CAI: Cerca i Anàlisi d'Informació
Grau en Ciència i Enginyeria de Dades, UPC

9. Public-key cryptography

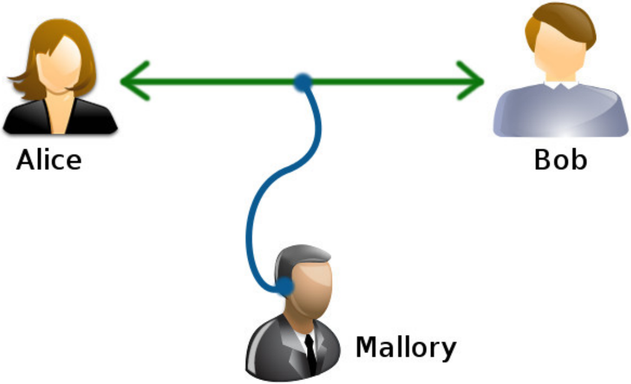
December 20, 2019

Slides by Marta Arias, José Luis Balcázar, Ramon Ferrer-i-Cancho, Ricard Gavaldà, Department of Computer Science, UPC

Contents

The problem

Alice and Bob want to exchange messages privately.
But malicious Eve may be listening (insecure channel)



https://images.app.goo.gl/rE6nJUADdaSAs8PU9

Private-key encryption

Solution: Encryption and decryption functions E and D.
E and D fixed and possibly known to everybody.

- Alice and Bob agree on a secret key k
- Alice sends a message m to Bob by computing $E(m, k)$
- Bob decodes the message by computing $D(m, k)$

Private-key encryption

Assumptions:

- $D(E(m, k), k) = m$ for all m
- Computing $E(m, k)$ and $D(m, k)$ is easy
- Computing x from $E(x, k)$ is impossible (or very hard) without knowing k.

We often we have $E(D(x, k), k) = x$ too.
Then we speak of *symmetric key encryption.*

Private-key encryption

Example: Choose $k \in \{0..25\}$

$$E(x,k) = (x+k) \bmod 26, \quad D(x,k) = (x-k) \bmod 26.$$

Example: If messages have n bits, then k is a random n-bit string and use exclusive-or:

$$E(x,k) = D(x,k) = x \oplus k.$$

If k is used repeatedly and messages are in English, these schemes is easy to attack using statistical properties of English text.

Private-key encryption, better schemes

- DES (56 bit keys) and 3DES (168 bits):
  - 16 iterations of substitutions + shiftings (3x for 3DES)
  - 70's-90's. Not considered safe against Big Brother any more
  - Still useful for weak attackers, and used in some chip cards.
- AES (Advanced Encryption Standard)
  - US standard from 2001
  - 128 - 256 bit keys

Private-key encryption, problems

Three problems:

1. Alice and Bob must keep k secret.
2. Alice and Bob must physically meet to agree on k, or use a secure channel, or use a trusted third party.
3. If Alice wants to talk to many people, a distinct k must be agreed for each.

Public-key cryptography solves 2 and 3.

What is a public-key encryption scheme?

- A way of generating "random" pairs of keys (e, d)
- An encryption function $E(x, e)$
- A decryption function $D(y, d)$

with the properties:

- Generating pairs and computing E and D must be feasible (poly-time)
- $D(E(x, e), d) = x$
- Obtaining x from $E(x, e)$ knowing e but not d must be unfeasible (non poly-time)

E and D fixed and possibly known to everybody.
Encrypting and decripting keys are different $\rightarrow$ "asymmetric encryption scheme".

Why is this wonderful?

Alice wants *anybody* to be able to send her private messages:

- Alice generates her pair (e, d)
- Keeps d private, but publishes e *to the world*
- Bob sends her message m by using Alice's public key e: sends her $y = E(m, e)$
- Alice retrieves m by computing $D(y, d)$
- Mallory sees y and e, but can't obtain m

Glitch

Mallory can:

- Generate (e, d)
- Tell the world "I am Alice and my public key is e"
- Unsuspecting Bob sends Alice a private message $E(m, e)$, that Mallory hears.
- Mallory can decode m using d!

