4/6/22

Lecture 4: Real World Cryptanalysis

Logistics

- HW1 due Friday 5pm
- Please tell us if none of the OH times work for you!

Today

- Recap: Random Oracles
- How RSA Can Break in Practice
  - GCD Attack (2012)
  - Infineon Bug

Recap: Random Oracles

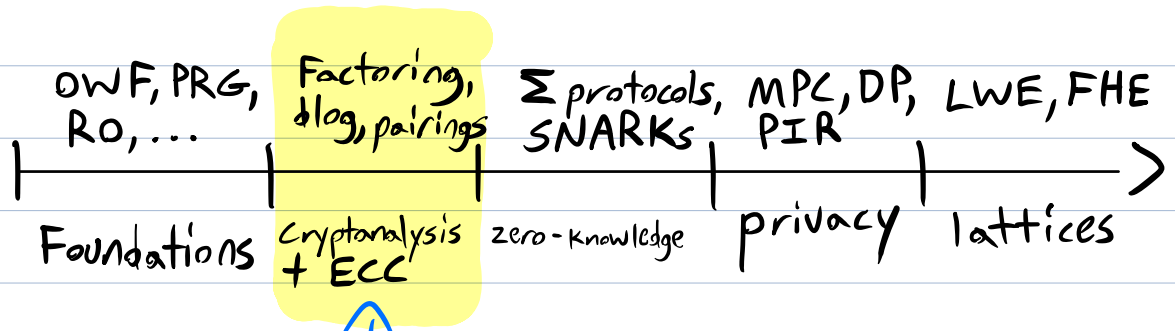
Pragmatic tool for building simple schemes w/ heuristic (but plausible) security in practice

e.g.) $F(k, x) = H(x)^k$ is a secure PRF if H is a truly random function

Random Oracle World	Real World
 $F(k, x) = H(x)^k$	$F(k, x) = SHA3(x)^k$
We can prove that no PPT adversary can distinguish F from a truly random function (assuming hardness of DDH)	We can't prove that F is a secure PRF. But any attack against it must exploit the fact that SHA3 is not a random function (or break DDH)

Real World Cryptanalysis

So far...



We're Here!

Strong Math foundations => Deployed Crypto is unbreakable? right?

Nope! :(

Why?

- Crypto is misused
  - Poor API Designs
  - implementation bugs
  - side-channels
  - bad randomness
  - unsafe optimizations
- Some "hard" problems aren't actually hard

"sub-exponential" time algorithms
- Factoring
- dlog in $\mathbb{Z}_p^*$ (next lecture)

↳ Specific parameters (e.g. key sizes, choice of group, etc) matter a lot!

GCD Attack on RSA (Lenstra et al 2012, Heninger et al 2012)

"Weakness in numbers"

Background

- RSA is used everywhere for encryptions & signatures SSH, TLS, IPSec, PGP, ...

- If you connect to an SSH or TLS host, it will send you its public key

$pk = (N, e)$

- e: encryption exponent (often $e = 65537$)
- $N = pq$ for large primes p, q

$sk = (p, q)$

Idea: Scan the entire IPv4 address space to collect all public keys
Takes 5 min w/ specialized tool (Zmap) & a good connection

aaa.bbb.ccc.ddd → 32 bit addresses
→ $2^{32} \approx 4$ billion

Results:

- collected keys from { 12.8 million TLS hosts
  { 10.2 million SSH hosts
  - 5% of TLS hosts & 10% of SSH hosts don't have unique keys!
  - private keys can be recovered for { 64K TLS hosts (0.5%)
    { 2.5K SSH hosts (0.03%)

What happened?

The scan found many RSA moduli sharing exactly one prime factor

This is worse than sharing both factors (i.e. a duplicate key)

$N_1 = pq_1$, $N_2 = pq_2$ for distinct primes p, q_1, q_2

Both N_1 & N_2 are hard to factor on their own, but together...

$$\gcd(N_1, N_2) = p$$

($n = \log N$ bit length of modulus)

gcd can be computed in time poly(n)

↳ Euclid's algorithm (~300 BC)

Actually $\tilde{O}(n)$ (hides log factors)

Problem

The scan found ~ 9 million RSA keys $\approx 2^{23}$

Computing pairwise gcd among k keys:

$\Omega(k^2)$ gcds

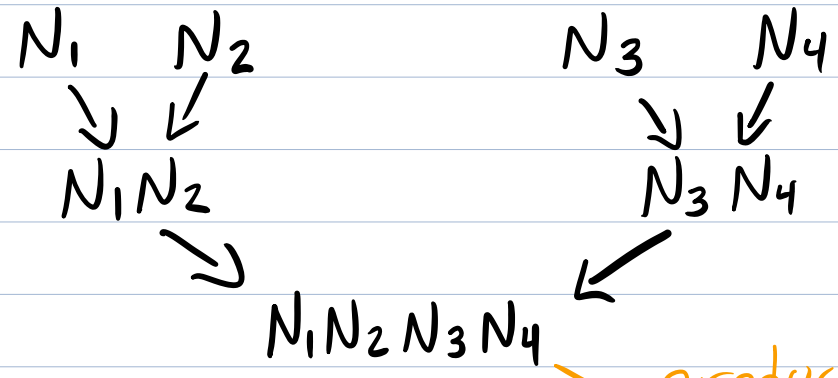
$\Rightarrow \sim 2^{46}$ gcds! ($\sim$ 30 CPU years)

↳ feasible but expensive (NSA can do this)

GCD tree idea:

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

Using a tree!



product of all moduli

$\Rightarrow$ total cost $\tilde{O}(kn)$

# of keys

bit length per key

(2) Compute $r_i = \Pi \bmod N_i^2$

$r_2 = \Pi \bmod N_2^2$

...

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

Why does this work?

$\Pi = N_1 N_2 N_3 N_4$

Corrected: 4/7/22

assume $N_1 = pq_1$, q_1 does not divide N_3 or N_4
$N_2 = pq_2$

By the definition of r_1, we can express Π as:

$$N_1 N_2 N_3 N_4 = K N_1^2 + r_1$$

(K: quotient; r_1: remainder)

Lemma: $N_1 \mid r_1$ (N_1 divides r_1)

pf: $N_1 N_2 N_3 N_4 - K N_1^2 = r_1$

$N_1 (N_2 N_3 N_4 - K N_1) = r_1$

$$\underbrace{N_2 N_3 N_4 - k N_1}_{\text{whole \#}} = \frac{r_1}{N_1}$$ □

$$\gcd\left(\frac{r_1}{N_1}, N_1\right)$$

$$= \gcd\left(N_2 N_3 N_4 - k N_1, N_1\right)$$

$$= \gcd\left(\underbrace{p q_2 N_3 N_4 - k p q_1}_{p \text{ divides}}, \underbrace{p q_1}_{\text{\&}}\right)$$

Now need to show that q_1 does not divide either

Because $q_1 \mid k p q_1$ but $q_1 \nmid p q_2 N_3 N_4$, q_1 can't divide their sum

So p is the GCD

$$= p$$

We got it! ☺

So, compute $r_1, r_2, \ldots, r_k$ & do k gcds (instead of k^2)

How do we compute r_i?

Naively, each $\Pi \bmod N_i^2$ costs $\tilde{O}(kn)$

Π is kn bits!

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

How about a tree again?

$\Pi = N_1 N_2 N_3 N_4$

$\Pi \bmod (N_1 N_2)^2$ $\quad$ $\Pi \bmod (N_3 N_4)^2$

Already computed this when computing Π!

$\Pi \bmod N_1^2$	$\Pi \bmod N_2^2$	$\Pi \bmod N_3^2$	$\Pi \bmod N_4^2$
$\|\|$	$\|\|$	$\|\|$	$\|\|$
r_1	r_2	r_3	r_4

Values shrink at each step

$\Rightarrow \tilde{O}(kn)$ cost in total!

This means all pairwise gcds in

$\tilde{O}(kn)$ time instead of $\tilde{O}(k^2 n)$

5 hours 30 years

How did RSA keys end up sharing a prime factor?

1) By chance? (not likely)

2) Very bad implementations
(e.g. IBM mgmt interfaces)

keyGen( ) {

// hard coded values – m primes

primes = {$p_1, \ldots, p_m$}

$p \xleftarrow{R}$ primes

$q \xleftarrow{R}$ primes

return $N = pq$

}

Only $\binom{m}{2} \approx m^2$ possible moduli

What's the probability that 2 moduli share exactly one factor?

