RSA Crypto System

November 1, 2018

Abstract

Diffie-Hellman key exchange protocol over insecure channel gave rise to the exciting field of public key cryptosystem. The first practical scheme was devised by Rivest, Shamir and Adleman in 1977, based on the assumption that factorization of sufficiently large integer into their prime factors is hard. As a engineer, security practitioner, it is vital for us to understand RSA cryptosystem thoroughly. In this article, I explain RSA crypto primitives, correctness proof, and various known attacks. I also present benchmarking results and highlight performance of RSA crypto primitives.

1 Introduction

Primary application of Cryptography is to enable two parties share data confidentially.

- *Confidentiality* - ensure message remain hidden from all adversaries observing the channel
- *Integrity* - ensure message is not tampered by any adversary controlling the channel
- *Authentication* - enable receiving entity to verify the sending entity
- *Non-repudiation* - ensures the recipient that the sender can not later denial of the message sent

There are two types of crypto systems: *Symmetric* and *Asymmetric* deployed in practice. In symmetric cryptography, parties first share the key and use the same key to encrypt/decrypt the messages over channel. For example, when user pay using chip card via insert/tap, chip uses the secret key to prepare cryptogram (MAC) and send to POS terminal for authorization. In this article, we focus on asymmetric cryptography and attempt explain RSA crypto system thoroughly.

1.1 Asymmetric cryptography

Asymmetric cryptography is also known as public key cryptography(Shown in Figure .1). Two parties(e.g. Alice and Bob) generate key pair(secret and public key) individually, and share public keys with each other. Alice uses Bob's public key to encrypt the message and Bob decrypt the message using his secret key.

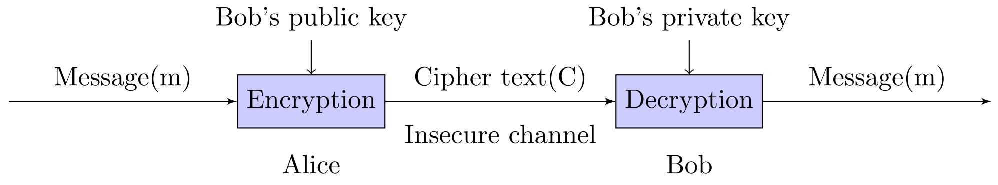

Figure 1: Public key cryptography

In the case of symmetric key cryptography Alice and Bob share a secret key via secure channel. The secret key can be used to encrypt as well as to decrypt the message from ciphertext. In general, symmetric key cryptography is significantly faster than asymmetric cryptography.

1.1.1 Advantages of Asymmetric cryptography

- In asymmetric key cryptography there is no need to share secret keys among parties a priori, thus eliminates the key distribution problem.
- In asymmetric key cryptography each user has to maintain a key pair(public key, private key). public keys are accessible to the public, so anyone can send a message to the recipient by using recipient's public key. Where as in symmetric key cryptography for 'n' number of users, each user has to manage (n-1) number of keys. In asymmetric key cryptography, one has to manage just one private key securely.

1.2 Security Assumptions :

Security of the Public key cryptosystem can be realized using various computationally hard problems — factorization (RSA), DLP (Discrete Logarithm Problem). In this article our main concern is RSA asymmetric algorithm.

2 Security definitions

2.1 Cipher text only attack(COA) or known cipher text attack :

In cipher text only attack, attacker has access to the set of cipher texts. If the attacker is able to get the plain text or the key that is used to decrypt the message with the available cipher-text then the attack is known as cipher text only attack. Every modern cipher tries to provide protection against cipher text only attack because the attack can be easily carried out. The attacker just need to eavesdrop on the channel to get the cipher text.

2.2 Known plain text attack :

Attacker has access to some of the plain texts and it's corresponding cipher-texts. With this information, if the attacker is able to decode some other cipher text which is generated using the same key then the attack is called as known plain-text attack. In order to carry out this attack, attacker just needs to eaves drop on the public communication channel to get cipher texts. Corresponding plain texts often can be guessed in some cases. For example, every conversation starts with hi or hello. All classical ciphers are susceptible to known plain text attack. For example shift cipher(Ceaser cipher), If you are able to decode one letter from the cipher text then you could simply decrypt entire cipher text.

