Computer Security: Public Key Crypto

B. Jacobs and J. Daemen
Institute for Computing and Information Sciences – Digital Security
Radboud University Nijmegen
Version: fall 2016

Outline

The blessings of crypto

Using crypto . . .

- Alice can protect her private data
- Alice and Bob can set up a secure channel
  - ensure confidentiality of content
  - ensure authenticity of messages
  - with respect to any adversary Eve
  - over any communication medium
- GlobalCorp. Inc. can protect its business
  - secure financial transactions
  - hide customer database from competitors
  - patch its products in the field for security/functionality
  - protect intellectual property in software, media, etc.
  - enforce its monopoly on games/accessories/etc.

The curse of crypto

- Alice and Bob need to share a cryptographic key
- GlobalCorp. Inc. needs to roll out cryptographic keys
- . . . in a way such that Eve cannot get her hands on them
- The security is only as good as the secrecy of these keys

Important lesson:

- Cryptography does not solve problems, but only reduces them to . . .
  - securely generating cryptographic keys
  - securely establishing cryptographic keys
  - keeping the keys out of Eve's hands

Key establishment

How do Alice and Bob establish a shared secret?

- When they physically meet:
  - exchange on a piece of paper or business card (unique pairs)
  - on a USB stick: requires trust in stick and PC/smartphone
  - but all cryptography requires trust in devices!
- When they don't meet it is harder. Two cases:
  - there is a common and trusted friend
  - no such friend
- For GlobalCorp. Inc. key management is much harder
  - *Eve* is ubiquitous
  - keys must be protected *in the field*

Remote key establishment with trusted third party

Alice and Bob have a common friend Wally they trust

- Both share a key with Wally
- Alice generates a key K and sends it encrypted to Wally
- Wally decrypts Alice's key, encrypts and sends it to Bob
- Bob decrypts the cryptogram and now has K
- Alice and Bob both rely on Wally's honesty
  - Did Wally send K to Bob and not to Eve?
  - Wally now has K and can eavesdrop on intercepted messages
- Problem is now: getting Wally out of the equation

Remote key establishment with trusted third parties

Alice and Bob have a number of common friends they trust

- For each friend
  - Alice randomly generates a secret key K_i
  - Alice sends it to Bob via the common friend
- Alice and Bob take sum of all K_i as shared key K
- Remaining risks: conspiracy
  - if friends collaborate, they can still cheat
- Remaining risks: denial-of-service
  - a misbehaving friend can disturb the process and prevent key setup
  - identifying saboteur is not easy

Remote key establishment w/o trusted third party

- Tamper-evident physically unclonable envelopes
  - tamper-evident: you cannot open it without leaving traces
  - unclonable: cannot fabricate one *looking the same*
- Sending by secure envelope:
  - Alice sticks a 5 Euro banknote on the envelope with superglue
  - Alice writes down the serial number of the banknote
  - Alice sends a key K to Bob in the envelope
  - upon receipt Bob checks that the envelope has not been opened
  - Bob calls Alice and they check the banknote's serial number
  - Bob gets the key K from the envelope
- Expensive and time-consuming

Challenges in keys management for GlobalCorp. Inc.

- Bank: getting keys in all banking cards
- Microsoft: getting software verification key in all PCs
- Spotify or NetFlix: getting keys in user PC/laptop/smartphones
- Government: getting keys in ID cards and travel passports
- More complex eco-systems
  - WWW: establishing keys between User PCs and internet sites
  - Public sector: keys for OV-Chipkaart
  - Mobile phone: imagine roaming
- etc.

Public Key cryptography to the rescue!

Public key crypto wish list

Lamport hash chains (see exercises, assignment 7): authentication without shared secret!

It would be nice to:

- Authenticate an entity without sharing a key with that entity
- Authenticate documents without writer's secret key:
  - Electronic Signatures!
- Set up a key remotely without the need for secret channel

Public key cryptography can do all that!

. . . and much more

Public key functionality

Public key crypto involves a counter-intuitive idea: use one **key pair** per user, consisting of

- private key PrK: never to be revealed to the outside world
- public key PK: to be published and distributed freely

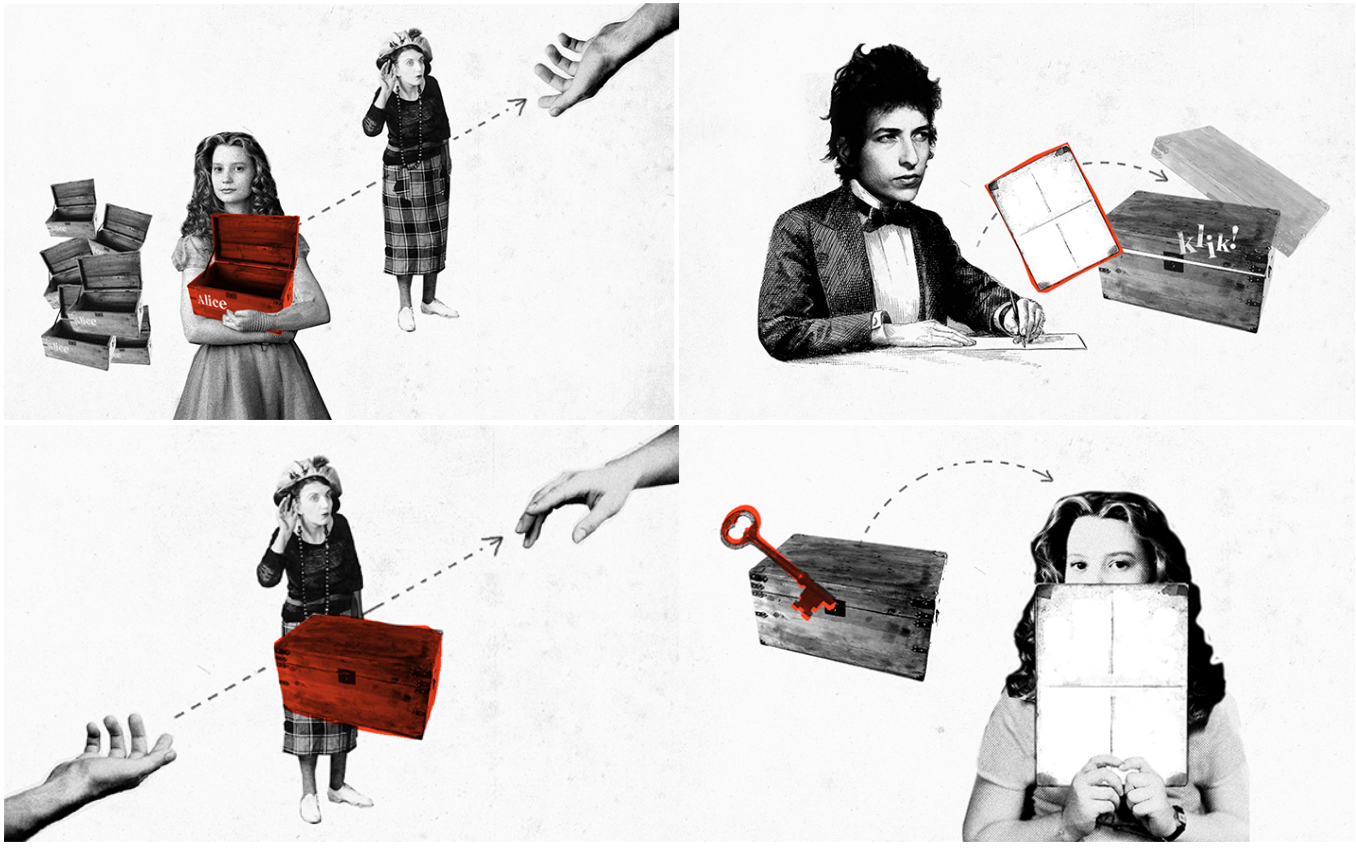
There are different types of public-key cryptosystems. Most used:

- Signature schemes
  - Alice uses PrK_A for signing message: $m, [m]_{PrK_A}$
  - anyone can use PK_A for verifying Alice's signatures
- Encryption schemes
  - anyone can use PK_A to encipher a message meant for Alice
  - Alice can decipher cryptogram with PrK_A
- Key establishment
  - Bob uses PrK_B and PK_A to compute secret K_{AB}
  - Alice uses PrK_A and PK_B to compute secret K_{AB}

Public key encryption as a *form of translation*

- Translation dictionaries
  - Private key *PrK*: Dictionary Ourgeze → Dutch
  - Public key *PK*: Dictionary Dutch → Ourgeze
- Say Alice keeps the last copy of the Ourgeze → Dutch Dictionary
  - Encryption: translate to Ourgeze using *PK*
  - Decryption: translate from Ourgeze using *PrK*
- Private key can be reconstructed from public key!
  - Not secure?
  - In pre-computer time this was a huge task!
- Same for actual public key cryptography
  - private key *PrK* can be computed from public key *PK*
  - just hope it's difficult: *many tried but none succeeded (so far)*
  - this is the basis of quasi all cryptographic security!

