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

Yateesh Agrawal 170050005

Aman Kansal 170050027

Aditya Sharma 170050043

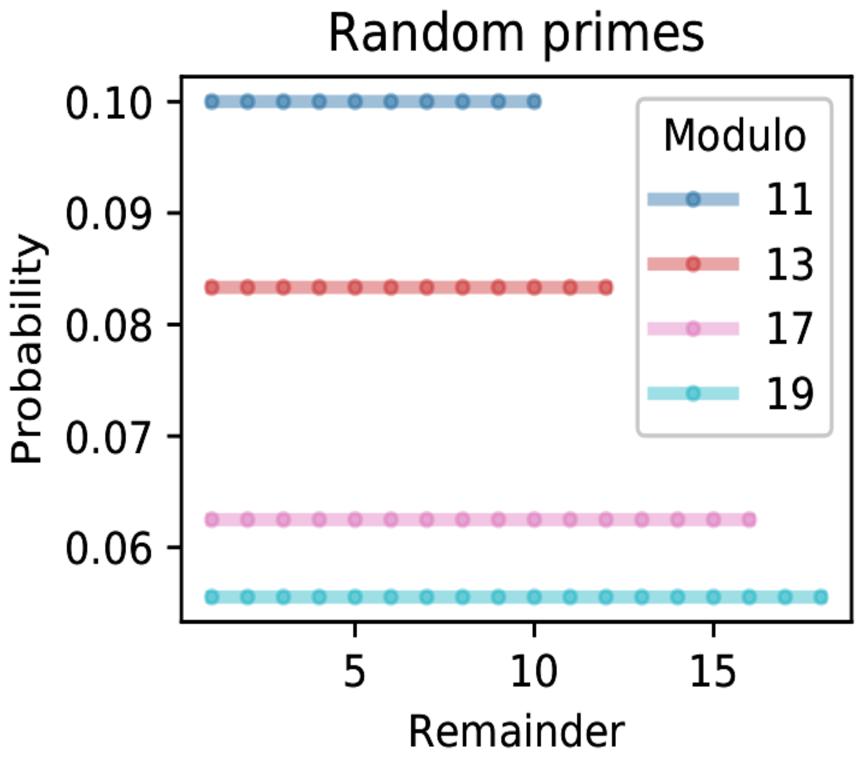
Suraj 170050044

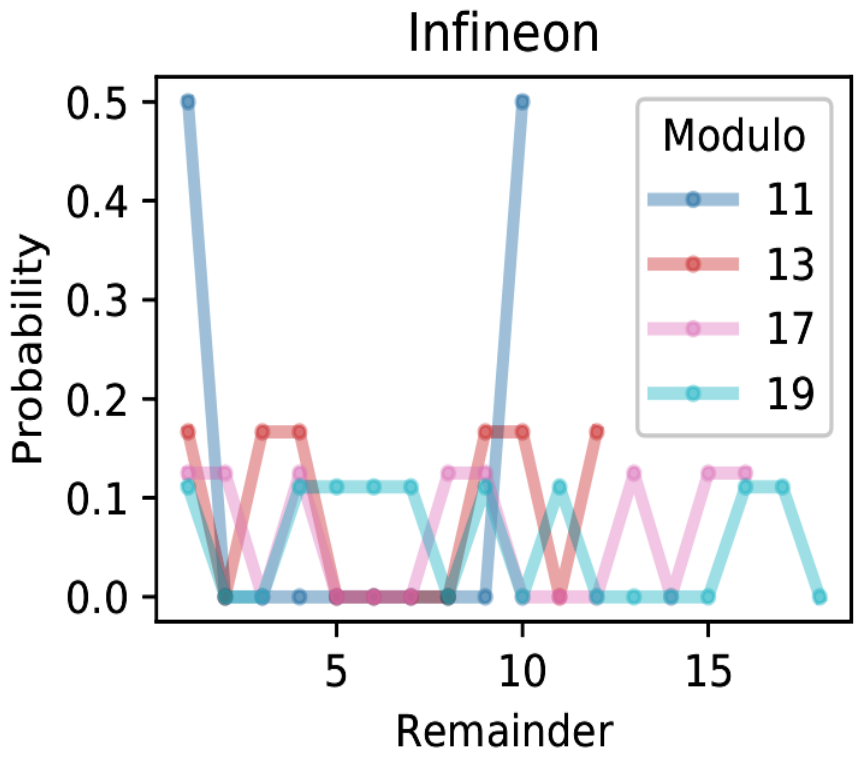
What the project is based on?

