BROADCASTING AND LOW EXPONENT RSA ATTACK

Project Report

Submitted to

Dr. Mark Stamp

Department of Computer Science

San Jose State University

March 22, 2012

Presented By:

Govind Kalyankar	**Krishna Nitin Tenali**
Graduate Student (SJSU id: 008138221)	*Graduate Student (SJSU id: 008061872)*
Department of Computer Science	*Department of Computer Science*
San Jose State University	*San Jose State University*

Table of Contents

1.0 Project Overview:

In this project we are dealing with decrypting a cipher which was encrypted using RSA. We are employing the Chinese remainder theorem to decrypt the ciphertext.

- To study RSA algorithm in detail.
- To understand the Chinese remainder theorem.
- To be able to successfully decrypt the ciphertext using the Chinese remainder theorem.
- To be able to use the public exponents and moduli from the given certificates of the three business partners.
- Suggest methods that can be implemented in order to avoid attacks.

2.0 Introduction:

2.1 RSA:

RSA algorithm is a public key cryptosystem [named after the inventors Rivest, Shamir and Adleman], like other public key cryptosystem this is also based on a complex mathematical problem. This algorithm is based on the factoring problem where a public and a private key pair are selected.

2.2 Cube Root Attack:

When a small encryption exponent such as e=3 is used and if $M < N^{1/3}$.
The Ciphertext $C = M^e \bmod N$
Since $M < N^{1/3} \bmod N$ has no effect.
$C = M^e = M^3$
$M = \sqrt[3]{C}$ (the cube root of Ciphertext will give the message).

2.2 Chinese Remainder Theorem:

Chinese remainder theorem is a result about congruencies in number theory. Using the theorem we can determine a number n that when divided by some given divisors leaves given remainders.

Using the theorem can find that the value of number n is 23, that when divided by 3 leaves a remainder of 2, when divided by 5 leaves a remainder of 3, and when divided by 7 leaves a remainder of 2.

Theorem: Let $n_1,n_2,...,n_r$ be positive integers such that $\gcd(n_i,n_j)=1$ for $i \neq j$. Then the system of linear congruence's

$x \equiv c_1 \pmod{n_1}$; $x \equiv c_2 \pmod{n_2}$; ... ; $x \equiv c_r \pmod{n_r}$

has a simultaneous solution which is unique modulo $n_1n_2...n_r$.

3.0 RSA Algorithm:

In Public key cryptography both sender and receiver have a key pair (e, d) where 'e' is the public exponent or encryption exponent known to the others and 'd' is the secret exponent or decryption exponent known merely to the owner. The public exponent is used to encrypt the message sent (or signing the message), and the private exponent is used to decrypt the ciphertext (or verifying the message).

3.1 Key Generation:

- Generate two large random prime p and q.
- Compute ***n = pq*** and ***(phi) φ = (p-1) (q-1).***
- Choose an integer *e*, 1 < e < phi, such that **gcd (e, phi) = 1**.
- Compute the secret exponent *d*, 1 < d < phi, such that ***ed ≡ 1 (mod phi).***
- The public key is (n, e) and the private key (d). We keep the values of d, p, q and phi also secret.

3.2 Encryption:

- Obtains the recipient's **public key (n, e)**.
- Represents the plaintext message as a positive integer *m*, 1 < m < n.
- Computes the ciphertext $\boldsymbol{c = m^e \bmod n}$.

3.3 Decryption:

- Uses his **private key (n, d)** to compute $\boldsymbol{m = c^d \bmod n}$.
- Extracts the plaintext from the message representative *m*.

3.4 Digital Signature:

Sender A does the following:-

1. Creates a *message digest* of the information to be sent.
2. Represents this digest as an integer *m* between 1 and *n*-1.
3. Uses her *private* key (n, d) to compute the signature $s = m^d \bmod n$.
4. Sends this signature *s* to the recipient, B.

3.5 Signature verification:

Recipient B does the following:-

1. Uses sender A's public key (n, e) to compute integer $v = s^e \bmod n$.
2. Extracts the message digest from this integer.
3. Independently computes the message digest of the information that has been signed.
4. If both message digests are identical, the signature is valid.

3.6 Example:

Key generation:
Entity A chooses the primes p = 2357, q = 2551.
Compute n = pq = 6012707 and $\phi = (p-1)(q-1) = 6007800$.
A chooses e = 3674911 and using the Euclidean algorithm,
Finds d = 422191 such that $ed \equiv 1 \pmod{\phi}$.
A's public key is the pair (n = 6012707; e = 3674911),
While A's private key is d = 422191.

