Journal of Computer Science 2 (8): 665-671, 2006
ISSN 1549-3636

Mathematical Attacks on RSA Cryptosystem

[1]Imad Khaled Salah, [2]Abdullah Darwish and [3]Saleh Oqeili
[1]University of Jordan, Jordan
[2]Royal Scientific Society, Jordan
[3]Amman Arab University for Graduate Studies, (sabbatical leave from Al Balqa Applied University), Jordan

Abstract: In this paper some of the most common attacks against Rivest, Shamir, and Adleman (RSA) cryptosystem are presented. We describe the integer factoring attacks, attacks on the underlying mathematical function, as well as attacks that exploit details in implementations of the algorithm. Algorithms for each type of attacks are developed and analyzed by their complexity, memory requirements and area of usage.

Key words: RSA, cryptography, cryptanalysis, attack, factorization, low-exponent, one-way function

INTRODUCTION

While cryptography is the science concerned with the design of ciphers, cryptanalysis is the related study of breaking ciphers. We begin by describing a simplified version of RSA encryption. Let N = p*q be the product of two large primes of the same size. Let e, d be two integers satisfying e*d≡1 mod Ø(N) where Ø(N) = (p-1)*(q-1) is the Euler's totient function of N. We call N the RSA modulus, e the encryption exponent, and d the decryption exponent. The pair (N,e) is the public key which is used to encrypt messages. The pair (N,d) is called the private key and is known only to the recipient of encrypted messages.

A message is an integer M, to encrypt M, one computes $C = M^e \bmod N$. To decrypt the ciphertext C, the legitimate receiver computes $C^d \bmod N$.

In an encryption scheme, the main objective of the attacker is to recover the plaintext m from the related cipher text. If he/she is successful, we say he/she has broken the system. In the case of digital signature, the goal of the attacker is to forge signatures. A more ambitious attack is to recover the private key d. If achieved, the attacker can now decrypt all cipher texts and forge signatures at will. In this case the only solution is to revocation of the key.

This research gives a brief description of the main attacks against RSA cryptosystem. Some of integer factoring attacks, attacks on the underlying mathematical function and attacks which exploit implementation are presented. This study superiors from others by writing simple algorithms and analysis for each attack. Some of these attacks apply only to the encryption scheme, some result in the private key recovery[1-3].

Integer Factoring Attacks: The problem of integer factorization is one of the oldest in number theory and the adverts of computers have simulated considerable progress in recent years. However, the security of many cryptographic techniques depends upon the intractability of the Integer Factorization Problem (IFP). The security of RSA cryptosystem is initially related to the IFP. If an adversary can factor the public modulus, N into its two prime factors, p and q, he can efficiently calculate the private exponent.

Factoring algorithms comes in two parts: special purpose and general purpose algorithms. The efficiency of special purpose depends on the unknown factors, whereas the efficiency of the latter depends on the number to be factored. Special purpose algorithms are best for factoring numbers with small factors, but the numbers used for the modulus in the RSA do not have any small factors. Therefore, general purpose factoring algorithms are the more important ones in the context of cryptographic systems and their security. Table 1 summarizes the running time for integer factoring algorithms. The first four rows are special purpose algorithm, and last rows are general purpose algorithms.

The notation o(1) denotes a function of N that approaches 0 as $n \to \infty$, p denotes the smallest prime factor of N, and Euler's constant e=2.718.

Table 1: Factoring algorithms running time

Pollard's Rho algorithm	$O(\sqrt{p})$
Pollard's p-1 algorithm	$O(p^{\sim})$ where $p^{\sim}$ is the largest prime factor of p-1.
Pollard's p+1 algorithm	$O(p^{\sim})$ where $p^{\sim}$ is the largest prime factor of p+1.
Elliptic Curve method (ECM)	$O(e^{(1+o(1))\,(2\ln p \ln\ln p)^{1/2}})$
Quadratic Sieve (Q.S.)	$O(e^{(1+o(1))\,(\ln N \ln\ln N)^{1/2}})$
Number Filed Sieve (NFS)	$O(e^{(1.92+o(1))(\ln N)^{1/3}(\ln\ln N)^{2/3}})$

