Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/092L28

Probabilistic Analysis on JPV Algorithm and Improving It using GCD Function

Hosung Jo[1] *and Heejin Park*[2,*]

[1] Department of Electronics and Computer Engineering, Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul 133-791, Korea
[2] Department of Computer Science and Engineering, Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul 133-791, Korea

Received: 8 Jun. 2014, Revised: 6 Aug. 2014, Accepted: 8 Aug. 2014
Published online: 1 Apr. 2015

Abstract: JPV algorithm, proposed by Joye et al. was predicted to be faster than the combined prime generation algorithm but it runs slower in practice. This discrepancy is because only the number of Fermat test calls was compared in estimating its total running time. We present a probabilistic analysis on the total running time of JPV algorithm. This analysis is very accurate and corresponds to the experiment with only 1-2% error. Furthermore, we propose an improved JPV algorithm that uses GCD function. It is faster than JPV algorithm and similar to the combined algorithm with the same space requirement.

Keywords: Prime generation, Primality test, Public-key cryptosystem, GCD function

1 Introduction

Generating large primes is important to enhance the security strength of public-key cryptosystem [1,2] such as RSA [3], ElGamal [4], and DSS [5] because if a prime used in cryptosystem is bigger, cryptosystem is more secure. However, generating large primes takes long time, so we need to generate large primes efficiently. Prime generation [6] consists of random number generation and primality test. Since the primality test is much-more time-consuming than the random number generation, reducing the running time of primality test is more important in developing an efficient prime generation algorithm.

Primality tests are divided into two categories; deterministic primality tests and probabilistic primality tests. Deterministic primality tests such as trial division [7], Pocklington's test [8], elliptic curve analogue [9], Jacobi sum test [10], Maurer's algorithm [11] and Shawe-Taylor's algorithm [12] certify that a random number is a prime with probability 1. Probabilistic primality tests such as Fermat Test [13], Miller-Rabin test [14], Solovay-Strassen test [15], Frobenius-Grantham primality test [16] and Lehmann primality test [17] certify that a random number is a prime with high probability that is very close to 1.

Practically, two or more primality tests are combined to speed up prime generation. A combination of trial division and Miller-Rabin test is widely used. Trial division divides an n-bit random number r to all primes at most $\sqrt{r}$. Miller-Rabin test checks whether one of following condition is satisfied when $r-1=2^j q$ $(0 \leq j \leq k)$; If $j=0$, $a^q \bmod r = 1$ or $a^{2^j q} \bmod r = n-1$. Maurer [11] proposed a probabilistic analysis of an expected running time for the combined algorithm. He also showed how to compute the optimal value of g (g_{opt} hereafter) which makes the combined algorithm fastest.

Joye et al. [18] introduced JPV algorithm which removes trial division from the combined algorithm. JPV algorithm has 2 characteristics: The first one is that it generates an odd random number which is relatively prime to every prime less than some bound k. The second one is that it generates a new random number from the previous one by simple computation, instead of generating a random number again. JPV algorithm claimed that it is faster than the combined algorithm by 30 to 40%. However, Joye et al. did not compare the total running time of each algorithm but the number of Fermat test calls because there was no probabilistic analysis on the running time of JPV algorithm. Thus, developing the probabilistic analysis for the running time of JPV algorithm is required for more accurate comparison.

* Corresponding author e-mail: hjpark@hanyang.ac.kr

Table 1: Denotations about the running time

	Running Time	Time for 1 execute
Random Number Generation	T_{RND}	T_{rnd}
Trial Division	T_{TD}	T_{td}
Miller-Rabin Test	T_{MR}	T_{mr}

In this paper, we first propose a probabilistic analysis on the expected running time of JPV algorithm and compare the expected running time of JPV algorithm with that of the combined algorithm. According to our analysis, JPV algorithm is slower than the combined algorithm in generating a 512-bit prime and the experimental results correspond to this analysis. In addition, we propose an improved JPV algorithm and it shows similar performance with the combined algorithm when the same size of memory space is used.

This paper is organized as follows. Section II introduces the combined algorithm and JPV algorithm. In Section III, we probabilistically analyze the expected running time of JPV algorithm. We also compare the combined algorithm and JPV algorithm. In addition, we introduce an improved JPV algorithm. We conclude in Section IV.

2 Previous Work

2.1 The Combined Algorithm

The combined algorithm consisting of random number generation, trial division and probabilistic primality test is as follows.

