MATEC Web of Conferences **44**, 01032 (2016)
DOI: 10.1051/matecconf/20164401032
© Owned by the authors, published by EDP Sciences, 2016

An RSA Scheme based on Improved AKS Primality Testing Algorithm

Han Wei Wu [a], Cai Mao Li, Hong Lei Li, Jie Ding, Xiao Ming Yao

College of Information Science and Technology, Hainan University, Haikou ,China 570228

Abstract. In applied cryptography, RSA is a typical asymmetric algorithm, which is used in electronic transaction and many other security scenarios. RSA needs to generate large random primes. Currently, primality test mostly depends on probabilistic algorithms, such as the Miller-Rabin primality testing algorithm. In 2002, Agrawal et al. published the Agrawal–Kayal–Saxena (AKS) primality testing algorithm, which is the first generic, polynomial, deterministic and non-hypothetical algorithm for primality test. This paper proves the necessary and sufficient condition for AKS primality test. An improved AKS algorithm is proposed using Fermat's Little Theorem. The improved algorithm becomes an enhanced Miller-Rabin probabilistic algorithm, which can generate primes as fast as the Miller-Rabin algorithm does.

Keywords: RSA, Miller-Rabin, AKS algorithm, Primality testing

1 Introduction

The RSA algorithm is a typical asymmetric algorithm, which is used for security applications in the information society, including ID authentication and electronic transaction in electronic transaction scenarios [1]. Compared with other public key algorithms, RSA is slow but its strength is its ability to be used for encryption and digital signature. Its key generation principle states that it should first choose two large primes *P* and *Q*, computing *n*=*P***Q*. Then, it will calculate the Euler function $\phi(n) = (P-1)*(Q-1)$, and randomly select a positive integer *e* so that $1 < e < \phi(n)$. Meanwhile, they need to be coprime, i.e. $gcd(\phi(n), e) = 1$. Later on, the private key *d* needs to be computed via $d * e = 1 \, mod \, \phi(n)$, where *n* and *e* are public keys. The security of RSA depends on the difficulty in decomposing the large integer *n*.

From the procedures above, it can be seen that the RSA key scheduling algorithm needs to generate two large primes. To ensure security strength, the two generated large primes are usually longer than 512 bits. In December 12, 2009, the large integer with a number of RSA-768 (768-bit key) was decomposed successfully [2]. This event poses threats against the security of existing 1024-bit key, highlighting the need for general users to upgrade to keys of 2048- or longer bits. Rapid generation of large primes is essential for RSA. The current approach for generation of large primes is to randomly generate a large integer, and then perform the primality test. Common approaches for primality test include the probabilistic Miller-Rabin test method [3], Solovay-Strass test method [8], and the Lucas-Lehmer N-1 and N+1 deterministic test methods [10][11]. Agrawal, Kayal and Saxena [4] successfully addressed the worldwide problem of testing primality in polynomial time by proposing an algorithm called AKS. Their algorithm determines the input integer is a prime or a composite number. It is the first published generic, polynomial, deterministic and non-hypothetical algorithm for primality test. All of the previous primality test algorithms can have at most three of the four properties above.

Despite its ability to perform primality test in polynomial time, the overheads that it incurs during operation in terms of transport and storage complexity are so large that it is infeasible in practice. Major improvements achieved include the improved algorithm by Bernstein [5] and the optimized algorithm by Jin Zhengping based on that of Bernstein [6]. The algorithm by Bernstein is more efficient. For positive integers 3640471 and 4295884871, the running time of the original AKS and Bernstein algorithms is dozens or hundreds of hours. The improved algorithm by Bernstein still costs tens of seconds. For primes as long as 40 plus digits, the improved version still requires several hours. After a thorough analysis of the improved version by Bernstein, Jin Zhengping et al. demonstrated not only enormous improvements over other algorithms but also some problems that it has. Meanwhile, they stated that it needs to be further improved for practical applications of primality test. None of existing AKS algorithms are useful for practical security applications, because algorithm completeness cannot be achieved without enormous computing and storage overheads.

