

RSA: Security Considerations (1)

- At a high-level, it is enough to know that the security of RSA is based on factoring ...
 - If you have an efficient method to factor N , then you can recover p and q (and hence $\phi(N)$): this implies you can decrypt and encrypt freely.
 - The opposite implication doesn't hold: "breaking" RSA doesn't mean you necessarily have to have an efficient factoring method.
- ... but doesn't reduce to factoring: they aren't equivalent.
- Question: what other mathematics-oriented attacks on RSA are there ?
- Answer: at least
 - using knowledge of d or $\phi(N)$ to factor N ,
 - problems based on use of a shared modulus,
 - problems based on use of small exponents, and
 - using the "deterministic" nature of RSA encryption.

Attack #1: "Indirect" Factoring (1)

Theorem (Lagrange + Euler)
 If G is a group of size $|G| = n$, then for all $x \in G$ we have $x^n = 1$.
 This implies that $x^{\phi(N)} \equiv 1 \pmod{N}$
 for all $x \in \mathbb{Z}_N^*$ since $|\mathbb{Z}_N^*| = \phi(N)$.

Theorem (Fermat)
 If one is tasked with factoring n and can write $n = x^2 - y^2$
 i.e., write n as the difference of two squares, then since $n = (x + y) \cdot (x - y)$
 one has broken n into two smaller factors by doing so.

Attack #1: "Indirect" Factoring (2)

- Imagine you somehow know (N, d) , the private key associated with a public key (N, e) ; since

$$e \cdot d \equiv 1 \pmod{\phi(N)}$$

it must be the case that for some s

$$\begin{aligned} e \cdot d - 1 &\equiv 0 \pmod{\phi(N)} \\ e \cdot d - 1 &= s \cdot (\rho - 1) \cdot (q - 1) \end{aligned}$$

- Therefore
 - $e \cdot d - 1$ is even, since $\rho - 1$ and $q - 1$ are even, and
 - for any $x \neq 0$

$$x^{e \cdot d - 1} \equiv 1 \pmod{N}$$

by the theorem of Lagrange.

- Since $(e \cdot d - 1)/2$ is an integer, we can pick any x and compute

$$y_0 = x^{(e \cdot d - 1)/2} = \sqrt{x^{e \cdot d - 1}} \pmod{N}.$$

Attack #1: "Indirect" Factoring (3)

- Then, since

$$\begin{aligned} x^{e \cdot d - 1} &\equiv 1 \pmod{N} \\ x^{((e \cdot d - 1)/2)} &\equiv 1 \pmod{N} \\ y_0^2 &\equiv 1 \pmod{N} \\ y_0^2 - 1 &\equiv 0 \pmod{N} \end{aligned}$$

computing

$$\gcd(y_0 - 1, N)$$

yields a factor of N iff. $y_0 \not\equiv \pm 1 \pmod{N}$.

- What if we fail, i.e., $y_0 \equiv \pm 1 \pmod{N}$?

- If $y_0 \equiv -1 \pmod{N}$ then pick another x and try again.
- If $y_0 \equiv +1 \pmod{N}$ then for $t > 0$, define

$$y_t = x^{(e \cdot d - 1)/2^{t+1}} = \sqrt{y_{t-1}} \pmod{N}$$

and try again, i.e., as long as $y_{t-1} \equiv \pm 1$ is even we can take another square root and try again.

- Moral: given any private key d , it is feasible to factor the N associated with it.

Attack #1: "Indirect" Factoring (4)

Example

p	=	5547523367 ₍₁₀₎
q	=	3114435011 ₍₁₀₎
N	= $p \cdot q$	= 17277400998525402037 ₍₁₀₎
$\phi(N)$	= $(p - 1) \cdot (q - 1)$	= 172774009989863443660 ₍₁₀₎
e	=	8957 ₍₁₀₎
d	= $e^{-1} \pmod{\phi(N)}$	= 13483201174404987293 ₍₁₀₎
x	=	4208378838 ₍₁₀₎
y_0	= $x^{(e \cdot d - 1)/2^{0+1}} \pmod{N}$	= 1 ₍₁₀₎
y_1	= $x^{(e \cdot d - 1)/2^{1+1}} \pmod{N}$	= 1 ₍₁₀₎
y_2	= $x^{(e \cdot d - 1)/2^{2+1}} \pmod{N}$	= 1 ₍₁₀₎
y_3	= $x^{(e \cdot d - 1)/2^{3+1}} \pmod{N}$	= 13091272773695815080 ₍₁₀₎
$\gcd(y_3 - 1, N)$	=	3114435011 ₍₁₀₎ = q

Attack #1: "Indirect" Factoring (5)

- Recalling that

$$\phi(N) = (\rho - 1) \cdot (q - 1),$$

imagine we write

$$\begin{aligned} f(x) &= (x - p) \cdot (x - q) \\ &= x^2 - x \cdot q - x \cdot p + p \cdot q \\ &= x^2 - x \cdot (\rho + q) + N \\ &= x^2 - x \cdot s + N \end{aligned}$$

where clearly we know that

$$s = \rho + q = N + 1 - \phi(N).$$

- So if we somehow know $\phi(N)$ and can compute s , we can try to recover $\rho + q$, i.e., factor N , ...

