Fast Prime Generation Algorithms using proposed GCD test on Mobile Smart Devices

Hosung Jo
Embedded Software Research Center,
Hanyang University,
Seoul, Korea

Heejin Park*
Division of Computer Science and Engineering,
Hanyang University,
Seoul, Korea

***Abstract*—As mobile smart devices are widely used, mobile security becomes more and more important. However, the performance of these devices are not powerful enough to use the same security algorithms as PC's. Public key cryptosystem such as RSA needs big primes to enhance the security, however, a generating big primes takes a substantial time even on a PC. In this paper, we proposed two prime generation algorithms for mobile smart devices using GCD primality test. We analyzed and compared the running times of our algorithm with the widely used TD-MR combination on Samsung Galaxy Tab 10.1. The experimental results showed only a 2% error and our algorithm is about 20% faster than the TD-MR combination.**

Keywords—Prime, Prime generation, Primality test, RSA, Public-key cryptosystems, GCD

I. Introduction

Recently mobile smart devices are widely used. As vulnerabilities and attacks against mobile devices become more serious, mobile security becomes more important. Using big primes in the cryptosystems such as RSA [1], ElGmal [2] and DSS [3] is highly recommended. Smart computing of a fast and efficient generating big primes is more important to mobile smart devices, because the mobile smart devices do not have enough computing power and resources as much as PC [4][5].

A prime is a natural number that is bigger than 1 and has no positive divisors except 1 and itself. Prime generation consists of a random number generation and a primality test. The random number generation makes a n-bit positive odd random number r, and the primality test examines whether r is a prime or not. As the primality test takes much longer time than the random number generation, developing fast primality tests is more important. There are two kinds of the primality test: deterministic primality tests and probabilistic primality tests. A deterministic primality test certifies that r is a prime with probability one such as trial division [8], greatest common divisor (GCD) test [9], elliptic curve analogue[10][11], and Maurer's algorithm [12]. A probabilistic primality test certifies that r is a prime with high probabilities which are very close to 1 such as Fermat test [9], Miller-Rabin test [13][14] and Solovay-Strassen test [15].

For practical uses, these primality tests are used together to reduce the running time of the prime generation. One of the popular combinations is trial division and Miller-Rabin test (TD-MR combination hereafter). The combination of GCD test and Miller-Rabin test is a common alternative to TD-MR combination [16][17].

In this paper, we propose two prime generation algorithms using GCD test and probabilistically analyze the expected running time of our algorithms. Then we compare the running times of our algorithms with TD-MR combination on mobile smart devices. This paper is organized as follows. Section 2 introduces some preliminaries briefly. Section 3 describes our contribution: the introduction and analysis of our algorithms and the comparison between each combinations. Finally, we conclude in Section 4.

II. Preliminaries

A. TD-MR Combination

TD-MR combination consists of a random number generation, a trial division, and a probabilistic primality test. The random number generation repeatedly makes a n-bit positive odd random number r until finding a prime. Trial division divides a random number r by p_i's that are primes less than or equal to $\sqrt{n}$. If r is not divided by any p_i, a probabilistic primality test such as Miller-Rabin test is performed. The procedure of TD-MR combination 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 $k$ 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 et al. [12] analyzed this TD-MR combination very accurately and also provided the probabilistic analysis of the expected running time for TD-MR combination. 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, the total running time, T is as follows.

$$T = N_T \cdot (T_{RND} + T_{TD} + T_{MR}) \tag{1}$$

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

** This work was supported by the research fund of Signal Intelligence Research Center supervised by the Defense Acquisition Program Administration and the Agency for Defense Development of Korea.

