Attacks on the RSA Cryptosystem

Notes by Henry Corrigan-Gibbs

MIT - 6.5610
Lecture 14 (March 22, 2023)

Warning: This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- Recap: The RSA function
- Rabin's function
- Hastad's broadcast attack
- Fault attack
- Small-root attacks

After giving a recap of the RSA function, we will discuss a number of practical attacks that come about from various misuses of the RSA function. Dan Boneh has a nice survey [1] of these attacks and many others.

Recap: RSA Function

Trapdoor one-way permutations

As we discussed last time, RSA implements a *trapdoor one-way permutation* ("trapdoor OWP"). A trapdoor one-way permutation over space $\mathcal{X}$ is a triple of efficient algorithms:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$. The key-generation algorithm takes as input the security parameter $\lambda \in \mathbb{N}$, expressed as a unary string, and outputs a secret key and a public key.
- $F(\mathsf{pk}, x) \to y$. The evaluation algorithm F takes as input the public key pk and an input $x \in \mathcal{X}$, and outputs a value $y \in \mathcal{X}$.
- $I(\mathsf{sk}, y) \to x'$. The inversion algorithm I takes as input the secret key sk and a point $y \in \mathcal{X}$, and outputs its inverse $x' \in \mathcal{X}$.

In the RSA construction, the input space $\mathcal{X}$ depends on the public key, but we elide that technical detail here.

Correctness. For all $\lambda \in \mathbb{N}$, $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$, and $x \in \mathcal{X}$:

$$I(\mathsf{sk}, F(\mathsf{pk}, x)) = x.$$

Security. For all efficient adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr\left[\mathcal{A}(\mathsf{pk}, F(\mathsf{pk}, x)) = x \colon \begin{array}{c} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ x \xleftarrow{R} \mathcal{X} \end{array}\right] \le \text{negl}(\lambda).$$

IMPORTANT*:* Just as a one-way function is only hard to invert on a *randomly sampled input*, a trapdoor one-way function is only hard to invert on a randomly sampled input. Many of the cryptographic failures of RSA that we will see today come from misuses of the RSA function; such as assuming that it is hard to invert on non-random inputs.

Applications. A trapdoor OWP immediately gives schemes for key exchange, public-key encryption, and digital signatures.

RSA construction

Since we went through the RSA construction quickly last lecture, let's recap it here.

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$.
  - Sample two random λ-bit primes p and q ($\lambda \approx 1024$) such that $p \equiv q \equiv 5 \pmod 6$.

  - Set $N \leftarrow p \cdot q$.
  - Output $\mathsf{sk} \leftarrow (p, q)$, and $\mathsf{pk} = N$.
- $F(\mathsf{pk} = N, x \in \mathbb{Z}_N^*) \to y$.
  - Output $y \leftarrow x^3 \bmod N$.
- $I(\mathsf{sk} = N, y \in \mathbb{Z}_N^*) \to x'$.
  - Reduce y modulo p and modulo q.
  - Find a cube root of y modulo p and modulo q. We argued last time that when p and q are congruent to 5 modulo 6, we can compute these cube roots as:

$$x'_p = y^{\frac{p+1}{6}} \pmod{p} \quad \text{and} \quad x'_q = y^{\frac{q+1}{6}} \pmod{q}.$$

  - Reconstruct the solution x' modulo N using the Chinese Remainder Theorem. Concretely, compute $\alpha = x'_p q^{-1} \bmod p$ and $\beta = x'_q p^{-1} \bmod q$. Then

$$x' = \alpha q + \beta p \pmod{N}.$$

  - Output x'.

Notice that $x' \bmod p = \alpha q \bmod p = x'_p q^{-1} q \bmod p = x'_p$. We also have that $x' \bmod q = x_q \bmod q$.
So $x'^3 = y$ modulo p and modulo q, and thus $x'^3 = y \bmod N$.

Why should the RSA function be hard to invert? Ideally we would be able to say that any algorithm for inverting the RSA trapdoor OWP on random inputs gives us an algorithm for factoring. But we have no idea whether that is the case.