Attack #1: "Indirect" Factoring (6)

- ... by solving for $f(x) = 0$ to get

$$\begin{aligned} \rho &= \frac{s + \sqrt{s^2 - 4 \cdot N}}{2} \\ q &= \frac{s - \sqrt{s^2 - 4 \cdot N}}{2} \end{aligned}$$

- Moral: given $\phi(N)$, it is feasible to factor N ; if you have $\phi(N)$ however, this is already quite bad !

Attack #1: "Indirect" Factoring (7)

Example

p	=	5547523367 ₍₁₀₎
q	=	3114435011 ₍₁₀₎
N	= $p \cdot q$	= 17277400998525402037 ₍₁₀₎
$\phi(N)$	= $(p - 1) \cdot (q - 1)$	= 172774009989863443660 ₍₁₀₎
e	=	8957 ₍₁₀₎
d	= $e^{-1} \pmod{\phi(N)}$	= 13483201174404987293 ₍₁₀₎
s	= $N + 1 - \phi(N)$	= 8661958378 ₍₁₀₎
ρ'	= $\frac{s + \sqrt{s^2 - 4 \cdot N}}{2}$	= 5547523367 ₍₁₀₎ = p
q'	= $\frac{s - \sqrt{s^2 - 4 \cdot N}}{2}$	= 3114435011 ₍₁₀₎ = q

Attack #2: Shared Modulus (1)

- Imagine an example scenario where a server communicates with a number of clients; the i -th client uses the public key

$$(N, e_i),$$

and holds the private key

$$(N, d_i).$$

- Now the i -th client can proceed as follows ...

- Start by computing ρ and q from N since the client knows d_i as part of the private key it holds.
- Next compute $\phi(N) = p \cdot q$ since it now knows ρ and q .
- Finally, for any $j \neq i$ it can now compute

$$d_j = e_j^{-1} \pmod{\phi(N)}.$$

- ... which basically means any client can recover the private key of any other client !

Attack #2: Shared Modulus (2)

- Now imagine there is an adversary which **doesn't** share N , i.e., N and each e_i are known since they are public but it doesn't know **any** d_i .
- The i -th client sends the **same** plaintext message to clients j and k , i.e., computes the ciphertexts

$$\begin{aligned} C_j &= M^{e_j} \pmod{N} \\ C_k &= M^{e_k} \pmod{N} \end{aligned}$$

and communicates them ...

- ... whereby the adversary intercepts them, computes

$$\begin{aligned} t_j &= e_j^{-1} \pmod{e_k} \\ t_k &= (t_j \cdot e_j - 1) / e_k \end{aligned}$$

and can then recover M as

$$\begin{aligned} C_j^{t_j} \cdot C_k^{-t_k} &= M^{e_j \cdot t_j} \cdot M^{-e_k \cdot t_j} \pmod{N} \\ &= M^{e_k \cdot t_k + 1} \cdot M^{-e_k \cdot t_j} \pmod{N} \\ &= M^{e_k \cdot t_k + 1 - e_k \cdot t_j} \pmod{N} \\ &= M^1 \pmod{N} \\ &= M \pmod{N} \end{aligned}$$

Attack #2: Shared Modulus (3)

Example

p	=	5547523367 ₍₁₀₎
q	=	3114435011 ₍₁₀₎
N	=	$p \cdot q = 17277400998525402037$ ₍₁₀₎
$\phi(N)$	=	$(p-1) \cdot (q-1) = 1727740099863443660$ ₍₁₀₎
e_1	=	8957 ₍₁₀₎
e_2	=	3559 ₍₁₀₎
e_3	=	7297 ₍₁₀₎
M	=	14523729606508473667 ₍₁₀₎
$C_1 = M^{e_1} \pmod{N}$	=	6218826128113818226 ₍₁₀₎
$C_2 = M^{e_2} \pmod{N}$	=	1937603238466376862 ₍₁₀₎
$t_1 = e_1^{-1} \pmod{e_2}$	=	693 ₍₁₀₎
$t_2 = (t_1 \cdot e_1 - 1) / e_3$	=	338 ₍₁₀₎
$M' = C_1^{t_1} \cdot C_2^{-t_2}$	=	14523729606508473667 ₍₁₀₎ = M

Attack #3: Small Exponent (1)

- Clearly having a small private key d is a bad idea; this allows adversary to employ brute force search (or the continued fractions attack of Wiener) ...
- ... but a small public key e is a popular way to improve performance.
- Imagine an example scenario where a **server** communicates with a number of **clients**; the i -th client uses the public key

$$(N, e_i),$$

where $e_i = 3$ for all i , and holds the private key

$$(N, d).$$

Attack #3: Small Exponent (2)

- The server sends the **same** plaintext message to clients i, j and k , i.e., computes the ciphertexts

$$\begin{aligned} C_i &= M^{e_i} = M^3 \pmod{N_i} \\ C_j &= M^{e_j} = M^3 \pmod{N_j} \\ C_k &= M^{e_k} = M^3 \pmod{N_k} \end{aligned}$$

and communicates them ...