If the bit-length of r is n, $N = \frac{n \ln 2}{2} \approx 0.347n$. Let T_d 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_{TD} and T_{MR} are as follows.

$$T_{TD} = T_d(1 + \sum_{j=1}^{k} \prod_{i=1}^{j} (1 - \frac{1}{p_i})), T_{MR} = T_{mr}(\prod_{i=1}^{k} (1 - \frac{1}{p_i})) \tag{2}$$

Therefore, T with k small primes is as follows.

$$T = \frac{n \ln 2}{2}(T_{RND} + T_d(1 + \sum_{j=1}^{k} \prod_{i=1}^{j} (1 - \frac{1}{p_i})) + T_{mr}(\prod_{i=1}^{k} (1 - \frac{1}{p_i})) \tag{3}$$

The optimal number of primes that makes the running time fastest is as follows [10].

$$k_{opt} = \frac{T_{mr}}{T_{div}} \tag{4}$$

B. GCD-MR Combination

GCD-MR combination consists of random number generation, GCD test, and probabilistic primality test is as follows. The GCD test computes the greatest common divisor of two integers a and b. If GCD of r and small primes less than or equal to $\sqrt{n}$ is 1, r is a prime. However, when r is large, the running times of both primality tests are very slow. The only difference between TD-MR combination and GCD-MR combination is that a GCD test is used instead of trials division. Therefore, the running time can be modeled as follows where T_{GCD} is the running time of GCD test.

$$T = \frac{n \ln 2}{2}(T_{RND} + T_{GCD} + T_{mr} \prod_{i=1}^{k} (1 - \frac{1}{p_i})) \tag{5}$$

T_{GCD} is the product of T_{gcd} and N_{gcd} where T_{gcd} is the running time that computes the greatest common divisor between r and p_i and N_{gcd} is the number execution of GCD tests. N_{gcd} is exactly the same as the number execution of division in equation (2). Then T_{GCD} is as follows.

$$T_{GCD} = T_{gcd} \cdot (1 + \sum_{j=1}^{k} \prod_{i=1}^{j} (1 - \frac{1}{p_i}))$$

There are two famous gcd algorithms: Euclid's gcd algorithm and Binary gcd algorithm. In practice, these algorithms are combined as Hybrid gcd algorithm. Euclid's gcd algorithm is used until two integers have similar bit-lengths and then binary gcd algorithm is used as follows. Therefore, the running time of Hybrid gcd algorithm is a sum of the running times for Euclid gcd and Binary gcd. Let T_{hgcd}, T_{euc} and T_{bin} be the running times for Hybrid gcd, Euclid gcd, and Binary gcd respectively. Then, the running time of hybrid gcd is as follows.

$$T_{hgcd} = T_{euc} + T_{bin}$$

Since two gcd algorithms are dominated by longer bit-length of parameters, T_{hgcd} is divided into two cases where $r > \Pi$ and $r < \Pi$.

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

III. Our Contribution

We propose two prime number generation algorithms. The first algorithm is a PGCD-MR combination and the second algorithm is a MGCD-MR combination.

A. Proposed PGCD-MR Combination and its Analysis

Since the running time of GCD test is slower than the running time of division, if the number of GCD test and the number of division are the same, GCD-MR combination is always slower.

However, GCD test is able to compute the greatest common divisor with r and all p_i at once without any extra cost, because the running time of GCD test is dominated by the longer bit-length between a and b. That is, while the bit-length of r is bigger than the bit-length of p_i, the running time of GCD test is not changed. Also, the following is obvious.

$$GCD(r, p_1 \cdot p_2) = 1$$
$$\Leftrightarrow (GCD(r, p_1) = 1 \ \ and \ \ GCD(r, p_2) = 1)$$

We define PGCD that computes the greatest common divisor between r and Π_k where Π_k is the product of small primes.

$$PGCD(r, k) = GCD(r, \Pi_k)$$

Then, the procedure of PGCD-MR combination is as follows.

PGCD-MR Combination(n, k)

1) Random Number Generation
   - Generate an n-bit odd random number r.
2) GCD test on r and Π_k
   - Computes GCD(r, Π_k)
   - If the result is not 1, 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.

