Ten Years of RSA Cheating Cryptosystems

Jihoon Cho

jhcho@math.uwaterloo.ca

Dept. of Combinatorics and Optimization
University of Waterloo

Abstract

RSA cheating cryptosystems were first introduced by Anderson and developed by Young and Yung with their SETUP mechanism. Recently, Crépeau and Slakmon suggested very simple backdoors for this purpose. This paper sums up these methods. We also describe the LLL-reduction algorithm, which is useful to prove that Anderson's trapdoor is insecure and to factor integers where the high bits of their prime factors are known.

1 Introduction

Cryptographic devices might be considered black boxes in the sense that the users trust internal designs and they usually have no access to check the authenticity and integrity of the software. At the same time, the manufacturers are reluctant to publish the source code in order to protect their intellectual property. Under this black-box cryptosystem environment, can we trust the algorithms inside cryptographic devices?

Actually, it is possible to construct cryptographic systems that leak information so that the manufacturer may attack the users by recovering their secret information from the output of cryptographic systems. Such a cryptographic system is called a *Secretly Embedded Trapdoor with Universal Protection* (*SETUP*) *mechanism*. One requirement is that the manufacturer should make this cheating software in such a way that the presence of this

setup mechanism is unnoticeable by the users as well as others and it should be hard to reverse-engineer by a third party. That is, the setup mechanism should be designed to give an advantage only to the manufacturer. There have been suggested several setups in RSA, ElGamal, DSA, and Kerberos. In this paper, we focus our attention on setups for RSA.

Anderson [1] proposed a cheating RSA-key generation algorithm by embedding a trapdoor in the setup, but it was proven to be insecure using the lattice basis reduction algorithm by Kaliski [5]. According to PAPs, setup mechanisms of Young and Yung [9] [10], the manufacturer designs the setup to leak secret information in the representation of the public RSA modulus n and these schemes work for any public exponent e. The attackers use either their own RSA private/public key pairs or ones of ElGamal encryption scheme. However, this PAP not only contains a critical deficiency but also does their running time not compare well with the standard RSA-key generation algorithm. Later, an analogous system to PAPs that is more simple and secure was put forward by Crépeau and Slakmon [3]. They also presented extremely simple RSA setups which have nearly the same running time as the original. In contrast with PAPs, these three schemes generate genuinely a random p and q and embed a simple backdoor in the representation of a public exponent e.

2 Anderson's SETUP Algorithm

2.1 Anderson's RSA trapdoor

Anderson [1] suggested a cheating RSA public-key generator whereby the designer can easily factor the public modulus, while it is supposed to be difficult for other parties to find this trapdoor and factor the modulus. Assuming we use 256 bits primes, he proposed the following algorithm.

Algorithm 1. Anderson's trapdoor in RSA-key generator

1. choose a 200-bit prime number A to be a secret.
2. choose a random number between 0 and 2^{100} and find the next prime q.
3. repeat step 2 until $p = r(A, q)A + q$ is prime.
4. generate another prime number p' with the same process as steps 2 and 3.
5. set $n \leftarrow pp'$.
6. we do exactly the same procedure as RSA-key generator after this.

In this algorithm, the $r(A, q)$ is 56 bits long and q is a prime number satisfying $q < \sqrt{A}$. Since A and q are relative prime, we can find such a prime p due to the following.

Theorem 1. Dirichlet's theorem
For any two positive coprime integers A and B, there are infinitely many primes of the form $A + nB$, where $n > 0$.

In other words, we can say there are infinitely many primes which are congruent to A modulo B. If we suppose users implement this cheating RSA-key generator, the designer of the hardware can recover the user's secret key by factoring the public RSA modulus n as follows.

$$\begin{aligned} n &= pp' \\ &= (rA + q)(r'A + q') \\ &= rr'A^2 + (rq' + r'q)A + qq'. \end{aligned}$$

The designer reduces the modulus n modulo A to recover qq', which is 200 bits long, so qq' can be easily factored. Having found q and q', the designer can recover r and r', and hence p and p'.

2.2 Breaking Anderson's Trapdoor

In fact, the secret information can be detected using LLL-reduction algorithm[1] [5]. From now on, we describe how to find the trapdoor. Let $p_0, p_1, \ldots, p_k$ be the trapped primes and $r_0, r_1, \ldots, r_k$ be the corresponding parameters (*i.e.* $p_i = r_i A + q_i$ for $i = 0, 1, \ldots, k$). Then, we have

$$\left| r_0 \frac{p_i}{p_0} - r_i \right| < \frac{2}{\sqrt{A}}, \tag{1}$$

for $i = 1, 2, \ldots, k$. These inequalities are called *simultaneous Diophantine Equation* [7], and they are classified as *unusually good* if $2/\sqrt{A} < p_0^{-1/k}$, which is the case when $k \geq 3$. The simultaneous Diophantine approximation problem of finding $r_0, r_1, \ldots, r_k$ is easily set up as a problem of finding a short vector in a lattice L. Actually, there is no known polynomial time

[1]The appendix includes an introduction to lattices and details of the LLL-reduction algorithm

algorithm for finding the shortest vector in a lattice, although this problem is not proven to be NP-hard. However, there are polynomial time algorithms to find relatively short vectors in a lattice.

Suppose we are looking for a denominator r_0 which produces an approximation vector

$$\xi = \left(\frac{r_1}{r_0}, \frac{r_2}{r_0}, \ldots, \frac{r_k}{r_0}\right) \text{ to } \alpha = \left(\frac{p_1}{p_0}, \frac{p_2}{p_0}, \ldots, \frac{p_k}{p_0}\right)$$

and satisfying the condition (1). Then we consider a lattice $L \subset \mathbb{Z}^{k+1}$ with a basis

$$\begin{aligned} b_1 &= (\lambda p_0,\ 0,\ 0, \ldots,\ 0) \\ b_2 &= (0,\ \lambda p_0,\ 0, \ldots,\ 0) \\ b_3 &= (0,\ 0,\ \lambda p_0, \ldots,\ 0) \\ &\cdots\cdots \\ b_k &= (0,\ 0,\ 0, \ldots,\ \lambda p_0,\ 0) \\ b_{k+1} &= (-\lambda p_1, -\lambda p_2, \ldots, -\lambda p_k, 1), \end{aligned}$$

