Factoring a Multiprime Modulus N with Random Bits

Routo Terada(✉) and Reynaldo Cáceres Villena

Department of Computer Science, University of São Paulo, São Paulo, Brazil
rt@ime.usp.br, reynaldo.cv@gmail.com

Abstract. In 2009, Heninger and Shacham presented an algorithm using the Hensel's lemma for reconstructing the prime factors of the modulus $N = r_1 r_2$. This algorithm computes the prime factors of N in polynomial time, with high probability, assuming that a fraction greater than or equal to 59 % random bits of its primes r_1 and r_2 is given. In this paper, we present the analysis of Hensel's lemma for a multiprime modulus $N = \prod_{i=1}^{u} r_i$ (for $u \geq 2$) and we generalise the Heninger and Shacham's algorithm to determine the minimum fraction of random bits of its prime factors that is sufficient to factor N in polynomial time with high probability.

Keywords: Factoring a multiprime modulus N · Random key bits leakage attack · Cold boot attack

1 Introduction

According to Skorobogatov [14], we know the recovery of information (bits) of the RAM can be done with a certain error due to the data remanent property of RAM and this error can be decreased by cooling techniques studied by Halderman [4]. This attack is known as cold boot attack [2] and it is able to make a copy of the DRAM used by the decryption process of an RSA cryptosystem with some private key identification techniques ([4,11,13]). Then, it may identify the set of correct bits of the secret key sk of the **Basic RSA cryptosystem.**[1] Inspired by this attack the underlying ideas are used to identify and recover the secret key bits of **a multiprime RSA cryptosystems**[2].

It's widely known that the main drawback of Basic RSA cryptosystem is the relatively expensive encryption and decryption operations. It is relevant to mention there is an advantage to use more than two primes in the RSA modulus N. The decryption process is faster when it is done partially with respect to each prime modulus and then combine them using the Chinese Remainder Theorem

R.C. Villena—Supported by CAPES, Brazil.

[1] Basic RSA is when the modulus N is product of two primes.

[2] Multi-prime RSA is a generalization of the Basic RSA where the modulus N is the product of two or more primes.

© Springer International Publishing Switzerland 2015
Y. Desmedt (Ed.): ISC 2013, LNCS 7807, pp. 185–196, 2015.
DOI: 10.1007/978-3-319-27659-5_13

(see [8,12]). The more prime factors in the modulus N, the faster is the decryption process, providing a practical solution to the high cost of decryption. The advantages for performing these operations in parallel are that the number of bit operations is at most $\frac{3}{2u^3}n^3$ and the required space is only $\lg(r_i) = \frac{n}{u}$ where $n = \lg(N)$ (number of bits of modulus N) and u is the number of prime factors of N. The time and space used to perform the decryption process is lower for values u greater than 2 but the risk of the modulus N to be factored without extra information is increased [7].

As mentioned before, inspired by cold boot attacks, it was shown that if queries to an oracle for a relatively small number of bits of the secret key is given, it is possible to factor the Basic RSA modulus $N = pq$ in polynomial time with:

- $\frac{1}{4}n$ LSB (Least Significant Bits) or $\frac{1}{4}n$ MSB (Most Significant Bits) of p [3].
- a maximum of $\log\log(N)$ unknown blocks and $ln(2) \approx 0.70$ fraction of known bits of p [6].
- a fraction δ of random bits of p and q greater than or equal to $2 - 2^{\frac{1}{2}} \approx 0.59$ [5].

In comparison, for a multiprime modulus $N = r_1 r_2 ... r_u$, $\frac{i}{i+1}\frac{n}{u}$ LSB or $\frac{i}{i+1}\frac{n}{u}$ MSB of r_i, for $1 \leq i \leq u-1$ is required [7].

An example: to factor a 3-prime modulus N ($u = 3$) the minimum requirement is:

- $\frac{n}{6}$ LSB or $\frac{n}{6}$ MSB of r_1, and $\frac{2n}{9}$ LSB or $\frac{2n}{9}$ MSB of r_2
- $0.38n$ fraction of bits of primes r_1 and r_2.