PGCD is always performed once, then the running time of PGCD, T_{PGCD} is as follows.

$$T_{PGCD} = T_{gcd}(k) \cdot 1$$

Therefore, the running time can be modeled as follows.

$$T = \frac{n \ln 2}{2}(T_{RND} + T_{gcd}(k) + T_{mr} \prod_{i=1}^{k} (1 - \frac{1}{p_i})) \tag{6}$$

For the performance analysis of PGCD, we compared the running time of PGCD and trial division. For a fair comparison between PGCD and trial division, we computed the number of primes when the bit-length of Π_k are 32, 64, 128, 256, 1,024, 2,048, and 4,096 bits. Table II shows the result.

TABLE I. BIT LENGTH OF Π_k FOR PGCD TEST

Π_k	32	64	128	256	512	1,024	2,048	4,096
k	9	15	25	43	74	131	232	417

We compared T_{TD} and T_{PGCD} in Table II and Fig 1. The result shows that the running time of PGCD is faster than or

TABLE II. RUNNING TIME COMPARISON BETWEEN TD AND PGCD

	1024 bit	1024 bit	2048 bit	2048 bit
bit	PGCD (ns)	TD (ns)	PGCD (ns)	TD (ns)
32	2,189	6,939	3,886	12,288
64	7,021	9,681	10,640	17,144
128	13,459	13,631	18,297	24,138
256	28,671	19,843	35,983	35,138
512	70,199	29,338	82,290	51,952

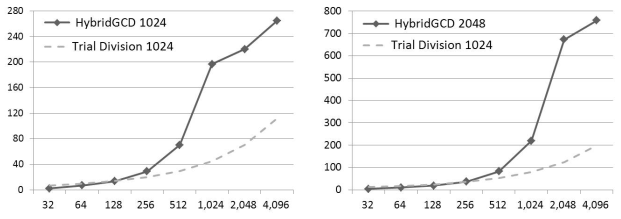

Fig. 1. Comparison of Running time of Trial Division and PGCD

similar to the running time of trial division when the bit-length of Π_k are 32, 64, 128, 256 and 512 bits.

Now, we compared the expected running times and measured running times when PGCD-MR combination is used and 1,024-bit prime is generated. Table III shows the comparison results.

TABLE III. COMPARISON BETWEEN EXPECTED AND MEASURED RUNNING TIME OF PGCD-MR COMBINATION

k	Expected (ms)	Measured(ms)	Error(%)
10	271	272	0.3
15	240	240	0.0
25	214	215	0.4
43	192	193	0.5
74	183	185	1.0
131	169	171	1.1

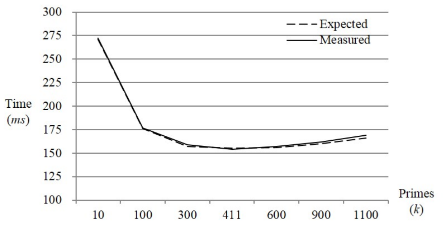

Fig. 2. Comparison of expected and measured running time of PGCD-MR

The error rate is up to a 1.2%. The result shows that our probabilistic analysis is good. As the total running time of PGCD-MR combination is decreasing for a while and increasing later, an optimal point exists. We will find a integer k_{opt} that satisfies the following equation.

$$T_{gcd}(k+1) - T_{gcd}(k) \approx T_{mr}(\prod_{i=1}^{k+1}(1-\frac{1}{p_i}) - \prod_{i=1}^{k}(1-\frac{1}{p_i}))$$

PGCD-MR combination runs fastest when k satisfies the equation below where a is a constant and T_{mr} is the running time of one Miller-Rabin test.

$$\frac{a}{T_{mr}} = \frac{1}{p_{k+1}\log(p_{k+1})}(\prod_{i=1}^{k}(1-\frac{1}{p_i})) \qquad (7)$$

B. Proposed MGCD-MR Combination and its Analysis

As we showed before PGCD is faster than or similar to the running time of trial division when the bit-length of Π_k are 32, 64, 128, 256 and 512 bits. However, PGCD becomes slower as the bit-length of Π_k becomes bigger and bigger. To solve this problem, we suggest using MGCD test. The key idea of MGCD is dividing Π_k into the several proper bit-length of Π_{k_j}. PGCD becomes slower because the bit-length of Π_k is keeping increasing, so we can choose the proper b bit that makes MGCD fastest.

