Build X Algorithms: Cryptography

Prof. Luca Viganò

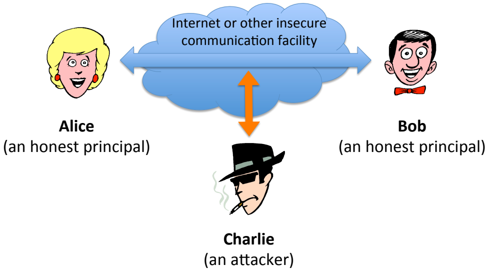
Department of Informatics
King's College London, UK

15 February 2016

Outline

Table of contents I

Cryptography in Network Security

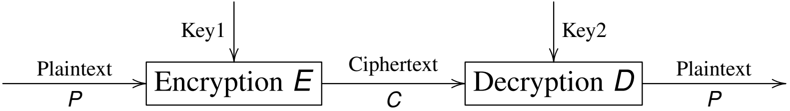


How do we turn an insecure communication facility (like the Internet) into a secure one?

Where security means that one of more security properties (e.g., confidentiality, integrity, authentication, non-repudiation, anonymity, unobservability, timeliness, availability, etc.) are guaranteed.

Cryptography is the enabling technology.

General cryptographic schema



where $E(\text{Key1}, P) = C$ and $D(\text{Key2}, C) = P$.

- **Symmetric algorithms**:
  - Key1 = Key2, or are easily derived from each other.
- **Asymmetric (or public key) algorithms**:
  - Different keys, which cannot be derived from each other.
  - Public key can be published without compromising private key.
- Encryption and decryption should be easy, if keys are known.
- **Security depends only on secrecy of the key, not on the algorithm.**

Encryption/decryption

- $\mathcal{A}$, the alphabet, is a finite set.
- $\mathcal{M} \subseteq \mathcal{A}^*$ is the message space. $M \in \mathcal{M}$ is a plaintext (message).
- $\mathcal{C}$ is the ciphertext space, whose alphabet may differ from $\mathcal{M}$.
- $\mathcal{K}$ denotes the key space of keys.
- Each $e \in \mathcal{K}$ determines a bijective function from $\mathcal{M}$ to $\mathcal{C}$, denoted by E_e. E_e is the encryption function (or transformation).

  Note: we will write $E_e(P) = C$ or, equivalently, $E(e, P) = C$.
- For each $d \in \mathcal{K}$, D_d denotes a bijection from $\mathcal{C}$ to $\mathcal{M}$. D_d is the decryption function.
- Applying E_e (or D_d) is called encryption (or decryption).

Encryption/decryption (cont.)

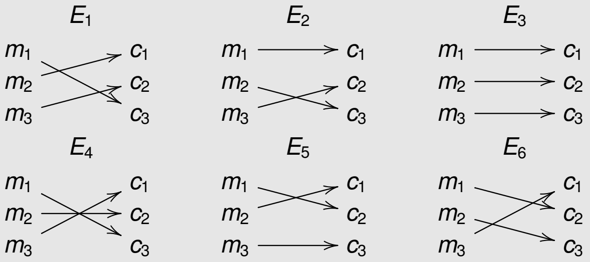
- An encryption scheme (or cipher) consists of a set $\{E_e \mid e \in \mathcal{K}\}$ and a corresponding set $\{D_d \mid d \in \mathcal{K}\}$ with the property that for each $e \in \mathcal{K}$ there is a unique $d \in \mathcal{K}$ such that $D_d = E_e^{-1}$; i.e.,

$$D_d(E_e(m)) = m \qquad \text{for all } m \in \mathcal{M}\,.$$

- The keys e and d above form a key pair, sometimes denoted by (e, d). They can be identical (i.e., the symmetric key).
- To construct an encryption scheme requires fixing a message space $\mathcal{M}$, a ciphertext space $\mathcal{C}$, and a key space $\mathcal{K}$, as well as encryption transformations $\{E_e \mid e \in \mathcal{K}\}$ and corresponding decryption transformations $\{D_d \mid d \in \mathcal{K}\}$.

An example

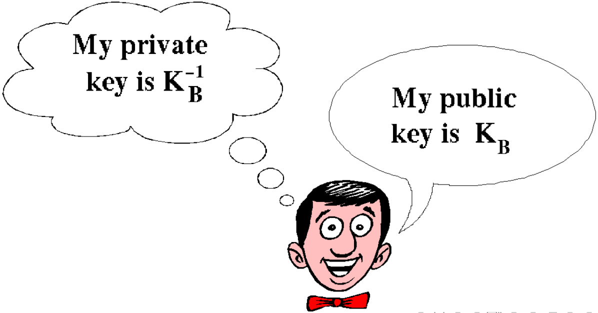
Let $\mathcal{M} = \{m_1, m_2, m_3\}$ and $\mathcal{C} = \{c_1, c_2, c_3\}$.
There are 3! = 6 bijections from $\mathcal{M}$ to $\mathcal{C}$.
The key space $\mathcal{K} = \{1, 2, 3, 4, 5, 6\}$ specifies these transformations.



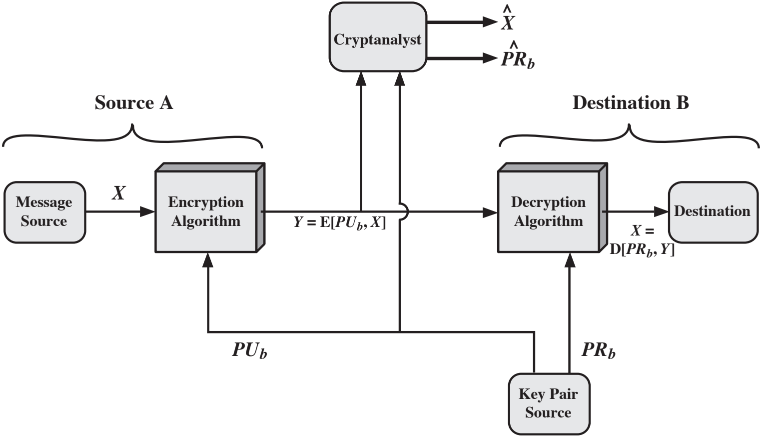
Suppose Alice and Bob agree on the transformation E_1.
To encrypt m_1, Alice computes $E_1(m_1) = c_3$.
Bob decrypts c_3 by reversing the arrows on the diagram for E_1 and observing that c_3 points to m_1.

Public-key cryptography

- Let $\{E_e \mid e \in \mathcal{K}\}$ and $\{D_d \mid d \in \mathcal{K}\}$ form an encryption scheme.
- Consider transformation pairs (E_e, D_d) where knowing E_e it is infeasible, given $c \in \mathcal{C}$, to find an $m \in \mathcal{M}$ such that $E_e(m) = c$.
- This implies it is **infeasible to determine** d **from** e.
- Hence, E_e constitutes a trap-door one-way function with trapdoor d (as explained in more detail later).
- Called **public key** as e can be public information:



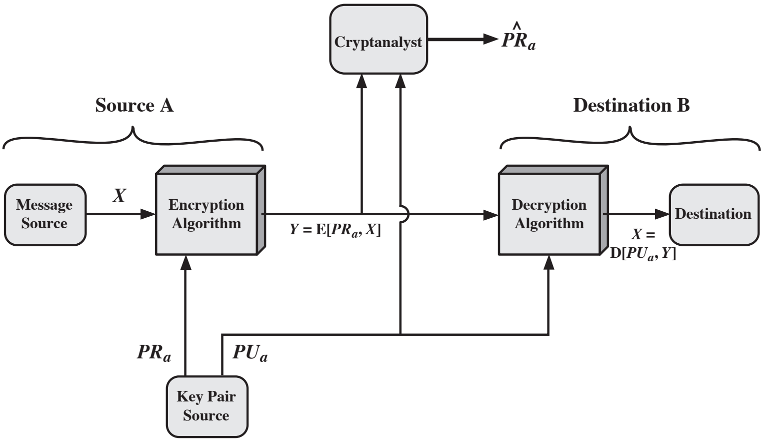
Public-key cryptosystem: secrecy (confidentiality)



Secrecy (confidentiality)

- X is a secret intended for B.
- Only B, who possesses PR_b, can decrypt $Y = E(PU_B, X)$.

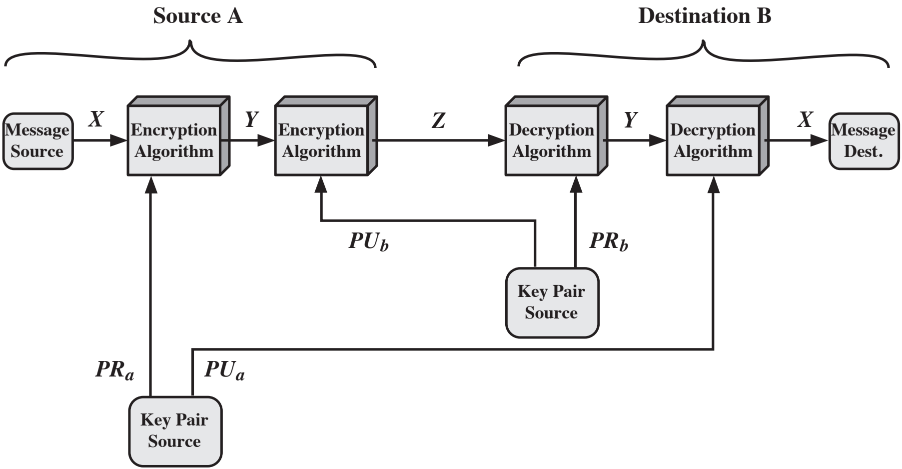
Public-key cryptosystem: authentication



Authentication

- Only A, who possesses PR_a, can have generated $Y = E(PR_A, X)$.
- Note that everybody can decrypt Y (and read X) as PU_a is public.

