

A Tutorial paper on Hastad Broadcast Attack

Abhay Chennagiri
Department of Computer Science
University of California, Santa Barbara
Email: abhay@cs.ucsb.edu

Abstract—The RSA cryptosystem has been around for around half a century now. Since its invention, a number of attacks have been tried to defeat the system but it's correct and efficient implementations have withstood these exploits quite successfully. The RSA cryptosystem leverages the hardness of factorizing prime numbers and the only legitimate threat that is looming over it's breakability is the rise of fully fledged quantum computers. However, a number of researchers believe that quantum computing is in a very nascent stage and that RSA is going to remain the de facto standard in the coming several decades. I personally believe there is still a lot of life left in RSA and it's a great encryption technique to learn about. RSA can be susceptible to a number of attacks if the implementations do not meet the standards. One such attack is the Hastad Broadcast Attack. This short tutorial paper gives a brief overview of this attack using a CTF problem presented in Plaid CTF 2017.

Keywords—Hastad Broadcast, RSA, CRT, Coppersmith

I. INTRODUCTION

Before we understand the Hastad Broadcast Attack, let's understand how and why the RSA cryptosystem works.

A. History of RSA

The August 1977 edition of the Scientific American, a very popular science magazine described a new encryption which would go on to take the internet by storm later. A group of three computer scientists; Ronald Rivest, Adi Shamir and Len Adleman from MIT came up with this remarkable encryption algorithm. It was a response to the open problem posed by Diffie and Helman at that time. Diffie and Helman hypothesized the concept of an asymmetric public-private key cryptosystem. They left open the problem of realizing a one-way function which was exactly what the trio from MIT invented (The existence of complete one way functions not proven yet!).

In order to test their hypothesis, they generated a large number having 129 digits. This number is popularly known as RSA - 129. This number was basically a product of two large prime numbers. They challenged the scientific community to factorize RSA-129 and put up a prize money of 100\$ for the person who solved it. Martin Gardner, a very popular American science writer wrote in his column that it would take approximately 40 quadrillion years to find the prime factors of RSA-129. Interestingly, it took only 17 years to solve this thanks to better algorithms and the advances in computing. A need for bigger prime numbers was realized. In fact, there exists something called as RSA factoring Challenge put forth by RSA laboratories to encourage research into computational number theory and the practical difficulty of factoring large integers.

B. Hard Problem - Prime Factorization

It was mentioned in the previous section that a one way function was invented. In computer science, a one way function is a function that is easy to compute on every input, but hard to invert given the image of a random input. In the case of RSA, it is easy to use the public key to encrypt but it is very hard to decrypt it without knowing the private key. In our context, the term hard implies that it belongs to the class of NP i.e. it doesn't have a polynomial time solution. As the wikipedia says, the terms "easy" and "hard" are usually interpreted relative to some specific computing entity; typically cheap enough for the legitimate users and expensive for any malicious users. The existence of a completely one way function is still an open conjecture. Solving this would mean that the classes P and NP are not equal which is one of the millenial problems.

The difficulty of factorizing large numbers is sometimes attributed to it's randomness. But it seems very less likely that even if conjectures like Goldback and Twin Primes are solved, the problem of prime factorization would become simple. It was recently discovered that primality testing is an easy problem i.e there exists a polynomial time solution to this problem but this has in no way affected the time complexity of factorizing large numbers. However, there is an interesting implication to this problem by Shor's algorithm, using which we can factorize large numbers in polynomial time using Quantum computers. A number of people in the cryptographic community believe that it may take a while before fully blown quantum computers come into existence at which point the hardness of RSA will cease to exist.

C. RSA algorithm

In this section, let's understand how the RSA algorithm works and also why it works.