where λ is a scaling factor. If α has an approximation vector ξ, then the vector

$$\begin{aligned} v &= r_1 b_1 + r_2 b_2 + \cdots + r_k b_k + r_0 b_{k+1} \\ &= (\lambda(r_1 p_0 - r_0 p_1), \ldots, \lambda(r_k p_0 - r_0 p_k), r_0) \end{aligned}$$

has length

$$\|v\| = \sqrt{k(\lambda^2 p_0^2)(p_0^{-1/k})^2 + r_0^2},$$

where we choose λ in such a way not to make the r_0^2 term too big. Since this vector v is quite short compared to the original basis vectors, we hope that the vector v will show up in the reduced basis for the lattice L obtained by the LLL-reduction algorithm. We note that the desired denominator r_0 of ξ is directly recoverable as the last coordinate of v. After finding r_0, we can easily determine $r_1, r_2, \ldots, r_k$. Then, we choose small prime numbers $t_1, t_2, \ldots, t_l$ such that $\prod_{j=1}^{l} t_j > A$. Assuming $\gcd(r_i, t_j) = 1$, we know the fact that

$$A \mod t_j \neq r_i^{-1} p_i,$$

since $q_i = p_i - r_i A$ is a prime number and $t_j \nmid q_i$. Then, we can determine A by the Chinese Remainder Theorem. Anderson's trapdoor is now broken.

Hence, the user, who finds the trapdoor A, either stops using the hardware or is able to attack the other users of the same hardware by factoring their public modulus and finding their secret keys.

3 SETUP Mechanism of Young and Yung

The *Pretty-Awful-Privacy* (PAP) comes from the *Pretty-Good-Privacy* (PGP), which is widely-used software for secure email communications. We describe the two PAPs as SETUP mechanisms designed by Young and Yung. As for one of PAPs, attackers possesses RSA public and private keys for their purpose. For the other one, the public and private keys of ElGamal encryption scheme are employed for the attack. After seeing two types of PAPs, we introduce an analogous but more simple and secure scheme presented by Crépeau and Slackmon.

3.1 PAP as a Setup

Before seeing PAP, we first observe a simple SETUP mechanism which looks like the first PAP in the sense that the attacker has his or her own keys, say (E, N) and D, and it hides enough information within generated keys (e, n) to allow the attacker to find d from (e, n). We assume that the public exponent e is generated randomly from $\{0,1\}^k$ and the attacker's modulus N is k bits long.

Algorithm 2. Key generator

1. choose randomly k-bit primes p and q.
2. compute $e \leftarrow p^E \bmod N$.
3. if $\gcd(e, \phi(n)) = 1$
   then compute $d \leftarrow e^{-1} \bmod \phi(n)$.
   else choose new p and q.

The attacker can factor the modulus n by simply computing $e^D \bmod N$ and getting p. However, this is not effective for RSA cryptosystems such as PGP which uses a small exponent e. Hence, this attack is likely to be noticed. The problem of the above SETUP can be solved by hiding our secret information in the representation of n rather than e. In this case, we don't have to choose a big exponent e. Now we introduce a stronger version

of RSA SETUP, namely PAP. We assume (E, N) and D are the attacker's keys and N and n are k bits and $2k$ bits long, respectively. Let the public exponent e be fixed.

Algorithm 3. PAP as a setup

1. choose a random k-bit prime p.
2. randomize p using keyed randomizing function F.
   i.e. $p' \leftarrow F(p)$ with key $K + i$, where $i = 0$ as an initial value.
3. while $i < B_1$ do
   if $p' < N$, then go to step 4.
   else go to step 2 increasing i by 1.
   go to step 1.
4. set $p'' := (p')^E \mod N$.
5. randomize p'' using keyed randomizing function G.
   i.e. $p''' \leftarrow G(p')$ with key $K + j$, where $j = 0$ as an initial value.
6. while $j < B_2$ do
   set $X \leftarrow (p''' :$ random k-bit string).
   compute a quotient q when dividing X by p.
   if q is prime, then go to step 7.
   else go to step 5 increasing j by 1.
   go to step 1.
7. set $n \leftarrow pq$.
8. if $\gcd(e, \phi(n)) = 1$, then go to step 10.
   else go to step 9.
9. while $\gcd(e, \phi(n)) > 1$, increase e by 2.
10. return (p, q, e).

In step 6, we note that p''' is k bits, so we choose randomly k-bit string from $\{0, 1\}^k$, obtaining $2k$-bit X. Besides getting the pseudo-randomness of p', the function F is employed to get rid of the possibility that p' is not less than the attacker's modulus N. Also, we use the randomizing function G to avoid the excessive encryptions in the step 5. We need to choose suitable B_1 and B_2 in such a way that these numbers guarantee relatively high probability for finding valid p and q. We can achieve this using the Prime Number Theorem [2].

Given public keys generated by the above algorithm, the author of this scheme can attack the users as follows.

Algorithm 4. Attack

1. set U be the first k bits of n.
2. compute $p'' \leftarrow G^{-1}(U)$ using keys $K + j$ $(0 \leq j \leq B_2 - 1)$.
3. compute $p' \leftarrow (p'')^D \mod N$.
4. compute $p \leftarrow F^{-1}(p')$ using keys $K + i$ $(0 \leq i \leq B_1 - 1)$.
5. if there exists p such that $p|n$
   then we are done.
   else set $U \leftarrow U + 1$ and go to step 2.

When we divide X by p, the bit size of the remainder r is not bigger than k since p is k bits long. Hence, we obtain

$$p''' = U \quad \text{or} \quad p''' = U + 1,$$

since there is a chance the sum of the second k-bit string of n and the remainder r is $k + 1$ bits long. In step 2 and 4, we obtain all possible values for p'' and p, then we attack successfully if we find p such that $p|n$.

3.2 PAP Using Discrete Log Attack

Given a random value in some range, we are required to have new random value as input with a larger range in our new scheme. For instance, given x which is uniformly distributed in $[1, R]$, we want x' to be uniformly distributed in $[1, S]$, where $2R > S$. This can be done using the *Probabilistic Biased Removal Method* (PBRM), which can be described as follows.

We flip an unbiased coin, and we get a head (H) or a tail (T).
1. if $x \leq S - R$ and H, set $x' \leftarrow x$.
2. if $x \leq S - R$ and T, set $x' \leftarrow S - x$.
3. if $x > S - R$ and H, set $x' \leftarrow x$.
4. if $x > S - R$ and T, we flip again.

