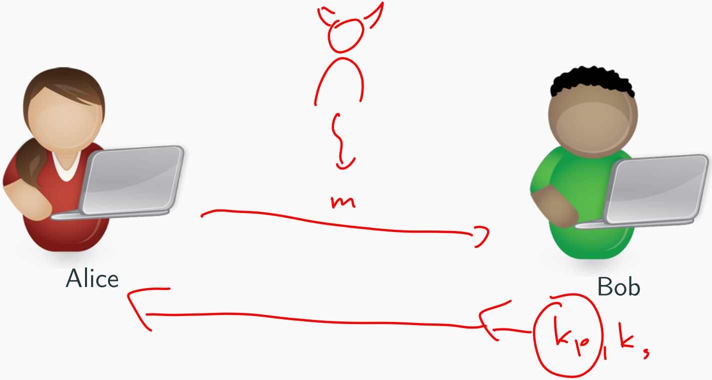
Public-Key Encryption I

Basics, RSA

Problem setup



History I

"There are many cases where we can easily and infallibly do a certain thing but may have much trouble in undoing it. . .

...Can the reader say what two numbers multiplied together will produce the number 8616460799? I think it unlikely that anyone but myself will ever know."

–William Stanley Jeavons

The Principles of Science (1874)

History II

introduced idea of PK crypto

- Ellis
- Cocks
- Williamson

- Rivest
- Shamir
- Adleman
- Diffie
- Hellman

History II (cont'd)

- James H. Ellis
- Clifford Cocks
- Malcolm J. Williamson

- Ron Rivest
- Adi Shamir
- Leonard Adleman
- Whitfield Diffie
- Martin Hellman

History II (cont'd)

PK idea

- James H. Ellis
- Clifford Cocks
- Malcolm J. Williamson

GCHQ (UK)

- Ron Rivest
- Adi Shamir
- Leonard Adleman
- Whitfield Diffie
- Martin Hellman

Academia

History III

"An ingenious scheme intended for the encipherment of speech over short metallic connections was proposed by Bell Telephone Laboratories (Ref. 1) in which the recipient adds noise to the line over which he receives the signal. If this noise is sufficiently large compared with the message it can effectively disguise it. The recipient however can subtract the noise from the signal he receives and so obtain the original message."

–James H. Ellis

The Possibility of Secure Non-Secret Digital Encryption (1970, classified GCHQ report)

History III

"An ingenious scheme intended for the encipherment of speech over short metallic connections was proposed by Bell Telephone Laboratories (Ref. 1) in which the recipient adds noise to the line over which he receives the signal. If this noise is sufficiently large compared with the message it can effectively disguise it. The recipient however can subtract the noise from the signal he receives and so obtain the original message."

–James H. Ellis

The Possibility of Secure Non-Secret Digital Encryption (1970, classified GCHQ report)

"Prototype RSA" first implemented at GCHQ by Clifford Cocks in 1973. (Public discovery: Rivest, Shamir, Adleman in 1978)

"Prototype Diffie-Hellman key exchange" implemented by Malcolm J. Williamson at GCHQ im 1974. (Public discovery: Diffie, Hellman in 1976).

History does not follow a straight line. . .

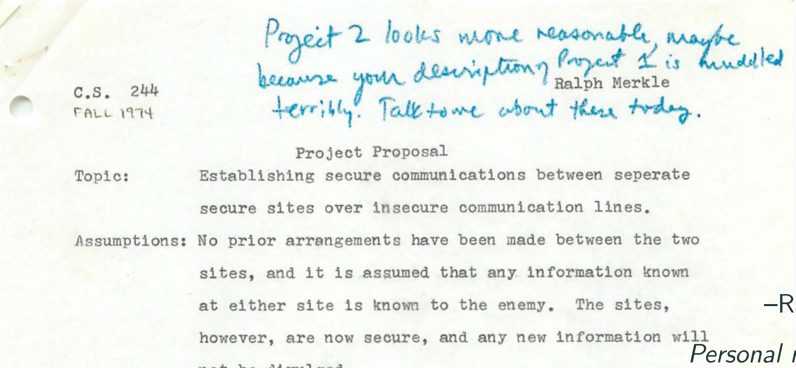
"In the Fall of 1974, as an undergraduate, I enrolled in CS244, the Computer Security course offered at UC Berkeley and taught by Lance Hoffman. We were required to submit two project proposals, one of which we would complete for the course. I submitted a proposal for what would eventually become known as Public Key Cryptography – which Hoffman rejected. I dropped the course, but kept working on the idea."

–Ralph Merkle

Personal recollections

History does not follow a straight line. . .

"In the Fall of 1974, as an undergraduate, I enrolled in CS244, the Computer Security course offered at UC Berkeley and taught by Lance Hoffman. We were required to submit two project proposals, one of which we would complete for the course. I submitted a proposal for what would eventually become known as Public Key Cryptography – which Hoffman rejected. I dropped the course, but kept working on the idea."