Public key encryption as *self-locking boxes*



Public key crypto: some history

- The idea of public key crypto and first key-establishment scheme
  - Ralph Merkle, Withfield Diffie, Martin Hellman in 1976
  - supposedly already invented at GCHQ in 1969
- The first public key signature and encryption scheme
  - published by Rivest, Shamir and Adleman (RSA) in 1978
  - supposedly already invented at GCHQ in 1970
- Elliptic Curve Cryptography
  - published independently by Koblitz and Miller in 1985
  - GCHQ must have overlooked this
  - the dominant public key cryptosystem today
- Nowadays literally thousands of public key systems
- Current research focuses on: post-quantum crypto
  - quantum computers would break all of the above
  - NSA/GCHQ, Google, etc. could possibly build one
  - public-key crypto that resists quantum attacks

Some notation that is used but not explained

- $\mathbb{Z}$: the set of integers: $\{\ldots -3, -2, -1, 0, 1, 2, 3, \ldots\}$
- $a \in A$: this means that a is an element of a set A. For example, $2 \in \mathbb{Z}$ means 2 is an element of the set of integers, or equivalently, 2 is an integer
- $\forall$: *for all*. E.g., $\forall a \in \mathbb{Z} : a + 1 \in \mathbb{Z}$ means: for every element of the set of integers, that element plus one is also an integer
- $\exists$: *exists*. E.g., $\forall a \in \mathbb{Z}, \exists b \in \mathbb{Z} : a + b = 0$ means: for every integer there exists an integer that added to that integer gives 0
- $|n|$: the length of the integer n in bits

Prime numbers and factorization

- 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.
- Easy 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

$$1 \equiv 13 \pmod{12} \quad \text{ie} \quad \text{"1 and 13 are the same modulo 12"}$$

- Similarly for 24-hour clocks:

$$\begin{aligned} 5 &\equiv 29 \pmod{24} \quad \text{since } 5 + 24 = 29 \\ 5 &\equiv 53 \pmod{24} \quad \text{since } 5 + (2 \cdot 24) = 53 \\ 19 &\equiv -5 \pmod{24} \quad \text{since } 19 + (-1 \cdot 24) = -5 \end{aligned}$$

- In general, for $N > 0$ and $n, m \in \mathbb{Z}$,

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

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 classes or just residues
- In practice we treat them simply as numbers

Residues form a "ring"

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

$$\begin{aligned} 10 + 6 &\equiv 1 \quad & 6 - 10 &\equiv 11 \\ 3 + 2 &\equiv 5 \quad & 0 - 14 &\equiv 1 \\ 4 \cdot 5 &\equiv 5 \quad & 10 \cdot 10 &\equiv 10 \end{aligned}$$

- 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)

iCIS | Digital Security
Radboud University

Mod and div, and Java (and C too)

- 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 differently from mod, on negative numbers.

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

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:

$$n = m \cdot (n \mathit{\ div\ } m) + (n \bmod m)$$

E.g., $15 \mathit{\ div\ } 7 = 2$ and $15 \bmod 7 = 1$, and $15 = 7 \cdot 2 + 1$.

iCIS | Digital Security
Radboud University

Groups: definition

- Couple $(A, \star)$ of a set A and a binary operation $\star$
- The binary operation must satisfy following properties:

closed:	$\forall a, b \in A:$	$a \star b \in A$
associative:	$\forall a, b, c \in A:$	$(a \star b) \star c = a \star (b \star c)$
neutral element:	$\exists e \in A, \forall a \in A:$	$a \star e = e \star a = a$
inverse element:	$\forall a \in A, \exists a' \in A:$	$a \star a' = a' \star a = e$
abelian (optional)	$\forall a, b \in A$	$a \star b = b \star a$

- Notational conventions

Additive:	$(A, +)$	$e = 0$	$a' = -a$
Multiplicative:	$(A, \times)$	$e = 1$	$a' = a^{-1}$ or $1/a$

- Groups can be finite or infinite, depending on A

Terminology: Group order

Order of a finite group $(A, \star)$, denoted $\#A$, is number of elements in A

Examples of groups and non-groups [for info only]

- Groups
  - $(\mathbb{Z}, +), (\mathbb{Q}, +), (\mathbb{R}, +), (\mathbb{Z}, +)$
  - $(\mathbb{Q} \setminus \{0\}, \times), (\mathbb{R} \setminus \{0\}, \times), (\mathbb{C} \setminus \{0\}, \times)$
- Non-groups
  - $(\mathbb{N}, +)$: no neutral element, no inverses
  - $(\mathbb{Z} \setminus \{0\}, \times)$: elements without inverse
  - $(\mathbb{Q}, \times)$: zero has no inverse

$(\mathbb{Z}_n, +)$: A simple example of a finite group

- $\mathbb{Z}_n$: integers smaller than n including zero
- Operation: addition modulo n

  (1) $c \leftarrow a + b$

  (2) if $c \geq n$, $c \leftarrow c - n$
- Notation: $a + b \bmod n$ or just $a + b$
- Order of the group $\#(\mathbb{Z}_n, +)$ is n
- One group $(\mathbb{Z}_n, +)$ for each integer n

Cyclic behaviour in finite groups

- Consider a sequence:
  - $i = 1 : a$
  - $i = 2 : a \star a$
  - $i = 3 : a \star a \star a$
  - ...
  - $i = n : [n]a$ (additive) or a^n (multiplicative)
- In a finite group $(A, \star)$:
  - $\forall a \in A$ this sequence is periodic
  - period of this sequence: order of a, denoted $\#a$

Terminology: Order of a group element

The order of a group element a, denoted $\#a$, is the smallest integer n such that $a^n = 1$ (multiplicative) or $[n]a = 0$ (additive).

iCIS | Digital Security
Radboud University

Cyclic groups and generators

- Consider the set $[0]g, [1]g, [2]g, \ldots$
- This is a group, called a cyclic group, denoted: $\langle g \rangle$
  - Neutral element $[0]g$
  - Inverse of $[i]g$: $[\#g - i]g$
- g is called generator
- Example of cyclic group $(\mathbb{Z}_n, +)$
  - generator: $g = 1$
  - $[i]g = i$

Example on orders: $(\mathbb{Z}_{21}, +)$

- Order of $\mathbb{Z}_{21}$: 21
- Order of 0: 1
- Order of 1: 21
- Order of 2: 21
- Order of 3: 7
- ...

Shortcut: *find the smallest i such that $i \cdot x$ is a multiple of n*

Fact: order of an element in $(\mathbb{Z}_n, +)$

$\#x = n/\gcd(n, x)$ with
$\gcd(n, x)$: the greatest common divisor of x and n

Greatest common divisor

- Definition:
$$\begin{aligned}\gcd(n,m) &= \text{greatest integer } k \text{ that divides both } n \text{ and } m \\ &= \text{greatest } k \text{ with } n = k \cdot n' \text{ and } m = k \cdot m', \\ &\qquad \text{for some } n', m'\end{aligned}$$
- Examples:
$$\gcd(20,15) = 5 \qquad \gcd(78,12) = 6 \qquad \gcd(15,8) = 1$$
- Properties:
  - $\gcd(n,m) = \gcd(m,n)$
  - $\gcd(n,m) = \gcd(n,-m)$
  - $\gcd(n,0) = n$

Terminology: relative prime (or coprime)

If $\gcd(n,m) = 1$, one calls n, m relative prime or coprime

Euclidean Algorithm

Property (assume $n > m > 0$):

- $\gcd(n, m) = \gcd(m, n \bmod m)$

This can be applied iteratively until one of arguments is 0

Example:

$$\begin{aligned}\gcd(171, 111) &= \gcd(111, 171 \bmod 111) = \gcd(111, 60)\\ &= \gcd(60, 111 \bmod 60) = \gcd(60, 51)\\ &= \gcd(51, 60 \bmod 51) = \gcd(51, 9)\\ &= \gcd(9, 51 \bmod 9) = \gcd(9, 6)\\ &= \gcd(6, 9 \bmod 6) = \gcd(6, 3)\\ &= \gcd(3, 6 \bmod 3) = \gcd(3, 0) = 3\end{aligned}$$

Variant allowing negative numbers :

$$\begin{aligned}\gcd(171, 111) &= \gcd(111, 171 \bmod 111) = \gcd(111, -51)\\ &= \gcd(51, 111 \bmod 51) = \gcd(51, 9)\\ &= \gcd(9, 51 \bmod 9) = \gcd(9, -3)\\ &= \gcd(3, 9 \bmod 3) = \gcd(3, 0) = 3\end{aligned}$$

Subgroups

- $(B, \star)$ is a subgroup of $(A, \star)$ if
  - B is a subset of A
  - $e \in B$
  - $\forall a, b \in B : a \star b \in B$
  - $\forall a \in B$: the inverse of a is in B

Lagrange's Theorem

If $(B, \star)$ is a subgroup of $(A, \star)$: $\#B$ divides $\#A$

- Case of cyclic subgroup: $\forall a \in A :< a >$ is a subgroup of $(A, \star)$

Corollary (for order of elements)

For any element $a \in A$: $\#a$ divides $\#A$

$(\mathbb{Z}_n, \times)$: A group?

- $\times$: Multiplication modulo n
- are group conditions satisfied?
  - closed: yes!
  - associative: yes!
  - neutral element: 1
  - inverse element: no, 0 has no inverse
- Let us exclude 0: so $(\mathbb{Z}_n \setminus \{0\}, \times)$
- Check properties again with multiplication table
- Examples:

(1) $(\mathbb{Z}_5 \setminus \{0\}, \times)$: OK!
(2) $(\mathbb{Z}_{21} \setminus \{0\}, \times)$: NOK!

$(\mathbb{Z}_p^*, \times)$ with prime p: a cyclic group!

- $\mathbb{Z}_p^*$ denotes $\mathbb{Z}_p$ with 0 removed
- Order of the group is $p-1$
- Group turns out to be cyclic
- Inverse of an element x:
  - Order of an element divides order of the group $p-1$
  - $x^{p-1} = 1$
  - So $x^{-1} = x^{(p-1)-1} = x^{p-2}$
  - Problem: this costs $p-3$ multiplications (at first sight ...)

Multiplicative prime groups

$(\mathbb{Z}_p^*, \times)$ is a cyclic group of order $p-1$

$(\mathbb{Z}_p^*, \times)$ with prime p

Multiplicative prime groups

$(\mathbb{Z}_p^*, \times)$ is a cyclic group of order $p-1$

Alternative way of seeing it:

- Find a generator $g \in (\mathbb{Z}_p^*, \times)$
- Write elements as powers of the generator: g^i
- Multiplication: find c such that $g^c = g^a \times g^b$
- Clearly: $g^a \times g^b = g^{a+b} = g^{a+b \bmod p-1}$
- So $c = a + b \bmod p - 1$

$(\mathbb{Z}_p^*, \times)$ is just $(\mathbb{Z}_{p-1}, +)$ in disguise!

Example: $(\mathbb{Z}_{23}^*, \times)$ and $(\mathbb{Z}_{22}, +)$ are similar

Back to $(\mathbb{Z}_n, \times)$ with n no prime

- We remove 0: $\mathbb{Z}_n^*$
- Inspection of multiplication table reveals some $a \times b = 0$
  - this implies $a \cdot b = k \cdot n$ for some k
  - a cannot be a multiple of n as $a < n$
  - b cannot be a multiple of n as $b < n$
  - from factorization of both sides, a must be multiple of factor of n
  - same for b
  - so a is not coprime to n and b is not coprime to n
- We now re-define $\mathbb{Z}_n^*$:
  - all integers $< n$ coprime to n, so with $\gcd(x, n) = 1$
  - closed: if $\gcd(a, n) = 1$ and $\gcd(b, n) = 1$, then $\gcd(ab, n) = 1$
  - does every element have an inverse?