TD-MR combination (n,k)

1.Random Number Generation
  -Generate an n-bit odd random number r.
2.Trial division on r with k small primes
  -Divides r by k small primes.
  -If r is divided by any prime, go to Step 1.
3.Miller-Rabin test on r
  -Perform Miller-Rabin test on r.
  -If r passes, return r as a prime.
  -Otherwise, go to Step 1.

Maurer [11] introduced a probabilistic analysis of the expected running time of the combined test. Let N_T be the number of generated random numbers until a prime is found. Let T_{RND}, T_{TD}, and T_{MR} be the average running times of random number generation, trial division, and Miller-Rabin test, respectively.

Then, when n-bit prime is generated with k primes, the total run time, $T(n,k)$, is as follows.

$$T(n,k) = N_T \cdot (T_{RND} + T_{TD} + T_{MR}) \quad (1)$$

If the bit-length of r is n, $N_T = \frac{n\ln 2}{2} \approx 0.347n$. Let T_{td} and T_{mr} be the running times of one division and one Mill-Rabin test, respectively. Let k be the number of small primes used in the trial division and p_i be the ith odd prime number so that $p_1 < p_2 \ldots < p_k$. Then, T_{RND}, T_{TD} and T_{MR} are as follows.

$$T_{RND} = 1 \cdot T_{rnd} \quad (2)$$

$$T_{TD} = T_{td}(1 + \sum_{j=1}^{k}\prod_{i=1}^{j}(1 - \frac{1}{p_i})) \quad (3)$$

$$T_{MR} = T_{mr}(\prod_{i=1}^{k}(1 - \frac{1}{p_i})) \quad (4)$$

Therefore, $T(n,k)$ is as follows.

$$T(n,k) = \frac{n\ln 2}{2}(T_{rnd} + T_{td}(1 + \sum_{j=1}^{k}\prod_{i=1}^{j}(1 - \frac{1}{p_i})) + T_{mr}(\prod_{i=1}^{k}(1 - \frac{1}{p_i}))$$

The optimal number of primes that makes the run time fastest is as follows.

$$g_{opt} = \frac{T_{mr}}{T_{div}} \quad (5)$$

2.2 JPV algorithm

Joye et al. proposed JPV algorithm that does not use the trial division. JPV algorithm consists of 5 steps; precomputation, invertible number generation, candidate generation, primality test, and invertible number regeneration.

1.**Precomputation**
Compute integer values η, Π, ρ, and $\lambda(\Pi)$. The range of a n-bit random number is $2^{n-1}+1 \le q \le 2^n - 1$. We define W_{max} as $2^n - 1$ and W_{min} as $2^{n-1}+1$. η is the product of k different small primes and Π and ρ are multiples of η and also satisfy inequalities $\Pi \le W_{max} - W_{min}$ and $\rho \ge W_{min}$. The function λ is the Carmichael function [19] that is computed in the following way. If $\Pi = p_1 p_2 \ldots p_k$, $\lambda(\Pi)$ is the least common multiple of each $\lambda(p_i^{\delta_i})$. When p_i is odd, $\lambda(p_i^{\delta_i}) = p_i^{\delta_i - 1}(p_i - 1)$. When p_i is even and $\delta_i \ge 3$, $\lambda(2^{\delta_i}) = 2^{\delta_i - 2}$. When p_i is even and $\delta_i > 3, \lambda(2) = 1$ and $\lambda(4) = 2$.

2.**Invertible number generation**
Generate an integer c that is relatively prime to Π. First, generate a random number c smaller than Π and exam whether c is relatively prime to Π. If c and Π are relatively prime, return c. Otherwise, add 1 to c

Table 2: Precomputed values for the 512-bit prime generation

	Precomputed values
η	b16b d1e0 84af 628f e508 9e6d abd1 6b5b 80f6 0681 d6a0 92fc b1e8 6d82 876e d719 2100 0bcf dd06 3fb9 081d fd07 a021 af23 c735 d52e 63bd 1cb5 9c93 cbb3 98af d
Π	1,729 $\cdot\eta$
ρ	4,120 $\cdot\eta$
$\lambda(\Pi)$	1dc6 c203 d4cc 7800 33f9 c5d8 d97a a246 8a54 e370 0

and exam again. This step is repeated until finding c which is relatively prime to Π. In this algorithm, if $c^{\lambda(\Pi)} \bmod \Pi = 1$, c is relatively prime to Π.