Then, it is easy to show x' is uniformly distributed in $[1, S]$. Since x is uniformly distributed in $[1, R]$, a value x can be chosen with a probability $1/R$. When $x \leq S - R$, x' takes x with a probability $1/2R$ and takes $S - x$ with a probability $1/2R$. When $x > S - R$, x' will set to be x with a

probability $1/2R$. Therefore, the probability to take a particular value x' in $[1, S]$ is $1/2R$, which implies x' is uniformly distributed in $[1, S]$.

Our new strong setup is based on the discrete log attack, so this version of PAP contains the attacker's ElGamal public key (Y, g, P) and private key X, where P has the same bit size as the factor p of n. We assume P is M bits long and K is a fixed symmetric key. Let the public key e be fixed.

Algorithm 5. Strong Setup PAP

1. choose a random $c_1 \in \mathbb{Z}_{P-1}$.
2. compute $z \leftarrow g^{c_1 - Wt} Y^{-c_1}$.
3. apply **PBRM** to z. *i.e.* $z' \leftarrow$**PBRM**(z).
4. compute $z'' \leftarrow H(z')$.
5. set the least significant bit of z'' to be 1.
6. set $i \leftarrow 0$ and while $i \leq B_1$ do
   set $p \leftarrow z'' + i$
   if p is prime, then go to step 7.
   else increase i by 2.
   go to step 1.
7. set $u \leftarrow$ **PBRM**$(\ g^{c_1} \mod P)$.
8. set $j \leftarrow 0$ and while $j < B_2$ do
   set $U \leftarrow G(u, K + j)$.
   choose the number N uniformly at random from $\{0,1\}^M$.
   set $n' \leftarrow (U : N)$.
   compute q such that $n' = pq + r$.
   if q is prime, then set $n \leftarrow n' - r$ and go to step 10.
   increase j by 1.
   go to step 1.
10. compute private key d given n and e.

In step 2, we perform an ElGamal encryption with a message z obtaining ciphertext (r, s), where $r = g^{c_1} \mod P$ and $s = Y^{c_1} z \mod P$ given a random value $c_1 \in \mathbb{Z}_{P-1}$. Then we set $r = s$ to compute z. Moreover, we apply g^{Wt} for z with t that is chosen uniformly at random from $\{0, 1\}$. The reason we use W is to avoid the detection of setup in the case the hash function H is invertible. We suppose the PBRM in step 3 is not applied (i.e. $z = z'$) and users have an excess to c_1 and z''. We also assume W is not used (i.e. $t = 0$). Under this circumstances, the setup might be detected by the users on the probabilistic basis as follows; The user can compute z, since the hash

function H is invertible. First, the user supposes the private key X is odd. On this assumption,

if c_1 is odd, then g^{c_1}/z would be a quadratic residue mod P
if c_1 is even, then g^{c_1}/z would be a quadratic non-residue mod P,

since $Y^{c_1}z = g^{c_1}$. Now the user supposes X is even. Then, g^{c_1}/z is always a quadratic residue mod P regardless of c_1. Hence, the user might find out the probabilistic pattern, which means the attack is not secure.

We might also consider the case that p is bigger than P. Thus, we apply PBRM in step 3 on the assumption that the domain of H is larger than P. The step 5 is done to make z'' odd and we try to find the prime number from z'' in the next step, where we set the limit of i to be some constant B_1 and we go back to the first stage if i exceeds B_1 to ensure the appreciable probability for finding a valid prime. In step 8, the randomizing keyed-function, G, applies a key $K+j$ to the data a obtaining a value $b = G(a, K+j)$. This function takes advantage of the pseudo-randomness and reducing the complexity just like the randomizing function in the previous PAP scheme. The remaining portions of this scheme are similar to the previous PAP.

Given public keys generated by the above algorithm, the author of this scheme can attack the users as follows. Note that $|\ |$ denotes a bit size.

Algorithm 6. Attack

1. set U be the upper $|n| - M$ bits of n.
2. compute $u \leftarrow G^{-1}(U)$ using keys $K + j$ $(0 \leq j \leq B_2 - 1)$.
3. if $u \geq P$, then set $(g^{c_1} \mod P) \leftarrow S - u$.
   else set $(g^{c_1} \mod P) \leftarrow u$.
4. compute $z \leftarrow g^{c_1 - Wt}(g^X)^{-c_1}$ with the private key X.
5. do the same steps as algorithm 5 (from step 3 to step 6).
6. if there exists p such that $p|n$
   then we are done.
   else set $U \leftarrow U + 1$ and go to step 2.

The PAP has an advantage that it allows any size of public exponent e. However, it has a critical flaw that the most significant bits of n are uniformly distributed which does not happen normally. We suppose we are using 1024-bit modulus n and we choose two 512-bit primes p, q. The two most significant bits of $n = pq$ have value 01 or 00 with a probability 0.38, 10 with 0.48 and 11 with 0.14. Of course, this implies that the most significant

bits of n are not uniformly distributed as it is the case with the standard RSA key generator.

3.3 Hidden Prime Factor

Crépeau and Slakmon [3] suggested the advanced type of the PAP version, which is very similar to PAP but the distribution of n, p, q is indistinguishable from the honest one. In this scheme, we hide the first half bits of p in the expression of n. Subsequently, *Coppersmith's partial information attack* allows to recover p and q from n. Given k-bit integer n, we denote $n\rceil^{k'}$ as first k' bits and $n\rfloor^{k'}$ as last k' bits of n, for $1 \leq k' \leq k$. We define a π_α to be a permutation from odd integers less than n to themselves with an imbedded backdoor α.

Algorithm 7. Hidden Prime Factor

1. pick a random prime p of the appropriate size such that $\gcd(e, p-1) = 1$.
2. pick a random odd number q' and compute $n' = pq'$.
3. compute $n \leftarrow (n'\rceil^{\frac{k}{8}} : \pi_\alpha(p\rceil^{\frac{k}{4}}) : n'\rfloor_{\frac{5}{8}k})$
   and compute odd q by setting $q \leftarrow \lfloor \frac{n}{p} \rfloor + (1 \pm 1)/2$.
4. while $\gcd(e, q-1) > 1$ or q is composite do
   pick a random even number m such that $|m| = k/8$.
   set $q \leftarrow q \oplus m$ and $n \leftarrow pq$.
5. compute $d \leftarrow e^{-1} \bmod \phi(n)$.
6. return (p, q, e, d).