Hence more and more bits are required to factor a modulus N with u greater than 2.

The case that we analyzed is to factor a multiprime modulus N with a fraction δ of random bits of its primes, where δ was computed to factor N in polynomial time with high probability. Hence we show that a multiprime modulus N offers more security than a basic modulus N. Our main result is that:

- To factor the integer $N = \prod_{i=1}^{u} r_i$ in polynomial time, using Hensel's lemma, a fraction $\delta = 2 - 2^{\frac{1}{u}}$ of random bits of its primes is sufficient.

With this result, there is a need of $\delta \geq 2 - 2^{\frac{1}{3}} \approx 0.75$ fraction of random bits of prime factors to factor a 3-prime modulus N. Therefore 3-prime is better, adversarially, than $\delta \geq 2 - 2^{\frac{1}{2}} \approx 0.59$ for a basic modulus N.

Kogure et al. [9] proved a general theorem to factor a multi-power modulus $N = r_1^m r_2$ with random bits of its prime factors. The particular cases of Takagi's variant of RSA [15] and Paillier Cryptosystem [10] are addressed. The bounds for expected values in our cryptanalysis are derived directly, without applying their theorem.

2 Algorithm to Factor a Multiprime Modulus N

We consider an equation of integer $N = \prod_{i=1}^{u} r_i$ that can be expressed as a polynomial

$$f(x_1, x_2, ..., x_u) = N - \prod_{i=1}^{u} x_i$$

with solution (roots) $r = \langle r_1, r_2, ...r_u \rangle$. We applied the idea in Heninger and Shacham's algorithm, to rebuild the primes r_i beginning with their LSB and MSB. In others words, we can lift all roots of the polynomial $f(x_1, x_2, ..., x_u) \pmod{2^{j+1}}$ from a root of the polynomial $f(x_1, x_2, ..., x_u) \pmod{2^j}$. Applying this scheme, we can lift from one root of polynomial $f(x_1, x_2, ..., x_u) \pmod 2$ to obtain all roots of $f(x_1, x_2, ..., x_u) \pmod{2^2}$. Then, from these roots, lift all roots of $f(x_1, x_2, ..., x_u) \pmod{2^3}$ and so on, up to all roots for $f(x_1, x_2, ..., x_u) \pmod{2^{\frac{n}{u}}}$. One of these roots is the solution for $\langle r_1, r_2, .., r_u \rangle$, because we assume N is balanced[3] hence the length of its primes is of $\frac{n}{u}$ bits.

$$\langle r_1, r_2, .., r_u \rangle \in \text{ roots of } f(x_1, x_2, .., x_u) \pmod{2^{\frac{n}{u}}}$$

To understand the relation between a root of $f(x_1, x_2, .., x_u) \pmod{2^j}$ and the roots of $f(x_1, x_2, .., x_u) \pmod{2^{j+1}}$ we introduce below the Hensel's lemma for multivariate polynomials.

Lemma 1 (Multivariate Hensel's Lemma *[5]*). *Let* $f(x_1, x_2, ..., x_n) \in \mathbb{Z}[x_1, x_2, ..., x_n]$ *be a multivariate polynomial with integer coefficients. Let* π *be a positive integer and* $r = (r_1, r_2, ..., r_n) \in \mathbb{Z}^n$ *be a solution for* $f(x_1, x_2, ..., x_n) \equiv 0 \pmod{\pi^j}$. *Then* r *can be lifted to a root* $r + b \pmod{\pi^{j+1}}$, *if* $b = (b_1\pi^j, b_2\pi^j, ..., b_n\pi^j)$, $0 \leq b_i \leq \pi - 1$, *satisfies*

$$f(r + b) = f(r) + \sum_{i=1}^{n} b_i \pi^j f_{x_i}(r) \equiv 0 \pmod{\pi^{j+1}}$$

where f_{x_i} *is the partial derivative of* f *with respect to* x_i, *or equivalently*