Corresponding Author: Saleh, Oqeili, PhD, Al-Balqa Applied University, Jordan

Wiener's Attack: To reduce the work load of the exponentiation, one may wish to use a small value of private key rather than a random value[4]. Since modular exponentiation takes time linear in log (private key), a small private key can improve performance by at least a factor of 10 (for 1024 bits modulus). For instance, if a smart card is used to sign messages, it will have to compute exponentiations C^d mod N, where C is a cipher text, d is private key and N is RSA modulus. If the card has limited computing power, a relatively small value of d would be handy.

In this section, we present an attack, due to Wiener [4], that succeeds in computing the secret decryption exponent under certain conditions.

Theorem 1: Let N=p*q with q < p< 2*q, let d < $1/3*N^{0.25}$. Given public key (N,e) with e*d≡1 mod Ø(N). Attacker can efficiently recover d.

Proof: The proof is using continued fractions technique. Since d is calculated in equation e*d≡1 mod Ø(N), it follows that there is an integer k such that e*d-k*Ø(N)=1. Therefore, we have that:

$$\left|\frac{e}{\text{Ø(N)}}-\frac{k}{d}\right|=\frac{1}{d*\text{Ø(N)}}.$$

Since $N=p*q>q^2$, we have that $q<\sqrt{N}$, hence:

$0<N-\text{Ø(N)}=p+q-1<2*q+q-1<3*\sqrt{N}$. Now, we see that

$$\left|\frac{e}{N}-\frac{k}{d}\right|=\left|\frac{e*d-k*N}{d*N}\right|=$$

$$\left|\frac{1+k*(\text{Ø(N)}-N)}{d*N}\right|<\frac{3*k*\sqrt{N}}{d*N}<\frac{3*k}{d*\sqrt{N}}.$$

Since k<d, we have that $3*k<3*d<N^{0.25}$, and hence

$$\left|\frac{e}{N}-\frac{k}{d}\right|<\frac{1}{d*N^{0.25}}.$$

Finally, since $3*d<N^{0.25}$, we have that

$$\left|\frac{e}{N}-\frac{k}{d}\right|<\frac{1}{3*d^2}$$

As note from mathematical proof there are at most log N fractions $\frac{k}{d}$ with d < N approximately $\frac{e}{N}$ so tightly, and they can be obtained by computing the log N convergents of the continued fraction expansion of $\frac{e}{N}$.

Algorithm: (Wiener's attack).

Input:a public key (N,e),a continued fractions of N/e:$[q_1,q_2,\ldots,q_m]$.

Output:a non-trivial factors p and q of N.

Algorithm:

Set $c_0=1$, $c_1=1$, $d_0=0$, $d_1=1$, i=1;

while i<=m do

Calculate $z=(c_i*e-1)/d_i$;

If z is an integer then

Let p and q be the roots of the equation: $x^2-(N-z+1)x+N=0$;

If p and q are positive integers then return (p,q);

i=i+1;

$c_i=q_i*c_{i-1}+c_{i-2}$;

$d_i=q_i*d_{i-1}+d_{i-2}$;

return "failure";

End Algorithm (Wiener's attack).

Note that this attack is efficient and practical, and thus is a concern only if the private key d is chosen to be small relative to N. For example, if N is a 1024 bits number, then d must be at least 256 bits long in order to prevent Wiener's attack. Wiener proposed[4] certain techniques that avoid his attack The first technique is to use a large encryption exponent, say $e\tilde{}=e+c*\text{Ø(N)}$ for some large c. For a large enough $e\tilde{}$, the factor k in the proof is so large the Wiener's attack can not be mounted, regardless of how small d is.