Project 2 looks more reasonable, maybe because your description of Project 1 is muddled terribly. Talk to me about these today.

C.S. 244
FALL 1974

Ralph Merkle

Project Proposal

Topic: Establishing secure communications between seperate secure sites over insecure communication lines.

Assumptions: No prior arrangements have been made between the two sites, and it is assumed that any information known at either site is known to the enemy. The sites, however, are now secure, and any new information will

–Ralph Merkle

Personal recollections

History does not follow a straight line. . .

"In the Fall of 1974, as an undergraduate, I enrolled in CS244, the Computer Security course offered at UC Berkeley and taught by Lance Hoffman. We were required to submit two project proposals, one of which we would complete for the course. I submitted a proposal for what would eventually become known as Public Key Cryptography – which Hoffman rejected. I dropped the course, but kept working on the idea."

"I showed an early draft to Bob Fabry, then on the faculty at Berkeley, who. . . said "Publish it, win fame and fortune!". . . As I was to learn, Fabry's response was rare. . . . sent my submitted paper out for review and received the following response from an "experienced cryptography expert" whose identity is unknown to this day:"

"I am sorry to have to inform you that the paper is not in the main stream of present cryptography thinking and I would not recommend that it be published in the Communications of the ACM. Experience shows that it is extremely dangerous to transmit key information in the clear."

–Ralph Merkle

Personal recollections

Public-key encryption

Definition 1: PK encryption system

A public-key encryption system (or simply an asymmetric cipher) over $(\mathcal{K}_p, \mathcal{K}_s, \mathcal{M}, \mathcal{C})$ is a tuple $\mathcal{E} = (Gen, E, D)$, where:

- Gen is an input-less randomized algorithm which produces a pair of keys $(k_p, k_s) \in \mathcal{K}_p \times \mathcal{K}_s$. Possible outputs of Gen are called valid keys for $\mathcal{E}$, and we denote by $\mathcal{V}(\mathcal{E}) \subseteq \mathcal{K}_p \times \mathcal{K}_s$ the set of all $\mathcal{E}$'s valid keys;
- $E\colon \mathcal{K}_p \times \mathcal{M} \to \mathcal{C}$ is an encryption algorithm, and
- $D\colon \mathcal{K}_s \times \mathcal{C} \to \mathcal{M}$ is a decryption algorithm s.t.

$$\forall m \in \mathcal{M}\ \forall (k_p, k_s) \in \mathcal{V}(\mathcal{E}) : \quad D(k_s, E(k_p, m)) = m.$$

Trapdoor functions

It is evident that at the heart of an asymmetric cipher $\mathcal{E}$ there must be a trapdoor function: a function $f : \mathcal{K}_p \times \mathcal{M} \to \mathcal{C}$ such that:

1. There exist two efficient (polynomial) algorithms *Fwd* and *Inv* satisfying:
   - given $m \in \mathcal{M}$ and $k_p \in \mathcal{K}_p$, the algorithm *Fwd* computes $f(k_p, m)$, and
   - given $c \in \mathcal{C}$ and $k_s \in \mathcal{K}_s$, the algorithm *Inv* computes an element of $\mathcal{M}$ s.t. for all $m \in \mathcal{M}, (k_p, k_s) \in \mathcal{V}(\mathcal{E})$ it holds

$$Inv(k_s, f(k_p, m)) = m$$

2. For any $(k_p, k_s) \in \mathcal{V}(\mathcal{E})$, if we do not know k_s, it is intractable to compute, for any $c \in \mathcal{C}$, an element $m \in \mathcal{M}$ s.t. $f(k_p, m) = c$.

Trapdoor functions in modular arithmetic

Let's consider $\mathcal{M} = \mathcal{C} = \mathcal{K}_p = \mathcal{K}_s = \mathbb{Z}_N^\times$. The earliest (and today still utilized) asymmetric ciphers used one of the following FWD functions:

- $f(k_p, m) = m^{k_p} \pmod N$ $\longrightarrow$ c $\quad \sqrt[k_p]{c} \pmod N$ $\longleftarrow$
- $f(k_p, m) = {k_p}^m \pmod N$ – turning this into true trapdoor more intricate

$c \qquad \log_{k_p}(c) = m \pmod N$

Towards RSA

Idea: for a good choice of N one can devise encryption and decryption exponents $1 \leq e, d \leq N$ s.t.

- for all $m \in \mathbb{Z}_N^\times$ $(m^e)^d \equiv m \pmod{N}$, and
- given just N and e, it is computationally hard to recover d (and thus, as we shall see, also m)

In the following couple of slides:

- We will infer the construction of e, d for N prime (the simplest case).
- We will show that choosing N prime is not secure.
- We will show how to choose N securely.
- All of this we will do using three simple insights from group theory.