- Mathematical attack
- Exploiting the structure of primes generated by NIST FIPS 140-2 certified library - RSALib
- Factorizing N into p & q

Motivation

Distribution of RSA keys modulo small primes:

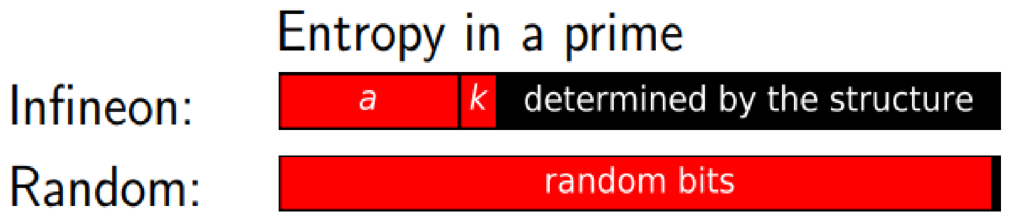




RSA keys and Vulnerability

Structure of primes generated by the library

$$N = p * q$$
$$p_{ideal} = \text{random prime}$$
$$p_{Infineon} = (k * M + 65537^a \bmod M); \; a, k \in \mathbb{Z}$$
$$M = 2 * 3 * 5 * 7 * \cdots * P_n$$



- All RSA primes (as well as the moduli) generated by the *RSALib* have the above form.
- *M* is a known Primorial, the product of Consecutive Primes (*n* = 39 for 512-bit key and 71 for 1024-bit key).
- For 512-bit key, M = 2*3*5*…*167 = 219-bit number.
- We therefore see that there exists a pattern that deviates from actual random prime. This is exploited.
  Sizes of *k* and *a* are small (e.g., *k* has 256 − 219 = 37 bits and *a* has 62 bits for 512-bit RSA).

Reason for "Structured Primes"

- Some constructions safer against factorization methods such as Polard (p-1) method.
- Some constructions have at least one large factor of p-1 & p+1. Therefore, for small integers B, it will not be a B-smooth number.
- Speedup of keypair generation, since testing random candidate values for primality is time consuming, especially on restricted devices like smartcards.

Fingerprinting

$$N = p*q$$

$$N = (k * M + 65537^{a} \bmod M) * (l * M + 65537^{b} \bmod M) \text{ for } a,b, k,l \in Z$$

