

RSA encryption

Jack Zhao Jin

Denison University

slowrusher@slowrusher.com

Abstract

We conduct a survey of different RSA encryption techniques. In this paper we will discuss the basic RSA encryption algorithm, why it works, and why it is secure. Later, we present a few selected recent research issues and directions. This paper gives a general overview of RSA, how it is used and researched today.

1. Introduction

The idea of public key cryptography is closely related to the idea of one-way functions. Given an argument x , it is easy to compute the function value $f(x)$, whereas it is intractable to compute x from $f(x)$. Here “intractable” is understood in the sense of complexity theory. A receiver can receive secured information from the sender by publicizing the function f , and then the sender computes $f(x)$ from x , and sends $f(x)$ to the receiver. Only the receiver can calculate x from $f(x)$, assuming it is intractable to compute x from $f(x)$.

RSA encryption is the best known and most widely used public key encryption technology. It was invented by R. L. Rivest, A. Shamir, and L. Adleman at MIT in

1978, hence the name RSA. RSA encryption relies on having extremely large keys that are too hard to factor. Like all great encryption technologies, an encrypted message is very hard to decipher with the given public keys.

The basic RSA encryption method is based on a public key and a private key. The public key is published, and is used to encrypt messages. The private key is kept secret by the party who is receiving the encrypted messages. The brilliant part about RSA encryption is that you can not decrypt the encrypted message using the public key without discovering the private keys first. To generate a private key using the technologies today from the public key would take hundreds of years. Because of the security of the RSA encryption, RSA encryption is used all over the Internet. For example, a bank can allow its clients to send encrypted data to the bank, and only the bank can decrypt the encrypted message in a reasonable amount of time.

Much research and implementation work have been done on RSA since 1978. Some research has attempted to seek flaws in some RSA implementation, while other research made RSA more secure or faster. In section 2, we will discuss the generic encoding and decoding techniques. In section 3, we will discuss a wide variety of ways to generate keys. In section 4, we will review some real world applications of RSA. And finally, in section 5, we will discuss attacks on RSA.

2. Generic Encoding and Decoding

To implement RSA encryption, two large distinct prime numbers p and q must be generated and kept secret. Let $n = p * q$. The n is sometimes called the modulus, and the

value of n is made public. The second public key generated is e , such that $\gcd(e, \phi(n)) = 1$, where $\phi(n) = (p-1)(q-1)$. $\phi(n)$ is where $\phi(1) = 1$ and $\phi(r^b) = r^b - r^{b-1}$, where r is prime. To encrypt an integer c , c must be smaller than n . $E(c)$ represents the encrypted value of c , $E(c) = c^e \pmod{n}$. For the RSA encryption algorithm to work, every decryption must be unique.

To decode $E(c)$, a positive integer d must be computed from the original two primes, p and q . The variable d is defined to be the inverse of e modulo $\phi(n)$, which can be computed by extended Euclidean algorithm. Once d is known, the original message c can be found by computing $c = (E(c))^d \pmod{n}$.

Salomaa [3, pp.126] showed a proof that proved every decryption is unique in the following way. Let $c = w^e \pmod{n}$ be the encrypted value of w . If decryption is unique then $w = c^d \pmod{n}$. we need to show that there is a positive integer j such that $ed = j\phi(n) + 1$. Since neither p nor q divides w , by Euler's Theorem, $w^{ed-1} = 1 \pmod{n}$. Then $c^d = (w^e)^d = w \pmod{n}$.

S. C. Coutinho's showed a proof that proved the RSA algorithm is correct in the following way [1, pp. 166]. In order for the RSA algorithm to work, $D(E(c)) = c$, for $n > c$, where $E(x)$ is the encryption function and $D(x)$ is the decryption function. In the RSA algorithm, $D(E(c)) = c^{ed} \pmod{n}$. Since d is the inverse of $e \pmod{\phi(n)}$, we can conclude that $c^{ed} = c \pmod{n}$ using Euler's Theorem. But in order to apply Euler's theorem we must show that n and b are co-prime. This means that, when we break the message into blocks, we need to make sure that the blocks are co-prime to n . But then

he proves that n and b do not have to be co-primes by using Fermat's Theorem. So, in conclusion, $D(E(c)) = c$ for $n > c$.

