Computer Security: Public Key Crypto

B. Jacobs

Institute for Computing and Information Sciences – Digital Security
Radboud University Nijmegen

Version: fall 2013

Outline

Public key background

- A big problem in secret key crypto is key managment:
  - N users need $\frac{N(N-1)}{2}$ different keys
- Public key crypto involves a revolutionary idea: use one **key pair** per user, consisting of
  - a public key, for:
    1. encryption
    2. checking signatures
  - a private key, for:
    1. decryption
    2. putting signatures

Using locks to explain the (encryption) idea

- Suppose Alice wants to sent Bob an encrypted message
- Bob first sends Alice his open padlock
  - only Bob has the private key to open it
  - but Alice (or anyone else) can close it
  - this open padlock corresponds to Bob's public key

- Alice puts the message in a box, and closes it with Bob's padlock
  - the box can be seen as a form of encryption
- Upon receiving the box, Bob uses his private key to open the padlock (and the box), and reads the message.
- Question: how do you know for sure this is Bob's lock?

Public key crypto: historical essentials

- The idea of public key crypto:
  - first invented in 1969 by James Ellis of GCHQ
  - first published in 1976 by Diffie & Hellman
- Implementations of public key crypto:
  - first one by Clifford Cocks (GCHQ), but unpublished
  - Rivest, Shamir and Adleman (RSA) first published in 1978, using the difficulty of prime number factorisation
  - several alternatives exist today, notably using "El-Gamal" on "elliptic curves"

Public key equation

- Let's write a key pair as:
  - K_e for encryption / public key
  - K_d for decryption / private key
- Let's further write the relevant operations as:
  - $\{m\}_{K_e}$ for encryption of message m with public key K_e
  - $[n]_{K_d}$ for decryption of message n with private key K_d
- The relevant equations are:

$$[\{m\}_{K_e}]_{K_d} \quad = \quad m$$

- But for certain systems (like RSA) one also has:

$$\{[m]_{K_d}\}_{K_e} \quad = \quad m$$

Key pair requirements

1. Encryption and decryption use different keys:
   - encryption uses the public "encryption" key
   - decryption the private "decryption" key
2. Encryption is one-way: it can not be inverted efficiently without the private key.
3. The private key cannot be reconstructed (efficiently) from the public one.
4. Encryption can withstand chosen plaintext attacks
   - needed because an attacker can generate arbitrary many pairs $\langle m, \{m\}_{K_e}\rangle$

Number theoretic ingredients I

- Recall that that a number is prime if it is divisible only by 1 and by itself.

  Prime numbers are: 2, 3, 5, 7, 11, 13, ... (infinitely many)
- Each number can be written in a unique way as product of primes (possibly multiple times), as in:

  $$30 = 2 \cdot 3 \cdot 5 \qquad 100 = 2^2 \cdot 5^2 \qquad 12345 = 3 \cdot 5 \cdot 823$$
- Finding such a prime number factorisation is a computationally hard problem
- In particular, given two very large primes p, q, you can publish $n = p \cdot q$ and no-one will (easily) find out what p, q are.
- Eeasy for $55 = 5 \cdot 11$ but already hard for $1763 = 41 \cdot 43$
- In 2009 factoring a 232-digit (768 bit) number $n = p \cdot q$ with hundreds of machines took about 2 years

Modular (clock) arithmetic

- On a 12-hour clock, the time '**1 o'clock**' is the same as the time '**13 o'clock**'; one writes

  $1 \equiv 13 \pmod{12}$ ie "1 and 13 are the same modulo 12"
- Similarly for 24-hour clocks:

  $$\begin{array}{rcll} 5 & \equiv & 29 \pmod{24} & \text{since } 5 + 24 = 29 \\ 5 & \equiv & 53 \pmod{24} & \text{since } 5 + (2 \cdot 24) = 53 \\ 19 & \equiv & -5 \pmod{24} & \text{since } 19 + (-1 \cdot 24) = -5 \end{array}$$
- In general, for $N > 0$ and $n, m \in \mathbb{Z}$,

  $n \equiv m \pmod{N} \iff$ there is a $k \in \mathbb{Z}$ with $n = m + k \cdot N$

  In words, the difference of n, m is a multiple of N.

Numbers modulo N

How many numbers are there modulo N?

One writes $\mathbb{Z}_N$ for the set of numbers modulo N. Thus:

$$\mathbb{Z}_N = \{0, 1, 2, \cdots N-1\}$$

For every $m \in \mathbb{Z}$ we have $m \bmod N \in \mathbb{Z}_N$.

Some Remarks

- Sometimes $\mathbb{Z}/N\mathbb{Z}$ is written for $\mathbb{Z}_N$
- Formally, the elements m of $\mathbb{Z}_N$ are *equivalence classes* $\{k \mid k \equiv m \pmod{N}\}$ of numbers modulo N
- These classes are also called residue classeses or just residues
- In practice we treat them simply as numbers.

Residues form a "ring"

- Numbers modulo N can be added, subtracted and multiplied: they form a "ring"
- For instance, modulo $N = 15$

  $$\begin{array}{rcl@{\qquad}rcl} 10 + 6 & \equiv & 1 & 6 - 10 & \equiv & 11 \\ 3 + 2 & \equiv & 5 & 0 - 14 & \equiv & 1 \\ 4 \cdot 5 & \equiv & 5 & 10 \cdot 10 & \equiv & 10 \end{array}$$
- Sometimes it happens that a product is 1

  For instance (still modulo 15): $4 \cdot 4 \equiv 1$ and $7 \cdot 13 \equiv 1$
- In that case one can say:

  $$\frac{1}{4} \equiv 4 \qquad \text{and} \qquad \frac{1}{7} \equiv 13$$

Multiplication tables

For small N it is easy to make multiplication tables for $\mathbb{Z}_N$.

For instance, for $N = 5$,

$\mathbb{Z}_5$	**0**	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

- **Note**: every non-zero number $n \in \mathbb{Z}_5$ has a an inverse $\frac{1}{n} \in \mathbb{Z}_5$
- This holds for every $\mathbb{Z}_p$ with p a prime number (more below)

Mod and div, and Java

- For $N > 0$ and $m \in \mathbb{Z}$ we write $m \bmod N \in \mathbb{Z}_N$
  - $k = (m \bmod N)$ if $0 \leq k < N$ with $k = m + x \cdot N$ for some x
  - For instance $15 \bmod 10 = 5$ and $-6 \bmod 15 = 9$
- % is Java's remainder operation. It behaves different from mod, on negative numbers.

$$\begin{array}{rcl@{\qquad\qquad}rcl} 7 \,\%\, 4 & = & 3 & 7 \bmod 4 & = & 3 \\ -7 \,\%\, 4 & = & -3 & -7 \bmod 4 & = & 1 \end{array}$$

  This interpretation of % is chosen for implementation reasons.
  [One also has $7 \,\%\, -4 = 3$ and $-7 \,\%\, -4 = -3$, which are undefined for mod]
- We also use integer division *div*, in such a way that:

$$n \;=\; m \cdot (n \; div \; m) + (n \bmod m)$$

  Eg. $15 \; div \; 7 = 2$ and $15 \bmod 7 = 1$, and $15 = 7 \cdot 2 + 1$.