Roots of integers modulo prime

Euler's totient function

Let us consider the group $\mathbb{Z}_p^\times$ for p prime. Note that $\underline{|\mathbb{Z}_p^\times|} = \underline{\varphi(p)} =$

Roots of integers modulo prime

Let us consider the group $\mathbb{Z}_p^\times$ for p prime. Note that $|\mathbb{Z}_p^\times| = \varphi(p) = p - 1$.

$m^k \equiv m^{(k \bmod 16)}$

We want to design e, d such that for all m it holds $(m^e)^d \equiv m \pmod{p}$.

$(m^e)^d \equiv m \pmod{\cancel{p}\ N}$

$m^{e\cdot d} \equiv m \pmod{\cancel{p}\ N} \quad / m^{-1}$

$m^{e\cdot d-1} \equiv \underbrace{1}_{m^0} \pmod{\cancel{p}\ N}$

$e\cdot d - 1 \equiv 0 \pmod{\varphi(N) = \cancel{p-1}}$

$e \cdot d \equiv 1 \pmod{\cancel{p-1}\ \varphi(N)}$

→ d must be a multiplicative inverse of $e \pmod{\cancel{p-1}\ \varphi(N)}$

$e \in \mathbb{Z}^\times_{\varphi(N)}$, put $d = e^{-1} \pmod{\cancel{p-1}\ \varphi(N)}$

e, N public ...

N needs to be composite!

Composite moduli

For composite moduli, we can use the same construction as in the previous slide, with $p-1$ replaced by $\varphi(N)$. We need to be able to compute $\varphi(N)$ so as to set up e, d (more on that later).

I.e., we want $\varphi(N) \mid e \cdot d - 1$, i.e. $e \cdot d \equiv 1 \pmod{\varphi(N)}$. Hence, we need to choose e coprime to $\varphi(N)$ and let d be its multiplicative inversion modulo $\varphi(N)$.

Note: knowledge of N, e is required for encryption (computing $m^e \pmod{N}$), but to recover d, the adversary seems to require the knowledge of $\varphi(N)$. How to choose N so that it is difficult to compute $\varphi(N)$?

$\varphi(N)$ of composite N

Theorem 1: Prime factorization theorem

Each integer $N \geq 2$ can be written as

$$N = p_1^{n_1} \cdot p_2^{n_2} \cdot \dots \cdot p_k^{n_k},$$

where $p_1, \dots, p_k$ are pairwise distinct primes and all the n_i are positive. Moreover, N can be written in only one such way, up to re-ordering of the terms.

Theorem 2: Product formula

Let N have a prime factorization $p_1^{n_1} \cdot p_2^{n_2} \cdot \dots \cdot p_k^{n_k}$. Then

$$\varphi(N) = p_1^{n_1-1}(p_1 - 1)p_2^{n_2-1}(p_2 - 1) \cdots p_k^{n_k-1}(p_k - 1)$$

The most straightforward way of computing $\varphi(N)$ is to factor N. (Later we will prove that there cannot exist a more efficient way.) But integer factoring is widely regarded as a computationally hard problem $\Rightarrow$ trapdoor!

(Vanilla) RSA

In RSA, the key generation proceeds as follows:

- Two prime numbers p, q of roughly the same bitlength and satisfying some additional conditions are generated randomly. Then, we compute $N = p \cdot q$.
- We compute a number e co-prime to $\varphi(N) = (p-1) \cdot (q-1)$ and its multiplicative inversion d in $\mathbb{Z}^{\times}_{\varphi(N)}$ (i.e., $e \cdot d \equiv 1 \pmod{(p-1) \cdot (q-1)}$).
- We output $k_p = (N, e)$ and $k_s = (N, d)$ (by convention we write just $k_s = d$).

The RSA trapdoor function works as follows:

- $Fwd((N, e), m) = m^e \pmod{N}$ ("encryption")
- $Inv(d, c) = c^d \pmod{N}$ ("decryption").

$(m^e)^d = m$

(Vanilla) RSA

In RSA, the key generation proceeds as follows:

- Two prime numbers p, q of roughly the same bitlength and satisfying some additional conditions are generated randomly. Then, we compute $N = p \cdot q$.
- We compute a number e co-prime to $\varphi(N) = (p-1) \cdot (q-1)$ and its multiplicative inversion d in $\mathbb{Z}^{\times}_{\varphi(N)}$ (i.e., $e \cdot d \equiv 1 \pmod{(p-1) \cdot (q-1)}$).
- We output $k_p = (N, e)$ and $k_s = (N, d)$ (by convention we write just $k_s = d$).

$\sqrt[e]{c}$

The RSA trapdoor function works as follows:

- $Fwd((N, e), m) = m^e \pmod{N}$ ("encryption")
- $Inv(d, c) = c^d \pmod{N}$ ("decryption").

$$\frac{m^e,\ e,\ N}{m}$$

However, this "vanilla" (or "textbook") RSA should never be used directly for encryption $\Rightarrow$ insecure!

Is RSA indeed a trapdoor function?

Is RSA indeed a trapdoor function?