3.**Candidate generation**
Generate a candidate r for Fermat test. The candidate r is the sum of c and ρ. If both c and ρ are odd, add η to r to make r even.

4.**Primality test**
Perform Fermat test on r. If r is a prime, return r and terminate. Otherwise, go to step 5.

5.**Invertible number regeneration**
Generate a new invertible number $c' = 2c \bmod \Pi$. After generating c', return to step 3 and c' replaces c to generate r.

JPV algorithm was compared with combined algorithm about the number of Fermat test calls [18]. According to the comparison result, JPV algorithm is 30 to 40% faster than combined algorithm when 10 small primes are used. In addition, as the size of a generated prime is getting larger, the gap between JPV algorithm and the combined algorithm is wider.

However, this comparison is not appropriate in two ways. One is that combined algorithm uses only 10 small primes. Usually, as the number of small primes used in trial division increases, the number of Miller-Rabin calls decreases. Therefore, using only 10 primes for trial division is not relevant. The other is that comparing only the number of Fermat test calls is not a good metric for the real running time. Because as the number of primes in the trial division increases, the running time of the trial division is increasing and thus the total running time would be increased.

3 Contribution

We first probabilistically analyze JPV algorithm in Section 3.1. Then we compare the running time of JPV algorithm with that of the combined algorithm in Section 3.2. Finally, we introduce our improved JPV algorithm and its running time in Section 3.3. Note that we will use Miller-Rabin test as the probabilistic primality test instead of Fermat test.

3.1 Probabilistic Analysis of JPV Algorithm

The total running time of JPV algorithm can be estimated by the sum of the running times of 2-5 steps because step 1 is precomputed. Let T_i denote the running time of step i and N_i the number of iterations of step i. We first consider the number of iterations of each step. Since step 2 is performed only one time, N_2=1. The iteration number of step 3 and 4 are same, thus $N_3 = N_4$. Step 5 is performed when Miller-Rabin test is failed, so $N_5 = N_4 - 1$. Then T_{JPV} is represented as follows.

$$T_{JPV} = T_2 + (T_3 + T_4)N_4 + T_5(N_4 - 1) \tag{6}$$

We consider the running time of each step. T_2 is the sum of the running time to test that c is relatively prime to Π and the running time to regenerate c. Let T_{rnd} denote the running time to generate c, T_{lam} the running time of computing $(c^{\lambda(\Pi)} \bmod \Pi)$, T_{add} the running time of adding 2 integers, and N_{lam} the number of $(c^{\lambda(\Pi)} \bmod \Pi)$ computations. Then, the total running time of step 2 is as follows.

$$T_2 = T_{rnd} + (T_{lam} + T_{add})N_{lam} - T_{add} \tag{7}$$

T_3 is the time to generate an n-bit odd candidate r from c. In step 3, if r is odd, one addition is necessary. Otherwise, two addition are necessary. Since the probability that r is odd is 1/2, 1.5 addition is executed on average.

$$T_3 = 1.5 \cdot T_{add} \tag{8}$$

T_4 is the running time of Miller-Rabin test on the candidates those are not divided by k small primes,

$$T_4 = T_{mr} \tag{9}$$

T_5 is the time to regenerate an invertible number c when r is not a prime. In step 5, $(c \leftarrow 2c \bmod \Pi)$ can be computed by 1.5 addition on average as follows. First, $(c + c)$ is computed and if $2c \geq \Pi$, it computes $(2c - \Pi)$. Because the probability that $2c$ is bigger than Π is 1/2 and normally the running time of an addition and a subtraction are similar, 1.5 addition is required on average in step 5.

$$T_5 = 1.5 \cdot T_{add} \tag{10}$$

Now, we know T_2, T_3, T_4, and T_5, then T_{jpv} is like as follows.

$$T_{JPV} = T_{rnd} + (N_{lam} + 3N_4 - 2.5)T_{add} + N_{lam}T_{lam} + N_4T_{mr} \tag{11}$$

In the equation (11), $T_{rnd}, T_{lam}, T_{add}$, and T_{mr} are different by the experimental machine and environment. Therefore, those can be measured by experiments. N_{lam} and N_4 can be computed by probabilistic analysis. N_{lam} is the number of computations of $(c^{\lambda(\Pi)} \bmod \Pi)$ until we find c relatively prime to Π. Since c and Π are relatively prime and $\Pi = p_1 p_2 \dots p_k$, the probability that c and Π are relatively prime is $\prod_{i=1}^{k}(1-\frac{1}{p_i})$. Since the expected number of evaluating $(c^{\lambda(\Pi)} \bmod \Pi)$ is the inverse of the probability of it, N_{lam} is as follows.