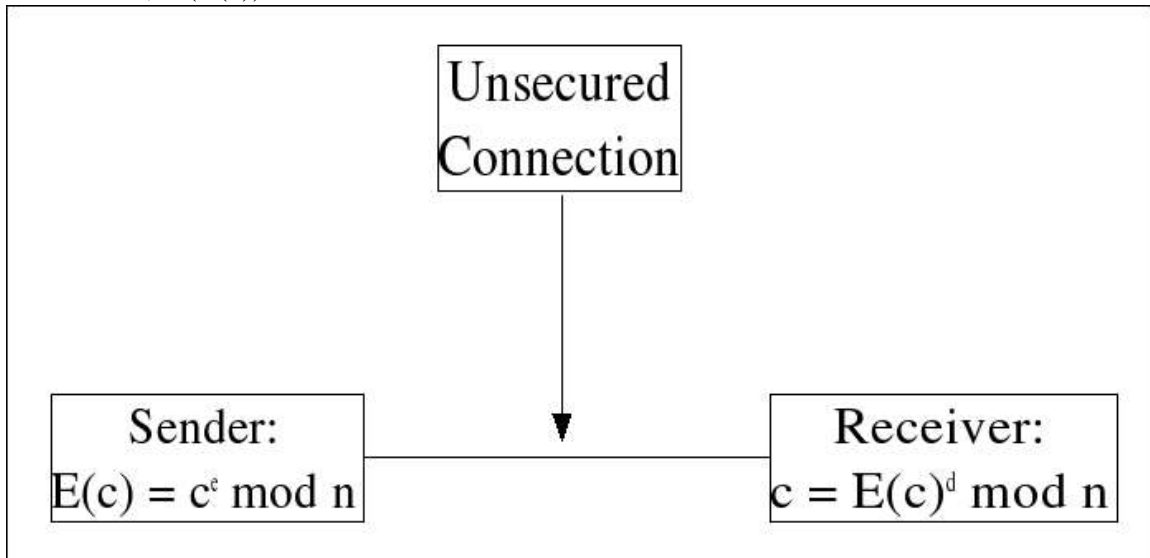


Fig 1. sender sends c^e modulo n , and receiver decrypts the message by executing message^d modulo n .

3. Generating Keys

In this section we'll discuss several methods to generate p and q . Some of the methods described here are not used to generate RSA keys, but they have historical or theoretical importance.

Primes can be generated using the polynomial formula. For example, given the formula $f(x) = x^2 + 1$, it can generate many primes for x is between 1 and 10, although not every x generates a prime. The general idea is to use a quadratic function $f(x) = ax^2 + bx + c$, where variables a , b , c are all integers, and $a > 0$. This algorithm for finding primes isn't very good since one cannot find all the possible primes with this formula.

[3, pp. 49]

Primes can also be found using exponential formulas. For example, the famous formula generates the Mersenne numbers, $M(n) = 2^n - 1$. Mersenne made the claim that when $n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257$, $M(n)$ is prime. The original lists was missing $M(61)$, $M(89)$, and $M(107)$. The largest prime number found up to 1999 was $M(3,021,377)$, which has 1819050 digits. [3, pp 52]

A mathematician named Fermat later discovered another exponential formula: $F(n) = 2^{(2^n)} + 1$. He believed that $F(n)$ is prime for $n = 0$ to 6 , and conjectured that all $F(n)$ is prime for all integers $n \geq 0$. But $n = 5$ failed, and it is very difficult to calculate Fermat numbers since it's a double exponential formula.

Another way to discover very large primes is through probabilistic primality testing. If an integer m goes through a composite test and passes the test, then it's definitely composite. If it fails the test, then m might be prime. The probability of m being prime increases as the number of failed composite tests increases. The probability of the RSA algorithm failing is very low because the RSA algorithm will probably work the same as if p is prime, since $E(D(c))$ will probably $= c$ for all c within a reasonable size.

