Return of Coppersmith's Attack

Ref: The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli

Suraj 170050044 **Aditya** 170050043 **Aman** 170050027 **Yateesh** 170050005

Introduction

This paper is based on exploiting algorithmic flaw in the construction of primes for RSA key generation. The primes generated by the library suffer from a significant loss of entropy. The method requires no additional information except for the value of the public modulus and does not depend on a weak or a faulty random number generator.
Reasons to construct a candidate number (N) from several smaller (randomly) generated components instead of generating it randomly.

1. To be resistant against Pollard's p-1 method
2. For all primes p, the values of p1 and p+1 have at least one large (101-bit or larger) factor each
3. To speed-up the process of keypair generation, since testing random candidate values for primality is time consuming

RSA Cryptography

1. Select two distinct large primes p and q.
2. Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$.
3. Choose a public exponent $e < \phi(N)$, e coprime to $\phi(N)$
   —>e with low hamming weight for faster encryption.
4. Compute the private exponent d as $e - 1 \bmod \phi(N)$.
   Public key-(e,N) Private key-(p,q) or (d,N)
5. Attacks on RSA relies on factorising N and finding the primes p and q.

Structure of primes

Most likely introduced to speed up prime generation. Motivation :-
The keys exhibited a non-uniform distribution of ($p \ mod \ x$) and ($N \ mod \ x$) for small primes x. All RSA primes (as well as the moduli) generated by the RSALib have the following form:-

$$p = k * M + (65537^a \ mod \ M) \qquad ---(1)$$

$$k, a - unknown \quad M - known$$

M = Product of first n successive primes ($P_n\#$) – n related to key size.

The value n = 39 (i.e., M = 2 * 3 * · · · * 167) is used to generate primes for an RSA key with a key size within the [512, 960] interval. The values n = 71, 126, 225 are used for key sizes within intervals [992, 1952], [1984, 3936], [3968, 4096].

Important property
Size of M is very large. So, size of k and a are small. For 512-bit key, M is of 219 bits. Size of k = 256-219 = 37 (because k*M is 256-bits) and a is of 62 bits.
Significant loss in entropy (only 99 bits of entropy). Reduced from 2256 to 299.

Fingerprinting : (Detection method)

$$N = p * q$$

$$N = (k * M + 65537^a modM)(l * M + 65537^b modM)$$

$$\implies N \equiv 65537^{a+b} \equiv 65537^c \ mod \ M$$

The public modulus N is generated by 65537 in the multiplicative group Z_{M^*} . The existence of the discrete logarithm $c = (log_{65537}N) \ mod \ M$ is used as the fingerprint of the public modulus N generated by the RSALib and the value of c, if exists, can be calculated using *Pohlig-Hellman algorithm.*

Pohlig Hellman Algorithm

Time Complexity:- $O\left(\sum_i e_i(logn + \sqrt{p_i})\right)$

The algorithm can be used to efficiently compute a discrete logarithm for a group G, whose size |G| is a smooth number (having only small factors). The group G = [65537] (subgroup of ZM* generated by 65537). The size of G is a smooth number (e.g.,

$$\|G\| = 2^4 * 3^4 * 5^2 * 7 * 11 * 13 * 17 * 23 * 29 * 37 * 41 * 53 * 83$$

for 512-bit RSA) regardless of the key size. The smoothness of G is a direct consequence of the smoothness of M. Since M is smooth (M is a primorial, $M = 2 * 3 * 5 * .... * P_n$), the size of Z M is even "smoother" ($\|ZM\| = \phi(M)$). The size $\|G\|$ is a divisor of $\|ZM\|$ (from Lagrange's theorem), and it is therefore smooth as well.

- Very fast algorithm to verify whether a key originates from the inspected library based on the properties of the public modulus(N).
- Negligible false negative and false positive rates (observed as zero within a million keys tested), as guaranteed by the very rare properties.

False positives
The primes generated by this library are from a group with size $|\mathrm{G}| = 2^{62,09}$ while ideally, it should be $|Z_M * | = \phi(\mathrm{M}) = 215.98$ for 512-bit RSA. The probability that a random 512-bit modulus N is an element of G is $2^{62-216} = 2^{-154}$. So, an RSA key was generated by the RSALib if and only if the Pohlig-Hellman algorithm can find the discrete logarithm $log_{65537} N mod M$.

