Attacks on RSA

Tanja Lange

Eindhoven University of Technology

2WF80: Introduction to Cryptology

How does RSA get broken in practice?

How does RSA get broken in practice?

- Too small primes (several TI signing keys had 512 bits.)

How does RSA get broken in practice?

- Too small primes (several TI signing keys had 512 bits.)
- Bad randomness

How does RSA get broken in practice?

- Too small primes (several TI signing keys had 512 bits.)
- Bad randomness
  - Too few primes (Debian RNG failure, 2008)
  - Repeated primes

How does RSA get broken in practice?

- Too small primes (several TI signing keys had 512 bits.)
- Bad randomness
  - Too few primes (Debian RNG failure, 2008)
  - Repeated primes findable by gcd computation (improved version for internet scale) (https://factorable.net/index.html, similar independent result, both 2012).
  - Broken RNG leading to patterns (https://smartfacts.cr.yp.to/, 2013)
- Primes chosen in too few residue classes (Return of Coppersmith (ROCA), 2017)
  This needs more math than we have covered.

There are enough primes for everybody!

The prime-number theorem says that there are about

$$n/\ln(n)$$

primes up to n.

There are enough primes for everybody!

The prime-number theorem says that there are about

$$n/\ln(n)$$

primes up to n.

That means roughly

$$(2^{2048}/\ln(2^{2048})) - (2^{2047}/\ln(2^{2047})) = 1.1377 \cdot 10^{613}$$

primes with 2048 bits.

That's ample for $7.3 \cdot 10^9$ people – even with multiple RSA keys.

There are enough primes for everybody!

The prime-number theorem says that there are about

$$n/\ln(n)$$

primes up to n.

That means roughly

$$(2^{2048}/\ln(2^{2048})) - (2^{2047}/\ln(2^{2047})) = 1.1377 \cdot 10^{613}$$

primes with 2048 bits.

That's ample for $7.3 \cdot 10^9$ people – even with multiple RSA keys.

No chance that two people randomly get the same key.

But finding larger primes takes longer

The prime-number theorem says that there are about

$$n/\ln(n)$$

primes up to n.

But finding larger primes takes longer

The prime-number theorem says that there are about

$$n/\ln(n)$$

primes up to n.

That means each number has a $1/\ln(n)$ chance of being prime. This gets worse for larger numbers.

Roughly

- 354 trials to find a 512-bit prime,
- 710 trials to find a 1024-bit prime,
- 1419 trials to find a 2048-bit prime.

ROCA attack happened because some developer tried to shave of a bit of runtime and went for numbers that are more likely to be prime

But finding larger primes takes longer

The prime-number theorem says that there are about

$$n/\ln(n)$$

primes up to n.

That means each number has a $1/\ln(n)$ chance of being prime. This gets worse for larger numbers.

Roughly

- 354 trials to find a 512-bit prime,
- 710 trials to find a 1024-bit prime,
- 1419 trials to find a 2048-bit prime.

ROCA attack happened because some developer tried to shave of a bit of runtime and went for numbers that are more likely to be prime and thus made the primes findable.

Make sure to sample primes randomly.

Short summary of factorization methods

- For small factors: trial factorization.

Short summary of factorization methods

- For small factors: trial factorization.
- For medium factors: $p - 1$ method (see below), generalization in ECM (using elliptic curves; stay on for 2MMC10), Pollard's rho method (stay on for 2MMC10).
- For RSA numbers: Number field sieve
  - Works by turning hard factorization of one number into many easier factorizations.
  - Uses sieving (think of Eratosthenes) to find small factors.
  - Uses the above to find medium size factors.
  - Also needs a stage of linear algebra at the end.
- The number field sieve has subexponential complexity, so we need to more than double the bit length to make the attack twice as hard.

$p - 1$ method

We know from Fermat's little theorem that

$$a^{p-1} \equiv 1 \bmod p$$

for p prime and $\gcd(a, p) = 1$.

If p is a factor of n then p divides

$$\gcd(a^{p-1} - 1, n).$$

$p - 1$ method

We know from Fermat's little theorem that

$$a^{p-1} \equiv 1 \bmod p$$

for p prime and $\gcd(a, p) = 1$.

If p is a factor of n then p divides

$$\gcd(a^{p-1} - 1, n).$$

To find p, compute $\gcd(a^s - 1, n)$ for s with many small prime factors.

$p-1$ method

We know from Fermat's little theorem that

$$a^{p-1} \equiv 1 \bmod p$$

for p prime and $\gcd(a,p)=1$.

If p is a factor of n then p divides

$$\gcd(a^{p-1}-1, n).$$

To find p, compute $\gcd(a^s-1, n)$ for s with many small prime factors.

Let $s = 232792560 = \text{lcm}\{1,2,3,4,5,\dots,20\}$. Then 2^s-1 is divisible by

- 70 of the 168 primes $\leq 10^3$;
- 156 of the 1229 primes $\leq 10^4$;
- 296 of the 9592 primes $\leq 10^5$;
- 470 of the 78498 primes $\leq 10^6$; etc.

$p - 1$ method in practice

Pick large s with many small factors and random a. Compute

$$a^s \bmod n$$

using fast exponentiation.

$p-1$ method in practice

Pick large s with many small factors and random a. Compute

$$a^s \bmod n$$

using fast exponentiation. The gcd computation reduces modulo n in the first step, so keep numbers small!

$p - 1$ method in practice

Pick large s with many small factors and random a. Compute

$$a^s \bmod n$$

using fast exponentiation. The gcd computation reduces modulo n in the first step, so keep numbers small!

If this fails, increase s or pick a different a.

We could compute $\gcd(a, n)$ but this is unlikely to help.

$p-1$ method in practice

Pick large s with many small factors and random a. Compute

$$a^s \bmod n$$

using fast exponentiation. The gcd computation reduces modulo n in the first step, so keep numbers small!

If this fails, increase s or pick a different a.

We could compute $\gcd(a, n)$ but this is unlikely to help.

"Real" $p-1$ computations have a second phase in which they increase s by larger prime numbers only.