Coutinho [1] also shows a method for generating primes. To select a prime, choose p to be between $4r/10$ and $45r/100$ digits, where r is the approximated number of digits for n . Then choose q to be close to $10r/p$ digits. Once the prime is obtained, the numbers $p - 1$, $q - 1$, $p + 1$, $q + 1$ must not only have small factors, otherwise they'll be easy prey for some factorization algorithms. To obtain the prime, first he suggested

we should get a simple distribution of primes. Let $\pi(x)$ denote the approximation of the number of primes before x . According to the prime number theorem, if x is large, then $\pi(x)$ is approximately equal to $x/\log x$. Let x be some very large positive and let ϵ be some positive number, then $\pi(x + \epsilon) - \pi(x)$ would approximate the number of primes between $x + \epsilon$ and x . Suppose we want to find a prime near x . Let x be around 10^{127} and ϵ to be around 10^4 . $\pi(x + \epsilon) - \pi(x)$ would be approximately 34, which means that there are approximately 34 primes between x and $x + \epsilon$.

Coutinho presented a strategy for proving a given odd number k to be a prime number, and the strategy is used on odd numbers between x and $x + \epsilon$.

Step 1:

The first step of his strategy was to check if k is divisible by a prime smaller than 5000. This step would be a very quick and efficient way to eliminate most of the odd numbers between x and $x + \epsilon$. If the odd number passes step one, apply Miller's test to x .

Step 2:

The Miller's test is a probabilistic primality test. It takes an odd integer $n > 0$ and a base b , where $1 < b < n - 1$. After it's done, it either says n is a composite or n is inconclusive. Here is Miller's test's algorithm:

Step 2.1: Divide $n - 1$ by 2 as many times as necessary in order to find an odd

co-factor. Thus we have found positive integers k and q , so that $n - 1 =$

$2^k q$ and q is odd.

Step 2.2: Begin by setting $i = 0$ and $r =$ the residue of $b^q \pmod n$.

Step 2.3: If $i = 0$ and $r = 1$, or if $i \geq 0$ and $r = n - 1$, the output is inconclusive, otherwise go to step 2.4.

Step 2.4: Increase i by 1 and replace r by the residue $r^2 \pmod n$, go to step 2.5

Step 2.5: If $i < k$, return to step 2.3; otherwise the output is composite

Step 3:

Assuming that the output of Miller's test to all these bases was inconclusive, apply the following primality test to n . This primality test was by Brillhart, Lehmer, and Selfridge written in 1975.

Let $n > 0$ be an odd integer so that $n - 1 = p_1^{e_1} \dots p_r^{e_r}$, where $p_1 < \dots < p_r$ are positive prime numbers. If, for each $i = 1, \dots, r$ there exist integers b_i ($2 \leq b_i \leq n - 1$) such that b_i^{n-1} is equivalent to 1 mod n and $b_i^{(n-1)/p_i}$ is not equivalent to 1 mod n , then n is prime.

Proof of the primality test:

Let $i = 1$; the same argument applies to $i = 2, \dots, r$. First, we must compute the order of b_1 in $U(n)$; let's denote it by s_1 . It follows from the key lemma and from equation $b_1^{n-1} = 1 \pmod n$ that s_1 divides $n - 1$. Hence, the primes that appear in the factorization of s_1 are among the primes p_1, \dots, p_r . Thus $s_1 = p_1^{k_1} \dots p_r^{k_r}$, where $k_1 < e_1, \dots, k_r \leq e_r$. On the other hand, we know that $b_1^{(n-1)/p_1} \not\equiv 1 \pmod n$. Therefore, $(n - 1)/p_1$ is not divisible by s_1 . But $(n - 1)/p_1 = p_1^{e_1-1} p_2^{e_2} \dots p_r^{e_r}$. Comparing the factorizations of s_1

and $(n - 1)/p_1$, and keeping in mind that s_1 does not divide $(n - 1)/p_1$, we see that $k_1 = e_1$. In other words, $p_1^{e_1}$ divides s_1 .