We don't know. The integer factoring problem is widely believed to be intractable on classical computers, though it is not known to be NP-hard.

Is RSA indeed a trapdoor function?

We don't know. The integer factoring problem is widely believed to be intractable on classical computers, though it is not known to be NP-hard.

Also note that breaking RSA = given N, e and $m^e \pmod N$, compute $m \pmod N$, i.e. extract the e-th root modulo N. This can be in principle done without factoring N, its just that factoring is the most efficient known method for it. However:

> If we can compute $\varphi(N)$ fast, we can factor N fast. (I.e., there is no more efficient way of computing $\varphi(N)$ than by factoring N and applying the product formula.)

p, $p/2$, $2p$

$N = p \cdot q$

$$\varphi(N) = (p-1)\cdot(q-1) = \underbrace{\underbrace{p \cdot q}_{N} \underbrace{- p - q}_{-(p+q)} + 1}_{\varphi(N)}$$

$-(p+q) \rightsquigarrow \cancel{p+q}$

$+$

$$(p+q)^2 = \underbrace{p^2 + 2pq + q^2}\ \underbrace{-4pq}_{N} = \underbrace{p^2 - 2pq + q^2}_{\varphi(N)} = \sqrt{(p-q)^2}$$

$\cancel{p-q}$

Is RSA indeed a trapdoor function?

We don't know. The integer factoring problem is widely believed to be intractable on classical computers, though it is not known to be NP-hard.

Also note that breaking RSA = given N, e and $m^e \pmod{N}$, compute $m \pmod{N}$, i.e. extract the e-th root modulo N. This can be in principle done without factoring N, its just that factoring is the most efficient known method for it. However:

> If we can compute the decryption exponent d fast, we can factor N fast. (I.e., there is no more efficient way of computing d than by factoring N.)

Is RSA indeed a trapdoor function?

We don't know. The integer factoring problem is widely believed to be intractable on classical computers, though it is not known to be NP-hard.

Also note that breaking RSA = given N, e and $m^e \pmod{N}$, compute $m \pmod{N}$, i.e. extract the e-th root modulo N. This can be in principle done without factoring N, its just that factoring is the most efficient known method for it. However:

What is not known is whether the following holds:

> Open question: If one can recover m from $m^e \pmod{N}$ fast, can one factor N fast?

Some partial results indicate that this might not be true.

RSA is only a trapdoor function given our current state of knowledge.

Choose N that is hard to factor!

- To prevent factoring, N has to be sufficiently large (typical bitsizes $2048, 3072, 4096$).

Choose N that is hard to factor!

- To prevent factoring, N has to be sufficiently large (typical bitsizes $2048, 3072, 4096$).
- There are factoring algorithms whose runtime is dominated by a term exponential in the bitsize of the smallest prime factor of N. (Pollard's ρ algorithm, Lenstra's elliptic curve algorithm...).

$N = p \cdot q$ ✓

- In particular, factoring smooth numbers (those with only small prime factors) is easy. Hence, p, q should be roughly of similar bitsize.

"Algoritmy teorie čísel"

Choose N that is hard to factor!

- To prevent factoring, N has to be sufficiently large (typical bitsizes $2048, 3072, 4096$).
- There are factoring algorithms whose runtime is dominated by a term exponential in the bitsize of the smallest prime factor of N. (Pollard's ρ algorithm, Lenstra's elliptic curve algorithm…).
- In particular, factoring smooth numbers (those with only small prime factors) is easy. Hence, p, q should be roughly of similar bitsize.
- However, $p - q$ should not be too small. Otherwise, $p, q \approx \sqrt{N}$ and N could be factored by exhaustive search for p, q in the vicinity of the (likely irrational) number $\sqrt{N}$.

$\sqrt{N}$

Choose N that is hard to factor!

- To prevent factoring, N has to be sufficiently large (typical bitsizes $2048, 3072, 4096$).
- There are factoring algorithms whose runtime is dominated by a term exponential in the bitsize of the smallest prime factor of N. (Pollard's ρ algorithm, Lenstra's elliptic curve algorithm...).
- In particular, factoring smooth numbers (those with only small prime factors) is easy. Hence, p, q should be roughly of similar bitsize.
- However, $p - q$ should not be too small. Otherwise, $p, q \approx \sqrt{N}$ and N could be factored by exhaustive search for p, q in the vicinity of the (likely irrational) number $\sqrt{N}$.
- There are factoring algorithms that work well if N has a factor p s.t. $p - 1$ or $p + 1$ is smooth (Pollard's $p - 1$ algorithm, Williams's $p + 1$ algorithm). Choice of such p and q should be avoided.

$N = p \cdot q$

$p-1$

$p+1$

Choose N that is hard to factor!

