Introduction to cryptography

4. Public-key techniques

Gilles Van Assche
Olivier Markowitch

INFO-F-405
Université Libre de Bruxelles
2021-2022

Confidentiality

Encryption

- plaintext $\Rightarrow$ ciphertext
- Under key $k_E \in K$

Decryption

- ciphertext $\Rightarrow$ plaintext
- Under key $k_D \in K$

Symmetric cryptography: $k_E = k_D$ is the secret key.

Asymmetric cryptography: k_E is public and k_D is private.

Authenticity

Authentication

- message $\Rightarrow$ (message, **tag**)
- Under key $k_{\mathsf{A}} \in K$

Verification

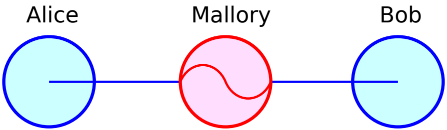
- (message, **tag**) $\Rightarrow$ {message, $\perp$}
- Under key $k_{\mathsf{V}} \in K$

Symmetric cryptography: $k_{\mathsf{A}} = k_{\mathsf{V}}$ is the secret key.
The tag is called a *message authentication code* (MAC).

Asymmetric cryptography: k_{A} is private and k_{V} is public.
The tag is called a *signature*.

Public keys

Problem: man-in-the-middle attack



[By Miraceti on Wikipedia]

- Alice gets Mallory's public key, thinking it is Bob's
- Bob gets Mallory's public key, thinking it is Alice's
- Mallory decrypts and re-encrypts traffic at will

Binding a public key to an identity

- certificates and public-key infrastructure (PKI)
- web of trust (GPG, GnuPG)

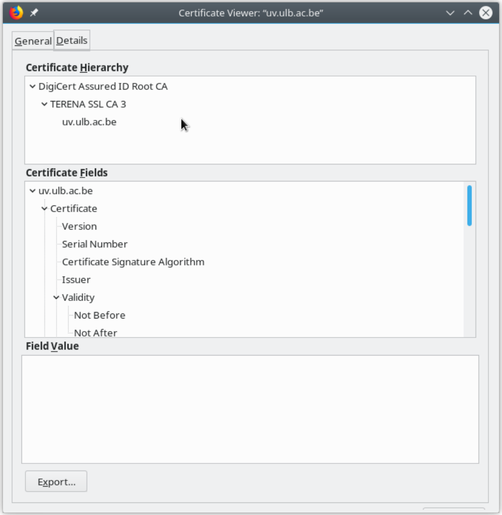
Certificates

A **public key certificate**

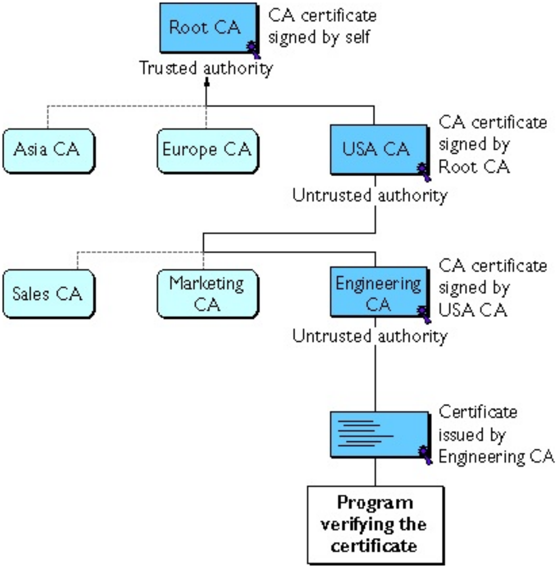
- allows to bind a public key with the identity of its owner
- contains at least the public key, the information allowing to identify its owner and a digital signature on the key and the information
- is signed by a certification authority

Example of certificate

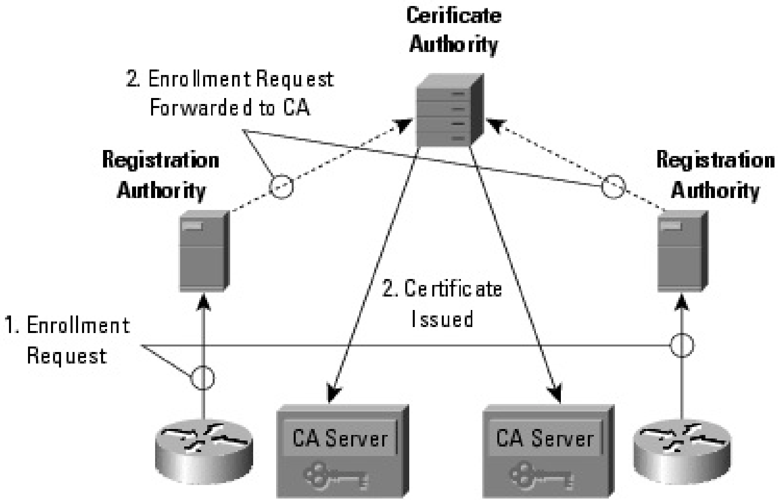
Example of certificate



Certificate hierarchy



Public key infrastructure



Web of trust

Manual key verification over an authentic channel

- compare the fingerprint (hash function)
- if correct, sign it

Public keys are distributed with their signatures.

A key is trusted

- if the key is signed by me, or
- if the key is signed by someone I trust.

Different kinds of public-key algorithms

Depending on the problem on which they are based:

- factorization
- discrete logarithm problem
  - modular exponentiation
  - elliptic curves
- lattices
- error correcting codes
- multivariate polynomials
- hash functions
- isogeny graphs of elliptic curves
- ...

In practice: hybrid encryption

To improve both efficiency and bandwidth we can combine asymmetric encryption with symmetric encryption.

Alice computes the encryption c of the message m intended for Bob in the following way:

1. Alice chooses randomly the symmetric key k;
2. Alice computes $(c_k, c_m) = \left(\text{Enc}_{\text{Bob's public key}}(k), \text{Enc}_k(m)\right)$.

Bob decrypts as follows:

1. Bob recovers $k = \text{Dec}_{\text{Bob's private key}}(c_k)$
2. Bob recovers $m = \text{Dec}_k(c_m)$

Primes

There are infinitely many primes

Suppose that $p_1 = 2 < p_2 = 3 < ... < p_r$ are all the existing primes
Let $P = p_1 p_2 ... p_r + 1$
P cannot be a prime and then let $p \neq 1$ be one of the existing primes that divides P
But p cannot be any of p_1, p_2, ..., p_r, because otherwise p would divide the difference $P - p_1 p_2 ... p_r = 1$, and p would be equal 1
So this prime p is still another prime, and p_1, p_2, ..., p_r would not be all of the existing primes

Groups

A group is a set G along with a binary operation $\circ$ that satisfy the following properties:

- **closure**: $\forall g, h \in G$, $g \circ h \in G$
- **associativity**: $\forall g_1, g_2, g_3 \in G$, $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$
- **identity**: there exists an identity $e \in G$ such that $\forall g \in G$, $e \circ g = g \circ e = g$
- **inverse**: $\forall g \in G$, there exists $h \in G$ such that $g \circ h = h \circ g = e$

Order of an element in a finite group

The **order of an element** a **of a group** G is the smallest positive integer $m = \mathrm{ord}(a)$ such that $[m]a = e$, where

- e denotes the identity element of the group and
- $[m]a = a \circ a \circ \cdots \circ a$ denotes the group operation applied to m copies of the element a.

Theorem: For any a, its order $\mathrm{ord}(a)$ divides the size $|G|$ of the group.

An element g is said to be a **generator** of G if $\mathrm{ord}(g) = |G|$. (A generator can be used to generate all the elements of the group.)

Order of an element in a finite group

The **order of an element** a **of a group** G is the smallest positive integer $m = \mathrm{ord}(a)$ such that $[m]a = e$, where

- e denotes the identity element of the group and
- $[m]a = a \circ a \circ \cdots \circ a$ denotes the group operation applied to m copies of the element a.

Theorem: For any a, its order $\mathrm{ord}(a)$ divides the size $|G|$ of the group.

An element g is said to be a **generator** of G if $\mathrm{ord}(g) = |G|$. (A generator can be used to generate all the elements of the group.)

Order of an element in a finite group

The **order of an element** a **of a group** G is the smallest positive integer $m = \mathrm{ord}(a)$ such that $[m]a = e$, where

- e denotes the identity element of the group and
- $[m]a = a \circ a \circ \cdots \circ a$ denotes the group operation applied to m copies of the element a.

Theorem: For any a, its order $\mathrm{ord}(a)$ divides the size $|\mathrm{G}|$ of the group.

An element g is said to be a **generator** of G if $\mathrm{ord}(g) = |\mathrm{G}|$. (A generator can be used to generate all the elements of the group.)

Modulo

What is **modulo**?

- A binary operation: $a \bmod n = r$ iff $a = qn + r$ and $0 \leq r < n$
- An equivalence relation: $a \equiv b \pmod{n}$
  - iff there exists an integer k such that $(a - b) = kn$
  - iff $(a \bmod n) = (b \bmod n)$

What is **modular arithmetic?** It is arithmetic operations where all that matters is equivalence modulo some fixed integer n.

Modular additions and subtractions

$\mathbb{Z}_n, +$ is the set $\{0, 1, \ldots, n-1\}$ together with addition modulo n

$\mathbb{Z}_n, +$ is a group, with

- the identity is 0
- the inverse of x is $-x \bmod n$

1 is a generator of the group because all the elements can be represented by adding 1 to itself an approriate number of times.

Greatest common dividor

$\gcd(a, b)$ denotes the greatest common divisor of a and b

Bezout: For any positive integers a and b, there exist integers x and y such that $ax + by = \gcd(a, b)$.

Let $S = \{ax + by \ : \ x, y \in \mathbb{Z}\}$ and $d = \min(S \cap \mathbb{N}_{>0})$
Divide a by d: $a = qd + r$ with $0 \leq r < d$
But $r = a - qd = a - q(ax + by) = (1 - qx)a - (qy)b$ so $r \in S$ and $r < d$ hence $r = 0$ and d divides a
Similarly, d divides b and therefore d divides $\gcd(a, b)$
However $\gcd(a, b)$ divides all elements of S, so $d = \gcd(a, b)$

Corollary: a and b are relatively prime (or equivalently $\gcd(a, b) = 1$) iff there exist integers x and y such that $ax + by = 1$.