Public-key cryptosystem: secrecy and authentication



$$Z = E(PU_b, E(PR_a, X)) \quad \text{and} \quad X = D(PU_a, D(PR_b, Z))$$

Requirements for public-key cryptography

1. It is computationally easy for any principal B to generate a pair (public key PU_b, private key PR_b).
2. It is computationally easy for sender A, knowing PU_b and M, to generate
$$C = E(PU_b, M) \,.$$
3. It is computationally easy for receiver B to decrypt C using PR_b to recover M:
$$M = D(PR_b, C) = D(PR_b, E(PU_b, M)) \,.$$
4. It is computationally infeasible for an adversary
   - knowing PU_b to determine PR_b,
   - knowing PU_b and C to recover M.
5. (Useful, but not always necessary) The two keys can be applied in either order:
$$M = D(PU_b, E(PR_b, M)) = D(PR_b, E(PU_b, M)) \,.$$

Requirements for Public-Key Cryptography (cont.)

- These are difficult requirements.
- As a matter of fact only a few algorithms enjoying the above requirements have received widespread acceptance so far, e.g.,

Algorithm	**Encryption/Decryption**	**Digital signature**	**Key exchange**
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

- We will focus on RSA and Diffie-Hellman.

One-way function

One-way function

A function $f : X \to Y$ is a **one-way function**, if f is "easy" to compute for all $x \in X$, but f^{-1} is "hard" (or "infeasible") to compute.

- **Easy**: generally, defined to mean a problem that can be solved in polynomial time as a function of input length.
  - If input length is n bits, then time to compute function is proportional to n^a, where a is a fixed constant.
- **Infeasible**: effort to solve problem grows faster than polynomial time as a function of input size.
  - Time to compute function proportional to 2^n for input length n bits.
    - Difficult to determine if a particular algorithm exhibits this complexity.
    - Computational complexity traditionally focuses on worst-case or average-case complexity of an algorithm, but cryptography requires that it be infeasible to invert a function for virtually all inputs.

One-way function: examples

- **Square root.**
  - If you know $x = 512$, $f(x) = x^2 = 512^2 =$

One-way function: examples

- **Square root.**
  - If you know $x = 512$, $f(x) = x^2 = 512^2 = 262144$ is easy to compute.

One-way function: examples

- **Square root.**
  - If you know $x = 512$, $f(x) = x^2 = 512^2 = 262144$ is easy to compute.
  - If you know $f(x) = 262144$, $x = \sqrt{x^2} = \sqrt{262144}$ is difficult to compute.

One-way function: examples

- **Square root.**
  - If you know $x = 512$, $f(x) = x^2 = 512^2 = 262144$ is easy to compute.
  - If you know $f(x) = 262144$, $x = \sqrt{x^2} = \sqrt{262144}$ is difficult to compute.
- **Modular cube roots.**
  - Select primes $p = 48611$ and $q = 53993$.
  - Let $n = p \times q = 2624653723$ and $X = \{1, 2, \ldots, n-1\}$.
  - Define $f : X \to \mathbb{N}$ by $f(x) = x^3 \bmod n$.
  - Example: $f(2489991) = 1981394214$.
  - Computing f is easy.
  - Inverting f is hard: find x which is cubed and yields remainder!

One-way function: examples

- **Square root.**
  - If you know $x = 512$, $f(x) = x^2 = 512^2 = 262144$ is easy to compute.
  - If you know $f(x) = 262144$, $x = \sqrt{x^2} = \sqrt{262144}$ is difficult to compute.
- **Modular cube roots.**
  - Select primes $p = 48611$ and $q = 53993$.
  - Let $n = p \times q = 2624653723$ and $X = \{1, 2, \ldots, n-1\}$.
  - Define $f : X \to \mathbb{N}$ by $f(x) = x^3 \bmod n$.
  - Example: $f(2489991) = 1981394214$.
  - Computing f is easy.
  - Inverting f is hard: find x which is cubed and yields remainder!

This is useful because:

- Encryption is (very) easy whereas decryption is (very) difficult.
- The idea is: "$f(x)$ acts as a public key and x as a private key".

Trapdoor one-way function

- A trapdoor one-way function is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known.
  - With additional info, inverse can be calculated in polynomial time.

Trapdoor one-way function

A **trapdoor one-way function** is a one-way function $f_k : X \to Y$ where, given extra information k (the trapdoor information) it is feasible to find, for $y \in Image(f)$, an $x \in X$ where $f_k(x) = y$.

- Hence, a *trapdoor one-way function is a family of invertible functions* f_k such that computing

$Y = f_k(X)$	is easy if k and X are known
$X = f_k^{-1}(Y)$	is easy if k and Y are known
$X = f_k^{-1}(Y)$	is infeasible if Y is known but k is not known

- **Example:** Computing modular cube roots is easy when p and q are known (basic number theory).

Table of contents I

Prime factorization

- Numbers: **naturals** $\mathbb{N} = \{0, 1, 2, \ldots\}$, **integers** $\mathbb{Z} = \{0, 1, -1, \ldots\}$, **primes** $\mathcal{P} = \{2, 3, 5, 7, ...\}$.
- To **factor** a number a is to write it as a product of other numbers, e.g., $a = b \times c \times d$.
- Multiplying numbers is easy, factoring numbers appears hard. We cannot factor most numbers with more than 1024 bits.
- The **prime factorization** of a number a amounts to writing it as a product of powers of primes:

$$a = \prod_{p \in \mathcal{P}} p^{a_p} = 2^{a_2} \times 3^{a_3} \times 5^{a_5} \times 7^{a_7} \times 11^{a_{11}} \times \ldots \quad \text{where } a_p \in \mathbb{N}$$

  For any particular value of a, most of the exponents a_p will be 0, e.g.,

$$\begin{aligned} 91 &= 7 \times 13 \\ 3600 &= 2^4 \times 3^2 \times 5^2 \\ 11011 &= 7 \times 11^2 \times 13 \end{aligned}$$

Divisors

$a \neq 0$ **divides** b (written $a \mid b$) if there is an m such that $m \times a = b$.

- Examples: $3 \mid 6$ and $7 \mid 21$.

a **does not divide** b (written $a \nmid b$) if there is no m such that $m \times a = b$.

- Examples: $3 \nmid 7$, $3 \nmid 10$ and $7 \nmid 22$.

Relatively prime numbers & greatest common divisor

Two natural numbers a, b are **relatively prime** if they have no common divisors/factors apart from 1, i.e., if their **greatest common divisor** gcd is equal to 1

$$\gcd(a, b) = 1\,.$$

Relatively prime numbers & greatest common divisor

Two natural numbers a, b are **relatively prime** if they have no common divisors/factors apart from 1, i.e., if their **greatest common divisor** gcd is equal to 1

$$\gcd(a, b) = 1 \,.$$

- For example, 8 and 15 are relatively prime since

Relatively prime numbers & greatest common divisor

Two natural numbers *a*, *b* are **relatively prime** if they have no common divisors/factors apart from 1, i.e., if their **greatest common divisor** gcd is equal to 1

$$\gcd(a, b) = 1 .$$

- For example, 8 and 15 are relatively prime since
  - factors of 8 are 1, 2, 4, 8,

Relatively prime numbers & greatest common divisor

Two natural numbers a, b are **relatively prime** if they have no common divisors/factors apart from 1, i.e., if their **greatest common divisor** gcd is equal to 1

$$\gcd(a, b) = 1\,.$$

- For example, 8 and 15 are relatively prime since
  - factors of 8 are 1, 2, 4, 8,
  - factors of 15 are 1, 3, 5, 15,

Relatively prime numbers & greatest common divisor

Two natural numbers a, b are **relatively prime** if they have no common divisors/factors apart from 1, i.e., if their **greatest common divisor** gcd is equal to 1

$$\gcd(a, b) = 1 .$$

- For example, 8 and 15 are relatively prime since
  - factors of 8 are 1, 2, 4, 8,
  - factors of 15 are 1, 3, 5, 15,
  - and 1 is the only common factor.

Relatively prime numbers & greatest common divisor

Two natural numbers *a*, *b* are **relatively prime** if they have no common divisors/factors apart from 1, i.e., if their **greatest common divisor** gcd is equal to 1

$$\gcd(a, b) = 1\,.$$

- For example, 8 and 15 are relatively prime since
  - factors of 8 are 1, 2, 4, 8,
  - factors of 15 are 1, 3, 5, 15,
  - and 1 is the only common factor.
- Conversely, we can determine the greatest common divisor by comparing their prime factorizations and using least powers, e.g.
  - $150 = 2^1 \times 3^1 \times 5^2$ and $18 = 2^1 \times 3^2$, thus $\gcd(18, 150) = 2^1 \times 3^1 \times 5^0 = 6$.

Relatively prime numbers & greatest common divisor

Two natural numbers *a*, *b* are **relatively prime** if they have no common divisors/factors apart from 1, i.e., if their **greatest common divisor** gcd is equal to 1

$$\gcd(a, b) = 1 \,.$$

- For example, 8 and 15 are relatively prime since
  - factors of 8 are 1, 2, 4, 8,
  - factors of 15 are 1, 3, 5, 15,
  - and 1 is the only common factor.
- Conversely, we can determine the greatest common divisor by comparing their prime factorizations and using least powers, e.g.
  - $150 = 2^1 \times 3^1 \times 5^2$ and $18 = 2^1 \times 3^2$, thus $\gcd(18, 150) = 2^1 \times 3^1 \times 5^0 = 6$.
  - $60 = 2^2 \times 3 \times 5$ and $14 = 2 \times 7$, thus $\gcd(60, 14) = 2$.

Table of contents I

Greatest common divisor and Euclid's algorithm

- gcd can be computed quickly using **Euclid's algorithm.**

$$\begin{array}{lcrcl} \mathrm{gcd}(60,14) & : & 60 & = & (4 \times 14) + 4 \\ \mathrm{gcd}(14,4) & : & 14 & = & (3 \times 4) + 2 \\ \mathrm{gcd}(4,2) & : & 4 & = & 2 \times 2 \end{array}$$

- **Extended Euclid's algorithm** computes $x, y \in \mathbb{Z}$ such that

$$\mathrm{gcd}(a,b) = (x \times a) + (y \times b)$$

Here $2 = 14 - 3 \times (60 - (4 \times 14)) = (-3 \times 60) + (13 \times 14)$

Euclid's Algorithm

Euclid's algorithm is based on the theorem

$\gcd(a, b) = \gcd(b, a \bmod b)$ for any nonnegative integer a and any positive integer b.