Given the backdoor α and public knowledge (n, e), an attack can be performed as follows. From the public modulus n, the author of this scheme first computes $p\rceil^{\frac{k}{4}} = \pi_\alpha^{-1}(n\rceil^{\frac{3}{8}k}\rfloor_{\frac{k}{4}})$, where n is a k-bit integer. Then Coppersmith's partial information attack [4] makes it possible to factor n as p and q. Here we give a sketch of this procedure. If we set $P = p\rceil^{\frac{k}{4}} 2^{\frac{3}{4}k}$, we also get $Q = q\rceil^{\frac{k}{4}} 2^{\frac{3}{4}k}$ by division of n by p. Then p and q can be written as

$$p = P + p_0 \quad \text{and} \quad q = Q + q_0,$$

where P, Q are known but p_0, q_0 are unknown. Also, we have the following equation,

$$\begin{aligned} n = pq &= (P + p_0)(Q + q_0) \\ &= PQ + Pq_0 + Qp_0 + p_0 q_0. \end{aligned}$$

If we set $p_0 = x$ and $q_0 = y$, we have

$$\begin{aligned} f(x,y) &= (P+x)(Q+y) - n \\ &= PQ + Py + Qx + xy - n. \end{aligned}$$

Then factoring n is equivalent to finding the solution (p_0, q_0) to the polynomial equation $f(x,y) = 0$. In general, we define

$$D = \max_{ij}\{|f_{ij}|X^i Y^j\}$$

for a polynomial $f(x,y) = \sum_{ij} f_{ij}\ x^i y^j$ with $|x| < X$ and $|y| < Y$. Let α be a degree in each variable. As long as $(XY) < D^{2/3\alpha}$, we can find the solution to $f(x,y) = 0$ by reducing the problem to a lattice problem and then using the LLL-reduction algorithm. For details, see [4]. Given our polynomial $f(x,y) = PQ + Py + Qx + xy - n$, we have

$$D = \max\{XY, QX, PY, |PQ - n|\}$$

and the absolute values of x and y are bounded. *i.e.* $|x| < X = P/n^{\frac{1}{4}}$ and $|y| < Y = Q/n^{\frac{1}{4}}$. Then $(XY)^{\frac{3}{2}} < D$ holds, since

$$\frac{(XY)^{\frac{3}{2}}}{PY} = \left(\frac{PQ}{n}\right)^{\frac{1}{2}} < 1.$$

As the candidates for the permutation π_α, we might consider $\pi_\alpha : N \longrightarrow N$ which is defined by

$$x \longmapsto x \oplus (2\alpha)\rfloor_{|x|} \quad \text{or} \quad x \longmapsto x^{-1} \mod \alpha$$

where $|x|$ denotes a bit size of x and N is the set of odd integers smaller than n. It is easy to show that these are bijective and map odd integers to themselves. These permutations are simple to use, but in fact these are not secure for Algorithm 5. First, the permutation $\pi_\alpha(x) = x \oplus (2\alpha)\rfloor_{|x|}$ is insecure, since given two pairs of prime factors (p,q) and (p',q'), we have

$$\begin{aligned} (n \oplus n')\rceil^{\frac{3}{8}k}\rfloor_{\frac{k}{4}} &= \pi_\alpha(p\rceil^{\frac{k}{4}}) \oplus \pi_\alpha(p'\rceil^{\frac{k}{4}}) \\ &= (p\rceil^{\frac{k}{4}} \oplus (2\alpha)\rfloor_{|p\rceil^{\frac{k}{4}}|}) \oplus (p'\rceil^{\frac{k}{4}} \oplus (2\alpha)\rfloor_{|p'\rceil^{\frac{k}{4}}|}) \\ &= (p' \oplus p)\rceil^{\frac{k}{4}}, \end{aligned}$$

which does not happen normally in the standard system.

If we use the permutation $\pi_\alpha(x) = x^{-1} \bmod \alpha$, then we have

$$n\rceil^{\frac{3}{8}k}\rfloor_{\frac{k}{4}}p\rceil^{\frac{k}{4}} = 1 \mod \alpha,$$

which implies $n\rceil^{\frac{3}{8}k}\rfloor_{\frac{k}{4}}p\rceil^{\frac{k}{4}} - 1$ is a multiple of α. Hence, one may recover the backdoor α after several runs. However, the following permutation can be used for our algorithm without any deficiency;

$$\pi_{\alpha,\beta}(x) = (x \oplus (2\beta)\rfloor_{|x|})^{-1} \mod \alpha.$$

4 Relative Simple RSA SETUP

Previous SETUP mechanisms created public modulus n in such a way that the manufacturer can factor n. From now on, we present extremely simple schemes which generate genuinely random primes p and q but embed simple backdoors in private and public exponent pair (d, e), so that the manufacturer can factor n from public information (n, e). These methods are indistinguishable in a running time analysis, since these are slightly changed from a standard RSA key generator.

4.1 Hidden Small Private Exponent δ

This scheme employs Wiener's attack for RSA small private exponent. First, this method generates the private and public keys (δ, ϵ), where δ is small enough to use Wiener's attack. Then, this *weak pair* is transformed into random-looking private and public keys (d, e), where there exists secret relation between these two pairs and this information can be used for an attack. The main idea of this algorithm is to imbed a backdoor α to hide the value of δ in the expression of the public key e. We note that n is a k-bit integer. We now describe this algorithm in detail. Note that $|\ |$ denotes a bit size.

Algorithm 8. RSA-HSPE$_\alpha$

1. choose random primes p, q such that $n = pq$ is k bits long.
2. repeat the following.
   pick an odd random number δ such that $\gcd(\delta, \phi(n)) = 1$ and $|\delta| \leq k/4$.
   compute $\epsilon \leftarrow \delta^{-1} \bmod \phi(n)$ and $e \leftarrow \pi_\alpha(\epsilon)$.

until $\gcd(e, \phi(n)) = 1$.
3. compute $d \leftarrow e^{-1} \bmod \phi(n)$.
4. return (p, q, e, d).

Given the secret backdoor α and public information (n, e), the attack can be launched as follows. The author of the scheme first recovers ϵ by computing $\epsilon = \pi_\alpha^{-1}(e)$ with a backdoor α. Then, he applies Wiener's attack to find δ from (n, ϵ). Once the pair (δ, ϵ) is recovered, he can factor n using RSA-FACTOR algorithm [8].

We can actually reduce the complexity of finding δ by updating δ with $\delta/\gcd(\delta, \phi(n))$ if $\gcd(\delta, \phi(n)) > 1$. The running time of this cheating algorithm is nearly the same as the original RSA key generation algorithm, since steps 1, 3 and 4 are exactly the same and the number of gcd calculation inside the main loop is roughly three times the original. Moreover, the computation of the permutation π_α is negligible for performing gcds.

