Lecture 1: Introduction to Public key cryptography

Thomas Johansson
Key distribution

- Symmetric key cryptography: Alice and Bob share a common secret key.
- Some means of distributing a copy of the secret key
- *physical distribution.*
Assume that $n$ users are connected in a network and any two of them may want to communicate.

This would require each user to securely store $n - 1$ different symmetric keys (one for each other user), resulting in a total of $n(n - 1)/2$ keys.

If the network is connecting 2000 university students, then there will be roughly 2 million different keys.

A huge key management problem with questions like; How do you add a new user to the system? What if a user’s key is compromised? How long should a key be considered valid and how should we refresh them?
A better solution to the key distribution problem is obtained if we use symmetric *key distribution protocols*.

A trusted third party (TTP). Each user has a unique secret key shared with the TTP.

When two users would like to communicate, they establish a shared secret key, usually called a *session key*, by interacting with the TTP.

There are still drawbacks that can be serious problems in certain situations. For example, we need access and trust to a TTP and we still need to distribute one key shared with the TTP for each user.
Public-key cryptography

an extremely elegant solution...

- We assume two *different* keys, one for encryption of a plaintext message and another for decryption of a ciphertext message.
- Allow the encryption key to be *public*!
- Anyone with access to the public encryption key to send an encrypted plaintext to the receiver.
Informal definition.

Definition

A one-way function \( f(x) \) is a function from a set \( \mathcal{X} \) to a set \( \mathcal{Y} \) such that \( f(x) \) is easy to compute for all \( x \in \mathcal{X} \), but for “essentially all” elements \( y \in \mathcal{Y} \) it is “computationally infeasible” to find any \( x \in \mathcal{X} \) such that \( f(x) = y \).

- “easy” and “computationally infeasible”
- “essentially all elements”
Example

\[ X = Y = \mathbb{Z}_{29} \] and consider \( f : X \mapsto Y \) described by

\[ f(x) = 2^x \text{ mod } 29. \]

a) Roughly, how many operations do you need to calculate \( f(5) = 2^5 \)?

Calculate \( 2^2 = 2 \cdot 2 = 4 \), then \( 2^4 = 4 \cdot 4 = 16 \) and finally multiplying with 2, i.e., \( 2^5 = 16 \cdot 2 = 3 \). This procedure required 3 multiplications.
Example

b) Roughly, how many operations do you need to find an $x \in X$ such that $f(x) = 5$?

One way would be to calculate $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, \ldots until we reach $2^x = 5$. We find \ldots, $2^{22} = 5$ and this procedure resulted in 21 multiplications.
A function that looks like a one-way functions is the function $f : \mathbb{Z}_n \mapsto \mathbb{Z}_n$, with

$$f(x) = x^3 \mod n,$$

where $n$ is the number

$$n = 279978339112213278708294676387226016210704467869554285375600099293261284001076093456710529553618223519109513657886371059544820065767750985805576135790987349501441788631789462951872378692218.$$
A **trapdoor one-way function** $f(x)$ is a one-way function $f : X \mapsto Y$ such that if one knows some specific information $T$, called the *trapdoor information*, then $f(x)$ is computationally easy to invert $f$, i.e., for any $y \in Y$ it is easy to find a $x \in X$ such that $f(x) = y$. For anyone without knowledge of the trapdoor information $T$, $f(x)$ is a one-way function.
Example 2 again

- The function $f(x) = x^3 \mod n$ we had before is a trapdoor one-way function.
- We said that the function looked like a one-way function. However, there is some secret information about this number that can be used.
- The number $n$ is in fact the product of two large prime numbers. The trapdoor information $T$ is in this case these two prime numbers. Knowing them, you can easily invert the $f$ function.
Example 2 again

The RSA-200 number $n$ above was factored in 2003 and the factors are

$353246193440277012127260497819846436867119740019762502364930346877676121253679423200058547956528088$

and

$79258699544783330333470858414800596877379758573642199607343303414557678782818152135381409304740185$

It took time equivalent to more than 55 years on a single modern PC to factor this number.
A public-key encryption scheme is a set of encryption transformations \( \{ E_e : e \in K \} \) and a set of decryption transformations \( \{ D_d : d \in K \} \). For each \( e \in K \) there is a corresponding \( d \in K \) such that \( D_d(E_e(M)) = M \), \( \forall M \). Furthermore, after choosing such a pair \((e, d)\), the public key \( e \) (or the public parameter) is made public, while the associated secret key \( d \) is kept secret. For the scheme to be secure, it must be computationally infeasible to compute \( d \) as well as computing \( E_e^{-1}(C) \), knowing the public value \( e \). Constructed through a set of trapdoor one-way functions \( \{ f_i(x) \} \).
The concepts brought forward in this section can only be realized under assumptions of computational limitations.