For example:

- $\gcd(55, 22) = \gcd(22, 55 \bmod 22) = \gcd(22, 11) = 11.$

Euclid's algorithm

Euclid(a, b)
1 **if** $b = 0$
2   **then return** a
3   **else return** Euclid(b, $a \bmod b$)

For example:

- Euclid$(30, 21)$ = Euclid$(21, 9)$ = Euclid$(9, 3)$ = Euclid$(3, 0)$ = 3.

Extended Euclid's Algorithm

Extend Euclid's algorithm to compute integer coefficients x, y such that

$$d = \gcd(a, b) = (a \times x) + (b \times y)$$

Extended Euclid's algorithm

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid$(b,\ a \bmod b)$
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

where $q = \lfloor a/b \rfloor$ is the **quotient of the division** (for $a = (q \times b) + r$).

Note: the *d* here is the greatest common **d**ivisor, not to be confused with the *d* that is (part of) an RSA private key (discussed later on).

Extended Euclid's Algorithm: example

$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y

Extended Euclid's Algorithm: example

Extended-Euclid$(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$

Extended-Euclid(a, b)
if $b = 0$
  then return $(a, 1, 0)$
$(d', x', y') \leftarrow$ Extended-Euclid$(b,\ a \bmod b)$
$(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
return (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78				

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
1 **if** $b = 0$
2   **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1			
78	21				

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1			
78	21	3			
21	15				

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
if $b = 0$
  then return $(a, 1, 0)$
$(d', x', y') \leftarrow$ Extended-Euclid$(b,\ a \bmod b)$
$(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
return (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1			
78	21	3			
21	15	1			
15	6				

Extended Euclid's Algorithm: example

Extended-Euclid$(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid$(b,\ a \bmod b)$
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1			
78	21	3			
21	15	1			
15	6	2			
6	3				

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1			
78	21	3			
21	15	1			
15	6	2			
6	3	2			
3	0				

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1			
78	21	3			
21	15	1			
15	6	2			
6	3	2			
3	0	–			

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
if $b = 0$
  then return $(a, 1, 0)$
$(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
$(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
return (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1			
78	21	3			
21	15	1			
15	6	2			
6	3	2			
3	0	–	3	1	0

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
if $b = 0$
  then return $(a, 1, 0)$
$(d', x', y') \leftarrow$ Extended-Euclid$(b,\ a \bmod b)$
$(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
return (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3		
78	21	3	3		
21	15	1	3		
15	6	2	3		
6	3	2	3		
3	0	–	3	1	0

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3		
78	21	3	3		
21	15	1	3		
15	6	2	3		
6	3	2	3	0	1
3	0	–	3	1	0

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3		
78	21	3	3		
21	15	1	3		
15	6	2	3	1	−2
6	3	2	3	0	1
3	0	–	3	1	0

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
if $b = 0$
  then return $(a, 1, 0)$
$(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
$(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
return (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3		
78	21	3	3		
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	–	3	1	0

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3		
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	–	3	1	0

Extended Euclid's Algorithm: example

$$\text{Extended-Euclid}(99, 78) = 3 = (99 \times (-11)) + (78 \times 14)$$

Extended-Euclid(a, b)
1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ Extended-Euclid(b, $a \bmod b$)
4 $(d, x, y) \leftarrow (d', y', x' - (\lfloor a/b \rfloor \times y'))$
5 **return** (d, x, y)

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3	−11	14
78	21	3	3	3	−11
21	15	1	3	−2	3
15	6	2	3	1	−2
6	3	2	3	0	1
3	0	–	3	1	0

Each line shows one level of the recursion.

Table of contents I

Modular arithmetics

Remainder

- $\forall a\, n.\ \exists q\, r.\ (a = (q \times n) + r)$ where $0 \leq r < n$.

  Here r is the **remainder**, which we write as

$$r = a \bmod n\,.$$

Congruent modulo

- $a, b \in \mathbb{Z}$ are **congruent modulo** n, if $a \bmod n = b \bmod n$.

  We write this as

$$a =_n b\,.$$

Modulo operator has following properties (of congruences)

- Reflexivity: $a =_n a$.
- Symmetry: If $a =_n b$ then $b =_n a$.
- Transitivity: If $(a =_n b$ and $b =_n c)$ then $a =_n c$.

Other properties of the modulo operator

$(a \bullet b) =_n (a \bmod n) \bullet (b \bmod n) \qquad \text{for } \bullet \in \{+, -, \times\}$

i.e., $(a \bullet b) \bmod n = [(a \bmod n) \bullet (b \bmod n)] \bmod n$

Example:

$$2 = (5 \times 6) \bmod 4$$

Other properties of the modulo operator

$(a \bullet b) =_n (a \bmod n) \bullet (b \bmod n) \qquad \text{for } \bullet \in \{+, -, \times\}$

i.e., $(a \bullet b) \bmod n = [(a \bmod n) \bullet (b \bmod n)] \bmod n$

Example:

$$\begin{aligned} 2 &= (5 \times 6) \bmod 4 \\ &= [(5 \bmod 4) \times (6 \bmod 4)] \bmod 4 \end{aligned}$$

Other properties of the modulo operator

$(a \bullet b) =_n (a \bmod n) \bullet (b \bmod n) \qquad \text{for } \bullet \in \{+, -, \times\}$

i.e., $(a \bullet b) \bmod n = [(a \bmod n) \bullet (b \bmod n)] \bmod n$

Example:

$$\begin{aligned} 2 &= (5 \times 6) \bmod 4 \\ &= [(5 \bmod 4) \times (6 \bmod 4)] \bmod 4 \\ &= (1 \times 2) \bmod 4 = 2 \bmod 4 = 2 \end{aligned}$$

Other properties of the modulo operator

$(a \bullet b) =_n (a \bmod n) \bullet (b \bmod n) \qquad \text{for } \bullet \in \{+, -, \times\}$

i.e., $(a \bullet b) \bmod n = [(a \bmod n) \bullet (b \bmod n)] \bmod n$

Example:

$$\begin{aligned} 2 &= (5 \times 6) \bmod 4 \\ &= [(5 \bmod 4) \times (6 \bmod 4)] \bmod 4 \\ &= (1 \times 2) \bmod 4 = 2 \bmod 4 = 2 \end{aligned}$$

If $a \times b =_n a \times c$ and a relatively prime to n, then $b =_n c$.

Other properties of the modulo operator

$(a \bullet b) =_n (a \bmod n) \bullet (b \bmod n) \qquad \text{for } \bullet \in \{+, -, \times\}$

i.e., $(a \bullet b) \bmod n = [(a \bmod n) \bullet (b \bmod n)] \bmod n$

Example:

$$\begin{aligned} 2 &= (5 \times 6) \bmod 4 \\ &= [(5 \bmod 4) \times (6 \bmod 4)] \bmod 4 \\ &= (1 \times 2) \bmod 4 = 2 \bmod 4 = 2 \end{aligned}$$

If $a \times b \ =_n \ a \times c$ and a relatively prime to n, then $b =_n c$.

Example:

- $8 \times 4 =_3 8 \times 1$.

Other properties of the modulo operator

$(a \bullet b) =_n (a \bmod n) \bullet (b \bmod n)$ for $\bullet \in \{+, -, \times\}$

i.e., $(a \bullet b) \bmod n = [(a \bmod n) \bullet (b \bmod n)] \bmod n$

Example:

$$\begin{aligned} 2 &= (5 \times 6) \bmod 4 \\ &= [(5 \bmod 4) \times (6 \bmod 4)] \bmod 4 \\ &= (1 \times 2) \bmod 4 = 2 \bmod 4 = 2 \end{aligned}$$

If $a \times b =_n a \times c$ and a relatively prime to n, then $b =_n c$.

Example:

- $8 \times 4 =_3 8 \times 1$.
- 8 is relatively prime to 3.

Other properties of the modulo operator

$(a \bullet b) =_n (a \bmod n) \bullet (b \bmod n) \qquad \text{for } \bullet \in \{+, -, \times\}$

i.e., $(a \bullet b) \bmod n = [(a \bmod n) \bullet (b \bmod n)] \bmod n$

Example:

$$\begin{aligned} 2 &= (5 \times 6) \bmod 4 \\ &= [(5 \bmod 4) \times (6 \bmod 4)] \bmod 4 \\ &= (1 \times 2) \bmod 4 = 2 \bmod 4 = 2 \end{aligned}$$

If $a \times b =_n a \times c$ and a relatively prime to n, then $b =_n c$.

Example:

- $8 \times 4 =_3 8 \times 1$.
- 8 is relatively prime to 3.
- So: $4 =_3 1$.

If $a_1 =_n b_1$ and $a_2 =_n b_2$, then

$$(a_1 + a_2) =_n (b_1 + b_2) \quad \text{and} \quad (a_1 \times a_2) =_n (b_1 \times b_2)$$

- This can also be expressed as

$$[(a_1 \bmod n) + (a_2 \bmod n)] \bmod n = (a_1 + a_2) \bmod n$$

  and

$$[(a_1 \bmod n) \times (a_2 \bmod n)] \bmod n = (a_1 \times a_2) \bmod n$$

- Example: Let $r_a = a \bmod n$ and $r_b = b \bmod n$.
  Then, there are integers j and k such that

$$a = r_a + jn \text{ and } b = r_b + kn$$

  and we can proceed as follows:

$$(a + b) \bmod n \quad = \quad (r_a + jn + r_b + kn) \bmod n$$

If $a_1 =_n b_1$ and $a_2 =_n b_2$, then

$$(a_1 + a_2) =_n (b_1 + b_2) \quad \text{and} \quad (a_1 \times a_2) =_n (b_1 \times b_2)$$

- This can also be expressed as

$$[(a_1 \bmod n) + (a_2 \bmod n)] \bmod n = (a_1 + a_2) \bmod n$$

  and

$$[(a_1 \bmod n) \times (a_2 \bmod n)] \bmod n = (a_1 \times a_2) \bmod n$$

- Example: Let $r_a = a \bmod n$ and $r_b = b \bmod n$.
  Then, there are integers j and k such that

$$a = r_a + jn \text{ and } b = r_b + kn$$

  and we can proceed as follows:

$$\begin{aligned} (a+b) \bmod n &= (r_a + jn + r_b + kn) \bmod n \\ &= (r_a + r_b + (j+k)n) \bmod n \end{aligned}$$

