The RSA cryptosystem

Mathematical Cryptography, Lectures 5 - 6

Contents

The RSA cryptosystem

- The RSA cryptosystem is a public key cryptosystem.
- Its security is based on the prime factorization problem.
- Published by Ronald Rivest, Adi Shamir, Leonard Adleman, USA, 1977. Patented in 1983.
- Independently investigated by James Ellis, Clifford Cocks, Malcolm Williamson, England, 1969-75, but not published until 1997.

The RSA cryptosystem

Generating a key pair

Alice wants to receive encrypted messages from Bob. Messages are natural numbers less than a number N.

Alice chooses two different primes p, q such that $n = pq > N$.
She calculates $\varphi(n) = (p-1)(q-1)$.
Than she chooses $e \in \mathbb{N}$ so that $\gcd(e, \varphi(n)) = 1$.
She computes $d = e^{-1}$ in $\mathbb{Z}_{\varphi(n)}$, which exists due to relative primality.

- The public key is (n, e), Alice places it in a "telephone book".
- The private key is (n, d), Alice keeps it in secret.

The RSA cryptosystem

Message encryption and decryption

- Bob wants to send Alice a message $a < N < n$ (an open message). He takes Alice's public key (n, e) and encrypts the message as follows:

  $$a^e = b \text{ in } \mathbb{Z}_n, \text{ where } 0 \leq b < n.$$

  Bob sends the message $b < n$ (a cipher message).
- Alice uses her private key (n, d) to decrypt as follows:

  $$b^d = a \text{ in } \mathbb{Z}_n, \text{ where } 0 \leq a < n.$$

Here n is the modulus, e is the encryption exponent, d is the decryption exponent of the RSA protocol scheme.

The RSA cryptosystem

The RSA protocol works correctly:

Proposition

Let $n = pq$, where $p \neq q$ are primes, and let $e, d \in \mathbb{N}$ satisfy $ed = 1$ in $\mathbb{Z}_{\varphi(n)}$.
Then for every $a \in \mathbb{Z}_n$, $(a^e)^d = a$ in $\mathbb{Z}_n$.

The proof follows from the Euler-Fermat's theorem in case a is relatively prime to n, for another a we must use the Chinese reminder theorem as well.

The RSA cryptosystem

Proposition

Let $n = \prod_{i=1}^{k} p_i$, where $p_1, \ldots, p_k$ are pairwise relatively prime, and let $e, d \in \mathbb{N}$ satisfy $ed = 1$ in $Z_{\varphi(n)}$.
Then for every $a \in \mathbb{Z}_n$, $(a^e)^d = a$ in $\mathbb{Z}_n$.

The RSA protocol can be used any "square free" modulus n and it will work correctly.

However, if $p^2 \mid n$ for some prime p, then messages divisible by p, but not divisible by the maximum power p^m contained in the factorization of n, would not be correctly decrypted.

The RSA cryptosystem

To generate an RSA key pair we need:

- to generate random large primes (the probabilistic Miller-Rabin's primality test is used; it takes $O(l^4)$ to generate an $l-$digit prime number)
- to compute the inverse of an exponent e in $Z_{\varphi(n)}$ (the extended Euclidean algorithm; required time is $O(\mathrm{len}(n)^2)$)

The following number sizes are used (see Shoup, 2008; twice as long numbers are common today).

- both primes p, q are about 512 bits long, the module $n = pq$ is 1024 bits long
- a private exponent d has up to 1024 bits (at least 512 bits)
- a public exponent e is much shorter (but at least 17 bits long)

The RSA cryptosystem

For encryption and decryption we need:

- faster exponentiation in $\mathbb{Z}_n$ (repeated squaring algorithm; required time is $O(\mathrm{len}(n)^3)$)
- decryption can be done residually in $\mathbb{Z}_p$ and $\mathbb{Z}_q$ (we use the Euler-Fermat's theorem and the Chinese reminder theorem; required time is $O(2(\frac{\mathrm{len}(n)}{2})^3) = O(\frac{1}{4}\,\mathrm{len}(n)^3)$, so it runs four times faster)

In practice, RSA protocol is used only to exchange keys for a symmetric cryptosystem. Messages are encrypted with a symmetric protokol (as DES, AES, IDEA), which is much faster (hundred or thousand times faster).

Security of the RSA cryptosystem

The prime factorization problem

The security of the RSA cryptosystem is based on time complexity of the prime factorization problem.