Factorization

Coppersmith showed how to use the algorithm to factorize RSA modulus N when high bits of a prime factor p (or q) are known. We slightly modified the method to perform the factorization with known structured primes. The fraction of known bits of the factor determines the optimal parameters (100 % success rate, best speed) of the algorithm. Coppersmith's algorithm is slowest when using the required minimum of known bits (half of the bits of the factor). With more bits known, the running time of the algorithm decreases.

Naive Algorithm

To find a factor (p), we have to find the integers k, a. A naıve algorithm would iterate over all options of $65537^a mod\ M$ and use Coppersmith's algorithm to attempt to find k.
The cost of the method is given by the number of guesses (ord) of a and the complexity of Coppersmith's algorithm. The term ord represents the multiplicative order of 65537 in the group $Z_{M^*}(ord = ord_M(65537))$. The number of attempts is too high even for small key sizes.

Main Idea

The bit size of M is analogous to the number of known bits in Coppersmith's algorithm. It is sufficient to have just $log_2(N)/4$ bits of p for Coppersmith's algorithm. In our case, the size of M is much larger than required ($log_2(M) > log_2(N)/4$). The main idea is to find a smaller M' with a smaller corresponding number of attempts $ord_{M'}(65537)$.

We are looking for M' such that:

- primes (p, q) are still of the same form – M' must be a divisor of M
- Coppersmith's algorithm will find k' for correct guess of a' – enough bits must be known ($log_2(M') > log_2(N)/4$)

- Overall time of the factorization will be minimal – number of attempts ($ord'_M(65537)$) and time per attempt (running time of Coppersmith's algorithm) should result in a minimal time.
- Trade-off between the number of attempts and the computational time per attempt as Coppersmith's algorithm runs faster when more bits are known.

Computing M'

The optimization of parameters is performed only once for all RSA keys of a given size. The running time (worst case) :

$$Time = ord_{M'}(65537) * T(M', m, t)$$

of the method is determined by the number of guesses of a' ($ord_{M'}(65537)$) and the average running time per attempt (computation of k' using Coppersmith's algorithm).
The running time of Coppersmith's algorithm is dominated by LLL reduction, and so by n= m+t of the square matrix. We focus on decreasing the value $ord' = ord_{M'}(65537)$. The most common key lengths used the following m, t values: m = 5, t = 6 for 512, m = 4, t = 5 for 1024, m = 6, t = 7 for 2048, m = 25, t = 26 for 3072, m = 7, t = 8 for 4096.

Optimizing M'

M' is selected as a candidate for an optimal M' (with the best m, t) if the value $ord_{M'}(65537)$ is sufficiently small but the size M' is large enough (Coppersmith's algorithm requires ($log_2(M') > \frac{log_2(N)}{4}$). The general strategy is to maximize the size of M' and simultaneously minimize the corresponding order. The value of M' was found in two steps : Using a greedy algorithm with a "tail brute force phase") to find an "almost" optimal M' denoted by M'greedy with the corresponding order ord'greedy. The value ord'greedy was used to reduce the search space of a "local" brute force search for a better M'. We used the below Algorithm that given ord' looks for the maximal M' (divisor of M) such that given ord' equals $ord_{M'}(65537)$.

Algorithm

Input : primorial M, ord' – divisor of ordM(65537)
Output : M' of maximal size with ordM'(65537) | ord'
$M' \leftarrow M$;
forall primes Pi | M do
$ordP_i \leftarrow ordP_i(65537)$;
. if $ordP_i$ does not divide ord' then
. $M' \leftarrow \frac{M'}{Pi}$;
. end
end
return M'

Greedy heuristic

In the greedy strategy, we try to minimize $ord_{M'}(65537)$ and simultaneously maximize the size of M' (to get $log_2(M') > \frac{log_2(N)}{4}$). In each iteration, we reduce (divide) ord' by some prime power divisor $p_j^{e_j}$ and compute the corresponding M' of maximal size using the above Algorithm. In the greedy choice, we select the most "valuable" prime power divisor $p_j^{e_j}$ of ord' that provides a large decrease in the order ord' at a cost of a small decrease in the size of M'. The divisor is chosen as the highest reward-at-cost value defined as:

$$\frac{\Delta size\ of\ ord_{M'}}{\Delta size\ of\ M'} = \frac{log_2(ord_{M'_{old}}) - log_2(ord_{M'_{new}})}{log_2(M'_{old}) - log_2(M'_{new})}$$