Greatest common dividor

$\gcd(a, b)$ denotes the greatest common divisor of a and b

Bezout: For any positive integers a and b, there exist integers x and y such that $ax + by = \gcd(a, b)$.

Let $S = \{ax + by \ : \ x, y \in \mathbb{Z}\}$ and $d = \min(S \cap \mathbb{N}_{>0})$
Divide a by d: $a = qd + r$ with $0 \leq r < d$
But $r = a - qd = a - q(ax + by) = (1 - qx)a - (qy)b$ so $r \in S$ and $r < d$ hence $r = 0$ and d divides a
Similarly, d divides b and therefore d divides $\gcd(a, b)$
However $\gcd(a, b)$ divides all elements of S, so $d = \gcd(a, b)$

Corollary: a and b are relatively prime (or equivalently $\gcd(a, b) = 1$) iff there exist integers x and y such that $ax + by = 1$.

Greatest common dividor

$\gcd(a, b)$ denotes the greatest common divisor of a and b

Bezout: For any positive integers a and b, there exist integers x and y such that $ax + by = \gcd(a, b)$.

Let $S = \{ax + by \ : \ x, y \in \mathbb{Z}\}$ and $d = \min(S \cap \mathbb{N}_{>0})$
Divide a by d: $a = qd + r$ with $0 \leq r < d$
But $r = a - qd = a - q(ax + by) = (1 - qx)a - (qy)b$ so $r \in S$ and $r < d$ hence $r = 0$ and d divides a
Similarly, d divides b and therefore d divides $\gcd(a, b)$
However $\gcd(a, b)$ divides all elements of S, so $d = \gcd(a, b)$

Corollary: a and b are relatively prime (or equivalently $\gcd(a, b) = 1$) iff there exist integers x and y such that $ax + by = 1$.

Modular multiplications and multiplicative inversion

$\mathbb{Z}_n^*, \times$ is the set $\{x : 0 < x < n \text{ and } \gcd(x, n) = 1\}$ together with multiplication modulo n

$\mathbb{Z}_n^*, \times$ is a group, with

- the identity is 1
- the inverse of x is denoted $x^{-1} \bmod n$ and is such that $x^{-1}x \equiv 1 \pmod{n}$

The inverse $x^{-1} \bmod n$ exists and is unique iff $\gcd(x, n) = 1$

- $xy \equiv 1 \pmod{n}$ means that $xy = 1 + kn$ for some integer k, so $xy + kn = 1$ and $\gcd(x, n) = 1$
- $\gcd(x, n) = 1$ implies that there exist integers y and z such that $yx + zn = 1$, so $xy = 1 - zn$ and $xy \equiv 1 \pmod{n}$

Modular multiplications and multiplicative inversion

$\mathbb{Z}_n^*, \times$ is the set $\{x \,:\, 0 < x < n \text{ and } \gcd(x, n) = 1\}$ together with multiplication modulo n

$\mathbb{Z}_n^*, \times$ is a group, with

- the identity is 1
- the inverse of x is denoted $x^{-1} \bmod n$ and is such that $x^{-1}x \equiv 1 \pmod{n}$

The inverse $x^{-1} \bmod n$ exists and is unique iff $\gcd(x, n) = 1$

- $xy \equiv 1 \pmod{n}$ means that $xy = 1 + kn$ for some integer k, so $xy + kn = 1$ and $\gcd(x, n) = 1$
- $\gcd(x, n) = 1$ implies that there exist integers y and z such that $yx + zn = 1$, so $xy = 1 - zn$ and $xy \equiv 1 \pmod{n}$

Euler's $\Phi(n)$ function

We note $\Phi(n)$ the number of integers smaller than n and that are relatively prime with n. So $|\mathbb{Z}_n^*| = \Phi(n)$.

If $n = \prod_{i=1}^{r} p_i^{e_i}$ for distinct primes $p_1, \ldots, p_r$, then

$$\Phi(n) = n \cdot \prod_{i=1}^{r} \left(1 - \frac{1}{p_i}\right)$$

Particular cases:

- Prime p: $\Phi(p) = p - 1$
- Product of two distinct primes p and q: $\Phi(pq) = (p-1)(q-1)$

Euler's $\Phi(n)$ function

We note $\Phi(n)$ the number of integers smaller than n and that are relatively prime with n. So $|\mathbb{Z}_n^*| = \Phi(n)$.

If $n = \prod_{i=1}^{r} p_i^{e_i}$ for distinct primes $p_1, \ldots, p_r$, then

$$\Phi(n) = n \cdot \prod_{i=1}^{r} \left(1 - \frac{1}{p_i}\right)$$

Particular cases:

- Prime p: $\Phi(p) = p - 1$
- Product of two distinct primes p and q: $\Phi(pq) = (p-1)(q-1)$

Fermat and Euler's theorems

Fermat's Little Theorem: let p be a prime and a an integer not a multiple of p, then $a^{p-1} \equiv 1 \pmod{p}$.

Euler's Theorem: let a and n two relatively prime integers, then $a^{\Phi(n)} \equiv 1 \pmod{n}$.

As a consequence, the exponent can be reduced modulo $\Phi(n)$:

$$a^e \equiv a^{e \bmod \Phi(n)} \pmod{n}$$

Fermat and Euler's theorems

Fermat's Little Theorem: let p be a prime and a an integer not a multiple of p, then $a^{p-1} \equiv 1 \pmod{p}$.

Euler's Theorem: let a and n two relatively prime integers, then $a^{\Phi(n)} \equiv 1 \pmod{n}$.

As a consequence, the exponent can be reduced modulo $\Phi(n)$:

$$a^e \equiv a^{e \bmod \Phi(n)} \pmod{n}$$

Generator in $\mathbb{Z}_n^*$

The group $\mathbb{Z}_n^*$ has size $\Phi(n)$, hence a generator g is an element with $\mathrm{ord}(g) = \Phi(n)$. Such a generator exists when n is 2, 4, p^a or $2p^a$, with p an odd prime.

Example: consider $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$.

- $\mathrm{ord}(1) = 1 : 1^1 = 1$
- $\mathrm{ord}(2) = 3 : 2^1 = 2,\ 2^2 = 4,\ 2^3 = 1$
- $\mathrm{ord}(\mathbf{3}) = 6 : 3^1 = 3,\ 3^2 = 2,\ 3^3 = 6,\ 3^4 = 4,\ 3^5 = 5,\ 3^6 = 1$
- $\mathrm{ord}(4) = 3 : 4^1 = 4,\ 4^2 = 2,\ 4^3 = 1$
- $\mathrm{ord}(\mathbf{5}) = 6 : 5^1 = 5,\ 5^2 = 4,\ 5^3 = 6,\ 5^4 = 2,\ 5^5 = 3,\ 5^6 = 1$
- $\mathrm{ord}(6) = 2 : 6^1 = 6,\ 6^2 = 1$

Generator in $\mathbb{Z}_n^*$

The group $\mathbb{Z}_n^*$ has size $\Phi(n)$, hence a generator g is an element with $\text{ord}(g) = \Phi(n)$. Such a generator exists when n is 2, 4, p^a or $2p^a$, with p an odd prime.

Example: consider $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$.

- $\text{ord}(1) = 1 : 1^1 = 1$
- $\text{ord}(2) = 3 : 2^1 = 2,\ 2^2 = 4,\ 2^3 = 1$
- $\text{ord}(\mathbf{3}) = 6 : 3^1 = 3,\ 3^2 = 2,\ 3^3 = 6,\ 3^4 = 4,\ 3^5 = 5,\ 3^6 = 1$
- $\text{ord}(4) = 3 : 4^1 = 4,\ 4^2 = 2,\ 4^3 = 1$
- $\text{ord}(\mathbf{5}) = 6 : 5^1 = 5,\ 5^2 = 4,\ 5^3 = 6,\ 5^4 = 2,\ 5^5 = 3,\ 5^6 = 1$
- $\text{ord}(6) = 2 : 6^1 = 6,\ 6^2 = 1$

The Chinese remainder theorem (CRT)

Let $\{m_1, \ldots, m_k\}$ be a set of relatively prime integers, i.e., $\forall 1 \leq i \neq j \leq k, \gcd(m_i, m_j) = 1$, and let m be their product $m = m_1 \times \cdots \times m_k$.

Then the following system of equations

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ \vdots \\ x \equiv a_k \pmod{m_k} \end{cases}$$

has one and only one solution modulo m

Solving a CRT problem

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ \vdots \\ x \equiv a_k \pmod{m_k} \end{cases}$$

Recipe:

- Compute $M_i = m/m_i$, and notice that
  - $\gcd(M_i, m_i) = 1$
  - $M_i \equiv 0 \pmod{m_j}$ for any $j \neq i$
- Compute $c_i = M_i^{-1} \bmod m_i$
- Let
$$x = \sum_i a_i c_i M_i$$

Indeed, $x \equiv a_i c_i M_i \equiv a_i \pmod{m_i}$.

RSA

Ronald Rivest, Adi Shamir and Leonard Adleman

RSA key generation

The user generates a public-private key pair as follows:

- Privately generate two large distinct primes p and q
- Choose a public exponent $3 \leq e \leq (p-1)(q-1)-3$
  - it must satisfy $\gcd(e,(p-1)(q-1)) = 1$
  - often, one chooses $e \in \{3, 17, 2^{16}+1\}$ then generates the primes
- Compute the private exponent $d = e^{-1} \bmod (p-1)(q-1)$
- Compute the public modulus $n = pq$ (and discard p and q)

The public key is (n, e) and the private key is (n, d).

RSA "textbook encryption" (don't use!)

From plaintext $m \in \mathbb{Z}_n$ to ciphertext $c \in \mathbb{Z}_n$ and back:

- **Encryption:** $c = m^e \bmod n$
- **Decryption:** $m = c^d \bmod n$

Why is it correct?

$$
\begin{aligned}
c^d &\equiv (m^e)^d \equiv m^{ed} \\
&\equiv m^{ed \bmod \Phi(n)} && \text{by Euler's theorem} \\
&\equiv m^{ed \bmod (p-1)(q-1)} && \text{since } p \text{ and } q \text{ are distinct primes} \\
&\equiv m^1 && \text{by definition of } e \text{ and } d \\
&\equiv m \pmod n
\end{aligned}
$$

RSA "textbook encryption" (don't use!)