If one could decompose a module n into primes $n = pq$, then he can compute a private key from a public key and decrypt a message. This is a *brute force attack*.

The problem of factorizing n into primes is an exponential or subexponential problem:

- dividing by all primes up to $\sqrt{n}$ takes $O(2^{\frac{1}{2}\operatorname{len}(n)})$ time
- the fastest algorithm so far (the General number field sieve) requires time $O(2^{(c+o(1))\operatorname{len}(n)^{1/3}\operatorname{len}(\operatorname{len}(n))^{2/3}})$, where $c < 2$

Security of the RSA cryptosystem

Discrete root's modulo n

The discrete power function $\rho : \mathbb{Z}_n \to \mathbb{Z}_n : x \mapsto x^e$ is so called *one-way function*. If we want to invert it and to compute the discrete $e-$th root of b in $\mathbb{Z}_n$, we need to know the factorization of n in order to count d for which $ed = 1$ in $Z_{\varphi(n)}$. The exponent d is a *backdoor* to inverting of this function.

Without knowledge of d, the e-th root of b in $\mathbb{Z}_n$ can only be calculated by *brute force*, i.e. by successive powering by e of all $a \in \mathbb{Z}_n$ until we get $a^e = b$ in $\mathbb{Z}_n$. This requires exponential time $O(2^{\mathrm{len}(n)})$ again.

Security of the RSA cryptosystem

Can $\varphi(n)$ or d be found without factorizing n into primes?

Proposition

Let n be the product of two distinct primes. Knowledge of these primes is equivalent to knowledge of $\varphi(n)$.

Proof: $n = pq$, $\varphi(n) = (p-1)(q-1) = n - (p+q) + 1$.
If we know n and $\varphi(n)$, then we know the product π and the sum σ of two unknown numbers p, q. Therefore, these numbers are solutions of a quadratic equation $x^2 - \sigma x + \pi = 0$.

Note

We show later that we can effectively factorize any n using a knowledge of $\varphi(n)$.

Security of the RSA cryptosystem

Proposition

Let (n, e) be a public key of an RSA protocol.
Knowledge of the private exponent d allows us to factorize n effectively.

Lemma

If we find the nontrivial square root of 1 in $\mathbb{Z}_n$ (i.e., $b \neq \pm 1$ for which $b^2 = 1$ in $\mathbb{Z}_n$), then we can factorize n.

To prove the proposition, we know that $ed = 1$ in $\mathbb{Z}_{\varphi(n)}$. Hence $ed - 1 = k\varphi(n) = k(p-1)(q-1)$, where $p-1$, $q-1$ are even numbers. For arbitrarily $a \in \mathbb{Z}_n^*$, $a^{\varphi(n)} = 1$ in $\mathbb{Z}_n$ (Euler-Fermat). Since $a^{ed-1} = 1$ in $\mathbb{Z}_n$, $b = a^{(ed-1)/2}$ satisfy $b^2 = 1$ in $\mathbb{Z}_n$ (the exponent $\frac{ed-1}{2}$ is an integer). If $b \neq \pm 1$, we can factorize n.

Security of the RSA cryptosystem

Algorithm for factorizing modulus n from knowledge of public and private exponents

Input: public key (n, e) and private key (n, d) of the RSA protocol
Output: a prime p, which is a factor of the module n

- Calculate r such that $ed - 1 = 2^r l$, where l is odd
- repeat
  - choose $a \in \mathbb{Z}_n$ randomly
  - $d \leftarrow \gcd(a, n)$
  - if $d > 1$ then output d (and stop)
  - $c \leftarrow a^l$ in $\mathbb{Z}_n$ ($a \in \mathbb{Z}_n^*$ now)
  - while $c^2 \neq 1$ do $c \leftarrow c^2$ in $\mathbb{Z}_n$ enddo (we know that $c^{2^r} = 1$)
- until $c \neq \pm 1$
- output $\gcd(c - 1, n)$

Security of the RSA cryptosystem

Time complexity of the algorithm

The probability of finding the non-trivial square root of 1 in $\mathbb{Z}_n$ by choosing $a \in \mathbb{Z}_n^*$ randomly equals to $\frac{1}{2}$, in case $n = pq$. So we can expect two repeat-cycles to find it.