A second technique uses the Chinese Remainder Theorem (CRT) to speed up decryption, even if d is not small. Let d be a large decryption exponent such that both $d_p=d$ mod p-1 and $d_q=d$ mod q-1 are small. Then, can decrypt a given cipher text C as follows. Compute $m_p=C^{dp}$ mod p and $m_q=C^{dq}$ mod q, and use the CRT to obtain the unique solution m modulo N=p*q of the two equations $m=m_p$ mod p and $m=m_q$ mod q. The point is that although d_p and d_q are small, d can be chosen to resist Wiener's attack.

Boneh and Durfee[5] show that as long as $d<N^{0.292}$, an attacker can efficiently recover d from (N,e).

Low Public Exponent Attacks: A user of the RSA cryptosystem may wish reducing encryption or signature-verification, it is customary to use a small public exponent e. The common choices of a public exponent e are 3 or $2^{16}+1$. When the value $2^{16}+1$ is used, signature verification requires 17 multiplications, as opposed to roughly 1000 when a random e<=Ø(N) is used. If the public exponent is small and the plaintext m is very short, then the RSA function may be easy invert. Unlike Wiener's attack, attacks that apply when a small e is used are far from a total break. In this section, we explain some of these attacks and show how they work.

Hastad Broadcasting Attack

Theorem 2: Suppose $N_1,N_2,\ldots,N_k$ are relatively prime integers and set N_{min}=minimum (N_i). Let $g_i(x) \in Z_{Ni}[x]$ be k polynomials of maximum degree d. Suppose there exists a unique $m < N_{min}$ satisfying: $g_i(m)=0$ mod N_i for all $i \in \{0,\ldots,k\}$. Furthermore, suppose k>d, there is an efficient algorithm which given $(N_i,g_i(x))$ for all i, computes m.

Proof: The mathematical proof can be found in[6].

In other words, if three parties participating in the same system encrypt the same message m using the same public exponent e=3, although perhaps different modulus n_1,n_2, and n_3, then one can easily compute m from the three cipher texts:
$c_1=m^3 \bmod n_1$
$c_2=m^3 \bmod n_2$
$c_3=m^3 \bmod n_3$
Using CRT one can compute the unique solution $C=m^3 \bmod n_1*n_2*n_3=m^3$. Hence, one can compute m from C by ordinary root extraction.

Algorithm: (Hastad Broadcasting Attack)
Input:a system of cipher texts equations with different modulus.
Output: an original message m.

Algorithm:
For simplicity choose e=3.
Given the following equations:
$c_1=m^3 \bmod n_1$
$c_2=m^3 \bmod n_2$
$c_i=m^3 \bmod n_i$
compute $N=n_1*n_2*\ldots*n_i$
compute $M_i=N/n_i$, for i=1,2,…
compute $y_i \equiv m_i^{-1} \bmod n_i$, for i=1,2,…
compute $X=(c_1*M_1*y_1+c_2*M_2*y_2+\ldots+c_i*M_i*y_i) \bmod N$.
Compute original message $m=X^{1/3}$.

End Algorithm (Hastad Broadcasting attack).

Hasted shows[6] that small public exponents can be dangerous when the same plaintext is sent to many different recipients, (i.e.) this attack works in efficient manner where public exponent e is small. To foil this attack, we can use larger exponents or send not exactly the same message by adding a time-stamp, for example, this latter solution does not always prevent the recovery of messages. Suppose that the messages are linearly related: $m_i=\alpha_i*m+\beta_i \bmod n_i$, where α_i and β_i are known constants. The corresponding cipher texts are $c_i=m_i^{ei} \bmod n_i$. In this case, we have:

Corollary 1: In the RSA cryptosystem, a set of k linearly related messages encrypted with public encryption keys e_i and RSA-moduli n_i can be recovered if $k>e*(e+1)/2$ and $n_i>2^{(e+1)*(e+2)/4}\ (e+1)^{e+1}$, where $e=\max(e_i)$.

Table 2 summarizes the number of messages required in the RSA cryptosystem to mount successful Hastad's attack. Note that, the "-" means that the RSA system is not defined for this value of e.

Hastad's attack depends on CRT step to recover original message, so it needs $O((\log N)^2)$ bit operations in total, where $N=\prod_{i=1}^{k} n_i$.