Greatest common divisors

- Recall:

$$\begin{array}{rcl} gcd(n, m) & = & \text{“greatest common divisor of } n \text{ and } m\text{”} \\ & = & \text{greatest } k \text{ with } k \text{ divides both } n, m \\ & = & \text{greatest } k \text{ with } n = k \cdot n' \text{ and } m = k \cdot m', \\ & & \quad \text{for some } n', m' \end{array}$$

- Examples:

$$gcd(20, 15) = 5 \qquad gcd(78, 12) = 6 \qquad gcd(15, 8) = 1$$

- If $gcd(n, m) = 1$ one calls n, m relative prime

GCD computation

Euclid's algorithm:

$$\begin{array}{rcl} gcd(n, m) & = & \textbf{if } m = 0 \textbf{ then } n \\ & & \textbf{else } \; gcd(m, n \bmod m) \end{array}$$

Example:

$$\begin{array}{rcl} gcd(78, 12) & = & gcd(12, 78 \bmod 12) \\ & = & gcd(12, 6) \\ & = & gcd(6, 12 \bmod 6) \\ & = & gcd(6, 0) \\ & = & 6. \end{array}$$

Extended GCD computation

The extended GCD algorithm $egcd(n, m)$ returns a pair $x, y \in \mathbb{Z}$ with $n \cdot x + m \cdot y = gcd(n, m)$.

$$\begin{array}{rcl} egcd(n, m) & = & \textbf{if } n \bmod m = 0 \textbf{ then } \langle 0, 1 \rangle \\ & & \textbf{else let } \langle x, y \rangle = egcd(m, n \bmod m) \\ & & \quad \textbf{in } \langle y, \; x - (y \cdot (n \; div \; m)) \rangle \end{array}$$

This *egcd* is useful for computing inverses $\frac{1}{m} \bmod n$, when $gcd(m, n) = 1$.

Extended GCD correctness

Claim $egcd(n, m) = \langle x, y \rangle \Longrightarrow n \cdot x + m \cdot y = gcd(n, m)$.

$$\begin{array}{rcl} egcd(n, m) & = & \textbf{if } n \bmod m = 0 \textbf{ then } \langle 0, 1 \rangle \\ & & \quad \%\text{ in this case } m \text{ divides } n \text{, so } gcd(n, m) = m \\ & & \textbf{else let } \langle x, y \rangle = egcd(m, n \bmod m) \\ & & \quad \%\text{ may assume } mx + (n \bmod m)y = gcd(n, n \bmod m) \\ & & \textbf{in } \langle y, \; x - (y \cdot (n \; div \; m)) \rangle \\ & & \quad \%\text{ use } n = m \cdot (n \; div \; m) + (n \bmod m) \end{array}$$

[Correctness proof for the induction step:

$$\begin{array}{rl} & n \cdot y + m \cdot (x - (y \cdot (n \; div \; m))) \\ = & (m \cdot (n \; div \; m) + (n \bmod m)) \cdot y + m \cdot x - m \cdot y \cdot (n \; div \; m) \\ = & m \cdot y \cdot (n \; div \; m) + (n \bmod m) \cdot y + m \cdot x - m \cdot y \cdot (n \; div \; m) \\ = & m \cdot x + (n \bmod m) \cdot y \\ = & gcd(m, n \bmod m) \\ = & gcd(n, m) \quad \text{see the induction step of } gcd \end{array}$$

]

Extended GCD example

$$\begin{array}{rl} & egcd(78, 12) \\ = & \langle y, x - (y \cdot (78 \; div \; 12)) \rangle \\ & \quad \text{where } \langle x, y \rangle = egcd(12, 78 \bmod 12) = egcd(12, 6) \\ = & \langle y, x - (y \cdot 6) \rangle \\ & \quad \text{where } \langle x, y \rangle = \langle 0, 1 \rangle, \qquad \text{since } 12 \bmod 6 = 0 \\ = & \langle 1, 0 - 1 \cdot 6 \rangle \\ = & \langle 1, -6 \rangle \end{array}$$

Indeed: $1 \cdot 78 - 6 \cdot 12 = 78 - 72 = 6 = gcd(78, 12)$

Relative primes lemma

Lemma [Important]

$gcd(m, N) = 1$ iff m has an inverse modulo N (ie. in $\mathbb{Z}_N$)

Proof ($\Rightarrow$) Suppose $gcd(m, N) = 1$. Extended gcd yields x, y with $m \cdot x + N \cdot y = 1$. This means $m \cdot x \equiv 1 \bmod N$. Hence $\frac{1}{x} = m$.

Note: thus, *egcd* is useful for computing modular inverses!

($\Leftarrow$) Suppose $m \cdot x \equiv 1 \bmod N$, say $m \cdot x = 1 + N \cdot y$. Then $m \cdot x - N \cdot y = 1$. But $gcd(m, N)$ divides both m and N, so it divides $m \cdot x - N \cdot y = 1$. But if $gcd(m, N)$ divides 1, it must be 1 itself. □

Corollary

For p a prime, every non-zero $n \in \mathbb{Z}_p$ has an inverse ($\mathbb{Z}_p$ is a field)

More on relative primes

One writes:

$$\begin{array}{rcl} \mathbb{Z}_N^* & = & \{m \in \mathbb{Z}_N \mid m \text{ has an inverse } \bmod N\} \\ & = & \{m \in \mathbb{Z}_N \mid m, N \text{ are relative prime}\} \\ & = & \{m \in \mathbb{Z}_N \mid gcd(m, N) = 1\} \\ \phi(N) & = & \text{the number of elements in } \mathbb{Z}_N^* \\ & = & \text{Euler's totient function (for } N\text{)} \end{array}$$

Facts

1. $\mathbb{Z}_N^*$ is closed under multiplication (the "multiplicative" group)
2. $\phi(p) = p - 1$, for p a prime, since $\mathbb{Z}_p^* = \{1, 2, \ldots, p - 1\}$
3. $\phi(p \cdot q) = (p - 1) \cdot (q - 1)$, for p, q prime (proof e.g. via Chinese Remainder Theorem: $\mathbb{Z}_{p \cdot q} \cong \mathbb{Z}_p \times \mathbb{Z}_q$)

Multiplicative group example

Take $N = 10 = 2 \cdot 5$, so that $\phi(N) = (2 - 1) \cdot (5 - 1) = 4$.