However, if we restrict the size of e to a small value, then this attack fails because e is also k bits long.

4.2 Hidden Small Public Exponent ϵ

The next two schemes are based on the recent results of Boneh, Durfee and Frankel [11]. We state the theorems without proof.

Theorem 2. Let n be the RSA public modulus such that $n = pq$ and $p < 4q$.

(a) Let t be an integer in the interval $[|n|/4, |n|/2]$ and e be a prime in the interval $[2^t, 2^{t+1}]$. Given (n, e) and the t most significant bits of d, we can compute the whole of d.

(b) Let t be an integer in the interval $[1, |n|/2]$ and e be an integer in the interval $[2^t, 2^{t+1}]$. Given (n, e), the t most significant bits of d, and the $|n|/4$ least significant bits of d, we can compute the whole d.

In order to apply the above theorems, we first generate a random private and public key (δ, ϵ) with small ϵ, and transform these into a random looking pair (d, e), which can be inverted to (δ, ϵ) with a backdoor by the author of these schemes. From public information (n, e), the author can compute ϵ and the partial knowledge of δ. Then the above theorems allow to recover all of d. We first observe the case of hidden small prime public exponent ϵ. The main idea of this algorithm is to embed a backdoor α to hide the value of

ϵ and partial knowledge of δ in the expression of the public key e. We note that n is a k-bit integer.

Algorithm 9. RSA-HSPPE$_\alpha$

1. choose random primes p, q such that $n = pq$ is k bits long.
2. repeat the following.
   pick a random prime number ϵ such that $\gcd(\epsilon, \phi(n)) = 1$ and $|\epsilon| = k/4$.
   compute $\delta \leftarrow \epsilon^{-1} \bmod \phi(n)$ and $e \leftarrow \pi_\alpha(\delta \rceil \frac{k}{4} : \epsilon)$.
   until $\gcd(e, \phi(n)) = 1$.
3. compute $d \leftarrow e^{-1} \bmod \phi(n)$.
4. return (p, q, e, d).

Given the secret backdoor α and public information (n, e), the attack can be performed as follows. The author of the scheme first computes $\pi_\alpha^{-1}(e)$ with a backdoor α, obtaining the first $k/4$ significant bits of δ and ϵ. Then Theorem 2(a) allows to recover the whole δ. Once the pair (δ, ϵ) is recovered, he can factor n using the RSA-FACTOR algorithm.

As the advantage of this scheme, we might consider its simplicity and the small size of keys. Since the concatenations $(\delta \rceil \frac{k}{4} : \epsilon)$ is $k/2$ bits long, we may generate public exponent e in the range of $[\sqrt{n}, \phi(n)]$ with extra random padding if necessary.

We can also reduce the complexity of finding δ by updating δ with $\delta/\gcd(\delta, \phi(n))$ if $\gcd(\delta, \phi(n)) > 1$. However, this algorithm is not competitive in terms of running time, since it takes time to create a prime ϵ even though ϵ is only $k/4$ bits long.

Now we describe another similar version of hidden small public exponent ϵ, where ϵ does not need to be prime.

Algorithm 10. RSA-HSPE$_\alpha$

1. choose random primes p, q such that $n = pq$ is k bits long.
2. repeat the following.
   pick a random prime number ϵ such that $\gcd(\epsilon, \phi(n)) = 1$ and $|\epsilon| = t$.
   compute $\delta \leftarrow \epsilon^{-1} \bmod \phi(n)$ and $e \leftarrow \pi_\alpha(\delta \rceil \frac{k}{4} : \delta \rfloor_{\frac{k}{4}} : \epsilon)$.
   until $\gcd(e, \phi(n)) = 1$.
3. compute $d \leftarrow e^{-1} \bmod \phi(n)$.
4. return (p, q, e, d).

Given the secret backdoor α and public information (n, e), the attack can be performed as follows. The author of the scheme first computes $\pi_\alpha^{-1}(e)$ with a backdoor α, obtaining the first $k/4$ significant bits of δ, the least significant bits of δ and ϵ. Then Theorem 2(b) allows to recover the whole δ. Once the pair (δ, ϵ) is recovered, he can factor n using the RSA-FACTOR algorithm. With regards to the running time, this scheme compares very well with the original RSA key generation algorithm

5 Conclusion

We introduced several types of setup mechanisms for RSA cryptographic device. Some of them are not secure, which either allows the third party to reverse-engineer the setup mechanism or implies that the users may detect the existence of the setup. On the other hand, the others have survived cryptanalysis. These results mean there maybe exist unknown RSA setups and the current RSA-cryptosystem users are being attacked by the manufacturer of the devices.

References

[1] R. J. Anderson, "A practical RSA trapdoor", Electronics Letters, 1993, vol.29, no.11, p.995

[2] G. E. Andrews, "Number Theory", 1971, Dover Publication Inc., p.100

[3] C. Crépeau, A. Slakmon, "Simple backdoors for RSA key generation", Cryptographers' Track at RSA Conference, LNCS 2612, 2003, pp.403-416

[4] D. Coppersmith, "Finding a small root of a bivariate integer equation : factoring with high bits known", Advances in Cryptology: Proceedings of EUROCRYPT '96, LNCS 1070, 1996, p.178

[5] B. S. Kaliski, "Anderson's RSA trapdoor can be broken", Electronics Letters, 1993, vol.29, no.15, p.1387

[6] A. K. Lenstra, H. W. Lenstra, and L. Lovasz, "Factoring polynomials with rational coefficients", Mathematische Annalen, 1982, vol.261, pp.515-534

[7] J. C. Lagarias, "Knapsack public key cryptosystems and diophantine approximation", Advances in Cryptology: Proceedings of CRYPTO '83, 1984, pp.3-23

[8] D. R. Stinson, "Cryptography-Theory and Practice (2nd edition)", CHAPMAN & HALL/CRC, p.197

[9] A. Young, M. Yung, "The dark side of black-box cryptography or: Should we trust capstone?", Advances in Cryptology - CRYPTO '96, LNCS 1109, 1996, pp.89-103

[10] A. Young, M. Yung, "Kleptography : Using cryptography against cryptography", Adbvances in Cryptology - Eurocrypt '97, LNCS 1233, 1997, pp.62-74

[11] D. Boneh, G. Durfee, and Y. Frankel, "An attack on RSA given a small fraction of the private key bits", Advances in Cryptology - Asiacrypt '98, LNCS 1514, 1998, pp.25-34

[12] C. P. Schnorr and M. Euchner, "Lattice Basis Reduction and Solving Subset Sum Problems", Fundamentals of Computation Theory '91, LNCS 591, 1991, pp.68-85

Appendix

Generalities on Lattices

Definition 1. Let $\mathbb{R}^n$ be the n-dimensional real vector space and let $B = \{b_1, b_2, \ldots, b_n\}$ be a set of n linearly independent vectors in $\mathbb{R}^n$. Then the *lattice* generated by B is the set

$$L = \{t_1 b_1 + \ldots + t_n b_n |\ t_1, \ldots, t_n \in \mathbb{Z}\},$$

and B is called a *basis* of a lattice L.