When we use RSA, we just make the assumption that RSA is secure—that computing cube roots modulo a composite of unknown factorization is hard:

Definition 1 (RSA Assumption, with public exponent 3)**.** For all efficient adversaries $\mathcal{A}$ there exists a negligible function $\text{negl}(\cdot)$ such that:

$$\Pr\left[\mathcal{A}(N, y) = x : \begin{array}{c} p, q \xleftarrow{R} \mathsf{Primes}_\lambda \\ N \leftarrow p \cdot q \\ x \xleftarrow{R} \mathbb{Z}_N^* \\ y \leftarrow x^3 \bmod N \end{array}\right] \leq \text{negl}(\lambda)$$

Here we use Primes_λ to note the set of λ-bit primes.

Reminder: Greatest common divisor

Recall that, for positive integers a, b, their *greatest common divisor*, $\gcd(a, b)$ is the largest integer that divides both numbers.

Theorem 2 (Euclid, etc.)**.** *There is an efficient algorithm that computes the greatest common divisor of two positive integers.*

Rabin's trapdoor one-way function

Rabin proposed a trapdoor one-way *function* (not a permutation), that is secure if factoring is hard. (In contrast, for all we know, factoring could be hard but inverting the RSA function could be easy.)

Rabin's trapdoor OWF, on RSA modulus $N = p \cdot q$ is just

$$F_{\text{Rabin}}(\mathsf{pk} = N, x \in \mathbb{Z}_N^*) := x^2 \bmod N.$$

So it is just RSA with public exponent $e = 2$.

Inverting Rabin's one-way function is almost exactly as with inverting RSA, except that we now need to compute square roots of $y = x^2$ modulo the prime factors p and q of N. We will not give the argument here, but the security analysis of Rabin encryption shows that computing square roots modulo composites is as hard as factoring.

Why don't we use Rabin encryption and signatures. I think this is just a historical peculiarity. If Rabin's construction had been first, maybe we would have?

When $p \equiv q \equiv 3 \bmod 4$, there are two square roots of y modulo p, and they are:

$$r = \pm y^{\frac{p+1}{4}} \bmod p$$

The argument is essentially as we used for cube roots in the last lecture:

When we write $-a \bmod p$, we mean the integer $p - a$.

$$r^2 = (-r)^2 = y^{\frac{p+1}{2}} = y^{\frac{p-1}{2}} y = (\alpha^2)^{\frac{p-1}{2}} y = y \pmod{p}.$$

This last congruence comes from the fact that when y is a quadratic residue modulo p (i.e., a square in $\mathbb{Z}_p^*$), we can write $y = \alpha^2 \bmod p$, for some $\alpha \in \mathbb{Z}_p^*$.

Inverting Rabin's OWF is as hard as factoring N. This is a beautiful result. An alternative way to state it is that computing square roots modulo composites is as hard as factoring.

We show that, given an efficient algorithm $\mathcal{A}$ for inverting Rabin's function, we can produce an efficient algorithm $\mathcal{B}$ for factoring RSA moduli.

We construct the algorithm $\mathcal{B}(N = p \cdot q)$ as follows:

- Choose a random $x \xleftarrow{\text{R}} \mathbb{Z}_N^*$ and square it: $y \leftarrow x^2 \bmod N$.
- Run $r \leftarrow \mathcal{A}(N, y)$.
- Compute $p' \leftarrow \gcd(N, x \pm r)$.
- If either value of $p' \neq 1$, p' is a factor of N.
  Otherwise output "FAIL."

To explain why this algorithm works, we have that

$$\begin{aligned} r^2 &= x^2 && \pmod N \\ (r+x)(r-x) &= 0 && \pmod N \end{aligned}.$$

So this relation holds modulo each of the prime factors of N:

$$\begin{aligned} (r+x)(r-x) &= 0 \pmod p \\ (r+x)(r-x) &= 0 \pmod q. \end{aligned}$$

Now for every value of x there are four values of r that can satisfy these congruences. In particular, the value r can satisfy:

$$\begin{array}{lllll} r-x=0 \pmod p & \text{and} & r-x=0 \pmod q & \\ r-x=0 \pmod p & \text{and} & r+x=0 \pmod q & (*) \\ r+x=0 \pmod p & \text{and} & r-x=0 \pmod q & (*) \\ r+x=0 \pmod p & \text{and} & r+x=0 \pmod q & \end{array}$$

So there are four possible roots of y modulo N.

One way to understand this is that y has two roots modulo p and two roots modulo q. So there are $2 \cdot 2 = 4$ possible combinations of these two roots modulo N.

When $r - x = 0$ modulo both p and q, then $r = x \bmod N$. When $r + x = 0$ modulo both p and q, then $r = -x \bmod N$. In these cases, our factoring algorithm fails.