Thus $\mathbb{Z}_{10}^*$ has 4 elements m with $gcd(m, 10) = 1$, namely: 1, 3, 7, 9

They form a multiplication table:

$\mathbb{Z}_{10}^*$	**1**	**3**	**7**	**9**
1	1	3	7	9
3	3	9	1	7
7	7	1	9	3
9	9	7	3	1

- **NOTE**: 3 is a generator: each element in $\mathbb{Z}_{10}^*$ occurs as $3^n = 3 \cdot 3 \cdots 3$, for some n.
- Namely: $3^0 = 1$, $3^1 = 3$, $3^2 = 9$, $3^3 = 3 \cdot 9 \equiv 7$.
- In general a finite group G is cyclic if $G = \{g^0, g^1, \ldots g^n\}$ for some $n \in \mathbb{N}$ and generator $g \in G$.

Two theorems [Background info]

Euler's theorem

If $gcd(m, N) = 1$, then $m^{\phi(N)} \equiv 1 \bmod N$

PROOF Write $\mathbb{Z}_N^* = \{x_1, x_2, \ldots, x_{\phi(N)}\}$ and form the product: $x = x_1 \cdot x_2 \cdots x_{\phi(N)} \in \mathbb{Z}_N^*$. Form also $y = (m \cdot x_1) \cdots (m \cdot x_{\phi(N)}) \in \mathbb{Z}_N^*$. Thus $y \equiv m^{\phi(N)} \cdot x$. Since m is invertible the factors $m \cdot x_i$ are all different and equal to a unique y_j; thus $x = y$. Hence $m^{\phi(N)} \equiv 1$. □

Fermat's little theorem

If p is prime and $gcd(m, p) = 1$ then $m^{p-1} \equiv 1 \bmod p$

PROOF Take $N = p$ in Euler's theorem and use that $\phi(p) = p - 1$. □

This is often used to test if a number p is actually prime: just try out if $m^{p-1} \equiv 1$ for many m (with $gcd(m, p) = 1$).

RSA, set-up

1. A user chooses:
   - two large primes p, q (each at least 1024 bits)
   - a number $e \in \mathbb{Z}_\phi^*$ where $\phi = \phi(p \cdot q) = (p - 1) \cdot (q - 1)$
2. The public key is now (n, e), where $n = p \cdot q$
3. The private key is (n, d), where $d = \frac{1}{e} \in \mathbb{Z}_\phi^*$, computed via *egcd*, so that $e \cdot d \equiv 1 \bmod \phi$

Note

- if the factorisation $n = p \cdot q$ is found by an attacker, the private exponent d dan be computed from the public exponent e (see later for a simple example)
- hence the security of RSA depends on the difficulty of factoring

RSA in action

- Encrypt $\{m\}_{(n,e)} = m^e \bmod n$ where the plaintext m is a number $m \in \mathbb{Z}_n$
- Decrypt $[k]_{(n,d)} = k^d \bmod n$
- Correctness Modulo n we have:

$$\begin{array}{rcll} [\{m\}_{(n,e)}]_{(n,d)} & = & [m^e]_{(n,d)} & \\ & = & (m^e)^d & \\ & = & m^{e \cdot d} & \\ & = & m^{1+k \cdot \phi} & \text{since } e \cdot d \equiv 1 \bmod \phi \\ & = & m \cdot (m^\phi)^k & \\ & = & m \cdot 1^k & \text{by Euler's theorem} \\ & = & m. & \end{array}$$

(Strictly speaking this proof only works for $m \in \mathbb{Z}_n^*$ but the result also holds for $m \in \mathbb{Z}_n$.)

Computing exponents via "repeated squaring"

Via the binary expansion of an exponent, modular exponentation can be done without big numbers. Example:

$$
\begin{array}{rcll}
8^7 \bmod 15 & \equiv & 8 \cdot 8^6 & \\
 & \equiv & 8 \cdot (8^2)^3 & \\
 & \equiv & 8 \cdot 64^3 & \\
 & \equiv & 8 \cdot 4^3 & \text{since } 64 \equiv 4 \bmod 15 \\
 & \equiv & 8 \cdot 4 \cdot 4^2 & \\
 & \equiv & 32 \cdot 16 & \\
 & \equiv & 2 \cdot 1 & \text{since } 32 \equiv 2 \bmod 15 \text{ and } 16 \equiv 1 \bmod 15 \\
 & \equiv & 2. &
\end{array}
$$

If you use linux, the shell program bc is very handy.
Typing in bc: 8^7%15 gives 2.

Simple RSA calculation (required skill)

- Take $p = 5, q = 11$, so that $n = p \cdot q = 55$ and $\phi = (5-1) \cdot (11-1) = 4 \cdot 10 = 40$.
- Choose $e = 3 \in \mathbb{Z}_{40}^*$, indeed with $gcd(40, 3) = 1$
- Compute $d = \frac{1}{e} = \frac{1}{3} \in \mathbb{Z}_{40}^*$ via $egcd(40, 3)$: it yields $x, y \in \mathbb{Z}$ with $40x + 3y = 1$, so that $d = \frac{1}{3} = y$.
- By hand: $egcd(40, 3) = (1, -13)$ (indeed with $40 \cdot 1 + 3 \cdot -13 = 40 - 39 = 1$)
- Hence $3 \cdot -13 \equiv 1 \bmod 40$, so $d = \frac{1}{3} = -13 \equiv 27 \bmod 40$.
- Let message $m = 19 \in \mathbb{Z}_n$ and encode $\{m\}_{(n,e)} = \{19\}_{(55,3)} = 19^3 \bmod 55 = 39$.
- Decode $[39]_{(n,d)} = [39]_{(55,27)} = 39^{27} \bmod 55 \equiv 19$!

Taking a small exponent e makes encryption fast;
this is often done, with typical values: $e = 3, 5, 17, 65537$

More RSA calculations

- Assume we have as public key $(91, 5)$.
  - **Question**: what is the corresponding private key?
  - These numbers are so small that it can be done by hand (this should not be possible in practice!)
- We have $p \cdot q = 91$, with only solution: $p = 7, q = 13$
- Hence $\phi = (p-1) \cdot (q-1) = 6 \cdot 12 = 72$
- We know $e = 5$, indeed with $gcd(72, 5) = 1$.
  - What is $d = \frac{1}{5} \bmod 72$?
- Calculate yourself: $egcd(72, 5) = \langle -2, 29 \rangle$, indeed with $-2 \cdot 72 + 29 \cdot 5 = -144 + 145 = 1$.
- Hence $29 \cdot 5 \equiv 1 \bmod 72$, and thus $d = \frac{1}{5} = 29$.
  - The private key is thus $(91, 29)$.

