$ [5] in the general case (these results are obtained for $r$ MS bits output per iteration and prime moduli, but we believe that with appropriate modifications they hold also for $r$ LS bits and RSA moduli). We remark that the general case attacks in [5] use low-dimension lattices and are rigorously proven. A heuristic extension of these attacks to high dimension lattices using the Coppersmith method [15] suggests that the RSA PRG is insecure asymptotically with $r \geq (1 - \frac{1}{e+1})n$ (we omit details of these calculations here). These lower bounds for insecure values of $r$ are greater by a factor of about 2 than the upper bounds on $r$ for which our security proof applies. Closing this remaining gap between best attack and best proof is an interesting open problem.

## 2 Preliminaries

**Notation.** For integers $x$ and $N$, we use $[x]_N$ to denote the remainder $x$ mod $N$. We use $L_r(x) = [x]_{2^r}$ to denote the $r$ least significant bits of the binary representation of $x$. Similarly, we use $M_r(x) = (x - L_{n-r}(x))/2^{n-r}$ (where $n$ is the bit length of $x$) to denote the $r$ most significant bits of the binary representation of $x$. For $x \in \mathbb{Z}_N$, we use $\widehat{M}_{N,r}(x)$ to denote any approximation of $x$ with additive error $|x - \widehat{M}_{N,r}(x)| \leq N/2^r$.

**Probability Distributions and Distinguishers.** Let $\mathcal{D}$ denote a probability distribution over $\{0,1\}^\ell$. We denote by $s \leftarrow \mathcal{D}$ the assignment to $s$ of a random element sampled from the distribution $\mathcal{D}$. If $S$ denotes a set then we let $s \in_R S$ denote the assignment to $s$ of a *uniformly* random element

sampled from $S$. Let $\mathcal{D}_1$ and $\mathcal{D}_2$ denote two probability distributions on some finite set. We say that an algorithm $\mathsf{D}$ is a $(T, \delta)$ distinguisher between $\mathcal{D}_1$ and $\mathcal{D}_2$ if $\mathsf{D}$ runs in time at most $T$ and has distinguishing advantage at least $\delta$ between $\mathcal{D}_1$ and $\mathcal{D}_2$, i.e. $|\Pr_{s\leftarrow\mathcal{D}_1}[\mathsf{D}(s) = 1] - \Pr_{s\leftarrow\mathcal{D}_2}[\mathsf{D}(s) = 1]| \geq \delta$. The *statistical distance* between two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ is $\frac{1}{2}\sum_s |\mathcal{D}_1(s) - \mathcal{D}_2(s)|$. It gives an upper bound on the distinguishing advantage of any distinguisher between $\mathcal{D}_1$ and $\mathcal{D}_2$, regardless of run-time.

**Pseudorandom Bit Generators (PRGs).** We use the following definition of pseudorandom generators and their concrete pseudorandomness.

**Definition 2.1 ($(T, \delta)$ PRG).** *A $(T, \delta)$ Pseudorandom Generator (family)* PRG *is a collection of functions $G_N : \mathcal{S}_N \to \{0,1\}^\ell$ indexed by $N \in \mathcal{I}_n$. Here $\mathcal{I}_n$ (PRG function index space) and $\mathcal{S}_N$ (PRG seed domain) are both efficiently samplable subsets of $\{0,1\}^n$, where $n$ is the security parameter. We require that any (probabilistic) distinguisher algorithm* $\mathsf{D}$ *running in time $T$ has distinguishing advantage at most $\delta$ between the* pseudorandom distribution $\mathcal{D}_{P,\ell}$ *and the* random distribution $\mathcal{D}_{R,\ell}$ *on $\ell$-bit strings, which are defined as follows:*

$$\mathcal{D}_{P,\ell} = \{s : N \in_R \mathcal{I}_n; x_0 \in_R \mathcal{S}_N; s = G_N(x_0)\}$$

*while*

$$\mathcal{D}_{R,\ell} = \{s : s \in_R \{0,1\}^\ell\}.$$

*If algorithm* $\mathsf{D}$ *runs in time $T$ and has distinguishing advantage at least $\delta$ between $\mathcal{D}_{P,\ell}$ and $\mathcal{D}_{R,\ell}$, we say that* $\mathsf{D}$ *is a $(T, \delta)$ distinguisher for* PRG.

**The RSA Inversion Problem.** The classical RSA inversion problem is defined as follows.

**Definition 2.2 ($(n, e)$-RSA problem).** *Let $e$ be a fixed integer. Let $\mathcal{I}_n$ denote the set of all $n$-bit RSA moduli $N = pq$ (for $p$,$q$ primes of $n/2$ bits each) such that $\gcd(e, (p-1)(q-1)) = 1$. The $(n, e)$-RSA inversion problem is the following: given $N \in_R \mathcal{I}_n$ and $y = [x^e]_N$ for $x \in_R \mathbb{Z}_N$, find $x$. We say that algorithm* $\mathsf{A}$ *is a $(T, \epsilon)$ inversion algorithm for $(n, e)$-RSA if* $\mathsf{A}$ *runs in time $T$ and succeeds with probability $\epsilon$ over the choice of $N \in_R \mathcal{I}_n$, $x \in_R \mathbb{Z}_N$ and the random coins of* $\mathsf{A}$.

**Lattices.** Let $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be a set of $n$ linearly independent vectors in $\mathbb{R}^n$. The set

$$\mathcal{L} = \{\mathbf{z} \colon \mathbf{z} = c_1\mathbf{b}_1 + \ldots + c_n\mathbf{b}_n; c_1, \ldots, c_n \in \mathbb{Z}\}$$

is called an *$n$-dimensional (full-rank) lattice* with *basis* $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$. Given a basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ for a lattice $\mathcal{L}$, we define the associated *basis matrix* $M_{\mathcal{L},\mathbf{B}}$ to be the (full-rank) $n \times n$ matrix whose $i$th row is the $i$th basis vector $\mathbf{b}_i$ for $i = 1, \ldots, n$. The quantity $|\det(M_{\mathcal{L},\mathbf{B}})|$ is independent of $\mathbf{B}$. It is called the *determinant* of the lattice $\mathcal{L}$ and denoted by $\det(\mathcal{L})$. Given any basis of a lattice $\mathcal{L}$, the well-known LLL algorithm [31] outputs in polynomial time a *reduced basis* for $\mathcal{L}$ consisting of short vectors. We use the following result [12] bounding the length of those vectors.

**Lemma 2.1.** *Let $\mathcal{L}$ be a lattice of dimension $d$ with basis matrix $\mathbf{B}_\mathcal{L}$ in lower diagonal form whose diagonal elements are greater or equal to 1. Then the Euclidean norm of the first two vectors in the LLL reduced basis for $\mathcal{L}$ is at most $2^{d/2}(\det(\mathcal{L}))^{\frac{1}{d-1}}$.*

# 3 High Level Overview of the Fischlin-Schnorr Security Proof

**The RSA PRG.** We begin by recalling the RSA PRG construction.

**Definition 3.1 ($(n, e, r, \ell)$-RSAPRG Pseudorandom Generator).** *The psuedorandom generator family $(n, e, r, \ell)$-*RSAPRG *is defined as follows. The PRG function index space $\mathcal{I}_n$ is the set of all*

*n-bit RSA moduli $N = pq$ (for p,q primes of $n/2$ bits each) such that $\gcd(e, (p-1)(q-1)) = 1$. Given index $N \in \mathcal{I}_n$ the PRG seed domain is $\mathbb{Z}_N$. Assume that $\ell$ is a multiple of $r$. Given a seed $x_0 \in_R \mathbb{Z}_N$, the PRG function $G_N : \mathbb{Z}_N \to \{0,1\}^\ell$ is defined by*

$$G_N(x_0) = (s_0, \ldots, s_{\ell/r-1}) : s_i = L_r(x_i), x_{i+1} = [x_i^e]_N \text{ for } i = 0, \ldots, \ell/r - 1.$$

As will become clear below, our result builds on the Fischlin-Schnorr result in essentially a 'black box' way, so our result can be understood without knowing most of the internal details of the reduction in [20]. Hence, in this section we provide only a very high-level overview of the basic security reduction [20] of the RSA PRG (in the case of $r$ LS bits output per iteration) from the RSA assumption. For the sake of completeness, we provide in the appendices the main ideas of the proofs of the main Lemmas.

Using our notation, the Fischlin-Schnorr security result can be stated concretely as follows.

**Theorem 3.1 (Fischlin-Schnorr [20]).** *For all $n \geq 2^9$, any $(T, \delta)$ distinguisher $\mathsf{D}$ for $(n, e, r, \ell)$-$\mathsf{RSAPRG}$ can be converted into a $(T_{INV}, \delta/9)$ inversion algorithm $\mathsf{A}$ for the $(n, e)$-RSA problem with run-time at most*

$$T_{INV} = 2^{2r+14}(\ell/\delta)^6 n \log(n) \cdot (T + O(\ell/r \log(e) n^2)). \quad (1)$$

*Proof.* We are given a distinguisher $\mathsf{D}$ with run-time $T$ and distinguishing advantage $\mathsf{Adv}(\mathsf{D}) \geq \delta$ between the *pseudorandom distribution* $\mathcal{D}_{P,\ell}$ (obtained by iterating $m = \ell/r$ times and outputting $r$ LS bits per iteration) and the *random* distribution $\mathcal{D}_{R,\ell}$ on $\ell$ bit strings, namely:

$$\mathcal{D}_{P,\ell} = \{G_N(x_0) : N \in_R \mathcal{I}_n; x_0 \in_R \mathbb{Z}_N\}$$

while

$$\mathcal{D}_{R,\ell} = \{s : s \in_R \{0,1\}^\ell\}.$$

We use $\mathsf{D}$ to construct the $(n, e)$-RSA inversion algorithm $\mathsf{A}$ as follows.

As a first step, we note that the pseudorandom distribution $\mathcal{D}_{P,\ell}$ is taken over the random choice of modulus $N \in_R \mathcal{I}_n$ as well as random seed $x_0 \in_R \mathbb{Z}_N$. For the remainder of the proof, we wish to fix $N$ and find a lower bound on the distinguishing advantage $\mathsf{Adv}_N(\mathsf{D})$ between $\mathcal{D}_{R,\ell}$ and the pseudorandom distribution $\mathcal{D}_{P,\ell,N}$ taken over just the random choice of $x_0 \in_R \mathbb{Z}_N$ for this fixed $N$, that is:

$$\mathcal{D}_{P,\ell,N} = \{G_N(x_0) : x_0 \in_R \mathbb{Z}_N\}.$$

To do so, we use an averaging argument over $N$ (refer to Appendix A for a proof).

**Lemma 3.1.** *There exists a subset $\mathcal{G}_n \subseteq \mathcal{I}_n$ of size at least $|\mathcal{G}_n| \geq \delta/2|\mathcal{I}_n|$ such that $\mathsf{D}$ has distinguishing advantage at least $\delta/2$ between the distributions $\mathcal{D}_{P,\ell,N}$ and $\mathcal{D}_{R,\ell}$ for all $N \in \mathcal{G}_n$.*

From now on we assume that $N \in \mathcal{G}_n$ (which happens with probability at least $\delta/2$ over $N \in_R \mathcal{I}_n$) so that $\mathsf{D}$ has distinguishing advantage at least $\delta/2$ between $\mathcal{D}_{P,\ell,N}$ and $\mathcal{D}_{R,\ell}$ (We remark that this first step is actually omitted in [20] which always assumes a fixed $N$; however we add this step since we believe it is essential for a meaningful security proof: to demonstrate an efficient algorithm for RSA inversion contradicting the RSA assumption, one must evaluate its success probability over the random choice of modulus $N$, since for any fixed $N$ an efficient algorithm always exists; it has built into it the prime factors of $N$).