If $a_1 =_n b_1$ and $a_2 =_n b_2$, then

$$(a_1 + a_2) =_n (b_1 + b_2) \quad \text{and} \quad (a_1 \times a_2) =_n (b_1 \times b_2)$$

- This can also be expressed as

$$[(a_1 \bmod n) + (a_2 \bmod n)] \bmod n = (a_1 + a_2) \bmod n$$

  and

$$[(a_1 \bmod n) \times (a_2 \bmod n)] \bmod n = (a_1 \times a_2) \bmod n$$

- Example: Let $r_a = a \bmod n$ and $r_b = b \bmod n$.
  Then, there are integers j and k such that

$$a = r_a + jn \text{ and } b = r_b + kn$$

  and we can proceed as follows:

$$\begin{array}{rcl} (a+b) \bmod n & = & (r_a + jn + r_b + kn) \bmod n \\ & = & (r_a + r_b + (j+k)n) \bmod n \\ & = & (r_a + r_b) \bmod n \end{array}$$

If $a_1 =_n b_1$ and $a_2 =_n b_2$, then

$$(a_1 + a_2) =_n (b_1 + b_2) \quad \text{and} \quad (a_1 \times a_2) =_n (b_1 \times b_2)$$

- This can also be expressed as

$$[(a_1 \bmod n) + (a_2 \bmod n)] \bmod n = (a_1 + a_2) \bmod n$$

  and

$$[(a_1 \bmod n) \times (a_2 \bmod n)] \bmod n = (a_1 \times a_2) \bmod n$$

- Example: Let $r_a = a \bmod n$ and $r_b = b \bmod n$.
  Then, there are integers j and k such that

$$a = r_a + jn \text{ and } b = r_b + kn$$

  and we can proceed as follows:

$$\begin{aligned}
(a+b) \bmod n &= (r_a + jn + r_b + kn) \bmod n \\
&= (r_a + r_b + (j+k)n) \bmod n \\
&= (r_a + r_b) \bmod n \\
&= [(a \bmod n) + (b \bmod n)] \bmod n
\end{aligned}$$

- $a = q \times n + r$ with $q = \lfloor a/n \rfloor$ and $0 \leq r < n$ and $r = a \bmod b$
- For any integer a, we can rewrite this as follows:
  $a = \lfloor a/n \rfloor \times n + (a \bmod n)$

Then, for example:

- $11 \bmod 7 = 4$
- $-11 \bmod 7 = -4$ ($= 3$ when reasoning modulo 7)
- $73 =_{23} 4$
- $21 =_{10} -9$
- $147 =_{220} -73$

- If $a =_n 0$ then $n \mid a$
- $a =_n b$ if $n \mid (a - b)$

- To demonstrate the last point, if $n \mid (a - b)$, then $(a - b) = k \times n$ for some k.

  So we can write $a = b + (k \times n)$.

  Therefore, (a mod n) = (remainder when $b + (k \times n)$ is divided by n) = (remainder when b is divided by n) = (b mod n).
- Then, for example:
  - $23 =_5 8$ because $23 - 8 = 15 = 5 \times 3$
  - $-11 =_8 5$ because $-11 - 5 = -16 = 8 \times (-2)$
  - $81 =_{27} 0$ because $81 - 0 = 81 = 27 \times 3$

Modular arithmetics: two theorems

Theorem

Suppose that $a, b \in \mathbb{Z}$ are relatively prime. There is a $c \in \mathbb{Z}$ satisfying $(b \times c) \bmod a = 1$, i.e., we can compute $b^{-1} \bmod a$.

Proof: From Extended Euclidean Algorithm, there exist $x, y \in \mathbb{Z}$ where

$$1 = (a \times x) + (b \times y)$$

Since $a \mid (a \times x)$, we have $(b \times y) \bmod a = 1$.
Assertion follows with $c = y$.

Modular arithmetics: two theorems

Theorem

Suppose that $a, b \in \mathbb{Z}$ are relatively prime. There is a $c \in \mathbb{Z}$ satisfying $(b \times c) \bmod a = 1$, i.e., we can compute $b^{-1} \bmod a$.

Proof: From Extended Euclidean Algorithm, there exist $x, y \in \mathbb{Z}$ where

$$1 = (a \times x) + (b \times y)$$

Since $a \mid (a \times x)$, we have $(b \times y) \bmod a = 1$.
Assertion follows with $c = y$.

Fermat's little theorem

For a and n relatively prime and n prime

$$a^{n-1} =_n 1$$

Example: $4^6 \bmod 7 = (16 \times 16 \times 16) \bmod 7 = (2 \times 2 \times 2) \bmod 7 = 1$.

Table of contents I

Euler Totient Function

- When doing arithmetic modulo n.
- Complete set of **residues** is $0, \ldots, n-1$.
- **Reduced set of residues** consists of those numbers (*residues*) that are relatively prime to n.

  For instance, for $n = 10$:
  - complete set of residues is $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
  - reduced set of residues is $\{1, 3, 7, 9\}$.
- Number of elements in reduced set of residues is called the **Euler Totient Function** $\phi(n)$.
  - In other words, $\phi(n)$ **is the number of positive integers less than** n **which are relatively prime to** n, i.e.,
    $\phi(n)$ is the number of $a \in \{1, 2, \ldots, n-1\}$ with $\gcd(a, n) = 1$.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6	8

Euler's Totient Function and Euler's Theorem

Properties:

- $\phi(1) = 1$.
- $\phi(p) = p - 1$ if p is prime.
- $\phi(p \times q) = \phi(p) \times \phi(q) = (p - 1) \times (q - 1)$ if p and q are prime and $p \neq q$.

So that Fermat's little theorem (for a and n relatively prime and n prime) can be rewritten to

Euler's Theorem

$a^{\phi(n)} =_n 1$ for all a, n such that $\gcd(a, n) = 1$.

Examples:

- If $a = 3$ and $n = 10$, then $\phi(10) = 4$ and $3^4 = 81 =_{10} 1$
- If $a = 2$ and $n = 11$, then $\phi(11) = 10$ and $2^{10} = 1024 =_{11} 1$

Table of contents I

Rivest, Shamir, Adleman: RSA Algorithm

- Named after inventors: Rivest, Shamir, Adleman, 1978.
- Published after 1976 challenge by Diffie and Hellman.
- RSA algorithm is a block cipher in which the plaintext and ciphertext are integers between 0 and $n - 1$ for some n.
  - A typical size for n is 1024 bits, or 309 decimal digits.
  - That is, n is less than 2^{1024}.
- Security comes from difficulty of factoring large numbers. Keys are functions of a pairs of large, $\geq$ 100 digits, prime numbers.
- Most popular public-key algorithm. Used in many applications, e.g., PGP, PEM, SSL, ...

RSA algorithm

Ingredients:

	p, q, two prime numbers	private, chosen
	$n = p \times q$ (or pq for short)	public, calculated
	e, with $\gcd(\phi(n), e) = 1$; $1 < e < \phi(n)$	public, chosen
	$d = e^{-1} \bmod \phi(n)$	private, calculated

- Generation of a public/private key pair:
  1. Generate two (large) distinct primes p and q.
  2. Compute $n = p \times q$ and $\phi(n) = (p-1) \times (q-1)$.
  3. Select an e, with $1 < e < \phi(n)$, relatively prime to $\phi(n)$.
  4. Compute $d = e^{-1} \bmod \phi(n)$.
  5. Publish (e, n), keep (d, n) private, discard p and q.
- Encryption with key (e, n)
  1. Break message M into blocks $M_1 M_2 \cdots$ with $M_i < n$
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n):
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate two (large) distinct primes p and q.
  2. Compute $n = p \times q$ and $\phi(n) = (p-1) \times (q-1)$.
  3. Select an e, $1 < e < \phi(n)$, relatively prime to $\phi(n)$.
  4. Compute $d = e^{-1} \bmod \phi(n)$.
  5. Publish (e, n), keep (d, n) private, discard p and q.
- Encryption with key (e, n) :
  1. Break message M into blocks $M_1 M_2 \cdots$ with $M_i < n$.
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n) :
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q$ and $\phi(n) = (p - 1) \times (q - 1)$.
  3. Select an e, $1 < e < \phi(n)$, relatively prime to $\phi(n)$.
  4. Compute $d = e^{-1} \bmod \phi(n)$.
  5. Publish (e, n), keep (d, n) private, discard p and q.
- Encryption with key (e, n) :
  1. Break message M into blocks $M_1 M_2 \cdots$ with $M_i < n$.
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n) :
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Select an e, $1 < e < \phi(n)$, relatively prime to $\phi(n)$.
  4. Compute $d = e^{-1} \bmod \phi(n)$.
  5. Publish (e, n), keep (d, n) private, discard p and q.
- Encryption with key (e, n) :
  1. Break message M into blocks $M_1 M_2 \cdots$ with $M_i < n$.
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n) :
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Choose $e = 79$ (randomly in the interval [1..3220])
  4. Compute $d = e^{-1} \bmod \phi(n)$.
  5. Publish (e, n), keep (d, n) private, discard p and q.
- Encryption with key (e, n) :
  1. Break message M into blocks $M_1 M_2 \cdots$ with $M_i < n$.
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n) :
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Choose $e = 79$ (randomly in the interval [1..3220])
  4. Compute $d = 79^{-1} \bmod 3220 = 1019$.
  5. Publish (e, n), keep (d, n) private, discard p and q.
- Encryption with key (e, n) :
  1. Break message M into blocks $M_1 M_2 \cdots$ with $M_i < n$.
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n) :
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Choose $e = 79$ (randomly in the interval $[1..3220]$)
  4. Compute $d = 79^{-1} \bmod 3220 = 1019$.
  5. Public key $(e, n) = (79, 3337)$, private key $(d, n) = (1019, 3337)$
- Encryption with key (e, n) :
  1. Break message M into blocks $M_1 M_2 \cdots$ with $M_i < n$.
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n) :
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Choose $e = 79$ (randomly in the interval [1..3220])
  4. Compute $d = 79^{-1} \bmod 3220 = 1019$.
  5. Public key $(e, n) = (79, 3337)$, private key $(d, n) = (1019, 3337)$
- Encryption with key $(e, n) = (79, 3337)$:
  1. Break message M into blocks $M_1 M_2 \cdots$ with $M_i < n$.
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n) :
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Choose $e = 79$ (randomly in the interval $[1..3220]$)
  4. Compute $d = 79^{-1} \bmod 3220 = 1019$.
  5. Public key $(e, n) = (79, 3337)$, private key $(d, n) = (1019, 3337)$
- Encryption with key $(e, n) = (79, 3337)$:
  1. Break message M into blocks, e.g., 688 232 687 966 668 ...
  2. Compute $C_i = M_i^e \bmod n$.
- Decryption with key (d, n) :
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Choose $e = 79$ (randomly in the interval [1..3220])
  4. Compute $d = 79^{-1} \bmod 3220 = 1019$.
  5. Public key $(e, n) = (79, 3337)$, private key $(d, n) = (1019, 3337)$
- Encryption with key $(e, n) = (79, 3337)$:
  1. Break message M into blocks, e.g., 688 232 687 966 668 ...
  2. Compute $C_1 = 688^{79} \bmod 3337 = 1570$, $C_2 = \ldots$
- Decryption with key (d, n) :
  1. Compute $M_i = C_i^d \bmod n$.