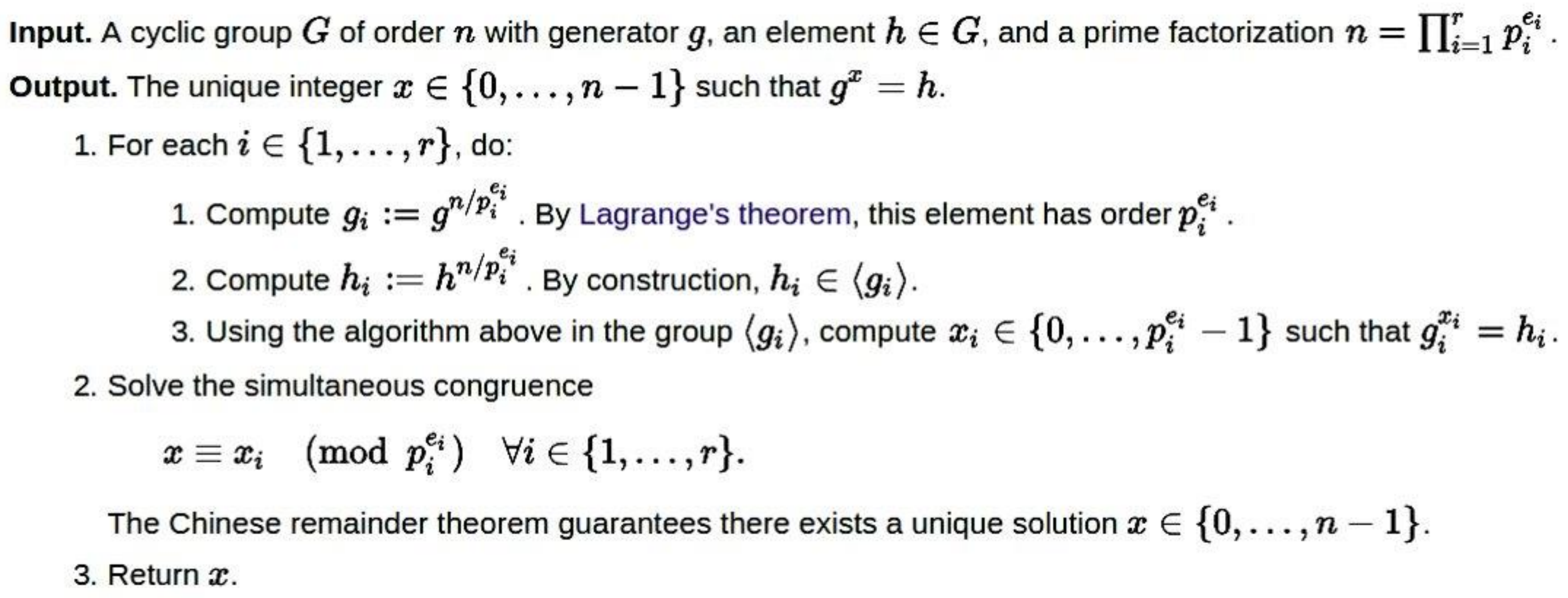
$$\Rightarrow N \equiv 65537^{a+b} \equiv 65537^{c} \bmod M$$

Task : To check the existence of c and finding it's value, if exists.

Property exploited : M has small factors (Smooth number) and hence is vulnerable to Pohlig-Hellman attack.

False positives : The existence of the discrete logarithm serves as a very strong fingerprint of the keys.
While random primes modulo M cover the entire Z_M^*, the RSALib generates primes from the group *G* - a tiny portion of the whole group.
For example, $|G| = 2^{62.09}$ while $|Z_M^*| = \phi(M) = 2^{215.98}$ for 512-bit RSA.
The probability that a random 512-bit modulus N is an element of G is $2^{62-216} = 2^{-154}$.
Hence, we can make the following conclusion with high confidence: an RSA key was generated by the RSALib if and only if the Pohlig-Hellman algorithm can find the discrete logarithm ($\log_{65537} N \bmod M$).
No false positives found within a million of tested keys.

Pohlig – Hellman algorithm



Input. A cyclic group G of order n with generator g, an element $h \in G$, and a prime factorization $n = \prod_{i=1}^{r} p_i^{e_i}$.

Output. The unique integer $x \in \{0, \ldots, n-1\}$ such that $g^x = h$.

1. For each $i \in \{1, \ldots, r\}$, do:
    1. Compute $g_i := g^{n/p_i^{e_i}}$. By Lagrange's theorem, this element has order $p_i^{e_i}$.
    2. Compute $h_i := h^{n/p_i^{e_i}}$. By construction, $h_i \in \langle g_i \rangle$.
    3. Using the algorithm above in the group $\langle g_i \rangle$, compute $x_i \in \{0, \ldots, p_i^{e_i} - 1\}$ such that $g_i^{x_i} = h_i$.
2. Solve the simultaneous congruence

$$x \equiv x_i \pmod{p_i^{e_i}} \quad \forall i \in \{1, \ldots, r\}.$$

   The Chinese remainder theorem guarantees there exists a unique solution $x \in \{0, \ldots, n-1\}$.