$$\frac{f(r)}{\pi^j} + \sum_{i=1}^{n} b_i f_{x_i}(r) \equiv 0 \pmod{\pi}$$

Analyzing the polynomial $f(x_1, x_2, ..., x_u) = N - \prod_{i=1}^{u} x_i$ with the Hensel's lemma we obtained the following results. Let $r = (r'_1, r'_2, ..., r'_u)$ be a solution for $f(x_1, x_2, ..., x_u) \equiv 0 \pmod{2^j}$ where a root for $f(x_1, x_2, ..., x_u) \equiv 0 \pmod{2^{j+1}}$ is defined as $r = (r'_1 + 2^j b_1, r'_2 + 2^j b_2, ..., r'_u + 2^j b_u)$ where b_i represents a bit $r_i[j]$ such that

$$\left(N - \prod_{i=1}^{u} r'_i\right)[j] \equiv \sum_{i=1}^{u} r_i[j] \pmod 2. \tag{1}$$

[3] N is the product of u primes with the same bit length, as in the Basic RSA.

It is an equation modulus 2. If all values $r_i[j]$ are considered as unknowns, it is an equation modulus $\pi = 2$ with u variables. An equation with u variables has a total number of solutions equal to $\pi^{u-1} = 2^{u-1}$. In other words, we get a maximum of 2^{u-1} roots for $f(x_1, x_2, .., x_u) \pmod{2^{j+1}}$ from a root of $f(x_1, x_2, .., x_u) \pmod{2^j}$. Furthermore, if a fraction of random bits are known, this number of roots decreases, as we show below.

Let $root[j]$ be a set of all possible roots of the polynomial $f(x_1, x_2, .., x_u) \pmod{2^{j+1}}$. $root[0]$ is a set of single elements because the r_i's are primes hence the single root of $f(x_1, x_2, .., x_u) \pmod 2$ is $\langle 1_1, 1_2, .., 1_u \rangle$. With the definition of $root[j]$ we developed the following algorithm to factor a multiprime modulus N. In Algorithm 1 there are the following inputs: a big integer N, an integer u that is the number of prime factors of N. A fraction δ of random bits of the primes is known and given as $\tilde{r_1}, \tilde{r_2}...\tilde{r_n}$

Algorithm 1. Factoring N

Input: $N, u, \langle \tilde{r_1}, \tilde{r_2}, ..., \tilde{r_u} \rangle$
Output: $root[\frac{n}{u}]$ where $\langle r_1, r_2, ..., r_u \rangle$ is in $root[\frac{n}{u}]$

1 $root[0] = [\langle 1_1, 1_2, ..., 1_u \rangle]$;
2 $j = 1$;
3 **for each** $\langle r'_1, r'_2, ..., r'_u \rangle$ **in** $root[j-1]$ **do**
4 **for all possible** *(as explained below (*))* $\langle r_1[j], r_2[j], ..., r_u[j] \rangle$ **do**
5 **if** $(N - \prod_{i=1}^{u} r'_i)[j] \equiv \sum_{i=1}^{u} r_i[j] \pmod 2$ **then**
6 $root[j].add(\langle r'_1 + 2^j r_1[j], r'_2 + 2^j r_2[j], ..., r'_u + 2^j r_u[j] \rangle)$
7 **if** $j < \frac{n}{u}$ **then**
8 $j := j + 1$;
9 go to step 3;
10 **return** $root[\frac{n}{u}]$;

The algorithm begins with a set $root[0]$ of single roots $\langle 1_1, 1_2, ..., 1_u \rangle$. (*) In Line 4 for each root $\langle r'_1, r'_2, ..., r'_u \rangle \in root[j-1]$ we generate all permutations for bits $\langle r_1[j], r_2[j], ..., r_u[j] \rangle$. The number of values of $r_i[j]$ is one only if this bit is known in $\tilde{r_i}$, otherwise there are two values, 0 or 1. Each result $\langle r_1[j], r_2[j], ..., r_u[j] \rangle$ of this permutation is analyzed in Line 5 by the Hensel's Eq. (1); it is added to the set $root[j]$ if the equivalence is true. This procedure is done until the set $root[\frac{n}{u}]$ obtained contains the prime factors of N.