$$N_{lam} = \prod_{1\le i\le k}(\frac{1}{1-\frac{1}{p_i}}) = \prod_{1\le i\le k}(\frac{p_i}{p_i-1}) \qquad (12)$$

N_4 is the number of generated candidate r thus it is the inverse of the probability of r being a prime. The probability of r being a prime can be obtained using the conditional probability as follows. Let A be the event that r and Π are relatively prime and B the event that r passes Miller-Rabin test. Then, the probability that r is a prime is the conditional probability $P(B|A)$. Since $P(B|A) = P(B \cap A)/P(A)$ and $P(B \cap A) = P(B)$, $P(B|A) = P(B)/P(A)$ where $\Pi = p_1 p_2 \dots p_k$ and $P(A) = \prod_{i=1}^{k}\left(1-\frac{1}{p_i}\right)$. Because $P(B)$ is an inverse of average numbers of trials until finding a prime, $P(B) = \frac{1}{0.347n}$. Therefore, $P(B|A) = \frac{1}{0.347n\prod_{i=1}^{k}\left(1-\frac{1}{p_i}\right)}$ and N_4 is an inverse of $P(B|A)$. So N_4 is represented as follows.

$$N_4 = 0.347n \prod_{1\le i\le k}(1-\frac{1}{p_i}) \qquad (13)$$

Finally, we can get Theorem 1 by gathering all the information of equation (11).

Theorem 1 The running time of JPV algorithm for generating an n-bit prime is as follows.

$$\begin{aligned} T_{JPV}(n,k) = {} & T_{rnd} + \prod_{1\le i\le k}(\frac{p_i}{p_i-1})T_{lam} \\ & + (1.041n \prod_{1\le i\le k}(1-\frac{1}{p_i}) \\ & + \prod_{1\le i\le k}(\frac{p_i}{p_i-1}) - 2.5))T_{add} \\ & + 0.347n \prod_{1\le i\le k}(1-\frac{1}{p_i})T_{mr} \end{aligned}$$

where $p_1, p_2, \dots, p_k$ are factors of Π, T_{rnd} is the time required to generate a random number , T_{add} is the time required for an addition, T_{lam} is the running time of $(c^{\lambda(\Pi)} \bmod \Pi)$, and T_{mr} is the running time of Miller-Rabin test.

Table 3: Measured values

	Measured time (ns)
T_{add}	430
T_{rnd}	15,544
T_{mr}	4,734,181
T_{lam}	1,500,133

Table 4: Comparison of the expected and measured time of JPV algorithm

	Expected (ns)	Measured (ns)	Error (%)
T_{JPV}	166,239,125	168,336,171	1.2

3.2 Comparing theoretical results to experimental results

In order to find out how accurate our probabilistic analysis is, we compute the expected running time and compare the result with the measured running time when a 512-bit prime is generated on a Pentium 4 3.0Ghz with 1GB main memory. The programming environment for implementation is JAVA JDK 5.0 in OpenSSL [20] and GNUCrypto [21].

In order to compute the expected running time of JPV algorithm by Theorem 1, T_{rnd}, T_{lam}, T_{mr}, and T_{add} should be measured. When a 512-bit prime is generated, we used the values η, Π, ρ, and $\lambda(\Pi)$ as shown in Table 2, which are provided by Joye et al. [18]. Table 3 shows the measured values T_{add}, T_{rnd}, T_{mr}, and T_{lam}. We performed each operation 1 million times for 512-bit random numbers and measured the total running time. Finally we compute the average of the total running time.

Then, we compute the expected running time and compare the expected running time and measured running time of JPV algorithm. We generated 512-bit prime for 1,000,000 times. Table 4 shows the comparison results that the expected running time and the measured running time of the JPV algorithm are very similar, which implies that our probabilistic analysis is quite accurate.

3.3 JPV algorithm vs. Combined algorithm

In this subsection, we compare the running time of JPV algorithm with that of the combined algorithm. First, we compare JPV algorithm with the optimized combined algorithm that uses g_{opt}. Then, we compare JPV algorithm with the combined algorithm when the combined algorithm uses the memory size as same as JPV algorithm requires (This will be called a space-limited combined algorithm hereafter).