RSA in practice

- Using RSA in its naive, purely mathematical form is not secure
  - some basic mathematical properties give unwanted properties
  - eg. $\{m_1\}_{(n,e)} \cdot \{m_2\}_{(n,e)} \equiv m_1^e \cdot m_2^e \equiv (m_1 \cdot m_2)^e \equiv \{m_1 \cdot m_2\}_{(n,e)}$
- An attacker can thus manipulate encrypted messages
- Therefor, standards like PKCS#1 have been defined that destroy such structure
  - it involves adding random data, as padding

PKCS#1 basics (from RSA Laboratories)

INPUT: Recipient's RSA public key, (n, e) of length $k = |n|$ bytes; data D (eg. a session key) of length $|D|$ bytes with $|D| \leq k - 11$.
OUTPUT: Encrypted data block of length k bytes

1. Form the k-byte encoded message block, EB
   $$EB = 00 \,\|\, 02 \,\|\, PS \,\|\, 00 \,\|\, D$$
   where PS is a random string $k - |D| - 3$ non-zero bytes (ie. at least eight random bytes)
2. Convert the byte string, EB, to an integer, m, most significant byte first: $m = \mathtt{StringToInteger}(EB, k)$.
3. Encrypt with the RSA algorithm $c = m^e \bmod n$
4. Convert the resulting ciphertext, c, to a k-byte output block: $OB = \mathtt{IntegerToString}(c, k)$
5. Output OB.

PKCS#1 Example

Assume a RSA public key (n, e) with n 1024 bit long.

As data D, take a (random) AES-128 session key, such as:

D = 4E636AF98E40F3ADCFCCB698F4E80B9F

The resulting message block, EB, after encoding but before encryption, with random padding bytes shown in green, is:

```
EB = 0002257F48FD1F1793B7E5E02306F2D3
     228F5C95ADF5F31566729F132AA12009
     E3FC9B2B475CD6944EF191E3F59545E6
     71E474B555799FE3756099F044964038
     B16B2148E9A2F9C6F44BB5C52E3C6C80
     61CF694145FAFDB24402AD1819EACEDF
     4A36C6E4D2CD8FC1D62E5A1268F49600
     4E636AF98E40F3ADCFCCB698F4E80B9F
```

Such random padding makes $m^e \bmod n$ different each time

Public key generation

```
// standard lengths:512,1024,1536,2048,3072
int RSAlength = 1024;
KeyPairGenerator kpg =
    KeyPairGenerator.getInstance("RSA");
kpg.initialize(RSAlength);
// may take some time for big lengths
KeyPair kp = kpg.generateKeyPair();
```


Extracting public key info from a Java keypair

```
RSAPublicKey pubkey =
    (RSAPublicKey)kp.getPublic();
BigInteger
    n = pubkey.getModulus(),
    e = pubkey.getPublicExponent();
```


Extracting private key info from a Java keypair

```
RSAPrivateCrtKey privkey =
    (RSAPrivateCrtKey)kp.getPrivate();
BigInteger
    p = privkey.getPrimeP(),
    q = privkey.getPrimeQ(),
    d = privkey.getPrivateExponent(),
    phi = p.subtract(
        BigInteger.ONE).multiply(
        q.subtract(BigInteger.ONE));
```


RSA encryption & decryption

```
Cipher rsaCipher =
Cipher.getInstance("RSA/ECB/PKCS1Padding");
rsaCipher.init(Cipher.ENCRYPT_MODE, pubkey);
byte[] cleartext = ...
// encipher
byte[] ciphertext =
    rsaCipher.doFinal(cleartext);
// decipher
rsaCipher.init(Cipher.DECRYPT_MODE, privkey);
byte[] decipher =
    rsaCipher.doFinal(ciphertext);
```


RSA encryption & decryption "by hand"

```
BigInteger message = ...
BigInteger enc = message.modPow(e, n);
BigInteger dec = enc.modPow(d, n);
```


What is new with public key crypto

- Key management: every user only needs one key pair
  - but how do I obtain your public key (securely!)
  - where do I keep my private key?
  - what if my private key is lost or stolen?
- Digital signatures with public key crypto
  - What is such a signature?
- In general asymmetric (public key) crypto operations are more complicated and slower than in symmetric (secret key)
  - For encryption public key crypto is typically used to encrypt a session key for symmetric encipherment of the cleartext

Confidentiality

Assume

- each user X has keypair (e_X, d_X)
- each user X somehow knows the public key e_Y of each other user Y (more about this later)

Confidential exchange of a message m proceeds via:

$$A \longrightarrow B : \{m\}_{e_B}$$

Note

- After encryption, A cannot read the ciphertext
- If A is sloppy with her private key d_A, this need not affect B
- Integrity is not guaranteed (like in the symmetric case)

Integrity

The symmetric approach does not work in the asymmetric case:

$$A \longrightarrow B : m, \{h(m)\}_{e_B}$$

- What is the problem?
- Integrity is combined with non-repudiation via a digital signature

Authentication

The challenge-response approach works also in the asymmetric case:

$$\begin{array}{l} A \longrightarrow B : \{N\}_{e_B} \\ B \longrightarrow A : N \end{array} \quad \text{or} \quad \begin{array}{l} A \longrightarrow B : \{N\}_{e_B} \\ B \longrightarrow A : \{N\}_{e_A} \end{array}$$

Like for integrity, authentication is often combined with non-repudiation, in a signature (see later)

Needham-Schroeder two-way authentication

- Originally proposed in 1978; flaw discovered only in 1996 by Gavin Lowe (via formal methods, namely model checking)
- Simple fix exists

Needham-Schroeder: original version + attack

Protocol

$$\begin{array}{l} A \longrightarrow B : \{A, N_A\}_{e_B} \\ B \longrightarrow A : \{N_A, N_B\}_{e_A} \\ A \longrightarrow B : \{N_B\}_{e_B} \end{array}$$

Attack

$$\begin{array}{l} A \longrightarrow T : \{A, N_A\}_{e_T} \\ \quad T \longrightarrow B : \{A, N_A\}_{e_B} \\ \quad B \longrightarrow T : \{N_A, N_B\}_{e_A} \\ T \longrightarrow A : \{N_A, N_B\}_{e_A} \\ A \longrightarrow T : \{N_B\}_{e_T} \\ \quad T \longrightarrow B : \{N_B\}_{e_B} \end{array}$$

Subtle interpretation of the attack

If A is so silly to start an authentication with an untrusted T (who can intercept), this T can make someone else, namely B, think he is talking to A while he is talking to T.

Needham-Schroeder: fix

$$\begin{array}{l} A \longrightarrow B : \{A, N_A\}_{e_B} \\ B \longrightarrow A : \{N_A, B, N_B\}_{e_A} \\ A \longrightarrow B : \{N_B\}_{e_B} \end{array}$$

Non-repudiation

- Recall that RSA not only satisfies $[\{m\}_e]_d = m$, but also $\{[m]_d\}_e = m$.
- This can be used for a digital signature
- Basic form:

$$A \longrightarrow B \colon m, [h(m)]_{d_A}$$

  - What does B need to check?
  - What does he know?
  - Not only integrity, but also authenticity and non-repudiation (A cannot later deny having sent this message)
  - Implicitly: the message m contains a timestamp, just like with ordinary signatures
  - Why does this *not* work in the symmetric case (with a shared key)?