We use the Euclidean algorithm and the repeated squaring algorithm, where the exponent $ed - 1$ is assumed to be of size $O(n)$. The expected running time of the algorithm is $O(2\,\mathrm{len}(n)^3)$.

Summary

We have shown that to obtain any useful information to decrypt - whether $\varphi(n)$ or d - it is equivalent to being able to factorize the modulus n. Thus, it requires (at least so far) an exponential or subexponential time.

Common modulus attacks

The insider attack

The administrator designs k key pairs with the same modulus n, let's denote the key pair of $i-$th participant as (n, e_i), (n, d_i) for $1 \leq i \leq k$.

Any participant can factorize n by knowing his private key d_i, using the previous algorithm. Then he can compute the private keys of the other participants and decrypt their messages.

A defense against the attack

To generate always a new modulus n is necessary.

Common modulus attacks

The outsider attack

The administrator designs k key pairs with the same modulus n, let's denote the key pair of $i-$th participant as (n, e_i), (n, d_i) for $1 \leq i \leq k$.

Eve aevesdrops messages sent to two (or to $s \leq k$) participants whose public keys were relatively prime. Moreover, Eve knows that these encrypted messages come from the same open message:
Eve has $b_1, \ldots, b_s$, where $b_i = a^{e_i}$ in $\mathbb{Z}_n$, and she wants to find a.

Bezout's theorem says, that $1 = \gcd(e_1, \ldots, e_s) = t_1 e_1 + \ldots + t_s e_s$ for suitable $t_i \in \mathbb{Z}$.
Eve can compute $b_1^{t_1} \cdot \ldots \cdot b_s^{t_s} = a^{t_1 e_1 + \ldots + t_s e_s} = a^1 = a$ in $\mathbb{Z}_n$.

Common modulus attacks

The outsider attack

Note that some of the coefficients of t_i must be negative, so $b_i^{t_i} = (b_i^{-1})^{|t_i|}$ in $\mathbb{Z}_n$ for $t_i < 0$. However, the cipher message b_i does need to be invertible in $\mathbb{Z}_n$!

In that case $\gcd(b_i, n) = p$, Eve factorizes modulus $n = pq$ and computes any private key (n, d_j).

A defense against the attack

To generate always a new modulus n is necessary.

Common public exponent attacks

The Hastad's broadcast attack

Suppose that k participants have key pairs with the same small public exponent e, where $e \leq k$. Let us denote the public key of the $i-$th participant by (n_i, e).

Moreover, the same open message $a < n_i$ (for all $1 \leq i \leq k$) was encrypted and sent to every participant. Eve intercepted these cipher messages $b_1, \ldots, b_k$, where $b_i = a^e$ in Z_{n_i}.

Thus Eve knows the reminder of a^e modulo each n_i. We can assume that the set $n_1, \ldots, n_k$ is pairwise relatively prime. Eve can easily test this by Euclidean algorithm, and if not, she will find a factor p of some n_i and compute the private key (n_i, d_i).

If yes, the Chinese reminder theorem can be used and Eve computes $b = a^e$ in $\mathbb{Z}_n$, where $n = \prod_{i=1}^{k} n_i$. Since $a^e < n$, then $b = a^e$ in $\mathbb{Z}$ and Eve can compute $a = \sqrt[e]{b}$ in $\mathbb{Z}$.

Common public exponent attacks

Defenses against the Hastad's attack

1) Do not use a too small public exponent e.

A public exponent $e > 2^{16}$, at least $17-$bits long, is recommended. This already guarantees the security and it still allow a fast encryption.

2) We can add some random bits to the message a before each encryption, so we will encrypt a different message a_i for each partitipant.

Adding random bits is a necessary common practice in all deterministic cryptosystems. Without it, encryption would not have a semantic security, which means - if Eve can guess which message is Bob sending to Alice, she can easily verify her proposal by encrypting it.

Small private exponent attacks

Proposition (The Wiener's attack)

Let $n = pq$ be chosen for the RSA protocol, where $q < p < cq$ for small $c \in \mathbb{N}$, $e < \varphi(n)$, $d < \frac{1}{c+1} n^{\frac{1}{4}}$. Then the private key (n, d) can efficiently compute from the public key (n, e).

The proof is based on an approximation of a number using continued fractions. They can be computed by Euclidean algorithm.

$$\frac{a}{b} = q_1 + \cfrac{1}{q_2 + \cfrac{1}{\vdots \cfrac{}{q_{\lambda-1} + \frac{1}{q_\lambda}}}}$$

For a rational number $\frac{a}{b}$, the continued fraction has a finite length $\lambda < 2 \min\{\mathrm{len}(a), \mathrm{len}(b)\}$.