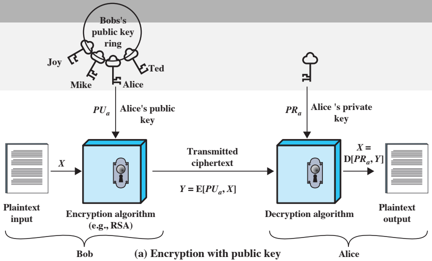
RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Choose $e = 79$ (randomly in the interval [1..3220])
  4. Compute $d = 79^{-1} \bmod 3220 = 1019$.
  5. Public key $(e, n) = (79, 3337)$, private key $(d, n) = (1019, 3337)$
- Encryption with key $(e, n) = (79, 3337)$:
  1. Break message M into blocks, e.g., 688 232 687 966 668 ...
  2. Compute $C_1 = 688^{79} \bmod 3337 = 1570$, $C_2 = \ldots$
- Decryption with key $(d, n) = (1019, 3337)$:
  1. Compute $M_i = C_i^d \bmod n$.

RSA algorithm: example

- Generation of a public/private key pair:
  1. Generate $p = 47$, $q = 71$.
  2. Compute $n = p \times q = 3337$ and $\phi(n) = (p-1) \times (q-1) = 46 \times 70 = 3220$
  3. Choose $e = 79$ (randomly in the interval [1..3220])
  4. Compute $d = 79^{-1} \bmod 3220 = 1019$.
  5. Public key $(e, n) = (79, 3337)$, private key $(d, n) = (1019, 3337)$
- Encryption with key $(e, n) = (79, 3337)$:
  1. Break message M into blocks, e.g., 688 232 687 966 668 ...
  2. Compute $C_1 = 688^{79} \bmod 3337 = 1570$, $C_2 = \ldots$
- Decryption with key $(d, n) = (1019, 3337)$:
  1. Compute $M_1 = 1570^{1019} \bmod 3337 = 688$, $M_2 = \ldots.$

RSA: another example



Alice generates a public/private key pair.
Bob encrypts using Alice's public key.
Alice decrypts using her private key.

- Keys can be generated as follows:
  - Select two prime numbers, $p = 17$ and $q = 11$.
  - Calculate $n = p \times q = 17 \times 11 = 187$.
  - Calculate $\phi(n) = (p - 1) \times (q - 1) = 16 \times 10 = 160$.
  - Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
  - Determine d (e.g., using Extended Euclid's algorithm) such that $d \times e = 1 \bmod 160$ and $d < 160$.
    The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$.

Note: the d here is the private key, not to be confused with the d that is the greatest common **d**ivisor in the Extended Euclid's algorithm.

Resulting keys are public key $PU_a = (e, n) = (7, 187)$ and private key $PR_a = (d, n) = (23, 187)$.

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.
- d can be computed using the Extended Euclid algorithm

$$D = \mathrm{gcd}(A, B) = A \times x + B \times y$$

as follows:

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.
- d can be computed using the Extended Euclid algorithm

$$D = \gcd(A, B) = A \times x + B \times y$$

  as follows:
  - Since d is such that $e \times d =_{\phi(n)} 1$, we can compute

$$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d$$

  - It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.
- d can be computed using the Extended Euclid algorithm

$$D = \gcd(A, B) = A \times x + B \times y$$

  as follows:
  - Since d is such that $e \times d =_{\phi(n)} 1$, we can compute

$$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d$$

  - It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.
- In this case:

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.
- d can be computed using the Extended Euclid algorithm

$$D = \mathrm{gcd}(A, B) = A \times x + B \times y$$

  as follows:
  - Since d is such that $e \times d =_{\phi(n)} 1$, we can compute

$$1 = \mathrm{gcd}(\phi(n), e) = \phi(n) \times x + e \times d$$

  - It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.
- In this case:

$$1 = \mathrm{gcd}(160, 7) = 160 \times x + 7 \times y$$

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.
- d can be computed using the Extended Euclid algorithm

$$D = \gcd(A, B) = A \times x + B \times y$$

  as follows:
  - Since d is such that $e \times d =_{\phi(n)} 1$, we can compute

$$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d$$

  - It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.
- In this case:

$$1 = \gcd(160, 7) = 160 \times x + 7 \times y$$

A	B	$\lfloor A/B \rfloor$	D	x	y
160	7	22	1	-1	23
7	6	1	1	1	-1
6	1	6	1	0	1
1	0	–	1	1	0

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.
- d can be computed using the Extended Euclid algorithm

$$D = \gcd(A, B) = A \times x + B \times y$$

as follows:

  - Since d is such that $e \times d =_{\phi(n)} 1$, we can compute

$$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d$$

  - It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.
- In this case:

$$1 = \gcd(160, 7) = 160 \times x + 7 \times y$$

A	B	$\lfloor A/B \rfloor$	D	x	y
160	7	22	1	-1	23
7	6	1	1	1	-1
6	1	6	1	0	1
1	0	–	1	1	0

That is, $1 = \gcd(160, 7) = 160 \times (-1) + 7 \times 23$.

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.
- d can be computed using the Extended Euclid algorithm

$$D = \mathrm{gcd}(A, B) = A \times x + B \times y$$

as follows:

  - Since d is such that $e \times d =_{\phi(n)} 1$, we can compute

$$1 = \mathrm{gcd}(\phi(n), e) = \phi(n) \times x + e \times d$$

  - It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.

- In this case:

$$1 = \mathrm{gcd}(160, 7) = 160 \times x + 7 \times y$$

A	B	$\lfloor A/B \rfloor$	D	x	y
160	7	22	1	-1	23
7	6	1	1	1	-1
6	1	6	1	0	1
1	0	–	1	1	0

That is, $1 = \mathrm{gcd}(160, 7) = 160 \times (-1) + 7 \times 23$. Check: $7 \times 23 =_{160} 1$.

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$, $e = 7$.
- d can be computed using the Extended Euclid algorithm

$$D = \gcd(A, B) = A \times x + B \times y$$

  as follows:
  - Since d is such that $e \times d =_{\phi(n)} 1$, we can compute
  $$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d$$
  - It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.
- In this case:

$$1 = \gcd(160, 7) = 160 \times x + 7 \times y$$

A	B	$\lfloor A/B \rfloor$	D	x	y
160	7	22	1	-1	23
7	6	1	1	1	-1
6	1	6	1	0	1
1	0	–	1	1	0

  That is, $1 = \gcd(160, 7) = 160 \times (-1) + 7 \times 23$. Check: $7 \times 23 =_{160} 1$.
  So, we can pick $d = y = 23$.

RSA algorithm: another example (cont.)

- We have $n = 187$, $\phi(n) = 160$.
- Note that if we had picked $e = 23$, then $d = 7$.
  - Since d is such that $e \times d =_{\phi(n)} 1$, we can compute

$$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d$$

- In this case:

$$1 = \gcd(160, 23) = 160 \times x + 23 \times y$$

A	B	$\lfloor A/B \rfloor$	D	x	y
160	23	6	1	-1	7
23	22	1	1	1	-1
22	1	22	1	0	1
1	0	–	1	1	0

That is, $1 = \gcd(160, 23) = 160 \times (-1) + 23 \times 7$. Check: $23 \times 7 =_{160} 1$.

So, we can pick $d = y = 7$.

RSA algorithm: a remark on the computed d

It must be $1 < d < \phi(n)$, so when $y < 0$ we simply reason modulo $\phi(n)$.

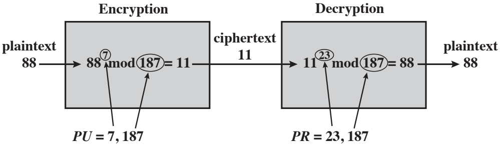
Consider, for example, $\phi(n) = 220$ and $e = 3$:

$$1 = \gcd(\phi(n), e) = \phi(n) \times x + e \times d = \gcd(220, 3) = 220 \times x + 3 \times y$$

A	B	$\lfloor A/B \rfloor$	D	x	y
220	3	73	1	1	-73
3	1	3	1	0	1
1	0	–	1	1	0

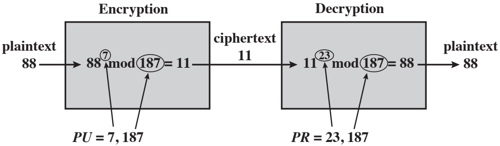
That is, $1 = \gcd(220, 3) = 220 \times 1 + 3 \times (-73) = 220 - 219$.
So, we can pick $d = 147$, i.e., $-73 \mod 220$.

RSA algorithm: another example (cont.)