But in the two cases marked (*) above, $\gcd(N, r \pm x)$ does reveal a factor of N. Say that $r - x = 0 \bmod p$ and $r + x \neq 0 \bmod q$. Then $(r - x)$ is a multiple of p but not of q, so $\gcd(r - x, N) = p$, and we are done.

To argue that the good case happens with probability 1/2: the algorithm $\mathcal{A}$ has no information about which root x of y algorithm $\mathcal{B}$ picked. So $\mathcal{A}$ will output one of the "useful" roots of x with probability at least $1/2$.

Why is Rabin as hard as factoring while RSA is not necessarily? This is strange. This reduction $\mathcal{B}$ we constructed critically relies on the fact that there are multiple square roots of y modulo N. In contrast, there is a *single* cube root of every $y \in \mathbb{Z}^*N$ modulo N, when we pick p and q as we do in the RSA key-generation process. So the trick we used here to factor N does not work for exactly the reason that the RSA function is a permutation.

Whenever $p = q = 2 \bmod 3$ there will be exactly one cube root of every $y \in \mathbb{Z}^*N$. If we picked p or q to be $1 \bmod 3$, then there could be multiple roots.

We can define other Rabin/RSA variants for which inverting is as hard as factoring: $y = x^4 \bmod N$, for example.

Greatest-common-divisor attack

A few years ago, a group of researchers [2] downloaded all of the public RSA keys they could find on the Internet. They noticed that there were pairs of RSA public keys (N, N') such that

$$N = p \cdot q \quad \text{and} \quad N' = p \cdot q',$$

where p, q, q' are distinct primes.

This typically happens when the RSA key-generation algorithm uses a poor source of randomness. For example, when a network router boots up for the first time, it might generate a keypair immediately after booting. If two devices begin in exactly the same state (as they are on first boot), they could end up generating the same first prime p before their states diverge and they generate distinct second primes $q \neq q'$.

This is more likely to happen on embedded devices, which don't have the usual set of peripherals (disk, mouse, etc.) that standard OSes use as sources of environmental randomness.

One surprising fact is that given (N, N') of this form, anyone can factor both moduli!

If $N = pq$ and $N' = pq'$, we have $\gcd(N, N') = p$, and we can factor both moduli. Implementing the 2012 attack [2] required computing the pairwise greatest common divisors of many millions of public keys. To do so, they used a slightly more involved algorithm, but with this same general idea.

Broadcast attack

Last time we saw how to construct an unauthenticated key-exchange scheme from a trapdoor OWP. Here we show how misuse of RSA-based key exchange can lead to a devastating attack.

Say that Alice wants to establish a shared secret with three other people, who have RSA public keys N_1, N_2, N_3.

Alice might plan to use this shared secret for establishing an encrypted group chat with her three friends.

Let $B = \min\{N_1, N_2, N_3\}$ To do so, Alice samples a random session key $r \xleftarrow{R} \mathbb{Z}_B$ and publishes:

$$\begin{aligned} y_1 &\leftarrow r^3 \pmod{N_1}, && (1)\\ y_2 &\leftarrow r^3 \pmod{N_2}, \text{ and} && (2)\\ y_3 &\leftarrow r^3 \pmod{N_3}. && (3) \end{aligned}$$

So the eavesdropper sees the values (N_1, N_2, N_3) and (y_1, y_2, y_3). We will show that an attacker can efficiently recover the session key r, given only these values.

Why doesn't this attack contradict our assumption that the RSA function is a secure trapdoor OWP? Even if the RSA function is *secure* as a trapdoor OWP, it may still be possible for an eavesdropper to recover the secret session key r! This is so because, the trapdoor OWP definition only requires that the function is hard to invert when a random r is samples, and $F(\mathsf{pk} = N, r) := r^3 \bmod N$ is published. Trapdoor OWP security says nothing about taking the *same* value r and publishing its image under $F(\mathsf{pk}_1, \cdot)$, $F(\mathsf{pk}_2, \cdot)$, and $F(\mathsf{pk}_3, \cdot)$. In fact, it *is* possible to recover r

The attack. Let $M = N_1 N_2 N_3$. By the Chinese Remainder Theorem, from last lecture, (1)–(3), the eavesdropper can efficiently compute the unique integer y, such that $1 \le y \le M$, and such that

$$y = r^3 \pmod{M}.$$

But notice now that since $r < B = \min\{N_1, N_2, N_3\}$, we have that $r^3 < B^3 \le M$. Since r^3 is smaller than the modulus M, we know that the exponentiation doesn't "wrap around" the modulus M. In other words, the following relation holds over the integers:

$$y = r^3 \quad \in \mathbb{Z}.$$

So the eavesdropper can just compute the cube root of y over the integers to recover the secret value r.