$$MGCD(r, \Pi_k) = 1$$
$$\Leftrightarrow GCD(r, \Pi_{k_1}) = 1, GCD(r, \Pi_{k_2}) = 1, ...GCD(r, \Pi_{k_s}) = 1$$

MGCD computes the greatest common divisor between r and Π_{k_i} sequentially until finding the gcd of r and Π_{k_i} is not one.

$$T_{MGCD}(r, \Pi_k) = T_{GCD}(r, \Pi_{k_1}) + ... + T_{GCD}(r, \Pi_{k_s})$$

Then, the procedure of MGCD-MR combination is as follows.

MGCD-MR Combination(n, k)

1) Random Number Generation
   - Generate an n-bit odd random number r.
2) GCD test on r and Π_{k_j}
   - Divide Π_k into the proper length of Π_{k_j}
   - Computes GCD(r, Π_{k_j}) sequentially
   - If any result is not 1, 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.

Therefore, the running time of MGCD is as follows.

$$T_{MGCD} = T_{gcd} \cdot (1 + \prod_{i=1}^{p_{k_1}}(1-\frac{1}{p_i}) + ... + \prod_{i=1}^{p_{k_s}}(1-\frac{1}{p_i})$$

Finally, the expected total running time of MGCD-MR combination can be computed by the following equation.

$$T = \frac{n \ln 2}{2}(T_{RND} + T_{MGCD} + T_{mr}\prod_{i=1}^{k}(1-\frac{1}{p_i})) \qquad (8)$$

The expected running time of MGCD-MR combination is similar to the measured running time. This means our analysis of MGCD-combination is good.

MGCD-MR combination test is also a convex function. However, there is a possibility that the value of Π_{k_j} could be changed when $(k+1)$th prime is used. When $(k+1)$th prime is multiplied to Π_k, if the bit length of Π_{k_s} is bigger than b bits, then $(k+1)$th prime makes $\Pi_{k_{s+1}}$. Therefore, we

TABLE IV. Comparison between expected and measured running time of MGCD-MR

k	Expected(ms)	Measured(ms)	Error $(\%)$
50	1,560	1,573	0.82
400	1,104	1,117	1.16
1017	979	981	0.2
1600	989	994	0.60
3200	997	1,017	1.96
6400	1,081	1,101	1.80

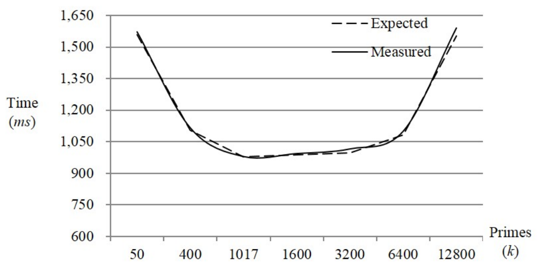

Fig. 3. Comparison of expected and measured running time of MGCD-MR

separate this into two cases: case 1 is when the number of Π_k is not changed and case 2 is when the number of Π_k is changed. When the number of Π_{k_j} is not changed, the k_{opt} is as follows:

$$T_{mr} \approx p_{k+1}(\prod_{i=1}^{k}(\frac{p_i}{p_i+1})$$

When the number of Π_k is changed, the k_{opt} is as follows:

$$(p_{k+1}-1) \approx (\frac{T_{mr}}{T_{gcd}})$$

C. Comparison of Total running times

Finally, we compared the running times of three combined primality test, which are TD-MR combination, PGCD-MR combination and MGCD-MR combination. We measured the total running time of each combination in Samsung Galaxy Tab 10.1 for 100,000 time and averaged them when 1,024 bit prime is generated. Table V and figure 4 shows the comparison results. In Table V, the fastest running time of TD-MR combination is 1,638 ms, the fastest running time of PGCD-MR combination is 1,422 ms, and the fastest running time of MGCD-MR combination is 1,367ms. Hence, our MGCD-MR combination is 20% faster than the previous TD-MR combination in mobile device.