Extended Euclidean Algorithm

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

Our earlier example:

$$\begin{aligned} -51 &= 171 - 2 \cdot 111 \\ 9 &= 111 + 2 \cdot (-51) \\ 3 &= (-51) + 6 \cdot 9 \\ 0 &= (-9) + 3 \cdot 3 \end{aligned}$$

And now backward substitution:

$$\begin{aligned} 3 &= (-51) + 6 \cdot 9 \\ 3 &= (-51) + 6 \cdot (111 + 2 \cdot (-51)) \\ 3 &= (-51) + 6 \cdot 111 + 12 \cdot (-51) \\ 3 &= 6 \cdot 111 + 13 \cdot (-51) \\ 3 &= 6 \cdot 111 + 13 \cdot (171 - 2 \cdot 111) \\ 3 &= 6 \cdot 111 + 13 \cdot 171 - 26 \cdot 111 \\ 3 &= 13 \cdot 171 - 20 \cdot 111 \end{aligned}$$

Relative primes lemma

Relative primes Lemma [Important]

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

Proof ($\Rightarrow$) Extended gcd yields x, y with $m \cdot x + N \cdot y = \gcd(m, N) = 1$. Taking both sides modulo N gives $m \cdot x \bmod N = 1$, or $x = m^{-1}$
($\Leftarrow$) We have $m \cdot x \equiv 1 \bmod N$ so there is an integer y such that $m \cdot x = 1 + N \cdot y$ or equivalently $m \cdot x - N \cdot y = 1$. Now $\gcd(m, N)$ 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. □

Note: Multiplicative inverse can be computed with extended Euclidean algorithm (with version more optimized than our example)

Corollary

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

What is the order of $(\mathbb{Z}_n^*, \times)$?

We now know $(\mathbb{Z}_n^*, \times)$ is a group but don't know its order

Definition: Euler's totient function

Euler's totient function of an integer n, denoted $\phi(n)$, is the number of integers smaller than and coprime to n.

- Clearly $\#(\mathbb{Z}_n^*, \times) = \phi(n)$
- For prime p, all integers 1 to $p-1$ are coprime to p: $\phi(p) = p - 1$
- If $n = a \cdot b$ with a and b coprime: $\phi(a \cdot b) = \phi(a)\phi(b)$
- For the power of a prime p^n: $\phi(p^n) = (p-1)p^{n-1}$
- Computing $\phi(n)$:
  - factor n into primes and their powers
  - apply $\phi(p^n) = (p-1)p^{n-1}$ to each of the factors
- Computing $\phi(n)$ is as hard as factoring n
- As $x^{\phi(n)} \bmod n = 1$, we can compute $x^{-1} = x^{\phi(n)-1} \bmod n$

iCIS | Digital Security
Radboud University

Number-theoretic theorems [Background info]

Euler's theorem (Lagrange's theorem applied to $(\mathbb{Z}_N^*, \times)$)

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 m is not a multiple of p then $m^{p-1} \equiv 1 \bmod p$

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

Used as primality test for p: try out if $m^{p-1} \equiv 1$ for many m.

iCIS | Digital Security
Radboud University

Exponentiation by Square-and-Multiply

- Computing a^e mod n in naive way takes $e - 1$ modular multiplications
- Infeasible if a, e and n are hundreds of decimals
- More efficient method: square-and-multiply
- Example: computing g^{12} with *left-to-right* square-and-multiply
  - $g^2 = g \times g$
  - $g^4 = g^2 \times g^2$
  - $g^8 = g^4 \times g^4$
  - $g^{12} = g^8 \times g^4$
- Only 3 squarings and 1 multiplication
- Instead of 11 in naive method

Exponentiation by Square-and-Multiply (cont'd)

- Computing g^{12} with *right-to-left* square-and-multiply
  - $g^2 = g \times g$
  - $g^3 = g^2 \times g$
  - $g^6 = g^3 \times g^3$
  - $g^{12} = g^6 \times g^6$
- Many variants exist, typical computation cost for $a^e \bmod N$:
  - $|e|$ squarings, with $|e|$ the bitlength e
  - 1 to $|e|$ multiplications, depending on e and method
- Relatively cheap
  - This is why RSA, DH and elliptic curve crypto actually work
  - Computing $x^{-1} \bmod n$ by $x^{\phi(n)-1} \bmod n$ much cheaper than by extended Euclidean algorithm

Ron Rivest, Adi Shamir, Leonard Adleman

Designed their famous cryptosystem in 1977-1978

What is the RSA cryptosystem?

RSA is a *trapdoor one-way function* $y = f(x)$

- given x, computing $y = f(x)$ is easy
- given y, finding x is difficult
- given y and trapdoor info: computing $x = f^{-1}(y)$ is easy

The RSA public key function:

$$y = x^e \bmod n \text{ with } x, y \in \mathbb{Z}_n$$

Features

- (n, e) is the public key, specific for a particular user
- Modulus $n = p \cdot q$ with p and q large primes
- Exponent e, usually a small prime, e.g., $2^{16} + 1$
- Trapdoor: knowledge of p and q

The RSA private key

The order of the group $(Z_n^*, \times)$ is $\phi(n)$ so $\forall x \in Z_n^*$:

$$x^{\phi(n)} \bmod n = x^{(p-1)(q-1)} = 1$$

Let d satisfy

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

then (omitting $\bmod n$)

$$(x^e)^d = x^{e \cdot d} = x^{1-k \cdot \phi(n)} = x \cdot x^{-k\phi(n)} = x \cdot (x^{\phi(n)})^{-k} = x$$

(Conclusion actually holds for all $x \in \mathbb{Z}_n$)
So the RSA private key operation

$$x = y^d \bmod n \text{ with } d = e^{-1} \bmod \phi(n)$$

(for info, can be optimized using so-called Chinese Remainder Theorem)

RSA public key pair

- Public key: public exponent and modulus (e, n)
- Private key: private exponent and modulus (d, n)
- Modulus:
  - $n = p \cdot q$ with p and q large primes
- Public exponent e
  - often small prime, e.g., $2^{16} + 1$: makes computing x^e light
  - $p - 1$ and $q - 1$ shall be coprime to e
- Private exponent d
  - exponent d is inverse of e modulo $(p - 1)(q - 1)$
  - length of d is close to that of n: x^d much slower than x^e
- Security of RSA relies on difficulty of factoring n
  - factoring n allows computing d from (e, n)
  - p and q shall be large enough and unpredictable by attacker
  - given n, knowledge of $\phi(n)$ allows factoring n and computing d

Factoring n, given $\phi(n)$

Assume modulus n is known

Knowledge of $\phi(n)$ allows factoring n

We have two equations $n = p \cdot q$ and $\phi(n) = (p-1)(q-1)$ or

$$n = p \cdot q \text{ and } \phi(n) = p \cdot q - p - q + 1$$

Subtracting them: $q = n - \phi - 1 - p$ and filling in in first equation:

$$n = p \cdot (n - \phi(n) - 1 - p)$$

Working out gives the following quadratic equation in p:

$$p^2 + (\phi(n) - n - 1) \cdot p + n = 0$$

Can be solved in usual way. E.g., $n = 91$ gives $q^2 - 20q + 91 = 0$, and so:

$$q = \frac{20 \pm \sqrt{400 - 4 \cdot 91}}{2} = \frac{20 \pm 6}{2} = 13, 7$$

Difficulty of factoring

- State of the art of factoring: two important aspects
  - reduction of computing cost: Moore's Law
  - improvements in factoring algorithms
- Factoring algorithms
  - Sophisticated algorithms involving many subtleties
  - Typically two phases:
    - distributed phase: equation harvesting
    - centralized phase: equation solving
  - Best known: general number field sieve (GNFS)
- These advances lead to increase of advised RSA modulus lengths
  see http://www.keylength.com/

Factoring records

number	digits	date	sieving time	alg.
C116	116	mid 1990	275 MIPS years	mpqs
RSA-120	120	June, 1993	830 MIPS years	mpqs
RSA-129	129	April, 1994	5000 MIPS years	mpqs
RSA-130	130	April, 1996	1000 MIPS years	gnfs
RSA-140	140	Feb., 1999	2000 MIPS years	gnfs
RSA-155	155	Aug., 1999	8000 MIPS years	gnfs
C158	158	Jan., 2002	3.4 Pentium 1GHz CPU years	gnfs
RSA-160	160	March, 2003	2.7 Pentium 1GHz CPU years	gnfs
RSA-576	174	Dec., 2003	13.2 Pentium 1GHz CPU years	gnfs
C176	176	May, 2005	48.6 Pentium 1GHz CPU years	gnfs
RSA-200	200	May, 2005	121 Pentium 1GHz CPU years	gnfs
RSA-768	232	Dec., 2009	2000 AMD Opteron 2.2 Ghz CPU years	gnfs

Using RSA for encryption

The naive way:

- Bob sends message to Alice enciphered with Alice's public key
  - Bob codes his message as an integer $m \in \mathbb{Z}$
  - e.g., standardized in PKCS#1 `StringToInteger`
  - Bob computes $c = m^e \bmod n$ with (e, n) the public key of Alice
- Alice deciphers received cryptogram
  - Alice computes $m = c^d \bmod n$ with (d, n) her private key
  - Alice decodes m as a message
  - using PKCS#1 `IntegerToString`

PKCS: series of industry standard from RSA Laboratories.