[a] Corresponding author: wuhanwei@hainu.edu.cn

This work is supported by National Natural Science Foundation of China (No. 61462023), Natural Science Foundation of Hainan Province (No. 614231, 613158, 20156250).

This paper begins with the necessary and sufficient condition for AKS primality test, and does not compute the congruent polynomial as the original algorithm does. The proposed algorithm is rendered non-deterministic by choosing smaller test sets. It is proved strictly that the proposed algorithm is equal to the strong-constraint Miller-Rabin test. Prime generation of the proposed algorithm achieves a substantial speedup. A complete example of RSA based on the improved AKS test algorithm is implemented on computer using Python (64-bit Windows 7, Python 3.4.3, 8G memory, AMD FX8300@3.30G, 256 G solid-state drives). Comparison shows that the executing efficiency of the improved AKS algorithm can rival the industry-standard Miller-Rabin algorithm. Therefore, the proposed algorithm is useful for practical applications.

2 Common Primality Testing Algorithms

2.1 Fundamental concepts and principles of AKS

AKS primality test is mainly based on the following theorem: the integer n (≥2) is a prime when and only when

$$(x-a)^n \equiv (x^n - a)\ (mod\ n) \qquad (1)$$

This congruent polynomial is true for all integers a which are coprime with n. This theorem is a generic representation of Fermat's Little Theorem.

Proof:

1. First prove that (1) holds when n is a prime.

Based on Fermat's Little Theorem $a^{n-1} \equiv 1\ (mod\ n)$, where n is a prime, it can be deduced that

$$(x-a)^n \equiv (x-a)^{n-1}\cdot(x-a) \equiv (x-a)\ mod\ n$$

$$(x^n-a) \equiv (x^{n-1}\cdot x - a) \equiv (x-a)\ \ mod\ n$$

Therefore, (1) is true.

2. Proceed to prove that if (1) holds, then n is a prime.

For each $0 < i < n$, we have

$$(x-a)^n = x^n + \sum_{i=1}^{n-1}\binom{n}{i}x^i(-a)^{n-i} + (-a)^n$$

The number of interest, n, is either composite or prime. If n is prime, then $n|\binom{n}{i}, 0 < i < n$. The conclusion is true.

If n is composite, then there must exist a prime divisor of n, denoted with q. Consider that $q^k \parallel n$, i.e. $q^k|n$. But $q^{k+1} \nmid n$. Then, it can be proved that $q^k \nmid \binom{n}{q}$, where$\binom{n}{q} = \frac{n(n-1)\cdots(n-q+1)}{q!}$.

Because $\binom{n}{q} = \frac{n(n-1)\cdots(n-q+1)}{q!}$, we can let $n = q^k m$ and *gcd*(*m, q*)=1. Thus, $1 \le i \le q-1$ for $1 \le i \le q-1$. Otherwise, $q|n-(n-i) = i$. Therefore, $q|n-(n-i) = i$, it can be deduced that

$$\binom{n}{q} = \frac{n}{q}\,\frac{(n-1)\cdots(n-q+1)}{(q-1)!}$$

$$= q^{k-1}m\frac{(n-1)\cdots(n-q+1)}{(q-1)!}$$

But

$$\text{gcd}\left(m\frac{(n-1)\cdots(n-q+1)}{(q-1)!},\text{q}\right) = 1$$

Hence, we have $q^k \nmid \binom{n}{q}$. The largest common divisor of n and a can be obtained quickly using the extended Euclidean algorithm. If the two numbers are not coprime, then it can be inferred directly that n is composite. Thus, $gcd(n = q^k, a)$=1 is requested. Then, the coefficient module of x^q is not zero. $(x-a)^n - (x^n - a)$ modulo n is not always equal to zero. (1) is false and contradictory to the original condition. Therefore, if (1) is true, then *n* is prime.

Although the primality test can be done based on the defined formulation, the time needed increases exponentially. To reduce computational complexity, AKS switches to the following congruent polynomial.

The steps of the algorithm are as follows:

Input: integer *n*>1

① If there exist a > 0 and b > 1, and $n = a^b$; then output the composite number.