We can never have unconditional security in a public-key cryptosystem.

Some assumption is required to be true for the scheme to be secure. We assume that the opponent cannot solve a certain (computationally difficult but solvable) problem. For example, factor a given very large number.
The RSA public-key encryption scheme works as follows. Let \( n = pq \), where \( p \) and \( q \) are two large primes. Let \( \mathcal{M} = \mathcal{C} = \mathbb{Z}_n \). Pick a number \( e \) relatively prime to \( \phi(n) \) and calculate a number \( d \) such that 
\[
ed = 1 \mod \phi(n).
\]
The public key is the two numbers \((n, e)\) and the (public) encryption transformation \( E(M) \) is
\[
E(M) = M^e \mod n.
\]
The secret key is the number \( d \) (as well as \( p, q \) and \( \phi(n) \)) and the secret decryption transformation \( D(C) \) is
\[
D(C) = C^d \mod n.
\]
Verify that decrypting a ciphertext returns the encrypted plaintext.

\[
\text{D}(C) = C^d = (M^e)^d = M^{ed} \mod n.
\]

Now we note that \( ed = 1 \mod \phi(n) \), which means that we can write

\[
ed = 1 + t \cdot \phi(n),
\]

for some integer \( t \). So we can continue

\[
\text{D}(C) = M^{ed} = M^{(1 + t \cdot \phi(n))} = M \cdot M^{t \cdot \phi(n)} \mod n.
\]

From Euler’s formula we know that \( x^{\phi(n)} = 1 \) for any \( x \in \mathbb{Z}_n^* \). So assuming that \( M \) is invertible we have

\[
\text{D}(C) = M \cdot M^{t \cdot \phi(n)} = M \cdot 1 \mod n.
\]
Let $p = 47$ and $q = 167$ be two primes. Their product is $n = pq = 7849$.

Compute $\phi(n) = (p - 1)(q - 1) = 7636$. Next, we must choose a value $e$ such that $\gcd(e, \phi(n)) = 1$. We can check that $e = 25$ is such a value.

When $\gcd(e, \phi(n)) = 1$ we know that $e = 25$ has a multiplicative inverse in $\mathbb{Z}_{\phi(n)}$. We use Euclidean algorithm and Bezout’s lemma to find the inverse $d = 2749$, i.e.,

\[ e \cdot d = 25 \cdot 2749 = 1 \mod 7636. \]

We publish our public key $(n, e) = (7849, 25)$. 

Example

Anyone with access to our public key can now send us an encrypted message $M \in \mathbb{Z}_{7849}$, by calculating

$$C = M^e \mod n.$$  

Alice sends message $M = 2728$. She then computes

$$C = 2728^{25} = 2401 \mod 7849.$$  

When Bob receives the ciphertext $C$, he computes

$$M = C^d = 2401^{2749} \mod n$$

and, as by magic, $M = 2401^{2749} = 2728 \mod n$ so he recover the correct message.
Security of the RSA cryptosystem

If we can factor $n$ then we can compute $\phi(n)$ and $d$.

RSA relies on the factoring problem.

This does not mean that breaking RSA is equivalent to solving a factorization problem. It is not known whether RSA can be broken without factoring $n$. 
Examine the implementation of the RSA operation, i.e., computing

\[ M^e \mod n. \]

We need \( L = \lceil \log_2 n \rceil \) bits to store a value in \( \mathbb{Z}_n \).

A good idea to reduce modulo \( n \) as soon as one receives an intermediate value outside the interval \([0, n - 1]\).
Calculating $2728^3 \mod 7849$ is done by first computing

$$2728^2 = 7441984,$$

then computing

$$7441984 \mod 7849 = 1132.$$

Continuing,

$$1132 \times 2728 = 3088096$$

and finally

$$3088096 \mod 7849 = 3439.$$ 

So $2728^3 \mod 7849 = 3439$. 
Exponentiation

How can we do exponentiation in a fast way, i.e., computing $M^e \mod n$.

The *square and multiply algorithm*:

First rewrite $e$ as a sum of powers of 2, i.e.,

$$e = e_0 + e_1 \cdot 2 + e_2 \cdot 2^2 + \cdots + e_{L-1}2^{L-1},$$

where $e_i \in \{0, 1\}$, $0 \leq i \leq L - 1$. We then perform $L - 1$ squarings of $M$, computing the values

$$M^2, M^4 = (M^2)^2, M^8 = (M^4)^2, \ldots,$$

in $\mathbb{Z}_n$. Finally we perform at most $L - 1$ multiplications by computing $M^e$ as

$$M^e = M^{e_0} \cdot (M^2)^{e_1} \cdot (M^4)^{e_2} \cdots (M^{2^{L-1}})^{e_{L-1}}.$$
Example

Compute $2728^{25}$ in $\mathbb{Z}_n$ using the square and multiply algorithm.