The expected running time of the optimized combined algorithm is computed by the probabilistic analysis proposed by Maurer. In order to compute the expected

Table 5: Comparison result of algorithms

	Expected(ns)	Measured(ns)	Space(bit)
T_{JPV}	166,239,125	168,336,171	1,687
$T_{optimized}$	145,121,995	144,968,267	5,856
$T_{space-limited}$	158,798,392	158,450,260	1,687

running time, we measure T_{exp} and T_{div} and compute g_{opt}. Since $T_{mr} = 4,734,181$ (ns) and $T_d = 1,894$ (ns), g_{opt} is approximately 2,499. The number of primes less than 2,499 is 366 from 3 to 2,477. Let $P_{mr}(g)$ denote the probability that the combined algorithm that uses primes less that g performs Miller-Rabin test. Let $N_d(g)$ denote the number of divisions performed in trial division that uses primes less that g. Then, $P_{mr}(2,499) = 0.14318$ and $N_d(2,499) = 65.18$. With these values, we compute the expected running time of the optimized combined algorithm as follows.

$$T_{optimized}(512,2499) = 145,121,995\ (ns)$$

The expected running time of JPV algorithm is slower than that of the optimized combined algorithm and the measured running times of both algorithms show similar gap. Even though the optimized combined algorithm is faster than JPV algorithm, the optimized combined algorithm requires 4 times more space than JPV algorithm requires. Therefore, we compare JPV algorithm and the combined algorithm that uses the same space as JPV algorithm requires. The space to save η, Π, ρ, and $\lambda(\Pi)$ required by the JPV algorithm is 1,687 bit.

In order to compare the two algorithms when both algorithms use the same space, we restrict the combined algorithm such that it uses as many small primes for the trial division as can be stored in 1,687 bits. If a prime is stored in a 16-bit word, 105 small primes from 3 to 577 can be stored in 1,687 bits. When the combined algorithm uses 105 small primes, $P_{mr}(g)$ = 0.175619 and $N_{div}(g)$ = 24.74. With these, the expected running time of the combined algorithm using the same space is as follows.

$$T_{space-limited}(512,105) = 158,798,392\ (ns)$$

Table 5 shows that JPV algorithm still runs slower than the combined algorithm even though the combined algorithm uses less primes than the optimized combined algorithm. However, the gap between the running time of JPV algorithm and the combined algorithm is not wide.

3.4 Performance Improvement on JPV algorithm

We introduce a new method to improve the performance on JPV algorithm. JPV algorithm perform ($c^{\lambda(\Pi)} \bmod \Pi$) computation to test if c is relatively prime to Π. However, this computation includes a modular exponentiation that takes a long time. Instead of this, we can use GCD function [7] to test if c is relatively prime to Π. GCD function is a function to compute the greatest common divisor of two integers. If the greatest common divisor of two integers is equal to 1, two integers are relatively prime.

We propose a probabilistic analysis for the expected running time of JPV algorithm with GCD operation where $T_{gcd}(a,b)$ is the running time of the GCD operation on a and b. Hereafter, let JPV algorithm with GCD operation be the improved JPV for convenience. Because the improved JPV algorithm is the same as JPV algorithm except for GCD function, the expected running time of the improved JPV algorithm can be computed by substituting the running time of GCD function, T_{gcd} for T_{lam}. We first introduce two famous gcd algorithms: Euclid's gcd algorithm, $EGCD(a,b)$ and Binary gcd algorithm, $BGCD(a,b)$.

$EGCD(a,b)$

1.If b is 0, return a
2.Otherwise, return $EGCD(b, a \bmod b)$

$BGCD(a,b)$

1.If $a > b$, and both are odd, $gcd(a,b) = gcd(\frac{(a-b)}{2}, b)$
2.If a is odd and b is even, $cd(a,b) = gcd(a, \frac{b}{2})$
3.If a is even and b is odd, $gcd(a,b) = gcd(\frac{a}{2}, b)$
4.If both a and b are even, $gcd(a,b) = 2gcd(\frac{a}{2}, \frac{b}{2})$
5.If $a < b$, $swap(a,b)$
6.return $BGCD(a,b)$

In practice, two algorithms are combined to use and that is called hybrid gcd algorithm, $HGCD(a,b)$. Euclid's gcd algorithm is used until two integers have similar bit-lengths and then binary gcd algorithm is used as follows.