TABLE V. Total running Time Comparison

k	TD-MR	PGCD-MR	MGCD-MR
10	2,086	2,118	2,120
25	1,760	1,732	1,745
43	1,663	1,580	1,560
67	1,638	1,494	1,450
74	1,640	1,479	1,473
131	1,707	1,422	1,402

IV. Conclusion

In this paper, we proposed two prime number generation algorithms: PGCD-MR combination and MGCD-MR combination. We probabilistically analyzed our algorithms to expect

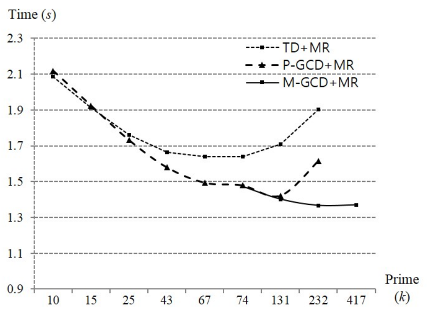

Fig. 4. Comparison of Total running time

their running times. Our analysis expected their running times well with at most a 2% error rate. We also compared the total running times between TD-MR combination, PGCD-MR combination and MGCD-MR combination in Samsung Galaxy Tal 10.1. The result shows that MGCD-MR combination is 20% faster than the previous TD-MR combination in mobile smart devices when 1,024 bit prime generated.

References

[1] Rivest, R.L., Shamir, A. and Adleman, L.: A method for obtaining digital signatures an public-key cryptosystems, *Communications of the ACM*, 21 (2) 120-126 (1978).

[2] ElGmal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory*, 31 (4) 469-472 (1985).

[3] National Institute for Standards and Technology, Digital Signature Standard(DSS), *Fedral Register*, 169 (1991).

[4] Rhee, K., Jeon, W., and Won, D. : Security Requirements of a Mobile Device Management System, *International Journal of Security and Its Applications*, 6 (2) 353-358 (2012)

[5] Rhee, K., Kim, H., and Na, H. Y. : Security Test Methodology for an Agent of a Mobile Device, *International Journal of Security and Its Applications*, 6 (2) 137-142 (2012)

[6] Cohen, Henri, and Arjen K. Lenstra. "Implementation of a new primality test." *Mathematics of computation*, 48 (177), 103-121 (1987)

[7] Hurd, Joe. "Verification of the MillerRabin probabilistic primality test." *The Journal of Logic and Algebraic Programming* 56 (1), 3-21 (2003)

[8] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C.: *Introduction to Algorithms, 2nd ed, MIT press*, (1991).

[9] Menezes, A.J., van Oorschot, P.C. and Vanstone, S.A.: Handbook of Applied Cryptography, *CRC Press*, (1997).

[10] Atkin, A.O.L. and Morain, F.: Elliptic curves and primality proving, *Mathematics of Computation*, 61 29-63 (1993).

[11] Edoh, K : Elliptic Curve Cryptography on PocketPCs, *International Journal of Security and Its Applications*, 3 (3) 23-34 (2009)

[12] Maurer, U.M.: Fast Generation of Prime Numbers and Secure Public-Key Cryptographic Parameters, *Journal of Cryptology*, 8 (3) 123-155 (1995).

[13] Miller, G.L.: Riemann's Hypothesis and Tests for Primality, *Journal of Computer Systems Science*, (1976).

[14] Rabin, M.O.: Probabilistic Algorithm for Primality Testing, *Journal of Number Theory*, 12 128-138, (1980).

[15] Solovay, R. and Strassen V.: A fast Monte-Carlo test for primality, *SIAM Journal on Computing*, 6 84-85 (1977).

[16] Jo, H and Park, H : Performance analysis and improvement of JPV primality test for smart IC cards, *Big Data and Smart Computing 2014*, 271-275, (2014).

[17] Jo, H and Park, H : Probabilistic Analysis on JPV algorithm and Improving it using GCD Function, *Applied mathematics and Information sciences*, 9, 2L, 1-8, (2015).