- Let's continue the previous example.
- To encrypt a plaintext input $M = 88$, we need to calculate $C = M^e \bmod n = 88^7 \bmod 187 = 11$.
- We can do this by exploiting properties of modular arithmetic:
  - $88^7 \bmod 187 = ((88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)) \bmod 187$
  - $88^1 \bmod 187 = 88$
  - $88^2 \bmod 187 = 7744 \bmod 187 = 77$
  - $88^4 \bmod 187 = 59,969,536 \bmod 187 = 132$
  - $88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894,432 \bmod 187 = 11$

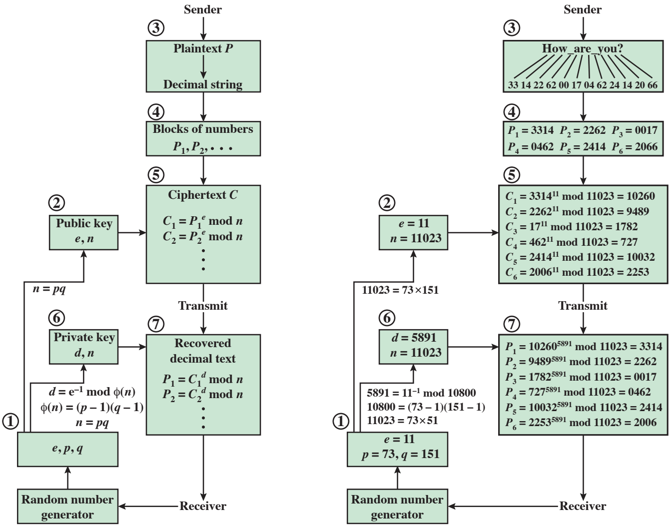
RSA algorithm: another example (cont.)



- For decryption, we calculate $M = C^d \bmod n = 11^{23} \bmod 187$:
  - $11^{23} \bmod 187 = ((11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)) \bmod 187$
  - $11^1 \bmod 187 = 11$
  - $11^2 \bmod 187 = 121$
  - $11^4 \bmod 187 = 14{,}641 \bmod 187 = 55$
  - $11^8 \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$
  - $11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79{,}720{,}245 \bmod 187 = 88$

Use of RSA to process multiple blocks of data: example

- In this simple example, plaintext is an alphanumeric string.
- Each plaintext symbol is assigned a unique code of 2 decimal digits (e.g., a = 00, A = 26).
- A plaintext block consists of 4 decimal digits, or 2 alphanumeric characters.
- Circled numbers indicate order in which operations are performed.



(a) General approach (b) Example

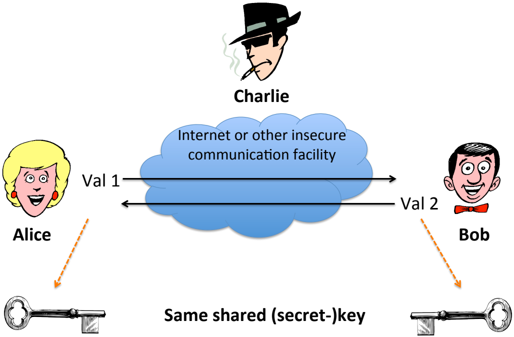
RSA Security

- Computation of secret d given (e, n).
  - As difficult as factorization. If we can factor $n = p \times q$ then we can compute $\phi(n) = (p-1) \times (q-1)$ and hence $d = e^{-1} \bmod \phi(n)$.
  - No known polynomial time algorithm.
    But given progress in factoring, n should have at least 1024 bits.
- Computation of M_i, given C_i, and (e, n).
  - Unclear (= no proof) whether it is necessary to compute d, i.e., to factorize n.

Hence: Progress in number theory could make RSA insecure.

Table of contents I

Diffie-Hellman key exchange: in a nutshell



- A simple public-key algorithm that enables two users to establish a secret key using a public-key scheme based on discrete logarithms.
- The protocol is secure only if the authenticity of the two participants can be established.

Background on discrete logarithms

- A **primitive root** s of a prime number p is a number whose powers generate $1, \ldots, p-1$.
  So s^0 mod p, s^1 mod p, s^2 mod p, ..., s^{p-1} mod p are distinct, i.e., a permutation of 1 through $p-1$. Hence:

  $$\forall b \in \mathbb{Z}. \exists i \in \{0, \ldots, p-1\}. \; b = s^i \bmod p$$

  In words: for any integer b and a primitive root s of prime number p, we can find a unique exponent i such that

  $$b = s^i \bmod p$$

  where $0 \leq i \leq (p-1)$.
  i is called the **discrete logarithm** of b for base s, mod p.

- Computing discrete logarithms appears infeasible today.

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q.

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.
2. A and B generate random numbers X_A and X_B (respectively) both less than q.
   X_A and X_B are the **private keys**.

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.
2. A and B generate random numbers X_A and X_B (respectively) both less than q.
   X_A and X_B are the **private keys**.
3. A computes $Y_A = \alpha^{X_A} \bmod q$, B computes $Y_B = \alpha^{X_B} \bmod q$.
   Y_A and Y_B are the **public keys** (a.k.a. "Diffie-Hellman half keys").

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.
2. A and B generate random numbers X_A and X_B (respectively) both less than q.
   X_A and X_B are the **private keys**.
3. A computes $Y_A = \alpha^{X_A} \bmod q$, B computes $Y_B = \alpha^{X_B} \bmod q$.
   Y_A and Y_B are the **public keys** (a.k.a. "Diffie-Hellman half keys").
4. A and B exchange Y_A and Y_B.

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.
2. A and B generate random numbers X_A and X_B (respectively) both less than q.
   X_A and X_B are the **private keys**.
3. A computes $Y_A = \alpha^{X_A} \bmod q$, B computes $Y_B = \alpha^{X_B} \bmod q$.
   Y_A and Y_B are the **public keys** (a.k.a. "Diffie-Hellman half keys").
4. A and B exchange Y_A and Y_B.
5. A computes $K_A = Y_B^{X_A} \bmod q$, B computes $K_B = Y_A^{X_B} \bmod q$.

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.
2. A and B generate random numbers X_A and X_B (respectively) both less than q.
   X_A and X_B are the **private keys**.
3. A computes $Y_A = \alpha^{X_A} \bmod q$, B computes $Y_B = \alpha^{X_B} \bmod q$.
   Y_A and Y_B are the **public keys** (a.k.a. "Diffie-Hellman half keys").
4. A and B exchange Y_A and Y_B.
5. A computes $K_A = Y_B^{X_A} \bmod q$, B computes $K_B = Y_A^{X_B} \bmod q$.
   Keys are equal, i.e., $K_A = K_B$:

$$K_A \quad = \quad Y_B^{X_A} \bmod q$$

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.
2. A and B generate random numbers X_A and X_B (respectively) both less than q.
   X_A and X_B are the **private keys**.
3. A computes $Y_A = \alpha^{X_A} \bmod q$, B computes $Y_B = \alpha^{X_B} \bmod q$.
   Y_A and Y_B are the **public keys** (a.k.a. "Diffie-Hellman half keys").
4. A and B exchange Y_A and Y_B.
5. A computes $K_A = Y_B^{X_A} \bmod q$, B computes $K_B = Y_A^{X_B} \bmod q$.
   Keys are equal, i.e., $K_A = K_B$:

$$\begin{aligned} K_A &= Y_B^{X_A} \bmod q \\ &= (\alpha^{X_B} \bmod q)^{X_A} \bmod q = (\alpha^{X_B})^{X_A} \bmod q \end{aligned}$$

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.
2. A and B generate random numbers X_A and X_B (respectively) both less than q.
   X_A and X_B are the **private keys**.
3. A computes $Y_A = \alpha^{X_A} \bmod q$, B computes $Y_B = \alpha^{X_B} \bmod q$.
   Y_A and Y_B are the **public keys** (a.k.a. "Diffie-Hellman half keys").
4. A and B exchange Y_A and Y_B.
5. A computes $K_A = Y_B^{X_A} \bmod q$, B computes $K_B = Y_A^{X_B} \bmod q$.
   Keys are equal, i.e., $K_A = K_B$:

$$\begin{aligned} K_A &= Y_B^{X_A} \bmod q \\ &= (\alpha^{X_B} \bmod q)^{X_A} \bmod q = (\alpha^{X_B})^{X_A} \bmod q \\ &= \alpha^{X_A X_B} \bmod q = (\alpha^{X_A})^{X_B} \bmod q \end{aligned}$$

Diffie-Hellman key exchange

1. Principals share a prime number q and an integer α that is a primitive root of q. Both q and α may be public, or A could send them in the first message.
2. A and B generate random numbers X_A and X_B (respectively) both less than q.
   X_A and X_B are the **private keys**.
3. A computes $Y_A = \alpha^{X_A} \bmod q$, B computes $Y_B = \alpha^{X_B} \bmod q$.
   Y_A and Y_B are the **public keys** (a.k.a. "Diffie-Hellman half keys").
4. A and B exchange Y_A and Y_B.
5. A computes $K_A = Y_B^{X_A} \bmod q$, B computes $K_B = Y_A^{X_B} \bmod q$.
   Keys are equal, i.e., $K_A = K_B$:

$$
\begin{aligned}
K_A &= Y_B^{X_A} \bmod q \\
&= (\alpha^{X_B} \bmod q)^{X_A} \bmod q = (\alpha^{X_B})^{X_A} \bmod q \\
&= \alpha^{X_A X_B} \bmod q = (\alpha^{X_A})^{X_B} \bmod q \\
&= (\alpha^{X_A} \bmod q)^{X_B} \bmod q = Y_A^{X_B} \bmod q \;=\; K_B
\end{aligned}
$$

Diffie-Hellman key exchange: ingredients

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

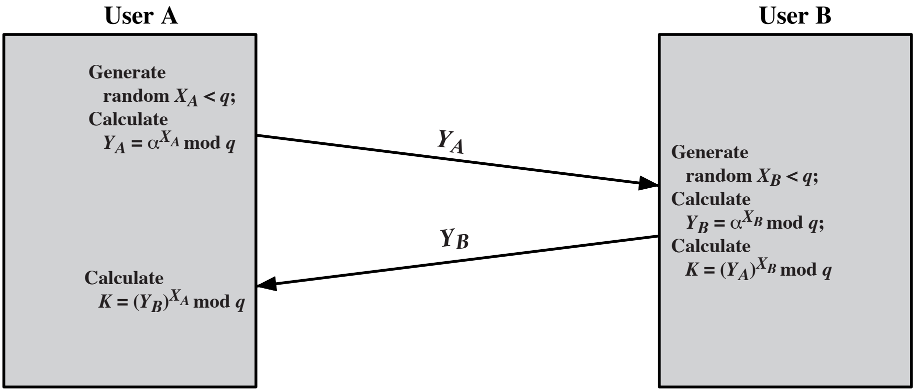
Calculation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Calculation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Diffie-Hellman key exchange: figure



Diffie-Hellman key exchange: strengths

- The shared secret key is never transmitted (not even in encrypted form)... it is created “out of nothing”!
  - $Y_A = \alpha^{X_A} \bmod q$ and $Y_B = \alpha^{X_B} \bmod q$ are the public keys.
  - X_A and X_B are the private keys.
  - Because X_A and X_B are private, an adversary C only has the following ingredients to work with: q, α, Y_A and Y_B.
  - Thus, C must take a discrete logarithm to determine the key. For example, to determine the private key of user B, C must compute
  $$X_B = \mathrm{dlog}_{\alpha,q}(Y_B)$$
- Security of Diffie-Hellman key exchange lies in the fact that
  - it is relatively easy to calculate exponentials modulo a prime, but
  - it is very difficult to calculate discrete logarithms (e.g., it is considered infeasible for large primes).