Recall that s_1 is the order of b_1 in $U(n)$. By Lagrange's theorem, s_1 divides the order of $U(n)$. Thus s_1 divides $\phi(n)$. Since $p_1^{e_1}$ divides s_1 , it follows that $p_1^{e_1}$ divides $\phi(n)$. A similar argument can be used for $i = 2, \dots, r$, so the congruences of the test imply that $p_1^{e_1}, p_2^{e_2}, \dots, p_r^{e_r}$ divide $\phi(n)$. These are pairwise co-prime, because they are powers of distinct primes. The product of $p_1^{e_1} \dots p_r^{e_r} = n - 1$ also divides $\phi(n)$. Since $\phi(n) \leq n - 1$, we must have that $\phi(n) = n - 1$. Hence n is prime.

4. Real World Applications

4.1 Signatures

A signature is used to identify if a message sent from the sender to the receiver is indeed legit. If the receiver does not use signatures, then anyone can replicate an encrypted message and send it to the receiver, and the receiver would have no way of telling which message is legit and which isn't. The way an electronic message can be signed in the following way. Let E_c and D_c be the sender's encryption and decryption functions, and let E_b and D_b be the encryption and decryption functions for the receiver. Let c be a block of message the sender wishes to send to the receiver. To send the message securely, the sender must send the encrypted message $E_b(c)$. To make sure that the message is also signed, the sender will send the receiver the encrypted message $E_b(D_c(c))$. After receiver receive $E_b(D_c(c))$, it will use its decode function to decode E_b

(x), then apply the sender's public function $E_c(x)$ to get c. Note that $E_c(x)$ is public and it is available to the receiver. This is enough to convince the receiver the message is indeed from the sender because of the following. If the message was not sent by the sender, then the resulting message $E_c(D_x(c))$ would not make any sense to the receiver, D_x is the decryption function of some other sender [1] [6].

4.2 Smart Cards

Another real world application of RSA is smart cards. Smart cards are like credit cards except it has an integrated circuit embedded in them. Some smart cards even have a coprocessor build in them. A coprocessor in this case is a processor specialized for encryption. If a smart card with a tamper resistant area is needed, then we can use RSA on the smart cards. C. Lu, et. al. [7] proposed two simple and yet efficient algorithms for key generation on the smart cards. The key will be generated in matter of seconds, and the length of key will increase every few years as time goes on. The primes generated are around 1024 bits, and the primes are generated every time the smart card is used to increase security. Both of their algorithm involves in generating a random number in the desired range and testing it with sieve of Eratosthenes. After it passes the initial test, more probabilistic tests were done to the prime to ensure the prime is prime enough to be used.

4.3 Shared Key Generations

RSA keys can be generated by a number of parties. At the end of the computation, all of the parties are convinced that n is the modulus but none of the

parties know the factorization of n . Parties compute e together as well, but they each share the corresponding part of the private component d . An article by D. Boneh and M. Franklin proposed an efficient generation of shared RSA keys [5]. The parties will produce all the public keys without any one party knowing the private keys. The way it works is that each party i picks a secret number p_i with a desired length. Let $p = \text{sum of } p_1 \text{ to } p_k$. If p passes the trial division test, which is a type of probabilistic primality test, then the parties solve for q with the same procedure. Note how not one party knows exactly what p or q is. Let $n = (p_1 + \dots + p_k) * (q_1 + \dots + q_k)$. After n is computed, it goes through a biprimality test, where k parties engage in a private distributed computation to test that n is indeed the product of two primes. If the test fails, then p and q must be regenerated again. Once n passes the biprimality test, the parties engage in a private distributed computation to generate a shared secret decryption exponent d . This is useful when a group of companies want to share a set of RSA keys, and every party is needed to compute the decode function $D(x)$.

5. Attacks on RSA

As computer scientists invent new ways to generate keys, their algorithms are not always secure. Research is not only done on improving RSA algorithm, some research is done to disprove previous research. We will look closely into an article from Merkel [4] regarding attacks on RSA.

In many applications smart cards are required to generate RSA signatures. Due to the computation limits on today's smart cards, this task is done on a more powerful

but untrusted server. The idea is that the server assist the smart cards to generate keys without having enough information to generate the secret key.