Table 2: Basic Hasted's attack

E	No. of messages
2	-
3	7
4	-
5	16
7	29
33	562

Franklin-Reiter Related Messages Atack[7]
Theorem 3: Set e=3, and let (N,e) be an RSA public key. Let $m_1!=m_2 \in Z_N$ satisfy $m_1=f(m_2) \bmod N$ for some linear polynomial $f=a*x+b \in Z_N[x]$ with b!=0. Then, given (N,e,c_1,c_2,f), attacker can recover m_1, m_2 in time quadratic in log N.

Proof: Using an arbitrary e (rather than restricting to e=3). Since $c_1=m_1^e \bmod N$, we know that m_2 is a root of the polynomial $g_1(x)=f(x)^e-c_1 \in Z_N[x]$. Similarly, m_2 is a root of $g_2(x)=x^e-c_2 \in Z_N[x]$. The linear factor $x-m_2$ divides both polynomials.

Therefore, attacker calculates the greatest common divisor (gcd) of g_1 and g_2, if the gcd turns out to be linear, m_2 is found. The gcd can be computed in quadratic time in e and log N. Coppersmith[7] describes two methods to recover the plaintext m: Direct method and gcd method. Here, we write a Direct method steps and the latter method can be found in[8].

Algorithm: (Franklin-Reiter attack).
Input: Two ciphertexts c_1 and c_2, and f(m) such that: $f(m)=a*m+b$.
Output: an original message m.

Algorithm:
Calculate $f=b*(c_2+2*a^3*c_1-b^3) \bmod N$.
Calculate $g=a*(c_2-a^3*c_1+2*b^3) \bmod N$.
Find m such that $g*m \equiv f \bmod N$ using Extended Euclidean Algorithm.
End Algorithm (Franklin-Reiter attack).

Franklin-Reiter identified a new attack against RSA with public exponent e=3. If two messages differ only from a known fixed value Δ and are RSA encrypted under the same RSA modulus N, then it is possible to recover both of them. This situation occurs quite after, as for example, Letters sent to different addresses, Texts differing only from their date of compilation, Retransmission of a message with a new ID number due to an error.

Furthermore, Coppersmith[7] showed that if Δ (difference between the two messages) is unknown, then m_1 and m_2 can sometimes be recovered. In particular, this means that adding a random padding to the messages being encrypted does not always prevent the recovery of messages. It is estimated that, for RSA, this attack applies with a public encryption exponent e up to 32 bits. When e=3 the attack can be mounted as long as the pad length is less than $1/9^{th}$ the message

length, so, In general the related messages attack can be mounted if and only if the padding length is less than $\frac{N}{e^2}$ where (N,e) is a RSA public key. Table 3 shows this result.

Table 3: Random padding Δ tolerated for modulus N

E	N (in bits)	
	512	1024
3	56	113
5	20	40
7	10	20

The above attack takes $O((\log N)^2)$ time to calculate a gcd between two polynomials, for e >3 the attack takes time quadratic in e. Consequently, it can be applied only when a small public exponent e is used. For a large e the work in computing the gcd is prohibitive.

Partial Key Exposure attack

Theorem 4: Let N=p*q be an n-bit RSA modulus, let 1<=e, d<=Ø(N) satisfy e*d≡1 mod Ø(N). There is an algorithm that given the n/4 Least Significant Bits (LSB) of d computes all of d in polynomial time in n and e.

Proof: The proof of above theorem 4 is not provided here, it can be found in[9].

Algorithm: (Partial Key Exposure attack).
Input: a public key (N,e), let d_0=n/4 LSB of d.
Output: a private exponent d.

Algorithm:
$e*d_0=1+k*(N-s+1) \pmod{2^{n/4}}$.
Test a candidate value of k, 1<=k<=e.
Solve $p^2-s*p+N=0 \pmod{2^{n/4}}$.
Find $p_0=p \pmod{2^{n/4}}$.
Find q_0 such that $p_0*q_0=N \pmod{2^{n/4}}$.
Find x, y in polynomial $f(x,y)=(r*x+p_0)*(r*y-q_0)-N$, where $r= 2^{n/4}$.
Calculate $Ø(N)=(r*x+p_0-1)*(r*y+q_0-1)$.
Computes d in e*d-k*Ø(N)=1.
End Algorithm (Partial Key Exposure attack).