From plaintext $m \in \mathbb{Z}_n$ to ciphertext $c \in \mathbb{Z}_n$ and back:

- **Encryption:** $c = m^e \bmod n$
- **Decryption:** $m = c^d \bmod n$

Why is it correct?

$$\begin{aligned}
c^d &\equiv (m^e)^d \equiv m^{ed} \\
&\equiv m^{ed \bmod \Phi(n)} && \text{by Euler's theorem} \\
&\equiv m^{ed \bmod (p-1)(q-1)} && \text{since } p \text{ and } q \text{ are distinct primes} \\
&\equiv m^1 && \text{by definition of } e \text{ and } d \\
&\equiv m \pmod n
\end{aligned}$$

RSA "textbook signature" (don't use!)

From message $m \in \mathbb{Z}_n$ to signature $s \in \mathbb{Z}_n$ and back:

- **Signature:** Send (m, s), with $s = m^d \bmod n$
- **Verification:** Check $m \stackrel{?}{=} s^e \bmod n$

RSA and factorization

If one can factor n into $p \times q$, the private exponent d follows immediately. Conversely, from the knowledge of d, it is easy to factor n.

Proof: $ed \equiv 1 \pmod{\Phi(n)}$, so for any $a \in \mathbb{Z}_n^*$ we have $a^{ed-1} \equiv 1 \pmod{n}$. As $\Phi(n)$ is even, so is $ed - 1 = 2t$ and $a^{2t} \equiv 1 \pmod{n}$.

Define $z = a^t \bmod n$, so that $z^2 \equiv 1 \pmod{n}$. Assume $z \bmod n \neq \pm 1$ (otherwise change a). Hence n divides $z^2 - 1 = (z-1)(z+1)$.

Let $g_1 = \gcd(z-1, n)$ and $g_2 = \gcd(z+1, n)$. $g_1 \neq n \neq g_2$, as we assumed $z \bmod n \neq \pm 1$. Both g_1 and g_2 cannot be equal to 1 since $z^2 - 1$ is a multiple of n.

Hence g_1 or g_2 is p or q.

RSA and factorization

If one can factor n into $p \times q$, the private exponent d follows immediately. Conversely, from the knowledge of d, it is easy to factor n.

Proof: $ed \equiv 1 \pmod{\Phi(n)}$, so for any $a \in \mathbb{Z}_n^*$ we have $a^{ed-1} \equiv 1 \pmod{n}$. As $\Phi(n)$ is even, so is $ed - 1 = 2t$ and $a^{2t} \equiv 1 \pmod{n}$.

Define $z = a^t \bmod n$, so that $z^2 \equiv 1 \pmod{n}$. Assume $z \bmod n \neq \pm 1$ (otherwise change a). Hence n divides $z^2 - 1 = (z-1)(z+1)$.

Let $g_1 = \gcd(z-1, n)$ and $g_2 = \gcd(z+1, n)$. $g_1 \neq n \neq g_2$, as we assumed $z \bmod n \neq \pm 1$. Both g_1 and g_2 cannot be equal to 1 since $z^2 - 1$ is a multiple of n.

Hence g_1 or g_2 is p or q.

RSA and factorization

There is no known polynomial time algorithm to factor an integer (polynomial in the size of the integer).

But there exist subexponential algorithms. The currently best known algorithm (general number field sieve) factors an integer n asymptotically in time

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}\right).$$

For 128-bit security, NIST recommends n to be at least 3072-bit long (hence p and q are at least 1536-bit long).

[NIST SP 800-57, see also `https://www.keylength.com/`]

RSA and factorization

There is no known polynomial time algorithm to factor an integer (polynomial in the size of the integer).

But there exist subexponential algorithms. The currently best known algorithm (general number field sieve) factors an integer n asymptotically in time

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}\right).$$

For 128-bit security, NIST recommends n to be at least 3072-bit long (hence p and q are at least 1536-bit long).
[NIST SP 800-57, see also `https://www.keylength.com/`]

Some other attacks against RSA

- Cyclic attack
- Message factorization
- Short message attack

Short message attack: If e is small (e.g., $e = 3$), and if $m < \sqrt[e]{n}$, then $m^e \bmod n = m^e$ without any modular reduction. Therefore m is retrieved by simply computing $\sqrt[e]{c}$ over the integers!

Example: with $e = 3$ and n on 3072 bits, the attack works until m is a 1024-bit integer. (Problematic in the case of hybrid encryption, with m a secret key for symmetric encryption.)

Some other attacks against RSA

- Cyclic attack
- Message factorization
- Short message attack

Short message attack: If e is small (e.g., $e = 3$), and if $m < \sqrt[e]{n}$, then $m^e \bmod n = m^e$ without any modular reduction. Therefore m is retrieved by simply computing $\sqrt[e]{c}$ over the integers!

Example: with $e = 3$ and n on 3072 bits, the attack works until m is a 1024-bit integer. (Problematic in the case of hybrid encryption, with m a secret key for symmetric encryption.)

Some other attacks against RSA

- Cyclic attack
- Message factorization
- Short message attack

Short message attack: If e is small (e.g., $e = 3$), and if $m < \sqrt[e]{n}$, then $m^e \bmod n = m^e$ without any modular reduction. Therefore m is retrieved by simply computing $\sqrt[e]{c}$ over the integers!

Example: with $e = 3$ and n on 3072 bits, the attack works until m is a 1024-bit integer. (Problematic in the case of hybrid encryption, with m a secret key for symmetric encryption.)

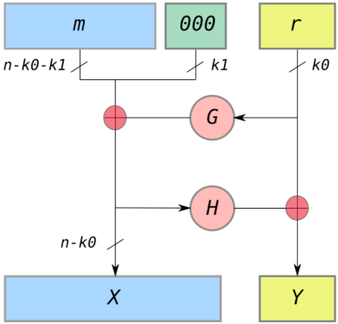
Some other attacks against RSA

- Cyclic attack
- Message factorization
- Short message attack

Short message attack: If e is small (e.g., $e = 3$), and if $m < \sqrt[e]{n}$, then $m^e \bmod n = m^e$ without any modular reduction. Therefore m is retrieved by simply computing $\sqrt[e]{c}$ over the integers!

Example: with $e = 3$ and n on 3072 bits, the attack works until m is a 1024-bit integer. (Problematic in the case of hybrid encryption, with m a secret key for symmetric encryption.)

RSA-OAEP: Optimal asymmetric encryption padding



[Mihir Bellare and Phillip Rogaway, Eurocrypt 1994]

Encryption: m ... then $c = (X \| Y)^e \bmod n$
Decryption: $(X \| Y) = c^d \bmod n$, then ... m

RSA-OAEP

- makes the scheme probabilistic;
- prevents that an adversary recovers any portion of the plaintext without being able to invert RSA;
- is shown to be IND-CCA secure, assuming that G and H behave as random oracles.

RSA-KEM: Key encapsulation method

In hybrid encryption, Alice does not have to choose the secret key, but she can let it be derived from some random bits.

To encapsulate, Alice does the following:

1. Alice chooses a random m of the same bit size as n
2. Alice encrypts $c = m^e \bmod n$, and she sends c to Bob
3. Alice computes $k = \text{hash}(m)$

To decapsulate, Bob does the following:

1. Bob recovers $m = c^d \bmod n$
2. Bob computes $k = \text{hash}(m)$

RSA-KEM: Key encapsulation method

In hybrid encryption, Alice does not have to choose the secret key, but she can let it be derived from some random bits.

To encapsulate, Alice does the following:

1. Alice chooses a random m of the same bit size as n
2. Alice encrypts $c = m^e \bmod n$, and she sends c to Bob
3. Alice computes $k = \text{hash}(m)$

To decapsulate, Bob does the following:

1. Bob recovers $m = c^d \bmod n$
2. Bob computes $k = \text{hash}(m)$

RSA-KEM: Key encapsulation method

In hybrid encryption, Alice does not have to choose the secret key, but she can let it be derived from some random bits.

To encapsulate, Alice does the following:

1. Alice chooses a random m of the same bit size as n
2. Alice encrypts $c = m^e \bmod n$, and she sends c to Bob
3. Alice computes $k = \mathrm{hash}(m)$

To decapsulate, Bob does the following:

1. Bob recovers $m = c^d \bmod n$
2. Bob computes $k = \mathrm{hash}(m)$

RSA: how *not* to sign

Recall the "textbook signature":

- **Signature:** Send (m, s), with $s = m^d \bmod n$
- **Verification:** Check $m \stackrel{?}{=} s^e \bmod n$

Forgery attack: (exploiting the multiplicative structure)

- Ask for the signature of m_1: $s_1 = m_1^d \bmod n$
- Ask for the signature of m_2: $s_2 = m_2^d \bmod n$
- Compute $m_3 = m_1 \times m_2 \bmod n$ and $s_3 = s_1 \times s_2 \bmod n$
- Submit (m_3, s_3) for verification:

$$\begin{aligned} m_3 &\equiv m_1 m_2 \\ &\equiv s_1^e s_2^e \\ &\equiv s_3^e \pmod{n} \end{aligned}$$

RSA: how *not* to sign

Recall the "textbook signature":

- **Signature:** Send (m, s), with $s = m^d \bmod n$
- **Verification:** Check $m \stackrel{?}{=} s^e \bmod n$

Forgery attack: (exploiting the multiplicative structure)

- Ask for the signature of m_1: $s_1 = m_1^d \bmod n$
- Ask for the signature of m_2: $s_2 = m_2^d \bmod n$
- Compute $m_3 = m_1 \times m_2 \bmod n$ and $s_3 = s_1 \times s_2 \bmod n$
- Submit (m_3, s_3) for verification:

$$\begin{aligned} m_3 &\equiv m_1 m_2 \\ &\equiv s_1^e s_2^e \\ &\equiv s_3^e \pmod{n} \end{aligned}$$

RSA with message recovery

If the message m is short enough, we can embed it in the signature.

Let $R(m) \to m'$ be a redundancy function from message space M to range $\mathcal{R} \subset \mathbb{Z}_n$.

- **Signature:** Compute $m' = R(m)$ then send $s = (m')^d \bmod n$
- **Verification:**
  - Recover $m' = s^e \bmod n$
  - If $m' \in \mathcal{R}$, return $m = R^{-1}(m')$ and accept it.
  - Otherwise, reject it.