There are two types of attacks on server aided RSA Protocols. Passive attacks only use the data obtained by the server during correct executions of protocol. Active attacks use information obtained by the server deviating from the specification of the protocol.

There are many server aided RSA protocols have been proposed since the birth of RSA, and not every one of them are secure. Merkle [4] shows how multi-round passive attacks on some server-aided RSA protocols can easily factor the modulus n . Merkle presented two attacks, one attack on the protocol RSA-S1, and another attack on the HSLY-protocol.

The RSA-S1 protocol uses the server to help a client generate RSA signatures. The client sends n , and d_1, \dots, d_m , such that d_1, \dots, d_m is $(\phi(n) - 1)^i$. The server then returns z_1, \dots, z_m , where $z_i = (\text{original message})^{d_i}$ modulo n . Assuming the protocol has been executed several times, the server has pairs of decomposition. The secret key d 's interval is then estimated from the information the server has, which yields a non-trivial knapsack problem. Once the knapsack problem is solved, d is cracked. RSA-S1 requires the attacker to have information on what the server is receiving from the client, once enough information has been received, the calculation of the private key d can begin. Merkel also proposed a countermeasure for his own attack by modifying how the client transmit parts of d to the server.

The HSLY-protocol is very similar to RSA-S1 protocol, with the exception that in HSLY-protocol many random numbers are generated, but they are only generated once. Client produces the number u by performing complex calculations from the random numbers generated, these calculations are also performed only once. The number u is then used to assist in sending information to the server. Merkel shows that the random numbers can be canceled out by using the information received in only two protocol requests, which leaks out crucial information about d . Variable n can then be factored in 2^{37} steps, instead of the 2^{64} steps the authors of HSLY-protocol had claimed.

6. Conclusion

Although the RSA encryption algorithm was originally developed in the 1970s, its basic algorithm has never changed. Let $E(x)$ denote the encryption function and $D(x)$ denote the decryption function. To encrypt a message c , $E(c) = c^e$ modulo n . $E(c)$ is then sent to the receiver through a data path where the message could be received by unwanted parties. To decrypt a message $E(c)$, $D(E(c)) = (E(c))^d$ modulo $n = c$.

Despite the fundamental idea of RSA remained the same from the 70s, new ways of generating primes and using RSA in practical situations was developed since then. Messages can now be signed, so that receiver can verify the legitimacy of the message. Primes can be generated very efficiently, and can even be generated with the help of another machine without leaking crucial information. RSA keys can now be generated and shared by different parties, such that all parties must participate in generating the

keys and decrypting the message. Not all implementations of RSA are secure, some of them have been proving to be insecure.

All in all, there are still much research to be done in this field, since the importance encryption and privacy are becoming much more important. There are many real world applications could use an better encryption scheme, imagine what would the world be like if emails were to have signatures and encryption schemes. There are many applications that use RSA, and there will be many more to come.

7. References

- [1] S. C. Courtinho. *The Mathematics of Ciphers: Number theory and RSA Cryptography*. A K Peters, Ltd. 1999
- [2] D. R. Stinson. *The Cryptography: Theory and Practice*. CRC Press, Inc. 1995
- [3] A. Salomaa. *Public-Key Cryptography*. Springer-Verlag Berlin 1996
- [4] Merkle, J. “Multi-round passive attacks on server-aided RSA protocols” presented at Proceedings of the 7th ACM conference on Computer and communications security. Pages 102-107, 2000.
- [5] D. Boneh, M. Franklin. “Efficient generation of shared RSA keys” on the *Journal of the ACM*. Pages 702-722, 2001.
- [6] R. Cramer, V. Shoup. “Signature schemes based on the strong RSA assumption” at the Proceedings of the 6th ACM conference on Computer and communication security. Pages 46-51, 1999.

- [7] C. Lu, A. L. M. Santos, F. R. Pimentel. “Implementation of fast RSA key generation on smart cards” presented at the Proceedings of the 2002 ACM symposium on Applied computing. Pages 214-220, 2002.