② Find the smallest r such that $\text{ord}_r(n) > \log^2(n)$.

③ If 1 < *gcd*(*a,n*) < n is true for some a ≤ r, then output the composite number. (gcd represents the largest common divisor of a and n.)。

④ If n ≤ r, output the prime number.

⑤ For all a from 1 to$\lfloor\sqrt{\varphi(r)}log\ (n)\rfloor$, if $(x+a)^n \neq x^n + a$ (mod n, $x^r - 1$), output the composite number.

⑥ Output the prime number.

Here, $ord_r(n)$ is the order of n mod r. In addition, log represents the logarithm with 2 as the base, $\varphi(r)$ represents the Euler function of r. If n is prime, then the algorithm can always return prime number. Due to primality of n, Steps 1 and 3 will never return composite number. Neither will Step 5, because (2) is true for all primes n. Hence, there is no doubt the algorithm steps 4 and 6 will return prime number. Correspondingly, if n is composite, then the algorithm will undoubtedly return composite number. If the algorithm returns prime number, then it is returned from Steps 4 or 6. In the former case, considering n ≤ r, it is certain that n has divisor a ≤ r such that 1 < gcd(a, n) < n. So composite number will be returned. The remaining case is Step 6, which is impossible, because the several equations tested in Step 5 can ensure that the output number is composite.

Many works have been done later to propose variants of the original algorithm. The best examples of improved algorithms include the method by Bernstein in [5]. Its complexity is $O(log^{4+o(1)}(n))$. A large prime number of n is requested for practical applications. In RSA, n usually has a binary length of over 1024 bits. So the improved version are unsuited for practical security applications.

2.2 Miller-Rabin algorithm

Primality test algorithms are classified into the probability-based method and the deterministic method. The probability-based method is fast but its misjudgment probability is high. The greatest property of the deterministic method is its ability to prevent misjudgment.

But its test speed is so low that it is not as feasible as the probability-based method in practical applications.

The Miller-Rabin algorithm achieves improvements by using Fermat's Little Theorem. It incorporates square root decision into Fermat's Little Theorem [3][7]. In the Miller-Rabin primality test, strong psedoprime decision is made by elegantly combining Format test with square root test, achieving a misjudgment probability of 1/4 per time. Choosing multiple base numbers in the test, the misjudgment probability can be reduced to an extremely low level. For example, choosing 64 base numbers in the test can lead to a misjudgment probability of 1/(464). In the test, n-1 is represented with the product of an odd number m and 2.

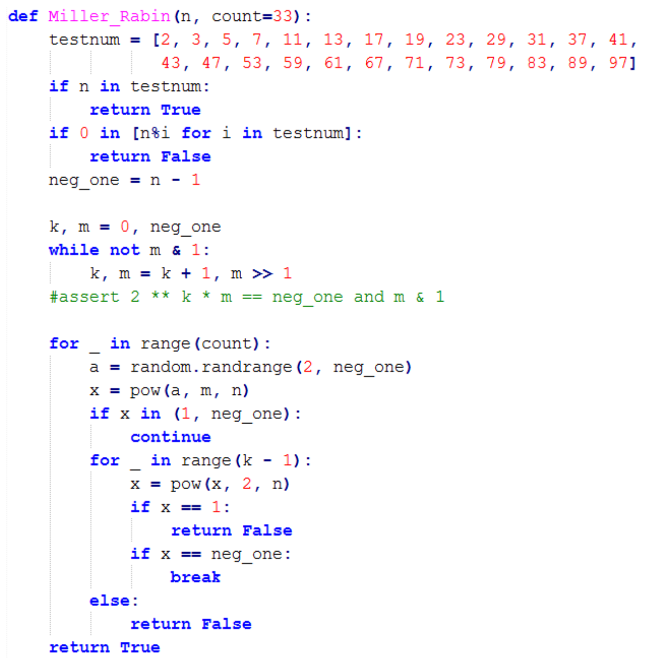
$$n-1=m\times 2^k$$

Fermat test of the base number a can be written as

$$a^{n-1}=a^{m\times 2^k}=[a^m]^{2^k}$$

Note that $a^{n-1}\ (mod\ n)$ is computed through k+1 steps rather than in a single step. In this way, square root test can be done in each step. If square root test fails, it means that n is composite. If each step is correct, then it can be known for sure that not only Fermat test but also all square root tests are passed.