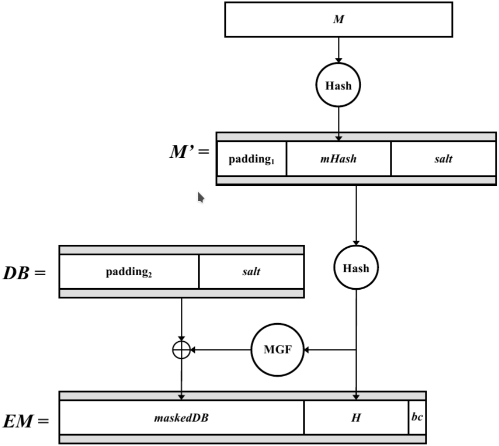
Not used much in practice.

RSA with full-domain hashing

Let H be an extendable output function (or take an old-style hash function and use MGF1).

- **Signature:**
  - Compute $h = H(m)$ so that h has the same bit size as n
  - Send (m, s), with $s = h^d \bmod n$
- **Verification:**
  - Compute $h = H(m)$ like above
  - Check $h \stackrel{?}{=} s^e \bmod n$

RSA with probabilistic signature scheme (PSS)



[PKCS #1 v2.2] [Bellare and Rogaway]

Signature:

- From m and random *salt*, compute EM
- Send (m, s), with $s = (EM)^d \bmod n$

Verification:

- Recover $EM = s^e \bmod n$
- Recover *salt*
- Check $H \stackrel{?}{=} \text{hash}(M')$

How to find large primes?

The number of primes up to n is $\pi(n)$ and for large n:

$$\pi(n) \sim \frac{n}{\ln n}.$$

The average prime gap for numbers of b bits is about $b \ln 2$.

Recipe:

- Draw a random number n
- Test co-primality with the first few primes $2, 3, 5, \ldots$
- Test **pseudo**-primality with, e.g., Miller-Rabin
- Otherwise, increment n and repeat

How to find large primes?

The number of primes up to n is $\pi(n)$ and for large n:

$$\pi(n) \sim \frac{n}{\ln n}.$$

The average prime gap for numbers of b bits is about $b \ln 2$.

Recipe:

- Draw a random number n
- Test co-primality with the first few primes $2, 3, 5, \ldots$
- Test **pseudo**-primality with, e.g., Miller-Rabin
- Otherwise, increment n and repeat

Exponentiation using square and multiply

To compute $a^e \bmod n$, write the exponent in binary and apply the **square and multiply** algorithm. Reduce the numbers modulo n as you compute.

Example: $26 = 11010_2$

a	initialization 1
$a \to a^2 \times a = a^3$	square and multiply 1
$a^3 \to (a^3)^2 = a^6$	square 0
$a^6 \to (a^6)^2 \times a = a^{13}$	square and multiply 1
$a^{13} \to (a^{13})^2 = a^{26}$	square 0

Optimization using the CRT

To speed up the decryption or signature generation, keep p and q and use the Chinese remainder theorem.

Instead of computing $m = c^d \bmod n$, compute:

$$m_p = c^d \bmod p = c^{d \bmod (p-1)} \bmod p$$
$$m_q = c^d \bmod q = c^{d \bmod (q-1)} \bmod q$$

Then recombine:

$$m = (m_p - m_q)(p^{-1} \bmod q)p + m_p$$

What is the discrete logarithm problem?

Domain parameters:

- Let p be a large prime.
- Let g be a generator of $\mathbb{Z}_p^*$, i.e., $\{g^i \bmod p \,:\, i \in \mathbb{N}\} = \mathbb{Z}_p^*$.

Discrete logarithm problem (DLP)

Given $A = g^a \bmod p$, find a.

There are currently no known polynomial time algorithm to solve this problem.

For 128-bit security, NIST recommends p to be at least 3072-bit long.
[NIST SP 800-57, see also `https://www.keylength.com/`]

What is the discrete logarithm problem?

Domain parameters:

- Let p be a large prime.
- Let g be a generator of $\mathbb{Z}_p^*$, i.e., $\{g^i \bmod p \,:\, i \in \mathbb{N}\} = \mathbb{Z}_p^*$.

Discrete logarithm problem (DLP)

Given $A = g^a \bmod p$, find a.

There are currently no known polynomial time algorithm to solve this problem.

For 128-bit security, NIST recommends p to be at least 3072-bit long.
[NIST SP 800-57, see also `https://www.keylength.com/`]

What is the discrete logarithm problem?

Domain parameters:

- Let p be a large prime.
- Let g be a generator of $\mathbb{Z}_p^*$, i.e., $\{g^i \bmod p \ : \ i \in \mathbb{N}\} = \mathbb{Z}_p^*$.

Discrete logarithm problem (DLP)

Given $A = g^a \bmod p$, find a.

There are currently no known polynomial time algorithm to solve this problem.

For 128-bit security, NIST recommends p to be at least 3072-bit long.
[NIST SP 800-57, see also `https://www.keylength.com/`]

Key generation

Domain parameters (public, common to all users):

- Let p be a large prime.
- Let g be a generator of $\mathbb{Z}_p^*$, i.e., $\{g^i \bmod p \;:\; i \in \mathbb{N}\} = \mathbb{Z}_p^*$.

The user generates a public-private key pair as follows:

- Privately choose a random integer $a \in [1, p-2]$
- Compute $A = g^a \bmod p$

The public key is A and the private key is a.

Key generation (group notation)

Domain parameters (public, common to all users):

- Let G be a group.
- Let $g \in G$ be a generator of G of order $q = |G|$.

The user generates a public-private key pair as follows:

- Privately choose a random integer $a \in [1, q-1]$
- Compute $A = [a]g$

The public key is A and the private key is a.

ElGamal encryption

Taher ElGamal

ElGamal encryption

Encryption of $m \in \mathbb{Z}_p^*$ with Alice's public key A:

- Choose randomly an integer $k \in [1, p-2]$
- Compute

$$\begin{cases} K = g^k \bmod p \\ c = mA^k \bmod p \end{cases}$$

Decryption of (K, c) by Alice with her private key a:

$$m = K^{-a} c \bmod p$$

Why is it correct?

$$K^{-a} \equiv (g^k)^{-a} \equiv (g^a)^{-k} \equiv (A^k)^{-1} \pmod p$$
$$(A^k)^{-1} c \equiv (A^k)^{-1} m (A^k) \equiv m \pmod p$$

ElGamal encryption

Encryption of $m \in \mathbb{Z}_p^*$ with Alice's public key A:

- Choose randomly an integer $k \in [1, p-2]$
- Compute

$$\begin{cases} K = g^k \bmod p \\ c = mA^k \bmod p \end{cases}$$

Decryption of (K, c) by Alice with her private key a:

$$m = K^{-a} c \bmod p$$

Why is it correct?

$$K^{-a} \equiv (g^k)^{-a} \equiv (g^a)^{-k} \equiv (A^k)^{-1} \pmod p$$
$$(A^k)^{-1} c \equiv (A^k)^{-1} m (A^k) \equiv m \pmod p$$

ElGamal encryption

Encryption of $m \in \mathbb{Z}_p^*$ with Alice's public key A:

- Choose randomly an integer $k \in [1, p-2]$
- Compute

$$\begin{cases} K = g^k \bmod p \\ c = mA^k \bmod p \end{cases}$$

Decryption of (K, c) by Alice with her private key a:

$$m = K^{-a} c \bmod p$$

Why is it correct?

$$K^{-a} \equiv (g^k)^{-a} \equiv (g^a)^{-k} \equiv (A^k)^{-1} \pmod p$$
$$(A^k)^{-1} c \equiv (A^k)^{-1} m (A^k) \equiv m \pmod p$$

Security of ElGamal encryption

K and k can be seen as an **ephemeral key** pair, created by the sender. K is part of the ciphertext, and k is protected by the DLP.

Caution: k must be secret and randomly drawn independently at each encryption!

- If k is known, one can compute A^k and recover m from c.
- If k is repeated to encrypt, say, m_1 and m_2, then we have

$$c_1 = m_1 A^k \bmod p \text{ and } c_2 = m_2 A^k \bmod p$$

and thus

$$c_1 c_2^{-1} \equiv m_1 m_2^{-1} \pmod p$$

Security of ElGamal encryption

K and k can be seen as an **ephemeral key** pair, created by the sender. K is part of the ciphertext, and k is protected by the DLP.

Caution: k must be secret and randomly drawn independently at each encryption!

- If k is known, one can compute A^k and recover m from c.
- If k is repeated to encrypt, say, m_1 and m_2, then we have

$$c_1 = m_1 A^k \bmod p \text{ and } c_2 = m_2 A^k \bmod p$$

and thus

$$c_1 c_2^{-1} \equiv m_1 m_2^{-1} \pmod{p}$$

The Diffie-Hellman problem

Domain parameters:

- Let p be a large prime.
- Let g be a generator of $\mathbb{Z}_p^*$, i.e., $\{g^i \bmod p \;:\; i \in \mathbb{N}\} = \mathbb{Z}_p^*$.

Diffie-Hellman problem

Given $X = g^x \bmod p$ and $Y = g^y \bmod p$, find $X^y = Y^x = g^{xy} \bmod p$.

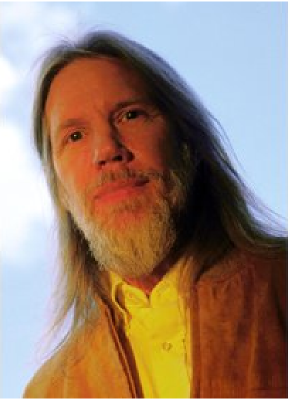
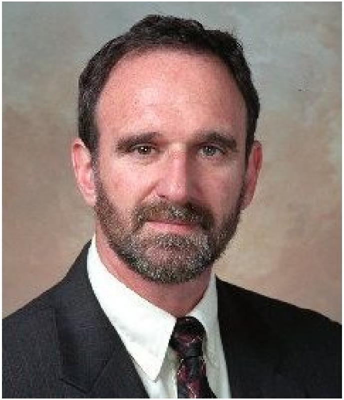
It is an easier problem than DLP (breaking DHP does not give you the exponents). However, there are currently no known polynomial time algorithm to solve the DHP either.

Security of ElGamal encryption

To recover k or a from K or $A \Rightarrow$ DLP.

But to break ElGamal, it is sufficient to recover $K^a = A^k = g^{ak} \bmod p$.
Hence, ElGamal encryption relies on the DHP.

Diffie-Hellman key agreement

Whitfield Diffie and Martin Hellman

Diffie-Hellman key agreement

Hybrid encryption: we do not need to choose the secret key!