Signature variations

- Both sign and encrypt:

$$A \longrightarrow B \colon \{m, [h(m)]_{d_A}\}_{e_B}$$

- Use fresh session key K for efficiency:

$$A \longrightarrow B \colon \{K\}_{e_B},\ K\{m,\ [h(m)]_{d_A}\}$$

This is basically what PGP (= Pretty Good Privacy) does, eg. for securing email. It is efficient, because m may be large.

Signature for authentication

One can also do a challenge-response with a signature:

$$A \longrightarrow B \colon N$$
$$B \longrightarrow A \colon [N]_{d_B}$$

Notes

- This requires a separate authentication keypair
  - you don't want to use your signing keypair for this, because the protocol asks you to sign any nonce N
  - this N could be the hash of "A gets everything B owns"
  - electronic identity cards (like eNIK in NL) thus have 2 keypairs, for signing and authentication
- This challenge-response is used in the e-passport:
  - it's called active authentication
  - aim: authenticity of the document, since the private key is hardware protected and cannot leave the chipcard

Digital signatures, in practice

- The private key is stored on a personal chipcard
  - the chip provides protected memory
  - access is personalised via a PIN
  - the key pair should be generated on-card
- A card reader is connected to a PC, with appropriate signing software, eg. as plugin for a mail client
- When the user agrees to sign a message:
  - the PIN has to be entered via the keyboard
  - the hash of the message is sent to the card, for on-card signing
- Lots of attack possibilities, esp. when the PC is corrupted
  - catch the PIN, for signing without the card owner
  - show a different message on the screen
- Possible solution: dedicated, tamper resistant, non-updateble signature devices (a bit like e-book readers, with only a screen, card reader and a keypad)

Modern smart card reader with pin pad

- This one is used in the context of the German e-Identity card *neue Personalausweis (nPA)*
- Interfaces for both contact and contactless cards
- Certified by BSI; cost: 30-50 €

Digital and ordinary signatures

- **Ordinary signature**
  - produced by human, expressing clear intent
  - the same on all documents
  - one person typically has one signature
  - technically not very secure, but embedded in established usage context
- **Digital signature**
  - produced by (smart card) device
  - different for each signed document
  - one person may have different signatures (key pairs), for different roles (eg. business, private)
  - technically secure, but broad experience still missing
  - Legal status when produced under appropriate conditions (see eg. `pkioverheid.nl` for details)

Client-side versus Server-side signatures

- So far we have discussed client-side signatures
  - private key is under physical control of the signer,
  - on own smart card, own USB stick or hard disk (with password protection)
- Alternative, server-side signature scenario:
  - private key is (in secure hardware module) on the server
  - signer authenticates to server, and then pushes *sign* button
  - signer is in logical control only
  - attempt to reduce non-repudiation to authentication
- Questions about server-side solutions:
  - Can the **sysadmin** sign on behalf of everyone else?
  - Strong authentication is nessary, requires PKI anyway
  - In practice this is done eg. with one-time-password via SMS
    - By *Digidentity*, still counting as qualified signature. Bizarre!

Blind signatures: what is the point?

- Suppose A wants B to sign a message m, where B does not know that he signs m
  - Compare: putting an ordinary signature via a carbon paper
- Why would B do such a thing?
  - for anonymous "tickets", eg. in voting or payment
  - the private key may be related to a specific (timely) purpose
  - hence B does have some control
- Blind signature were introduced in the earlier 80s by David Chaum

Blind signatures with RSA

Let (n, e) be the public key of B, with private key (n, d).

1. A wants to get a blind signature on m; she generates a random r, computes $m' = (r^e) \cdot m \bmod n$, and gives m' to B.
2. B signs m', giving the result $k = [m']_{(n,d)} = (m')^d \bmod n$ to A
3. A computes:

$$\frac{k}{r} = \frac{(m')^d}{r} = \frac{(r^e \cdot m)^d}{r} = \frac{r^{ed} \cdot m^d}{r} \equiv \frac{r \cdot m^d}{r} = m^d = [m]_{(n,d)}$$

Thus: B signed m without seeing it!

Blind signatures for e-voting tickets

- Important requirements in voting are (among others)
  - vote secrecy
  - only eligible voters are allowed to vote (and do so only once)
- There is a clear tension between these two points
- Usually, there are two separate phases:
  1. checking the identity of voters, and marking them on a list
  2. anonymous voting
- After step 1, voters get a non-identifying (authentic, signed) ticket, with which they can vote
- Blind signatures can be used for this passage from the first to the second phase

Blind signatures for untraceable e-cash

Assume bank B has key pairs (e_x, d_x) for coins with value x

$C \longleftrightarrow B$: authentication steps
$C \longrightarrow B$: "I wish to withdraw €15, as a €5 and a €10 coin"
$C \longrightarrow B$: $r_1^{e_5} \cdot h(c_1), r_2^{e_{10}} \cdot h(c_2)$ (with r_i, c_i random)
$B \longrightarrow C$: $\left(r_1^{e_5} \cdot h(c_1)\right)^{d_5} = r_1 \cdot h(c_1)^{d_5},\ \left(r_2^{e_{10}} \cdot h(c_2)\right)^{d_{10}} = r_2 \cdot h(c_2)^{d_{10}}$

As a result

- C can spend signed coins $(c_1, h(c_1)^{d_5}, 5)$; value is checkable
- the bank cannot recognise these coins: this cash is untraceable
- double spending still has to be prevented
  (either via a database of spent coins, or via more crypto)

Authorities don't want such untraceable cash, because they are afraid of black markets and loosing control

Public key problem

- A fundamental problem in public key crypto (that we side-stepped so far) is:
  - How do we know for sure what someone's public key is?
  - Trudy can try to make Alice use e_{Trudy} instead of e_{Bob}
- A Public Key Infrastructure (PKI) is used to provide certainty about public keys.
- Basic notion: Certificate, ie. signed statement:

$$\Big[\text{"}\textit{Trustee}\text{ declares that the public key of } X \text{ is } e_X;\ \text{this statement dates from (}\textit{start date}\text{) and is valid until (}\textit{end date}\text{), and is recorded with (}\textit{serial nr.}\text{)"}\Big]_{d_{Trustee}}$$

- There are standardised formats for certificates, like X509

Two possible PKI solutions

1. phone-book style ("trust what an authority says", top-down)
   - use a trusted list of pairs ⟨*name*, *pubkey*⟩
   - but who can be trusted to compile and maintain such a list?
   - this is done by a Certificate Authority (CA)
2. crowd style ("trust what your friends say", bottom-up)
   - pairs ⟨*name*, *pubkey*⟩ can be signed by multiple parties
   - trust such a pair if sufficiently many friends have signed it
   - this creates a web of trust

Certificate Authorities

- Main tasks of a CA:
  - registration of new certificates
  - publication of (valid) certificates
  - publication of revoked certificates, in a revocation list
- Most CAs are commercial companies, like VeriSign, Thawte, Comodo, or DigiNotar (now "dead")
- They offer different levels of certificates, depending on the thoroughness of identity verification in registration

Example verification, by VeriSign

VeriSign offers three assurance levels for certificates, see
verisign.com/repository/rpa.html

1. Class 1 certificate: only email verification for individuals: "authentication procedures are based on assurances that the Subscriber's distinguished name is unique within the domain of a particular CA and that a certain e-mail address is associated with a public key"
2. Class 2 certificate: "verification of information submitted by the Certificate Applicant against identity proofing sources"
3. Class 3 certificate: "assurances of the identity of the Subscriber based on the personal (physical) presence of the Subscriber to confirm his or her identity using, at a minimum, a well-recognized form of government-issued identification and one other identification credential."