- ... whereby the adversary intercepts them, computes a solution to

$$\begin{aligned} x &\equiv C_i \pmod{N_i} \\ x &\equiv C_j \pmod{N_j} \\ x &\equiv C_k \pmod{N_k} \end{aligned}$$

via CRT which means

$$x = M^3 \pmod{N_i \cdot N_j \cdot N_k}.$$

- Since $M^3 < N_i \cdot N_j \cdot N_k$, what the adversary actually gets is $x = M^3$ and hence can recover $M = \sqrt[3]{x}$ over the integers!

Attack #3: Small Exponent (3)

Example

p_0	=	5547523367 ₍₁₀₎
q_0	=	3114435011 ₍₁₀₎
N_0	=	$p_0 \cdot q_0 = 17277400998525402037$ ₍₁₀₎
$\phi(N_0)$	=	$(p_0-1) \cdot (q_0-1) = 1727740099863443660$ ₍₁₀₎
p_1	=	1372787921 ₍₁₀₎
q_1	=	450588137 ₍₁₀₎
N_1	=	$p_1 \cdot q_1 = 618561951819493177$ ₍₁₀₎
$\phi(N_1)$	=	$(p_1-1) \cdot (q_1-1) = 618561949996117120$ ₍₁₀₎
p_2	=	1829702921 ₍₁₀₎
q_2	=	3038374439 ₍₁₀₎
N_2	=	$p_2 \cdot q_2 = 5559322586130036319$ ₍₁₀₎
$\phi(N_2)$	=	$(p_2-1) \cdot (q_2-1) = 5559322581261959960$ ₍₁₀₎
M	=	1859787870 ₍₁₀₎
$C_0 = M^3 \pmod{N_0}$	=	3779752900690703226 ₍₁₀₎
$C_1 = M^3 \pmod{N_1}$	=	8422800463817478 ₍₁₀₎
$C_2 = M^3 \pmod{N_2}$	=	1732610277159946802 ₍₁₀₎
$X = \text{crt}((C_0, C_1, C_2), (N_0, N_1, N_2))$	=	6432654596241638235089403000 ₍₁₀₎
$M' = \sqrt[3]{X}$	=	1859787870 ₍₁₀₎ = M

Attack #4: Deterministic Encryption (1)

- From a security perspective, "textbook" RSA has some problems ...

 - It isn't **semantically secure**: two ciphertext messages C_1 and C_2 can be distinguished from each other.
 - It isn't **plaintext aware**: any random ciphertext $C \in \mathbb{Z}_N^*$ is valid; an adversary doesn't need to know the corresponding plaintext.
 - It is **malleable**: given one ciphertext

$$C_1 = M_1^e,$$

an adversary can make some valid $C_2 \neq C_1$ where C_2 and C_1 are "linked".

 - ... which we solve using a **padding scheme** such as **OAEP**.

Attack #4: Deterministic Encryption (2)

Example

Imagine A sends a message "A owes B \$100" encrypted under the public key (N, e) owned by B. To save space, the amount is the only thing sent, i.e.,

$$C_1 = 100^e \pmod{N}.$$

Due to the malleable (in particular the **homomorphic** property) of RSA encryption, B can compute

$$\begin{aligned} C_2 &= 2^e \cdot C_1 \pmod{N} \\ &= 2^e \cdot 100^e \pmod{N} \\ &= (2 \cdot 100)^e \pmod{N} \\ &= 200^e \pmod{N} \end{aligned}$$

and double the amount owed by A without knowing the plaintext!

Example

Imagine A is a bank robber planning a heist, and B is a police officer who has A under surveillance. B knows A will send either

$$\begin{aligned} M_1 &= \text{"proceed"} \\ M_2 &= \text{"abort"} \end{aligned}$$

to a henchman, encrypted as

$$\begin{aligned} C_1 &= M_1^e \\ C_2 &= M_2^e \end{aligned}$$

using the public key (N, e) . Due to the distinguishable (in particular non-randomised) nature of RSA encryption, B can intercept the message, compute

$$C_3 = M_3^e \pmod{N}$$

from $M_3 = \text{"proceed"}$, and then simply test if $C_3 = C_1$ or $C_3 = C_2$!

Conclusions

Further Reading

- D. Boneh. Twenty years of attacks on the RSA cryptosystem. In *Notices of the American Mathematical Society*, 46 (2), 203–213, 1999.
- M.J. Hinek. *Cryptanalysis of RSA and Its Variants*. Chapman & Hall, 2009. ISBN: 1-420-07518-7.