Using RSA for encryption: attention points

- m shall have enough entropy:
  - Eve can guess m and check if $c = m^e \bmod n$
  - e.g., EMV payment cards use RSA for encrypting PIN to card
  - method: add random string r: $m = \texttt{StringToInteger}(PIN, r)$
  - in symmetric encryption plaintext uniqueness (nonce) was sufficient
- Algebraic properties of RSA: (malleability)
  - e.g., given two cryptograms $c_1 = m_1^e$ and $c_2 = m_2^e$, Eve can construct a cryptogram for $m_3 = m_1 \cdot m_2 \bmod n$ as $c_3 = c_1 \cdot c_2 \bmod n$, without knowing m_1 or m_2 as $c_1 \cdot c_2 = m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e$
- Length of message is limited and defining modes is hard
- RSA decryption is relatively slow

Current advice by experts: don't encipher data with RSA

Using RSA for encryption: solutions

- Apply a hybrid scheme:
  - use RSA for establishing a symmetric key
  - encipher and authenticate with symmetric cryptography
- Sending an encrypted key
  - addition of redundancy before encryption
  - verification of redundancy after decryption
  - if NOK, return error
- Many proposals:
  - best known standard: PKCS #1 v1.5 and v2 (e.g. OAEP)
  - rather complex and not clear if objectives are achieved
- despite the problems, this is still the most widespread method

Example: PKCS#1 v1.5 padding for encryption

INPUT: Recipient's RSA public key, (n, e) of length $k = |n|/8$ bytes; payload D (e.g., a symmetric key) $|D| \leq 8(k - 11)$.
OUTPUT: Encrypted block of length k bytes

(1) Form the k-byte encoded block, EB

$$EB = 00 \parallel 02 \parallel PS \parallel 00 \parallel D$$

where PS is a random string $k - |D| - 3$ non-zero bytes
(ie. at least eight random bytes)

(2) Convert byte string EB to integer $m = \texttt{StringToInteger}(EB, k)$.

(3) Encrypt with RSA: $c = m^e \bmod n$

(4) Convert c to k-byte output block $OB = \texttt{IntegerToString}(c, k)$

(5) Output OB.

PKCS#1 v1.5 encryption padding example

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

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

$$D = \texttt{4E636AF98E40F3ADCFCCB698F4E80B9F}$$

Message block EB with random padding bytes shown in green:

$$EB = \texttt{0002257F48FD1F1793B7E5E02306F2D3}$$
$$\texttt{228F5C95ADF5F31566729F132AA12009}$$
$$\texttt{E3FC9B2B475CD6944EF191E3F59545E6}$$
$$\texttt{71E474B555799FE3756099F044964038}$$
$$\texttt{B16B2148E9A2F9C6F44BB5C52E3C6C80}$$
$$\texttt{61CF694145FAFDB24402AD1819EACEDF}$$
$$\texttt{4A36C6E4D2CD8FC1D62E5A1268F49600}$$
$$\texttt{4E636AF98E40F3ADCFCCB698F4E80B9F}$$

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

Using RSA for encryption: the solution

RSA Key Establishment Method (KEM)

- Bob randomly generates $r \in \mathbb{Z}_n$
- Bob sends $c = r^e \bmod n$ to Alice
- Alice deciphers c back to r
- both compute shared secret as hash(r)

RSA-KEM is THE sound way to use RSA for establishing a key

Using RSA for signatures

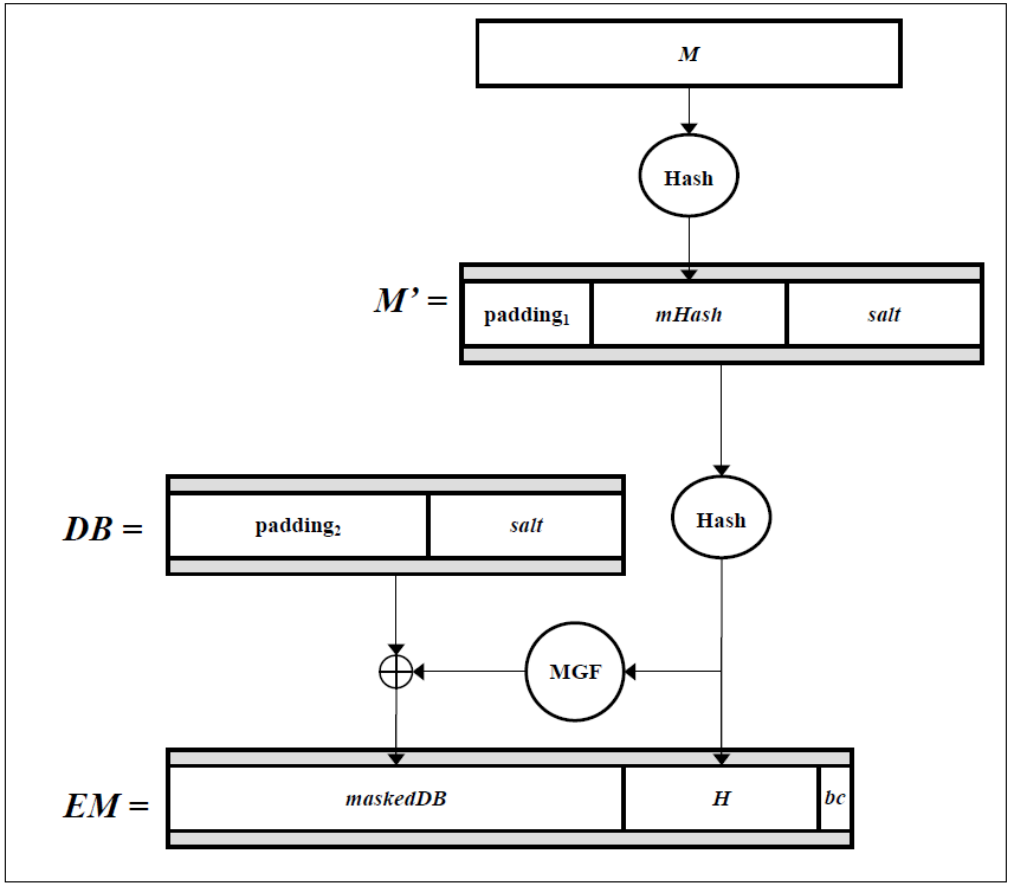
The naive way:

- Alice signs message with her private key
  - Alice codes her message as an integer m in $\mathbb{Z}_n$
  - Alice computes $s = m^d \bmod n$ with (d, n) her private key
- Bob verifies the signed message (m, s)
  - Bob computes $m' = s^e \bmod n$ with (e, n) the public key of Alice
  - He checks that $m' = m$

Using RSA for signatures: attention points

- Limitation on message length (and secure modes are hard to define)
  - instead of m, we input $\texttt{StringToInteger}(h(m))$
  - additional benefit: becomes much faster
- RSA malleability
  - given two signatures $s_1 = m_1^d$ and $s_2 = m_2^d$, Eve can construct a signature for $m_3 = m_1 \cdot m_2 \bmod n$ by computing $s_3 = s_1 \cdot s_2 \bmod n$.
  - generation of signature of message without knowing private key
- solution: specific padding schemes, e.g. PKCS # 1 v1.5 or v2 (PSS)
  - adds redundancy by padding
  - applies hashing for destroying algebraic structure
  - e.g., $s_1 \cdot s_2$ no longer verifies as a valid signature

RSA Probabilistic Signature Scheme (PSS)



(MGF = XOF)

RSA efficiency

- Private exponentiation:
  - Square and multiply
  - grows with the third power of the modulus length
  - e.g., modulus length $\times 2$: computation time goes $\times 8$
- Public exponentiation:
  - more efficient thanks to short public exponent
- Key generation:
  - randomly generating large primes p and q
  - About 15 to 40 times the effort of a private exponentiation

RSA key pair generation

A user generating an RSA key pair with given modulus length $|n|$:

- chooses the public exponent e
  - Often a small prime imposed by the context
  - Sometimes randomly generated per user, e.g. 256 bits
- randomly generates prime p of given length $\ell = |n|/2$
  - $p - 1$ shall be coprime to e
- randomly generates prime q such that $p \cdot q$ has length $|n|$
  - $q - 1$ shall be coprime to e
- computes modulus $n = p \cdot q$
- computes private exponent d as e^{-1} modulo $(p-1)(q-1)$
- Attention points [for info only]:
  - RSA works with p, q of any length but often software requires that $|n|$ is a multiple of 8 (or 32) and $|p| = |q| = |n|/2$
  - There are multiple valid values of $d < (p-1)(q-1)$ but just one $< \text{lcm}(p-1)(q-1) = (p-1)(q-1)/\gcd(p-1, q-1)$

Generation of random primes

Randomly generating a prime of given given length ℓ:

- generate string of $\ell - 2$ random bits
- put a 1 before and after
- interpret the result as an integer x: odd integer length ℓ
- repeat following loop:
  - if $\gcd(x-1, e) \neq 1$, go to last step of the loop
  - randomly choose b and do Fermat test: $b^{x-1} \bmod x = 1$?
  - if it fails the test, go to last step of the loop
  - do w more Fermat tests for randomly chosen b
  - if it passes all tests, return $p = x$
  - add 2 to x and try again

(Just an example, several other approaches)

Suggestion: program this in Python!

Generation of random primes: attention points

- Execution time: long and variable
  - takes several exponentiations
  - trial and error: sometimes lucky, sometimes not
- Optimization
  - trial division by small primes: $3, 5, 7, 11, \cdots$
  - fixing the base b to small numbers: $2, 3, \ldots$
  - variant of Fermat test: Rabin-Miller, slightly more efficient
- Security
  - result can still be non-prime but probability decreases with number of Fermat tests w
  - unpredictability of random generator is crucial!
- Special features
  - range of result is $[2^{\ell} + 1, 2^{\ell+1} - 1]$
  - for different range: fix most significant bits

RSA toy example, calculated by hand

- Choose $e = 3$
- Take $p = 5, q = 11$, so that $n = p \cdot q = 55$ and $\phi(n) = 40$
  - OK: both $p - 1$ and $q - 1$ are coprime to e
- Compute $d = \frac{1}{e} = \frac{1}{3} \in \mathbb{Z}_{40}^*$ with extended Euclidean algorithm:
  - it yields $x, y \in \mathbb{Z}$ with $40x + 3y = 1$, so that $d = \frac{1}{3} = y$
  - By hand: $3^{-1} \bmod 40 = -13 = 27$
    (indeed with $40 \cdot 1 + 3 \cdot -13 = 40 - 39 = 1$)
- Let message $m = 19 \in \mathbb{Z}_n$
  - encipher $c = m^e \bmod n = 19^3 \bmod 55 = 39$
  - decipher $m' = c^d \bmod n = 39^{27} \bmod 55 = 19$

The Achilles' Heel of (public key) cryptography

Cryptography does not solve problems, but only reduces them

- In public key cryptography, problems are reduced to:

Authentication of public keys

- How do we know whether PK_A actually belongs to Alice, when
  - we verify a signature with PK_{Alice}?
  - we establish a shared secret using PK_{Alice}?
  - we authenticate someone using PK_{Alice}?
- PK_{Alice} could actually be the public key of Trudy
- Need: authenticate link between public key and its owner
- In many practical systems this issue is not well addressed
  - one of main reasons for the miserable level of security in IT
  - same mistakes made again and again (see next slides)
  - problem of human behaviour rather than technology

Methods of public key authentication

Say, Bob wants to use Alice's public key

- He can obtain it via email, Alice's homepage, business card, ...