2.3 Chosen plain text attack :

Attacker some how influence the sender to encrypt the plain text of his choice. The corresponding cipher-text can be observed by the attacker when it was sent over the communication channel. With this information attacker tries to get the information about private key and about the system.

2.4 Chosen cipher text attack :

Chosen cipher text attack is even more powerful than the chosen plain text attack. In chosen cipher text attack, attacker able to get the decryption of cipher texts of his choice in addition to the encryptions of the plain texts of his own choice. Adversary have an access to these encryption and decryption blocks without any time limit. But attacker can not request these decryption oracle to get the plain text of the challenged cipher text. Chosen cipher text security(CCA-secure) possesses a property non-malleability. A non-malleable system is that, if a attacker modifies a given cipher text(it does not look at all like challenged cipher text) and send it to the decryption oracle. The decryption oracle either produces a message invalid cipher text or the decryption of cipher text(plain text). But the produced plain text should not have no relation to the original plaint text.

3 RSA Assumptions :

Security of RSA depends on the difficulty in factoring large integers.

For Example: p=885320963, q=238855417 then n=$p*q$=211463707796206571.

Given 'n', there is no known polynomial time algorithm exists to find p and q. If you have given n then coming with two unique prime numbers where it's product is equal to n, is very difficult. As 'n' value increases the difficulty of factoring that number goes to exponential increase.

On intel core machine, i5 (@ 1.8 GHz), it takes 34 ms to factorize 20 bit number. where as to factor 1024 bit number it takes $(2^{1024}/2^{20})*34ms = 5.83*10^{300}$ years approximately.

3.1 Text book RSA

3.1.1 Key generation :

1. Choose two large prime numbers p, q.
2. Calculate n, $\Phi(n)$. where $n = p*q$. $\Phi(n) = (p-1)(q-1)$ here $\Phi(n)$ is Euler's totient function.
3. Choose 'e'(encryption exponent) randomly such that $1 < e < \Phi(n)$ and gcd$(e, \Phi(n))$=1.
4. Calculate 'd'(decryption exponent) such that $e*d$= 1 (mod($\Phi(n)$)).
5. Public key p_k= (n,e).
6. Private key s_k=(n, d).

3.1.2 Encryption :

Let say Alice wants to send message 'm' to Bob. Alice uses Bob's public key(p_k) to encrypt message 'm'.
Cipher text(c)=m^e(mod n).

3.1.3 Decryption :

After Bob received a message from Alice, Bob uses his private key(s_k) to decrypt message.
message(m)=c^d(mod n).

3.1.4 Correctness of RSA

Let's take cipher text 'c'. In order to recover message 'm', Bob computes

1. $c^d = (m^e)^d$=m^{ed}(mod n).

   we know, $e*d$=1 (mod $\Phi(n)$)=1+k$\Phi(n)$. where k is some integer.

2. where c^d=m^{ed}(mod n)=$m^{1+k\Phi(n)}$ (mod n)=$m(m^{k\Phi(n)})$ (mod n).

   We know that, If x=a (mod p) and x=a (mod q), then x= a (mod $p*q$) where p, q are distinct prime numbers.

3. So if we could prove that c^d=m^{ed}(mod p)= m (mod p) and c^d=m^{ed}(mod q)= m (mod q) then automatically c^d=m^{ed}(mod n)= m (mod n).

4. Let's take c^d=m^{ed}(mod p)

case i : From Euler's theorem, if gcd(a, n)=1 then $a^{\Phi(n)}$ (mod n)=1 (mod n).

So, if gcd(m, p)=1 then $m^{\Phi(p)}$ (mod p)=1 (mod p).

c^d=$m(m^{k\Phi(n)})$(mod p)=$m(m^{k\Phi(p)*\Phi(q)})$ =$m*(1)^{k*\phi(q)}$(mod p)=m.

case ii : If gcd (m, p)=p then m= 0(mod p).

Therefore c^d=$m(m^{k\Phi(n)})$(mod p)=0 (mod p)=m.

Therefore c^d=m^{ed}(mod p)=m (mod p).

Similarly c^d=m^{ed} (mod q)= m (mod q)

Finally we could conclude that c^d=m^{ed} (mod $p*q$)= m (mod n).