Where do I find someone else's certificate?

- The most obvious way to obtain a certificate is: directly from the owner
- From a certificate directory or key server, such as:
  - pgp.mit.edu
    (you can look up BJ's key there, and see who signed it)
  - subkeys.pgp.net *etc.*
- Often "root certificates" are pre-configured, typically in browsers.
  - Eg. in *firefox* look under Preferences - Advanced - View Certificates
  - On the web:
    www.mozilla.org/projects/security/certs/included

Certificate usage examples

- Secure webaccess via server-side certificates (one way authentication only), recongnisable via: 

- Code signing, for integrity and authenticity of downloaded code
- Client-side certificates for secure remote logic (eg. in VPN = Virtual Private Network)
- Sensor-certificates in a sensor network, against spoofing sensors and/or sensor data

Revocation, via CRLs

Possible reasons for revocation

- certificate owner lost control over the private key
- crypto has become weak (think of MD5 or SHA-1 hash)
- CA turns out to unreliable (think of DigiNotar)

Certificate Revocation Lists (CRLs)

- maintained by CAs, and updated regularly (eg. 24 hours)
- must be consulted, in principle, before every use of a certificate; sometimes unpractical
- you can subscribe to revocation lists so that they are loaded automatically into your browser

Revocation, via OCSP

- CRLs are typically downloaded to a client; they require bandwidth and (secure) local storage
  - overflowing the list is possible attack scenario
- An alternative is OCSP = Online Certificate Status Protocol
  1. Suppose A wants to check B's certificate before use
  2. A sends an OCSP request to the CA, containing the serial number of B's certificate
  3. the CA looks up the serial number in its own (secure) database
  4. if not revoked, it returns a signed, successful OCSP response to A
- **Note**: with OCSP you reveal to the CA which certificates you actually use, and thus who you communicate with
  - also when you communicate with someone using OCSP

Certificate chains

Imagine you have certificates:

1. ["A's public key is e_A ..."]$_{d_B}$
2. ["B's public key is e_B ..."]$_{d_C}$

Suppose you have these 2 certificates, and C's public key

- What can you deduce?
- Who do you (have to) trust?
- To do what?

Example: active authentication in e-passport

- private key securely embedded in passport chip
- public key signed by producer (*Morpho* in NL)
- Morpho's public key signed by Dutch state

Web of trust: decentralised trust model I

Anarchistic form: key signing parties

- People meet to check each other's identity
- and exchange public key fingerprints: (truncated) hashes of public keys (BJ's is `0x576B9C3F`)
- later on, they look up the key corresponding to the fingerprint and sign it

(source: http://xkcd.com/364/)

Web of trust: decentralised trust model II

CAcert.org style: using assurers

- `cacert.org` provides free certificates, via a web-of-trust
- certificates owners can accumulate points by being signed by assurers
- if you have ≥ 100 points, you can become assurerer yourself

CAcert is poorly run and never managed to set up an audit in order to get its root key into mozilla (or other major browsers)

PKI vulnerabilities

- World-wide there are about 650 certificate authorities (CAs)
  - whatever these CAs sign is trusted by the whole world
  - everyone else along the certificate-chain must be trusted too
- This makes the PKI system fragile
  - CAs can sign anything, not only for their customers
  - e.g. rogue gmail certificates, signed by DigiNotar, appeared in aug.'11, but Google was never a customer of DigiNotar
- Available controls:
  - rogue certificates can be revoked (blacklisted), after the fact
  - browser producers can remove root certificates (of bad CAs)
  - compulsory auditing of CAs
  - via OCSP server logs certificate usage can be tracked

Small key problem in the wild (aug.-nov. 2011)

- What happened?
  - *F-secure* discovered a certificate used to sign malware
  - the malware targeted governments and defense industry
  - Relevant CA is *DigiCert* (Malaysia)
  - early nov: this CA is blocked both by Mozilla and Microsoft
- These certificates are based on 512 bit RSA keys
  - *Fox-IT* also found such malware (for "infiltrating high-value targets") and claims that public keys have been brute-forced
  - RSA-512 challenge broken around 2000
  - required time now: hours-weeks (depending on hardware)
  - malware signed with the resulting private key
- It is shocking to see that 512 bit certificates are apparently still (produced and) accepted: embarrassment to the industry

DigiNotar I: background

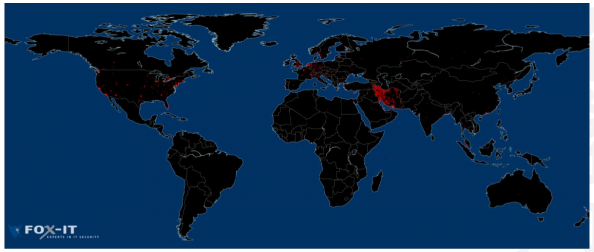
- The Dutch CA DigiNotar was founded in 1997, based on need for certificates among notaries
  - bought by US company *Vasco* in jan'11
  - "voluntary" bankruptcy in sept.'11
- DigiNotar's computer systems were infiltrated in mid july'11, resulting in rogue certificates
  - *DotNetNuke* CMS software was 30 updates ($\geq$ 3 years) behind
  - Dutch government only became aware on 2 sept.
  - it operated in "crisis mode" for 10 days
- About 60.000 DigiNotar certificates used in NL
  - many of them deeply embedded in infrastructure (for inter-system communication)
  - some of them need frequent re-issuance (short-life time)
  - national stand-still was nightmare scenario

DigiNotar II: act of war against NL?

- Hack claimed by 21 year old Iranian "Comodohacker"
  - he published proof (correct sysadmin password 'Pr0d@dm1n')
  - claimed to have access to more CAs (including GlobalSign)
  - also political motivation (pastebin.com/85WV10EL)

    Dutch government is paying what they did 16 years ago about Srebrenica, you don't have any more e-Government huh? You turned to age of papers and photocopy machines and hand signatures and seals? Oh, sorry! But have you ever thought about Srebrenica? 8000 for 30? Unforgivable... Never!
- Hacker could have put all 60K NL-certificates on the blacklist
  - this would have crippled the country
  - interesting question: would this be an act of war?
  - difficult but very hot legal topic: attribution is problematic
  - traditionally, in an "act of war" it is clear who did it.