That is, a lattice L is the set of linear combination of n linearly independent vectors $b_1, \ldots, b_n$ with integer coefficients. Given an ordered lattice basis $\{b_1, b_2, \ldots, b_n\} \subseteq \mathbb{R}^n$, the *Gram-Schmidt orthogonalization* $\{\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n\} \subseteq \mathbb{R}^n$ is defined by

$$\hat{b}_1 = b_1,\ \hat{b_i} = b_i - \textstyle\sum_{j=1}^{i-1} \mu_{i,j} \hat{b}_j \quad \text{for } i = 2, \ldots, n$$

where $\mu_{i,j} = < b_i, \hat{b}_j > / < \hat{b}_j, \hat{b}_j >$ is the *Gram-Schmidt coefficient.* Then we define the *determinant* of a lattice by

$$\det(L) = \prod_{i=1}^{n} \|\hat{b}_i\|.$$

Equivalently, the determinant can be represented by

$$\det(L) = \det([b_1, \ldots, b_n][b_1, \ldots, b_n]^T)^{1/2},$$

given a lattice basis $\{b_1, \ldots, b_n\}$.

Now, we are ready to define the crucial concept of a reduced basis.

Definition 2. An ordered basis $\{b_1, \ldots, b_n\} \subseteq \mathbb{R}^n$ of L is called *LLL-reduced* with δ if

$$\begin{aligned} |\mu_{i,j}| &\leq 1/2 \quad \text{for } \ 1 \leq j < i \leq n \\ \delta \|\hat{b}_{i-1}\|^2 &\leq \|\hat{b}_i + \mu_{i,j} \hat{b}_{i-1}\|^2 \quad \text{for } \ 1 < i \leq n, \end{aligned}$$

where $1/4 < \delta \leq 1$.

The first property is called *size-reduced*. The absolute value of the Gram-Schmidt coefficient, $|\mu_{i,j}|$, means the ratio of the length of the projection vector of b_i to $\hat{b}_j$. Hence, the *size-reduced* property implies that the vectors, $b_1, \ldots, b_n$ have short length or, equivalently, they are pairwise nearly orthogonal, which is the goal of lattice basis reduction algorithm.

From now on, we investigate what the second property implies. We fix $\delta = 3/4$. From LLL-reduced conditions, we obtain that

$$\begin{aligned} \|\hat{b}_i\|^2 &\geq (3/4 - \mu_{i,i-1}^2)\|\hat{b}_{i-1}\|^2 \\ &\geq 1/2\ \|\hat{b}_{i-1}\|^2 \end{aligned}$$

for $1 < i \leq n$, which implies

$$\|\hat{b}_j\|^2 \leq 2^{i-j}\|\hat{b}_i\|^2 \quad \text{for } 1 \leq j \leq i \leq n.$$

By the definition of Gram-Schmidt orthogonalization and the above fact, we have that

$$\begin{aligned} \|b_i\|^2 &= \|(\hat{b}_i + \sum_{j=1}^{i-1} \mu_{i,j}\hat{b}_j)\|^2 \\ &= \|\hat{b}_i\|^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2\|\hat{b}_j\|^2 \\ &\leq \|\hat{b}_i\|^2 + \sum_{j=1}^{i-1} \frac{1}{4}(2^{i-j})\|\hat{b}_i\|^2 \\ &= \big(1 + \frac{1}{4}(2^i - 2)\big)\|\hat{b}_i\|^2 \\ &\leq 2^{i-1}\|\hat{b}_i\|^2 \end{aligned}$$

Therefore, we obtain the following result.

Proposition 1. Let $\{b_1, b_2, \ldots, b_n\}$ be a LLL-reduced basis for a lattice L in $\mathbb{R}^n$, and let $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n$ be vectors defined as above. Then we have

$$\|b_j\|^2 \leq 2^{i-1}\|\hat{b}_i\|^2 \ \text{ for } \ 1 \leq j \leq i \leq n. \tag{2}$$

Let v be a nonzero lattice vector, *i.e.* $v \in L$, $v \neq 0$. Then, v can be written as

$$v = \sum_{i=1}^{n} r_i b_i = \sum_{i=1}^{n} \hat{r}_i \hat{b}_i.$$

with $r_i \in \mathbb{Z}$ and $\hat{r}_i \in \mathbb{R}$ $(1 \leq i \leq n)$. Suppose that i is the largest index with $r_i \neq 0$. Then we have

$$\begin{aligned} \sum_{k=1}^{i-1} r_k b_k + r_i b_i &= \sum_{k=1}^{i-1} \hat{r}_k \hat{b}_k + \hat{r}_i \hat{b}_i \\ &= \sum_{k=1}^{i-1} (\hat{r}_k - \mu_{i,k}) \hat{b}_k + \hat{r}_i b_i. \end{aligned}$$

We note that $\{b_1, b_2, \ldots, b_{i-1}\}$ and $\{\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{i-1}\}$ span the same space and b_i does not belong to this space. Inductively, we compute all the components of v in terms of $\hat{r}_k$ and $\mu_{i,k}$ relative to the basis $\{b_1, b_2, \ldots, b_n\}$, where $1 \leq k \leq i$. This gives the result, $r_i = \hat{r}_i$. Then we obtain

$$\|\hat{b}_i\|^2 \leq \hat{r}_i^2 \ \|\hat{b}_i\|^2 \leq \|v\|^2, \tag{3}$$

since $r_i \in \mathbb{Z}$ and $r_i \neq 0$. By setting $j = 1$ in (2), we have that

$$\|b_1\|^2 \leq 2^{i-1} \|\hat{b}_i\|^2 \leq \ 2^{n-1} \|\hat{b}_i\|^2. \tag{4}$$

Facts (3) and (4) imply the following result.