Like other methods for cracking RSA, this algorithm is effective under certain circumstances, when e is large this attack will fail.

One sees that the running time of this algorithm is most dependent on the first and last steps. The first step is solving a couple modular equations, the final step is actually factoring N, so the expected time for this attack is linear in O(e*log e). One searches for randomness in order to locate private keys in large volumes of data, such as the hard disk filling system, it should be clear how important the safe storage of the RSA private key is perhaps the best solutions is the use of tamper-resistant hardware modulus or tokens, in which the private key is securely stored and the private operation is performed.

Homomorphic Attacks: The inherent homomorphic structure of the RSA enables to mount some attacks. Let m_1 and m_2 be two plaintext messages, and let c_1 and c_2 be their respective the RSA encryptions. Observe that: $(m_1*m_2)^e \equiv m_1^e * m_2^e \equiv c_1*c_2 \pmod N$, this is called homomorphic property of RSA. In this section, we produce some attacks depend on this property of RSA.

Chosen Cipher text Atack: Given (N,e) as RSA public key, and cipher text c, then choose a cipher text and look at the plaintext, then repeat until they have figured out how to decrypt any message.

Algorithm: (Chosen Cipher text attack).
Input: a public key (N,e), and cipher text c.
Output: an original message m.

Algorithm:
Choose an integer randomly r less than N.
Compute three roots (x,y,z) as follows:
2.1. $x=r^e \bmod N$.
2.2. $y=x*c \bmod N$.
2.3. $z\equiv r^{-1} \bmod N$.
Send y to victim.
Victim computes $u=y^d \bmod N$, then send u to attacker.
Attacker recovers original message $m=z*u \bmod N$.
End Algorithm (Chosen Cipher text attack).

Usually, a Chosen Cipher text attack is based on the theoretical assumption that the attacker has access to a decryption device that returns the complete decryption for a Chosen Cipher text. Hence, if a public key cryptosystem is susceptible to a Chosen Cipher text attack, which often is considered to be only a theoretical weakness.

Chosen Cipher text attack requires more decryptions with each candidate key to identify the expected clear text statistics. In public key cryptosystems, it suffices to know the victim's public key, since the attacker can generate by himself the required clear text / cipher text pairs.

The main problem an applying this technique to the RSA scheme is that each modular exponentiations is very expensive, and, its time complexity grows cubically with the size N of the modulus. If we have to try about u possible substrings as candidate values for the decryption exponent d, we get a total complexity of $O(u*N^3)$, which is polynomial but impractical.

Common Modulus Attack: Let $c_1=m^{e1} \bmod N$, $c_2=m^{e2} \bmod N$ be the cipher texts corresponding to message m,

where $gcd(e_1,e_2)=1$, then attacker recovers original message $m=c_1^a*c_2^b \bmod N$ for $e_1*a+e_2*b=1$.

Algorithm: (Common Modulus attack)
Input: A modulus N, e_1, e_2, and two cipher texts c_1 and c_2, where $gcd(e_1,e_2)=1$.
Output: an original message m.

Algorithm: Find a, b such that: $e_1*a+e_2*b=1$, using Extended Euclidean Algorithm.
Computes original message $m=c_1^a*c_2^b \bmod N$.
End Algorithm (Common Modulus attack).

A Common Modulus attack can be used to recover the plaintext when the same message is encrypted to two RSA keys that use the same modulus. This algorithm works if and only if message sends with the same modulus and relatively prime encryption exponents. The main task of Common Modulus attack algorithm is computing a and b such that $e_1*a+e_2*b=1$, this process needs $O((\log k)^2)$, where k is maximum size of a or b.

Davida's Attack[10]: If the attacker can get access to the bin of victim, then he/she will be able to recover the original plaintext if the transformation is done in a clever way.