Domain parameters (p, g) and

- Alice's key pair $A = g^a \bmod p$
- Bob's key pair $B = g^b \bmod p$

Alice computes		Bob computes
$K_{AB} = B^a \bmod p$		$K_{AB} = A^b \bmod p$
$k_{AB} = \mathrm{hash}(K_{AB})$		$k_{AB} = \mathrm{hash}(K_{AB})$

$$\leftrightarrow$$

Secure channel using k_{AB} in a symmetric cipher

Why is it correct? Because $A^b \equiv g^{ab} \equiv B^a \pmod{p}$.

Diffie-Hellman key agreement

Hybrid encryption: we do not need to choose the secret key!

Domain parameters (p, g) and

- Alice's key pair $A = g^a \bmod p$
- Bob's key pair $B = g^b \bmod p$

Alice computes	Bob computes
$K_{AB} = B^a \bmod p$	$K_{AB} = A^b \bmod p$
$k_{AB} = \text{hash}(K_{AB})$	$k_{AB} = \text{hash}(K_{AB})$

$\leftrightarrow$

Secure channel using k_{AB} in a symmetric cipher

Why is it correct? Because $A^b \equiv g^{ab} \equiv B^a \pmod{p}$.

Diffie-Hellman key agreement

Hybrid encryption: we do not need to choose the secret key!

Domain parameters (p, g) and

- Alice's key pair $A = g^a \bmod p$
- Bob's key pair $B = g^b \bmod p$

Alice computes	Bob computes
$K_{AB} = B^a \bmod p$	$K_{AB} = A^b \bmod p$
$k_{AB} = \mathrm{hash}(K_{AB})$	$k_{AB} = \mathrm{hash}(K_{AB})$

$$\leftrightarrow$$

Secure channel using k_{AB} in a symmetric cipher

Why is it correct? Because $A^b \equiv g^{ab} \equiv B^a \pmod{p}$.

Ephemeral Diffie-Hellman key agreement

Goal: avoid using the same long-term keys for all communications.

Domain parameters (p, g) and

- Alice's long-term key pair $A = g^a \bmod p$
- Bob's long-term key pair $B = g^b \bmod p$

Alice		Bob
randomly chooses e		randomly chooses f
sends $E = g^e \bmod p$	$\leftrightarrow$	sends $F = g^f \bmod p$
along with $\text{sign}_a(E)$	$\leftrightarrow$	along with $\text{sign}_b(F)$
checks Bob's signature with B		checks Alice's signature with A
$K_{\text{AB}} = F^e \bmod p$		$K_{\text{AB}} = E^f \bmod p$
$k_{\text{AB}} = \text{hash}(K_{\text{AB}})$		$k_{\text{AB}} = \text{hash}(K_{\text{AB}})$

$\leftrightarrow$

Secure channel using k_{AB} in a symmetric cipher

Ephemeral Diffie-Hellman key agreement

Goal: avoid using the same long-term keys for all communications.

Domain parameters (p, g) and

- Alice's long-term key pair $A = g^a \bmod p$
- Bob's long-term key pair $B = g^b \bmod p$

Alice		Bob
randomly chooses e		randomly chooses f
sends $E = g^e \bmod p$	$\leftrightarrow$	sends $F = g^f \bmod p$
along with $\mathrm{sign}_a(E)$	$\leftrightarrow$	along with $\mathrm{sign}_b(F)$
checks Bob's signature with B		checks Alice's signature with A
$K_{\mathrm{AB}} = F^e \bmod p$		$K_{\mathrm{AB}} = E^f \bmod p$
$k_{\mathrm{AB}} = \mathrm{hash}(K_{\mathrm{AB}})$		$k_{\mathrm{AB}} = \mathrm{hash}(K_{\mathrm{AB}})$

$\leftrightarrow$

Secure channel using k_{AB} in a symmetric cipher

ElGamal signature

Signature of message $m \in Z_2^*$ by Alice with her private key a:

- Compute $h = \text{hash}(m)$
- Choose randomly an integer $k \in [1, p-2]$
- Compute $r = g^k \bmod p$
- Compute $s = k^{-1}(h - ar) \bmod (p-1)$
  - If $s = 0$, restart with a new k
- Send (r, s) along with m

Verification of signature (r, s) on m with Alice's public key A:

- Compute $h = \text{hash}(m)$
- Check $A^r r^s \stackrel{?}{\equiv} g^h \pmod p$

ElGamal signature

Signature of message $m \in Z_2^*$ by Alice with her private key a:

- Compute $h = \text{hash}(m)$
- Choose randomly an integer $k \in [1, p-2]$
- Compute $r = g^k \bmod p$
- Compute $s = k^{-1}(h - ar) \bmod (p-1)$
  - If $s = 0$, restart with a new k
- Send (r, s) along with m

Verification of signature (r, s) on m with Alice's public key A:

- Compute $h = \text{hash}(m)$
- Check $A^r r^s \stackrel{?}{\equiv} g^h \pmod{p}$

ElGamal signature

Why is it correct?

$$\begin{aligned} A^r r^s &\equiv (g^a)^r (g^k)^s \\ &\equiv g^{ar+ks \bmod (p-1)} \\ &\equiv g^{ar+kk^{-1}(h-ar) \bmod (p-1)} \\ &\equiv g^{ar+h-ar \bmod (p-1)} \\ &\equiv g^h \pmod{p} \end{aligned}$$

Security of ElGamal signature

r and k can be seen as an **ephemeral key** pair, created by the signer. r is part of the signature, and k is protected by the DLP.

Caution: k must be secret and randomly drawn independently at each encryption! (Even more so than with ElGamal encryption!!!)

- If k is known, one can recover a from s!
- If k is repeated to sign messages with hashes $h_1 \neq h_2$, then

$$s_1 - s_2 \equiv k^{-1}(h_1 - ar) - k^{-1}(h_2 - ar) \equiv k^{-1}(h_1 - h_2) \pmod{p-1}$$

and we can recover k then a from s_1 or s_2!

Security of ElGamal signature

r and k can be seen as an **ephemeral key** pair, created by the signer. r is part of the signature, and k is protected by the DLP.

Caution: k must be secret and randomly drawn independently at each encryption! (Even more so than with ElGamal encryption!!!)

- If k is known, one can recover a from s!
- If k is repeated to sign messages with hashes $h_1 \neq h_2$, then

$$s_1 - s_2 \equiv k^{-1}(h_1 - ar) - k^{-1}(h_2 - ar) \equiv k^{-1}(h_1 - h_2) \pmod{p-1}$$

and we can recover k then a from s_1 or s_2!

DSA: a compact variant of ElGamal

Domain parameters:

- Let p be a large prime, such that $p = nq + 1$.
  The bit size of q is limited to the size of the hash function. E.g., for 128-bit security, q has a size of 256 bits.
  Note that $|\mathbb{Z}_p^*| = \Phi(p) = nq$.
- Let f be an element of $\mathbb{Z}_p^*$ with $\text{ord}(f) = q$.
  For instance, take a generator g, then compute $f = g^n \bmod p$.

Alice generates her key pair: $A = f^a \bmod p$ with $a \in \mathbb{Z}_q$.

Signature (changes compared to ElGamal):

- Choose randomly an integer $k \in [1, q-2]$
- Compute $r = (f^k \bmod p) \bmod q$
- Compute $s = k^{-1}(h + ar) \bmod q$

DSA: a compact variant of ElGamal

Domain parameters:

- Let p be a large prime, such that $p = nq + 1$.
  The bit size of q is limited to the size of the hash function. E.g., for 128-bit security, q has a size of 256 bits.
  Note that $|\mathbb{Z}_p^*| = \Phi(p) = nq$.
- Let f be an element of $\mathbb{Z}_p^*$ with $\mathrm{ord}(f) = q$.
  For instance, take a generator g, then compute $f = g^n \bmod p$.

Alice generates her key pair: $A = f^a \bmod p$ with $a \in \mathbb{Z}_q$.

Signature (changes compared to ElGamal):

- Choose randomly an integer $k \in [1, q-2]$
- Compute $r = \left(f^k \bmod p\right) \bmod q$
- Compute $s = k^{-1}(h + ar) \bmod q$

Schnorr signature

Claus Schnorr

Schnorr signature

Signature of message $m \in Z_2^*$ by Alice with her private key a:

- Choose randomly an integer $k \in [1, p-2]$
- Compute $r = g^k \bmod p$
- Compute $e = \text{hash}(r \| m)$
- Compute $s = k - ea \bmod (p-1)$
- Send (s, e) along with m

Verification of signature (s, e) on m with Alice's public key A:

- Compute $r' = g^s A^e \bmod p$
- Compute $e' = \text{hash}(r' \| m)$
- Check $e' \stackrel{?}{=} e$

Schnorr signature

Signature of message $m \in Z_2^*$ by Alice with her private key a:

- Choose randomly an integer $k \in [1, p-2]$
- Compute $r = g^k \bmod p$
- Compute $e = \text{hash}(r \| m)$
- Compute $s = k - ea \bmod (p-1)$
- Send (s, e) along with m

Verification of signature (s, e) on m with Alice's public key A:

- Compute $r' = g^s A^e \bmod p$
- Compute $e' = \text{hash}(r' \| m)$
- Check $e' \stackrel{?}{=} e$

Solving DLP generically: baby-step giant-step

Solving DLP generically: baby-step giant-step

Discrete logarithm problem (DLP) in $\mathbb{Z}_p^*$

Given $A = g^a \bmod p$, find a.

Let $N = |\mathbb{Z}_p^*|$ and g be a generator of $\mathbb{Z}_p^*$.

- Let $m \approx \sqrt{N}$ and suppose that $a = a_0 + a_1 m$ with $a_0, a_1 < m$
- $A \equiv g^{a_0 + a_1 m} \pmod{p} \quad \Leftrightarrow \quad A g^{-a_1 m} \equiv g^{a_0} \pmod{p}$
- For $i = 0$ to $m - 1$ sequentially (baby steps)
  - Compute and store (i, g^i), i.e., multiply by g at each step
- For $j = 0$ to $m - 1$ sequentially (giant steps)
  - Compute Ag^{-jm}, i.e., multiply by g^{-m} at each step
  - If $Ag^{-jm} = g^i$ then $a = i + jm$ and **exit**

Time and memory $O(\sqrt{N})$

Solving DLP generically: baby-step giant-step

Discrete logarithm problem (DLP) for any group G

Given $A = [a]g$, find a.

Let $N = |G|$ and g be a generator of G.