3.2 Digital signature

Digital signature is used to sign electronic documents . It provides authenticity(i.e from whom we have received a message), integrity(message is not changed in the transit). Let say Alice wants to send message(m) to Bob. Alice uses Bob's public key(p_k) to encrypt the message. Bob receives the cipher text(c), but at this point Bob might not know that who sent the message. Because Bob's public key p_k is publicly available, any one can send a message. Digital signature help Bob to authenticate Alice, before processing the encrypted message.

3.2.1 Plain RSA signature scheme

Let's discuss plain RSA signature scheme. It involves three steps.

- *Key generation:* Generate public key $p_k = (n, e)$, private key $s_k = (n, d)$.
- *Signature generation:* Let's take the message(m), private key s_k and compute signature. $\sigma = m^d$(mod n).
- *Signature verification:* By using public key p_k, compute σ^e(mod n), see whether it is equal to message m or not. If it is equal then we can say that this is valid signature.

But this signature scheme is vulnerable to no-message attack and attacker is able to forge a signature on an arbitrary message.

- *No-message attack:* In this attack, attacker forge a signature just with a knowledge of public key. First attacker calculate uniform σ then calculates message using public key $p_k = (n, e)$ as m=σ^e(mod n). Attacker sends (m, σ) to the recipient. Here σ is a valid signature but it was not issued by a actual sender. In this attack, attacker chooses σ in such a way that to influence the resulting message.

- *Forging a signature on an arbitrary message :* This one is even more powerful than the previous attack. Here attacker requests signatures on 'n' number of random messages M=$m_1, .._m\, n$ from the sender. With this information attacker could generate signatures on $2^n - n$ number of other messages which are generated from product of subsets of M.

3.2.2 RSA-FDH signature scheme

In order to protect from the above mentioned attacks, the message(m) undergoes some transformation before signing it.

- *Signature generation:* Initially message(m) is sent to a hash function(H). The generated hash(H(m)) is encrypted by using Alice's private key(s_{k_A}=(n, d)). Finally we get signature(σ).

  σ=$H(m)^d$(mod n).

  Alice takes message, digital signature (m,σ) and again encrypts using Bob's public key(p_{k_B}). The generated cipher text(c) is send to the Bob.

  After Bob receives a cipher text(c), decrypt the cipher text(c) by using Bob's private key. He gets message(m), digital signature(σ).

- *Verify a digital signature:*

  Step 1:Take the message(m) and passed to the hash function. Finally it generates some hash code(H(m)).

  Step 2:Take the Digital signature(σ) and decrypt it using Alice's public key . Finally we get some hash code.

  If the hash code generated by the Step 1 and Step 2 are equal then we can say that signature is valid.

  $\sigma^e = H(m)$(mod n).

3.3 Attacks against Text Book RSA

3.3.1 Malleability Attack :

Text book RSA is malleable. A crypto algorithm is said to be malleable, if the attacker is able to change a cipher text(c) in such a way that it could produce related plain text. Attacker doesn't have any knowledge about plaintext. Let say cipher text(c)=m^e. Attacker changed it as c'=$2^e * c = (2 * m)^e$. Finally receiver decrypts c' and gets plaintext 2m. Malleability is not a desired property, for example if the message (m) is a bank transaction, attacker could simply change the amount. So this attack rises serious concerns particularly in the case of sensitive applications.

3.3.2 Chosen cipher text Attack :

Let say attacker have the information about cipher text(c)=m^e. Attacker wants to change the cipher text(c) as $c' = m^e * r^e$, where 'r' is some random number. Attacker may ask the person who have private key(d) to decrypt the unmalicious-looking cipher text(c'). Attacker gets $c'^d = m * r$. Dividing above equation by 'r', attacker finally able to get message 'm'.

3.3.3 Timing Attack :

This is one of the side channel attacks. Side channel attacks can be performed only if you know the details of the computer system in which the cryptographic algorithm is running. With this information attacker able to find the time needed to decrypt the several cipher texts and finally able to find out the secret key.

In order to protect from this attack, we can make sure that every decryption of the cipher text takes constant time. But it effects the system performance. Rather than the mentioned approach, RSA implementation uses 'Cryptographic blinding' technique. It works as follows, Instead of decrypting cipher text(c), first multiply with r^e where 'r' is a random number,'e' is encryption constant. We get $c' = c * r^e$, After decrypting $c'^d = m * r$. Multiply c'^d with r^{-1} we get message(m).