3. Return x.

Time complexity

$$\mathcal{O}\left(\sum_i e_i(\log n + \sqrt{p_i})\right)$$

The code takes around microseconds to find value of c.

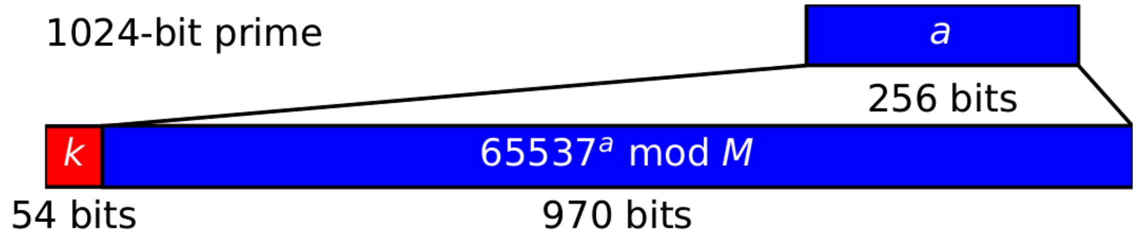
Factorisation

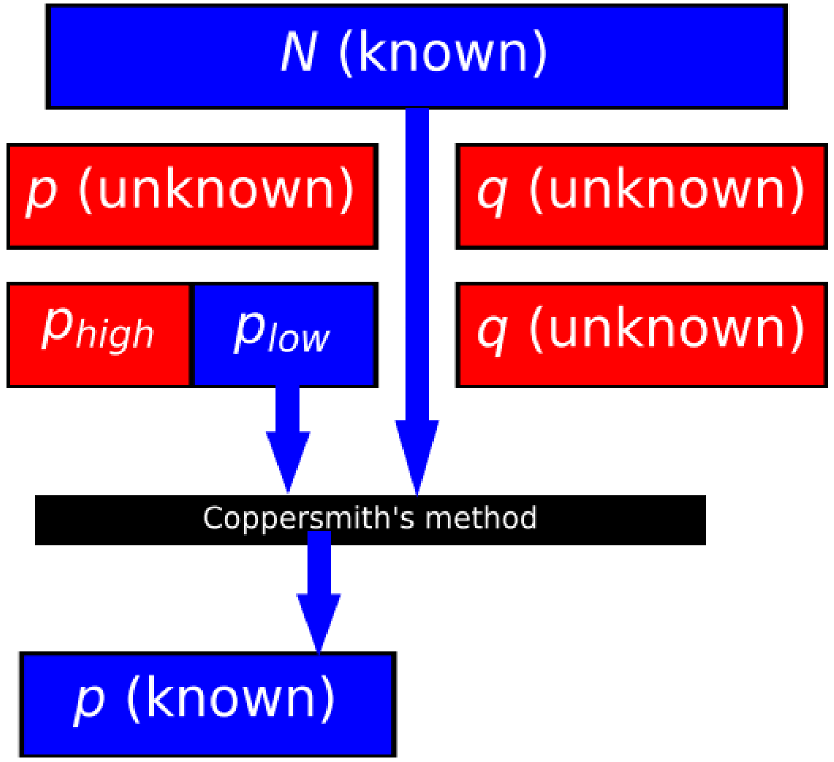
Coppersmith's Algorithm as a black-box

Partial knowledge of private key => Full private key

$$p = \mathbf{k} * M + 65537^{\mathbf{a}} \bmod M$$

Guess **a** and compute **k** using Coppersmith's method

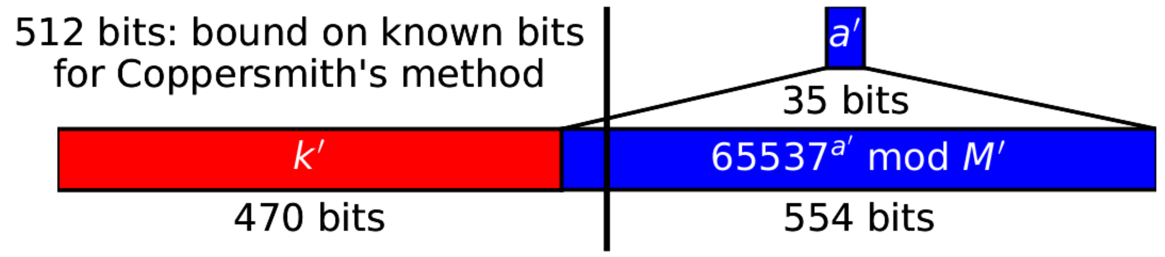




For 512-bit key : a is 62-bit number.
Number of guesses : 2^{62} (Not practical)
or multiplicative order of 65537 mod M = 2454106387091158800
= $2^{42.34}$ (also not practical)

Optimization

a is still too large – find a smaller M' (divisor of M)



Main Idea : Smaller divisor *M'* of *M*.

- 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 (*log2 (M') > log2 (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.

The optimized values of *M'* for different key lengths were found along with parameters *(m,t)* using a local brute force search optimized by the results of a greedy heuristic.

For 512-bit key : M' = 45092197507717713174331239195330276496994910 – 145.02 bits

Multiplicative order : 3603600 ($2^{21.78}$) -- Paper claims $2^{20.20}$

This can be further reduced by a factor of 2 using the range as I = [c/2,(c+ord)/2]. (Only ord/2 as compared to ord)

Because we can surely say that as (a+b = c mod ord), so atleast one of a or b must lie in range I.

Observations

For 256-bit key(N): although it is really very small

Order = 27720 ($2^{14.76}$)

Time taken to factorise -- 42.83, 14.91, 47.25, 10.01, 21.28, 19.86 seconds

For 512-bit key(N):

Order = 3603600($2^{21.78}$)