- Let $m \approx \sqrt{N}$ and suppose that $a = a_0 + a_1 m$ with $a_0, a_1 < m$
- $A = [a_0 + a_1 m]g \quad \Leftrightarrow \quad A \circ [-a_1 m]g = [a_0]g$
- For $i = 0$ to $m - 1$ sequentially (baby steps)
  - Compute and store $(i, [i]g)$, i.e., apply $\circ g$ at each step
- For $j = 0$ to $m - 1$ sequentially (giant steps)
  - Compute $A \circ [-jm]g$, i.e., apply $\circ[-m]g$ at each step
  - If $A \circ [-jm]g = [i]g$ then $a = i + jm$ and **exit**

Time and memory $O(\sqrt{N})$

Pohlig-Hellman

Let the size of the group be decomposed in prime factors:

$$N = |G| = \prod_i p_i^{e_i}.$$

Then, we can solve the DLP in time

$$O\left(\sum_i e_i \left(\log N + \sqrt{p_i}\right)\right).$$

Conclusions

The following conditions are **necessary** (but not sufficient!) for the DLP to be hard:

- The size of the group should be $\approx 2^{2s}$ for security strength s.
- The size of the group should be prime.

Note: we focus on the actual group used. In the case of DSA, it is the sub-group of size q generated by f.

What is an elliptic curve?

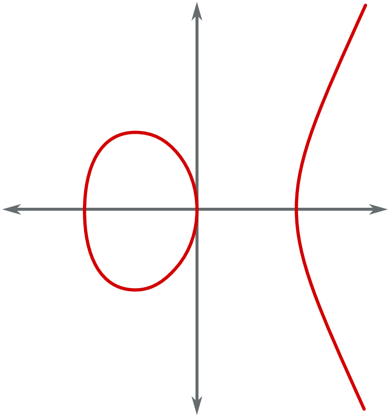
Given constants a and b, an **elliptic curve** is the set of points $(x, y) \in \mathbb{R}^2$ that satisfy the Weierstrass equation

$$y^2 = x^3 + ax + b$$

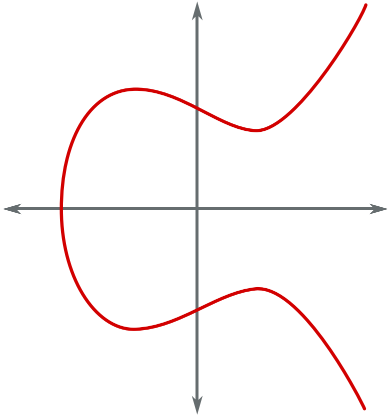
together with the **point at inifinity** denoted O. (One can view the point at infinity as $(0, \pm\infty)$.)

The Weierstrass equation is named after Karl Weierstrass.

Examples of elliptic curves over the reals



$y^2 = x^3 - x$



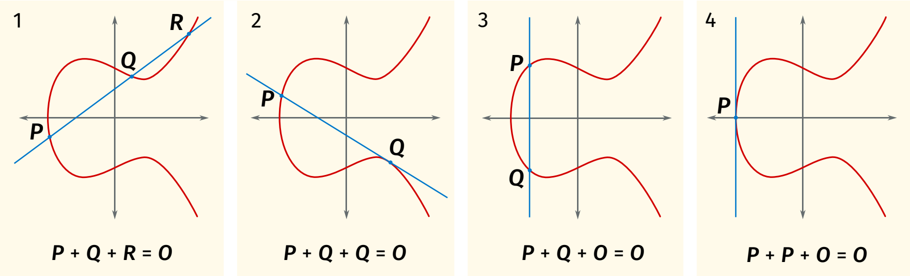
$y^2 = x^3 - x + 1$

[By YassineMrabet on Wikipedia]

Properties of elliptic curves

If a straight line meets an elliptic curve in two points, then it must cross a third point.

- A point on a tangeant counts for 2.
- Don't forget O!



[By SuperManu on Wikipedia]

Group law: point addition

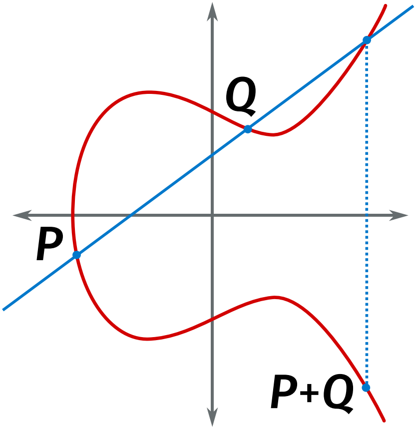
Let's build a rule to add points on the curve!

- Three aligned points must sum to O.
- The point at infinity O is the neutral element.

So:

- If P and P' are each other's reflections over the x axis, then P, P', O are aligned. So $P + P' + O = O$ and $-P = P'$.
- If P, Q, R are aligned, then $P + Q + R = O$ and $P + Q = -R$.

Group law, illustrated (general case)



$P \neq Q \neq -P$ and $P \neq O \neq Q$

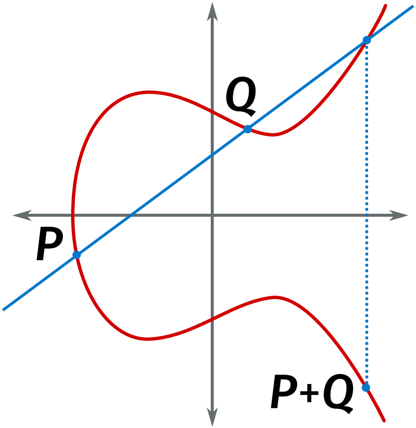
To add P and Q,

- draw a line from P to Q and note the third point where it intercepts the curve;
- reflect the third point.

$(x_P, y_P) + (x_Q, y_Q) = (x, y)$ with

$$s = (y_P - y_Q)/(x_P - x_Q)$$
$$x = s^2 - x_P - x_Q$$
$$y = s(x_P - x) - y_P$$

Group law, illustrated (general case)



$P \neq Q \neq -P$ and $P \neq O \neq Q$

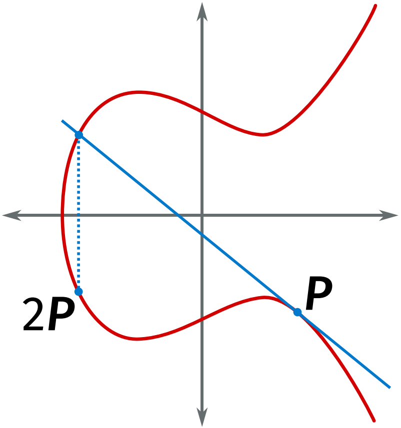
To add P and Q,

- draw a line from P to Q and note the third point where it intercepts the curve;
- reflect the third point.

$(x_P, y_P) + (x_Q, y_Q) = (x, y)$ with

$$s = (y_P - y_Q) / (x_P - x_Q)$$
$$x = s^2 - x_P - x_Q$$
$$y = s(x_P - x) - y_P$$

Group law, illustrated (point doubling)



$P \neq O$

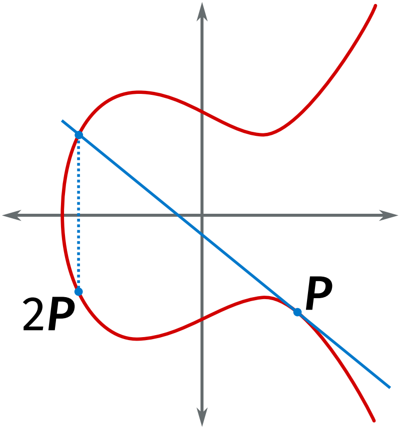
To add P to itself,

- draw a tangeant at P and note the “third” point where it intercepts the curve;
- reflect the “third” point.

$2(x_P, y_P) = (x, y)$ with

$$s = (3x_P^2 + a)/(2y_P)$$
$$x = s^2 - 2x_P$$
$$y = s(x_P - x) - y_P$$

Group law, illustrated (point doubling)



$P \neq O$

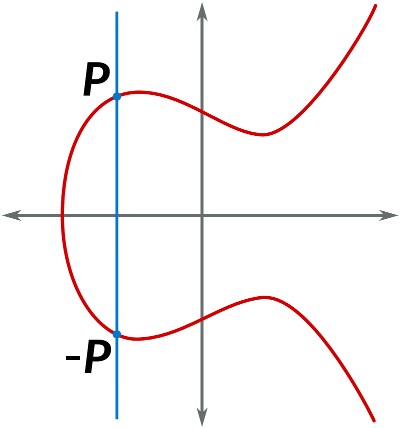
To add P to itself,

- draw a tangeant at P and note the "third" point where it intercepts the curve;
- reflect the "third" point.

$2(x_P, y_P) = (x, y)$ with

$$s = (3x_P^2 + a)/(2y_P)$$
$$x = s^2 - 2x_P$$
$$y = s(x_P - x) - y_P$$

Group law, illustrated (meet the point at infinity)



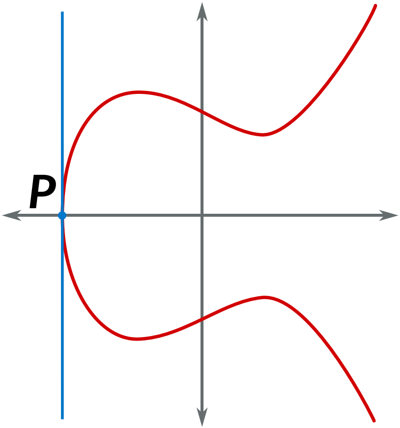
$P + (-P) = O$ because

- a vertical line meets the curve at infinity, and
- O is its own reflection through the x axis.

$P + O = P$ because

- a vertical line at P meets the curve at $-P$, and
- the reflection of $-P$ is P.

Group law, illustrated (point of order 2)



$P + P = 2P = O$ because

- the tangeant at P is a vertical line that meets the curve at infinity, and
- O is its own reflection through the x axis.

Note that $\mathrm{ord}(P) = 2$ because $2P$ is the neutral element.

Elliptic curves over (prime) finite fields

Fix a prime $p \geq 3$ and constants a and b. An **elliptic curve** is the set of points $(x, y) \in \mathbb{Z}_p^2$ that satisfy the Weierstrass equation

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

together with the **point at inifinity** denoted O.

Fact: the formulas for point addition and doubling also work here.

We can thus define a finite group comprising the points on the curve (including O, the neutral element), together with the point addition as operation.