There are essentially three methods:

- Manual: Bob relies on Alice alone
- Web of trust: Bob relies on their mutual friends
- Certificate Authority (CA): Bob relies on a central authority

...and: Trust on First Use (TOFU): Bob knocks on wood

Systems for public key authentication (and revocation) are called Public Key Infrastructures (PKI). Most of the time, the term PKI is used as a synonym of the CA method.

Manual public key validation

- Bob checks with Alice if his copy of PK_A matches that of Alice
  - e.g., face-to-face, via phone or video-call
  - email will NOT do
  - requires that Bob verifies he is actually talking to Alice
- Often one uses a hash
  - verifying $h(PK_B, Id_B)$ instead of key PK_B directly
  - hash function shall be 2nd preimage resistant
  - reader-friendly coding of the hash: fingerprint
- Most reliable method
  - very rarely used
  - main problem: requires users to be security-aware
- a public key crypto pioneer: Phil Zimmerman
  - 1991: creates PGP secure email, supporting key validation
  - now: at Silent Circle (e.g. blackphone), settling with TOFU
  - *you cannot be idealistic all your life*

Web-of-trust public key authentication

Crowd style ("trust what your friends say", bottom-up)

- Say Alice and Bob have a common friend: Wally
  - Bob already has an authentic copy of PK_W: Wally's public key
  - Wally already verified that his copy of PK_A is authentic
  - Bob asks Wally to sign $\langle \text{Alice}, PK_A \rangle$ with his private key PrK_W
  - Bob can now verify this signature (certificate) using PK_W
- For more assurance, Bob can ask multiple friends to sign $\langle \text{Alice}, PK_A \rangle$
- Difference with symmetric-key case
  - symmetric: Wally has shared key and can cheat undetectedly
  - here Wally can sign $\langle \text{Alice}, PK_{W'} \rangle$ instead of $\langle \text{Alice}, PK_A \rangle$
  - . . . and can decipher Bob's messages and/or sign as Alice
  - but: Bob and Alice can catch Wally by manual validation
- Feature introduced by Phil Zimmerman in PGP
  - same problem: requires security-aware users
  - PGP (and gpg) usage in practice nowadays: mostly TOFU

Web of trust: signing parties

- People meet to check each other's identity
- and exchange public key fingerprints: (truncated) hashes of public keys (BJ's is `0xA45AFFF8`)
  - beware of 2nd preimages!
- to later look up the keys corresponding to the fingerprint and sign them

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

Certificate Authority

Phone-book style ("trust what an authority says", top-down)

- use a trusted list of pairs $\langle name,\ PK_{\text{name}} \rangle$
- but who can be trusted to compile and maintain such a list?
- this is done by a Certificate Authority (CA)
  - a *super-Wally* that signs public keys to be trusted by everyone
- Basic notion: public key certificate, i.e. signed statement:

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

- There are standardised formats for certificates, like X.509
- The term (public key) certificate is often abused

Public Key Infrastructure (PKI)

- Certification Authority (CA)
  - generates public key certificates
  - publishes certificate revocation lists for compromised keys
  - can be done in multiple levels: root CA and intermediate ones
- Registration Authority
  - part of CA that verifies the identity of the user
  - expensive part: administrative and legal aspects
- Most CAs are commercial companies, like VeriSign, Thawte, Comodo, or DigiNotar (now "dead")
- 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

(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"

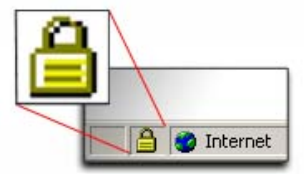
(3) Class 3 certificate: "assurances of the identity of the Subscriber based on the personal (physical) presence of the Subscriber to confirm his or her identity using, at a minimum, a well-recognized form of government-issued identification and one other identification credential."

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.*
- The root public keys are pre-configured, typically in browsers.
  - Often called "root certificates", but they aren't
  - E.g., in *firefox* look under Preferences - Advanced - View Certificates
  - On the web:
    `www.mozilla.org/projects/security/certs/included`

Certificate (PKI) usage examples

- "Secure webaccess" via server-side certificates, recognisable via:



  - Protocols: TLS and https
  - Allows user to authenticate website content
  - Protects confidentiality of web traffic between user and site
  - Allows continued usage of passwords and card nr. based credit car payments
- Code signing, for integrity and authenticity of downloaded code
- EMV payment with smart cards: VISA, Mastercard, Maestro
- Client-side certificates for secure remote logic (e.g., in VPN = Virtual Private Network)
- National ID cards and travel passports
- Sensor-certificates in a sensor network, against spoofing sensors and/or sensor data

Certificate Revocation, via CRLs

Revocation: declaring a public key certificate no longer valid

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 (e.g., 24 hours)
- should be consulted before every use of a certificate
- you can subscribe to revocation lists so that they are loaded automatically into your browser

This is the theory, in practice there is little follow-up

Revocation, via OCSP

- In off-line checking, CRLs require bandwidth and local storage
  - overflowing the list is possible attack scenario
- Alternative: OCSP = Online Certificate Status Protocol

(1) Suppose Bob wants to check Alice's certificate before use
(2) Bob sends OCSP request to CA with certificate serial nr.
(3) CA looks up serial number in its (supposedly) secure database
(4) if not revoked, it replies with a signed, successful OCSP response

- **Privacy issue**: with OCSP you reveal to CA which certificates you use, and thus who you communicate with
  - also when you communicate with someone using OCSP

Note: you are basically online with the CA, so long-term certificates are not really needed.

Certificate chains

Imagine you have certificates:

(1) ["A's public key is PK_A ..."]$_{PrK_B}$

(2) ["B's public key is PK_B ..."]$_{PrK_c}$

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

The trouble with PKI

- All participants need authentic copies of root CA public keys
  - a root CA cannot have a certificate, per definition
  - often does have a meaningless *self-signed certificate*
  - hardcoded in software or included in software releases
  - you are trusting Microsoft, Mozilla, Google, Apple, KPN ...
- Why most PKI's have failed up to now:
  - CAs in theory: trustworthy service providers that accept liability
  - CAs in practice: unreliable organizations only in it for the money
- Tension between (CA) PKI concept and the essence of public key crypto:
  - PK crypto: authentication and confidentiality without need for pre-shared keys or trusted third party
  - CA is nothing more than a trusted third party

Problems in the TLS (https) PKI

- In your browser there are about 650 CA root keys
  - Note: a common misnomer for CA root key is (CA) root certificate
  - whatever these CAs sign is shown as trusted by your browser
- 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 are rather weak:
  - 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
- root of the problem: lack of liability of software providers and CAs

CACert: attempt to merge web of trust with PKI

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

Weaknesses:

- CAcert is poorly run
- It never managed to set up an audit in order to get its root key into major browsers

Cool idea at first sight, but useless

- Security-aware users don't trust some self-declared CA
- CA is useful if liable, but CACert accepts none whatsoever

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

- What happened?
  - *F-secure* discovered a certificate over a key used to sign malware
  - the malware targeted governments and defense industry
  - certificate was provided by the CA *DigiCert* (Malaysia)
  - result: Mozilla and Microsoft blocked this CA
- certified public key was RSA key with modulus of 512 bits
  - *Fox-IT* also found such malware (for "infiltrating high-value targets") and claims that public keys have been brute-forced
  - required time to factor 512-bit modulus: hours-weeks (depending on hardware)
  - malware signed with the resulting private key
- 512-bit RSA keys accepted by a CA and in browsers as late as 2011: total incompetence

DigiNotar I: background

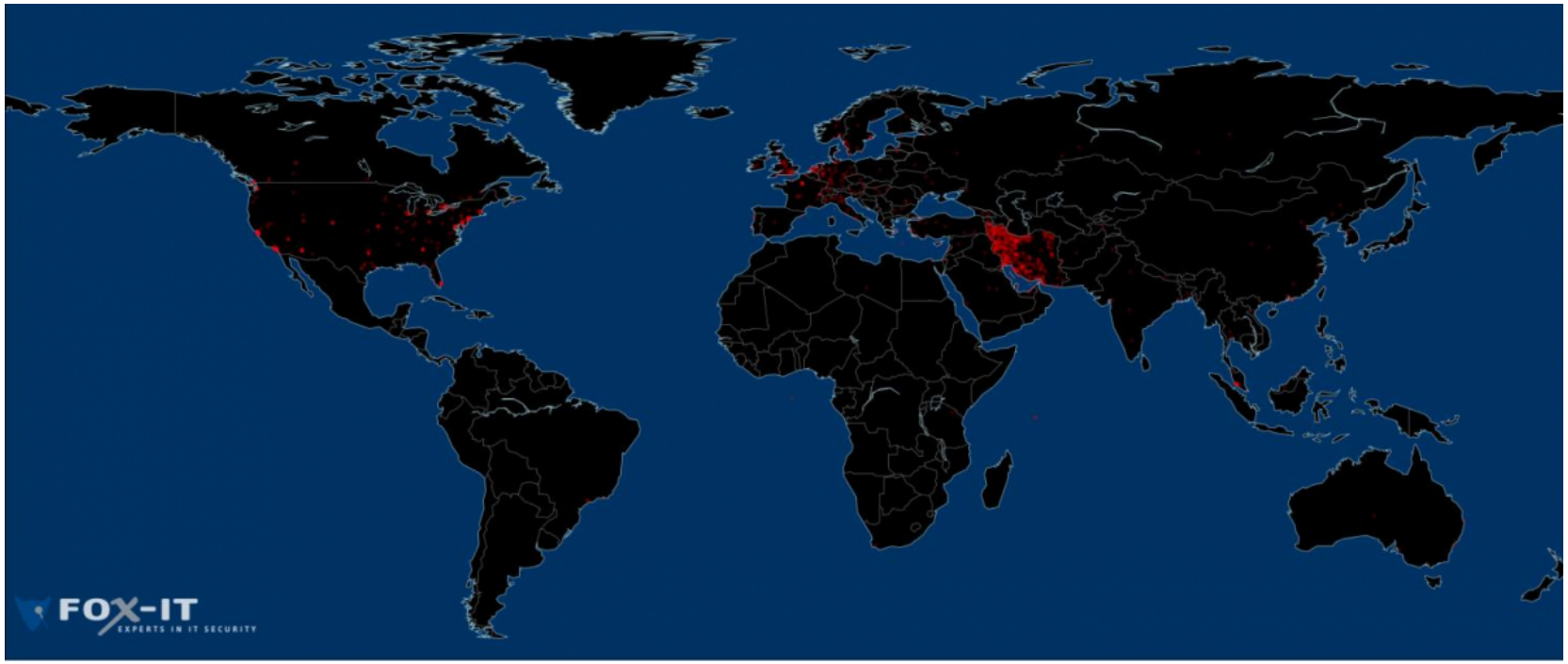
- The Dutch CA DigiNotar was founded in 1997, based on need for certificates among notaries
  - bought by US company *VASCO* in jan'11
  - "voluntary" bankruptcy in sept.'11
- DigiNotar's computer systems were infiltrated in mid july'11, resulting in rogue certificates
  - *DotNetNuke* CMS software was 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 possible 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 (see 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 if you still believe in CA's