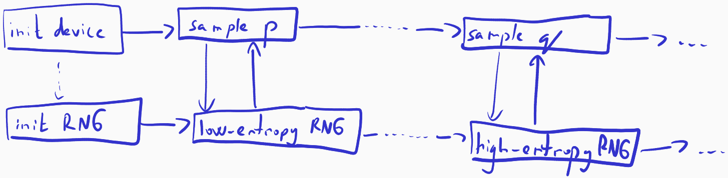
- To prevent factoring, N has to be sufficiently large (typical bitsizes $2048, 3072, 4096$).
- There are factoring algorithms whose runtime is dominated by a term exponential in the bitsize of the smallest prime factor of N. (Pollard's ρ algorithm, Lenstra's elliptic curve algorithm...).
- In particular, factoring smooth numbers (those with only small prime factors) is easy. Hence, p, q should be roughly of similar bitsize.
- However, $p - q$ should not be too small. Otherwise, $p, q \approx \sqrt{N}$ and N could be factored by exhaustive search for p, q in the vicinity of the (likely irrational) number $\sqrt{N}$.
- There are factoring algorithms that work well if N has a factor p s.t. $p - 1$ or $p + 1$ is smooth (Pollard's $p - 1$ algorithm, Williams's $p + 1$ algorithm). Choice of such p and q should be avoided.
- All of the aforementioned properties hold with high probability if p, q are randomly sampled from the set of all primes whose bitsize is roughly a half of the intended bitsize of N. How to sample: sample any number of the required bitsize and use an efficient primality test (e.g. Rabin's test) to check whether the generated number is a prime. Repeat until a prime is indeed found.

$N = p \cdot q$ ✓

On the importance of truly random prime generator

If an RSA key generator is flawed, it might be susceptible to generating, among multiple calls, two publick moduli $N_1 = p \cdot q_1$ and $N_2 = p \cdot q_2$ s.t. $q_1 \neq q_2$:

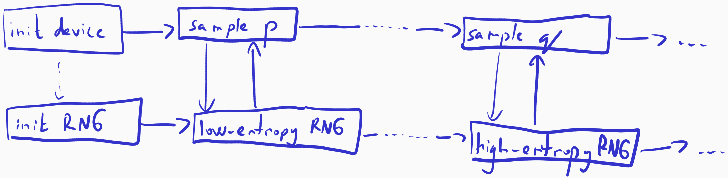
$\gcd(N_1, N_2) = p$



But then one can factor both N_1 and N_2

On the importance of truly random prime generator

If an RSA key generator is flawed, it might be susceptible to generating, among multiple calls, two publick moduli $N_1 = p \cdot q_1$ and $N_2 = p \cdot q_2$ s.t. $q_1 \neq q_2$:



But then one can factor both N_1 and N_2 by computing $\gcd(N_1, N_2) = p$.

An attack published in 2012 was able to factor about 0.3% public keys obtained from the Internet in this way.

Notes on encryption and decryption exponent

$\sqrt[e]{c} \pmod N$

- Computing m from $m^e \pmod N$ is already difficult for $e = 3$, assuming that $m^3 < N$.
- In practice, to make encryption fast, small encryption exponents are used $(3, 2^{16} + 1)$. Care must be taken during encryption that $m^e > N$.
- On the other hand, d should not be small: there is an efficient algorithm to compute d from N, e in situations where $d < \sqrt[4]{N}$ (Wiener's attack).
- Exponentiation by repeated squaring is performed to compute the exponentiation in polynomial time.

m^e $\quad m \cdot m \cdot m$

m^{16} $\quad m^{19}$ $\quad m \cdot m = m^2$ $\quad m^2 \cdot m^2 = m^4$

m^3

Why not use RSA directly for encryption?

Not randomized $\Rightarrow$ not CPA (and CCA) secure! E.g.: small message&exponent attack: if $m^e < N$, then m can be recovered by standard (non-modular) root computation.

Why not use RSA directly for encryption?

Not randomized $\Rightarrow$ not CPA (and CCA) secure! E.g.: small message&exponent attack: if $m^e < N$, then m can be recovered by standard (non-modular) root computation.

Another simple attack via Chinese remainder theorem:

Theorem 3: Chinese remainder theorem

Let $n_1, \dots, n_k$ be pairwise co-prime and $N = \prod_{i=1}^{k} n_i$ be their product. Then for every collection of integers $a_1, \dots, a_k$ s.t. $a_i \in \{0, 1, \dots, n_i - 1\}$ for all i, there exists a unique $x \in \underbrace{\{0, 1, \dots, N-1\}}_{\mathbb{Z}_N}$ s.t.

$$
\begin{aligned}
x &\equiv a_1 \pmod{n_1} \\
x &\equiv a_2 \pmod{n_2} \\
&\vdots \\
x &\equiv a_k \pmod{n_k}
\end{aligned}
$$

Moreover, x can be computed as follows: $x \equiv \sum_{i=1}^{k} a_i \cdot b_i \cdot (b_i^{-1} \pmod{n_i}) \pmod{N}$, where $b_i = \frac{N}{n_i}$.