Encryption:
To encrypt a message m = 5234673, B uses an algorithm for modular exponentiation to compute
$c = m^e \bmod n = 52346733674911 \bmod 6012707 = 3650502$;
And sends the ciphertext to A.

Decryption:
To decrypt c, A computes
$C^d \bmod n = 3650502422191 \bmod 6012707 = 5234673$

4.0 Attacks on RSA:

- **Relation to factoring** - RSA algorithm is based on the RSA problem or the factoring problem. In order to recover the plaintext from the corresponding ciphertext c, the attacker must find the factors of n [it is part of the public key (n, e)] and using the values should find the value of d. So the attacker must use brute force and try all possible factors of n. Hence it is imperative that the primes p and q are selected in such a way that the factoring in computationally infeasible.
- **Small encryption exponent** - In order to speed up the RSA it is desirable to select a small encryption exponent such as e=3 When encrypting using a low exponent (e.g. e=3), the following ***cube root attack*** is possible. If plaintext M satisfies $M < N^{1/3}$, then the resulting ciphertext is just M^e. The ciphertext can be easily decrypted by taking the nth root of the ciphertext over the integers. If the same plaintext is encrypted and sent to e or more recipients and if the receivers share the same exponent e, but different p and q then the ciphertext can be easily decrypted using the ***Chinese remainder theorem***.
- Since a small encryption exponent value like 3 is used RSA can be easily attacked. So it is recommended to use $e=2^{16+1}$. In this case the ***coppersmith's theorem*** which is based on lattice reduction algorithm. The theorem provides an algorithm for efficiently finding all roots of *f* modulo N that are less than $X = N^{1/d}$.
- **Forward search attack** - If the message space is small or predictable, an attacker can decrypt a ciphertext c by simply encrypting all possible plaintext messages until c is obtained.
- **Small decryption exponent** - Similar to encryption exponent e, it may seem desirable to select a small decryption exponent d in order to improve the efficiency of decryption.1 However, if gcd (p−1; q−1) is small, as is typically the case, and if d has up to approximately one-quarter as many bits as the modulus n, then using square-root discrete logarithm algorithms d can be found.
- **Common modulus attack** - It is sometimes suggested to select a single RSA modulus n, and then distribute a distinct encryption/decryption exponent pair (e_i, d_i) to each entity in a network. However, knowledge of any (e_i, d_i) pair allows for the factorization of the modulus n, and hence any entity could subsequently determine the decryption exponents of all other entities in the network.

- Since RSA does not have a random component a ***chosen plaintext attack*** can be employed on RSA by encrypting likely plaintext using the public key and test if they are equal to the ciphertext.
- **Cycling attacks** - Let c = me mod n be a ciphertext. Let k be a positive integer such that $(c^e)^k \equiv c \pmod n$ since encryption is a permutation on the message space {0, 1,.., n − 1} such an integer k must exist. For the same reason it must be the case that $(c^e)^{k-1} m \pmod n$. This observation leads to the following ***cycling attack*** on RSA encryption. An adversary computes $c^e \bmod n$, $(c^e)^2 \bmod n$, $(c^e)^3 \bmod n$;...until c is obtained for the first time. If $c^{ek} \bmod n = c$, then the previous number in the cycle, namely $c^{ek-1} \bmod n$, is equal to the plaintext m.
- **Multiplicative properties** - The product of two ciphertext is equal to the encryption of the product of the respective plaintexts this property of RSA can be used to employ a ***chosen ciphertext attack***. The attacker may ask the holder of the private key to decrypt an unsuspicious-looking ciphertext $c'=cr^e \pmod n$ for some value r chosen by the attacker. Because of the multiplicative property c' is the encryption of mr (mod n). Hence, if the attacker is successful with the attack, he will find mr (mod n) from which he can derive the message m by multiplying mr with the modular inverse r of modulo n.

5.0 The Chinese Remainder Theorem:

We can determine a number n that when divided by some given divisors leave given remainders.