Algorithm is summarized using Python in Fig.1.



```
def Miller_Rabin(n, count=33):
    testnum = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,
               43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
    if n in testnum:
        return True
    if 0 in [n%i for i in testnum]:
        return False
    neg_one = n - 1

    k, m = 0, neg_one
    while not m & 1:
        k, m = k + 1, m >> 1
    #assert 2 ** k * m == neg_one and m & 1

    for _ in range(count):
        a = random.randrange(2, neg_one)
        x = pow(a, m, n)
        if x in (1, neg_one):
            continue
        for _ in range(k - 1):
            x = pow(x, 2, n)
            if x == 1:
                return False
            if x == neg_one:
                break
        else:
            return False
    return True
```

Figure 1. Miller-Rabin algorithm.

Algorithm steps:

①. Choose primes below 100. If n belongs to this list，return true。If n is divisible by primes below 100, return the conclusion that n is composite.

② . Randomly choose a base number a and compute$T=a^m$, where $m=(n-1)/2^k$.

③. If T is either 1 or -1 and all subsequent square root tests can be passed, it means that n is a strong pseduprime, and return to Step 2, choose a new base number to test. If T are other numbers, go to the following step.

④. Conduct the worst-case test through K-1 steps. First compute $T=T^2 \bmod n$. If T=1, because the original value of T are other numbers, based on the square root test rule, if n is composite, then $T=\sqrt{1}=\pm 1$ or other numbers. Thus when T=1, return the conclusion that n is composite.

⑤. In Step 4, if T= -1, based on the square root test rule, if n is prime, then $T=\sqrt{1}=\pm 1$. So n will pass the test definitely. T= -1 for this step and 1 for the next step. Return the conclusion that n is prime。

⑥. If in Step 4, T is other numbers, compute k=k-1, and then jump to Step 4.

⑦. If primality of n cannot be determined when k=0, return the conclusion that n is composite.

3 Result and Analysis of the Improved AKS Algorithm

3.1. Miller-Rabin-based improved algorithm

Based on the Fermat's Little Theory, if n is prime, then (1) is true definitely. (1) can be rewritten as

$$(x-a)^{n-1}(x-a)\equiv(x^n-a)\ (mod\ n)$$

If n is prime, then it can undoubtedly justify gcd($(x-a)$,n)=1. That is, $(x-a)$ has an inverse element $(x-a)^{-1}$. It can be further deduced that

$$(x-a)^{n-1}\equiv(x^n-a)(x-a)^{-1}\ (mod\ n)\quad (2)$$

It can be derived from using Fermat's Little Theory again that the left and right sides of (2) mod n are invariably equal to 1.

Consider the left side of (2). If x is set to a fixed number and the value of the base number a is changed only

Rabin primality test. That is, for the base number $(x-a)$,

$$(x-a)^{n-1}=(x-a)^{m\times 2^k}=[(x-a)^m]^{2^k}$$

Consider the right side of (2). It is requested to justify gcd($(x-a)$,n)=1 and $(x^n-a)=(x-a)$. Further simplifying it yields $x^{n-1}=1(mod\ n)$. Thus, by using Fermat's Little Theory many times to narrow the range of the base number a, AKS can be expressed as a Miller-Rabin test algorithm with two additional constraints.

For an input large integer n of interest, the Miller-Rabin testing algorithm jointly performs a Fermat's test and a square root test. The newly added constraints introduce an additional Fermat's test and an additional greatest common divisor test. Neglecting the test for greatest common divisor, the improved AKS has a misjudgment probability of 1/8 per time. For a given misjudgment probability, the set of base numbers in the improved AKS algorithm is two thirds of that in the Miller-Rabin test algorithm.