DigiNotar III: rogue certificate usage (via OCSP calls)



Main target: 300K gmail users in Iran (via man-in-the-middle)

(More info: search for: *Black Tulip Update*, or for: *onderzoeksraad Diginotarincident*)

DigiNotar IV: certificates at stake

- DigiNotar as CA had its own root key in all browsers
  - it has been kicked out, in browser updates
  - Microsoft postponed its patch for a week (for NL only)!
  - the Dutch government requested this, in order to buy more time for replacing certificates (from other CAs)
- DigiNotar was also sub-CA of the Dutch state
  - private key of *Staat der Nederlanden* stored elsewhere
  - big fear during the crisis: this root would also be lost
  - it did not happen
  - alternative sub-CA's: Getronics PinkRoccade (part of KPN), QuoVadis, DigiDentity, ESG

DigiNotar V: Fox-IT findings

- DigiNotar hired security company *Fox-IT* (Delft)
  - Fox-IT investigated the security breach
  - published findings, in two successive reports (2011 & 2012)
- **Actual problem**: the serial number of a DigiNotar certificate found in the wild was not found in DigiNotar's systems records
- The number of rogue certificates is unknown
  - but OCSP logs report on actual use of such certificates
- Fox-IT reported "hacker activities with administrative rights"
  - attacker left signature *Janam Fadaye Rahbar*
  - same as used in earlier attacks on Comodo
- Embarrassing findings:
  - all CA servers in one Windows domain (no compartimentalisation)
  - no antivirus protection present; late/no updates
  - some of the malware used could have been detected

DigiNotar VI: lessons

- Know your own systems and your vulnerabilities!
- Use multiple certificates for crucial connections
- Strengthen audit requirements and process
  - only management audit was required, no security audit
  - the requirements are about 5 years old, not defined with "state actor" as opponent
- Security companies are targets, to be used as stepping stones
  - eg. march'11 attack on authentication tokens of RSA company
  - used later in attacks on US defence industry
- Alternative needed for PKI?
- Cyber security is now firmly on the (political) agenda
  - also because of "Lektober" and stream of (website) vulnerabilities
  - now almost weekly topic in Parliament (eg. breach notification and privacy-by-design)

DigiNotar VII: Finally (source: NRC 7/9/2011)

DigiNotar has not re-emerged: it had only one chance and blew it!

Discrete log problem

- The security of RSA depends on the difficulty of prime factorisation
  - this creates a "one-way function with a trapdoor"
- Another mathematical difficulty that is useful in cryptography is the discrete log problem
  - this applies to (multiplicative) groups like $\mathbb{Z}_N^*$
  - but also to (additive) groups of points on an elliptic curve.
- This elliptic curve crypto (ECC) is slowly replacing RSA, esp. because it involves shorter keys and is (thus) more efficient
  - roughly, 168 bit ECC keys correspond to 1024 bit in RSA

Logarithms

Recall: logarithm is the inverse of exponentiation

$$g^x = y \quad \Longleftrightarrow \quad x = \log_g(y).$$

The base g is often omitted when it is clear from the context

Now assume we have a finite cyclic group
$G = \{g^0 = 1, g^1 = g, g^2, g^3, \ldots, g^{N-1}\}$.

Discrete log problem: given $h \in G$, find $n < N$ with $h = g^n$

That is: $n = \log(h)$, wrt. base $g \in G$.

In general, this discrete log problem is computationally hard.
Intuitively, there is no better way than trying out all g^n.

Log example

Recall the multiplication table:

$\mathbb{Z}_{10}^*$	**1**	**3**	**7**	**9**
1	1	3	7	9
3	3	9	1	7
7	7	1	9	3
9	9	7	3	1

- 3 is generator: $3^0 = 1$, $3^1 = 3$, $3^2 = 9, 3^3 = 3 \cdot 9 \equiv 7$.
- Thus eg.

$$\begin{array}{rcl} \log_3(9) & = & 2 \\ \log_3(7) & = & 3 \end{array}$$

DH key exchange context


In a 1976 paper Whit Diffie and Martin Hellman published a crazy idea: how two people can agree on a secret key over an insecure line, without authentication

Parties A and B already share a publicly known group generator g.
(Alternatively, this info may be sent in the first message)
A and B exchange their own secrets $s_A, s_B \in \mathbb{N}$ in exponents:

$$\begin{array}{l} A \longrightarrow B\colon A, g^{s_A} \\ B \longrightarrow A\colon B, g^{s_B} \end{array}$$

Now they use as common key:

$$K_{AB} = g^{s_A s_B} = \left(g^{s_A}\right)^{s_B} = \left(g^{s_B}\right)^{s_A},$$

Both A and B can both compute this K_{AB}, but an eavesdropper in the middle does not have enough information to do so.

No free lunch: DH man-in-the-middle

DH does not involve authentication: it gives A and B a shared secret key, but they don't know who they share it with!

The main weakness of DH is a possible man-in-the-middle attack

$$\begin{array}{l} A \longrightarrow E\colon A, g^{s_A} \\ E \longrightarrow B\colon A, g^{s_E} \\ B \longrightarrow E\colon B, g^{s_B} \\ E \longrightarrow A\colon B, g^{s_E} \end{array}$$

Eve then has a shared key $K_{AE} = g^{s_A s_E}$ for communication with A and $K_{BE} = g^{s_B s_E}$ for communication with B. She sits quietly in the middle and translates back-and-forth.

Against man-in-the-middle for DH

Rivest and Shamir have a trick against such man-in-the-middle attacks: after key establishment A and B split the ciphertexts in halve, and send these halves interleaved. Split A's ciphertext as $c_A = c_A^1 \,\|\, c_A^2$, and similarly for B.

Thus:

$$\begin{aligned} A &\longrightarrow B \colon c_A^1 \\ B &\longrightarrow A \colon c_B^1 \\ A &\longrightarrow B \colon c_A^2 \\ B &\longrightarrow A \colon c_B^2 \end{aligned}$$

Since the attacker in the middle does not have enough information to translate the messages back-and-forth, the attack is quickly detected. Hence it can also be used at the beginning of a session to detect such a possible attacker.

DH in action I: cryptophones

- Diffie-Hellman key exchange is used within the "cryptophone" (`cryptophone.de`) for a fresh session key for each call
- Against man-in-the-middle attacks, a small part of the session key is shown on the phone's display, and can (or: should) be communicated by voice at the beginning of a call
- This requires discipline of the users (tricky): the two parties can make sure that they have the same key, implicitly using that they (often) know each other's voices.

