CS355 Lecture #4 :

Attacking deployed RSA.

Quick recap: Random Oracles.

The Random Oracle model (Bellare & Rogaway, CCS '93) is a pragmatic tool for building cryptosystems.

↳ but: security is heuristic when we instantiate!

ROM world	Real world
$H(x)$ is a public random function	$H(x) \triangleq \mathrm{SHA3}(x)$
Can prove properties of protocols that call $H(x)$.	Must assume that no adversary can exploit the structure of SHA3.

Today: Attacking real world deployments.

MSB: the problem is usually not "the crypto"

1. Reminder: RSA
2. Mining Your P's & Q's
   → randomness failure
3. Return of Coppersmith's Attack
   → "optimization" = ouch

$\boxed{1}$ Reminder : RSA [RSA77]

Needless to say : DO NOT IMPLEMENT BASED ON THIS DESCRIPTION !!!

Let N be an integer with (secret) factorization

$$N = pq.$$

$$PK \triangleq (N, e)$$

→ coprime to $(p-1)(q-1)$

⇒ usually $2^{16}+1$

$$SK \triangleq (p, q)$$

informationally equivalent

OR $d = e^{-1} \bmod (p-1)(q-1)$

- Recall: PK defines a permutation on $\mathbb{Z}$ mod N:

from CS255

$$x \longmapsto x^e \bmod N$$

Without SK, this seems hard to invert for big N (think: N is >2048 bits).

- Given SK, inverse is

$$y \longmapsto y^d \bmod N$$

⇒ $\mathbb{Z}_N^*$ has order $\varphi(N)=(p-1)(q-1)$

So $(x^e)^d \equiv x \mod N$.

RSA "trapdoor" permutation gives:

- → Encryption (e.g. RSA-OAEP$^+$ due to Shoup '01)
- → Signatures (e.g. RSA-PSS due to Bellare & Rogaway '96)

Never use the RSA permutation directly for encryption or signing!

↑ this is the most important item in today's lecture.

$\boxed{2}$ Mining your Ps & Qs

- Heninger, Durumeric, Wustrow, Halderman

⇒ USENIX Security 2012

- Also: Lenstra, Hughes, Augier, Bos, Kleinjung, Wachter 2012
  "Ron was wrong, Whit is right"

⇒ ePrint #2012/064.

<u>Idea</u>: if two RSA moduli $N_1 \neq N_2$ have nontrivial GCD, we can <u>factor</u> both. ← break!

$$\left.\begin{aligned} N_1 &= p\cdot q_1 \\ N_2 &= p\cdot q_2 \end{aligned}\right\} \mathrm{GCD}(N_1, N_2) = p$$

↳ now divide

Strategy:

1. Collect millions of RSA PKs
2. Check for nontrivial GCDs
3. Profit?

Step 1: Scan entire IPv4 address range for TLS certificates

$< 2^{32}$ addresses.

Zakir's software, zmap, does this in *minutes*!

OR, maybe: collect RSA pubkeys from GitHub?

⇒ millions are available...

Step 2: How do we check $\approx 2^{23}$ keys for pairwise GCDs?

→ Naïvely, 2^{46} pairs of keys ⟹ 30 CPU-years

↳ already OK for NSA.

→ Better: <u>GCD tree</u>

Daniel Bernstein, "How to find smooth parts of integers." Manuscript, 2004.

"$\tilde{O}$" hides log factors.

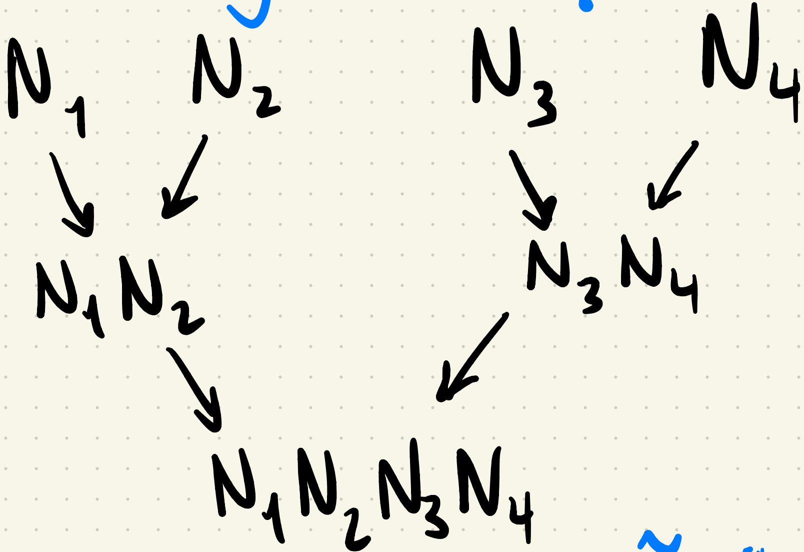
- GCD costs $\tilde{O}(n)$ for n-bit inputs
- For k keys, naive cost is $\tilde{O}(k^2 n)$
- GCD tree reduces cost to $\tilde{O}(kn)$

↳ Concretely: years become days.