For M'_{new} computed by Algorithm with M'_{old} ,$ord' = ord_{M'_{old}}/p_j^{e_j}$ as an input. The reward-at-cost represents the bit size reduction of the order at the cost of the bit size reduction of M'.
Example:
The initial $M'_{old} = P_{39} = 2*3*.....*167$. The factorization of the initial order is: $2^4*3^4*5^2*7*11*13*17*23*29*37*41*83$. There are 20 candidates $2^1, ..., 2^4, 3^1, ..., 3^4, 5^1, 5^2, 7, ..., 83$ for the most valuable prime power divisor $p_j^{e_j}$ of ord' in the first iteration.
For the candidate $p_j^{e_j} = 83^1$, Algorithm eliminates 167 from M since $83^1 | ord_{167} = 166$ and 831 does not divide ord' .
Algorithm returns $M'_{new} = M'_{old}/167$ for the input values M'old , $ord' = ord_{M'}/831$. The reward-at-cost for 83^1 is computed as $log_2 83/log_2 167 = 6.37/7.38$.
For the candidate 17^1, Algorithm eliminates 103, 137. So, reward-at-cost is $log_2 17/(log_2 103 * log_2 137)$ = 4,08/13,78. reward-at-cost for 17^1 is computed as $\frac{log_2 17}{log_2(103*137)} = \frac{4,08}{13,78}$, etc. The most valuable candidate in the first iteration is 831. In the second iteration, $ord_{M'} = 2*3*...*7*11*13*17*23*29*37*41$ and $M'_{old} = M/167$.
The best candidates for $p_j^{e_j}$. are found to be: $83^1, 53^1, 41^1, 29^1, 37^1, 23^1, 17^1, 3^2, 11^1$. The iterations end when the greedy heuristic computes M' that is too small ($log_2(M') < \frac{log_2(N)}{4}$). The resulting M' = 0x55eb8fbb4ca1e1879d77 from the previous iteration is computed by Algorithm for M' = M/83/53/ $\cdots$ /11.

Local brute force

There are two ways to perform the brute force search through all divisors ord' of $ord_M(65537)$ ($M' \| M \implies ord_{M'} \| ordM$) and compute the corresponding M' from ord' using Algorithm since the search space for ord' is significantly smaller than that for M'. For example, for 512-bit RSA keys, $M = P_{167}$ is product of 39 primes, i.e., there are 239 different divisors of M, while there are only 220 different divisors of $ord_M(65537) = 2^4*3^4*5^2*7*11*13*17*23*29*37*41*83$.

Further optimisation

Since both p, q are of the same form, our method can also find the factor q for $x \equiv b' mod\ ord'$. Hence, our method is looking simultaneously for p and q. This fact can be used

to halve the time needed to find one of the factors p, q of N . In order to optimize the guessing strategy, we are looking for the smallest subset (interval) of Zord' that contains either a' or b' . We use the value c' obtained during the fingerprinting (a discrete logarithm of N) to describe the desired interval. The interval is of the following form:

$$I = \left[\frac{c'}{2}, \frac{c' + ord'}{2}\right]$$

It is easy to see that either a' or b' ($c' \equiv a' + b' mod\ ord'$) occur in the interval I and that the size of the I is the smallest possible.

Problem Statement

The goal is to efficiently factorize N into prime factors pq.
Tools we have in hand:

1. Coppersmith's Method: This method finds small solutions univariate modular equations(if they exist) in Polynomial time.

$$p(x) \equiv 0\ (mod\ N)$$

2. Improvement of this method later introduced by Howgrave-Graham which simplifies the computation.

3. LLL Algorithm : This is used for finding LLL-reduced(nearly orthogonal) lattice basis in Polynomial time.

Conversion

We exploit the "structural form" of generated primes

$$p = k' * M' + (65537^{a'}\ mod\ M')$$

Hence we construct the polynomial:

$$f(x) = x * M' + (65537^{a'}\ mod\ M')$$

We know that $x_0 = k'$ will satisfy

$$f(x_0) \equiv 0\ (mod\ p)$$

To make the leading coefficient as 1 and to apply Coppersmith Method, we multiply by $(M')^{-1}\ mod\ N$, we get:

$$f(x) = x + ((M')^{-1}\ mod\ N) * (65537^{a'}\ mod\ M')\ (mod\ N)$$

Since p and q are of half the bit size as N, to find value of β we see that without loss of generality, $\beta = \frac{1}{2}$ satisfies

$$p < N^{\beta}$$

And for upper bound on x_o, we get

$$x_0 < 2 * \frac{N^{1/2}}{M'}$$

Note that the equation:

$$f(x) = x * M' + (65537^{a'} \ mod \ M')$$

has 2 unknowns x and a'. This leads to the idea that we "guess" the smaller variable a' and then proceed with x calculation. This will give us a guess of p which we can check with N to confirm the factorization.

Therefore we bring down the number of brute force guesses from size of p to size of a' itself(which is much smaller than p). This comes with some extra computation cost for each guess value of a'.

Generated prime is of the form

$$p = k * M + (65537^{a} \ mod \ M)$$

If we take another number M' such that

$$M' \mid M$$

We can also write the same equation with different coefficients $\{a', k', M'\}$ such that

$$p = k' * M' + (65537^{a'} \ mod \ M')$$

This can be viewed as a mapping

$$\{a, k, M\} \longrightarrow \{a', k', M'\}$$

It is sufficient to have information of just $(log_2 N)/4$ for Coppersmith's Attack. But if suppose we know the number 'a' by brute force, then we will have information on $(log_2 M) > (log_2 N)/4$. Thus by the above stated mapping, we can reduce the brute force guesses to find a' while the following equation still holds

$$(log_2 M') \geq (log_2 N)/4$$

so that we can still apply Coppersmith's attack.

Howgrave-Graham method This is a slight improvement in Coppersmith's method and it also uses LLL reduction to reduce the basis matrix. The basis matrix is formed from the coefficicents of the polynomial generated by the the following relations:

$$f_i(x) = x^j N^i f^{m-i}(x) \quad i = 0, ..., m-1 \quad j = 0, ..., \delta - 1$$

$$f_{i+m} = x^i f^m(x) \quad i = 0, .., t-1$$

This defines the basis matrix which influences the time of computation

Personal Section

Challenges in Understanding

All the team members in the group enjoy discrete mathematics. This is one of the key reasons we chose this project over others. The sheer depth of mathematics involved was overwhelming enough to give us a true insight of a Çryptography Project". Understanding various aspects of closest vector problem, LLL-algorithm, Coppersmith Algorithm and the precise implications of the statements was a challenging task. Not only this, to understand an attack, we need to understand previously discovered attacks which have been patched. It is not that a single algorithm is enough, there are several subroutines and instances of other algorithms which have to work together. This was all studies and we were able to complete the reading part in 2 weeks.

Challenges in Design and Implementation

We implemented Pohlig-Hellman algorithm, LLL algorithm, algorithm to optimise M and Howgrave-Graham algorithm (modified version of Coppersmith algorithm). We used python and *Sage* library for implementing these algorithms. Sage provides us with forming polynomial, making matrix , reducing the matrix using LLL algorithm (though we have implementing this independently) and calculating integral roots of the polynomial used to calculate p & q.

Interesting Ideas

- An interesting idea presented by paper is exploiting structural patterns in primes which reduce the entropy or randomness. This inherently means that knowing a few critical amount of bits allows you to know entire prime after some computational cost.
- Optimising M, as it was the most important part of the paper to make it practical
- The other novel idea is to use Closest Vector Problem and map it to find unknown bits of a structural entity. This is very powerful and has huge implications.

Critical Analysis

The paper is very well written. All the values provided which could have been tested by us were verified and found out to be very close to the reported values. We are really impressed by the combination of various fields of mathematics that come together in this mathematical formulation. Matrix reduction, discrete mathematics, combinatorics, etc. all have some application which makes this attack successful.

Observations

The value of M' found using the bruteforce method is of 146 bits for 512-bit key. With this value of M', value of ord' becomes 3603600 ($2^{20,02}$) which is practical and 512-bit key

takes around 2.1 hrs to factorise (average case is half of this).

Referencias

[1] The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. CCS'17, October 30-November 3, 2017, Dallas, TX, USA

[2] `https://github.com/CheckResearch/confccsNemecSSKM17_Experiment_01/tree/master/code`