Elliptic curves over (prime) finite fields

Fix a prime $p \geq 3$ and constants a and b. An **elliptic curve** is the set of points $(x, y) \in \mathbb{Z}_p^2$ that satisfy the Weierstrass equation

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

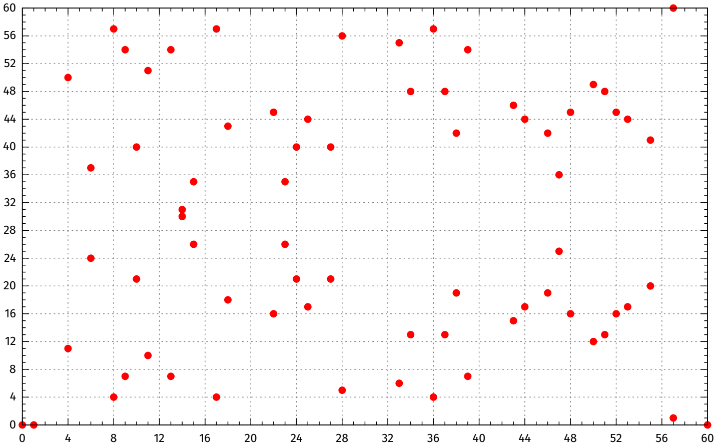
together with the **point at inifinity** denoted O.

Fact: the formulas for point addition and doubling also work here.

We can thus define a finite group comprising the points on the curve (including O, the neutral element), together with the point addition as operation.

Example of curve over $\mathbb{F}_{61}$

$$y^2 \equiv x^3 - x \pmod{61}$$



O is not shown but also belongs to the curve!

Number of points in a curve over a finite field

The number of points on a curve E is denoted $\#E$. This number depends on the parameters a and b.

Hasse's theorem

In $\mathrm{GF}(p)$, the number of points of an elliptic curve E satisfies

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}.$$

Let $\#E = hq$ with q the largest prime factor of $\#E$ (and h is called the co-factor). For security strength of s bits, we need $q \sim 2^{2s}$.

Number of points in a curve over a finite field

The number of points on a curve E is denoted $\#E$. This number depends on the parameters a and b.

Hasse's theorem

In $\mathrm{GF}(p)$, the number of points of an elliptic curve E satisfies

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}.$$

Let $\#E = hq$ with q the largest prime factor of $\#E$ (and h is called the co-factor). For security strength of s bits, we need $q \sim 2^{2s}$.

Example of elliptic curve (Weierstrass)

NIST P-256 [NIST FIPS 186-2 (2000)]

- Modulo $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$

$$y^2 = x^3 - 3x + 41058363725152142129326129780047268409114441015993725554835256314039467401291$$

- $\#E = p - 89188191154553853111372247798585809582$
- $\#E$ is prime, so $q = \#E$ and cofactor $h = 1$
- Base (generator) point $G = (G_x, G_y)$ with

$$G_x = 48439561293906451759052585252797914202762949526041747995844080717082404635286$$

$$G_y = 36134250956749795798585127919587881956611106672985015071877198253568414405109$$

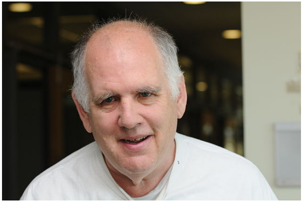
Montgomery curve

Given constants A and $B \in \mathbb{K}$, a **Montgomery curve** is the set of points $(x, y) \in \mathbb{K}^2$ that satisfy the equation

$$By^2 = x^3 + Ax^2 + x$$

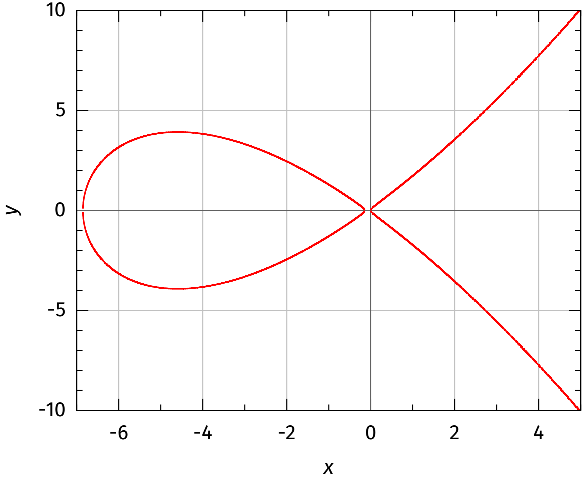
together with the **point at inifinity** denoted O. [Montgomery, M. of C. 1987]

- Birationally equivalent to a Weierstrass curve.
- In a finite field, a Weierstrass curve can be converted to a Montgomery curve only if $\#E$ is a multiple of 4.



Peter Montgomery

Example of Montgomery curve over the reals



$$3y^2 = x^3 + 7x^2 + x$$

[By Krishnavedala on Wikipedia]

Example of elliptic curve (Montgomery)

Curve25519 for Diffie-Hellman key exchange [D. J. Bernstein, PKC 2006]

- Modulo $p = 2^{255} - 19$

$$y^2 = x^3 + 486662x^2 + x$$

- $\#E = 8(2^{252} + 27742317777372353535851937790883648493)$
- Cofactor $h = 8$ and $q = 2^{252} + 277\ldots3$
- Base point of order q with $x = 9$

The y-coordinate is not taken into account to derive the secret key.

Edwards curve

Given constant $d \in \mathbb{K}$, an **Edwards curve** is the set of points $(x, y) \in \mathbb{K}^2$ that satisfy the equation

$$x^2 + y^2 = 1 + dx^2y^2.$$

- Birationally equivalent to a Weierstrass curve.
- In a finite field, a Weierstrass curve can be converted to an Edwards curve only if $\#E$ is a multiple of 4.
- **Complete addition formulas!**
  - No exceptions
  - No point at infinity, instead $O = (0, 1)$

Discovered by Harold Edwards in 2007

[Edwards, B. of the AMS, 2007]

Edwards curve

Given constant $d \in \mathbb{K}$, an **Edwards curve** is the set of points $(x, y) \in \mathbb{K}^2$ that satisfy the equation

$$x^2 + y^2 = 1 + dx^2y^2.$$

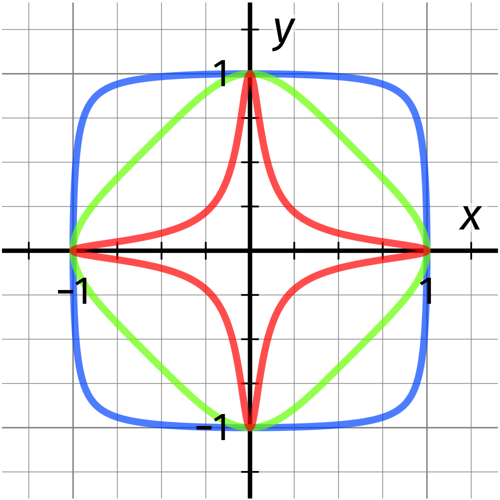
- Birationally equivalent to a Weierstrass curve.
- In a finite field, a Weierstrass curve can be converted to an Edwards curve only if $\#E$ is a multiple of 4.

- **Complete addition formulas!**
  - No exceptions
  - No point at infinity, instead $O = (0, 1)$

Discovered by Harold Edwards in 2007

[Edwards, B. of the AMS, 2007]

Example of Edwards curve over the reals



$x^2 + y^2 = 1 + dx^2y^2$ with $d \in \{-300, -\sqrt{8}, 0.9\}$

[By Georg-Johann on Wikipedia]

Edwards curve addition formula

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1 y_2 + x_2 y_1}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 - x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right)$$

Analogy with the addition of angles in a circle...

Edwards curve addition formula

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1 y_2 + x_2 y_1}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 - x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right)$$

Analogy with the addition of angles in a circle...

Example of elliptic curve (Edwards)

Ed448-Goldilocks [Hamburg, NIST ECC Workshop 2015]

- Modulo $p = 2^{448} - 2^{224} - 1$

$$x^2 + y^2 = 1 - 39081x^2y^2$$

- $\#E = 4(2^{446} - 13818066809895115352007386748515426880336692474882178609894547503885)$
- Cofactor $h = 4$ and $q = 2^{446} - 138\ldots5$
- Base point $G = (G_x, G_y)$ of order q with

$$G_x = -\frac{\sqrt{5}}{3} = \texttt{0xAAA...A9555...5}$$
$$G_y = \ldots$$

Twisted Edwards curve

Given constants $a, d \in \mathbb{K}$, a **twisted Edwards curve** is the set of points $(x, y) \in \mathbb{K}^2$ that satisfy the equation

$$ax^2 + y^2 = 1 + dx^2y^2.$$

- Birationally equivalent to Montgomery curves (one-to-one in a finite field)
- Same advantages as (untwisted) Edwards curves

[Bernstein, Birkner, Joye, Lange and Peters, Africacrypt 2008]

Dan Bernstein and Tanja Lange

Example of elliptic curve (twisted Edwards)

Ed25519 [Bernstein, Duif, Lange, Schwabe, Yang, J. Crypt. Eng., 2012]

- Modulo $p = 2^{255} - 19$

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$$

- $\#E = 8(2^{252} + 27742317777372353535851937790883648493)$
- Cofactor $h = 8$ and $q = 2^{252} + 277\ldots3$
- Base point of order q with $y = 4/5$ and x is even

Ed25519 is Curve25519 converted to the twisted Edwards form.