We now convert $\ell/r$-iteration distinguisher $\mathsf{D}$ into a 1-iteration distinguisher $\mathsf{D}'$. This is a 'hybrid' argument using the fact that the mapping $x \to [x^e]_N$ is a permutation on $\mathbb{Z}_N$. Note that the 'hybrid' argument underlying this reduction has been known since the work of [25, 11] and it is not explicitly included in [20]. Refer to Appendix B for a proof.

**Lemma 3.2 ($m = \ell/r$ iterations to 1 iteration.).** *Any $(T, \delta)$ distinguisher $\mathsf{D}$ between the $m$-iteration pseudorandom distribution $\mathcal{D}_{P,\ell,N}$ and the random distribution $\mathcal{D}_{R,\ell}$ can be converted into a $(T + O(m \log(e)n^2), \delta/m)$ 1-iteration distinguisher $\mathsf{D}'$ between the distributions*

$$\mathcal{D}'_{P,r,N} = \{(y = [x^e]_N, s = L_r(x)) : x \in_R \mathbb{Z}_N\}$$

*and*

$$\mathcal{D}'_{R,r,N} = \{(y = [x^e]_N, s) : x \in_R \mathbb{Z}_N; s \in_R \{0,1\}^r\}.$$

The main part of the Fischlin-Schnorr reduction [20] is the conversion of the distinguisher $\mathsf{D}'$ into an inversion algorithm that recovers the RSA preimage $x$ from $y = [x^e]_N$ *with the help of some additional information on* $x$, namely $r$ least-significant bits of $[ax]_N$ and $[bx]_N$ for some randomly chosen known $a, b \in \mathbb{Z}_N$, as well as rough approximations to $[ax]_N$ and $[bx]_N$. This is stated more precisely as follows (although it is not needed for understanding our result, we refer the reader to Appendix C for a sketch of the proof of this result).

**Lemma 3.3 (Distinguisher to Inverter).** *For all $n \geq 2^9$, any $(T, \delta)$ distinguisher $\mathsf{D}'$ between the distributions $\mathcal{D}'_{P,r,N}$ and $\mathcal{D}'_{R,r,N}$ (see Lemma 3.2) can be converted into an inversion algorithm $\mathsf{A}'$ that, given $N$ and $(y = [x^e]_N, a \in_R \mathbb{Z}_N, s_1 = L_r([ax]_N), u_1 = \widehat{M}_{N,k}([ax]_N), b \in_R \mathbb{Z}_N, s_2 = L_r([bx]_N), u_2 = \widehat{M}_{N,l}([bx]_N))$, for any $x \in \mathbb{Z}_N$ with $k = 3\log(r/\delta) + 4$ and $l = \log(r/\delta) + 4$, outputs $x$ with probability $\epsilon'_{INV} \geq 2/9$ (over the choice of $a \in_R \mathbb{Z}_N$, $b \in_R \mathbb{Z}_N$ and the random coins of $\mathsf{A}'$) and runs in time $T'_{INV} = 4n \log(n)(r/\delta)^2 \cdot (T + O(n^2))$. Here $\widehat{M}_{N,k}(x)$ denotes any approximation of $x$ with additive error $|\widehat{M}_{N,k}(x) - x| \leq N/2^k$.*

*Putting it Together.* On input $(N, y = [x^e]_N)$, the RSA inversion algorithm $\mathsf{A}$ runs as follows. It applies Lemmas 3.1 and 3.2 to convert the $(T, \delta)$ distinguisher $\mathsf{D}$ into a $(T + O(m \log(e)n^2), \delta/(2m))$ distinguisher $\mathsf{D}'$ between distributions $\mathcal{D}'_{P,r,N}$ and $\mathcal{D}'_{R,r,N}$ which works for at least a fraction $\delta/2$ of $N \in \mathcal{I}_n$. Then $\mathsf{A}$ applies Lemma 3.3 to convert $\mathsf{D}'$ into the inversion algorithm $\mathsf{A}'$. $\mathsf{A}$ now chooses random $a$ and $b$ in $\mathbb{Z}_N$. Since $\mathsf{A}$ does not know the 'extra information' $s_1 = L_r([ax]_N)$, $u_1 = \widehat{M}_{N,k}([ax]_N)$, $s_2 = L_r([bx]_N)$ and $u_2 = \widehat{M}_{N,l}([bx]_N))$ required by $\mathsf{A}'$, $\mathsf{A}$ just exhaustively searches through all $N_G$ possible values of $(s_1, u_1, s_2, u_2)$ and runs $\mathsf{A}'$ on input $(N, y = [x^e], \widehat{s}_1, \widehat{u}_1, \widehat{s}_2, \widehat{u}_2)$ for every guessed possibility $(\widehat{s}_1, \widehat{u}_1, \widehat{s}_2, \widehat{u}_2)$ until $\mathsf{A}'$ succeeds to recover $x$. Note that to find an approximation $\widehat{M}_{N,k}([ax]_N)$ correct within additive error $N/2^k$ it is enough to search through $2^{k-1}$ uniformly spaced possibilities $(N/2^{k-1})i$ for $i = 0, \ldots, 2^{k-1} - 1$. Since $k = 3\log(2mr/\delta) + 4 = 3\log(2\ell/\delta) + 4$ and $l = \log(2\ell/\delta) + 4$, there are at most

$$N_G = 64(2\ell/\delta)^4 2^{2r} \tag{2}$$

guessing possibilities for $L_r([ax]_N)$, $\widehat{M}_{N,k}([ax]_N)$, $L_r([bx]_N)$, $\widehat{M}_{N,l}([bx]_N)$ to search through. So the run-time bound of $\mathsf{A}$ is

$$T_{INV} = N_G \cdot (4n \log(n)(2\ell/\delta)^2) \cdot (T + O(m \log(e)n^2)) = 2^{2r+14}(\ell/\delta)^6 n \log(n) \cdot (T + O(m \log(e)n^2)). \tag{3}$$

For at least a fraction $\delta/2$ of $N \in \mathcal{I}_n$, with the correct guessed value of the 'extra information', $\mathsf{A}'$ succeeds with probability at least $2/9$ over the choice of $a, b$. Hence we conclude that the success probability of $\mathsf{A}$ is at least $\epsilon_{INV} \geq \delta/9$, as claimed. □

We can interpret Theorem 3.1 as follows. Suppose we assume that the expected run-time $T_{INV}/\epsilon_{INV}$ of any $(T_{INV}, \epsilon_{INV})$ RSA inversion algorithm is at least $T_L$. Then Theorem 3.1 can be used to convert a $(T, \delta)$ distinguisher for $(n, e, r, \ell)$-RSAPRG to an RSA inverter contradicting our hardness

assumption only if we output at most $r$ bits per iteration, where

$$r < \frac{1}{2}\log\left(\frac{1}{9\cdot 2^{14}\cdot n\log n\ell^6\delta^{-7}}\cdot\frac{T_L}{T}\right). \tag{4}$$

Hence asymptotically, if we take $T_L = poly(n)$ (i.e. assume no poly-time RSA algorithm) then we get $r = O(\log(n))$ bits per iteration. If we assume that $T_L = O(2^{cn^{1/3}(\log n)^{2/3}})$ for constant $c$ (run-time of the Number Field Sieve factoring algorithm [32]) then we can have $r = O(n^{1/3}\log^{2/3} n)$. But in any case, $r = o(n)$.

# 4 Our Modified Security Proof from an SSRSA Problem

We now explain how we modify the above reduction to solve a well-studied SSRSA problem and the resulting improved PRG efficiency/security tradeoff.

Our goal is to remove the search factor $N_G = 64\cdot 2^{2r}(2\ell/\delta)^4$ from the run-time bound (3) of the reduction in the proof of Theorem 3.1. The simplest way to do so is to provide the inversion algorithm $\mathsf{A}$ with the correct values for the 'extra information' required by the inversion algorithm $\mathsf{A}'$ of Lemma 3.3. This leads us to consider the following (not well-known) inversion problem that we call $(n,e,r,k,l)$-FSRSA :

**Definition 4.1 ($(n,e,r,k,l)$-FSRSA Problem.).** *Given RSA modulus $N$, and $(y = [x^e]_N, a \in_R \mathbb{Z}_N, s_1 = L_r([ax]_N), u_1 = \widehat{M_k}([ax]_N), b \in_R \mathbb{Z}_N, s_2 = L_r([bx]_N), u_2 = \widehat{M_l}([bx]_N))$, for $x \in_R \mathbb{Z}_N$, find $x$ (here $\widehat{M}_{N,k}(x)$ denotes any approximation to $x$ with additive error $|\widehat{M}_{N,k}(x) - x| \leq N/2^k$). We say that algorithm $\mathsf{A}$ is a $(T,\eta)$ inversion algorithm for $(n,e,r,k,l)$-FSRSA if $\mathsf{A}$ runs in time at most $T$ and has success probability at least $\eta$ (over the random choice of $N \in_R \mathcal{I}_n$, $x,a,b \in_R \mathbb{Z}_N$ and the random coins of $\mathsf{A}$, where $\mathcal{I}_n$ is the same as in Definition 2.2).*

With the search factor $N_G$ removed from the Fischlin-Schnorr reduction we therefore have that the hardness of the inversion problem $(n,e,r,k,l)$-FSRSA (with $k = 3\log(2\ell/\delta)+4$ and $l = \log(2\ell/\delta)+4$) suffices for the 'simultaneous security' of the $r$ least-significant RSA message bits (i.e. indistinguishability of distributions $\mathcal{D}'_{P,r,N}$ and $\mathcal{D}'_{R,r,N}$ in Lemma 3.2) and hence the pseudorandomness of $(n,e,r,\ell)$-RSAPRG, with a much tighter reduction than the one of Theorem 3.1 relative to the RSA problem.

**Theorem 4.1.** *For all $n \geq 2^9$, any $(T,\delta)$ distinguisher $\mathsf{D}$ for $(n,e,r,\ell)$-RSAPRG can be converted into a $(T_{INV}, \delta/9)$ inversion algorithm $\mathsf{A}$ for the $(n,e,r,k,l)$-FSRSA problem (with $k = 3\log(2\ell/\delta)+4$ and $l = \log(2\ell/\delta)+4$) with run-time at most*

$$T_{INV} = 16\cdot(\ell/\delta)^2 n\log(n)\cdot(T + O(\ell/r\log(e)n^2)). \tag{5}$$

*Proof.* We use the same inversion algorithm $\mathsf{A}$ as in the proof of Theorem 3.1, except that when applying Lemma 3.3, $\mathsf{A}$ runs inversion algorithm $\mathsf{A}'$ just once using the correct values of $(a, b, s_1 = L_r([ax]_N), u_1 = \widehat{M}_{N,k}([ax]_N), s_2 = L_r([bx]_N), u_2 = \widehat{M}_{N,l}([bx]_N))$ given as input to $\mathsf{A}$, eliminating the search through $N_G = 64(2\ell/\delta)^4 2^{2r}$ possible values for $(s_1,u_1,s_2,u_2)$.