This is why Certification Authorities exist: Parties trusted by everybody (conventionally) that certify that "Alice" is truly Alice, and where Alice's e can be obtained.

But, do such wonderful schemes exist?

Not if P=NP. Proof sketch:
We are Mallory and know e but not d. We want to decrypt.

- Get y (by listening in the insecure channel)
- Guess d
- Compute $x = D(y, d)$
- Verify that $E(x, e) = y$
- So x is “the right” one

So specific public key schemes must rely on some computational problem assumed to be in NP but not in P.

Diffie-Hellman key exchange protocol

- Alice and Bob (publicly) agree on a prime p and a generator g of Z_p^*
- Alice secretly chooses x, then (insecurely) sends $g^x \bmod p$ to Bob
- Bob secretly chooses y, then (insecurely) sends $g^y \bmod p$ to Alice
- Alice privately computes $k = (g^x)^y \bmod p = g^{xy} \bmod p$
- Bob privately computes $k = (g^y)^x \bmod p = g^{xy} \bmod p$
- Alice and Bob now use k as a private key to communicate

Diffie-Hellman exchange: Example

- Alice and Bob agree, in the clear, to use $p = 17$ and $g = 3$
- Alice secretly and randomly chooses $x = 15$
- Alice computes $3^{15} \bmod 17 = 6$ and sends it to Bob
- Bob secretly and randomly chooses $y = 13$
- Bob computes $3^{13} \bmod 17 = 12$ and sends it to Alice
- Alice computes $12^{15} \bmod 17 = 10$
- Bob computes $6^{13} \bmod 17 = 10$
- Now Alice and Bob have agreed to use secret key 10

Exercise: Check these computations.
Note: Of course in practice one does not use 17, but a prime with > 1000 digits.

Key step: Modular exponentiation

$$g^x \bmod p = \begin{cases} (g^2 \bmod p)^{x/2} \bmod p & \text{if } x \text{ even} \\ (g \cdot (g^{x-1} \bmod p)) \bmod p & \text{if } x \text{ odd} \end{cases}$$

$O(N)$ multiplications of $O(N)$-bit numbers if p has N bits.

Discrete Logarithm

Modular exponentiation: given g, p, x, compute fast
$s = g^x \bmod p$

Discrete logarithm: given g, p, s, find x such that $s = g^x \bmod p$

$g^x \bmod p$ for $p = 19$, $g = 14$, $x = 0..18$:

[1, 14, 6, 8, 17, 10, 7, 3, 4, 18, 5, 13, 11, 2, 9, 12, 16, 15, 1]

Brute force is $O(p) = O(2^N)$ time, or precomputed huge table.

Diffie-Hellman exchange: Assumptions

Discrete Logarithm assumption: The following is hard:

Given p, g and s, find x such that $g^x = s \bmod p$

In the previous example, computing x from 6, or y from 12, faster than brute force.

Clearly, if we can do this, we can break Diffie-Helllman.

Diffie-Hellman exchange: Assumptions

Diffie-Hellman assumption: The following is hard:

Given p, g, s and t, find u such that if $s = g^x \bmod p$ and $t = g^y \bmod p$, then $t = g^{xy} \bmod p$

In the previous example, computing 10 having seen 6 and 12, faster than brute force.
One way is to first get x from 6 and y from 12, i.e., solve the Discrete Logarithm problem. But perhaps not the only way.

Diffie-Hellman reduces to Discrete Logarithm, but no reduction the other way is known.