3.4 Practical RSA

3.4.1 RSA PKCS#1 v1.5 Padding Scheme

A crypto system is said to be semantically secure, if it could produce different cipher texts for the same plaintext. Let say plaintext(m) is encrypted and produced cipher text(c), If you encrypt 'm' again then it should produce some other cipher text(c'). So attacker doesn't get a chance to learn about plain text. Our Text book RSA is not semantically secure because encrypting the same plain text produces same cipher text.

In order to solve this problem, standards such as PKCS was developed to pad the message(m) with random number of bits before going to the RSA encryption.

We know RSA public key is p_k=(e, n), where 'e' is the encryption exponent, n is a large prime number. let k denotes length of 'n' in bytes. Let's take message 'm' to be encrypted of its length(l) in multiples of bytes and in the range of one to k-11 bytes. 'r' is randomly generated byte string of it's length is equal to the k-l-3 and none of its bytes should not be 0x00.

$\hat{\text{m}}$= [(0x00‖0x02‖r‖0x00‖m)]

The resulting message $\hat{\text{m}}$ is constructed with, initial two blocks of length 2-bytes(0x00$\|$0x02) then concatenated with randomly generated byte string 'r' and one byte of zero's followed by message 'm'.

Cipher text c=$\hat{\text{m}}^e$ (mod n).

If the random padding is too short then PKCS#1 v 1.5 is not CPA-secure and liable to many attacks. So length of random padding 'r' should be at least $\|$n$\|$ /e.

Decryption of cipher text: c^d(mod n)=$\hat{\text{m}}$. message 'm' can be recovered as the least significant bits(k - length of 'r'-2) of $\hat{\text{m}}$. When the receiver decrypts the cipher text, First receiver checks the most two significant bytes values and their values should be equal to the 0x00$\|$0x02. If not, receiver send a error message 'invalid cipher text'. Let see how this error message leads to serious consequences.

let say attacker eavesdrop on the public communication channel and got a cipher text 'c' which he wants to decrypt it. Attacker pick a random number r' and calculates $c' = r'c$. The resulting cipher text c' is sent to the receiver. The receiver checks whether c' in proper format or not. If not, receiver send a error message. Based on the receiver response attacker come to know that the decryption of c' can generate most two significant bytes values equal to the 0x00$\|$0x02 or not. Attacker also have his decryption oracle which checks the decryption of c''s most two significant bytes are equal to 0x00$\|$0x02 or not. Bleichenbacher proves that the attacker is able to decrypt 'c' using his decryption oracle.

In order to protect from these type of attacks a new standard *optimal asymmetric encryption padding*(OAEP) was developed.

3.4.2 RSA OAEP

RSA OAEP scheme follows the same idea as in the RSA PKCS#1 v1.5, it takes the message 'm', converts 'm' into $\hat{\text{m}}$ and then encrypt $\hat{\text{m}}$ to generate cipher text(c). But converting message(m) to $\hat{\text{m}}$ follows complex procedure than the RSA PKCS#1 v1.5.

- Let l is the length of message(m) in bits. Concatenate message(m) with k_1 number of zero's. we get m'=m$\|0^{k_1}$.
- Let 'r' is a random bit string of length 'k_0'. The length of l+$k_0 + k_1$ should be less than the number of bits of RSA modulus. Here 'r' is sent to a hash function(G), we get random bit string G(r) of length l+k_1 .
- Calculate s=$m' \oplus G(r)$.
- Here calculated s is sent to some other hash function(H). The output of the hash function is H(s) of length k_0.
- Calculate t=$r \oplus H(s)$.
- Calculate $\hat{\text{m}}$=$s \parallel t$.

- Encrypt m̂ with encryption key 'e', we get cipher text c=m̂^{e} (mod n).
- In order to recover message, receiver decrypts cipher text c^d(mod n)=m̂. Receiver knows the length of s and t, so he could able to recover s and t from m̂.
- Calculate r=H(s)$\oplus$ t, m'=G(r)$\oplus$s.
- After recovering m', receiver checks whether the k_1 number of its tailing bits are equal to 0 or not. If not, the receiver send a error message 'not valid cipher text'. if yes, receiver strips off the k_1 number of least significant 0's. The remaining l bits of m' is the actual message(m).