Algorithm: (Davida's attack)
Input: a public key(N,e), and a cipher text c.
Output: An original message m.

Algorithm: Attacker intercepts the cipher text c, and replaces it by $c\tilde{}=c*k^e \bmod N$, where k is a random number.
Victim receives $c\tilde{}$ and computes $m\tilde{}=c\tilde{}^d \bmod N$, since the message $m\tilde{}$ is meaningless it will be discarded.
If attacker can get access to m~, he/she van recover original message m by computing $m\tilde{}*k^{-1}\equiv c\tilde{}^d*k^{-1}\equiv c^d\equiv m \bmod N$.
End Algorithm (Davida's attack).

Davida's attack[10] is one kind of Garbage-man-in-the-middle attacks. The basic idea of these attacks relies on the possibility to get access to the bin of the recipient. In fact, if the cryptanalyst intercepts, transforms and re-sends a cipher text, then the corresponding plaintext will be meaningless when the authorized receiver will decrypt it, if the attacker can get access to this discard, he/she will be able to recover the original plaintext if the transformation is done in a clever way. In many situations, we can get access to the discards, as for example, Bad implementation of software or bad architectures, Negligent Secretaries**,** Recovering of previously deleted message by a tool like the <restore> command in Windows.

Note that, the last step of Davida's attack relies on the homomorphic nature of RSA, the running time of Davida's attack depends on this step which takes $O((\log N)^2)$ time to calculate modular inverse, where N is RSA modulus.

RSA Digital Signature Attacks: A digital signature of a message is a number dependent on some secret known to the signer, and additionally on the content of the message being signed.

In a digital signers environment the goal of an attacker is to forge signatures: that is, produce signatures which will be accepted as those of some other entity. In this section, we present two attacks against RSA digital signature scheme, the first is Blinding attack (or Chosen-Message attack) which is depending on homomorphic nature of RSA, the second is Lenstra's attach which is can applicable on all Chinese Remaindering based cryptosystems.

Blinding Attack:
Theorem 5: Given (N,e) as RSA public key, and message m, then it can recover an original signature s of message m as: $s=((m*r^e)^d)*r^{-1} \bmod N$, where r is a random number such that $0<r<N-1$.

Proof: Since gcd(r,e)=1, then by , there is u,v such that: $r*u+e*v=1$.
So, $d=d*(r*u+e*v)=d*r*u+v$ since $e*d\equiv 1 \bmod \Ø(N)$. Hence, signature $s\equiv m^d\equiv m^{d*r*u}*m^v\equiv(m^{d*r})^u*m^v\equiv s\tilde{}^u*m^v \bmod N$.

Algorithm: (Blinding attack).
Input: A public key (N,e), and a message m.
Output: An original signature of message m.

Algorithm:
Choose a random number r, such that $0<r<N-1$ and $gcd(e,r)=1$;
Compute $m\tilde{}=m*r^e \bmod N$, send $m\tilde{}$ to victim;
Victim computes $s\tilde{}=m\tilde{}^d \bmod N$, send $s\tilde{}$ to attacker;
Attacker computes the original signature s of message m as $s\tilde{}*r^{-1} \bmod N$;
End Algorithm (Blinding attack).

Let d victim's private key and (N,e) be his corresponding public key. Suppose an adversary wants victim's signature on a message m, being no fool, victim refuses to sign m. Attacker can try the following: he/she picks a random r such that $gcd(e,r)=1$ and sets $m\tilde{}=m*r^e \bmod N$, then asks victim to sign the random message $m\tilde{}$. Victim may be willing to provide his signature $s\tilde{}$ on the innocent looking $m\tilde{}$. But recall that $s\tilde{}=m\tilde{}^d \bmod N$, attacker now simply compute $s=s\tilde{}*r^{-1} \bmod N$ and obtains victim's signature s on the original message m. This technique, called Blinding because it enables attacker to obtain a valid signature on a message of his choice by asking victim to sign a random "blinded" message. Victim has no information as to what message he is actually signing.