3 Behavior and Complexity of the Algorithm to Factor N

We developed a brute-force search algorithm lifting all possible roots for the polynomial $f(x_1, x_2, ..., x_u) \pmod{2^j}$ for $1 \leq j \leq \frac{n}{u}$. This behavior is shown in Fig. 1 where we can see the root $r = \langle r_1, r_2, ..., r_u \rangle$ as a gray double circle. It was lifted in each level $j \in [1, \frac{n}{u}]$ from one root in $root[0]$. These roots are defined as good roots and are shown as no-color double circle. One good root in each level j always lifts the good root for the next level $j + 1$. Each incorrect root is represented as a single line circle.

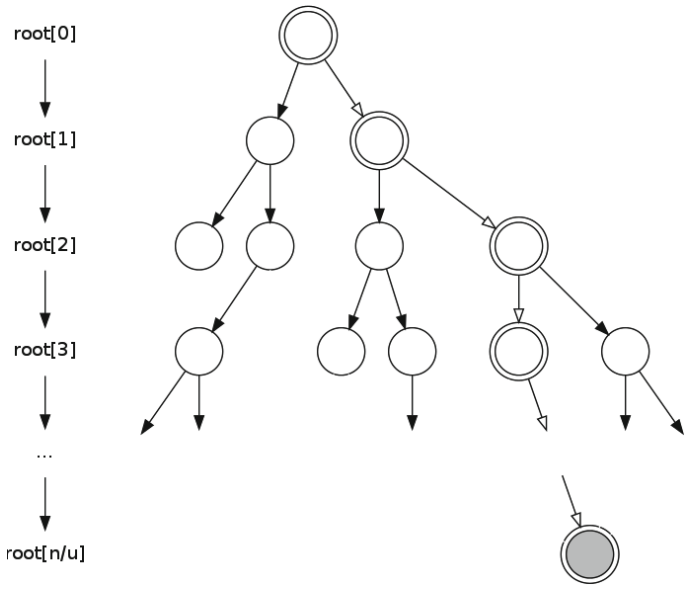

Fig. 1. (Factoring N) Behavior of Algorithm 1 to factor N

We know we have a number of $\frac{n}{u} + 1$ good roots in all executions of Algorithm 1, but the behavior of the algorithm is determined by the number of the lifted incorrect roots. Therefore the analysis was done with respect to incorrect roots, and to do this analysis, we defined the following random variables:

- Let G be the random variable for the number of incorrect roots lifted from a good root.
- Let B be the random variable for the number of incorrect roots lifted from an incorrect root.
- Let X_j be the random variable for the number of incorrect roots lifted at level j.

3.1 Number of Incorrect Roots Lifted from a Good Root (G)

All cases that may occur are described in the following Table where we have one good root $\langle r'_1, r'_2, ..., r'_u \rangle$ of $root[j-1]$ and let's define h as the number of unknown bits in $\langle r_1[j], r_2[j], ..., r_u[j] \rangle$.

There are h cases ($1 \leq h \leq u$ in Eq. (1)). It is an equation modulus 2 with h variables where the number of solutions is 2^{h-1}. Hence we get a total of 2^{h-1} roots for $root[j]$. In the case $h = 0$ we obtain a single root and it is the good root of $root[j]$ because it is built from the good root of $root[j-1]$ and all known bits are of the correct root. But we do not have the number of incorrect roots that were lifted. The number of incorrect roots lifted are in the third column

Table 1. (Factoring N) Number of incorrect roots lifted from a good root.

Cases	Number of lifted roots	Number of lifted incorrect roots
$1 \leq h \leq u$	2^{h-1}	$2^{h-1} - 1$
$h = 0$	1	0

of Table 1 and are the values of the second column decreased by 1 (because the good root in $root[j]$ is always lifted by the good root in $root[j-1]$). Therefore we can define the expected value of G as follows:

$$\mathbb{E}[G] = \sum_{h=1}^{u} (2^{h-1} - 1) \binom{u}{h} (1-\delta)^h \delta^{u-h} \tag{2}$$

where the probability of occuring h unknown bits in a set of u bits is $\binom{u}{h}(1-\delta)^h\delta^{u-h}$.