- Choose two different large random prime numbers p and q .
- Calculate $n = p * q$. n is the modulus for the public key and the private keys.
- Calculate the totient: $\phi(n) = (p - 1) (q - 1)$
- Choose an integer e such that $1 < e < \phi(n)$, and e is coprime to $\phi(n)$ i.e.: e and $\phi(n)$ share no factors other than 1; $\gcd(e, \phi(n)) = 1$. e is released as the public key exponent.
- Compute d to satisfy the congruence relation $d * e \equiv 1 \pmod{\phi(n)}$ i.e: $d * e = 1 + k\phi(n)$ for some integer k . d is kept as the private key exponent

D. Correctness of RSA

As mentioned in the previous section, the public key consists of N and e where e is the encryption key. Once it is published, anyone can use it to encrypt messages. The following is the encryption function: $f(M) \equiv M^e \pmod{N}$ where, M is the original message and is also a positive integer. The private key is a positive integer d that satisfies: $d \cdot e \equiv 1 \pmod{\phi(N)} = (p-1) \cdot (q-1)$

In other words, d is the multiplicative inverse of e in the modular arithmetic of modulo ϕ . The above condition is equivalent to: $de - 1 = (p-1) \cdot (q-1) \cdot k$ for some integer k . The number d is the decryption key that will be used to decode messages. So it should remain secret.

Once the intended receiver receives an encrypted message C , he or she uses the following decryption function to obtain the original message M . $g(C) \equiv C^d \pmod{N}$. What we prove is that the decryption function is to undo the encryption function.

In other words, if we raise the ciphertext C to the power d modulo N , we get back the original message. In the following subsections, we list out the tools needed to prove the correctness of RSA.

1) *Fermat's Little Theorem*: Pierre De Fermat was a famous mathematician who is probably very well known for his "Last Theorem". His little theorem is essential to the working of RSA and below is what it says. If p is a prime number and a is an integer such that a and p are relatively prime, then

$$a^{p-1} - 1 \text{ is an integer multiple of } p$$

or equivalently $a^{p-1} \equiv 1 \pmod{p}$.

2) *Euclid's Lemma*: Let a , b and d be integers where $d \neq 0$. Then if d divides $a \cdot b$ (symbolically $d \mid a \cdot b$), then either $d \mid a$ or $d \mid b$. Euclid's Lemma is needed to prove the following Lemma.

Lemma 2: Let M be an integer. Let p and q be prime numbers with $p \neq q$. Then if $a \equiv M \pmod{p}$ and $a \equiv M \pmod{q}$, then $a \equiv M \pmod{p \cdot q}$.

Proof: Suppose we have $a \equiv M \pmod{p}$ and $a \equiv M \pmod{q}$. Then for some integers i and j , we have: $a = M + p \cdot i$ and $a = M + q \cdot j$. Then $p \cdot i = q \cdot j$. This implies that p divides $q \cdot j$ ($p \mid q \cdot j$). By Euclid's lemma, we have either $p \mid q$ or $p \mid j$. Since p and q are distinct prime numbers, we cannot have $p \mid q$. So we have $p \mid j$ and that $j = p \cdot w$ for some integer w .

Now, $a = M + q \cdot j = M + q \cdot p \cdot w$, implying that $a \equiv M \pmod{p \cdot q}$. We now need to show that

$(M^e)^d = M^{ed} \equiv M \pmod{N = p \cdot q}$ We first show that $M^{ed} \equiv M \pmod{p}$ and $M^{ed} \equiv M \pmod{q}$. Then the desired result follows from Lemma 2. To show $M^{ed} \equiv M \pmod{p}$, we consider two cases: $M \equiv 0 \pmod{p}$ or $M \not\equiv 0 \pmod{p}$.

Case 1. $M \equiv 0 \pmod{p}$. Then M is an integer multiple of p , say $M = p \cdot w$ where w is an integer. Then $M^{ed} = (p \cdot w)^{ed} = p \cdot p^{ed-1} \cdot w^{ed}$. So both M and M^{ed} are integer multiples of p . Thus $M^{ed} \equiv M \pmod{p}$

Case 2. $M \not\equiv 0 \pmod{p}$. This means that p and M are relatively prime (having no common divisor other than 1). Thus

we can use Fermat's Little Theorem. We have $M^{p-1} \equiv 1 \pmod{p}$. From the way the decryption key d is defined above, we have $ed - 1 = (p-1) \cdot (q-1) \cdot k$ for some integer k . We then have: $M^{ed} = M^{ed-1} \cdot M = M^{(p-1) \cdot (q-1) \cdot k} \cdot M = (M^{p-1})^{(q-1) \cdot k} \cdot M \equiv (1)^{(q-1) \cdot k} \cdot M \pmod{p} \equiv M \pmod{p}$

The same reasoning can show that $M^{ed} \equiv M \pmod{q}$. By Lemma 2, it follows that $M^{ed} \equiv M \pmod{N} = p \cdot q$.