Algorithm steps:

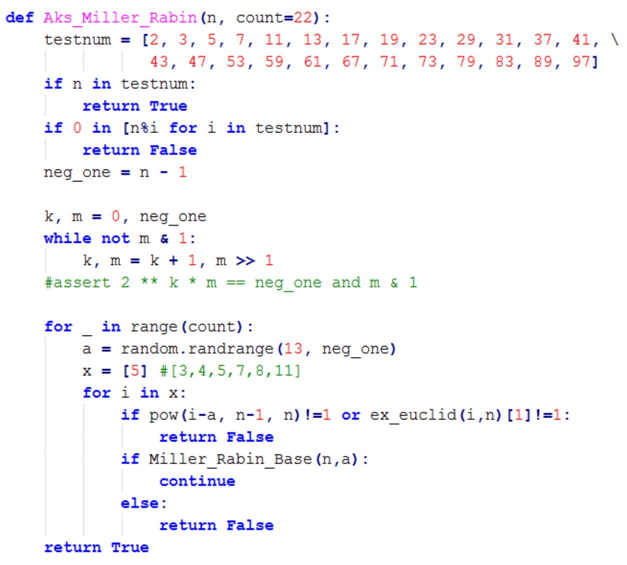
①. Choose primes below 100. If n belongs to this list，return true。If n is divisible by primes below 100, return the conclusion that n is composite.。

②. Randomly choose a base number a and then choose a fixed value for x. Compute gcd($(x-a)$,n)=1. If the condition is not met, return the conclusion that n is composite. If it meets the condition, go to the next step.

③. Compute to check whether $x^{n-1} = 1(mod\ n)$ is true。 If not, return the conclusion that n is composite. Otherwise, go to the next step.

④. Test $(x - a)$ using the Miller-Rabin algorithm. If the result is true, it means that n is prime. Otherwise, return the conclusion that n is composite.

Major Python codes of the algorithm are given in Fig.2. In this algorithm, the extended Euclidean algorithm is used to check whether two numbers are coprime. The built-in modulo operating function in Python is directly utilized for code complicity. Note that the Miller-Rabin algorithm is changed into Miller-Rabin-Base, where the division test for primes below 100 is omitted.



```
def Aks_Miller_Rabin(n, count=22):
    testnum = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, \
               43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
    if n in testnum:
        return True
    if 0 in [n%i for i in testnum]:
        return False
    neg_one = n - 1

    k, m = 0, neg_one
    while not m & 1:
        k, m = k + 1, m >> 1
    #assert 2 ** k * m == neg_one and m & 1

    for _ in range(count):
        a = random.randrange(13, neg_one)
        x = [5] #[3,4,5,7,8,11]
        for i in x:
            if pow(i-a, n-1, n)!=1 or ex_euclid(i,n)[1]!=1:
                return False
            if Miller_Rabin_Base(n,a):
                continue
            else:
                return False
    return True
```

Figure 2. The improved AKS algorithm.

3.2 Correctness and efficiency of the improved AKS algorithm

Multiple large random numbers are generated in larger ranges to demonstrate correctness of the proposed algorithm. Tests are done on the Miller-Rabin algorithm and the proposed algorithm, respectively. Discrepancy between test results is defined as an erroneous output. The results are given in Tab.1.

Table 1. Correctness test for the improved AKS algorithm

Length of N	Test numbers	Primes	Test time(s)	Errors
$2^{32} < n < 2^{64}$	10000	219	0.7082	0
$2^{80} < n < 2^{128}$	10000	106	1.0046	0
$2^{130} < n < 2^{180}$	10000	82	1.2839	0
$2^{256} < n < 2^{512}$	10000	34	5.3189	0
$2^{1024} < n < 2^{1028}$	10000	23	26.1501	0

Consider a random range in Tab.1, such as $2^{130} < n < 2^{180}$. A total of 10,000 random numbers is chosen. Each random number is subject to a Miller-Rabin test and an improved AKS test. 82 of the 10,000 numbers pass the primality test, costing 1.2839s. All 50,000 tests over different ranges of n show that the two algorithms provide the same results.