Proposition 2. Let $\{b_1, b_2, \ldots, b_n\}$ be a reduced basis of lattice L. Then

$$\|b_1\|^2 \leq 2^{n-1} \|v\|^2 \tag{5}$$

for every nonzero lattice vector $v \in L$.

Let $v_1, v_2, \ldots, v_l$ be linearly independent lattice vectors in L. Then we can express $v_j = \sum_{i=1}^{n} r_{ij} b_i$, where $r_{ij} \in \mathbb{Z}$ and $1 \leq j \leq l$. If we set i_j to be the largest i with $r_{ij} \neq 0$ for every fixed j, then by (3) we have

$$\|v_j\|^2 \leq \|\hat{b}_{i_j}\|^2 \ \text{ for } \ 1 \leq j \leq l. \tag{6}$$

Then, we renumber v_j such that $i_1 \leq i_2 \leq \cdots \leq i_l$. Suppose $j > i_j$ for $1 \leq j \leq l$. Then, $v_1, v_2, \ldots, v_j$ become linearly dependent, since all of them

belong to $\sum_{i=1}^{j-1} \mathbb{R}b_i$. This contradicts the assumption that $v_1, v_2, \ldots, v_l$ are linearly independent lattice vectors. Hence, with the fact $j \leq i_j$ and (6), we obtain that

$$\|b_j\|^2 \leq 2^{i_j-1}\|\hat{b}_{i_j}\|^2 \leq 2^{n-1}\|\hat{b}_{i_j}\|^2 \leq 2^{n-1}\|v_j\|^2$$

for $j = 1, 2, \ldots, l$. Hence, we have the following result.

Proposition 3. Let $\{b_1, b_2, \ldots, b_n\}$ be a reduced basis of lattice L in $\mathbb{R}^n$, and let $v_1, v_2, \ldots, v_l \in L$ be linearly independent lattice vectors. Then we have

$$\|b_j\|^2 \leq 2^{n-1} \max\{\|v_1\|^2, \|v_2\|^2, \ldots, \|v_l\|^2\} \tag{7}$$

for $j = 1, 2, \ldots, l$.

For our lattice L, we define the *i-th successive minimum* λ_i of lattice L relative the Euclidean norm $\| \ \|$ to be the smallest real number r such that there are i linearly independent vectors in L of length at most r. In other words, λ_i is the smallest radius of a ball that is centered at the origin and which contains i linearly independent lattice vectors. Then the previous results (2) and (7) induce the following conclusion.

Theorem 3. Every LLL-reduced basis $\{b_1, b_2, \ldots, b_n\} \subseteq \mathbb{R}^n$ for a lattice L with δ satisfies

$$2^{1-i}\lambda_i^2 \leq \|b_i\|^2 \leq 2^{n-1}\lambda_i^2,$$

for $1 \leq i \leq n$.

The above theorem allows the second property of LLL-reduced basis to give good approximations of the values $\|b_i\|$ to the successive minima λ_i. This guarantees the LLL-reduced basis has good approximation to the shortest lattice basis.

LLL-reduction Algorithm

We first describe a size reduction algorithm which is used as a subroutine in the LLL-reduction algorithm with integer lattice bases as input. This algorithm takes integral lattice basis $b_1, b_2, \ldots, b_n \in \mathbb{Z}^n$ and corresponding

Gram-Schmidt coefficients $\mu_{i,j}$ for $1 \leq j < i \leq n$ and returns lattice basis $b_1, b_2, \ldots, b_n$ with a size-reduced b_k and updated corresponding Gram-Schmidt coefficients. For the Gram-Schmidt coefficient $\mu_{i,j}$, we define $r_{i,j}$ as $r_{i,j} = \lceil \mu_{i,j} - 1/2 \rceil$. Then it is clear that $|\mu_{i,j} - r_{i,j}| < 1/2$ and we will apply this fact to get a size-reduced vector.

Algorithm 11. Size-Reduction(b_k)

for $j = k - 1$ downto 1
  do if $|\mu_{k,j}| > 1/2$,
    then $b_k \leftarrow b_k - r_{k,j} b_j$
      for $i = 1$ to n do $\mu_{k,i} \leftarrow \mu_{k,i} - r_{k,j}\mu_{j,i}$

The correctness can be shown by an induction on the number of iterations. The idea is that we first assume that $|\mu_{k,j}| > 1/2$ for some $1 \leq j \leq k-1$. Then we update b_k with $b_k - r_{k,j} b_j$ and also Gram-Schmidt coefficients $\mu_{k,i}$ with $\mu_{k,i} - r_{k,j}\mu_{j,i}$ for $1 \leq i \leq n$. Let's denote each updated vector and coefficient as b'_k and $\mu'_{k,i}$. The updated coefficient $\mu'_{k,i}$ is reasonable, since

$$\begin{aligned} \mu'_{k,i} &= \frac{< b'_k, \hat{b}_i >}{< \hat{b}_i, \hat{b}_i >} \\ &= \frac{< b_k - r_{k,i} b_j, \hat{b}_i >}{< \hat{b}_i, \hat{b}_i >} \\ &= \mu_{k,i} - r_{k,i}\mu_{j,i}. \end{aligned}$$

The updated coefficients give a desired result, $|\mu'_{k,j}| = |\mu_{k,j} - r_{k,j}| < 1/2$, for $i = j$. On the other hand, for $i = j+1, \ldots, k-1$ we don't have to check if $|\mu_{k,i}| \leq 1/2$, since

$$\begin{aligned} |\mu'_{k,i}| &= |\mu_{k,i} - r_{k,j}\mu_{j,i}| \\ &= |\mu_{k,i}|. \end{aligned}$$

This is clearly at most $1/2$. This is the reason why we do the iterations in reverse order from $k-1$ to 1.

Now we present a LLL-reduction algorithm which takes as an input ordered basis $b_1, b_2, \ldots, b_n \in \mathbb{Z}^n$ and $\delta \in [1/4, 1]$ and returns LLL-reduced lattice basis.