E. Attacks on RSA

Since its inception, the RSA system has been scrutinized heavily for exploits by a number of researchers. Although, a lot of them have been found, they work only if the RSA is poorly implemented in some way. However, a secure implementation of RSA is also not very straightforward.

In this report, we will take a look at the one of categories of attacks on RSA resulting due to use of low public key exponent. To reduce encryption time or signature verification, it is quite usual to use low values of e . The smallest possible value for e is 3, but to defend against certain attacks the value e needs to be $2^{16} + 1 = 65537$. When $e = 65537$, the signature verification takes 17 multiplications as opposed to roughly 25 when a random e is used. The most powerful attacks on low public exponent RSA are based on a theorem due to Coppersmith.

1) *Coppersmith Theorem*: Let N be an integer and $f \in \mathbb{Z}[x]$ be a monic polynomial of degree d over the integers. Set $X = N^{1/d-\epsilon}$ for $1/d > \epsilon > 0$. Then, given (N, f) the attacker can efficiently (in polynomial time) find all integers $x_0 < X$ satisfying $f(x_0) \equiv 0 \pmod{N}$.

This theorem basically states that there is an algorithm which can efficiently find all roots of f modulo N that are smaller than $X = N^{1/d}$. As X gets smaller, its runtime will decrease. This theorem's main idea is the ability to find all small roots of polynomials modulo a composite N . We use this theorem later in the example to find our secret message.

2) *Chinese Remainder Theorem*: CRT is one of the many powerful tools that comes to the rescue in a number of places. It is needed to understand Hastad Broadcast attack as well. If m_1, m_2, \dots, m_k are pairwise relatively prime positive integers, and if a_1, a_2, \dots, a_k are any integers, then the simultaneous congruences $x \equiv a_1 \pmod{m_1}$, $x \equiv a_2 \pmod{m_2}$, ..., $x \equiv a_k \pmod{m_k}$ have a solution, and the solution is unique modulo m , where $m = m_1 m_2 \dots m_k$

3) *Hastad Broadcast Attack*: We shall first discuss the simplest form of Hastad's attack to get a better idea. The general case uses the Coppersmith method.

Suppose one sender sends the same message M in encrypted form to a number of people $P_1; P_2; \dots; P_k$ each using the same small public exponent e , say $e=3$, and different moduli (N_i, e) . A simple argument shows that as soon as $k \geq 3$ ciphertexts are known, the message M is no longer secure: Suppose a person intercepts C_1, C_2 and C_3 , where $C_i \equiv M^3 \pmod{N_i}$. By the Chinese Remainder Theorem, he/she may compute $C \in \mathbb{Z}_{N_1 N_2 N_3}^*$ such that $C_i \equiv C \pmod{N_i}$. Then $C \equiv M^3 \pmod{N_1 N_2 N_3}$; however, since $M < N_i$ for all i 's, we have $M^3 < N_1 N_2 N_3$. Thus $C = M^3$ holds over the integers, and he/she can compute the cube root of C to obtain M .

Generalizations: The problem posed in the next section is based on the generalization of Hastad Broadcast attack. Hastad also showed that applying a linear-padding to M prior to encryption does not protect against this attack. He proved that a system of univariate equations modulo relatively prime composites, such as applying any fixed polynomial $g_i(M) \equiv 0 \pmod{N_i}$ could be solved if sufficiently many equations are provided. This attack suggests that randomized padding should be used in RSA encryption.

Theorem 2 (Hastad): Suppose $N_1 \dots N_k$ are relatively prime integers and set $N_{min} = \min_i N_i$. Let $g_i(x) \in \mathbb{Z}/N_i$ be k polynomials of maximum degree q . Suppose there exists a unique $M < N_{min}$ satisfying $g_i(M) \equiv 0 \pmod{N_i}$ for all $i \in 1, \dots, k$. Furthermore, suppose $k > q$. There is an efficient algorithm which, given $(N_i, g_i(x))$ for all i , computes M .