□

We defer to Section 6.1 our cryptanalysis of the $(n,e,r,k,l)$-FSRSA problem using the lattice-based method introduced by Coppersmith [15], which leads us to conjecture that the problem is hard whenever $r/n \leq 1/2 - 1/(2e) - (k+l)/2n - \epsilon$ for constant $\epsilon > 0$. This assumption together with the above reduction already implies the security of the efficient variants of $(n,e,r,\ell)$-RSAPRG with

$r = \Omega(n)$. Unfortunately, $(n, e, r, k, l)$-FSRSA is a new problem and consequently our conjecture on its hardness is not currently supported by extensive research. However, we will now show that in fact for $r/n = 1/2 - \max(k, l)/n - 1/e - \epsilon$ (note that this is smaller by $(\max(k, l) - (k + l)/2)/n + 1/(2e)$ than the largest secure value of $r/n$ conjectured above), the problem $(n, e, r, k, l)$-FSRSA is at least as hard as a specific $(1/e + \epsilon, e)$-SSRSA problem (i.e. with a specific univariate polynomial $f$ of degree $e$) which we call $(n, e, r, w)$-CopRSA and define as follows:

**Definition 4.2 ($(n, e, r, w)$-CopRSA Problem.).** *Given RSA modulus $N$, and $(y = [x^e]_N, s_L = L_r(x), s_H = M_{n/2+w}(x))$, for $x \in_R \mathbb{Z}_N$, find $x$ (here $M_k(x)$ denotes the $k$ most-significant bits of the binary representation of $x$). We say that algorithm $\mathsf{A}$ is a $(T, \eta)$ inversion algorithm for $(n, e, r, w)$-CopRSA if $\mathsf{A}$ runs in time at most $T$ and has success probability at least $\eta$ (over the random choice of $N \in_R \mathcal{I}_n$, $x \in_R \mathbb{Z}_N$ and the random coins of $\mathsf{A}$, where $\mathcal{I}_n$ is the same as in Definition 2.2).*

To see that $(n, e, r, w)$-CopRSA problem is a specific type of SSRSA problem, note that it is equivalent to finding a small solution $\bar{z} < 2^{n/2-(r+w)}$ (consisting of bits $r + 1, \ldots, (n/2 - w)$ of the randomly chosen integer $x$) to the equation $f(\bar{z}) \equiv y \bmod N$, where the degree $e$ polynomial $f(z) = (2^r z + s)^e$, where $s = s_H \cdot 2^{n/2-w} + s_L$ is known. Hence $(n, e, r, w)$-CopRSA is a $(1/e + \epsilon, e)$-SSRSA problem when $1/2 - (r + w)/n = 1/e + \epsilon$, i.e. $r/n = 1/2 - 1/e - \epsilon - w/n$.

**Theorem 4.2.** *Let $\mathsf{A}'$ be a $(T', \eta')$ attacker against $(n, e, r, w - 1, w - 1)$-FSRSA. Then we construct a $(T, \eta)$ attacker $\mathsf{A}$ against $(n, e, r, w)$-CopRSA with*

$$T = 4T' + O(n^2) \text{ and } \eta = \eta' - 4/2^{n/2}.$$

*Proof.* On input $(N, y = [x^e]_N, s_L = L_r(x), s_H = M_{n/2+w}(x))$, for $N \in_R \mathcal{I}_n$ and $x \in_R \mathbb{Z}_N$, the attacker $\mathsf{A}$ runs as follows:

- Choose a uniformly random $b \in_R \mathbb{Z}_N$.
- Compute an integer $c$ coprime to $N$ with $|c| < N^{1/2}$ such that $|[b \cdot c]_{\underline{N}}| < N^{1/2}$ (here $[z]_{\underline{N}} \in (-N/2, N/2)$ denotes the 'symmetrical' residue of $z$ modulo $N$, i.e. $[z]_{\underline{N}} \stackrel{\text{def}}{=} [z]_N$ if $[z]_N \in [0, N/2)$ and $[z]_{\underline{N}} \stackrel{\text{def}}{=} [z]_N - N$ if $[z]_N \in (N/2, N)$). It is well known that such a $c$ exists and can be computed efficiently (in time $O(n^2)$) using continued fractions (see, e.g. Lemma 16 in [35]).
- Observe that $[cx]_N = cx - \omega_c N$, where $\omega_c = \lfloor \frac{cx}{N} \rfloor$. Let $\widehat{x} = s_H \cdot 2^{n/2-w}$. Notice that $\widehat{x}$ approximates $x$ within additive error $\Delta_x \leq 2^{n/2-w}$ and consequently the rational number $\frac{c\widehat{x}}{N}$ approximates $\frac{cx}{N}$ within additive error $\frac{|c|\Delta_x}{N} \leq \Delta_x / N^{1/2} \leq 2^{n/2-w}/2^{(n-1)/2} < 1$, where we have used the fact that $|c| < N^{1/2}$ and $w \geq 1$. It follows that $\omega_c \in \{\lfloor \frac{c\widehat{x}}{N} \rfloor, \lfloor \frac{c\widehat{x}}{N} \rfloor \pm 1\}$ (where the $+$ sign applies if $c \geq 0$ and the $-$ sign applies otherwise). So $\mathsf{A}$ obtains 2 candidates for $\omega_c$.
- Using $L_r([cx]_N) = L_r(cx - \omega_c N) = L_r(L_r(c) \cdot L_r(x) - L_r(\omega_c N))$, $\mathsf{A}$ computes (with the known $s_L = L_r(x)$, $c$ and $N$) 2 candidates for $L_r([cx]_N)$ from the 2 candidates for $\omega_c$.
- Similarly, writing $[bcx]_N = [bc]_{\underline{N}} \cdot x - \omega_{bc} N$, with $\omega_{bc} = \lfloor \frac{[bc]_N x}{N} \rfloor$, using $|[bc]_{\underline{N}}| < N^{1/2}$ we obtain $\omega_{bc} \in \{\lfloor \frac{[bc]_N \widehat{x}}{N} \rfloor, \lfloor \frac{[bc]_N \widehat{x}}{N} \rfloor \pm 1\}$ (with $+$ sign if $[bc]_{\underline{N}} \geq 0$ and $-$ sign otherwise), so $\mathsf{A}$ also computes 2 candidates for $\omega_{bc}$ and two corresponding candidates for $L_r([bcx]_N) = L_r([bc]_{\underline{N}} x - \omega_{bc} N) = L_r(L_r([bc]_{\underline{N}}) L_r(x) - \omega_{bc} N)$.
- Using $\widehat{x}$ and the 2 candidates for $\omega_c$ computed above, $\mathsf{A}$ computes two candidate approximations $c\widehat{x} - \omega_c N$ for $[cx]_N$. Since $\widehat{x}$ approximates $x$ within additive error $\Delta_x \leq 2^{n/2-w}$ we have that $c\widehat{x} - \omega_c N$ approximates $[cx]_N$ within additive error $|c|\Delta_x \leq N^{1/2} 2^{(n-1)/2} / 2^{w-1/2} \leq N/2^{w-1}$ using $N \geq 2^{n-1}$.

- Similarly, using $\widehat{x}$ and the 2 candidates for $\omega_{bc}$ computed above, $\mathsf{A}$ computes two candidate approximations $[bc]_{\underline{N}}\widehat{x} - \omega_{bc}N$ for $[bcx]_N$, one of which has additive error $|[bc]_{\underline{N}}|\Delta_x \leq N/2^{w-1}$.
- Choose a uniformly random $a \in \mathbb{Z}_N^*$ and compute $y' = [(a^{-1}c)^e y]_N = [(a^{-1}cx)^e]_N$.
- Collecting all of the above information, $\mathsf{A}$ obtains 4 candidates for $(N, y' = [(a^{-1}cx)^e]_N, a, s_1 = L_r([cx]_N), u_1 = \widehat{M}_{N,w-1}([cx]_N), b' = [ab]_N, s_2 = L_r([bcx]_N), u_2 = \widehat{M}_{N,w-1}([bcx]_N))$. Note that this is a valid instance of $(n, e, r, w-1, w-1)$-FSRSA. Furthermore, it has almost exactly the correct distribution, since the triple $(x' = [a^{-1}cx]_N, a, b' = [ab]_N)$ is uniformly random in $\mathbb{Z}_N \times \mathbb{Z}_N^* \times \mathbb{Z}_N$ thanks to the uniformly random choice of $(x, a, b) \in \mathbb{Z}_N \times \mathbb{Z}_N^* \times \mathbb{Z}_N$. The FSRSA instance distribution is not exactly correct because here $a$ is uniform on $\mathbb{Z}_N^*$ while it should be uniform on $\mathbb{Z}_N$. However, simple calculation shows that the statistical distance between the uniform distribution on $\mathbb{Z}_N^*$ and the uniform distribution on $\mathbb{Z}_N$ is negligible, namely $1 - \phi(N)/N = (p + q - 1)/N \leq 4/2^{n/2}$.
- $\mathsf{A}$ runs $\mathsf{A}'$ on the above 4 candidate $(n, e, r, w-1, w-1)$-FSRSA instances. On one of those runs, $\mathsf{A}'$ outputs $x' = [a^{-1}cx]_N$ with probability at least $\eta - 4/2^{n/2}$, from which $x$ is easily recovered as $x = [ac^{-1}x']_N$.

Note that the run-time of $\mathsf{A}$ is bounded as $T \leq 4T' + O(n^2)$ and $\mathsf{A}$ succeeds with probability at least $\eta - 4/2^{n/2}$, as required. This completes the proof. □

So, combining Theorems 4.1 and 4.2, we conclude:

**Corollary 4.1.** *For all $n \geq 2^9$, any $(T, \delta)$ distinguisher $\mathsf{D}$ for $(n, e, r, \ell)$-RSAPRG can be converted into a $(T_{INV}, \epsilon_{INV})$ inversion algorithm $\mathsf{A}$ for the $(n, e, r, w)$-CopRSA problem (with $w = 3\log(2\ell/\delta) + 5$) with*

$$T_{INV} = 64 \cdot (\ell/\delta)^2 n \log(n) \cdot (T + O(\ell/r \log(e) n^2)) \text{ and } \epsilon_{INV} = \delta/9 - 4/2^{n/2}. \tag{6}$$

*Remark 1.* Our reduction (Theorem 4.2) from the CopRSA to FSRSA problem also extends with some small modifications to the case of even $e$ (details will be given in a later version of this paper). For $e = 8$, the resulting PRG actually gives better rate than the best odd exponent assuming the hardness of SSRSA.

*Remark 2.* Fischlin and Schnorr [20] also outline an alternative security reduction (worked out in detail and optimized for the Rabin iteration function by Sidorenko and Schoenmakers [39]) for the $(n, e, r, \ell)$-RSAPRG with $r > 1$ based on a general 'Computational XOR Lemma' [40, 22]. However, this alternative reduction has an inherent exponential run-time factor $2^{2r}$ which we do not know how to eliminate, even using our stronger SSRSA assumption on RSA inversion.