$HGCD(a,b)$

1.If $b = 0$, return a.
2.If the difference of bit length of (a and b) > 2, $HGCD(b, a \bmod b)$.
3.If the difference of bit length of (a and b) ≤ 2, $BGCD(a,b)$.

The running time of $HGCD(a,b)$ is a sum of the running times for $EGCD(a,b)$ in step 2 and for $BGCD(a,b)$ in step 3. Let T_E and T_B be the running times for $EGCD(a,b)$ and $BGCD(a,b)$, respectively. Then, the running time of gcd is as follows.

$$T_{gcd} = T_E + T_B$$

In $HGCD(a,b)$, $EGCD(a,b)$ performs 2 divisions on average, so the running time is $O((1+\log q)\log b)$, which is the running time for dividing a by b where q is the

Table 6: Comparison of gcd algorithms' running time

a = 1,024 (*bit*)	Euclid's gcd (*ns*)	Binary gcd (*ns*)	Hybrid gcd (*ns*)
32	3,641	110,064	2,189
128	26,249	117,397	13,459
512	166,020	147,588	70,199
2048	538,364	501,249	220,119
4096	617,104	1,572,442	265,022

a = 2,048 (*bit*)	Euclid's gcd (*ns*)	Binary gcd (*ns*)	Hybrid gcd (*ns*)
32	5,414	371,443	3,886
128	31,414	381,834	18,297
512	189,717	424,830	82,290
2048	1,664,242	670,746	672,259
4096	1,822,092	1,847,548	757,367

quotient $\lfloor a/b \rfloor$. The running time of $BGCD(a,b)$ is proportional to the bit length of the bigger of a and b, i.e., $O(\log^2 a)$.

As we already mentioned, there are three kinds of gcd algorithms: Euclid' gcd algorithm, binary gcd algorithm, and hybrid gcd algorithm. Therefore, we compare the running times of three gcd algorithms in order to choose the fastest gcd operation for the later comparison between trial division and gcd operation. We measure the running times of three gcd operations: $EGCD(a,b)$, $BGCD(a,b)$, and $HGCD(a,b)$. The bit-lengths of parameter a are 1,024 and 2,048 bits. The bit-lengths of parameter b are 32, 128, 512, 1024, 2048 and 4096 bits. We randomly generated both a and b at each time and averaged total running time of 1000000 tests. The comparison shows that hybrid gcd algorithm is always fastest in every case in Table 6.

In this situation, we compute the gcd of r and Π_k by using hybrid gcd algorithm where Π_k is the product of k small primes and $p_1 < p_2 < \ldots < p_k$. While the bit-length of r is fixed to n, the size of Π_k is varying as k increases. We divide the analysis of T_{gcd} into two cases when $r > \Pi_k$ and when $r < \Pi_k$.
When $r > \Pi_k$, T_{gcd} is:

$$T_{gcd} = T_E(r, \Pi_k) + T_B(r \bmod \Pi_k, \Pi_k) \tag{14}$$

Since the run time for $EGCD(a,b)$ is $O((1+\log q)\log b)$, $T_E(r,\Pi_k)$ is asymptotically as follows.

$$O((1+\log(\frac{r}{\Pi_k}))\log \Pi_k)$$
$$= O(\log \Pi_k + \log \Pi_k \log r - (\log \Pi_k)^2)$$

Therefore, the running time of $T_E(r,\Pi_k)$ can be represented as a quadratic function of $\log \Pi_k$ as follows.

$$T_E(r,\Pi_k) = x(\log \Pi_k)^2 + y(\log \Pi_k) + z, x < 0 \tag{15}$$

The running time $T_B(r \bmod \Pi_k, \Pi_k)$ is $O(\log^2 \Pi_k)$ because $r \bmod \Pi_k < \Pi_k$. Therefore, it is also a quadratic function of $\log \Pi_k$.