EdDSA, a deterministic variant of Schnorr

Key pair on curve E and base point G of order q

- Private key $(a, a') \quad \leftarrow \quad \text{hash}(\mathbf{a}^*)$
  - private scalar a
  - ephemeral derivation key a'
- Public key $A = [a]G$

Signature of message $m \in Z_2^*$ by Alice with her private key $\mathbf{a}^*$:

- Compute $k = \text{hash}(a' \| m)$
- Compute $R = [k]G$
- Compute $e = \text{hash}(R \| A \| m)$
- Compute $s = k + ea \bmod q$
- Send (R, s) along with m

Verification of signature (R, s) on m with Alice's public key A:

- Check $[s]G \stackrel{?}{=} R + [\text{hash}(R \| A \| m)]A$

EdDSA, a deterministic variant of Schnorr

Key pair on curve E and base point G of order q

- Private key $(a, a') \quad \leftarrow \quad \text{hash}(\mathbf{a}^*)$
  - private scalar a
  - ephemeral derivation key a'
- Public key $A = [a]G$

Signature of message $m \in Z_2^*$ by Alice with her private key $\mathbf{a}^*$:

- Compute $k = \text{hash}(a' \| m)$
- Compute $R = [k]G$
- Compute $e = \text{hash}(R \| A \| m)$
- Compute $s = k + ea \bmod q$
- Send (R, s) along with m

Verification of signature (R, s) on m with Alice's public key A:

- Check $[s]G \stackrel{?}{=} R + [\text{hash}(R \| A \| m)]A$

EdDSA, a deterministic variant of Schnorr

Key pair on curve E and base point G of order q

- Private key $(a, a') \quad \leftarrow \quad \text{hash}(\mathbf{a}^*)$
  - private scalar a
  - ephemeral derivation key a'
- Public key $A = [a]G$

Signature of message $m \in Z_2^*$ by Alice with her private key $\mathbf{a}^*$:

- Compute $k = \text{hash}(a' \| m)$
- Compute $R = [k]G$
- Compute $e = \text{hash}(R \| A \| m)$
- Compute $s = k + ea \bmod q$
- Send (R, s) along with m

Verification of signature (R, s) on m with Alice's public key A:

- Check $[s]G \stackrel{?}{=} R + [\text{hash}(R \| A \| m)]A$

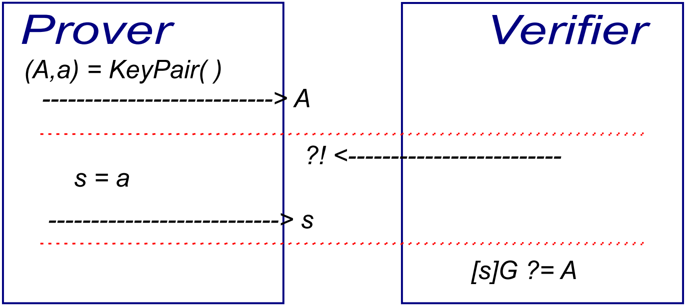
What is an identification protocol?

An identification protocol is a protocol where a *prover* wants to convince a *verifier* that (s)he knows the private key corresponding to his/her public key.

Public key $A = [a]G$: the prover wants to prove (s)he knows a.

A *simulator* wants to impersonate the prover.

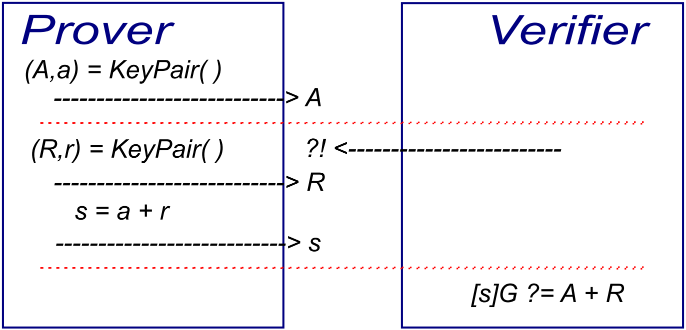
A simple (but flawed) identification protocol



[Figures courtesy of Benjamin Smith]

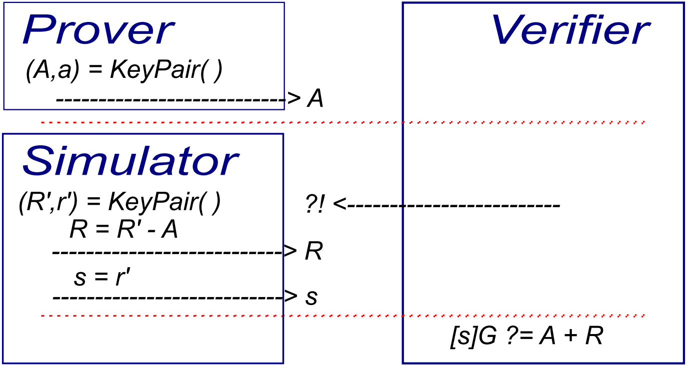
Problem: the prover has given away his/her private key and has therefore lost his/her identity.

Using an ephemeral key



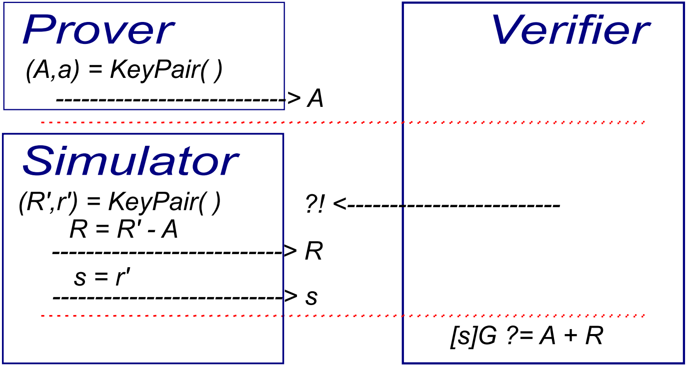
- The DLP makes it infeasible to recover r from R.
- s reveals nothing about a since r is random.
- $A + R = [a]G + [r]G = [s]G$

But cheating is possible



Problem: a simulator can easily impersonate the prover.

How to distinguish between prover and verifier?



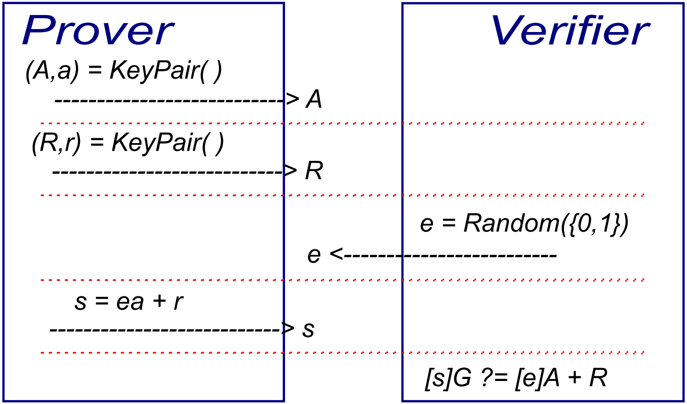
- Prover sends $s = a + r = \log(A + R)$, knows a and r
- Simulator sends $s = \log(A + R)$, does not know neither a nor r

How to fix the protocol?

Solution: ask either for s or for r.

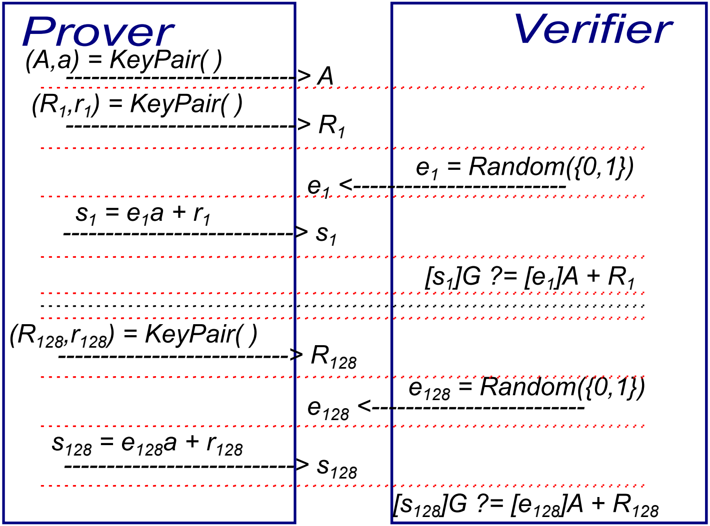
- If $[s]G = A + R$, the prover shows (s)he knows a, unless (s)he cheated
- If $[r]G = R$, the prover shows (s)he did not cheat, but does not prove (s)he knows a

Chaum-Evertse-Graaf

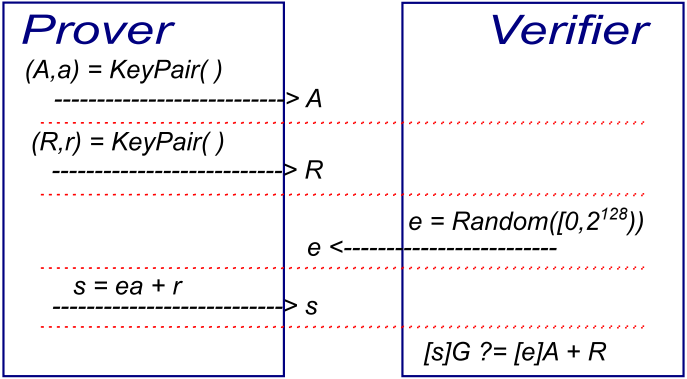


Cheater still has 50% chance of correctly anticipating *e*

Chaum-Evertse-Graaf with multiple rounds



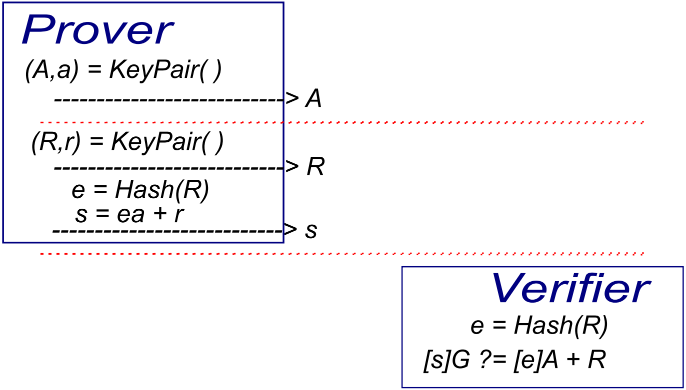
Schnorr's identification protocol



The Fiat-Shamir transform

The Fiat-Shamir replaces the verifier with a hash of the protocol's transcript. Hence, an interactive proof becomes non-interactive!

Fiat-Shamir: non-interactive Schnorr



The hash of R becomes the challenge! Intuitively, the prove does not control the hash of R.

Fiat-Shamir: non-interactive Schnorr (alternate)

```
Prover
(A,a) = KeyPair( )
    ---------------------------> A
.............................................................
 (R,r) = KeyPair( )
     e = Hash(R)
    ---------------------------> e
      s = ea + r
    ---------------------------> s
.............................................................
                                          Verifier
                                     R = [s]G - [e]A
                                     e ?= Hash(R)
```

We can send (e, s) instead of (R, s).

Committing to a message: Schnorr signatures

Signer

(A,a) = KeyPair( )

---------------------------> A

(R,r) = KeyPair( )   m <------------ Message

e = Hash(R,m)

---------------------------> e

s = ea + r

---------------------------> s

Verifier

R = [s]G - [e]A

e ?= Hash(R,m)

The challenge incorporates the message to sign.