# 5 Concrete Parameters and Estimated Performance

Using (6) we obtain an upper bound on the pseudorandom string length $\ell$ for a given security level $(T, \delta)$ and assumed expected run-time lower bound $T_L$ for breaking the $(n, e, r, 3\log(2\ell/\delta) + 5)$-CopRSA problem. Recall that the latter is a $(1/e + \epsilon, e)$-SSRSA problem when

$$r/n = 1/2 - 1/e - \epsilon - (3\log(2\ell/\delta) + 5)/n, \tag{7}$$

and that $(1/e + \epsilon, e)$-SSRSA problem is conjectured to take time $T_L = \min(T_F(n), T_C(n, \epsilon))$, where $T_F(n)$ is a lower bound for factoring $N$ and $T_C(n, \epsilon) = poly(n) \cdot 2^{\epsilon n}$ is the time for the Coppersmith attack on $(1/e + \epsilon, e)$-SSRSA.

Asymptotically, we therefore have for any constant $\epsilon > 0$ that $T_L = T_F(n)$ since $T_F(n)$ is subexponential in $n$, so for any $\ell/\delta = poly(n)$ and $e \geq 3$ we can use $r/n = 1/2 - 1/e - \epsilon - o(1)$, i.e. $r = \Omega(n)$.

The exact bound on $r$ for a given modulus length $n$ depends on the value of $\epsilon$ such that $T_F(n) = T_C(n, \epsilon)$. To estimate concrete values, we use the Number Field Sieve (NFS) factoring run-time model from [33]; namely we use Pentium II processor instructions as our time unit, and we assume that NFS run-time is $T_F(n) = c_F \exp(1.9229(n \ln(2))^{1/3}(\ln(n \ln(2))^{2/3})$, where constant $c \approx 17 \cdot 10^{-3}$ is determined from the estimated run-time $T(512) \approx 3 \cdot 10^{17}$ instructions taken to factor the 512-bit number RSA155. We also assume (conservatively) that the Coppersmith attack run-time on $(1/e + \epsilon, e)$-SSRSA is $T_C(n, \epsilon) = c_F \cdot 2^{\epsilon n}$. In table 1, we computed for each modulus length $n$ using (7) and (6) (neglecting the $O(\ell/r \log(e) n^2)$ overhead time) the largest values of $r$ and $\ell$ for which a distinguisher with run time $T = 2^{70}$ instructions and distinguishing advantage $\delta = \frac{1}{100}$ contradicts the assumed lower bound $T_L = T_F(n) = T_C(n, \epsilon)$ on the expected run-time $T_{INV}/\epsilon_{INV}$ of any $(1/e+\epsilon, e)$-SSRSA attacker. We assumed $e = 9$ (for this value of $e$ the throughput $r/4$ in bits per multiplication modulo $N$ is approximately maximised). The results are summarised in Table 1. Also shown in the rightmost 2 columns is the improved provable performance achievable using $e = 2$ together with the stronger FS-RSA assumption (see Section 6).

| $n$ **(bit)** | $\log(\ell)$ | **Rate,**$e = 9$ **(bit/mult)** | **Throughput (Mbit/s)** | **Rate,**$e = 2$ **(bit/mult)** | **Throughput (Mbit/s)** |
|---|---|---|---|---|---|
| 3072 | 9.3 | 267 | 1.31 | 660 | 3.2 |
| 4096 | 18.0 | 360 | 1.00 | 899 | 2.5 |
| 5120 | 25.4 | 454 | 0.80 | 1140 | 2.0 |
| 6144 | 32.0 | 549 | 0.67 | 1383 | 1.7 |

Table 1: Estimate of achievable performance for provable $T = 2^{70}$ instructions distinguishing time to achieve advantage $\delta = \frac{1}{100}$, using $e = 9$ (assuming hardness of the CopRSA SSRSA problem) and $e = 2$ (assuming hardness of FSRSA problem - see Section 6). Throughput ('Thrpt') columns are estimated throughput based on Wei Dai's Crypto++ benchmarks page [17] (for Pentium 4 2.1GHz processor) and extrapolation assuming classical arithmetic.

The above estimates suggest that we can (with $n = 6144$ bit) achieve a rate around 25000 cycles/byte (0.67 Mbit/s with 2.1 GHz clock) on a Pentium 4 Processor, outputting more than $2^{30}$ bits with provable $2^{70}$ instructions distinguishing run-time (under the $(1/e + \epsilon, e)$-SSRSA assumption). This seems to be close to practical requirements of some stream cipher applications (it is several hundred times faster than the basic Blum-Blum Shub generator outputting one bit per iteration with the same modulus length). Compared to the recent provably secure QUAD PRG construction [4] (based on the 'MQ' problem), our PRG seems to have a lower throughput, although it is difficult to make a fair comparison since unlike our figures above, the performance figures reported in [4] (between 3000 and 4500 cycles/byte on Pentium 4) are for a 'practical' choice of parameters, smaller than those for which the security proof can be applied. A possible advantage of our construction is its significantly smaller static parameters (i.e. non-secret parameters defining the pseudorandom generator) of length $n \approx 5$ kbit, while in [4] the static parameters are longer than 1 Mbit (this might allow our construction to be implemented with less code memory requirements). On the other hand, our construction has a longer state and is based on the hardness of factoring so is insecure against potential future quantum attacks, while the MQ problem in [4] may be secure even against such attacks.

# 6 Potential Improvements

## 6.1 Cryptanalysis of the FS-RSA Problem

As observed in Section 4, the $(n, e, r, k, l)$-FSRSA problem, although not well-known, gives a more direct proof of security for the RSA PRG than the SSRSA problem. Hence it is interesting to cryptanalyze this problem using the Coppersmith lattice-based attack methods [15] and see whether the problem may be hard for larger values of $r$ than the corresponding SSRSA problem, possibly leading to improved efficiency of the RSA PRG. Indeed, in this section we describe a 'Coppersmith-type' lattice attack on $(n, e, r, k, l)$-FSRSA (which we believe is essentially optimal) and show that it is likely to succeed only when $r/n \geq 1/2 - (k + l)/(2n) - 1/(2e)$. This value of $r/n$ is larger by about $1/(2e) + (\max(k, l)/n - (k + l)/(2n))$ than that the largest value for which the corresponding SSRSA problem in Section 4 is secure, leading to improved throughput for the RSA PRG by using this stronger assumption.

The attack on $(n, e, r, k, l)$-FSRSA problem works as follows. First we reduce the problem to solving two modular equations in two small unknowns $z_1$ and $z_2$. Namely, given $(y = [x^e]_N, a \in_R \mathbb{Z}_N, s_1 = L_r([ax]_N), u_1 = \widehat{M}_{N,k}([ax]_N), b \in_R \mathbb{Z}_N, s_2 = L_r([bx]_N), u_2 = \widehat{M}_{N,l}([bx]_N))$, we have

$$x^e \equiv y \pmod{N}, \tag{8}$$

$$[ax]_N = s_1 + \bar{z}_1' \cdot 2^r; |[ax]_N - u_1| \leq N/2^k \tag{9}$$

and

$$[bx]_N = s_2 + \bar{z}_2' \cdot 2^r; |[bx]_N - u_2| \leq N/2^l \tag{10}$$

where $\bar{z}_1' < N/2^r$ and $\bar{z}_2' < N/2^r$ consist of the $n - r$ MS bits of $[ax]_N$ and $[bx]_N$, respectively. Let $\widehat{z}_1 = \lfloor \frac{u_1 - s_1}{2^r} \rfloor$. From (9) we conclude that $|\bar{z}_1' - \widehat{z}_1| \leq |(\frac{[ax]_N - s_1}{2^r}) - (\frac{u_1 - s_1}{2^r})| + 1 \leq N/2^{r+k} + 1 \leq N/2^{r+k-1}$ (for $2^{r+k} < N$) and hence letting $\bar{z}_1 = \bar{z}_1' - \widehat{z}_1$ we obtain $[ax]_N = (s_1 + 2^r \widehat{z}_1) + 2^r \bar{z}_1$ where integer $|\bar{z}_1| < N/2^{r+k-1}$. Similarly, from (10) we obtain $[bx]_N = (s_2 + 2^r \widehat{z}_2) + 2^r \bar{z}_2$ where integer $|\bar{z}_2| < N/2^{r+l-1}$ (for $2^{r+l} \leq N$) and $\widehat{z}_2 = \lfloor (u_2 - s_2)/2^r \rfloor$. Treating the last two equations for $[ax]_N$ and $[bx]_N$ as congruences modulo $N$, we eliminate the unknown variable $x$ (by multiplying the second congruence by $[ab^{-1}]_N$ and subtracting from the first) to obtain a single linear polynomial $f(z_1, z_2)$ in two variables $z_1, z_2$, having the desired small unknowns $\bar{z}_1, \bar{z}_2$ as a zero modulo $N$ (i.e. $f(\bar{z}_1, \bar{z}_2) \equiv 0 \pmod{N}$), namely:

$$f(z_1, z_2) = \alpha \cdot z_1 + z_2 + \beta, \tag{11}$$

where $\alpha = [-ab^{-1}]_N$ and $\beta = [-a^{-1}b2^{-r}(s_1 + 2^r \widehat{z}_1) + 2^{-r}(s_2 + 2^r \widehat{z}_2)]_N$ are known. Also, substituting $x \equiv a^{-1}(s_1 + 2^r \widehat{z}_1) + 2^r a^{-1} \widehat{z}_1 \pmod{N}$ into (8) we obtain a degree $e$ univariate polynomial in $z_1$ having the small unknown $\bar{z}_1$ as a zero modulo $N$ (i.e. $g(\bar{z}_1) \equiv 0 \pmod{N}$):

$$g(z_1) = (z_1 + \widehat{\alpha})^e - \widehat{\beta}, \tag{12}$$

where $\widehat{\alpha} = [2^{-r} s_1 + \widehat{z}_1]_N$ and $\widehat{\beta} = [-(a2^{-r})^e y]_N$ are known. To find the small zero $(\bar{z}_1, \bar{z}_2)$ of (11) and (12) we use the bivariate modular polynomial lattice method of Coppersmith [15] as simplified by Howgrave-Graham [28] and used in many subsequent works. Namely, for an integer $m$ we use the polynomials $f(z_1, z_2)$ and $g(z_1)$ to construct the following family of polynomials $h_{i,k}(z_1, z_2)$ indexed by a pair of integers $i = 0, 1, \ldots, me$ (which we refer to as the 'block index') and $k = 0, \ldots, i$ for each block $i$ (which we call the 'inner index'):

$$h_{i,k}(z_1, z_2) = N^{me - (i - k + \lfloor \frac{k}{e} \rfloor)} z_1^{[k]_e} g(z_1)^{\lfloor \frac{k}{e} \rfloor} f(z_1, z_2)^{i-k}. \tag{13}$$

Observe that each of the polynomials $h_{i,k}(z_1, z_2)$ has $(\bar{z}_1, \bar{z}_2)$ as a zero modulo $N^{me}$, because $f(\bar{z}_1, \bar{z}_2)^{i-k} \equiv 0 \pmod{N^{i-k}}$ and $g(\bar{z}_1)^{\lfloor \frac{k}{e} \rfloor} \equiv 0 \pmod{N^{\lfloor \frac{k}{e} \rfloor}}$.