- Let $n_1, n_2,..., n_r$ be positive integers such that $\gcd(n_i, n_j)=1$ for $i \neq j$.
- Linear congruence's
  - $x \equiv c_1 \pmod{n_1}$; $x \equiv c_2 \pmod{n_2}$; ... ; $x \equiv c_r \pmod{n_r}$ has a simultaneous solution which is unique modulo $n_1n_2...n_r$.
- If c_1,c_2,c_3 are 1,2,3 and n_1,n_2,n_3 are 3,4,5.
- Chinese Remainder Theorem (CRT) tells us that since 3, 4 and 5 are coprime in pairs then there is a unique solution modulo 3 x 4 x 5 = 60.
- Using **Gauss's algorithm**, Let $N=n_1n_2...n_r$ then $x \equiv c_1N_1d_1 + c_2N_2d_2 + ... + c_rN_rd_r \pmod N$ where $N_i = N/n_i$ and $d_i \equiv N_i^{-1} \pmod{n_i}$.
- The latter modular inverse d_i is easily calculated by the extended Euclidean algorithm.

Example:
$n_1=3$, $n_2=4$, $n_3=5$
$N = n_1n_2n_3 = 3 \times 4 \times 5 = 60$
$c_1=1$, $c_2=2$, $c_3=3$.
$N_1 = N/n_1 = 20$; $d_1 = 20^{-1} \pmod 3 = 2$ [check: $2x20=40\equiv1 \pmod 3$]
$N_2 = N/n_2 = 15$; $d_2 = 15^{-1} \pmod 4 = 3$ [check: $3x15=45\equiv1 \pmod 4$]
$N_3 = N/n_3 = 12$; $d_3 = 12^{-1} \pmod 5 = 3$ [check: $3x12=36\equiv1 \pmod 5$]
$x \equiv c_1N_1d_1 + c_2N_2d_2 + c_3N_3d_3 \pmod N$
$x = (1x20x2) + (2x15x3) + (3x12x3) = 238 \equiv 58 \pmod{60}$

So a solution is $x = 58$. Any integer that satisfies $58 + 60k$ for any integer k is also a solution, but this algorithm gives a unique solution in the range $0 \leq x < n_1n_2n_3$.

6.0 RSA attack on a low exponent using Chinese remainder theorem:

6.1 Implementation in C:

In C, the length of the long type is 64-bit and that of the integer type is 32-bit. Now the length of the parameters of RSA algorithm is 512-bit. We adopt the Bigdigit class in built-in library to implement the Chinese remainder theorem and retrieve the plain text. We have used CRT_RSA.c to implement the following pseudocode.

Pseudo code:

1. Given Parameters n_1, n_2, n_3, e, c_1, c_2, c_3 where public pair = (n, e), c is ciphertext.
2. Convert given Hexadecimal modulus values of n_1, n_2, n_3 and c_1, c_2, c_3 each of 512 bits in Decimal using function bdConvFromHex ().
3. An eavesdropper has the public values n_1, n_2, n_3, c_1, c_2 and c_3.
4. Check that n_1, n_2, n_3 are coprime in pairs by computing GCD of each Pair using function bdGcd ().
5. Compute $N = n_1 * n_2 * n_3$ using function bdMultiply ().
6. Compute $N_i = N/n_i$ for i = 1, 2, 3.
7. Compute $d_i = N_i^{-1} \bmod n_i$ for i = 1, 2, 3 using function bdModInv ().

8. Compute $x = c_1 N_1 d_1 + c_2 N_2 d_2 + c_3 N_3 d_3 \pmod N$ using function bdModMult ().
9. Compute the integer cube root of x using function bdCubeRoot ().
10. Converting Obtained Bigdigit Hex value into String or plaintext.

6.2 Computation Results:

The following are the given cipher texts

C_1:
34d2fc2fa4785e1cdb1c09c9a5db98317d702aaedd2759d96e8938f740bf982e2a42b904e54dce016575142f1b0ed112cc214fa8378b0d5eebc036dc7df3eeea
C_2:
3ddd68eeff8be9fee7d667c3c0ef21ec0d56cefab0fa10199c933cffbf0924d486296c604a447f48b9f30905ee49dd7ceef8fc689a1c4c263c1b3a9505091b00
C_3:
956f7cbf2c9da7563365827aba8c66dc83c9fb77cf7ed0ca225e7d155d2f573d6bd18e1c18044cb14c59b52d3d1f6c38d8941a1d58942ed7f13a52caccc48154