We rewrite 25 as $25 = 16 + 8 + 1$ and get

$$2728^{25} = 2728 \cdot 2728^8 \cdot 2728^{16}.$$  

Performing a number of squarings we get

$$2728^2 = 1132, \quad 2728^4 = 1132^2 = 2037, \quad 2728^8 = 2037^2 = 5097,$$

$$2728^{16} = 5097^2 = 7068.$$  

Performing the multiplications we get

$$2728^{25} = 2728^{16+8+1} = 2728 \cdot 5097 \cdot 7068 = 2401.$$
How to check whether a given number $m$ is a prime or not?

Naive approach: test by trial division, i.e., we check if $x | m$ for all integers $x$, where $2 \leq x \leq \sqrt{m}$.

The problem is that if $m$ is a 1024 bit number then $\sqrt{m}$ is of size 512 bits and $2^{512}$ tests is not possible.
Fermat’s Little Theorem states that

\[ a^{m-1} \equiv 1 \pmod{m}, \]

if \( m \) is a prime and \( a \) such that \( 1 \leq a \leq m - 1 \).

When \( m \) is not a prime we cannot really know what we will receive when computing \( a^{m-1} \).

But if \( a^{m-1} \not\equiv 1 \pmod{m} \) we know for certain that \( m \) is not a prime number.
While \((k \text{ not big enough})\) {

1. Select a random \(a\) and compute \(A = a^{m-1}\).

2. If \(A \neq 1\) we know that \(m\) is composite and we end.

3. \(k = k + 1\)

} 

Assume \(m\) prime.
The error probability after repeating the test \( k \) time would then be less than \( 1/2^k \).

A composite \( m \) such that \( a^{m-1} \equiv 1 \pmod{m} \) is said to be a pseudo-prime to the base \( a \).

If a composite \( m \) is such that it is a pseudo-prime for every base \( a \) with \( \gcd(a, m) = 1 \), the the number is called a Carmichael number.

The smallest Carmichael number is 561.
Miller-Rabin test:

- Randomly choose a base $a$.
- Write $m - 1 = 2^b \cdot q$, where $q$ is odd.
- If $a^q \equiv 1 \pmod{m}$ or $a^{2^c \cdot q} \equiv -1 \pmod{m}$ for any $c < b$ then return “$m$ probably prime” and go back and choose a new base, otherwise return “$m$ definitely not prime”.

One can prove that $P(“m$ probably prime”|$m$ not prime) < 1/4.
The difficulty of factoring?

Some algorithms apply to numbers of special form.
Others are general, i.e., that they can factor any number.
Pollard’s \((p - 1)\)-method

Let \(n = pq\) and let \(p - 1\) factor as

\[ p - 1 = q_1 q_2 \cdots q_k, \]

where each \(q_i\) is a prime power.

Condition: each \(q_i < B\), where \(B\) is a predetermined “bound”.

Algorithm:

Compute \(a = 2^B! \mod n\) and compute the unknown prime \(p\) as

\[ p = \gcd(a - 1, n). \]
Assuming that every unknown prime power $q_i < B$, we have

$$(p - 1)|B!.$$

We compute $a = 2^B! \mod n$. Now let $a' = 2^B! \mod p$. Since $p|n$ we must have $a \mod p = a'$. Fermat’s little theorem states that

$$2^{p-1} = 1 \mod p$$

and since $(p - 1)|B!$ we also have $a' = 1 \mod p$ leading to $a = 1 \mod p$. So $p|(a - 1)$ and since $p|n$ we have $p = \gcd(a - 1, n)$. 
The primes $p$ and $q$ must then be chosen such that $p - 1$ and $q - 1$ each contains a large prime in their factorisation.

A usual approach is to generate a random bprime $p_1$ and then test whether $p = 2p_1 + 1$ is a prime number. If so, we choose $p$. 
Modern factoring algorithms

Quadratic sieve, Number field sieve, elliptic curve factorization, Pollard’s rho algorithm method,...

See project 1.
The computational complexity of factoring

- We often use the function

\[ L_N(\alpha, \beta) = \exp((\beta + o(1))(\log N)^\alpha (\log \log N)^{1-\alpha}). \]

- An algorithm with complexity \( O(L_N(\alpha, \beta)) \) for \( 0 < \alpha < 1 \) is said to have *sub-exponential* behaviour.