Time taken to factorise – 7m8.18sec for x = 100,000

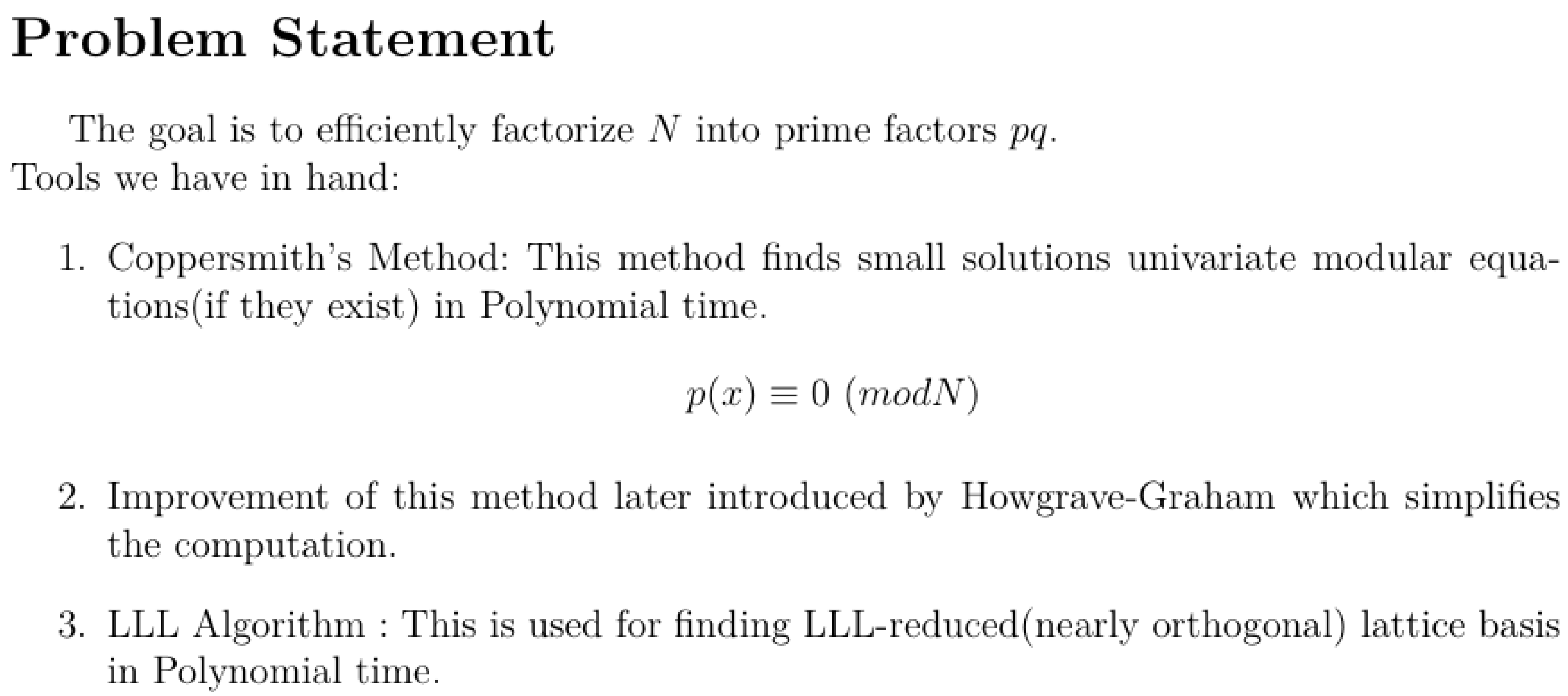
Worst case x can be 36 times the given value.

Hence, it can take 7*36 min = 4.2 hrs

As I said on last slide, we can reduce time by a factor of 2 (by reducing the size of interval I).

So, worst case time is 2.1 hrs (The paper claimed 1.93 CPU hrs)

Formulation of Problem



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.

$$p(x) \equiv 0 \ (modN)$$

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.

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.

$$problem \quad \rightarrow \quad f(x) \equiv 0 \bmod p \quad \rightarrow \quad g(x) = 0 \quad \rightarrow \quad x_0$$

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:

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

Coppersmith's Theorem

Theorem (Coppersmith). *Let N be an integer of unknown factorization, which has a divisor $b \geq N^{\beta}$, $0 < \beta \leq 1$. Let $0 < \epsilon \leq \frac{1}{7}\beta$. Furthermore, let $f(x)$ be a univariate monic polynomial of degree δ. Then, we can find all solutions x_0 for the equation*

$$f(x) = 0 \bmod b \quad \textit{with} \quad |x_0| \leq \frac{1}{2} N^{\frac{\beta^2}{\delta} - \epsilon}.$$

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'.

Mitigation and Disclosure

Vulnerability found in Jan 2017 and disclosed to Manufacturer in Feb 2017. Public disclosure of the discovered issue in the middle of October 2017 and these steps were taken to reduce impact of ROCA Attack.

- Since ROCA is mathemetical attack change of algorithm is inarguably best solution
- For RSALib in read only memory, some vendors with secure hardware implementation like Yubico imported keys from another libraries
- Implementations where protocol changes were easy shifted bit size to less effected sizes like 1952 and 3936.

Ending information

- Vulnerability of library and does not depend on weak or a faulty random number generator
- No additional information except the value of N
- Number of affected devices to be in the order of at least tens of millions
- **Real-World Impact Award** as per http://ccs2017.sigsac.org/awards.html