3) Unfortunate sequence of unlikely events

- Many embedded devices (e.g. routers) speak TLS

- Linux gathers "random" values from I/O devices (keyboard, mouse, etc)

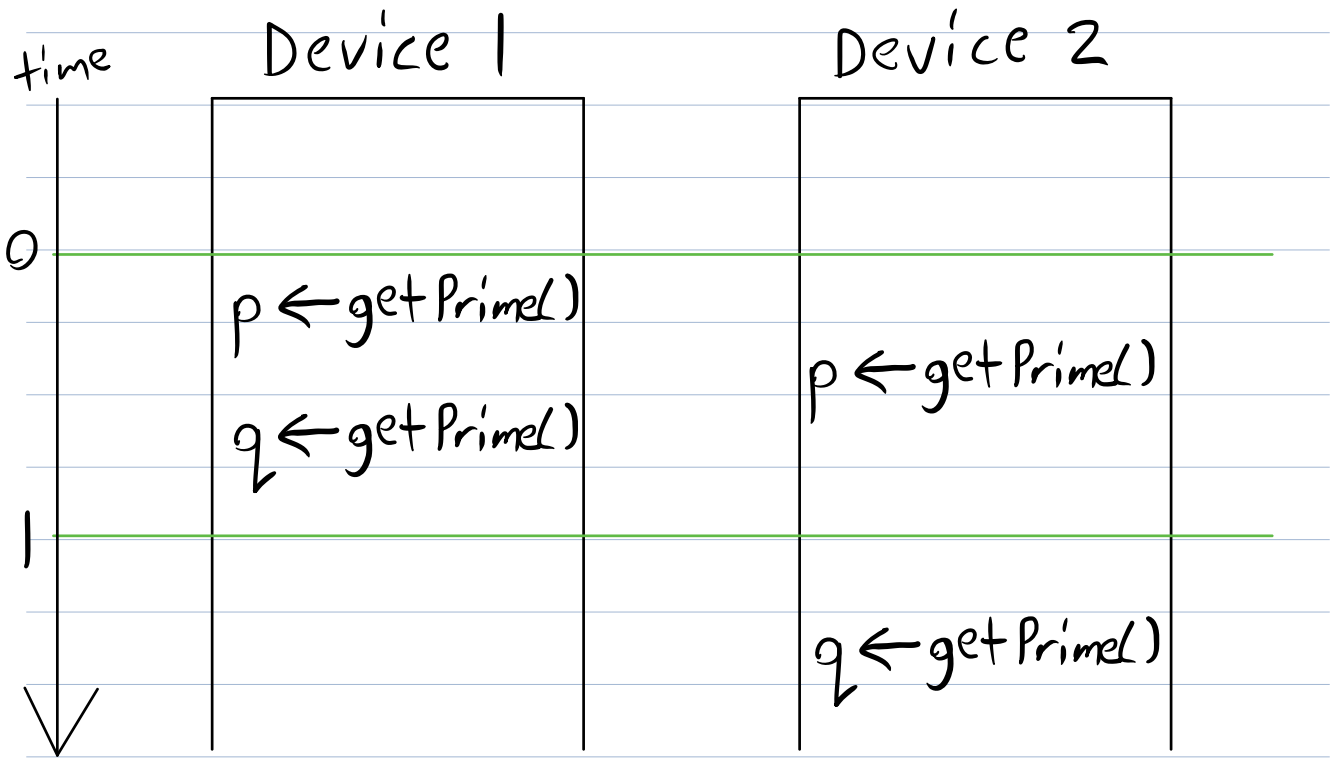
- But embedded devices have no I/O devices!

- on first boot, 2 devices from the same model have the same state ... & then generate an RSA key

Why only one shared prime?