CRT attack against RSA with small exponent

Let $e = 3$. Now assume that Alice sends the same message m to $e = 3$ recipients, each using a different modulus N_i. With high probability, the tree moduli N_1, N_2, N_3 are pairwise co-prime. Let $c_i = m^3 \pmod{N_i}$ be the ciphertext sent to recipient i.

- Necessarily $m < N_1, N_2, N_3$ (this holds in general for RSA messages).

$$\begin{tikzcd} & m \arrow[dl] \arrow[d] \arrow[dr] & \\ N_1 & N_2 & N_3 \end{tikzcd}$$

$\underbrace{m^e \pmod{N_1}}_{c_1}$ $\quad c_2 \quad c_3$

CRT attack against RSA with small exponent

Let $e = 3$. Now assume that Alice sends the same message m to $e = 3$ recipients, each using a different modulus N_i. With high probability, the tree moduli N_1, N_2, N_3 are pairwise co-prime. Let $c_i = m^3 \pmod{N_i}$ be the ciphertext sent to recipient i.

- Necessarily $m < N_1, N_2, N_3$ (this holds in general for RSA messages).
- But then $m^3 < N = N_1 \cdot N_2 \cdot N_3$.

CRT attack against RSA with small exponent

Let $e = 3$. Now assume that Alice sends the same message m to $e = 3$ recipients, each using a different modulus N_i. With high probability, the tree moduli N_1, N_2, N_3 are pairwise co-prime. Let $c_i = m^3 \pmod{N_i}$ be the ciphertext sent to recipient i.

- Necessarily $m < N_1, N_2, N_3$ (this holds in general for RSA messages).
- But then $m^3 < N = N_1 \cdot N_2 \cdot N_3$.
- Now let the adversary intercept c_1, c_2, c_3 and solve the following system using the Chinese remainder theorem:

$$
\begin{aligned}
x &\equiv c_1 \pmod{N_1} \\
x &\equiv c_2 \pmod{N_2} \\
x &\equiv c_3 \pmod{N_3}
\end{aligned}
$$

  Then $x \equiv c_i \pmod{N_i}$ for $i \in \{1, 2, 3\}$ and $x < N$. But also $m^3 \equiv c_i \pmod{N_i}$ and $m^3 < N$. By uniqueness of the solution, $x = m^3$ and m can be recover by computing (non-modular) $\sqrt[3]{x}$.
- Can be generalized to Håstad and Coppersmith attacks.

Typical use of RSA: salt, then encrypt

Typical defense against the aforementioned drawbacks is the use of randomized padding schemes that pad the plaintext with a randomly chosen cryptographic salt.

Typical use of RSA: salt, then encrypt

Typical defense against the aforementioned drawbacks is the use of randomized padding schemes that pad the plaintext with a randomly chosen cryptographic salt.

A practical, widely used instance of such a scheme is the one defined in PKCS#1 v1.5:

m - message to send securely (represented as a bitvector)
t - the bit-size of modulus N

PKCS v1.5 (m):

00	02	non-zero random bytes	00	m

byte

typically few hundreds of bits

t bits

encrypt with RSA

$x^3 < N$

$\left(\frac{1}{2}\right)^{250 \cdot \frac{2}{3}}$

RSA with PKCS#1 v1.5 padding (pseudocode)

In the following: $len(x)_8$ = bitsize of x in bytes, $len(x)$ = bitsize of x in bits

Algorithm 1: RSA with PKCS#1 v1.5 padding encryption

Input: public key $k_p = (N, e)$, message $m \in \mathbb{Z}_N^\times$
Output: $E(k_p, m)$
$t \leftarrow len(N)_8$;
if $len(m)_8 \geq t - 11$ **then return** *error: Message too long*;
$r \leftarrow$ sample randomly from $\{0,1\}^{t-8\cdot len(m)_8-24} \setminus \{x \mid x$ contains a zero byte$\}$;
$\bar{m} \leftarrow 0^8 \,||\, 00000010 \,||\, r \,||\, 0^8 \,||\, m$;
return $\bar{m}^e \pmod N$

Algorithm 2: RSA with PKCS#1 v1.5 padding decryption

Input: secret key (N, d), ciphertext $c \in \mathbb{Z}_N^\times$
Output: $D(k_s, m)$
$m \leftarrow c^d \pmod N$;
if *the second byte of* $m \neq 02$ **then return** *parse error*;
else return m;

Semantic security of public key cryptosystems: attack game

Given a cryptosystem $\mathcal{E} = (Gen, E, D)$, the semantic security attack game against $\mathcal{E}$ proceeds as follows:

Stage 1:

- The challenger samples $i \leftarrow \{0, 1\}$ uniformly ar random (and keeps it secret).
- The challenger computes $(k_p, k_s) \leftarrow Gen()$ and sends k_p to the adversary.
- The adversary computes (possibly in a randomized way) two messages, m_0 and m_1 of the same length and sends them to the challenger.