It follows that any integer linear combination of the polynomials $h_{i,k}(z_1, z_2)$ also has $(\bar{z}_1, \bar{z}_2)$ as a zero modulo $N^{me}$. Let $B_1 = N/2^{r+k-1}$ and $B_2 = N/2^{r+l-1}$ denote the upper bounds derived above on $|\bar{z}_1|$ and $|\bar{z}_2|$, respectively. We set up a lattice $\mathcal{L}$ to search for linear combinations of the polynomials $h_{i,k}(z_1, z_2)$, which have sufficiently small coefficients such that they have $(\bar{z}_1, \bar{z}_2)$ as a zero over the integers, not just modulo $N^{me}$. Given two such linearly independent polynomials we can take their resultant to obtain a single univariate polynomial equation in $z_1$ over the integers which is easy to solve. More precisely, we use the following Lemma due originally to Howgrave-Graham [28] (see also [12]). For a bivariate polynomial $h(z_1, z_2) = \sum_{i,j} c_{i,j} z_1^i z_2^j \in \mathbb{Z}[z_1, z_2]$, define the norm of $h(z_1, z_2)$ as $\|h(z_1, z_2)\| = (\sum_{i,j} |c_{i,j}|^2)^{1/2}$.

**Lemma 6.1.** *Let $h(z_1, z_2)$ be a polynomial in $\mathbb{Z}[z_1, z_2]$ having at most $d$ monomials. If (1) $h(\bar{z}_1, \bar{z}_2) \equiv 0 \pmod{N^{me}}$ with $|\bar{z}_1| < B_1$ and $|\bar{z}_2| < B_2$, and (2) $\|h(B_1 z_1, B_2 z_2)\| < N^{me}/\sqrt{d}$, then $h(\bar{z}_1, \bar{z}_2) = 0$ holds over $\mathbb{Z}$.*

The square basis matrix $B_{\mathcal{L}}$ for the lattice $\mathcal{L}$ has its rows and columns indexed by pairs of integers $(i, k)$ with $i$ increasing in the range $0, \ldots, me$ (block index) and $k = 0, \ldots, i$ (inner index) for each block index $i$ (i.e. the first block $i = 0, k = 0$ appears first, then the second block $i = 1$,$k = 0, 1$ and so on). On the $(i, k)$th row we place the coefficients of the polynomial $h_{i,k}(B_1 z_1, B_2 z_2)$. We order the monomials of $h_{i,k}(B_1 z_1, B_2 z_2)$ such that the $(i', k')$th column of the $(i, k)$th row of $B_{\mathcal{L}}$ contains the coefficient of the monomial $z_1^{k'} z_2^{i'-k'}$ of $h_{i,k}(B_1 z_1, B_2 z_2)$. Notice that with this ordering, the polynomial $h_{i,k}(B_1 z_1, B_2 z_2)$ is the first one to have a non-zero coefficient for the monomial $z_1^k z_2^{i-k}$ so that the basis matrix $\mathcal{B}_{\mathcal{L}}$ is in lower diagonal form, and has the following entry $\mathcal{B}_{\mathcal{L}}[(i, k), (i, k)]$ in the $(i, k)$th position along the diagonal:

$$\mathcal{B}_{\mathcal{L}}[(i, k), (i, k)] = N^{me-(i-k+\lfloor \frac{k}{e} \rfloor)} B_1^k B_2^{i-k}. \tag{14}$$

It follows that the determinant of lattice $\mathcal{L}$ is the product of these diagonal elements of $\mathcal{B}_{\mathcal{L}}$. A straightforward calculation using (14) gives:

$$\det(\mathcal{L}) = N^{me \cdot d(me) - W(m,e)} (B_1 B_2)^{D(me)/2}, \tag{15}$$

where the function $D(me)$ is given by

$$D(me) \stackrel{\text{def}}{=} \sum_{i=0}^{me} i(i+1) = \frac{1}{3} me(me + \frac{1}{2})(me+1) + \frac{1}{2} me(me+1), \tag{16}$$

the function $W(m, e)$ is given by

$$W(m, e) \stackrel{\text{def}}{=} \sum_{i=0}^{me} \sum_{k=0}^{i} (i-k) + \lfloor \frac{k}{e} \rfloor = \frac{1}{2} D(me) + \frac{e^2}{6} m(m - \frac{1}{2})(m-1) + \frac{(e+2)e}{4} m(m-1) + m, \tag{17}$$

and $d(me) = \frac{1}{2}(me+1)(me+2)$ denotes the dimension of $\mathcal{L}$. We run the LLL algorithm on the basis $\mathcal{B}_{\mathcal{L}}$ for lattice $\mathcal{L}$ and hope that it returns two sufficiently short linearly independent polynomials having $(\bar{z}_1, \bar{z}_2)$ as a zero over $\mathbb{Z}$. Let $h_1(z_1, z_2)$ and $h_2(z_1, z_2)$ denote the polynomials corresponding to the first two vectors in the reduced basis of $\mathcal{L}$ returned by LLL. By Lemma 2.1 (see Section 2; the lemma applies since all diagonal elements (15) of the lower-diagonal basis matrix $B_{\mathcal{L}}$ are greater than 1), we know that the norms $\|h_1(B_1 z_1, B_2 z_2)\|$ and $\|h_2(B_1 z_1, B_2 z_2)\|$ will be at most $2^{d(me)/2} \det(\mathcal{L})^{\frac{1}{d(me)-1}}$. Therefore, (recalling that $(\bar{z}_1, \bar{z}_2)$ is a zero of $h_1$ and $h_2$ modulo $N^{me}$), in order to apply Lemma 6.1 to conclude that $h_1(\bar{z}_1, \bar{z}_2) = h_2(\bar{z}_1, \bar{z}_2) = 0$ over $\mathbb{Z}$, we must have the following condition:

$$2^{d(me)/2} \det(\mathcal{L})^{\frac{1}{d(me)-1}} < \frac{N^{me}}{\sqrt{d(me)}}. \tag{18}$$

Plugging (15) into this condition, we obtain $(B_1B_2)^{1/2} < N^{\frac{W(m,e)-me}{D(me)}}/\gamma(me)$, where the factor $\gamma(me) \stackrel{\text{def}}{=} (\sqrt{d(me)}2^{d(me)/2})^{\frac{d(me)-1}{D(me)}}$ is independent of $n$ and so is of order $O(N^{o(1)})$ as $n$ increases.

Furthermore, from (16) and (17) we see that $D(me) = \frac{e^3}{3}m^3 + O(m^2)$ and $W(m,e) = \frac{1}{2}D(me) + \frac{e^2}{6}m^3 + O(m^2)$. For increasing parameter $m$, the leading $m^3$ terms dominate, and hence the ratio $\frac{W(m,e)-me}{D(me)}$ approaches asymptotically the value $\frac{1}{2} + \frac{e^2/6}{e^3/3} = \frac{1}{2} + \frac{1}{2e}$. So the attack success condition becomes $(B_1B_2)^{1/2} < N^{1/2+1/(2e)-o(1)}$ for large $n$ and $m$. Using $B_1 = \frac{N}{2^{r+k-1}}$ and $B_2 = \frac{N}{2^{r+l-1}}$ and $N < 2^n$ we obtain the asymptotic attack success bound

$$\frac{r}{n} > 1/2 - 1/(2e) - \frac{(k+l)}{2n} + o(1). \tag{19}$$

*Remark 1.* The above analysis proves that (when the asymptotic condition (19) is satisfied) the two polynomials $h_1(z_1, z_2)$ and $h_2(z_1, z_2)$ returned by the LLL algorithm will have $(\bar{z}_1, \bar{z}_2)$ as a zero over $\mathbb{Z}$, and hence $\bar{z}_1$ is a zero of the univariate polynomial $h(z_1) = Res_{z_2}(h_1(z_1, z_2), h_2(z_1, z_2))$ (resultant of $h_1$ and $h_2$ with respect to variable $z_2$). If $h(z_1)$ is non-zero, then it has at most $deg(h)$ zeros over $\mathbb{Z}$ which can be easily computed to recover $\bar{z}_1$ (and then $\bar{z}_2$). But it is possible that $h(z_1)$ is the zero polynomial, in which case the attack fails. Hence, in common with several other applications of Coppersmith's method to multivariate modular polynomials [12, 18, 7, 8, 19], this last step of the attack is a heuristic. We have performed several numerical experiments using NTL [38] which have confirmed the validity of this heuristic in practice – in all our experiments, the resultant of $h_1$ and $h_2$ was non-zero whenever the bound (18) was satisfied. The following table gives smallest values of $r/n$ for which our experimental attack succeeded in all 3 successive runs (using independent randomly chosen $N$, $a$,$b$ in each run). Note how we approach the asymptotic bound of (19) as $n$ and $m$ increase.

| $e$ | $n$ | $m$ | $d$ | $r_{exp}/n$ | $r_{bnd}/n$ | $r_{asymp}/n$ | Time(s) |
|---|---|---|---|---|---|---|---|
| 2 | 32 | 3 | 28 | 0.31 | 0.44 | 0.20 | 3 |
| 2 | 64 | 4 | 45 | 0.28 | 0.34 | 0.20 | 230 |
| 2 | 96 | 5 | 66 | 0.26 | 0.30 | 0.20 | 8228 |

Table 2: Attack experimental results (on Pentium 4 1.6GHz processor). In all cases, we set $k = n/16$ and $l = n/32$. Column $r_{exp}/n$ is the smallest value of $r/n$ for which attack succeeded (in all 3 independent runs), $r_{bnd}/n$ is the smallest value of $r/n$ for which the proven success bound (18) is satisfied, and $r_{asymp}/n$ is the asymptotic success lower bound (19) for large $n$ and $m$.

*Remark 2.* We conjecture that the bound (19) of our attack is essentially optimal for 'Coppersmith-type' lattice attacks on $(n, e, r, k, l)$-FSRSA. To give some intuition for this conjecture, we note that the 'linear information' components $(s_1, s_2, u_1, u_2)$ contain at most $2r+k+l$ bits of information about $x$ (using the known $a, b$). To obtain more information on $x$ one must use the degree $e$ polynomial $x^e - y \equiv 0 \pmod{N}$. So the problem is analogous to an SSRSA problem with $n - (2r + k + l)$ unknown bits. Since Coppersmith's original SSRSA algorithm for degree $e$ polynomials succeeds only when the number of unknown bits $s < n/e$, we expect that the problem can be solved only when $n - (2r + k + l) < n/e$, that is $r/n > 1/2 - 1/(2e) - (k + l)/(2n)$, which is the same as the success condition (19) for our attack.