RSA - Rivest, Shamir, Adleman (1977)

Key generation:

- Choose two large primes p and q (about $N/2$ bits each)
- Compute $n = p \cdot q$ (so n has about N bits)
- Choose e such that $gcd(e, (p-1)(q-1)) = 1$
- The public key is the pair (n, e)
- Let d be the multiplicative inverse of e, $\mathrm{mod}(p-1)(q-1)$

$$d \cdot e = 1 \mod (p-1)(q-1)$$

- The private key is the pair (n, d); keep p and q secret too

RSA - Rivest, Shamir, Adleman (1977)

Encrypting function $E(x,(n,e)) = x^e \bmod n$

Decrypting function $D(y,(n,d)) = y^d \bmod n$

Ingredients

How to find primes p, q

How to find and verify e

omitted; in practice, fixed, well studied e's used)

How to find d from e, p, q

Why is $D(E(x, (n, e)), (n, d)) = x$

Finding and verifying primes

Prime number theorem: The number of primes less than n is

$$\frac{n}{\log n}(1 + o(1))$$

So to look for a prime of about N digits:

- Picking one at random.
- With probability about $1/N$, it is a prime.
- Check for primality.
- If not prime, try again. You will find one soon.

Finding and verifying primes

There is a probabilistic poly-time algorithm A that, for each n,

- If n is prime, $A(n)$ says "prime" with probability $1 - \exp(-N)$, "?" otherwise.
- If n is composite, $A(n)$ says "composite" with probability $1 - \exp(-N)$, "?" otherwise.

where N is the number of digits of n.

Serious number theory involved.

How to find secret key d

We know e, p, q, let $\phi = (p-1)(q-1)$. We know $gcd(e, \phi) = 1$.

Want to find d such that $e \cdot d \bmod \phi = 1$.
i.e. such that there is k such that

$$d \cdot e + k \cdot \phi = 1$$

Can be done with Extended Euclidean algorithm

NOTE: We can do this *because* we know ϕ.

If we only know $n = pq$, but not p, q (or at least ϕ), we cannot do this - that would be breaking RSA!

Extended Euclidean Algorithm

Regular Euclidean algorithm: Given x, y, find $r = gcd(x, y)$.

Extended Euclidean algorithm: find also α, β such that

$$\alpha x + \beta y = r$$

(This is called Bézout's identity)

Extended Euclidean Algorithm

Idea for Euclid's algorithm: If $x > y$, apply repeatedly

$$gcd(x, y) = gcd(y, x \bmod y)$$

Suppose inductively that call to $gcd(y, x \bmod y)$ also returns α, β such that:

$$\alpha y + \beta(x \bmod y) = r$$

Then

$$\begin{aligned} r &= \alpha y + \beta(x \bmod y) \\ &= \alpha y + \beta(x - (x \text{ div } y)y) \\ &= \beta x + (\alpha - (x \text{ div } y))y \end{aligned}$$

Program this iteratively ("forward") rather than recursively ("backwards"), if desired.

Why do we decrypt well?

- If p and q primes, and
- $E(x,(n,e)) = x^e \bmod n$, and
- $D(y,(n,d)) = y^d \bmod n$, and
- $ed = 1 \bmod (p-1)(q-1)$, then
- E and D inverse permutations.

This uses Euler and Fermat's theorems, plus Chinese remainder theorem, plus the fact that Euler's totient function $\phi(n) =$ number of integers $< n$ that are coprime with n, plus $\phi(p) = p - 1$, $\phi(pq) = \phi(p-1)\phi(q-1)$.

Explained in many sources out there.

Is RSA secure?

Primality: Given an integer n, is n prime?

Trivial algorithm takes $O(\sqrt{n})$ time = $O(2^{N/2})$

Poly-time probabilistic algorithms since the 70's.

Proven to be in P in 2002. $O(N^{12})$. Exponent lowered since then.

Is RSA secure?

Factoring: Given n, get the prime decomposition of n.

In binary form, given n, m, does n have a factor less than m?