Semantic security of public key cryptosystems: attack game

Given a cryptosystem $\mathcal{E} = (Gen, E, D)$, the semantic security attack game against $\mathcal{E}$ proceeds as follows:

Stage 2:

- The challenger computes $c = E(k_p, m_i)$ and send it to the adversary.

Stage 3:

- The adversary performs its analysis of c.
- Finally, adversary outputs a guess $g \in \{0, 1\}$.

The adversary wins the game if $g = i$, otherwise it loses.

Attack game against PK cryptosystems: picture

Challenger

Adversary

$i \xleftarrow{R} \{0,1\}$

$(pk, sk) \leftarrow Gen()$ $\xrightarrow{pk}$ create m_1, m_2 s.t. $|m_0| = |m_1|$

$\xleftarrow{m_0,\, m_1}$

m_1^e, m_2^e

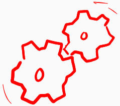
$c_1 \quad c_2$

$c \leftarrow E(pk, m_i)$

randomized

$\xrightarrow{c}$

analyze c



A wins if output $= i$ $\longleftarrow$ output $\in \{0,1\}$

m_i^e

Semantic security of public key cryptosystems

Definition 2: Semantic advantage

The semantic advantage of adversary $\mathcal{A}$ against cryptosystem $\mathcal{E}$ is the quantity

$$\mathcal{ADV}_{\mathcal{S}em}(\mathcal{E}, \mathcal{A}) = \mathbb{P}(\mathcal{A} \text{ wins the semantic attack game against } \mathcal{E}) - \frac{1}{2}.$$

Definition 3: Semantically secure cryptosystem

We say that $\mathcal{E}$ is an ε-semantically secure PK cryptosystem (where $\varepsilon > 0$) if for every efficient adversary it holds $ADV_{\mathcal{S}em}(\mathcal{E}, \mathcal{A}) \leq \varepsilon$.

We say that $\mathcal{E}$ is semantically secure if it is ε-semantically secure for a negligible value of ε.

Semantic security of public key cryptosystems II

The notions of CPA and CCA security for a public-key cryptosystem $\mathcal{E} = (Gen, E, D)$ are defined via modifications of the symmetric-key versions analogously to the previous slide:

- The CPA attack game is multi-round.
- For CCA security, the challenger initially computes $(k_p, k_s) \leftarrow Gen()$ and sends k_p to the adversary. In each plaintext query, the adversary's message $m_{i,b}$ is encrypted by the challenger using $E(k_p, \cdot)$. In each ciphertext query, the adversary's ciphertext c_i' is decrypted by the challenger using $D(k_s, \cdot)$.

Semantic security of public key cryptosystems III

Interesting security properties of PK cryptosystems that do not hold for symmetric ciphers:

Theorem 4: PK: Sem security $\Rightarrow$ CPA security

Let $\mathcal{E}$ be a public-key cryptosystem that is ε-semantically secure. Then $\mathcal{E}$ is $(\varepsilon \cdot N)$-CPA secure against any efficient adversary that makes at most N queries.

Theorem 5

Let $\mathcal{E}$ be a public-key cryptosystem that is ε-CCA secure against all adversaries that make at most one plaintext query. Then $\mathcal{E}$ is $\varepsilon \cdot N_p$-CCA secure againts all adversaries that make at most N_p plaintext queries.

Practical security of RSA with PKCS#1 v1.5 padding

RSA with PKCS#1 v1.5 padding is widely deployed in the Internet. However, if implemented incorrectly, it can be completely insecure.

00	02	r	00	m

If the decryption mechanism on a server reports parse errors to ciphertext senders, a malicious sender can test whether the second-to-last byte of a message encrypted in his ciphertext is 02.

$pk = (N, e)$

$m \rightsquigarrow m^e = c \longrightarrow c^d = m$

$m^e = c \cdot x^e \longrightarrow (c \cdot x^e)^d = (m^e \cdot x^e)^d =$

$m \cdot X$

$= (m \cdot x)^{e \cdot d} = m \cdot x$

Practical security of RSA with PKCS#1 v1.5 padding

RSA with PKCS#1 v1.5 padding is widely deployed in the Internet. However, if implemented incorrectly, it can be completely insecure.

$\mathbb{Z}_N \quad \mathbb{Z}_{2^k}$ $\quad \boxed{00 \mid 02 \mid \quad r \quad \mid 00 \mid m} \cdot 2^{\ell}$

If the decryption mechanism on a server reports parse errors to ciphertext senders, a malicious sender can test whether the second-to-last byte of a message encrypted in his ciphertext is 02.

This is exploited in practical chosen-ciphertext Bleichenbacher's attack: let (N, e) be the public key and suppose that the adversary intercepts a ciphertext c; for any $x \in \mathbb{Z}_N^{\times}$, the adversary can compute $x^e \cdot c \pmod{N}$ and send this to the server. If the server reports a parse error, the adversary knows that the second-to-last byte of $x \cdot PKCS(m)$ is different from 02. By trying various values of x, the adversary can simulate an analogue of right-shifts on bits, and eventually uncover the whole message.