### 6.2 Using Rabin Exponents ($e = 2$)

If we assume that the attack of the previous section is optimal so the $(n, e, r, k, l)$-FSRSA problem is hard when the bound (19) is violated, then we can allow $r/n$ to approach $1/4$ even for $e = 2$, with only one modular squaring required per iteration. It is shown in [20] that with appropriate modifications to the proof, Lemma 3.3 holds also for $e = 2$ if we replace the iteration function $x \to [x^e]_N$ by the 'absolute Rabin function' $f_a : x \to |[x^2]_N| \stackrel{\text{def}}{=} \min([x^2]_N, [N - x^2]_N)$, choose $N = pq$

to be a *Blum* RSA modulus with $p \equiv q \equiv 3 \pmod 4$, and choose the PRG seed $x_0 \in_R M_N$, where $M_N \stackrel{\text{def}}{=} \mathbb{Z}_N^*(+1) \cap (0, N/2)$, and $\mathbb{Z}_N^*(+1)$ denotes the subset of elements of $\mathbb{Z}_N^*$ having Jacobi symbol $+1$. Since $f_a$ permutes the set $M_N$, the proof of Lemma 3.2 holds as well. Refer to Table 1 for performance of this PRG variant, where it is assumed that the best attack on $(n, e, r, k, l)$-FSRSA with $r/n = 1/2 - 1/(2e) - \frac{(k+l)}{2n} + \epsilon$ takes time $\min(T_F(n), 2^{\epsilon n})$, where $T_F(n)$ is the time needed to factor $N$. We stress however that this assumption is new and needs further study.

# 7 Applications

We point out applications of our result on the security of efficient variants of the $(n, e, r, \ell)$-RSAPRG.

**Stream Cipher.** The most direct application is construction of an efficient stream cipher, using the well-known construction in which the ciphertext is obtained by XORing the PRG output with the message bit stream (where the secret key is the seed $x_0$ and prime factors of $N$). It is easy to show that the indistinguishability security of this stream cipher is equivalent to the pseudorandomness security of the PRG, and the computational efficiency of the cipher is also essentially the same as that of the PRG.

**Efficient RSA-Based IND-CPA Public Key Encryption Without Random Oracles.** Another application is the construction of efficient semantically secure (IND-CPA) RSA based public-key encryption schemes without random oracles, an idea which was first proposed (using the less efficient RSA PRG variants with $r = 1$) by Blum and Goldwasser [10]. In this setting, the public encryption key is $(N, e)$ and the secret key is $d = e^{-1} \bmod \phi(N)$, as in standard RSA. To encrypt an $\ell$-bit message $M$ under public key $(N, e)$, one picks a random seed $x_0 \in_R \mathbb{Z}_N$, expands it to a pseudorandom bit string $K = G_N(x_0) \in \{0,1\}^\ell$ using the $(n, e, r, \ell)$-RSAPRG, and computes $C = M \oplus K$ (as in the stream cipher construction above). The ciphertext for $M$ is $(C, x_m)$, where $m = \ell/r$ (here $x_i$ is the PRG state value after the $i$th iteration). Since $x_m = [x_0^{e^m}]_N$, the decryptor knowing the RSA trapdoor key $d$ can easily recover $x_0 = [x_m^{d^m}]_N$, thus obtaining $K = G_N(x_0)$ and then $M = C \oplus K$.

The IND-CPA security of the above scheme for $r/n = 1/2 - 1/e - \epsilon - o(1)$ follows from the hardness of the same $(1/e + \epsilon, e)$-SSRSA inversion problem which suffices for the pseudorandomness of $(n, e, r, \ell)$-RSAPRG. This is due to the fact (first exploited by Blum and Goldwasser [10] for the case $r = 1$) that the pseudorandomness security reduction for the $(n, e, r, \ell)$-RSAPRG, and in particular Lemma 3.2, easily extends to the case where the state $x_m$ is also known to the distinguisher (referring to the proof of Lemma 3.2 in Appendix B, this is simply because the distinguisher $\mathsf{D}'$ knows $(N, e)$ and $y = [x^e]_N = x_{i^*+1}$ for some $i^* \in \{0, \ldots, m-1\}$ and hence can easily compute the additional input $x_m = [y^{e^{m-1-i^*}}]_N$ to be given to distinguisher $\mathsf{D}$ in the reduction).

For short messages of length $r = (1/2 - 1/e - \epsilon - o(1))n$ ($m = 1$), the computational efficiency of the above scheme is about the same as other RSA based IND-CPA schemes using random oracles (e.g. OAEP [2]), with the advantage is that it achieves provable IND-CPA security based on a well-studied inversion problem without random oracles. Furthermore, the scheme can encrypt long $\ell$-bit messages with the same efficiency as the $(n, e, r, \ell)$-RSAPRG without using additional security assumptions. However, the scheme is not secure against chosen-ciphertext attacks, and it is an open problem to efficiently strengthen it for this security level without using random oracles.

We also remark that Goldreich describes a 'Randomized RSA' scheme (Construction 5.3.16 on page 416 in [23]) which is identical with the above scheme using parameters $m = \ell/r = 1$ and $r/n = 1/2$, and proves its security assuming the 'Large Hard Core Conjecture for RSA' (informally, this conjecture states that distinguishing the $r = n/2$ LS bits of $x \in_R \mathbb{Z}_N$ from $n/2$ independent random bits given $(N, e, y = [x^e]_N)$ is as hard as inverting the RSA function). Goldreich states([23], page 481) that 'Randomized RSA' is commonly believed to be secure, but leaves as an important open problem to find additional support for this belief. Our result makes progress in this direction, by

showing that the variant of 'Randomized RSA' with $r = (1/2 - 1/e - \epsilon - o(1))n$ is secure assuming the hardness of a well-studied $(1/e + \epsilon, e)$-SSRSA inversion problem.

# 8 Conclusion

We have shown that an efficient variant of the RSA PRG is provably secure assuming the hardness of a well-studied variant of the RSA inversion problem in which some of the plaintext bits are known. We see two avenues for further improvement. Even using the FSRSA assumption in Section 6, the PRG rate which we can prove secure is $r = (1/2 - 1/(2e) - \epsilon - o(1))n$ for 'small' $\epsilon$. Can this rate be improved using a different proof (but a similar inversion assumption) up to $r = (1 - 1/e - \epsilon - o(1))n$? The other question is whether the factor $\ell^2$ in the reduction run-time factor $O((\ell/\delta)^2 n \log(n))$ can be significantly reduced, to allow more bits $\ell$ to be generated for a given security level and modulus length $n$ (a result in [20] shows that that the factor $O((1/\delta)^2 n)$ is unlikely to be improved, since it is optimal for any inversion algorithm that does not use the non-linear information $y = [x^e]_N$ in processing the distinguisher answers for recovering $x$).

Finally, an interesting open problem is to construct additional efficient cryptographic primitives based on the SSRSA problem.

**Acknowledgements.** The authors would like to thank Scott Contini and Igor Shparlinski for enlightening discussions and encouragement and the anonymous referees for their useful comments. This work was supported by Australian Research Council Discovery Grants DP0663452, DP0451484 and DP0665035.

# References

[1] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr. RSA and Rabin Functions: Certain Parts Are as Hard as the Whole. *SIAM Journal on Computing*, 17(2):194–209, 1988.

[2] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In *EUROCRYPT '94*, volume 950 of *LNCS*, pages 92–111, Berlin, 1995. Springer-Verlag.

[3] M. Ben-Or, B. Chor, and A. Shamir. On the Cryptographic Security of Single RSA Bits. In *Proc. 15-th STOC*, pages 421–430, New York, 1983. ACM Press.

[4] C. Berbain, H. Gilbert, and J. Patarin. QUAD: a Practical Stream Cipher with Provable Security. In *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 109–128, Berlin, 2006. Springer-Verlag.

[5] S.R. Blackburn, D. Gomez-Perez, J. Gutierrez, and I.E. Shparlinski. Reconstructing Noisy Polynomial Evaluation in Residue Rings. *Journal of Algorithms*. (To Appear).

[6] S.R. Blackburn, D. Gomez-Perez, J. Gutierrez, and I.E. Shparlinski. Predicting Nonlinear Pseudorandom Number Generators. *Mathematics of Computation*, 74:1471–1494, 2004.

[7] J. Blömer and A. May. Low Secret Exponent RSA Revisited. In *CaLC 2001*, volume 2146 of *LNCS*, pages 110–125, Berlin, 2001. Springer-Verlag.

[8] J. Blömer and A. May. New Partial Key Exposure Attacks on RSA. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 27–43, Berlin, 2003. Springer-Verlag.

[9] L. Blum, M. Blum, and M. Shub. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing*, 15:364–383, 1986.

[10] M. Blum and S. Goldwasser. An Efficient Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information. In *CRYPTO '84*, volume 196 of *LNCS*, pages 289–302, Berlin, 1985. Springer-Verlag.

[11] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[12] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Trans. on Info. Theory*, 46(4):1339–1349, 2000.

[13] D. Boneh, S. Halevi, and N.A. Howgrave-Graham. The Modular Inversion Hidden Number Problem. In *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 36–51, Berlin, 2001. Springer-Verlag.

[14] D. Catalano, R. Gennaro, N. Howgrave-Graham, and P. Nguyen. Paillier's Cryptosystem Revisited. In *Proc. CCS '01*, New York, November 2001. ACM.

[15] D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *J. of Cryptology*, 10:233–260, 1997.

[16] D. Coppersmith. Finding Small Solutions to Low Degree Polynomials. In *CALC '01*, volume 2146 of *LNCS*, pages 20–31, Berlin, 2001. Springer-Verlag.

[17] W. Dai. *Crypto++ 5.2.1 Benchmarks*, 2006. `http://www.eskimo.com/~weidai/benchmarks.html`.

[18] G. Durfee and P.Q. Nguyen. Cryptanalysis of the RSA Schemes with Short Secret Exponent from Asiacrypt '99. In *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 14–29, Berlin, 2000. Springer-Verlag.

[19] M. Ernst, E. Jochemsz, A. May, and B. de Weger. Partial Key Exposure Attacks on RSA Up to Full Size Exponents. In *CRYPTO 2005*, volume 3494 of *LNCS*, pages 371–386, Berlin, 2005. Springer-Verlag.

[20] R. Fischlin and C.P. Schnorr. Stronger Security Proofs for RSA and Rabin Bits. *Journal of Cryptology*, 13:221–244, 2000.

[21] R. Gennaro. An Improved Pseudo-Random Generator Based on the Discrete-Logarithm Problem. *Journal of Cryptology*, 18:91–110, 2005.

[22] O. Goldreich. *Foundations of Cryptography, Volume I.* Cambridge University Press, Cambridge, 2003.

[23] O. Goldreich. *Foundations of Cryptography, Volume II.* Cambridge University Press, Cambridge, 2004.

[24] O. Goldreich and V. Rosen. On the Security of Modular Exponentiation with Application to the Construction of Pseudorandom Generators. *J. of Cryptology*, 16:71–93, 2003.

[25] S. Goldwasser and S. Micali. Probabilistic Encryption. *J. of Computer and System Sciences*, 28(2):270–299, 1984.

[26] S. Goldwasser, S. Micali, and P. Tong. Why and How to Establish a Private Code on a Public Network. In *Proc. FOCS '82*, pages 134–144. IEEE Computer Society Press, 1982.

[27] J. Håstad, A. Schrift, and A. Shamir. The Discrete Logarithm Modulo a Composite Hides $O(n)$ Bits. *J. Comp. and Syst. Sci.*, 47:376–404, 1993.

[28] N. Howgrave-Graham. Finding Small Roots of Univariate Polynomials Revisited. In *Cryptography and Coding*, volume 1355 of *LNCS*, pages 131–142, Berlin, 1997. Springer-Verlag.

[29] R. Impagliazzo and M. Naor. Efficient Cryptographic Schemes Provably as Secure as Subset Sum. *Journal of Cryptology*, 9:199–216, 1996.