- probably a race condition

```
getPrime( ) {
    x ← Hash(state, time)
    return nextPrime(x)
}
```


time
Device 1
Device 2
0
p ← getPrime()
q ← getPrime()
p ← getPrime()
1
q ← getPrime()

Infineon Attack (2017)

- one of the most shocking cryptographic attacks in years
  - lots of millions of smartcards recalled
    - >50% of Estonian eIDs were vulnerable

- amazing paper (linked online)

Background

- RSA is surprisingly fragile to key leakage
- many variants are broken

} HW2

- Optimizations can easily break security
  - smartcards are "weak" so optimizations are welcome

Standard RSA keyGen: $\lambda \approx 1024$

$$p, q \xleftarrow{R} \{\lambda\text{-bit primes}\}$$

Authors reverse engineered this

Infineon smartcard "optimized" keyGen:

$$p \leftarrow k \cdot M + (65537^a \bmod M)$$
$$q \leftarrow k' \cdot M + (65537^{a'} \bmod M)$$

for random k, k', a, a' so that p, q are prime

M is a public constant (970 bits)

(Flawed) intuition: { k is a random 54-bit #; a is a random > 100-bit # }

$\geq 2^{128}$ choices for p
$\geq 2^{128}$ choices for q
$\} \geq 2^{256}$ choices for N

But, Infineon moduli are actually easy to factor...

Main Tool: "Coppersmith's Attack"

If you know $1/2$ of the bits of p or q, then you can factor $N = pq$ in polynomial time!

So even if there are $> 2^{128}$ choices of p, q some partial info on them is fatal!

Thm: Let $N = pq$ be an RSA modulus w/ $p < q$

Let $f \in \mathbb{Z}_N[X]$ be a polynomial of degree δ

Then, we can find all solutions x_0 of

$$f(x) = 0 \bmod p \text{ s.t. } |x_0| \leq N^{\frac{1}{4\delta}}$$

in time $\text{poly}(\log N, \delta)$

Corollary: $f(x) = 0 \bmod p$ has at most $\text{poly}(\log N, \delta)$ solutions x_0 s.t. $|x_0| \leq N^{\frac{1}{4}}$

Attack Strategy:

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

1) Guess a
2) Use Thm to factor N ← let's do this first

Suppose we guessed a correctly.
How do we factor?

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

$$p = c_1 \cdot k + c_2$$

↑ we want k!

$\Rightarrow$ define $f(x) = c_1 \cdot x + c_2$

1) $f(k) = 0 \pmod p$

2) $\deg(f) = 1$ (f is linear)

3) $|k| \leq N^{1/4}$

↘ for a 2048-bit modulus N, we have

$p = K \cdot M + (\ldots \bmod M) \leq N^{1/2} = 2^{1024}$

$\Rightarrow K \leq \frac{2^{1024}}{2^{970}} = 2^{54} \ll N^{1/4}$

⇒ given a guess "a", recover all solutions k, & try to factor

How do we guess "a"? $p = kM + (65537^a \bmod M)$

equivalently: guess the value of $65537^a \bmod M$

Q: How many different such values are there?

A: order of 65537 in the group $\mathbb{Z}_M^*$

↳ size of the subgroup $\{65537^0, 65537^1, \ldots\}$ (all mod M)

How large is this subgroup? It depends!

- if M is prime ⇒ could be M-1 (the attack would be completely impractical)

- if M is product of small primes $\Rightarrow$ Could be very small

Which one did Infineon use ...? first 126 primes

$M = 2 \cdot 3 \cdot 5 \cdot \ldots$

- if 65537 generates a subgroup of order t in $\mathbb{Z}_M^*$, then there are t guesses for "a"

- Crucial extra trick in paper: "switch" M to M' (w/o knowing the factorization) so that the order of 65537 in $\mathbb{Z}_{M'}$ is minimized
  $\Rightarrow$ fewer guesses

Final Attack: Factor 2048-bit key w/ 2^{34} guesses ($\approx$ 140 CPU years)
- highly parallelizable
- $\approx$ \$40k on AWS

Bonus: Key Fingerprinting

Given an RSA public key, can I test whether it is an Infineon key?

- Original question asked by the same authors in an earlier paper. They showed that many public keys have statistical properties that reveal which device/library generated the key

  - potential privacy concern

Infineon key:

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

(p = first factor, q = second factor)

↳ $N = 65537^{a+a'} \bmod M$

⇒ N is in the subgroup of $\mathbb{Z}_N^*$ generated by 65537

⇒ This would be extremely unlikely if N was generated "normally"

testing if N is of the form $65537^c \bmod M$

$\approx$

Computing discrete log $c = \log_{65537} N \bmod M$

Wait! Isn't discrete log hard?

- Not when M is a product of small primes!!