  Security depends on the difficulty of computing discrete logarithms.

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).
- A and B select private keys $X_A = 97$ and $X_B = 233$.

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).
- A and B select private keys $X_A = 97$ and $X_B = 233$.
- Each computes its public key:

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).
- A and B select private keys $X_A = 97$ and $X_B = 233$.
- Each computes its public key:
  - A computes $Y_A = \alpha^{X_A} \bmod q = 3^{97} \bmod 353 = 40$.

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).
- A and B select private keys $X_A = 97$ and $X_B = 233$.
- Each computes its public key:
  - A computes $Y_A = \alpha^{X_A} \bmod q = 3^{97} \bmod 353 = 40$.
  - B computes $Y_B = \alpha^{X_B} \bmod q = 3^{233} \bmod 353 = 248$.

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).
- A and B select private keys $X_A = 97$ and $X_B = 233$.
- Each computes its public key:
  - A computes $Y_A = \alpha^{X_A} \bmod q = 3^{97} \bmod 353 = 40$.
  - B computes $Y_B = \alpha^{X_B} \bmod q = 3^{233} \bmod 353 = 248$.
- After they exchange public keys, each can compute the common secret key K:

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).
- A and B select private keys $X_A = 97$ and $X_B = 233$.
- Each computes its public key:
  - A computes $Y_A = \alpha^{X_A} \bmod q = 3^{97} \bmod 353 = 40$.
  - B computes $Y_B = \alpha^{X_B} \bmod q = 3^{233} \bmod 353 = 248$.
- After they exchange public keys, each can compute the common secret key K:
  - A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).
- A and B select private keys $X_A = 97$ and $X_B = 233$.
- Each computes its public key:
  - A computes $Y_A = \alpha^{X_A} \bmod q = 3^{97} \bmod 353 = 40$.
  - B computes $Y_B = \alpha^{X_B} \bmod q = 3^{233} \bmod 353 = 248$.
- After they exchange public keys, each can compute the common secret key K:
  - A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.
  - B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$.

Diffie-Hellman: example (calculating the secret key)

- A and B choose prime number $q = 353$ and $\alpha = 3$ (which is one of the primitive roots of 353).
- A and B select private keys $X_A = 97$ and $X_B = 233$.
- Each computes its public key:
  - A computes $Y_A = \alpha^{X_A} \bmod q = 3^{97} \bmod 353 = 40$.
  - B computes $Y_B = \alpha^{X_B} \bmod q = 3^{233} \bmod 353 = 248$.
- After they exchange public keys, each can compute the common secret key K:
  - A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.
  - B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$.
- Now they case use the symmetric key K to encrypt the messages they want to exchange.

Diffie-Hellman: example (attacking the key)

- Attacker C knows: $q = 353$, $\alpha = 3$, $Y_A = 40$ and $Y_B = 248$.
  - In this simple example, it would be possible by brute force to determine the secret key $K = 160$.
  - In particular, C can determine K by discovering a solution to
    - the equation $3^a \bmod 353 = 40$ or
    - the equation $3^b \bmod 353 = 248$.
  - Brute-force approach: calculate powers of 3 mod 353, stopping when the result equals either 40 or 248.
    - Desired answer is reached with the exponent value of 97, which provides $3^{97} \bmod 353 = 40$.
- With larger numbers, the problem becomes impractical.

Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).

Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).
1. A generates X_A and transmits $Y_A = \alpha^{X_A} \bmod q$ to B.

Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).
1. A generates X_A and transmits $Y_A = \alpha^{X_A} \bmod q$ to B.
2. C intercepts Y_A and transmits $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ to B. C also calculates $K_2 = (Y_A)^{X_{C_2}} \bmod q = (\alpha^{X_A} \bmod q)^{X_{C_2}} \bmod q = \alpha^{X_A X_{C_2}} \bmod q$.

Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).
1. A generates X_A and transmits $Y_A = \alpha^{X_A} \bmod q$ to B.
2. C intercepts Y_A and transmits $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ to B. C also calculates $K_2 = (Y_A)^{X_{C_2}} \bmod q = (\alpha^{X_A} \bmod q)^{X_{C_2}} \bmod q = \alpha^{X_A X_{C_2}} \bmod q$.
3. B receives Y_{C_1}, generates X_B and calculates $K_B = (Y_{C_1})^{X_B} \bmod q = \alpha^{X_{C_1} X_B} \bmod q$.

Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).
1. A generates X_A and transmits $Y_A = \alpha^{X_A} \bmod q$ to B.
2. C intercepts Y_A and transmits $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ to B. C also calculates $K_2 = (Y_A)^{X_{C_2}} \bmod q = (\alpha^{X_A} \bmod q)^{X_{C_2}} \bmod q = \alpha^{X_A X_{C_2}} \bmod q$.
3. B receives Y_{C_1}, generates X_B and calculates $K_B = (Y_{C_1})^{X_B} \bmod q = \alpha^{X_{C_1} X_B} \bmod q$.
4. B transmits $Y_B = \alpha^{X_B} \bmod q$ to A.

Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).
1. A generates X_A and transmits $Y_A = \alpha^{X_A} \bmod q$ to B.
2. C intercepts Y_A and transmits $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ to B. C also calculates $K_2 = (Y_A)^{X_{C_2}} \bmod q = (\alpha^{X_A} \bmod q)^{X_{C_2}} \bmod q = \alpha^{X_A X_{C_2}} \bmod q$.
3. B receives Y_{C_1}, generates X_B and calculates $K_B = (Y_{C_1})^{X_B} \bmod q = \alpha^{X_{C_1} X_B} \bmod q$.
4. B transmits $Y_B = \alpha^{X_B} \bmod q$ to A.
5. C intercepts Y_B and transmits $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ to A. C also calculates $K_1 = (Y_B)^{X_{C_1}} \bmod q = (\alpha^{X_B} \bmod q)^{X_{C_1}} \bmod q = \alpha^{X_B X_{C_1}} \bmod q = K_B$.

Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).
1. A generates X_A and transmits $Y_A = \alpha^{X_A} \bmod q$ to B.
2. C intercepts Y_A and transmits $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ to B. C also calculates $K_2 = (Y_A)^{X_{C_2}} \bmod q = (\alpha^{X_A} \bmod q)^{X_{C_2}} \bmod q = \alpha^{X_A X_{C_2}} \bmod q$.
3. B receives Y_{C_1}, generates X_B and calculates $K_B = (Y_{C_1})^{X_B} \bmod q = \alpha^{X_{C_1} X_B} \bmod q$.
4. B transmits $Y_B = \alpha^{X_B} \bmod q$ to A.
5. C intercepts Y_B and transmits $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ to A. C also calculates $K_1 = (Y_B)^{X_{C_1}} \bmod q = (\alpha^{X_B} \bmod q)^{X_{C_1}} \bmod q = \alpha^{X_B X_{C_1}} \bmod q = K_B$.
6. A receives Y_{C_2} and calculates $K_A = (Y_{C_2})^{X_A} \bmod q = \alpha^{X_{C_2} X_A} \bmod q = K_2$.

Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).
1. A generates X_A and transmits $Y_A = \alpha^{X_A} \bmod q$ to B.
2. C intercepts Y_A and transmits $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ to B. C also calculates $K_2 = (Y_A)^{X_{C_2}} \bmod q = (\alpha^{X_A} \bmod q)^{X_{C_2}} \bmod q = \alpha^{X_A X_{C_2}} \bmod q$.
3. B receives Y_{C_1}, generates X_B and calculates $K_B = (Y_{C_1})^{X_B} \bmod q = \alpha^{X_{C_1} X_B} \bmod q$.
4. B transmits $Y_B = \alpha^{X_B} \bmod q$ to A.
5. C intercepts Y_B and transmits $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ to A. C also calculates $K_1 = (Y_B)^{X_{C_1}} \bmod q = (\alpha^{X_B} \bmod q)^{X_{C_1}} \bmod q = \alpha^{X_B X_{C_1}} \bmod q = K_B$.
6. A receives Y_{C_2} and calculates $K_A = (Y_{C_2})^{X_A} \bmod q = \alpha^{X_{C_2} X_A} \bmod q = K_2$.