Can break the SSL implementation of RSA-based key exchange via $\approx$ millions of queries.

How TLS defends against Bleichenbacher

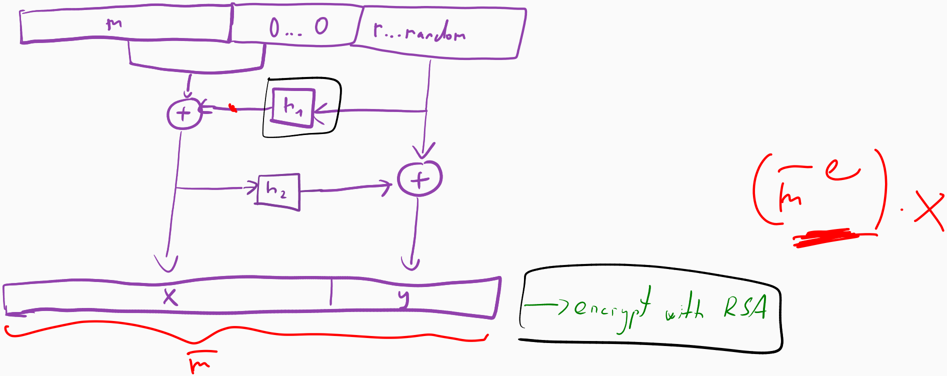
From TLS 1.2 standard:

> "In any case, a TLS server MUST NOT generate an alert if processing an RSA-encrypted pre-master secret message fails [...] Instead, it MUST continue the handshake with a randomly generated pre-master secret. It may be useful to log the real cause of failure for troubleshooting purposes; however, care must be taken to avoid leaking the information to an attacker (through, e.g., timing, log files, or other channels.)"
> (source: Boneh&Shoup)

Note: TLS 1.3 moved away from RSA altogether, in favour of discrete-logarithm encryption (in particular, elliptic-curve schemes).

Principled defense against Bleichenbacher: OAEP

Optimal Asymmetric Encryption Padding (1994): uses two hash functions h_1, h_2 (in practice, SHA-256 is used for both).



RSA-OAEP comes with a CCA-security theorem, which however makes stronger assumptions than RSA being trapdoor.

ISO standard RSA: CCA security by hybrid encryption with AE cipher

Uses an AE-enabled symmetric cipher $\mathcal{E}_S = (E_S, D_S)$ over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ and a hash function h whose hash space $\mathcal{H}$ equals $\mathcal{K}$. Also comes with a security theorem.

Algorithm 3: Encryption function of *ISO-RSA*.

Input: public key (N, e), message $m \in \mathcal{M}$

Output: $E_{ISO\text{-}RSA}((N, e), m)$

$x \leftarrow$ sample randomly from $\mathbb{Z}_N^\times$;

$y \leftarrow x^e \pmod N$;

$k \leftarrow h(x)$;

return $(y, E_S(k, m))$

Algorithm 4: Decryption function of *ISO-RSA*.

Input: secret key (N, d), ciphertext $(y, c) \in \mathbb{Z}_N^\times \times \mathcal{C}$

Output: $D_{ISO\text{-}RSA}((N, d), c)$

$x \leftarrow y^d \pmod N$; $k \leftarrow h(x)$;

$m \leftarrow D_S(k, c)$;

if $m = \perp$ **then return** *reject*;

else return m;

ISO standard RSA: CCA security by hybrid encryption with AE cipher

Uses an AE-enabled symmetric cipher $\mathcal{E}_S = (E_S, D_S)$ over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ and a hash function h whose hash space $\mathcal{H}$ equals $\mathcal{K}$. Also comes with a security theorem.

Algorithm 3: Encryption function of *ISO-RSA*.

Input: public key (N, e), message $m \in \mathcal{M}$
Output: $E_{ISO\text{-}RSA}((N, e), m)$
$x \leftarrow$ sample randomly from $\mathbb{Z}_N^\times$;
$y \leftarrow x^e \pmod{N}$;
$k \leftarrow h(x)$;
return $(y, E_S(k, m))$

Works for any trapdoor function!

Algorithm 4: Decryption function of *ISO-RSA*.

Input: secret key (N, d), ciphertext $(y, c) \in \mathbb{Z}_N^\times \times \mathcal{C}$
Output: $D_{ISO\text{-}RSA}((N, d), c)$
$x \leftarrow y^d \pmod{N}$; $k \leftarrow h(x)$;
$m \leftarrow D_S(k, c)$;
if $m = \perp$ **then return** *reject*;
else return m;

Hybrid encryption

- The hybrid encryption scheme exemplified in *ISO-RSA*(encrypt the message with a symmetric cipher and send the symmetric key via public-key encryption system) can be straightforwardly adapted to work with any trapdoor function.
- Also, hybrid encryption is the method of choice for sending long messages without key pre-negotiation.