Algorithm 12. LLL-Reduction

(Pre-computation)
    set $k = 2$ (k is a stage)
    compute $\mu_{i,j}$ for $1 \leq j < i \leq n$ and $\|\hat{b}_i\|^2$ for $i = 1, 2, \ldots, n$
(Main-computation)
    while $k \leq n$ do
        Size-Reduction(b_k)
        if $\delta\|\hat{b}_{k-1}\|^2 > \|\hat{b}_k + \mu_{k,k-1}\hat{b}_{k-1}\|^2$
          then swap b_k and b_{k-1}
                update $\|\hat{b}_k\|^2, \|\hat{b}_{k-1}\|^2$ and $\mu_{i,u}, \mu_{u,i}$
                    for $u = k-1, k$ and $i = 1, \ldots, n$.
          else $k \leftarrow k + 1$

Assuming a new LLL-reduced lattice basis $\{v_1, \ldots, v_n\}$ is constructed from a lattice basis $\{b_1, \ldots, b_n\}$ of L, we have

$$\|v_1\| \leq 2^{n/2} \det(L)^{1/n} \text{ and } \|v_2\| \leq 2^{n/2} \det(L)^{1/(n-1)},$$

where v_1 and v_2 are the two shortest vectors in the new basis [12].

Let's describe what happens in the above algorithm. We denote $\sum_{i=1}^{k-2} \mathbb{R}\hat{b}_i$ as the space of linear combinations of vectors $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{k-2}$ with any real number coefficients. Then we can check easily that $\hat{b}_{k-1}$ is the projection of b_{k-1} to the orthogonal complement of $\sum_{i=1}^{k-2} \mathbb{R}\hat{b}_i$ and also that $\hat{b}_k + \mu_{k,k-1}\hat{b}_{k-1}$ is the projection of b_k to the orthogonal complement of $\sum_{i=1}^{k-2} \mathbb{R}\hat{b}_i$, since $b_k = (\hat{b}_k + \mu_{k,k-1}\hat{b}_{k-1}) + \sum_{i=1}^{k-2} \mu_{k,j}\hat{b}_i$. Suppose we are at the k stage. Then we see if the two ordered vectors b_k and b_{k-1} satisfy the second condition of LLL-reduction. If they do, we proceed to next stage $k+1$ and investigate two vectors b_k and b_{k+1}. Otherwise, we swap two vectors b_k and b_{k-1} with the other vectors unchanged. We now describe what happens in this process. We denote W as the orthogonal complement of $\sum_{i=1}^{k-2} \mathbb{R}\hat{b}_i$. Then we have

$$\frac{1}{\delta}\|Proj_W\ b_k\|^2 < \|Proj_W\ b_{k-1}\|^2,$$

which implies that

$$\delta\|Proj_W\ b_k\|^2 < \|Proj_W\ b_{k-1}\|^2,$$

since $1/4 < \delta < 1$. After interchanging two vectors, we have the ordered lattice basis $\{b_1, b_2, \ldots, b_{k-2}, b_k, b_{k-1}, b_{k+1}, \ldots, b_n\}$. We rename this ordered basis as $\{b'_1, b'_2, \ldots, b'_n\}$ as follows:

$$b'_i = \begin{cases} b_{k-1} & \text{if } i = k \\ b_k & \text{if } i = k-1 \\ b_i & \text{otherwise.} \end{cases}$$

Now, we have that

$$\delta \| Proj_W \ b'_{k-1} \|^2 < \| Proj_W \ b'_k \|^2,$$

which is equivalent to

$$\delta \| \hat{b}'_{k-1} \|^2 < \| \hat{b}'_k + \mu_{k,k-1} \hat{b}'_{k-1} \|^2.$$

After this procedure, we know that b'_{k-1} and b'_k satisfy the second condition of the reduced basis, but we have to check b'_{k-1} and b'_{k-2}.

We now show the algorithm terminates after polynomially many iterations. We take a value of δ as $3/4$. Given a lattice basis $\{b_1, \ldots, b_n\}$, we define

$$D = \prod_{i=1}^{n} (\det[b_1, \ldots, b_i])^2.$$

Since $\det(L)^2 \in \mathbb{Z}$, we have $D^2 \in \mathbb{N}$ by the definition of D. We note that a call Size-Reduction does not change the absolute value of determinant. If two vectors b_k and b_{k-1} are exchanged, D is affected by a change from $(\det[b_1, \ldots, b_{k-2}, b_{k-1}])^2$ to $(\det[b_1, \ldots, b_{k-2}, b_k])^2$, while $(\det[b_1, \ldots, b_k])^2$ is unchanged. Then D is updated by

$$D \cdot \frac{(\det[b_1, \ldots, b_{k-2}, b_k])^2}{(\det[b_1, \ldots, b_{k-2}, b_{k-1}])^2},$$

which is equivalent to

$$D \cdot \frac{\| Proj_W \ b_k \|^2}{\| Proj_W \ b_{k-1} \|^2},$$

where W is the orthogonal complement of $\sum_{i=1}^{k-2} \mathbb{R} \hat{b}_i$. By the above argument, this can be written by

$$D \cdot \frac{\| \hat{b}_k + \mu_{k,k-1} \hat{b}_{k-1} \|^2}{\| \hat{b}_{k-1} \|^2}.$$

Since $\|\hat{b}_k + \mu_{k,k-1}\hat{b}_{k-1}\|^2 < \frac{3}{4}\|\hat{b}_{k-1}\|^2$, D is updated to a value less than $\frac{3}{4}D$. Hence, the number of iterations t is bounded by $(\frac{3}{4})^{2t}D^2 > 1$, which implies $t < \log_{3/4} D$. Therefore, the total number of iterations for while loop are at most $\log_{3/4} D + n$, where we observe k increases whenever a swap does not happen. Since D can be computed within polynomial time, we finally conclude that the LLL-reduction algorithm terminates in polynomially many iterations.

Finally, it remains to prove the correctness of the LLL-reduction algorithm. We show this by an induction on the number of stages. The induction hypothesis is that we have LLL-reduced lattice basis vectors $b_1, \ldots, b_{k-1}$. As a base case, it is clear for $k = 2$. Since a call Size-Reduction(b_k) only updates b_k and $\mu_{k,i}$ for $i = 1, 2, \ldots, k-1$, the vectors $b_1, \ldots, b_{k-1}$ are still LLL-reduced. If the condition in line 3 is not satisfied, the vectors $b_1, \ldots, b_k$ are LLL-reduced, since these are already size-reduced and $b_1, \ldots, b_{k-1}$ satisfies the second LLL-reduced condition. Hence, we conclude that the output of lattice basis vectors are LLL-reduced at the final stage $k = n + 1$.

If we let $M \in \mathbb{R}$ be such that $\|b_i\|^2 \leq M$ for $1 \leq i \leq n$, the total number of arithmetic operations is performed in $O(n^4 \log M)$ on integers of length $O(n \log M)$.