[30] D. Knuth. *Seminumerical Algorithms.* Addison-Wesley, Reading, MA, 1997.

[31] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[32] A.K. Lenstra. Unbelievable Security: Matching AES Security Using Public Key Systems. In *Asiacrypt 2001*, volume 2248 of *LNCS*, pages 67–86, Berlin, 2001. Springer-Verlag.

[33] A.K. Lenstra and E.R. Verheul. Selecting Cryptographic Key Sizes. *J. of Cryptology*, 14:255–293, 2001.

[34] S. Micali and C.P. Schnorr. Efficient, Perfect Polynomial Random Number Generators. *J. of Cryptology*, 3:157–172, 1991.

[35] P. Q. Nguyen and I. E. Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *J. Cryptology*, 15:151–176, 2002.

[36] P. Q. Nguyen and J. Stern. The Two Faces of Lattices in Cryptology. In *Cryptography and Lattices*, volume 2146 of *LNCS*, pages 146–180, Berlin, 2001. Springer-Verlag.

[37] S. Patel and G. Sundaram. An Efficient Discrete Log Pseudo Random Generator. In *CRYPTO '98*, volume 1462 of *LNCS*, pages 304–317, Berlin, 1998. Springer-Verlag.

[38] V. Shoup. *NTL: A library for doing number theory (version 5.3)*, 2003. Available from `http://shoup.net/ntl/`.

[39] A. Sidorenko and B. Schoenmakers. Concrete Security of the Blum-Blum-Shub Pseudorandom Generator. In *Cryptography and Coding 2005*, volume 3796 of *LNCS*, pages 355–375, Berlin, 2005. Springer-Verlag.

[40] U.V. Vazirani and V.V. Vazirani. Efficient and Secure Pseudo-Random Number Generation. In *Proc. FOCS '84*, pages 458–463. IEEE Computer Society Press, 1982.

# A Proof of Lemma 3.1

Let $p_0 = \Pr[\mathsf{D}(s) = 1 : s \in_R \{0,1\}^\ell]$ and for each $N \in I_n$, let $p_N = \Pr[\mathsf{D}(G_N(x_0)) = 1 : x_0 \in_R \mathbb{Z}_N]$. Then since $N$ is uniform in $I_n$ we have:

$$\delta \leq \mathsf{Adv}(\mathsf{D}) = \left| \sum_{N \in \mathcal{I}_n} \frac{1}{|\mathcal{I}_n|}(p_N - p_0) \right| \leq \frac{1}{|\mathcal{I}_n|} \sum_{N \in \mathcal{I}_n} |p_N - p_0| = \frac{1}{|\mathcal{I}_n|} \sum_{N \in \mathcal{I}_n} \mathsf{Adv}_N(\mathsf{D}). \tag{20}$$

Suppose towards a contradiction, that the set $\mathcal{G}_n$ of of all 'good' $N \in \mathcal{I}_n$ for which $\mathsf{Adv}_N(\mathsf{D}) \geq \delta/2$ was of size less than $\delta/2|\mathcal{I}_n|$. Then (because $|\mathcal{G}_n| < \delta/2|\mathcal{I}_n|$) the $N$'s in $\mathcal{G}_n$ contribute less than $\delta/2$ to the average on the right hand side of (20), while (because $\mathsf{Adv}_N(\mathsf{D}) < \delta/2$ for $N \in \mathcal{I}_n \setminus \mathcal{G}_n$) the $N$'s in $\mathcal{I}_n \setminus \mathcal{G}_n$ also contribute less than $\delta/2$, so the right hand side of (20) would be less than $\delta$, contradicting $\mathsf{Adv}(\mathsf{D}) \geq \delta$, as required.

# B Proof of Lemma 3.2

We use the following 'hybrid' argument. Consider the $m+1$ 'hybrid' distributions $\mathcal{D}_{H,0,N}, \ldots, \mathcal{D}_{H,m,N}$ where $\mathcal{D}_{H,i,N}$ has its first $i$ $r$-bit blocks random while the remaining $m-i$ pseudorandom. That is:

$$\begin{aligned}\mathcal{D}_{H,i,N} = \{s = (s_0, \ldots, s_{m-1}) : x_0 \in_R \mathbb{Z}_N; x_{j+1} = [x_j^e]_N, j = 0, \ldots, m-2;\\ s_j \in_R \{0,1\}^r, j = 0, \ldots, i-1, s_j = L_r(x_j), j = i, \ldots, m-1\}.\end{aligned}$$

Note that $\mathcal{D}_{H,0,N} = \mathcal{D}_{P,\ell,N}$ while $\mathcal{D}_{H,m,N} = \mathcal{D}_{R,\ell}$. For $i \in \{0, \ldots, m-1\}$, let $p_i = \Pr_{s \leftarrow \mathcal{D}_{H,i,N}}[\mathsf{D}(s) = 1]$. Note that the advantage of $\mathsf{D}$ can be written $|p_\ell - p_0| = |\sum_{i=0}^{m-1}(p_{i+1} - p_i)| \geq \delta$, so we have $\sum_{i=0}^{m-1} |(p_{i+1} - p_i)| \geq |\sum_{i=0}^{m-1}(p_{i+1} - p_i)| \geq \delta$. Hence there must exist $i^* \in \{0, \ldots, m-1\}$ such that $\mathsf{D}$ has distinguishing advantage at least $\delta/m$ between the neighbouring hybrid distributions $\mathcal{D}_{H,i^*,N}$ and $\mathcal{D}_{H,i^*+1,N}$. So on input $(y, s)$, the distinguisher $\mathsf{D}'$ runs $\mathsf{D}$ on input $(s_0, \ldots, s_{i^*-1}, s, s_{i^*+1}, \ldots, s_{m-1})$, where $(s_0, \ldots, s_{i^*-1})$ are chosen independently at random from $\{0,1\}^r$ while $s_j = L_r(x_j)$ for $j \geq i^*+1$, where $x_{i^*+1} = y$ and $x_j = [x_{j-1}^e]_N$ for $j > i^* + 1$. Thus it is easy to see that if $(y, s)$ comes from distribution $\mathcal{D}'_{P,r,N}$ then $\mathsf{D}$ sees the distribution $\mathcal{D}_{H,i^*,N}$ while if $(y, s)$ comes from $\mathcal{D}'_{R,r,N}$ then $\mathsf{D}$ sees the distribution $\mathcal{D}_{H,i^*+1,N}$ (we use here the fact that the mapping $x \to [x^e]_N$ permutes $\mathbb{Z}_N$ so $x_{i^*}$ is uniform in $\mathbb{Z}_N$ in the distributions $\mathcal{D}_{H,i,N}$). Note that $\mathsf{D}'$ performs at most $m$ evaluations of the mapping $x \to [x^e]_N$, and each of those takes time $O(\log(e)n^2)$. Hence $\mathsf{D}'$ is a $(T + O(m \log(e)n^2), \delta/m)$ distinguisher between $\mathcal{D}'_{P,r,N}$ and $\mathcal{D}'_{R,r,N}$, as required.

# C Proof of Lemma 3.3

For completeness we sketch the main ideas in the proof of Lemma 3.3 from [20]. For more details we refer the reader to [20] (the proof of simultaneous security of $r$ bits we give here is outlined on page 229 of [20] as an extension of the proof for $r = 1$).

The lemma is proved in two steps.

The first step is to convert the given $(T, \delta)$ distinguisher $\mathsf{D}'$ between distributions $\mathcal{D}'_{P,r,N} = \{(y = [x^e]_N, s = L_r(x)) : x \in_R \mathbb{Z}_N\}$ and $\mathcal{D}'_{R,r,N} = \{(y = [x^e]_N, s) : x \in_R \mathbb{Z}_N; s \in_R \{0,1\}^r\}$ to a $j$th bit predictor $O_j$ for some $j \in \{1, \ldots, r\}$. That is, letting $\ell_j(x)$ denote the $j$th LS bit of $x$ we have $O_j([x^e]_N, L_{j-1}(x)) = \ell_j(x)$ with probability at least $1/2 + \delta/r$ over $x \in_R \mathbb{Z}_N$ and the random coins of $O_j$ (notice that $O_j$ requires as input $[x^e]_N$ as well as the $j-1$ LS bits of $x$). This general conversion from distinguisher to predictor is originally due to Yao (see Lemma P1 in [30]). The idea is to first consider the $r$ 'hybrid' distributions $\mathcal{D}'_{H,0}, \ldots, \mathcal{D}'_{H,r}$ where $\mathcal{D}'_{H,j}$ is defined like $\mathcal{D}'_{P,r,N}$ except that the string $s$ has its first $j$ LS bits equal to $L_j(x)$ while the remaining $r-j$ bits random and independent of $x$. Since $\mathcal{D}'_{H,0} = \mathcal{D}'_{R,r,N}$ while $\mathcal{D}'_{H,r} = \mathcal{D}'_{P,r,N}$, it then follows (similar to the proof of Lemma 3.2) that $\mathsf{D}'$ has distinguishing advantage least $\delta/r$ between two neighbouring hybrid distributions $\mathcal{D}'_{H,j-1}$ and $\mathcal{D}'_{H,j}$ differing only in the $j$th bit for some $j \in \{1, \ldots, r\}$. The $j$th bit predictor $O_j$, on input $(y = [x^e]_N, L_{j-1}(x))$, chooses a random bit $b$ and $r-j$ other random bits $s_{j+1}, \ldots, s_r$ and runs $\mathsf{D}'$ on input $(y, s = (L_{j-1}(x), b, s_{j+1}, \ldots, s_r))$. If $\mathsf{D}'$ returns 1, then $O_j$ returns $b$, else $O_j$ returns $\bar{b}$ (the NOT of $b$). It is not difficult to show that the output bit of $O_j$ (or its NOT) equals $\ell_j(x)$ with probability at least $1/2 + \delta/r$.

The second step is to convert the $j$th bit predictor $O_j$ to an inversion algorithm to recover $x$ from $y = [x^e]_N$ together with the 'extra information' $(a \in_R \mathbb{Z}_N, s_1 = L_r([ax]_N), u_1 = \widehat{M}_{N,k}([ax]_N), b \in_R \mathbb{Z}_N, s_2 = L_r([bx]_N), u_2 = \widehat{M}_{N,l}([bx]_N))$.

The basic idea used in [20] is the 'binary division' process: Given an estimate $u$ for $[ax]_N$ with additive error $\epsilon$ (i.e. $x$ is in interval $[u-\epsilon, u+\epsilon]$), and the least significant bit $b = \ell_1([ax]_N)$ of $[ax]_N$, we can obtain an estimate $u'$ for $x' = [2^{-1}ax]_N$ within additive error $\epsilon/2$ (i.e. half the error) using $u' = 1/2(u + bN)$. This is because if $b = 0$ ($[ax]_N$ is even), then $[2^{-1}ax]_N = 2^{-1}[ax]_N$, while if $b = 1$ ($[ax]_N + N$ is even) then $[2^{-1}ax]_N = 2^{-1}([ax]_N + N)$. Thus, starting from a rough approximation $u_1$ of $[ax]_N$ with additive error of order $N \approx 2^n$, if we are given the least significant bits of $[2^{-t}ax]_N$ for $t = 0, 1, 2, 3..., n$ we apply the binary division process $n$ times to obtain an approximation $u_n$ to $[2^{-n}ax]_N$ with error less than $1/2$; hence rounding $u_n$ to the nearest integer recovers $[2^{-n}ax]_N$ exactly and then $x$ can be easily recovered since $a$ is known.