Small private exponent attacks

Let us denote the continued fraction for $\frac{a}{b}$ as $(q_1; q_2, \ldots, q_\lambda)$.
A subsequence $(q_1; q_2, \ldots, q_i)$ is called the $i-$th convergent of $\frac{a}{b}$.
(For an irrational number r, the infinite sequence of convergents rationally approximates r and converges to r.)

From the equality $ed - 1 = k\varphi(n)$, where k, d and $\varphi(n)$ are not known, the following can be derived (thanks to assumptions of the proposition): $|\frac{e}{n} - \frac{k}{d}| < \frac{1}{2d^2}$, $\gcd(k, d) = 1$.
This guarantees that the fraction $\frac{k}{d}$ must be one of the convergences of the rational number $\frac{e}{n}$ (which is a deeper result).

It remains to compute $O(\mathrm{len}(n))$ convergents of $\frac{e}{n}$ and to try for each of them, whether its denominator can be the private exponent we are looking for. For example, we can compute $\varphi(n)$ from the equation above and use it to factorize n, or we try to encrypt and decrypt random messages.

Small private exponent attacks

A defense against the Wiener's attack

Do not use a too small private exponent d.

It turns out that the Wiener's bound is not fixed. It is likely that with $d < n^{\frac{1}{2}}$ one can efficiently compute d from a public key e.
It is recommended to choose d of at least 512 bits length (when n has 1024 bits), but d of 1024 bits length are commonly used.

When decrypting, a large exponent does not matter, because we compute residually in $\mathbb{Z}_p$ and $\mathbb{Z}_q$ and we reduce the exponent modulo $p-1$ and $q-1$, respectively.
But even these residual exponents d_p, d_q must not be too small, since another attack exists that allows to factorize n in time $O(\min \sqrt{d_p}, \sqrt{d_q}) = O(2^{\frac{1}{2}\min\{\mathrm{len}(d_p),\mathrm{len}(d_q)\}})$.

Implementation attacks

1. Kocher: Measuring the decryption computational time. (The repeated squaring algorithm is driven by bits of the exponent d, so d can be reconstructed bit by bit from the time measurements. Probability and statistics are being used for it.)
2. Random errors in decoding. (If the residual counting produces an error in only one prime modulus, then one can factorize n.)
3. Error messages when receiving a cipher message in a non-standard form. (Eve sends the intercepted cipher message b to Alice several times, in the forms cb for different $c \in \mathbb{Z}_n$. Depending on receiving an error message or not, she is able to detect the open message a.)

The first two attacks involve sneaking a chip into Alice's computer.

Attacks on RSA cryptosystem

Conclusion

All the attacks point out what to be careful about when using the RSA protocol scheme. So far, no attack has been found which would devalued the RSA cryptosystem.

The RSA cryptosystem guarantees complete security in digital communication when implemented correctly.

Message authentication

Public key encryption opens the following questions:

- Can Bob be sure that Alice's public key was actually put into the "phone book" by Alice? What if Eve snuck it in so she could read Bob's messages to Alice?
  Alice's public key must have a *certificate* (= a certificate authority met Alice and confirmed that the key is hers).
- Can Alice be sure that the message was really sent by Bob? What if it was sent by Eve pretending to be Bob?
  Bob can attach a *digital signature* to his message. The signature is to guarantee that Bob is the author of the message and that the message has not been changed along the way.

Message authentication

The digital signature for RSA encryption

Bob can sign his open message with his private key before encrypting it with Alice's public key.

$$a \longrightarrow a^{d_B} \ v \ \mathbb{Z}_{n_B} \longrightarrow (a^{d_B})^{e_A} = b \ v \ \mathbb{Z}_{n_A}$$

Alice decrypts with her private key and then removes the signature with Bob's public key.

$$b \longrightarrow c^{d_A} \ v \ \mathbb{Z}_{n_A} \longrightarrow (c^{d_A})^{e_B} = a \ v \ \mathbb{Z}_{n_B}$$

If she obtains a meaningful message, Bob must have written it.

Note: The message must be signed first and encrypted after. Otherwise, Eve could remove the signature (with Bob's public key) and sign the message herself.