- 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
  - e.g., 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

    (e.g., 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!

Trust on first use (TOFU)

Per default, no public key validation

- Bob trusts that received public key is Alice's without validation
- Man-in-the-middle risk: Eve can substitute public key by hers
- Used by the cool crowd:
  - messaging service Signal
  - messaging service Whatsapp
  - secure mobile blackphone from Silent Circle
  - ...
- Sometimes presented as alternative to PKI
- How is it possible that people buy this nonsense?
  - it promises security without the effort, a.o., key management
  - similar to voting for populists and expecting improvement
  - or eating chocolate to feel better
- It is not all bad: systems do support manual key validation

Example of TOFU: WhatsApp

- There is a white paper describing the security protocol
  - not enough detail to know what they are doing exactly
  - e.g. what happens when replacing phone?
  - complex protocol with 4 layers of ECC and 3 of symmetric crypto
- Uses ECC public key pairs to establish symmetric keys
  - public key pairs generated at install time
  - distributed via central WhatsApp server without validation
- Manual validation by select *contact*, item *encryption*
  - not transparent nor user-friendly
- Preliminary conclusion
  - Reading WhatsApp white paper rings loud bells
  - a critical review / reverse engineering is strongly desirable

Entity authentication with electronic signatures

Challenge-response with electronic signature:

$$A \longrightarrow B\colon N, Id_A$$
$$B \longrightarrow A\colon [N, Id_A]_{PrK_B}$$

or mutual authentication

$$A \longrightarrow B\colon N_B, Id_A$$
$$B \longrightarrow A\colon [N_B, Id_A]_{PrK_B}, N_A, Id_B$$
$$A \longrightarrow B\colon [N_A, Id_B]_{PrK_A}$$

- Advantage: verifier does not require secret!
  - Prover does not need to trust verifier for protecting its keys
  - Same private key can be used to authenticate in several places
  - This creates privacy issues: linkability

The myth of non-repudiation

- Unique advantage of asymmetric crypto :
  - verification of public key signature does not require a secret key
  - so only the signer could have generated the signature
- Public-key advocates have used this to promote their crypto:

Public-key signatures support non-repudiation

Non-repudiation: inability after signing something to deny it

- Attributing a legal/business property to a cryptographic protocol
- Excuses for denying signature include, a.o.,
  - someone else used the private key on my PC or smart card
  - I did sign but not the document you are showing me
  - the crypto has been broken
  - ...
- It is about rules, terms and conditions and agreeing with them

Electronic vs. ordinary signatures

- **Ordinary signature**
  - produced by human, expressing clear intent
  - the same on all documents
  - one person typically has one signature
  - easy to forge, but embedded in established usage context
- **Electronic signature**
  - different for each signed document
  - person may have multiple key pairs, e.g., 1 business, 1 personal
  - electronic signatures can be legally recognized
    - In Europe: EU directive 1999/93/EC
    - requires certified secure signature-creation device
    - in practice: an ID chip card containing private key(s)
    - legal validity implies PKI with government-approved CA
    - conditions for NL at pkioverheid.nl
  - crypto is mature, deployment still problematic

Electronic signatures, the ID chip card

- The private keys should at all time be under control of the user
  - the ID card signs a string presented to it with its private key(s)
  - requires prior submission of a PIN
  - retrieving the private key from the chip should be hard
  - key pairs should be generated on-card
  - this makes generating certificates very problematic: how can the CA know the public key has been generated on a valid chip?
- In the design one anticipated two main use cases:
  - entity authentication with challenge-response: for access to web sites, infrastructure, etc.
  - document signing, where a hash is presented to a card
  - A user should be in control of whether he does one or the other

Electronic signatures, the ID chip card (cont'd)

- Two key pairs:
  - one for *active authentication*
  - one for *non-repudiation* (sic)
- each key has its own PIN
  - so the user is in principle aware of what (s)he is doing
- a more cost-effective solution
  - a single key pair for both operations
  - two separate PINs for the functions
  - distinguish hashes (sign) from challenges (auth) with domain separation
- Scenario upon presentation of x to chip (single-key case)
  - x can be $h(m)$ or a challenge
  - if *sign* PIN was presented, chip returns $[x|0]_{PrK}$
  - if *auth* PIN was presented, chip returns $[x|1]_{PrK}$
  - if no valid PIN was presented, chip returns error

Electronic signatures, the user interface

- Classical approach: card reader with IC card connected to PC
  - PC has dedicated signing software, e.g., as plugin for a mail client
  - guidance is done on PC screen
  - input must be done done on PC keyboard
- Lots of attack possibilities in the PC
  - intercept PINs, for signing without the card owner
  - show a different message on the screen, etc.
- attempt at dealing with PC problem
  - tamper-evident, dedicated, non-updateble signature devices
  - like e-book readers, with only a screen, card reader and keypad
  - simplicity and limited functionality allows getting security assurance for such a device
  - not cool: public would prefer a *secure* app on their smartphone

Example of modern card reader with pin pad

- For use with German e-Identity card *neue Personalausweis (nPA)*
- Interfaces for both contact and contactless cards
- Certified by BSI; cost: 30-50 €

Server-side signatures (beware of snake-oil)

- So far we have assumed that the signer has his private keys locally
  - solid: he signs with ID chip card in dedicated card reader
  - less solid: he signs with his smartphone or laptop
  - concerns: leakage of key pair or loss of private key
- Server-side signature approach:
  - private key is (in secure hardware module) on some remote server
  - keys very well protected against leakage and loss
  - signer authenticates to server, and then pushes *sign* button
  - attempt to address non-repudiation myth
- Problems of server-side signatures
  - can the **sysadmin** sign on behalf of everyone else?
  - strong user authentication requires secret keys anyway
  - example: *Digidentity*
    - uses one-time-password via SMS as user authentication
    - recognized as qualified signatures (what a wonderful world!)

Discrete logarithm in $(\mathbb{Z}_p^*, \times)$ with prime p

Remember: $(\mathbb{Z}_p^*, \times)$ is just $(\mathbb{Z}_{p-1}, +)$ in disguise!

- Let g be a generator of $(\mathbb{Z}_p^*, \times)$
- Let $A = g^a$ and $B = g^b$
  - then $A \times B = g^a \times g^b = g^{a+b \bmod p-1}$
  - multiplication $A \times B$ reduces to addition $a + b$
  - exponentiation A^e reduces to multiplication $a \cdot e$
- Requires knowledge of exponent a (and b), given A (and B)
- Finding this exponent is called discrete log
- Discrete log is hard if p is large

Example:

- discrete exp: find X that satisfies $X \equiv 2^{95} \pmod{149}$
- discrete log: find x that satisfies $2^x \equiv 124 \pmod{149}$

Discrete logarithm problem

Discrete log problem in a cyclic group $\langle g \rangle$

Given $h \in \langle g \rangle$, finding $n < \#g$ that satisfies $h = g^n$

- The discrete log problem is hard in $(\mathbb{Z}_p^*, \times)$ for large p
  - solving a discrete log problem modulo p with p an n-bit prime is about as hard as factoring an n-bit RSA modulus
- It is also hard for many other groups, e.g.,
  - in cyclic subgroups of large order q of $(\mathbb{Z}_p^*, \times)$ with $q \lll p$
  - elliptic curve groups
- Elliptic curve cryptography (ECC) (see later)
  - discrete log in ECC is much harder than for $(\mathbb{Z}_p^*, \times)$
  - for same security strength, compared to RSA:
    - shorter keys, signatures and cryptograms
    - faster key establishment, signing and key pair generation
    - but slower signature verification

Discrete log based crypto: key pairs

- Key pairs:
  - private key: $a \in \mathbb{Z}_{\#g}$
  - public key: $A = g^a \in \langle g \rangle$
  - domain parameters: $\langle g \rangle$, the cyclic group we work in
- Similarities with RSA
  - computing private key from public key is hard problem
  - public key authentication is crucial for security
  - there is mathematical structure
- Differences with RSA
  - domain parameters: you don't have that in RSA
  - key pair generation: take random a and compute $A = g^a$
- Key pairs for $(\mathbb{Z}_p^*, \times)$
  - private key: $a \in \mathbb{Z}_{p-1}$
  - public key: $A = g^a \in \mathbb{Z}_p^*$
  - domain parameters: p and g

Ralph Merkle, Martin Hellman, Whitfield Diffie

Invented public key cryptography in 1976!