3.2 Number of Incorrect Roots Lifted from an Incorrect Root (B)

Before computing the number of incorrect roots lifted from an incorrect root let's define c_1 as the computed value of

$$c_1 = \left(N - \prod_{i=1}^{u} r'_i \right)[j]$$

from the good root in $root[j-1]$. With this definition, we can observe two kinds of incorrect roots in $root[j-1]$: either the computed value of $(N - \prod_{i=1}^{u} r'_i)[j]$ is equal to c_1 or is different from c_1 (denoted by $\overline{c_1}$). Considering all this, we analyzed all cases for computing the number of incorrect roots lifted from an incorrect root in the next Table.

The cases where $1 \leq h \leq u$, the values c_1 and $\overline{c_1}$ are not important because in Hensel's Eq. (1) there is a modular equation of h variables, and a total of 2^{h-1} incorrect roots. For the case of c_1, the incorrect root is going to act like a good root, hence for $h = 0$ the incorrect root lifts an incorrect root. But in the case of $\overline{c_1}$ the incorrect root is dropped because we have a contradiction.

The computed value of $(N - \prod_{i=1}^{u} r'_i)[j]$ in an incorrect root can be 0 or 1 with probability $\frac{1}{2}$. The probabilities are $P((N - \prod_{i=1}^{u} r'_i)[j] = 1) = \frac{1}{2}$ and

Table 2. (Factoring N) Number of incorrect roots lifted from an incorrect root.

Cases	$(N - \prod_{i=1}^{u} r'_i)[j] = c_1$	$(N - \prod_{i=1}^{u} r'_i)[j] = \overline{c_1}$
$1 \leq h \leq u$	2^{h-1}	2^{h-1}
$h = 0$	1	0

$P((N - \prod_{i=1}^{u} r'_i)[j] = 0) = \frac{1}{2}$. In other words, the probabilities are the same because c_1 and $\overline{c_1}$ represent the value of one bit. Hence we have $P((N - \prod_{i=1}^{u} r'_i)[j] = c_1) = \frac{1}{2}$ and $P(N - \prod_{i=1}^{u} r'_i)[j] = \overline{c_1}) = \frac{1}{2}$.

Analyzing Table 2 and with the probabilities defined above, we can determine the expected value of B.

$$\mathbb{E}[B] = \sum_{h=1}^{u} 2^{h-1} \binom{u}{h} (1-\delta)^h \delta^{u-h} \frac{1}{2} + \sum_{h=1}^{u} 2^{h-1} \binom{u}{h} (1-\delta)^h \delta^{u-h} \frac{1}{2} + \binom{u}{0} (1-\delta)^0 \delta^{u-0} \frac{1}{2} \tag{3}$$

$$= \frac{(2-\delta)^u}{2}$$

3.3 Number of Incorrect Roots Lifted at Level j (X_j)

The expected value of the discrete random variable X_j is defined in the following recursion:

$$\mathbb{E}[X_j] = \mathbb{E}[X_{j-1}]\mathbb{E}[B] + \mathbb{E}[G]$$

because the number of incorrect roots at level j is equal to the number of incorrect roots lifted from the incorrect roots at level $j-1$ plus the number of incorrect roots lifted from the only one good root at level $j-1$. And we have $\mathbb{E}[X_1] = \mathbb{E}[G]$ because at level 0 there is no incorrect root, hence we can compute the closed form as follows:

$$\mathbb{E}[X_j] = \mathbb{E}[G] \frac{1 - \mathbb{E}[B]^j}{1 - \mathbb{E}[B]}. \tag{4}$$