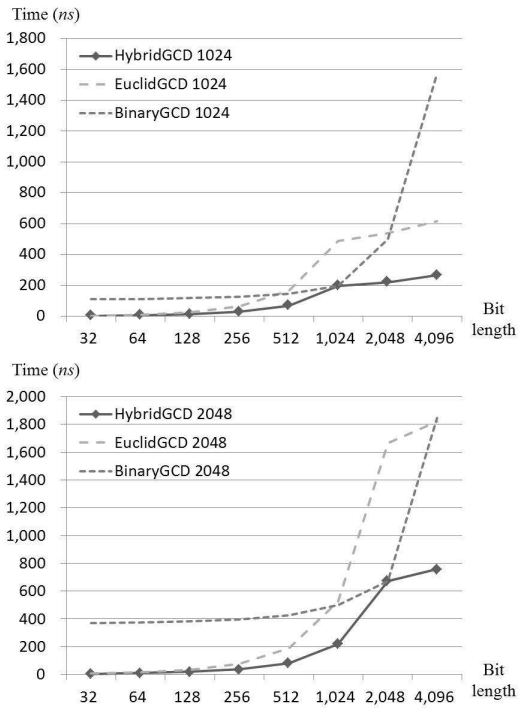

Fig. 1: Comparison of the running time of gcd algorithms

$$T_B(r \bmod \Pi_k, \Pi_k) = x'(\log \Pi_k)^2 + y'(\log \Pi_k) + z', x' > 0 \tag{16}$$

When $r < \Pi_k$, T_{gcd} is:

$$T_{gcd}(k) = T_E(\Pi_k, r) + T_B(r, \Pi_k \bmod r) \tag{17}$$

Since $r < \Pi_k$, the time complexity of $EGCD(\Pi_k, r)$ is like this.

$$O((1+\log(\frac{\Pi_k}{r}))\log r)$$
$$= O(\log r + \log r \log \Pi_k - (\log r)^2)$$

Because $\log r$ is a constant, $EGCD(\Pi_k, r)$ is a linear function of $\log \Pi_k$.

$$T_E = s(\log \Pi_k) + t \tag{18}$$

Since $r > \Pi_k \bmod r$, $T_B(r, \Pi_k \bmod r) = O(r^2)$, which is a constant. Overall, we get the following lemma.

Theorem 2 The running time of $HGCD(r,\Pi_k)$ is as follows.

$$T_{gcd}(k) = \begin{cases} u(\log \Pi_k)^2 + v(\log \Pi_k) + w & (r > \Pi_k) \\ u'(\log \Pi_k) + v' & (r < \Pi_k) \end{cases}$$

where Π_k is $p_1 p_2 \ldots p_k$ and p_i is a prime.

Table 7: comparison of T_E and T_B

	256 (bit)	512 (bit)	1,024 (bit)
$T_E(ns)$	961 - 1,462	1,146 - 4,284	1,581 - 10,804
$T_B(ns)$	36,773	102,568	314,373

To compute the expected running time, we implemented the gcd combined test and measured the running time for generating 1,024-bit primes 1,000,000 times.

When $r > \Pi_k$, $T_{gcd}(k)$ is computed by Theorem 2. Table 7 shows that the running time of $T_E(r,\Pi_k)$ and $T_B(r \bmod \Pi_k, \Pi_k)$ when r is 256, 512 or 1,024 bits.

However, $T_E(r,\Pi_k)$ is very small enough to neglect and $T_B(\Pi_k, r)$ is very similar to a linear function. Thus, when $r > \Pi_k$, $T_B(r \bmod \Pi_k, \Pi_k)$ approximates to a linear function of $\log \Pi_k$. Finally, we can expect the run time of $HGCD(a,b)$ by computing coefficients (u,v) and (u',v'). We compute the regression using 4 sample points for each case. The regression analysis shows a similar result with experimental results.

$$T_{gcd}(k) = \begin{cases} 201.3(\log \prod_{i=1}^{k} p_i) - 3,993 & (r > \Pi_k) \\ 17.1(\log \prod_{i=1}^{k} p_i) - 96,861 & (r < \Pi_k) \end{cases} \quad (19)$$

Because the improved JPV algorithm is the same as JPV algorithm except for GCD function, the expected running time of the improved JPV algorithm can be computed by substituting the running time of GCD function, T_{EUC} for T_{lam} in $T_{JPV(512,72)}$.

T_{gcd} is much faster than T_{lam} because $T_{gcd} = 101,198$ (ns) and $T_{lam} = 1,500,133$ (ns). The expected running time of the improved JPV algorithm using T_{gcd} is as follows.

$$T_{ImprovedJPV}(512) = 158,801,610\ (ns)$$

Figure 2 shows that the improved JPV algorithm using GCD function is better than the original JPV algorithm and the performance of the improved JPV algorithm is similar to the space-limited combined algorithm.