A low-level countermeasure that police and intelligence forces can use is jamming: disrupt the conversation as soon as the crypto is used. This forces the parties to communicate in insecure mode.

A similar thing is used for GSM: some countries (like Israel) force foreign phones into unencrypted A5/0 mode.

More about the cryptophone

- The source code of the cryptophone is available for inspection, to make sure that there are no:
  - design/programming errors
  - backdoors

  One of the people involved is Rop Gonggrijp
- The cryptophone is not only used by criminals, but also by businessman (some overlap), NGOs, government agencies, etc.
- They don't trust the level of protection, here or abroad (GSM encryption itself is weak)
- Usage is limited because both caller and callee must have such a cryptophone
- Despite questions in parliament, it is not forbidden (in NL)
- Today we see special soft(&hard)ware for smartphones

DH in action II: e-passports

- Earlier we have seen the Basic Access Control (BAC) protocol for e-passports
  - it gives a terminal that knows the Machine Readable Zone (MRZ) access to the passport chip
  - it is only used for the less sensitive data, that are also available from the passport paper
- There is also an Extended Access Control (EAC) protocol
  - for the more sensitive biometric date, like fingerprints (EAC is done after BAC)
  - introduced later (since 2006) by German BSI
  - involves two subprotocols
    - Chip Authentication (CA), which creates new Diffie-Hellman session keys
    - Terminal Authentication (TA), which checks via certificates if the terminal is allowed to read the biometric data
  - Here we sketch how CA works

Chip Authentication (from EAC)

$$PsP \xrightarrow[(s_P \text{ is fixed passport secret})]{g^{s_P}} Rdr$$

$$PsP \xleftarrow[(s_R \text{ fresh reader secret})]{g^{s_R}} Rdr$$

$K = g^{s_P s_R}$ is now a fresh shared DH-key;
it is split in two keys: $K_{\mathrm{enc}}, K_{\mathrm{mac}}$

$$PsP \xrightarrow{K_{\mathrm{mac}}\{h(g^{s_R})\}} Rdr$$

Rdr then knows for sure that PsP has the same session key K (which is stronger than the BAC keys), and that PsP knows the secret key s_P corresponding to its public key g^{s_P}.

Student feedback after exam in 2012 ☺

Public and private keys, in DL setting, for El Gamal

Fix a generator $g \in G$ in a finite group, say of size (order) N.

Simple key pair set-up

- Private key: $n \in \mathbb{N}$ with $n < N$
- Public key: $h = g^n \in G$

- The Discrete Log Problem (DLP) guarantees that the private key n cannot be computed from the public key $h = g^n$.
- Next step: how to en/de-crypt and sign with such a key pair (g^n, n)

El Gamal: randomised en/de-cryption

Encryption

- assume cleartext is represented as $m \in G$
- choose random number $r < N$
- define, for public key $h \in G$, $\quad \{m\}_h = (g^r, m \cdot h^r)$

Decryption

- Assume ciphertext $c = (c_1, c_2)$, with $c_i \in G$
- define, for private key $n < N$, $\quad [(c_1, c_2)]_n = \dfrac{c_2}{(c_1)^n}$

Correctness

- For $h = g^n$ we get:

$$[\{m\}_h]_n = [g^r, m \cdot (g^n)^r]_n = \frac{m \cdot g^{n \cdot r}}{(g^r)^n} = \frac{m \cdot g^{n \cdot r}}{g^{n \cdot r}} = m.$$

El Gamal style signature (aka. DSA)

Signing with private key n (using hash function H)

- assume you wish to sign message m
- choose random number $r < p - 1 = |\mathbb{Z}_p^*|$ with $gcd(r, p-1) = 1$, so that $r^{-1} \bmod p - 1$ exists, and put:

$$\text{sign}_n(m) = \left(g^r, \; \frac{H(m) - n \cdot g^r}{r} \bmod p - 1 \right)$$

Verification with public key $h \in \mathbb{Z}_p^*$

- assume you have a message m with signature (s_1, s_2)
- check the equation:

$$g^{H(m)} \stackrel{??}{=} (s_1)^{s_2} \cdot h^{s_1}$$

Notice: no decryption, just checking

Correctness if $h = g^n$ is the public key, then indeed:

- $r \cdot s_2 \equiv H(m) - n \cdot g^r = H(m) - n \cdot s_1 \bmod p - 1$ so that:
- $g^{H(m)} = g^{r \cdot s_2 + n \cdot s_1} = (g^r)^{s_2} \cdot (g^n)^{s_1} = (s_1)^{s_2} \cdot h^{s_1}$

Background on curves

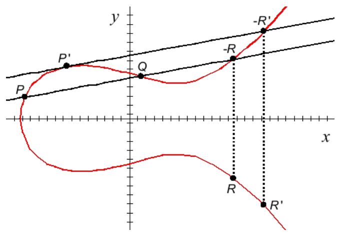
- Koblitz and Miller proposed the use of elliptic curves for cryptography in the mid 1980's
  - group operation is given by addition of points on a curve
  - nowadays this technology is widely accepted
- Provides the functionality of RSA and more
  - smaller keys
  - pairings (advanced, cool topic)
- Standard public key cryptography for embedded platforms (smart cards, eg. e-passport, sensors, etc.)
- Different key lengths (in bits) for comparable strength:

RSA	ECC
1024	160
2048	282
4096	409

Elliptic curve addition picture, over the real numbers

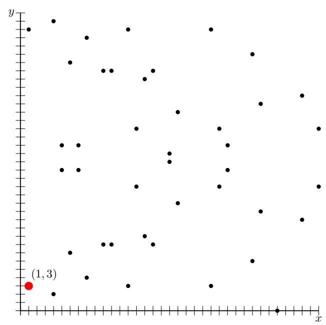
Elliptic curves are given by equations: $y^2 = x^3 + ax + b$.

Addition $P + Q = R$ and $P' + P' = 2 \cdot P' = R'$ is given by:

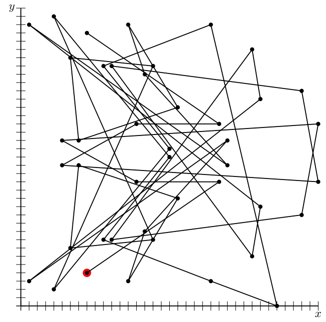


There are also explicit formulas for such additions.

Example curve: $y^2 = x^3 + 2x + 6$ over finite field $\mathbb{Z}_{37}$



Repeated addition: $n \cdot P$ goes everywhere



Given $Q = n \cdot P$, finding n involves basically trying all options.

Discrete Log and public keys for ECC

Since additive notation is use for curves the Discrete Log problem looks a bit funny:

Given $n \cdot P = P + \cdots + P$, it is hard to find the number n.

A keypair on a curve is thus a pair $(n \cdot P, n)$, for a point P and number n.