(Merkle)-Diffie-Hellman key exchange

- public-key based establishment of a shared secret
- Alice and Bob establish a secret key K_{AB}
  - Alice has $PrK_{\text{Alice}} = a$ and $PK_{\text{Alice}} = A\ (= g^a)$
  - Bob has $PrK_{\text{Bob}} = b$ and $PK_{\text{Bob}} = B\ (= g^b)$
- The protocol (simple static flavour): exchange of public keys

$$\text{Alice} \longrightarrow \text{Bob}: A$$
$$\text{Bob} \longrightarrow \text{Alice}: B$$

- Computation of the shared secret:
  - Bob uses his private key b to compute $K_{AB} = A^b$
  - Alice uses her private key a to compute $K_{AB} = B^a$
  - Correctness: $A^b = (g^a)^b = g^{a \cdot b} = (g^b)^a = B^a$

Diffie-Hellman key exchange: attention points

- Security
  - eavesdropper Eve needs either a or b to compute K_{AB}
  - given $\langle g \rangle, A$ and B, predicting K_{AB} should be hard
  - called the (decisional) Diffie-Hellman hardness assumption
  - seems as hard as the discrete log problem but no proof (yet)
- Domain parameters: both need to work in the same cyclic group
- Public key authentication
  - If Alice validated Bob's public key, she knows only Bob has K_{AB}
  - If Bob validated Alice's public key, he knows only Alice has K_{AB}
- Entity authentication?
  - can be done with symmetric crypto challenge-response using K_{AB}
  - along with encryption, message authentication
  - often one uses $h(\texttt{IntegerToString}(K_{AB}))$ for deriving symmetric keys from K_{AB}

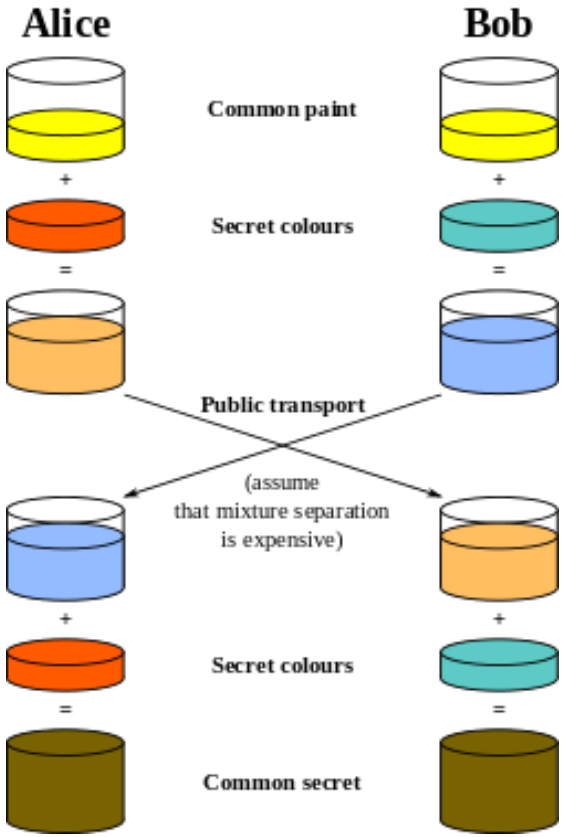
Diffie-Hellman key exchange: cases

- Mutual authentication: both parties authenticate public keys
- Unilateral authentication:
  - Alice authenticates Bob's public key but not vice versa
  - Alice still has guarantee that Bob is only other party having K_{AB}
  - only Bob can decipher what she enciphers with K_{AB}
  - only Bob can generate MACs with K_{AB}
- TLS (https) mostly uses unilateral authentication
  - browser authenticates public key of website
  - website does not authentication public key of browser
- Static Diffie-Hellman: Alice and Bob have long-term keys
  - limitation: K_{AB} is always the same
  - for symmetric crypto: requires nonces across multiple sessions
  - leakage of K_{AB}, a or b allows decryption of all past cryptograms
  - wish for forward secrecy: leakage of K_{AB}, a or b not affecting past cryptograms

Diffie-Hellman key exchange with forward secrecy

- Consider unilateral case where Bob does not validate Alice's key
  - Alice can generate fresh keypair (a, A) for each session/message
  - this is called an ephemeral key pair
  - leaking K_{AB} or a only affects single session/message
  - leaking b still affects all communication between Alice and Bob
- Dynamic Diffie-Hellman
  - Alice generates ephemeral key pair (a, A) per session
  - Bob generates ephemeral key pair (b, B) per session
  - auth. of A: Alice signs (Alice, A, N) with long term PrK_A
  - Bob verifies Alice's signature using the validated PK_A
  - in mutual authentication: also vice versa
  - now leakage of K_{AB}, a or b only affects a single session
  - after the session Alice deletes K_{AB} and a, Bob deletes K_{AB} and b
  - this offers forward secrecy
- Ephemeral key pairs in RSA would work too but very expensive

Diffie-Hellman explained via mixing of colours



(source: Wikipedia)

Diffie-Hellman in action: e-passports

- We saw the Basic Access Control (BAC) protocol for e-passports
  - terminal access to passport chip via Machine Readable Zone (MRZ)
  - restricted to less sensitive data, also on 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), using ephemeral Diffie-Hellman
    - Terminal Authentication (TA), using certificates: for giving access to biometric data
  - Here we sketch how CA works

Chip Authentication (from EAC)

$$PsP \xrightarrow[(s_p \text{ is static DH key})]{g^{s_p}} Rdr$$

$$PsP \xleftarrow[(s_R \text{ is ephemeral DH key})]{g^{s_R}} Rdr$$

$K = g^{s_P s_R}$:fresh shared secret;
derived to two keys: $K_{\text{enc}}, K_{\text{mac}}$

$$PsP \xrightarrow{K_{\text{mac}}\{g^{s_R}\}} Rdr$$

Rdr now authenticated PsP as it knows

- PsP must have shared secret K
- so PsP has private key s_P matching the public key g^{s_P}

NSA breaking encrypted connections

CCS 2015 paper *Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice* explains:

- Diffie-Hellman is used for VPNs, https websites, email, etc.
- Many implementation use the same domain parameters
  - a 1024 bit prime p
  - a particular generator $g \in \mathbb{Z}_p$
- A very large look-up table can be compiled
  - to efficiently solve discrete log in this group
  - authors estimate that this could be done for $100M
  - NSA may have budget for that
- This could explain suggestions in Snowden documents that the NSA has access to encrypted connections.

Student feedback after exam in 2012

El Gamal: discrete-log based encryption

Encryption with public key A

- convert cleartext M to element $m \in \langle g \rangle$
- randomly generate ephemeral key pair $(r, R = g^r)$
- define cryptogram as $\{m\}_A = (R,\ m \cdot A^r)$
- multiplying m with random A^r and giving R as side info

Decryption with private key a

- Assume ciphertext $c = (c_1, c_2)$, with $c_i \in \langle g \rangle$
- define recovered plaintext as $[(c_1, c_2)]_a = \dfrac{c_2}{(c_1)^a}$
- removing the factor A^r by dividing by $R^a = A^r$

Correctness

- For $A = g^a$ we get:

$$[\{m\}_A]_a = [R, m \cdot (g^a)^r]_a = \frac{m \cdot g^{a \cdot r}}{(g^r)^a} = \frac{m \cdot g^{a \cdot r}}{g^{a \cdot r}} = m$$

El Gamal style signature (AKA DSA)

Signing with private key a of message m

- randomly generate ephemeral key pair $(r, R = g^r)$ with $\gcd(r, \#g) = 1$

$$\text{sign}_a(m) = \left(R,\ \frac{h(m) - a \cdot R}{r} \bmod \#g \right)$$

Verification of $m, (s_1, s_2)$ with public key $A \in \langle g \rangle$

- check the equation:

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

Notice: no decryption, just checking

Correctness

- $r \cdot s_2 \equiv h(m) - a \cdot R = h(m) - a \cdot s_1 \bmod \#g$ so that:
- $h(m) \equiv r \cdot s_2 + a \cdot s_1 \pmod{\#g}$ and so:
- $g^{h(m)} = g^{r \cdot s_2 + a \cdot s_1} = (g^r)^{s_2} \cdot (g^a)^{s_1} = R^{s_2} \cdot (g^a)^{s_1} = (s_1)^{s_2} \cdot A^{s_1}$

Example calculation I

Take $G = \mathbb{Z}_p$ for $p = 107$ and $g = 10 \in G$ with order $q = 53$.

- **Keys:** private $x = 16$; public $y = g^x = 10^{16} = 69 \bmod 107$
- **Encryption:** of $m = 100 \in G$ with random $r = 42$ gives:

$$(g^r, y^r \cdot m) = (10^{42}, 69^{42} \cdot 100) = (4, 11)$$

- **Decryption:** of $(4, 11)$ is $\frac{11}{4^x}$
  - $4^x = 4^{16} = 29$ and $\frac{1}{29} = 48 \bmod 107$
  - Hence $\frac{11}{4^x} = 11 \cdot 48 = 100 \bmod 107$

(For modular calculation use eg: http://ptrow.com/perl/calculator.pl)

Example calculation II

Still with the same $p = 107, g = 10, q = 53, x = 16, y = 69$,

- **Sign:** $H(m) = 100$ with random $r = 33$
  - We have $g^r = 10^{33} = 102 \bmod 107$
  - and: $\frac{1}{r} = \frac{1}{33} = 45 \bmod 53$
  - next:

$$\frac{H(m) - x \cdot g^r}{r} = (100 - 16 \cdot 102) \cdot 45 = 5 \cdot 45 = 13 \bmod 53$$

  - The signature is thus: $(102, 13)$.
- **Verification:** of $(s_1, s_2) = (102, 13)$
  - first, $g^{H(m)} = 10^{100} = 34 \bmod 107$
  - and also: $(s_1)^{s_2} \cdot y^{s_1} = 102^{13} \cdot 69^{102} = 62 \cdot 4 = 34 \bmod 107$.

Background, for mathematicians only

- The primes $p = 107$ and $q = 53$ in the example satisfy $p = 2q + 1$
- We said we use $G = \mathbb{Z}_p$, but actually it's $G = \mathbb{Z}_p^*$
- The order of $\mathbb{Z}_p^*$ is $p - 1 = 2q$
- In general, if $g \in G$ is of order q, then it corresponds to a subgroup of G of order q, generated by $g^i \in G$
  - If this subgroup is of prime order q, then the "Decisional Diffie-Helmann" assumption is believed to hold
- Formally, we have an embedding of groups:

$$\mathbb{Z}_q \longrightarrow \mathbb{Z}_p^* = G \quad \text{given by} \quad i \longmapsto g^i$$

$\mathbb{Z}_q$ is identified with the subgroup $\langle g \rangle$ generated by G.