Proof: Since the N_i are relatively prime, the Chinese Remainder Theorem might be used to compute coefficients T_i satisfying $T_i \equiv 1 \pmod{N_i}$ and $T_i \equiv 0 \pmod{N_j}$ for all $i \neq j$. Setting $g(x) = \sum_i T_i \cdot g_i(x)$ we know that $g(M) \equiv 0 \pmod{\prod N_i}$. Since the T_i are nonzero we have that $g(x)$ is also nonzero. The degree of $g(x)$ is at most q . By Coppersmiths Theorem, we may compute all integer roots x_0 satisfying $g(x_0) \equiv 0 \pmod{\prod N_i}$ and $|x_0| < \prod N_i^{\frac{1}{q}}$. However, we know that $M < N_{min} < (\prod N_i)^{\frac{1}{k}} < (\prod N_i)^{\frac{1}{q}}$, so M is among the roots found by Coppersmith's theorem.

F. Practical Example

Here is the challenge which was posed in this year's PlaidCTF.

```
nbits = 1024
e = 5
flag = open("flag.txt").read().strip()
assert len(flag) <= 64
m = Integer(int(flag.encode('hex'), 16))
out = open("data.txt", "w")

for i in range(e):
    while True:
        p = random_prime(2^floor(nbits/2)-1,
            lbound=2^floor(nbits/2)-1,
            proof=False)
        q = random_prime(2^floor(nbits/2)-1,
            lbound=2^floor(nbits/2)-1,
            proof=False)
        ni = p*q
        phi = (p-1)*(q-1)
        if gcd(phi, e) == 1:
            break

    while True:
        ai = randint(1, ni-1)
        if gcd(ai, ni) == 1:
            break

    bi = randint(1, ni-1)
    mi = ai*m + bi
    ci = pow(mi, e, ni)
    out.write(str(ai)+'\n')
    out.write(str(bi)+'\n')
```

```
out.write(str(ci)+'\n')
out.write(str(ni)+'\n')
```

In the above code, we can see that public key exponent $e=5$, prime numbers p and q used to generate N are 1024 bits long. Also, we can observe that we get 5 encrypted messages of the same message M (which is read from the file `flag.txt` and hex encoded) but using different N values. And we can also see that encrypted messages(c_i) that is being printed onto `data.txt` is not a direct encryption but a linear padding is applied to M and then is raised to the power of e .

Here in this challenge, we are given a sage script called `generate.sage` which contains the above code. Sage is a free open-source mathematics software system. It combines the power of many existing open-source packages into a common Python-based interface. The file `flag.txt` is inaccessible. The four numbers a_i , b_i , c_i and n_i get written onto the file `data.txt` 5 times. It seems like all the necessary conditions required for Hastad Broadcast attack are there.

$c_i = m^5 \pmod{n_i}$ for $i \in 1, 2, \dots, 5$ Using CRT, we can compute c^* such that $c^* = c_i \pmod{n_i}$. CRT guarantees that $c^* < \prod n_i$ Since $m < \min(n_i)$ $m^5 < \prod n_i$ which means that $c^* = m^5$. So, can M be computed as the 5th root of C^* ? No, because the C that we have undergoes linear padding operation as mentioned before $m_i = a_i \cdot m + b_i$. Here a_i and b_i are randomly chosen integers. $c_i = (a_i \cdot m + b_i)^5 \pmod{n_i}$

The linear padding makes the standard broadcast attack infeasible, but we can make use of proof in Theorem 2 described in the previous section to our advantage.

We can use CRT to compute T_i such that $T_i = 1 \pmod{N_i}$ and $T_i = 0 \pmod{N_{i \neq j}}$ for all j . Then, we can construct the polynomial $g(x) = \sum T_i \cdot ((a_i \cdot m + b_i)^5 - c_i)$ We can see that $g(m) \pmod{n_i} = 0$ for all i because either $T_j = 0 \pmod{n_i}$ for $i \neq j$, or $(a_i \cdot m + b_i)^5 = c_i \pmod{n_i}$. Thus m is a root of $g(x)$ modulo M where $M = \prod n_i$ and since $m < n_i$ we must have $m^5 < \prod n_i$ as discussed above. So, $m < (\prod n_i)^{1/5} = M^{1/\deg(g)}$.