From Given Certificates:

n_1=
9623511e6769644d693e89f692ffc2558eef121d42ca98699781e139e29c2e1aa58d8883bbdba41165fdeb85a9a5648fc29a65d59e9401694dd11ae205f0ce3b
n_2=
ad4bc0f980f4523f490fc40c12efcecc1e8af67890b6562449876e8e091e861cda699e5a8eb309b0a9d6b293100c1229fbd18a5951f33b6fbab1fd8d90f7c829
n_3=
b7223364d88353ec02b0850e8a01d2ba9ca2663c32c15df7b596406c6fc1c171ac965a554b8b338f4bb046c543937b4b19c699864f1d0dd4be0177eccce0bb57
e=3

$N=n_1*n_2*n_3$

N=48b497d0a83ed41f3516cbf4d62720c8c7322f361ddd9363efd298669d226043f0e80c30a8e54ddf63f3d1922821e7d06df80437aca0dfdee9e7edf4f0019010e4116619cb43552d4d34a529af1766cd829e4713a8c97bb307883e21d3b31bc7e480d66

9dec5b9479d85d772a9c28bfc20b08ca45ec24362620ad5b36931e94c41f9ccf5034
b85cd94ec89d6381522a495b08f4321a5f118ffc31502fc630eabb49d6fa206a9d68e
74b02cbabb9e3b772b0ca8d88032205466f9f95dd1a7b115

$N_1 = N/n_1$; $N_2 = N/n_2$; $N_3 = N/n_3$;

N_1=

7bf84dcc57ebaecacd28b869bea5c4e744196cd07e6fb880943e29c3e78bd1223e4f
175c73f04548d1d4fe19ddc38b98c56a174e33816fd061ba9b358e52c994669ac10f
6ebf70e32fe020dc1477d7a5b6e3317580d4583a2260438bd215ec8e5c003f66d89
448f0fabf48daa6cd05aa7babadf76eb5dd822ab8751d134af8ef

N_2=

6b674dd1afe5a2e913e991332e8ecf7761986489d311a68de61304044803678d04
15b409fb04bfd45cfff46cbb1914a6f94554464c23e9cac59cf58b9ef442d74a712aae
36c3046f806631667074093b597e57ced733e124e1eff01365e1198d9b5a646e8ba
e9085bbfea33c7fbe0a328d4f085caa9c7f01d54adf96631b2f0d

N_3=

65a24b57209c2b3f45befe2b607c6705c18a3698f534d915328482943977b682bd1
159259accaf4efc15644e1180689331bca458fd685dfa556b7534d7b386df50194c1
18fa8ad62f9feb55b6bbee84857bdd223372c27be03adaa4d25af837bac835b659e
77904e488192bdc244a2a85c331d6c717476cf9d25f7ae3f9c1f73

d_1 is the modular multiplicative inverse of N_1 mod n_1, similarly d_2 and d_3 can be calculated.

d_1=8763f05a12b0a439e0ed70db83499817f83e88e3a6ce0ef5fc67bd12047e49217
ba18153e7c3886e47b9e7fc810c297569cedcebba73b63e1a46827633768554
d_2=261679ebd4461f0635889badb88aea53178cb77f607248d3c17e98c00f5a35659
485c7a1e3725c37a1d77d828b3923c4c62f308093ae91805d8f9451331c84a0
d_3=a0df4beeec3dce6f7a8747260755622f4082a13dcae24a019cf6131e898737577
9c20994b1cc38967a3e1303e7e3e3847ce30e4e668fc0df835cd9e97dbf5b25

$X \equiv c_1N_1d_1 + c_2N_2d_2 + c_3N_3d_3 \pmod N$

X ≡
4e0fe2c05bcb97d7696437b4fbaa0cca2de2968f2d9bfefe772852fa5591db0665f3e
7b6059535363646f1dcc8a715467d3cfd2cc9153a7efd24b77b806a333cbfd708142
38c06ad5a7bad600cbd1b800dd57fb22f1e0fbb86e0eacc6a224d88e29286f2c2445
6d1ea8354ea0ee0cd81153a83373f945c19b9e42893263b54a2abdd17a87a49d98
9fd9add49d43a8a07b37e396f3f0d0e358da97b0e5287efcbaae2ea92e6cbd961b0
856db17d0c1d03558fffa32b9cd01f49db75661

Converting the above big digit value to string, we get Plaintext as
Das fuer morgen festgelegte Meeting muss unbedingt stattfinden!