Message Authentication

The digital signature for RSA encryption

Usually only the "hash" of an open message is signed.
It's faster and still resistant to signature attacks.

Alice and Bob choose a hash function $h(x)$ (there is symmetry here).
Bob signs the hash of his open message with his private key, then encrypts the message and the signature with Alice's public key:

$$a \longrightarrow (a, h(a)^{d_B}) \text{ in } \mathbb{Z}_{n_B} \longrightarrow (a^{e_A}, (h(a)^{d_B})^{e_A}) = (b, c)$$

Alice decrypts both parts with her private key, and takes off Bob's signature from the hash with Bob's public key.

$$(b, c) \longrightarrow (b^{d_A}, c^{d_A}) = (a, m) \text{ in } \mathbb{Z}_{n_A} \longrightarrow (a, m^{e_B}) \text{ v } \mathbb{Z}_{n_B}$$

Then Alice computes the hash from the decrypted message.
If $h(a) = m^{e_B}$ v $\mathbb{Z}_{n_B}$, then Bob sent the message.

Message authentication

The digital signature for symmetric encryption

The digital signature for symmetric cryptosystems uses its symmetric key. This key r specifies the parameters of the hash function $h_r(x)$.

Signature and encryption: $a \longrightarrow (a, h_r(a)) \longrightarrow (b, c)$
Decryption: $(b, c) \longrightarrow (a, m)$, signature verification: $h_r(a) = m$

Both sides must know the encryption key and the signature key. Message Authentication Codes (MACs) use different sets of hash functions.

Message authentication

Hash functions

A set of hash functions $h_r(x)$, $r \in R$, must satisfy:

- $h_r(x)$ must significantly shorten a message to a constant length;
- each hash has roughly the same number of patterns, and these are spreaded over the message set such that a small change in the message causes a large change in the hash (nothing about the message can be guessed from the hash);
- any message can be shortend to any hash (preferably with equal probability), when using different keys, (or Eve cannot falsify Bob's signature under her message if she doesn't know the key r).

Message authentication

Hash functions

Examples of hash functions (see Shoup):

- Key $r = (r_0, r_1, \ldots, r_k) \in \mathbb{Z}_p^{\times(k+1)}$, p prime, $k \in \mathbb{N}$.
  $h_r : \mathbb{Z}_p^{\times k} \to \mathbb{Z}_p : (x_1, \ldots, x_k) \mapsto r_0 + r_1 x_1 + \cdots + r_k x_k$
- Key $r = (r_0, r_1) \in \mathbb{Z}_p \times \mathbb{Z}_p^*$, p prime, $m \in \mathbb{N}$, $m < p$.
  $\tilde{h}_r : \mathbb{Z}_p \to \mathbb{Z}_m : x \mapsto (r_0 + r_1 x) \bmod m$, where the *mod* operation returns the remainder after division.
- Key $r \in \mathbb{Z}_p$, p prime, $k \in \mathbb{N}$.
  $\bar{h}_r : \mathbb{Z}_p^{\times(k+1)} \to \mathbb{Z}_p : (x_0, x_1, \ldots, x_k) \mapsto x_0 + x_1 r + x_2 r^2 \cdots + x_k r^k$

h_r has a long key but a small p (corresponding to the hash length).
$\tilde{h}_r$ has a short key but a large p (as the message length).
$\bar{h}_r$ has both advantages, but slightly weaker parameters.

Message authentication

Hash functions

How to hash a binary message using these functions?

For example, we want to hash a message $a < n$, where n is 1024 bits long, to a 160−bits long hash.
We choose a prime p of 161 bits, i.e. $2^{160} < p < 2^{161}$ (such a prime exists by Bertrand's postulate).
We split the message a into messages $a_1, \ldots, a_7$ of length 160 bits, then $a_i < p$.
We use the function $h_r(x)$, where the key r will contain 8 numbers from $\mathbb{Z}_p$.

The RSA cryptosystem

Literature

- Velebil: Discrete mathematics. Chapters 3.5-6, Appendix A. ftp://math.feld.cvut.cz/pub/velebil/y01dma/dma-notes.pdf
- Boneh: Twenty Years of Attacks on the RSA Cryptosystem. https://crypto.stanford.edu/ dabo/papers/RSA-survey.pdf
- Shoup: A Computational Introduction to Number Theory and Algebra. Chapter 4.7, 8.7. http://shoup.net/ntb/