Executing efficiency comparison is then performed between the Miller-Rabin and the improved AKS algorithms. The set of base numbers has 33 base numbers for the Miller-Rabin algorithm and 22 base numbers for the improved AKS algorithm. The test time is shown in Fig.3.

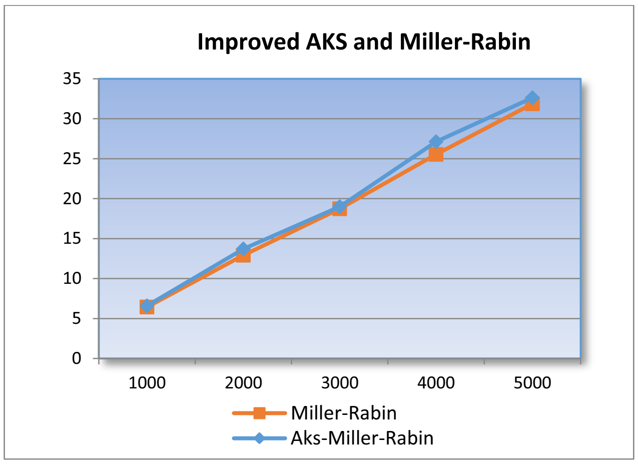


Figure 3. The execution efficiency of improved AKS algorithm.

4 Conclusion

AKS provides an effective approach for the worldwide problem of checking primality in polynomial time. Many improvements have been made on the original algorithm. The complexity of the computation procedures necessary to ensure algorithm completeness is too heavy. Great security strength can be achieved by using the public key scheme based on the elliptic curve discrete logarithm when the number n of interest has a length of 200 bits. The improved Bernstein algorithm can be used on the supercomputer to check the primality of n. But this type of complete algorithms is infeasible for RSA in electronic transaction.

This paper starts with analyzing the necessary and sufficient conditions for AKS and performs equivalence transformation on the algorithm's equations using Fermat's Little Theory. It is proven that this algorithm can be expressed as the Miller-Rabin testing algorithm with two additional constraints when the set of base numbers is narrowed. Experimental results show that the executing efficiency of the improved AKS algorithm can rival that of the Miller-Rabin algorithm. So the proposed algorithm is of value for practical applications.

References

1. RIVEST R，SHAMIR A, ALDEMAN L. A method for obtaining digital signatures and public-key cryptosystems[J].Communications of the ACM，1978，21(2):120-126
2. Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter L Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus[G]. // Advances in Cryptology-CRYPTO 2010. Santa Barbara:Springer, 2010: 333-350
3. Rabin M O. Probabilistic algorithm for testing primality[J]. Number Theory，1980，12(1):128-138
4. Manindra Agrawal, Neeraj Kayal and Nitin Saxena. PRIMES is in P [J]. Annals of Mathematics，2004，160(2):78-793
5. Bernstein D J. Proving primality in essentially quartic random time[J] . Math Comp , 2007 , 76(257):389-403
6. JIN Zheng-ping , WEN Qiao-yan. Implementations of the Improved AKS Primality Testing Algorithm[J]. Journal of Sichuan University Natural Science Edition, 2009, 41 (1):147-152 (in Chinese)
7. Miller, Gary L. Riemann's Hypothesis and Tests for Primality[J]. Journal of Computer and System Sciences ,1976,13 (3): 300-317
8. R. Solovay and V Strassen. A fast Monte-Carlo test for primality[J]. SIAM Journal on Computing，1977，6(1):84–86
9. Bruce, J W. A Really Trivial Proof of the Lucas-Lehmer Test[J]. The American Mathematical Monthly，1993，100(4): 370-371
10. SHAFI GOLDWASSER and JOE KILIAN.Primality Testing Using Elliptic Curves[J]. Journal of the ACM，1999，46(4): 450-472
11. Atkin, AOL and Morain F. Elliptic Curves and Primality Proving[J]. Mathematics of Computation, 1993, 61(203): 29-68