Since most signature schemes apply a one-way hash function to the message m, the attack is not a serious concern. Although a useful property of the RSA needed for implementing anonymous digital cash (cash that can be used to purchase goods, but does not reveal the identity of the person making the purchase). Let x be a size of public exponent e, and y is a size of modulus N, then time necessary for this attack is $O(x*y^2)$.

Lenstra's Attack[11]: Theorem 6. The secret factors p and q of the public modulus N can be recovered when the faulty signature is known.

Proof: the proof does not provide here , it can be found in[11].

Algorithm (Lenstra's attack)
Input: A public key (N,e), message m, and CRT signature system.
Output: A factors p and q of N.
Algorithm: A CRT signature system consists of the following steps:
Compute $d_p=d \bmod (p-1)$, $d_q=d \bmod (q-1)$;
Compute $m_p=m \bmod p$, $m_q=m \bmod q$;
Compute $s_p=m_p^{dp} \bmod p$, $s_q=m_q^{dq} \bmod q$;
Calculate the message signature s using CRT technique by s_p and s_q.
Then, suppose that an error occurs during the computation of s_p but not during that of s_q, but applying CRT will give the faulty signature $s^{\sim}$ for message m.
Attacker computes the secret factor $q=gcd(s^{\sim e}-m \bmod N,N)$.
End Algorithm (Lenstra's attack).

In case of computation error, the researchers showed how to recover the secret factors p and q of the public modulus N from two signatures of the same message: a correct one and a faulty one; Lenstra showed[11] that actually only the faulty signature is required. Note that some attacks applies to decryption process, if the attacker has access to the faulty encryption.
Shamir presented a simple solution to prevent the previous attack. The signer first chooses a small random number r relatively prime to N. Then he/she computes:

$s_{rp}=m^{d \bmod Ø(r*p)} \bmod r*p$.
$s_{rq}=m^{d \bmod Ø(r*q)} \bmod r*q$.

if $s_{rp}=s_{rq} \bmod r$, then the computations are assured correct and s is computed by applying the CRT on $s_{rp} \bmod p$ and $s_{rq} \bmod q$.
In fact, Lenstra's attack is applicable on a the RSA hardware devices (say smart cards), So, it can be considered as implementation attack on the RSA cryptosystem.

Timing attacks: Most of the attacks against RSA we have seen so far apply to the underlying cryptographic primitives and parameters. On the other side, implementation attacks target specific implementation leaked by the implementation of the RSA function. The attacks are usually applied against smart cards and security tokens, and are more effective when the attacker is in possession of the cryptographic module. Prevention of implementation attacks is hard, trying to reduce the amount of information leaked.

In this section, we present a Timing attacks as a good example of implementation attacks. In 1996, Kocher[12] demonstrated that an attacker can determine a private key by keeping track of how long a computer takes to decipher messages. Timing attacks are alarming for two reasons: it comes from a completely unexpected direction and it is a cipher text only attack. In order to describe this attack, we need to explain how Square-and-Multiply exponentiation is carried out. In order to compute $m=C^d \bmod N$ for decryption a cipher text, we show that at stage i, if d[i]=0, then the value m is not modified. However, if d[i]=1, then we multiply the previous result by C^{2*i}. The Timing attack uses the fact that when d[i]=1, an additional multiplication takes place. Now, assume that an attacker holds a smart card that decrypts. Then, the attacker asks it to decrypt a large number of random messages $C_1,C_2,\ldots,C_k \in Z_N$ and measures the time T_i that is takes to compute $C_i^d \bmod N$. These timing measurements are now used to obtain d, one bit at a time. Since d is odd, we know that d[0]=1. Initially $m=C^2 \bmod N$. Thus, if d[i]=1, the smart cards computes the product $m=(m*C) \bmod N$; otherwise, it does not. Let t_i be the time that it takes to compute $C_i*C_i^2$ where C_i is one of the random messages that was initially chosen.

By measuring the correlation between t_i and T_i it is possible for attacker to determine if d[1] equals 1 or 0, given d[0] and d[1], it is possible to do the same thing for d[2] and so on.