4 Conclusion

In this paper, we proposed a probabilistic analysis on JPV algorithm and compared the total running time of JPV algorithm with the combined algorithm. When a 512-bit prime is generated, the combined algorithm is better than JPV algorithm. Furthermore, we proposed a method to improve JPV algorithm. The improved JPV algorithm shows similar performance of the combined algorithm that uses the same size of space that JPV algorithm requires.

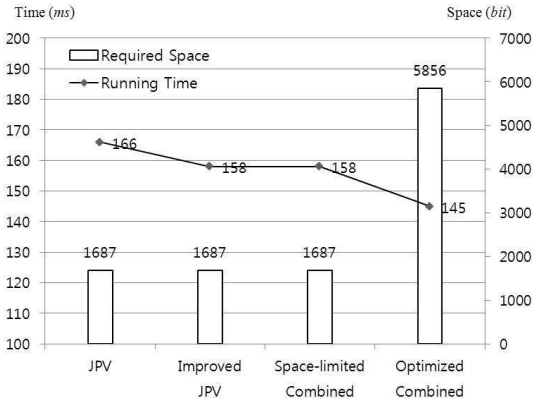

Fig. 2: Comparison of the running time of each algorithm

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012-0006999) and by Seoul Creative Human Development Program (HM120006).

References

[1] W. Diffe and M. E. Hellman, New directions in cryptography, IEEE transactions on Information Theory **22**, 644-643 (1976).
[2] Public-Key Cryptography Standards, PKCS #1 RSA Cryptography Standard.
[3] R. L.Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures an public-key cryptosystem, Communications of the ACM **21**, 120-126 (1978).
[4] T. ElGmal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Transactions on Information Theory **31**, 469-472 (1985).
[5] National Institute for Standards and Technology, Digital Signature Standard(DSS), Fedral Register **56**, 169 (1991).
[6] International Organizatoin ofr Standard, ISO/IEC 18032: Prime Number Generation (2005).
[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, 3nd ed, MIT press (2009).
[8] H. C. Pocklington, The determination of the prime or composite nature of large numbers by Fermat's theorem, Proc. of the Cambridge Philosophical Society **18**, 29-30 (1914).
[9] A. O. L. Atkin and F. Morain, Elliptic curves and primality proving, Mathematics of Computation **61**, 29-63 (1993).
[10] W. Bosma and M. P. van der Hulst, Faster primality testing, CRYPTO'89, LNCS **435**, 652-656 (1990).
[11] U. M. Maurer,Fast Generation of Prime Numbers and Secure Public-Key Cryptographic Parameters, Journal of Cryptology **8**, 123-155 (1995).
[12] J. Shawe-Taylor, , Generating strong primes, Electronics Letters**22**, 875-877 (1986).

[13] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, (1997).
[14] M. O. Rabin, Probabilistic Algorithm for Primality Testing, Journal of Number Theory **12**, 128-138 (1980).
[15] R. Solovay and V. Strassen, A fast Monte-Carlo test for primality, SIAM Journal on Computing **6**, 84-85 (1977).
[16] J. Grantham, A probable prime test with high confidence, Journal of Number Theory **72**, 32-47 (1998).
[17] D. J. Lehmann, On Primality tests, SIAM Journal of Computing **11**, 374-375 (1982).
[18] M. Joye, P. Paillier and S. Vaudenay, Efficient Generation of Prime Numbers, CHES 2000, LNCS **1965**, 340-354 (2000).
[19] R. D. Carmichael, On composite number P which satisfy the Fermat congruence $a^{P-1} = 1(\mathrm{mod}P)$, Amer, Math, Monthly **19**, 22-27 (1912).
[20] J. Viega, M. Messier, and P. Chandra, Network Security with OpenSSL, O'reilly Media (2002)
[21] The GNU Crypto Project, http://www.gnu.org/software/\discretionary{-}{}{}gnu-crypto

Hosung Jo received the M.S. degree in Information and Communication Engineering at Hanyang University, Seoul, Korea in 2007. He is currently a Ph.D. candidate under supervision of Prof. Heejin Park. His research interests are in the areas of cryptography, information security, and computer algorithm.

Heejin Park received the M.S. and Ph.D. degrees in Computer Engineering from Seoul National University in 1996, and 2001, respectively. From 2001 to 2002, he worked as a post-doctoral researcher for the Department of Computer Engineering at Seoul National University. From 2003 to 2003, he was a research professor at Ewha Womens University. He is currently an associate professor in the Department of Computer Science and Engineering at Hanyang University, Seoul, Korea. His research interests are in the areas of cryptography, information security, and computer algorithm.