Fault Attack

Many low-power devices, such as the chip in your chip-and-PIN credit card, perform RSA signatures using a secret key should be difficult to extract from the device. If an attacker gets physical control of the device, she can heat it up, which increases the rate of errors in intermediate computations.

The "Rowhammer" attack is a way to induce bit flips on a server even without physical access to the machine.

We show that if an attacker can obtain the output of a faulty signature computation, it can recover the signer's secret key.

How an RSA signature should work. To sign a message with an RSA secret key consisting of primes (p, q) and public key $N = pq$, the signer first hashes the message m using a hash function $H \colon \{0,1\}^* \to \mathbb{Z}_N^*$. The signature is then the cube root of $h = H(m) \bmod N$.

To compute the signature, as we showed at the start of this lecture, the attacker computes:

$$\sigma_p \leftarrow h^{1/3} \bmod p \qquad \text{and} \qquad \sigma_q \leftarrow h^{1/3} \bmod q.$$

The then attacker reconstructs the cube root $\sigma \in \mathbb{Z}_N^*$ of h modulo N using the Chinese Remainder Theorem.

The computation of σ_p and σ_q takes many thousands of CPU cycles, since each involves a big-integer modular exponentiation. So, it is absolutely possible for the computation of, say σ_p to be corrupted by a bit flip, while the computation of σ_q is correct.

Assume that this happens, and the signer publishes the resulting faulty signature $\hat{\sigma}$. We show how the attacker can use $\hat{\sigma}$ to factor the

The signer can detect whether a fault has happened by verifying the signature $\hat{\sigma}$ before publishing it and regenerating the signature if it is faulty.

It is conceivable that a second fault could happen during the signer's validity check, though the probability of a second fault occurring in exactly the right spot to cause the signer to miss a faulty signature is exceedingly small.

modulus N.

Observe that:

$$\begin{aligned} \hat{\sigma} &\neq H(m)^3 \pmod{p} &\implies&& \hat{\sigma} - H(m)^3 &\neq 0 \pmod{p} \\ \hat{\sigma} &= H(m)^3 \pmod{q} &\implies&& \hat{\sigma} - H(m)^3 &= 0 \pmod{q} \end{aligned}$$

So $\hat{\sigma}$ is a multiple of q, since it is zero modulo q, but is *not* a multiple of p, since it is non-zero modulo p. Therefore, we can write $\hat{\sigma} = k \cdot q$, where $k \in \mathbb{Z}$ is a positive integer that does not have p as a divisor. This implies that:

$$\gcd(N, \hat{\sigma}) = q,$$

so using the Euclidean algorithm we can recover a factor of N and we are done.

Finding small roots of polynomials

There is a wide class of very powerful attacks on RSA-style cryptosystems that come from "lattice-reduction" techniques.

This theorem, from a very nice survey of May [3], summarizes one of the main results in the area:

We require that f is has leading coefficient 1, but this is essentially without loss of generality for our applications since either the leading coefficient is invertible modulo N (in which case we can divide it off) or its not (in which case we have learned a factor of the modulus N).

Theorem 3 (Coppersmith, Howgrave-Graham, May, …)**.** *Let p be a divisor of N such that $p \geq N^\beta$ for some constant $0 < \beta \leq 1$. Let f be a polynomial of degree δ with integral coefficients. Then there is an efficient algorithm that outputs all integers x such that*

$$f(x) = 0 \bmod p \qquad \textit{and} \qquad |x| \leq N^{\beta^2/\delta}.$$

Here $|x| \leq B$ indicates that $x \in \{-B, \dots, -1, 0, 1, \dots, B\}$.

Factoring with bits known

As an example application of Theorem 3, imagine that we are able to learn the high-order 2/3 of the bits of a prime factor p of an RSA modulus $N = pq$. It is not immediately clear how to use these bits to factor N—a brute-force attack on the remaining bits of p would take exponential time.