3.4 Complexity of the Algorithm to Factor

The algorithm to factor N should run up to level $\frac{n}{u}$ and return the prime factors of N, thus the expected value of the number of incorrect roots analyzed by Algorithm 1 is defined as:

$$\mathbb{E}\left[\sum_{j=1}^{\frac{n}{u}} X_j\right] = \sum_{j=1}^{\frac{n}{u}} \mathbb{E}[X_j] \qquad \text{Property of expected value}$$

$$= \sum_{j=1}^{\frac{n}{u}} \mathbb{E}[G] \frac{1 - \mathbb{E}[B]^j}{1 - \mathbb{E}[B]} \qquad \text{Definition (4)}$$

$$= \frac{\mathbb{E}[G]}{1 - \mathbb{E}[B]} \sum_{j=1}^{\frac{n}{u}} 1 + \frac{\mathbb{E}[G]}{\mathbb{E}[B] - 1} \sum_{j=1}^{\frac{n}{u}} \mathbb{E}[B]^j$$

$$= \frac{\mathbb{E}[G]}{1 - \mathbb{E}[B]} \frac{n}{u} + \frac{\mathbb{E}[G]\mathbb{E}[B](\mathbb{E}[B]^{n/u} - 1)}{(\mathbb{E}[B] - 1)^2}.$$

The equation above is exponential on n and on $\mathbb{E}[B]$ but it can be bounded as follows. For values of $\mathbb{E}[B] > 1$ this function is actually exponential

$(\lim_{n\to\infty} \mathbb{E}[B]^{n/u} = \infty)$ but for values $\mathbb{E}[B] < 1$ we get $\lim_{n\to\infty} \mathbb{E}[B]^{n/u} < 1$. Therefore the expected number of analyzed incorrect roots is bounded by a linear equation on n for values $\mathbb{E}[B] < 1$.

$$\mathbb{E}\left[\sum_{j=1}^{\frac{n}{u}} X_j\right] = \frac{\mathbb{E}[G]}{1-\mathbb{E}[B]}\frac{n}{u} + \frac{\mathbb{E}[B]\mathbb{E}[G](\mathbb{E}[B]^{n/u}-1)}{(\mathbb{E}[B]-1)^2} \leq \frac{\mathbb{E}[G]}{1-\mathbb{E}[B]}\frac{n}{u} \text{ for } \mathbb{E}[B] < 1$$

In summary, we can factor the modulus $N = \prod_{i=1}^{u} r_i$ in polynomial time, given a δ fraction of random bits of its prime factors greater than $2-2^{\frac{1}{u}}$ (by Definition (3) $\mathbb{E}[B] = \frac{(2-\delta)^u}{2} < 1$) because the expected number of analyzed incorrect roots is $O(n)$.

$$\begin{aligned} \mathbb{E}[B] = \frac{(2-\delta)^u}{2} &< 1 \\ (2-\delta)^u &< 2 \\ 2-\delta &< 2^{\frac{1}{u}} \\ \delta &> 2-2^{\frac{1}{u}} \end{aligned}$$

Some results from this analysis are, to factor in polynomial time an integer:

- $N = \prod_{i=1}^{2} r_i$, $\delta \geq 0.59 (2-2^{\frac{1}{2}} \approx 0.5858)$ fraction of the bits of its prime factors is needed.
- $N = \prod_{i=1}^{3} r_i$, $\delta \geq 0.75 (2-2^{\frac{1}{3}} \approx 0.7401)$ fraction of the bits of its prime factors is needed.
- $N = \prod_{i=1}^{4} r_i$, $\delta \geq 0.82 (2-2^{\frac{1}{4}} \approx 0.8108)$ fraction of the bits of its prime factors is needed.

4 Implementation and Performance

The algorithm to factor N was implemented in 300 lines of code using the program language C with the library Relic-toolkit [1] that is focused for the implementation of cryptosystems, and was executed on a processor Intel Core I3 2.4 Ghz with 3 Mb of cache and 4 Gb of DDR3 Memory.

The experiments were executed for integers N ($n = 2048$ bits) and they were the product of u primes, $2 \leq u \leq 4$. For each value δ a total of 100 integers N were generated, and for each integer N 100 inputs with a fraction δ of bits of its primes were generated. The results of the total of 550000 experimental runs are shown in Tables 3, 4, 5 and Fig. 2.

With the results obtained by Heninger and Shacham in [5] we can say that the number of analyzed incorrect roots in an experiment has a low probability to surpass 1 million. In our experiments we did the same to avoid trashing, hence we canceled all experiments that surpassed one million analyzed incorrect roots.