3.5 RSA Implementation Issues

In RSA based crypto system, receiver who have the cipher text(c) can use *Chinese remainder theorem* for efficient decryption.

Decryption of Cipher text(c)=c^d(mod n)$\leftrightarrow$ (c^d (mod p),c^d (mod q)), where n=pq.

$m_p = c^d$ (mod p)=$c^{d(mod(p-1))}$ (mod p),

$m_q = c^d$ (mod q)=$c^{d(mod(q-1))}$ (mod q).

By using Chinese remainder theorem we can get $m \leftrightarrow (m_p, m_q)$.

For m-bit integer, exponentiation modulo takes $k * m^3$ operations, where k is some constant. We know that n=$p * q$, assume that each one p, q are m-bit long. So 'n' is 2m-bit long and its c^d (mod n) require $k * (2m)^3$= $k * 8m^3$ operations. By using Chinese remainder theorem c^d (mod p), c^d (mod q) each require $k * m^3$ operations, in total we require $k * 2m^3$ operations. Here by using Chinese remainder theorem, we are saving 3/4 th of the time.

For encryption constant e=65537=$2^{16}+1$, encryption operation require total 17 multiplications which are far less than the multiplications required for randomly selected 'e'. By using the mentioned value for 'e' we could increase the efficiency of encryption scheme.

3.6 Benchmarking

I implemented the RSA crypto system using JAVA on Intel machine(i5,@ 1.8GHz). The following figure highlights the performance of encryption, decryption, signature generation and verification. Results show that RSA performs poorly as the key size increases from 512 bits to 4096 bits. In the below figure X- axis indicates key length, Y-axis indicates number of operations(encryptions/decryptions/signature generations and verifications) per second.

3.7 Conclusion

RSA crypto system is deployed widely and it is strongly recommended for software engineers, practitioners and security enthusiasts to understand the system thoroughly. In this article, we explained text book RSA primitives, security definitions, various attacks and practical implementation of the RSA.

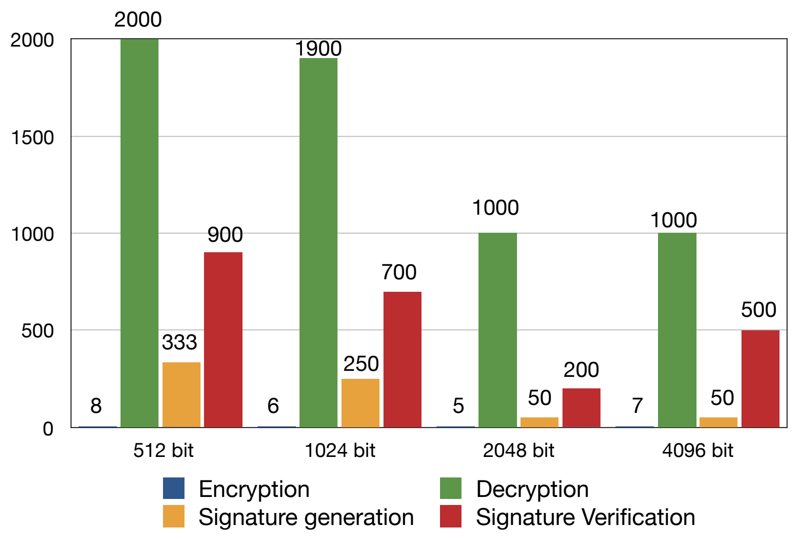

Figure 1: RSA Benchmarking.

References

[1] Lindell, Yehuda, and Jonathan Katz. *Introduction to modern cryptography.* . Chapman and Hall/CRC, 2014.

[2] Boneh, Dan. *Twenty years of attacks on the RSA cryptosystem.* Notices of the AMS 46.2 (1999): 203-213.

[3] `https://en.wikipedia.org/wiki/RSA_(cryptosystem)#Padding_schemes`

[4] `https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding`

[5] `https://en.wikipedia.org/wiki/Padding_(cryptography)`

[6] `https://crypto.stackexchange.com/questions/1448/definition-of-textbook-rsa`