For example, someone might be able to extract these bits off of a discarded hard drive they find in the trash.

Say that the $\lambda = \log_2 p$ and let $C \in \{0, \dots, 2^{2\lambda/3}\}$ be an integer representing our $2\lambda/3$ known bits. Then write

$$f(x) = 2^{\lambda/3} \cdot C + x \pmod{p}.$$

We know that there exists an $x_0 \in \mathbb{Z}$ with $|x_0| < p^{1/2} \leq N^{1/4}$ such that $f(x_0) \equiv= 0 \bmod p$.

If we can find such an x_0, we know that $f(x_0) \in \mathbb{Z}$ gives a multiple of N we can compute a factor of N as $\gcd(N, f(x_0))$. To do so, apply Theorem 3 with $\beta = \frac{1}{2} - \frac{1}{1000}$ and $\delta = 1$.

If we want to factor with the low-order $2\lambda/3$ bits known, we can apply Theorem 3 with the polynomial:

$$f(x) = 2^{2\lambda/3}x + C \pmod{p}.$$

Recovering small preimages

When we defined the RSA function, we required sampling the input x from the full domain $\mathbb{Z}_N^*$. But we could also consider variants in which x is sampled from some smaller subset of $\mathbb{Z}_N^*$. We show that if the subset is too small, the RSA function is easy to invert.

In particular, say that we sample $m \leftarrow \{0, \dots, N^{1/5}\}$ and compute

This type of attack explains why implementers often use RSA with public exponent $e = 2^{16} + 1$, instead of $e = 3$.

$$\begin{aligned} m &\leftarrow N^{1/5} \cdot C + x \\ y &\leftarrow m^3 \pmod{N}, \end{aligned}$$

where $C \in \{0, \dots, N^{4/5}\}$ is a public constant. The attacker gets (N, y) and wants to recover m.

Define $f(x) = (N^{1/5} \cdot C + x)^3$. Then any solution $f(x_0) = 0$ over the integers reveals the preimage of y under this modified RSA function. The attacker can again use Theorem 3, with $\beta = 1$ and $\delta = 3$.

The Infineon bug

Infineon is a vendor of smartcards used in many types of computer systems—for electronic ID cards, for hardware security modules in laptops and phones, for payment cards, and so on. Infineon cards use the RSA trapdoor OWP for signing and decryption.

A few years ago, some researchers discovered a critical flaw in the way Infineon devices generate RSA keys [4]. The fallout was widespread: many laptop vendors pushed firmware updates and the government of Estonia temporarily suspended the use of over 700k national ID cards.

What happened? Infineon engineers implemented an "optimized" RSA key-generation procedure that involved sampling primes p and q of a special form:

$$\begin{aligned} p &= a \cdot M + (65537^b \bmod M) \\ q &= a' \cdot M + (65537^{b'} \bmod M), \end{aligned}$$

where M is a public 970-bit constant, and a, b, a', b' are secret numbers roughly 128 bits long.

Notice that there are more than 2^{256} possible values of p and q, so a brute-force attack at guessing a and b is infeasible.

As we know from Theorem 3, if you are given a modulus $N = pq$, constructed with primes of this form, along with the value b, you can factor the modulus. (This is again the problem of factoring with bits known.) But how do we find the value b?

The value $65537^b \bmod M$ lies in the set:

$$\{65537^0 \bmod M, 65537^1 \bmod M, 65537^2 \bmod M, \dots\}.$$

How large can this set be? It is equal to the *order* of 65537 modulo M; its size on average depends on the factorization of M. The more factors M has, the smaller the order will be on average. Unfortunately for Infineon, they picked $M = 2 \cdot 3 \cdot 5 \cdot 7 \cdots$, which means that the order is likely to be small. In fact the order of 65537 for Infineon's choice of parameters was $\approx 2^{55}$—a feasible search space for a determined attacker.

The designers of the attack use additional cleverness to reduce the attack cost even further by replacing M with a different value M' such that 65537 has even smaller order modulo M'.

References

[1] Dan Boneh et al. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[2] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *USENIX Security*, 2012.

[3] Alexander May. Using LLL-reduction for solving RSA and factorization problems. In *The LLL Algorithm: Survey and Applications*, pages 315–348. Springer, 2009.

[4] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. The return of Coppersmith's attack: Practical factorization of widely used RSA moduli. In *ACM Conference on Computer and Communications Security*, 2017.