In Tables 3, 4 and 5 we have in the second and third column the minimum and maximum analyzed incorrect roots, respectively. The fourth and sixth column

Table 3. Results of total examined roots by the algorithm to factor $N = \prod_{i=1}^{2} r_i$, 2048 bits.

δ	Number of analyzed roots			# Exp (> 1M)	Average time (s)
	Minimum	Maximum	Average		
0.61	1983	945728	4949	0	0.115277
0.60	2233	789608	6344	0	0.119484
0.59	2411	928829	8953	2	0.187600
0.58	2631	987577	14736	7	0.250224
0.57	3436	994640	24281	29	0.531079
0.56	4012	998414	42231	134	0.722388

Table 4. Results of total examined roots by the algorithm to factor $N = \prod_{i=1}^{3} r_i$, 2048 bits.

δ	Number of analyzed roots			# Exp (> 1M)	Average time (s)
	Minimum	Maximum	Average		
0.77	1128	171142	2022	0	0.033884
0.76	1205	323228	2777	0	0.049238
0.75	1380	177293	3723	1	0.099373
0.74	1607	571189	5941	1	0.197553
0.73	1681	999766	11470	11	0.281414
0.72	2087	983404	23826	50	0.995017

Table 5. Results of total examined roots by the algorithm to factor $N = \prod_{i=1}^{4} r_i$, 2048 bits.

δ	Number of analyzed roots			# Exp (> 1M)	Average time (s)
	Minimum	Maximum	Average		
0.84	716	31447	1245	0	0.024748
0.83	823	67456	1649	0	0.040714
0.82	931	217391	2424	0	0.063754
0.81	1044	558521	4408	1	0.111688
0.80	1249	994386	9571	14	0.236320
0.79	1632	972196	24085	58	0.609435

contain the average number and the average time of analyzed incorrect roots in all experiments that did not surpassed one million of incorrect roots. The fifth column contains the number in all experiments that were canceled because it surpassed one million incorrect roots.

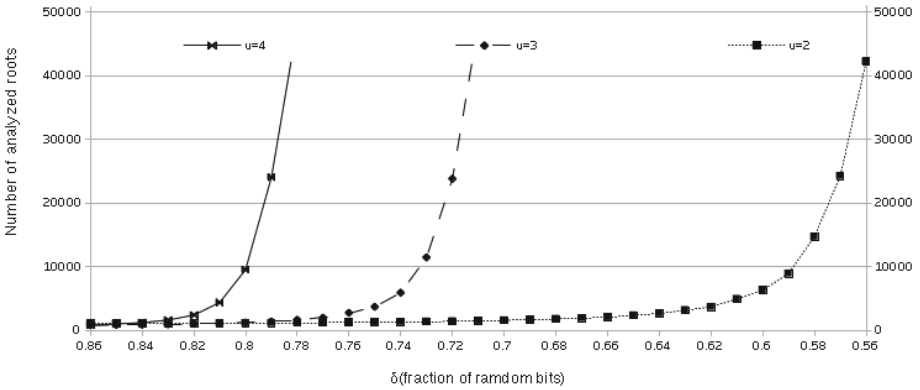

Fig. 2. (Factoring N) Average number of analyzed roots by the algorithm to factor $N = \prod_{i=1}^{u} r_i$ where $2 \leq u \leq 4$

Table 6. Comparison of basic, 2-power and 3-primes modulus N.

Cases			
	Basic Modulus N	2-power Modulus N	3-primes Modulus N
	$N = r_1 r_2$	$N = r_1^2 r_2$	$N = \prod_{i=1}^{3} r_i$
Random bits	$\delta \geq 0.59$	$\delta \geq 0.59$	$\delta \geq 0.75$

For all experiments, we obtained an average time less than 1 second to factor N. And only 305 experiments were canceled because they had over one million of analyzed incorrect roots. For the rest of experiments, the algorithm always returned the prime factors of N.