And hence we can find m using the Coppersmith's method. Coppersmith has an additional requirement that the polynomial need to be monic so we need to multiply the highest powered coefficient by it's inverse without affecting the roots.

Here is the solution code.

```
import binascii

data = open('data.txt', 'r')
d = data.read().split()
d = [Integer(a) for a in d]

a_i = []
b_i = []
n_i = []
c_i = []

for i in range(5):
    a_i.append(d[4*i])
    b_i.append(d[4*i + 1])
    c_i.append(d[4*i + 2])
    n_i.append(d[4*i + 3])
```

REFERENCES

- [1] Boneh, Dan. "Twenty years of attacks on the RSA cryptosystem." Notices of the AMS 46.2 (1999): 203-213.
- [2] Durfee, Glenn. Cryptanalysis of RSA using algebraic and lattice methods. Diss. stanford university, 2002.
- [3] Wikipedia contributors. "Coppersmith's attack." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 14 Jun. 2017. Web. 17 Jun. 2017
- [4] Ma, Dan. N.p., 2013. Web. 10 June 2017. <https://exploringnumbertheory.wordpress.com/2013/07/08/fermats-little-theorem-and-rsa-algorithm/>

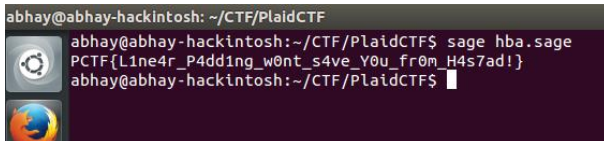
```
mat = [[1,0,0,0,0],[0,1,0,0,0],[0,0,1,0,0],
       [0,0,0,1,0],[0,0,0,0,1]]
t_i = []
for i in range(5):
    ti = crt( mat[i], n_i )
    t_i.append( ti )

G.<x> = PolynomialRing( Zmod( prod( n_i ) ) )
g_i = []
for i in range(5):
    g_i.append( ( t_i[i] * (( a_i[i] * x
                          + b_i[i] ) ** 5 - c_i[i] ) ) )

g = sum( g_i )
g = g.monic()
roots = g.small_roots()

l = len( hex( int( roots[0] ) ) )
print binascii.unhexlify( hex
                          ( int( roots[0] ) [ 2 : l - 1 ] ) )
```

This code runs for 2 seconds and prints the message as shown in the figure below



```
abhay@abhay-hackintosh: ~/CTF/PlaidCTF
abhay@abhay-hackintosh:~/CTF/PlaidCTF$ sage hba.sage
PCTF{Line4r_P4dd1ng_w0nt_s4ve_Y0u_fr0m_H4s7ad!}
abhay@abhay-hackintosh:~/CTF/PlaidCTF$
```

II. CONCLUSION

I studied a lot about number theory and prime numbers while doing this project. I discovered that there were a lot of open problems like Goldbach conjecture, Twin Prime conjecture in this domain which have remain unsolved for centuries and this fascinated me. RSA algorithm is one of the most widely used encryption algorithm on the internet and it's very essential to learn the nuances of how and why it works. For example, if a banking site is still using 512 bits prime numbers, you might be a little skeptical about making online transactions. We have also proved the correctness of RSA in this report. We can see that the underlying hard problem of factorizing large integers is still computationally very hard and it's going to remain hard until quantum computers decide to take over the computing world. We have learnt through an example that it can be dangerous to use low public key exponents. We also see that linear padding won't be of much use. Instead it would be better to use randomized padding. It might not be a very fruitful experience from a research point of view since all the low hanging fruits in this topic are already discovered, it is certainly a great topic to learn about before advancing to further complicated areas.

ACKNOWLEDGMENT

I would like to thank Prof. Koc for allowing me to work on this project and being so co operative in a lot of aspects.