  - these exponents i have to computed modulo q

Background on Elliptic Curve Cryptography

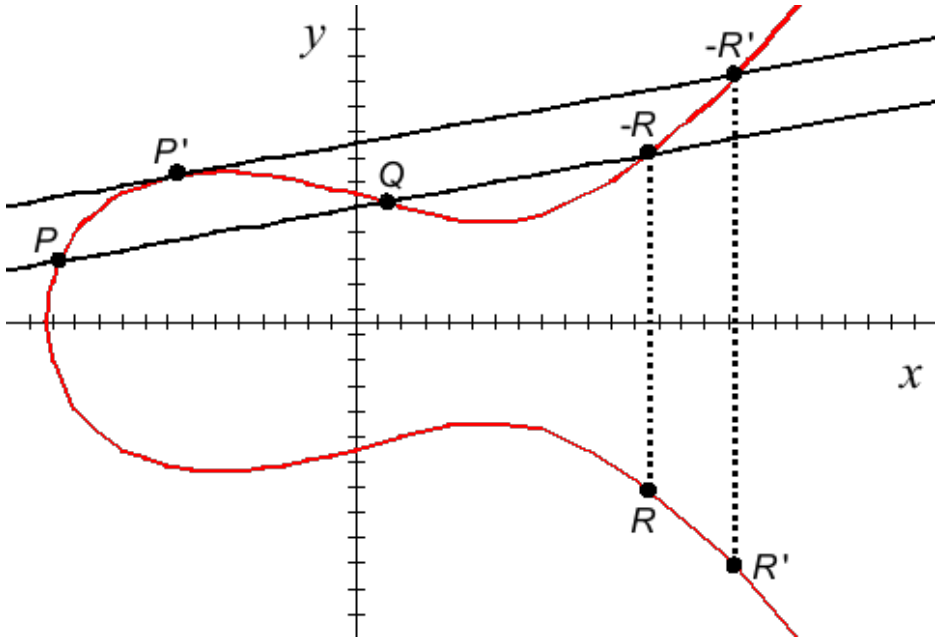
- Koblitz and Miller proposed the use of elliptic curves for cryptography in the mid 1980's
  - group operation is given by addition of points on a curve
  - mainstream public key crypto nowadays
- Provides the functionality of RSA and more
  - smaller keys
  - pairings (advanced, cool topic)
- Standard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

security strength	modulus length RSA	modulus length ECC
80	1024	160
128	3072	256
256	15360	512

Addition on an elliptic curve over the real numbers

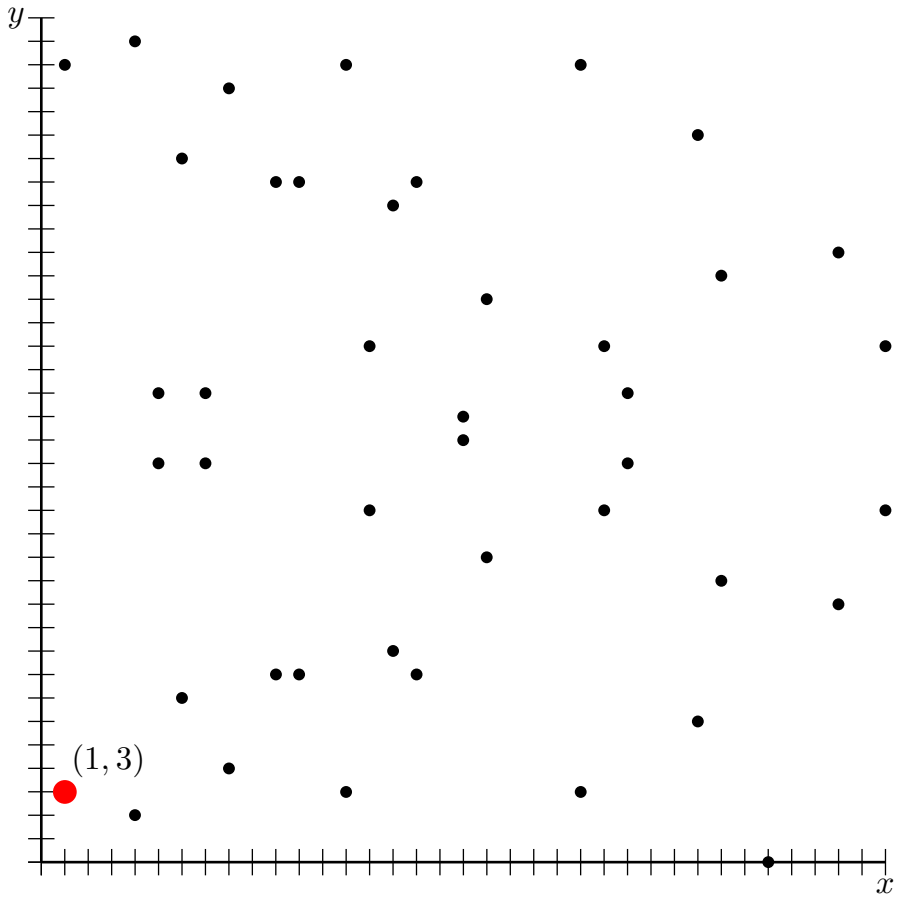
Elliptic curves are given by equations such as: $y^2 = x^3 + ax + b$

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

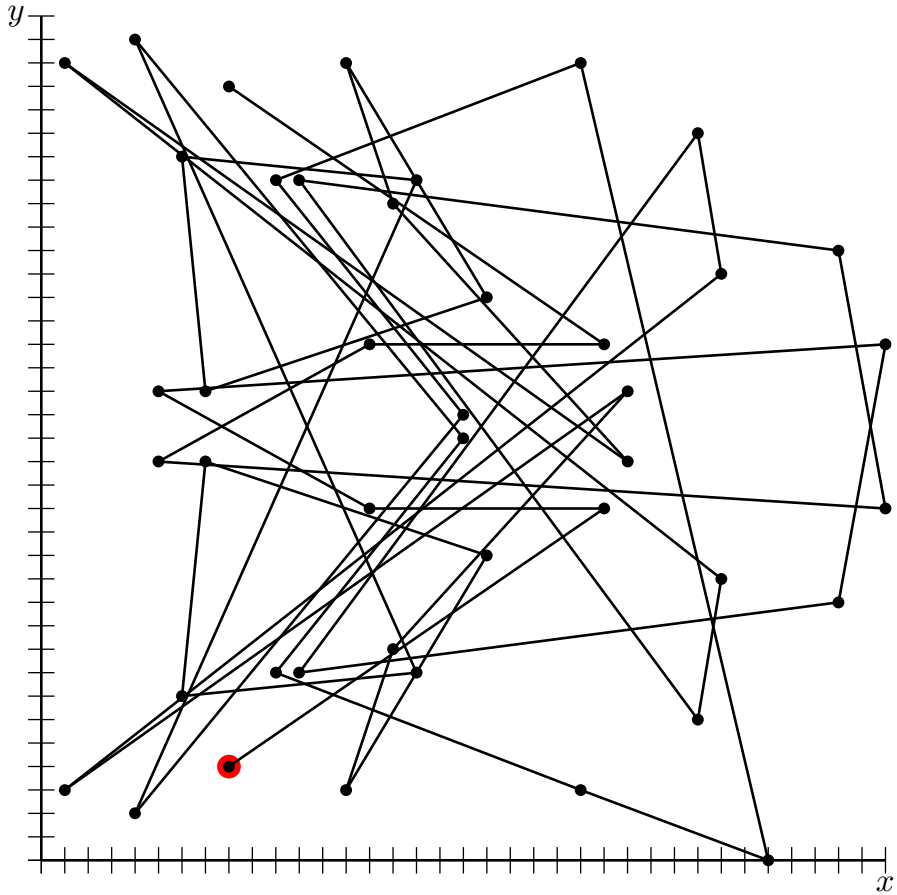


There are also explicit formulas for such additions.

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



Repeated addition: $n \cdot P$ goes everywhere



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

Discrete Log and public keys for ECC

ECC uses additive notation so discrete log problem looks a bit funny:

scalar multiplication: $[n] \cdot G = G + \cdots + G$
Given $[n] \cdot G$ and G, it is hard to find the scalar n.

Key pairs in ECC:

- Domain parameters: the prime p, the constants a and b, generator G and its order $\#G$
- Private key: an integer $a \in \mathbb{Z}_{\#G}$
- Public key: a point on the curve $A = [n]G$

On PGP

Use fresh session key K for efficiency:

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

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

Needham-Schroeder two-way authentication

- Originally proposed in 1978
- uses RSA encryption to achieve authentication
- Serious flaw discovered only in 1996 by Gavin Lowe
  - required formal methods, namely model checking
- Can simply be fixed
- Fix can be seen as just applying appropriate domain separation

Needham-Schroeder: original version + attack

Protocol

$$A \longrightarrow B: \{A, N_A\}_{e_B}$$
$$B \longrightarrow A: \{N_A, N_B\}_{e_A}$$
$$A \longrightarrow B: \{N_B\}_{e_B}$$

Attack

$$A \longrightarrow T: \{A, N_A\}_{e_T}$$
$$T \longrightarrow B: \{A, N_A\}_{e_B}$$
$$B \longrightarrow T: \{N_A, N_B\}_{e_A}$$
$$T \longrightarrow A: \{N_A, N_B\}_{e_A}$$
$$A \longrightarrow T: \{N_B\}_{e_T}$$
$$T \longrightarrow B: \{N_B\}_{e_B}$$

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: a fix

$$A \longrightarrow B: \{A, N_A\}_{e_B}$$
$$B \longrightarrow A: \{N_A, B, N_B\}_{e_A}$$
$$A \longrightarrow B: \{N_B\}_{e_B}$$

Signature variations

- Both sign and encrypt:

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

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", e.g., 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 losing control (see Bitcoin, later on)

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);
```