In other words, the attacker can determine the particular sequence of squarings and multiplications that the program went through. Based on the outcome, he/she can simply compute the secret exponent d stored on the card. There are simple countermeasures that can be used to confuse the Timing attack such as random delay by adding a random delay to the exponentiation algorithm.

PREVENTION AND COUNTERMEASURE

Years of cryptanalysis of RSA have provided us some very clever attacks, and although no devastating attack has ever been found, there are a number of issues users and developers alike must be aware when working with RSA. Points that deserve special attention are: key size, properties of parameters (primes, exponents) and encoding details.

Factoring modulus N: given the best factoring methods, no one should use a 512-bit modulus for security. An organization that is willing to invest several million in hardware and give a few months to crack a key could do so. 1024-bit keys are safe at the moment. However, over the next 20 years, we must move towards 2048-bit keys. The rate of progress in number theory and factoring in particular has been faster than expected.

Low private exponent attacks: a low private exponent should never be used. This is bad news for smart cards. Boneh and Durfee[5] prove that private exponent must be greater than $N^{0.292}$.

Low public exponent attacks: a low public exponent should not be allowed. There is already a lesson we can learn, sending related messages is dangerous.

Homomorphic attacks: the remedy is simple; the attacker should never be able to obtain the raw decryption of an arbitrary value. Another lesson is that the use of the same cryptosystem for decryption and signature is definitively not a good practice.

RSA Digital Signature attacks: always destroy encrypted messages, especially ones that look like garbage. You never know when you are being attacked. Also, never sign any random messages.

Implementation attacks: illustrates that a study of the underlying mathematical structure is not enough.

CONCLUSION

The RSA cryptosystem is the "de-facto" standard for Public-key encryption and signature worldwide. We survey, present, and analyze the most common against RSA attacks. Integer factoring methods, attacks on the underlying mathematical function, as well as attacks the exploit details in implementations of the algorithm are presented. It was shown that no attack algorithm can break RSA cryptosystem in efficient manner. Most attacks appear to be result of misuse of the system or bad choice of parameters. Analysis of the known attacks shows that RSA has not been proven to be unbreakable, but having survived a great deal of cryptanalytic security over the last twenty years.

REFERENCES

1. Boneh, D., 1999. Twenty years of attacks on the RSA Cryptosystem. Notices of the AMS, 46: 2003-2013.
2. Joye, M., 1997. Security Analysis of RSA-type Cryptosystems. Ph.D. Thesis, University of Catholiqu de Louvain.
3. Killean, R., 2004. RSA: Hacking and Cracking. (n.d.) Retrieved Oct. 2004, from http://www.members.tripord.com\irish_ronan\rsa.html
4. Wiener, M., 1990. Cryptanalysis of short RSA secret exponents. Proc. IEEE, 36: 553-558.
5. Boneh, D. and G. Durfee, 1999. Cryptanalysis of RSA with private key less than $N^{0.292}$. Proc. Eurocrypt'99, 1592: 1-11.
6. Hastad J., 1986. On using RSA with Low Exponent in a Public-Key Network. Advances in Cryptology, 218: 404-408.
8. Menezes, A., P. van Oorschot and S. Vanstone, 1996. Handbook of Applied Cryptography. CRC Press.
7. Coppersmith D., M. Franklin, J. Patarin and M. Reiter, 1996. Low Exponent RSA with Related messages. Advances in Cryptology, 1070: 1-9.
9. Boneh, D., G. Durfee and Y. Frankel, 1998. An attack on RSA given a small fraction of the private key bits. Proc. Asiacrypt'98, 1514: 25-34.
10. Davida G., 1982. Chosen Signature Cryptanalysis of the RSA (MIT) Public-Key Cryptosystem. Technical Report TR-CS-82-2, University of Wisconsin.
11. Lenstra, A., 1996. Memo on RSA signature generation in the presence of faults.
12. Kocher, P., 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. Advances in Cryptology, 1109: 104-113.