No polynomial algorithm is known. Pretty hard in practice for largish n

Think: Thousands of years even for Big Brother for n 2000 digits of so

But not expected to be NP-complete either, for at least two reasons:

- subexponential algorithms are known
- its complement is also in NP (guess true prime decomposition)

Is RSA secure?

The RSA problem: given $n = pq$ and e such that $gcd(e, (p-1)(q-1)) = 1$ and c, find m such that $me = cmodn$.
I.e., recover m from ciphertext c and public key (n, e) by taking e-th root of c
There is no known efficient algorithm for doing this.

If factoring is easy, then RSA is easy (RSA reduces to factoring):

- If an adversary can factor n, then it can compute p, q, then the private key (n, d) from the public key (n, e) in the same way that Alice did.

But no reduction from factoring to RSA is known:

- Breaking RSA does not require obtaining d, which would enable not just decyphering one message but the key to decypher all message.
- RSA could be breakable without factoring.

Comparisons

Pros of private key encryption:

- Encoding N bits with private key much faster than DH or RSA.
- Paritcularly, generating RSA key pairs quite slow.
- No complexity assumptions.

Comparisons

Pros of DH

- p and g precomputed, public and verified ("strong p")
- Key generation much faster (compute $g^x \bmod p$)
- Perfect Forward Secrecy:
  - Once a session terminates, throw away the secret parts (x, y, g^{xy})
  - If someone enters your system later, can't decrypt previous conversations.
  - In plain RSA exchanges, if someone enters your system and steals your private key, it can decrypt all your previous conversations.
  - (but solution: Alice creates random key, RSA-encrypts it with Bob's public key, sends to Bob).
  - (Distinction short-term (session) key, long-term key.)

Comparisons

Pros of RSA

- Alice and Bob can verify each other: Know who they are talking to.
- Mallory can pretend to be Alice or Bob during the DH exchange.
- “Man in the middle” attack
- Allows for Digital Signature: I can sign message m and
  - Everybody is sure it’s me who signed it
  - I can’t deny I signed it

Digital Signature

(Not specific to RSA. Works for any public-key scheme such that not only $D(E) = id$ but $E(D) = id$.)

- I want to sign message m undeniably. My keys are e, d.
- I publish $(m, D(m, d))$.
- People see (m, σ).
- They check (with my public e) that $E(\sigma, e) = m$.

Only I could create this from m

Digital Signature, anti-fakes

Glitch: People can generate fakes:

- Generate σ at random
- Publish $(E(\sigma, e), \sigma)$ in my name

(Not a big problem for signing PDF's; the fake will be garbage. But problematic in other uses).

Trick: Use a hash function h for which finding preimages is hard. Encoding:

- I publish $(m, D(h(m), d))$.
- People see (m, σ).
- They check (with my public e) that $E(\sigma, e) = h(m)$.

It seems hard to create a fake (x, σ) in this scheme.

TLS (and predecessor SSL)

Alice and Bob want to use keys of N bits
They want to exchange a message $m = m_1 m_2 \dots m_\ell$ where the $len(m_i) = N$

Option 1: Use public-key encoding on each m_i, so ℓ times complex calculations

Option 2, muuch faster:

1. Exchange ONCE a secret key k via DH (or RSA if you want to verify party's identity).
2. This is the *session key*, one per session.
3. Use private-key protocol (E, D) to send each $E(m_i, k)$ in the session.

(TLS protocol is this and much more engineering stuff)

Quantum

- Shor's algorithm (1994) solves the Discrete Logarithm and Factorization in quadratic time in an Ideal Quantum Computer.
- So DH and RSA can be broken by quantum computers.
- Intense work on Post-Quantum public-key cryptography scheme.
- They seem to exist. But less convenient (longer keys, less tested assumptions...).

Remember: Even if we succeed in building Ideal Quantum Computers, there is no theoretical evidence that they can solve NP-complete problems in polynomial time.

No reason to believe that NP $\subseteq$ "Quantum P" (QBP, technically)