GCD tree idea:

(1) Compute $\pi = \prod_{i=1}^{k} N_k$

→ using a tree!



⇒ total cost $\tilde{O}(kn)$

(2) Compute $r_1 = \pi \bmod N_1^2$
(and so forth)

(3) $GCD(\frac{r_1}{N_1}, N_1)$ gives a common factor between N_1 & some other N_i (etc.)

Why does (3) work?

$$\Pi = N_1 N_2 N_3 N_4$$

assume $N_1 = pq_1$, $N_2 = pq_2$, no other common factors

$$\Rightarrow \Pi = (pq_1)(pq_2) \times \underbrace{N_3 N_4}_{\hookrightarrow (\text{garbage})}$$

$$\Rightarrow r_1 = \Pi \bmod N_1^2$$

$$= p^2 q_1 q_2 (\text{garbage}) \bmod N_1^2$$

$$\Rightarrow \frac{r_1}{N_1} = pq_2 (\text{garbage}) \bmod N_1^2$$

$$= pq_2 (\text{garbage})$$

$$\Rightarrow \text{GCD}\left(\frac{r_1}{N_1}, N_1\right)$$

$$= \text{GCD}\left(pq_2(\text{garbage}), pq_1\right)$$

$$= p \longleftarrow \text{got it!}$$

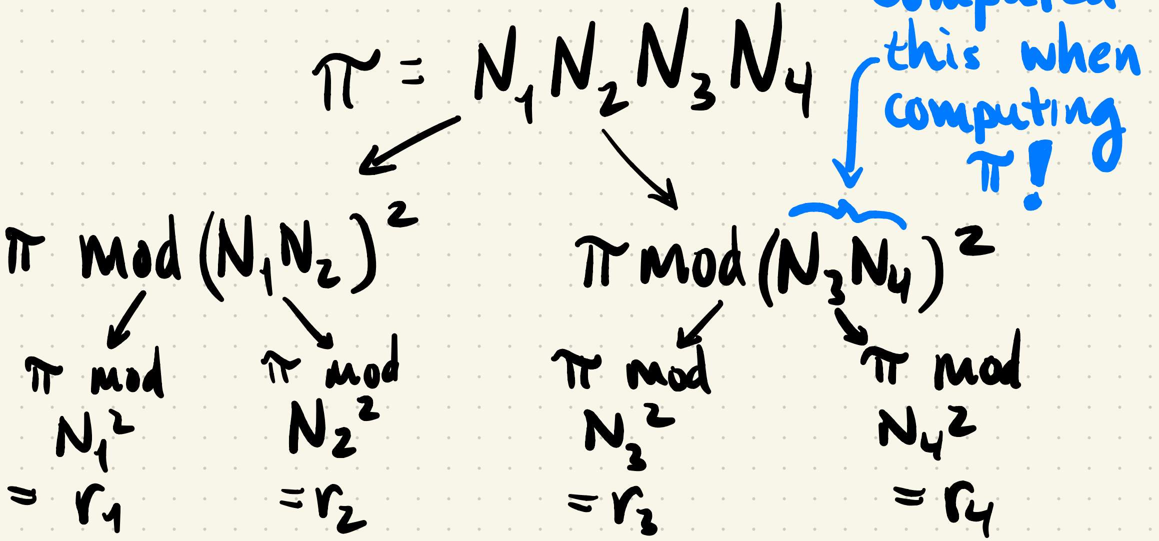
How do we compute r_i?

Naïvely, **each** $\pi \bmod N_i^2$ costs $\tilde{O}(kn)$

↳ π is $k \cdot n$ bits!

⇒ Computing all r_i costs $\tilde{O}(k^2 n)$

Better: a tree!



⇒ values **shrink** at each step

⇒ $\tilde{O}(kn)$ cost in total!

Profit?

Heninger et al. factored >64000 keys that they found in the wild!

⇒ WHY ARE KEYS SO BAD?

→ IBM made devices that chose from a list of 9 possible prime factors

→ Embedded systems often have trouble generating "good" randomness

↳ at boot, devices generate keys before they have gathered sufficient entropy

→ see also: Debian RNG fiasco 2008.