We now explain how to recover the LS bits of $[2^{-t}ax]_N$ for $t = 0, \ldots, n$ using the $j$th bit predictor $O_j$.

Suppose first that $O_j$ was a perfect bit predictor with success probability 1. For the case $j = 1$, to obtain the LS bit of $[2^{-t}ax]_N$ we use the known $y = [x^e]_N$ and the homomorphic property of RSA to compute $y_t = [([2^{-t}ax]_N)^e]_N = [(2^{-t}a)^e[x^e]_N]_N = [(2^{-t}a)^e y]_N$ and query $y_t$ to $O_1$, which returns the desired bit. For the case $j > 1$, notice that since $j \leq r$, the 'extra information' $s_1 = L_r([ax]_N)$ gives us $L_j([ax]_N)$. Now we use the relation $L_{j-1}([2^{-1}ax]_N) = 2^{-1}L_j(L_j([ax]_N) + \ell_1([ax]_N)N)$ to compute $L_{j-1}([2^{-1}ax]_N)$, which we provide as input to $O_j$ along with $y_1 = [([2^{-1}ax]_N)^e]_N$ to obtain $j$th bit $\ell_j([2^{-1}ax]_N)$. Combining this with $L_{j-1}([2^{-1}ax]_N)$ we obtain $L_j([2^{-1}ax]_N)$. Now we continue this process to obtain $L_j([2^{-t}ax]_N)$ for $t = 0, \ldots, n$ and hence recover $x$ as before.

To handle the case of imperfect predictors $O_j$ with success probability at least $1/2 + \delta/r$, we query the predictor $O_j$ in stage $t$ (when attempting to recover bit $\ell_j([2^{-t}ax]_N)$) many times on pairwise independent inputs and use a majority vote over the estimates of the bit $\ell_j([2^{-t}ax]_N)$ obtained from the outputs of $O_j$ on those inputs. Thanks to the pairwise independence of the queries to $O_j$, it is possible to prove using the Chebychev inequality that the error probability of the majority vote bit is reduced to $O(1/n)$ using $O(n(\delta/r)^{-2})$ queries. Further details follow. To make the queries pairwise independence, we use queries of the form $([(c_{t,i}x)^e]_N, L_{j-1}([c_{t,i}x]_N))$ where $c_{t,i} = (2i+1)[2^{-t}a]_N + b$, over values of $i$ in set $\{i : |2i+1| \leq m_t\}$, where $m_t = O(n(\delta/r)^{-2})$. The pairwise independence of the $c_{t,i}$'s follows from the pairwise independence of $a, b$ in $\mathbb{Z}_N$. To compute an estimate for the $j-1$ LS bits $L_{j-1}([c_{t,i}x]_N)$ needed as input to $O_j$ (and also to convert the output bit of $O_j$ for the query $([c_{t,i}x]_N, L_{j-1}([c_{t,i}x]_N))$, which estimates the bit $\ell_j([c_{t,i}x]_N)$ to a prediction for bit $\ell_j([2^{-t}ax]_N)$), we

use the fact that

$$[c_{t,i}x]_N = (1+2i)[2^{-t}ax]_N + [bx]_N - \omega_{t,i}N, \tag{21}$$

for some integer $\omega_{t,i}$. Since $L_{j-1}([2^{-t}ax]_N)$ is known from the previous stage (or initially, for $t=0$, from the given $L_j([ax]_N)$) and $L_j([bx]_N)$ is known from the given input, to compute $L_{j-1}([c_{t,i}x]_N)$ it is enough to know $\omega_{t,i}$. From (21), $\omega_{t,i} = \lfloor \frac{(1+2i)[2^{-t}ax]_N + [bx]_N}{N} \rfloor$, and hence an estimate for $\omega_{t,i}$ which is correct with high probability can be obtained by replacing in the last equation $[2^{-t}ax]_N$ and $[bx]_N$ by their approximations (which are also given for $t=0$ via $u_1$ and $u_2$ as input; the approximation of $[2^{-t}ax]_N$ for $t>0$ is even better due to the binary division process). Hence the answer to our $i$th query to $O_j$ gives an estimate for $\ell_j([c_{t,i}x]_N)$ which is correct if our estimate for $\omega_{t,i}$ is correct (so our query was really $([(c_{t,i}x)^e]_N, L_{j-1}([c_{t,i}x]_N))$) and $O_j$ was successful (we refer the reader to [20] for the details of the error probability analysis). To convert this estimate for $\ell_j([c_{t,i}x]_N)$ to an estimate for the desired bit $\ell_j([2^{-t}ax]_N)$ we reduce the equation (21) modulo $2^j$ to obtain

$$L_j([c_{t,i}x]_N) \equiv L_j(1+2i)L_j([2^{-t}ax]_N) + L_j([bx]_N) - L_j(\omega_{t,i}N) \pmod{2^j}.$$

Substituting $L_j([2^{-t}ax]_N) = L_{j-1}([2^{-t}ax]_N) + 2^{j-1}\ell_j([2^{-t}ax]_N)$ and $L_j([c_{t,i}x]_N) = L_{j-1}([c_{t,i}x]_N) + 2^{j-1}\ell_j([c_{t,i}x]_N)$, dividing by $2^{j-1}$ and observing that $L_j(1+2i) \equiv 1 \pmod 2$, we obtain a relation for $\ell_j([2^{-t}ax]_N)$ in terms of $\ell_j([c_{t,i}x]_N)$ and other known quantities, namely $\ell_j([2^{-t}ax]_N) \equiv \ell_j([c_{t,i}x]_N) + \frac{1}{2^{j-1}}(L_{j-1}([c_{t,i}x]_N) - L_j(1+2i)L_{j-1}([2^{-t}ax]_N) - L_j([bx]_N) + L_j(\omega_{t,i}N)) \pmod 2$.

Below we summarize the full inversion algorithm from [20]:

1. Using the 'extra information' $u_1, u_2$, compute estimates $\bar{u}_0 = u_1/N$ and $\bar{v} = u_2/N$ for $[ax]_N/N$ and $[bx]_N/N$ with additive errors $(\delta/r)^3/16$ and $(\delta/r)/16$ respectively. Let $\Sigma_0 = L_j(s_1) = L_j([ax]_N)$, and $\Theta = L_j(s_2) = L_j([bx]_N)$.

2. **Binary Division.** For each $t = 1, \ldots, n$ ($n = \log N$):

   - Compute an improved estimate $\bar{u}_t = 1/2(\bar{u}_{t-1} + \sigma_{t-1})$ (with half error magnitude) for ratio $[2^{-t}ax]_N/N$,where $\sigma_{t-1} = L_1(\Sigma_{t-1})$.
   - If $t \le \log(n)+3$, set $m_t = m'_t = 2^t(r/\delta)^2$ and define integer set $A = \{i : |2i+1| \le m_t\}$. Else if $t \ge \log(n)+4$, set $m_t = 16n(r/\delta)^2$, $m'_t = 2\log(n)(r/\delta)^2$, and define multiset $A$ by choosing $m'_t$ independent uniformly random elements $i$ from set $\{i : |2i+1| \le m_t\}$.
   - Compute estimate $S_t = \frac{1}{2} \cdot L_j(\Sigma_{t-1} + L_1(\Sigma_{t-1})N)$ for $L_{j-1}([2^{-t}ax]_N)$ (using estimate $\Sigma_{t-1}$ for $L_j([2^{-(t-1)}ax]_N)$).
   - Obtain $m'_t$ pairwise independent estimates $\{\sigma_{t,i}\}_{i\in A}$ for bit $\ell_j([2^{-t}ax]_N)$ by querying $O_j$ predictor $m'_t$ times. Namely, for each $i \in A$ do following:
     - Compute multiplier $c_{t,i} = (1+2i)[2^{-t}a]_N + b$, $[(c_{t,i}x)^e]_N = [c_{t,i}^e y]_N$, and an estimate $\widehat{\omega}_{t,i} = \lfloor \bar{u}_t(1+2i) + v \rfloor$ for $\omega_{t,i} = \lfloor \frac{(2i+1)[2^{-t}ax]_N + [bx]_N}{N} \rfloor$.
     - Compute estimate $C_{t,i} = L_{j-1}((2i+1)S_t + L_{j-1}(\Theta) - L_{j-1}(\widehat{\omega}_{t,i}N))$ for $L_{j-1}([c_{t,i}x]_N)$ (based on the relation $L_{j-1}([c_{t,i}x]_N) = L_{j-1}((2i+1)[2^{-t}ax]_N + [bx]_N - \omega_{t,i}N)$).
     - Run $O_j$ on input $(N, [(c_{t,i}x)^e]_N, C_{t,i})$ to obtain output bit $\widehat{\sigma}_{t,i}$ (an estimate for $\ell_j([c_{t,i}x]_N)$).
     - Compute estimate $\sigma_{t,i} = \widehat{\sigma}_{t,i} + \frac{1}{2^{j-1}}(C_{t,i} - (2i+1)S_t - \Theta + \widehat{\omega}_{t,i}N) \bmod 2$ for $\ell_j([2^{-t}ax]_N)$ (based on the relation $[c_{t,i}x]_N = (2i+1)[2^{-t}ax]_N + [bx]_N - \omega_{t,i}N$).
   - Compute $\sigma_t$ as the majority value over the estimates $\{\sigma_{t,i}\}_{i\in A}$ (estimate for bit $\ell_j([2^{-t}ax]_N)$).
   - Compute estimate $\Sigma_t = S_t + \sigma_t 2^{j-1}$ for $L_j([2^{-t}ax]_N)$.

4 Now $\bar{u}_n N$ rounded to nearest integer is equal to $[2^{-n}ax]_N$ so $x$ can be recovered efficiently as follows: $x = [2^n a^{-1}\lfloor \bar{u}_n N + \frac{1}{2} \rfloor]_N$.

**The Success Probability.** It is shown in [20] (pages 227-228 and 231) that with the choice of parameters above (the 'SMAJ' version on page 231), the algorithm succeeds with probability at least $2/9$ for all $n \geq 2^9$ (we remark that although the success probability analysis in [20] is carried out for the $j = 1$ case, it extends without modification to the arbitrary $j$ case (see [20], page 229), since the errors still occur only due to incorrect $O_j$ answers or $\omega_{t,i}$ estimation errors, as explained above).

**The Run-Time of the Reduction.** The number of times that $\mathsf{A}$ runs the predictor $O_j$ is $\sum_{t=1}^{n} m'_t = (\sum_{t=1}^{\log(n)+3} 2^t + \sum_{t=\log(n)+4}^{n} 2\log(n)) \cdot (r/\delta)^2 \leq (2n\log(n) + 16n) \cdot (r/\delta)^2 \leq 4n\log(n)(r/\delta)^2$ for all $n \geq 2^9$, and for each run of $O_j$, $\mathsf{A}$ performs a constant number of additions and multiplications/divisions on numbers of length $O(n)$ bits, which take time $O(n^2)$. Hence the run-time of $\mathsf{A}$ is at most $4n\log(n)(r/\delta)^2(T + O(n^2))$, as claimed.