7.0 Attempting Brute force attack by factoring N:

RSA algorithm is based on the factoring problem. Since p and q are two prime numbers such n=pq. We used the Python program along with sagemath to find two prime factors of n. Since we have p and q, d can be calculated using ed = 1 mod (p-1) (q-1). As per our analysis we understand that d is greater than 50000.

Given Hexadecimal value of n_1=
9623511e6769644d693e89f692ffc2558eef121d42ca98699781e139e29c2e1aa58d
8883bbdba41165fdeb85a9a5648fc29a65d59e9401694dd11ae205f0ce3b
Decimal value of
n_1=786336282839694542267164165109290086878741830441658273480655459
846602888310796983973207571010092091107396804826519715254540481749
8266214859796653183913531

Python script:

```
def get_primes(n):
primes=[False,False]+[True]*(n-1)
next_p=(i for i,j in enumerate(primes) if j)
while True:
p=next(next_p)
yield p
primes[p*p::P]=[False]*((n-p*p)//p+1)
```

p=342632330648354211252647766081634405379257059979623465969778034
62033841059628723

q=377318081621938460678418953889955311049944229578257670222228038
4917551

Brute force attack on Low Exponent Attack on RSA for a smaller n:

n=1966981193543797

e=323815174542919

$$\frac{1}{6} \quad 1942891047257513$$

$$\frac{27}{164} \qquad \frac{53105688625038715}{27}$$

$$\frac{121}{735} \qquad \frac{238004153289045464}{121}$$

$$\frac{578}{3511} \qquad 1966981103495136$$

$$\frac{6237}{37886} \qquad \frac{12268061702733029233}{6237}$$

$$\frac{1157192}{7029241} \qquad \frac{1138087450659621247239}{1157192}$$

$$\frac{3701767}{22485994} \qquad \frac{428312121875354669205}{217751}$$

$$\frac{28456944}{172858711} \qquad \frac{1166130701536020677446}{592853}$$

$$\frac{16257705941}{98755723481} \qquad \frac{319786018361122593305810 38}{16257705941}$$

$$\frac{191318803041}{1162145931191} \qquad \frac{376320487552956799050286528}{191318803041}$$

So we get two candidates for phi(n): 1942891047257513 and 1966981103495136. But phi(n) is even, so we
try the latter.
solve(x^2-(n-1966981103495136+1)*x+n)
if { [x = 37264873] ,[x = 52783789] }

Sure enough, this is the factorization of n:
ifactor(n)
Thus we got p and q after a long manual factorization
p=37264873, q= 52783789

8.0 How to prevent this type of attack:

1. Use a larger exponent, like 65537 (0x10001). This makes it harder to use the above method.
2. Padding is important to prevent attacks, since it would be more difficult for the attacker. But it is also important that the padding should not be larger than the necessary length. Uniformly spreading this system across the message gives the same inefficiency. Consider a message that is 500 characters long. The 6 bit representation contains 3000 bits. The 8 bit version contains an extra 1000. This expands the length of the message by a full third, far greater than we needed. Therefore the padding should be spread out, one character every five, Therefore if the encryption exponent is 3 the padding should be applied to all cases, not just ones where the

message is small. Obviously, the recipient needs to know how to remove the random bytes after decrypting the message.
3. Don't use the same RSA key for encryption and signing.
4. If using PKCS#v1.5 encoding, use e=0x10001 for your public exponent.
5. Always format your input before encrypting or signing.

9.0 Conclusion:

In this project, we have studied the RSA algorithm and Chinese reminder theorem in detail. We have been able to successfully perform an attack on RSA which uses a low exponent using Chinese remainder theorem and retrieve the original plain text message.

10.0 References:

[1] **"*Applied Cryptanalysis: Breaking Ciphers in the Real World*"**, Mark Stamp and Richard M. Low, Wiley-IEEE Press, May 2007.

[2] "***Research and Implementation of RSA Algorithm in Java*",** Jiezhao Peng and Qi Wu, International Conference on Management of e-Commerce and e-Government.

[3] "***The Handbook of Applied Cryptography*"**, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.

[4] "***An Efficient RSA Public Key Encryption Scheme*",** Sattar J Aboud1, Mohammad A AL-Fayoumi1, 2Mustafa Al-Fayoumi and 3Haidar S Jabbar , Fifth International Conference on Information Technology: New Generations.

[5] "***RSA Problem***", Ronald L. Rivest, and Burt Kaliski, RSA Laboratories,December 10, 2003.

[6] http://www.di-mgt.com.au/crt.html