3 Return of Coppersmith's Attack

Nemec, Sys, Svenda, Klinec, Matyas

ACM CCS 2017.

Idea: for RSA moduli $N = pq$ where p & q have special structure, we can factor N.

we'll see what this means soon.

Why special structure?

$\Rightarrow$ generate keys using fewer random bits on embedded devices (made by Infineon)

again!

Result: millions of devices were recalled!

The special structure:

$$p = k \cdot M + (65537^a \bmod M)$$

$$q = l \cdot M + (65537^b \bmod M)$$

for a public constant M, a primorial — the product of the first j primes.

Aside: why this choice of M?

↳ guarantees p & q are not divisible by small primes

⇒ fewer primality tests, so faster key generation

⇒ a tragic optimization

Intuition leads us astray:

$$p = k \cdot M + (65537^{a} \bmod M)$$

for 1024-bit p, M is

primorial notation

"$P_{126}\#$" — 971 bits.

$\Rightarrow$ k has ≈ 53 bits,

a has > 100 bits

$> 2^{128}$ choices for p —

$\Rightarrow$ what could go wrong?

Coppersmith's algorithm (Eurocrypt '96) lets us recover p, q from N in polynomial time if we know $> 1/2$ the bits of either!

Theorem (Coppersmith '96):

Let $N = pq$ be an RSA modulus.

Let $f \in \mathbb{Z}_N[x]$ be a polynomial of degree d.

Then we can find all integers x_0 s.t. $f(x_0) = 0 \bmod p$ where $|x_0| \leq N^{1/4d}$ in time polynomial in d and $\log N$.

Note: Since we find all such solutions, there can only be $\text{poly}(d, \log N)$ of them.

Recall: $p = kM + (65537^a \bmod M)$

Attack: (1) guess a

(2) recover k w/ Coppersmith.

Step (2) first: given a,

$$p = \underset{M}{c_1} \cdot k + \underset{65537^a \bmod M}{c_2}$$

$$f(x) \triangleq c_1 x + c_2$$

$$\Rightarrow \underline{f(k) = p \equiv 0 \mod p}$$

Coppersmith? $\deg(f) = 1$

- So we will get candidate k values up to $N^{1/4}$. p is 1024 bits, M is 971 bits $\Rightarrow$ real $k \ll N^{1/4}$
- For each candidate k, try factoring N. (We know there won't be too many...)

Step (1): Guessing a's value.

Really, guess $C_2 = 65537^a \bmod M$

$\Rightarrow$ the constant term of $f(x)$.

How many values of C_2 are there?

Equivalently: what is the size of the subgroup of $\mathbb{Z}_M^*$ generated by 65537?

$\Longrightarrow$ If M were prime, could be as large as $M-1$.

$\Longrightarrow$ But M is Smooth, so the subgroup is small (and its size is easy to compute).

many small factors

So: compute size of subgroup r, then "guess" $0 \leq a < r$ and run step (2).

Optimizing the attack:

- M is "too big":
  Coppersmith gives solutions up to $N^{1/4}$, but we only need ≈ 53-bit k

Idea: pick M' dividing M s.t. $1024 - \log_2(M') \leq N^{1/4}$

this ensures that we find the solution with Coppersmith

$\Rightarrow$ now $p = k'M' + 65537^{a'} \bmod M'$

order of 65537 is smaller in $\mathbb{Z}_{M'}^*$ than in $\mathbb{Z}_M^*$ — fewer guesses

Trade-off: bigger M' makes Coppersmith faster but takes more guesses for c_2

$\Rightarrow$ optimize!

Results

Key size	Cost to break on AWS	
512 bits	40 minutes, 6.3¢	✓
1024 bits	32 days, \$76	✓
2048 bits	46 years, \$40k	✓
3072 bits	10^{25} years, $\$10^{28}$	X
4096 bits	4×10^{8} years, $\$3 \times 10^{11}$	X

Note: attack is trivially parallelized!

(How?)

Identifying bad moduli

Recall:

$$p = k \cdot M + (65537^a \bmod M)$$

$$q = \ell \cdot M + (65537^b \bmod M)$$

So $N = pq \equiv 65537^{a+b} \bmod M$

A random RSA modulus has vanishingly small chance of having this form — $\lll 2^{-100}$

So: check if $N \bmod M$ has a discrete log to the base 65537.

$\Longrightarrow$ This is easy because M is smooth (many small factors)

$\Rightarrow$ Next lecture: discrete log!