Figure 2 shows the average number of analyzed roots of our experiments to factor N with $n = 2048$ bits. We observe the exponential growth of $\mathbb{E}\left[\sum_{j=1}^{\frac{n}{u}} X_j\right]$ for values of δ lower than $2 - 2^{\frac{1}{u}}$ ($\mathbb{E}[B] > 1$) for u in $[2, 4]$.

5 Concluding Remarks

We designed an algorithm to factor a multiprime integer N based on Hensel's lemma.

The statistical analysis shows that factoring a multiprime modulus N for u greater than 2 offers more security (adversarially thinking) than for $u = 2$, assuming a fraction of random bits of its primes is given. Table 6 shows a comparison done for Basic, 2-power and 3 multiprime modulus N.

Using a multi-power $N = r_1^m r_2$ allows a faster decryption process than using a basic $N = r_1^1 r_2$. There is no advantage to factor a multi-power modulus N with random bits because for any value of $m \geq 1$ we always need a fraction $\delta \geq 0.59$ of random bits. Due to page number restrictions, our analysis and

implementation results to factor a general multipower modulus $N = r_1^m r_2$ (for the cases even and odd m) with random bits of its prime factors are given in the extended version of this paper to be published elsewhere.

The advantages of using a multiprime modulus N are: the decryption is faster, and with respect to cold boot attacks, the attacker needs more than $2^{\frac{1}{u}} - 2^{\frac{1}{2}}$ fraction of random bits if a basic modulus N is used. Therefore if $u > 2$, to factor N is harder but there is a limit for the value u. When u is large the modulus N can be factored without extra information using the algorithm called Number Field Sieve (NFS)[4] or by an elliptic curve method (ECM)[5].

Acknowledgment. We thank anonymous referees who pointed out the work by Kogure et al. [9].

References

1. Aranha, D.F., Gouvêa, C.P.L.: RELIC is an Efficient Library for Cryptography. http://code.google.com/p/relic-toolkit/
2. Bar-El, H.: Introduction to side channel attacks. White Paper, Discretix Technologies Ltd. (2003)
3. Boneh, D.: Twenty years of attacks on the RSA cryptosystem. Not. AMS **46**(2), 203–213 (1999)
4. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold-boot attacks on encryption keys. Commun. ACM **52**(5), 91–98 (2009)
5. Heninger, N., Shacham, H.: Reconstructing RSA private keys from random key bits. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 1–17. Springer, Heidelberg (2009)
6. Herrmann, M., May, A.: Solving linear equations modulo divisors: on factoring given any bits. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 406–424. Springer, Heidelberg (2008)
7. Hinek, M.J.: On the security of multi-prime RSA. J. Math. Cryptol. **2**(2), 117–147 (2008)
8. Jonsson, J., Kaliski, B.: Public-key cryptography standards (PKCS)# 1: Rsa cryptography specifications version 2.1. Technical report, RFC 3447, February 2003
9. Kogure, J., Kunihiro, N., Yamamoto, H.: Generalized security analysis of the random key bits leakage attack. In: Yung, M., Jung, S. (eds.) WISA 2011. LNCS, vol. 7115, pp. 13–27. Springer, Heidelberg (2012)
10. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
11. Ptacek, T. : Recover a private key from process memory (2006). http://chargen.matasano.com/chargen/2006/1/25/recover-a-private-key-from-process-memory.html
12. Quisquater, J.J., Couvreur, C.: Fast decipherment algorithm for RSA public-key cryptosystem. Electron. Lett. **18**(21), 905–907 (1982)

[4] It is an algorithm to factor an integer N with a very good performance.
[5] It is an algorithm to compute a non-trivial factor of N.

13. Shamir, A., van Someren, N.: Playing 'Hide and Seek' with stored keys. In: Franklin, M.K. (ed.) FC 1999. LNCS, vol. 1648, pp. 118–124. Springer, Heidelberg (1999)
14. Skorobogatov, S.: Low temperature data remanence in static ram. University of Cambridge Computer Laborary Technical Report 536 (2002)
15. Takagi, T.: Fast RSA-type cryptosystem modulo $p^k q$. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 318–326. Springer, Heidelberg (1998)