- Number Field Sieve: This is currently the most successful method for numbers with more than 100 decimal digits. It can factor numbers of the size of \( 2^{512} \) and has complexity \( L_N(1/3, 1.923) \).

- It is recommended that one takes \( N \) of size around 1024 bits to ensure medium-term security. For long-term security one would need to take a size of over 2048 bits.
Some basic cryptographic primitives:

- **Public key encryption**: A message is encrypted with a recipient’s public key and cannot be decrypted by anyone except the recipient possessing the private key.

- **Digital signatures**: A message signed with a sender’s private key can be verified by anyone who has access to the sender’s public key, thereby proving that the sender signed it and that the message has not been tampered with (authenticity and nonrepudiation).

- **Key exchange**: is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key.
A major problem is linking a public key to an entity or principal,
The main tool in use today is the *digital certificate*. A special trusted third party, called a certificate authority, or CA, is used to vouch for the validity of the public keys.
A CA based system works as follows:

- All users have a trusted copy of the public key of the CA. For example, embedded in your browser when you buy your computer.
- The CA sign data strings containing the following information
  
  (Alice, ..., Alice’s public key, CA’s signature).

This data string, and the associated signature is called a digital certificate. The CA will only sign this data if it truly believes that the public key really does belong to Alice.

- When Alice now sends you her public key, contained in a digital certificate, you now trust that the public key is that of Alice, since you trust the CA and you checked the signature.
The Discrete Logarithm Problem

- Let \((G, \ast)\) be an abelian group.
- **Discrete Logarithm Problem:** Given \(g, h \in G\), find an \(x\) (if it exists) such that \(g^x = h\).
- The difficulty of this problem depends on the group \(G\)
  - Very easy: polynomial time algorithm, e.g. \(\mathbb{Z}_N, +\)
  - Hard: sub-exponential time algorithm, e.g. \(\mathbb{F}_p^*\).
  - Very hard: exponential time algorithm, e.g. elliptic curve groups.
Diffie Hellman Key Exchange

allows two parties to agree a secret key over an insecure channel without having met before.

\[ G = \mathbb{F}_p^* \text{ and } g \in \mathbb{F}_p^* \]

The basic message flows for the Diffie Hellman protocol

- Alice selects secret \( a \), Bob selects secret \( b \).
- Alice sends \( A = g^a \) to Bob. Bob sends \( B = g^b \) to Alice.
- Alice computes \( K = B^a \). Bob computes \( K = A^b \). Note \( B^a = A^b = g^{ab} = K \).

Man in the middle attack
Example DH key exchange

Domain parameters be given by $p = 2147483659$ and $g = 2$.

Alice Bob

\[ a = 12345 \]
\[ b = 654323 \]
\[ A = g^a = 428647416 \]
\[ A = 428647416 \]
\[ B = 450904856 \]
\[ B = g^b = 450904856 \]

Bob computes $A^b = 428647416^{654323} (\mod p) = 1333327162$, Alice computes $B^a = 450904856^{12345} (\mod p), = 1333327162$. 
The basic idea behind public key signatures is as follows:
Message + Alice’s private key = Signature,
Message + Signature + Alice’s public key = YES/NO.

(signature scheme with appendix)

three important security properties:
message integrity the message has not been altered in transit,
message origin the message was sent by Alice,
non-repudiation Alice cannot claim she did not send the message.
A cryptographic hash function $h$ is a function which takes arbitrary length bit strings as input and produces a fixed length bit string as output, the hash value.

- Preimage Resistant: It should be hard to find a message with a given hash value.
- Second Preimage Resistant: Given one message it should be hard to find another message with the same hash value.
- Collision Resistant: It should be hard to find two messages with the same hash value.
Signatures with RSA

Message $m$ for signing, we first compute $h(m)$ and then apply the RSA signing transform to $h(m)$, i.e.

$$s = h(m)^d \mod N.$$ 

Transmit $(m, s)$.

Verifying: Recover $h'$, i.e.

$$h' = s^e \mod N.$$ 

Then compute $h(m)$ from $m$ and check whether $h' = h(m)$. 

If they agree accept the signature as valid, otherwise reject.
Security assumptions for Signatures

- **Key-only attack:** Eve knows the public key and verification algorithm.
- **Known message attack:** Eve knows also a list of signed messages.
- **Chosen message attack:** Eve gets a list of messages of her choice signed.

Adversarial goals:

- **Total break:** Find the secret signing key.
- **Selective forgery:** Try to get a valid signature on a *given* message not signed before.
- **Existential forgery:** Try to get a valid signature on a *any* message not signed before.