Now A and B think that they share a secret key,

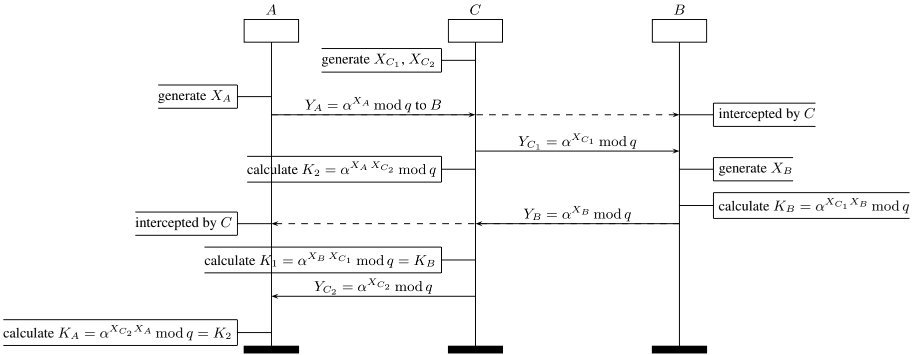
Diffie-Hellman key exchange: weakness

Keys are **unauthenticated** and thus Diffie-Hellman key exchange is vulnerable to the following **man-in-the-middle attack**:

0. Attacker C prepares for the attack by generating two random private keys X_{C_1} and X_{C_2} and then computing the corresponding public keys $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ and $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ (since α and q are public).
1. A generates X_A and transmits $Y_A = \alpha^{X_A} \bmod q$ to B.
2. C intercepts Y_A and transmits $Y_{C_1} = \alpha^{X_{C_1}} \bmod q$ to B. C also calculates $K_2 = (Y_A)^{X_{C_2}} \bmod q = (\alpha^{X_A} \bmod q)^{X_{C_2}} \bmod q = \alpha^{X_A X_{C_2}} \bmod q$.
3. B receives Y_{C_1}, generates X_B and calculates $K_B = (Y_{C_1})^{X_B} \bmod q = \alpha^{X_{C_1} X_B} \bmod q$.
4. B transmits $Y_B = \alpha^{X_B} \bmod q$ to A.
5. C intercepts Y_B and transmits $Y_{C_2} = \alpha^{X_{C_2}} \bmod q$ to A. C also calculates $K_1 = (Y_B)^{X_{C_1}} \bmod q = (\alpha^{X_B} \bmod q)^{X_{C_1}} \bmod q = \alpha^{X_B X_{C_1}} \bmod q = K_B$.
6. A receives Y_{C_2} and calculates $K_A = (Y_{C_2})^{X_A} \bmod q = \alpha^{X_{C_2} X_A} \bmod q = K_2$.

Now A and B think that they share a secret key, but instead A shares secret key $K_A = K_2$ with C and B shares secret key $K_B = K_1$ with C.

DH key exchange: man-in-the-middle attack



Now A and B think that they share a secret key, but instead A shares secret key $K_A = K_2$ with C and B shares secret key $K_B = K_1$ with C.

DH key exchange: man-in-the-middle attack

- All future communication between Bob and Alice is compromised in the following way.
  1. A sends an encrypted message M, i.e., $E(K_2, M)$.
  2. C intercepts the encrypted message and decrypts it to recover M.
  3. C sends to Bob either
     - $E(K_1, M)$, if C simply wants to eavesdrop on the communication without altering it, or
     - $E(K_1, M')$, where M' is any message, if C wants to modify the message going to B.

- The Diffie-Hellman key exchange is vulnerable to such an attack because it does not authenticate the participants.
  - This vulnerability can be overcome with the use of **digital signatures and public-key certificates** to achieve mutual authentication between A and B.
  - Typically: add an exchange of digitally signed identification (ID) tokens.

Group Diffie-Hellman (for three or more parties)

Given a Diffie-Hellman group (α, q), three honest parties Alice, Bob and Carol can generate together a secret key $K = \alpha^{X_A X_B X_C} \bmod q$ by:

1. Alice chooses a random large integer X_A and sends to Bob: $Y_A = \alpha^{X_A} \bmod q$
2. Bob chooses a random large integer X_B and sends to Carol $Y_B = \alpha^{X_B} \bmod q$
3. Carol chooses a random large integer X_C and sends to Alice: $Y_C = \alpha^{X_C} \bmod q$
4. Alice sends to Bob $Y'_C = Y_C^{X_A} \bmod q$
5. Bob sends to Carol $Y'_A = Y_A^{X_B} \bmod q$
6. Carol sends to Alice $Y'_B = Y_B^{X_C} \bmod q$
7. Alice computes: $K = {Y'_B}^{X_A} \bmod q$
8. Bob computes: $K = {Y'_C}^{X_B} \bmod q$
9. Carol computes: $K = {Y'_A}^{X_C} \bmod q$

Can be extended to more parties by adding more rounds of computations.

Table of contents I

Idea

What do weapons of mass destruction

Idea

What do weapons of mass destruction, a drink

Idea

What do weapons of mass destruction, a drink and Ali Baba's cave all have in common?

Idea

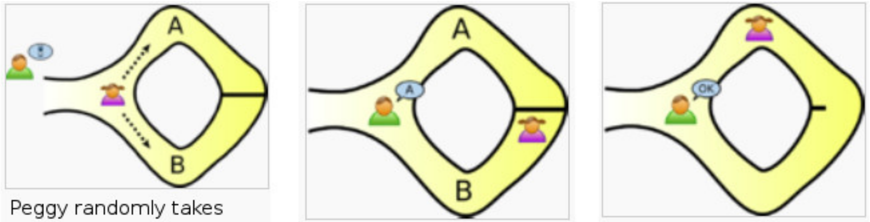
What do weapons of mass destruction, a drink and Ali Baba's cave all have in common?

Zero-knowledge proofs

In zero-knowledge proofs we can usually specify a statement that is being proved.

- Definitely, that statement is revealed to the verifier
- The verifier (or others) should not learn anything else
- Everybody can draw conclusions from everything they learned

Zero-knowledge proofs: Ali Baba’s cave

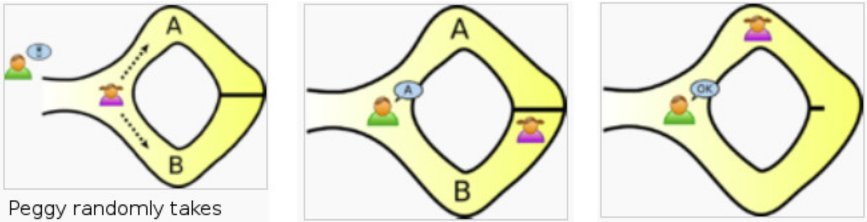


Peggy randomly takes either path A or B, while Victor waits outside

Victor chooses an exit path

Peggy reliably appears at the exit Victor names

- A cave has a door that opens only when a secret word is spoken.
- Peggy (the Prover) wants to convince Victor (the Verifier) that she knows the secret word, but without reveling it!
- If they walk to the door together, Peggy will be able to open it but then Victor will learn the secret word.
- So, they carry out a zero-knowledge protocol.

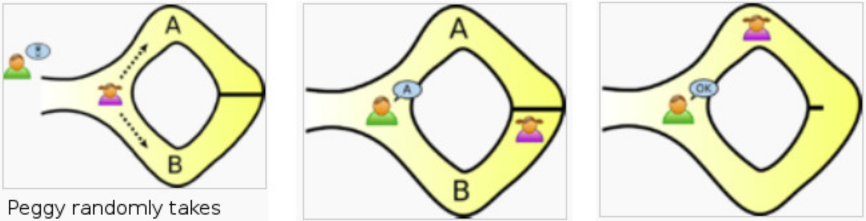


Peggy randomly takes either path A or B, while Victor waits outside

Victor chooses an exit path

Peggy reliably appears at the exit Victor names

1. Victor stands at the cave's entrance, while Peggy walks to the door.
2. Victor walks to the bifurcation of the cave's paths and shouts to Peggy either to
   - come out of the left path A or
   - come out of the right path B.
3. Peggy complies, using the secret word to open the door, if needed.
4. Peggy and Victor repeat the experiment (steps 1-3) *n* times.



Peggy randomly takes either path A or B, while Victor waits outside

Victor chooses an exit path

Peggy reliably appears at the exit Victor names

- Now assume that Peggy doesn't actually know the secret word.
- Then she can only come out the way she went in.
  - After 1 round, she has only 1 chance out of 2 of fooling Victor.
  - After n rounds, she has only 1 chance out of 2^n of fooling Victor.
- So, after a while, Victor will be convinced that Peggy knows the secret.
- In other words: Peggy wins if she passes the test all of the time.
  - The probability that Peggy wins is very low if she does not know the secret word: after n rounds, it is $(1/2)^n = \frac{1}{2^n}$.

Zero-knowledge proofs: the idea

- In a challenge-response protocol, the Prover proves that she knows a secret.
  - If a symmetric cryptosystem is used, then the Verifier also knows the secret.
  - If a public-key signature system is used, then Verifier does not know the secret.
- An example of a zero-knowledge protocol is the Fiat-Shamir Identification Protocol.

Example: Fiat-Shamir Identification Protocol

- Three principals:
  - **Prover Peggy**,
  - **Verifier Victor** and
  - **Trusted Third Party Trent**.
- Setup:
  - Trent chooses two large prime numbers p and q to calculate $n = p \times q$.
  - n is announced to the public, whereas p and q are kept secret.
  - Peggy chooses a **secret number** s between 1 and $n - 1$, and calculates $v = s^2 \bmod n$.
    Peggy keeps s as her **private key** and registers v as her **public key** with the third party.
- Victor knows $v = s^2 \bmod n$, but does not know s.
- Squaring modulo n is easy to compute but square root modulo n is probably not (we believe...).
- **Goal**: Peggy wants to convince Victor that she knows the secret s but Victor should not learn s!

- Verification of Peggy by Victor then proceeds in 4 steps:

- Verification of Peggy by Victor then proceeds in 4 steps:
  1. Peggy chooses a random number r between 0 and $n - 1$.
     r is called the **commitment**.

- Verification of Peggy by Victor then proceeds in 4 steps:
  1. Peggy chooses a random number r between 0 and $n - 1$.
     r is called the **commitment**.
     Peggy then calculates the **witness** $x = r^2 \bmod n$ and sends it to Victor.

- Verification of Peggy by Victor then proceeds in 4 steps:
  1. Peggy chooses a random number r between 0 and $n - 1$.
     r is called the **commitment**.
     Peggy then calculates the **witness** $x = r^2 \bmod n$ and sends it to Victor.
  2. Victor sends the **challenge** c to Peggy, where c is either 0 or 1.

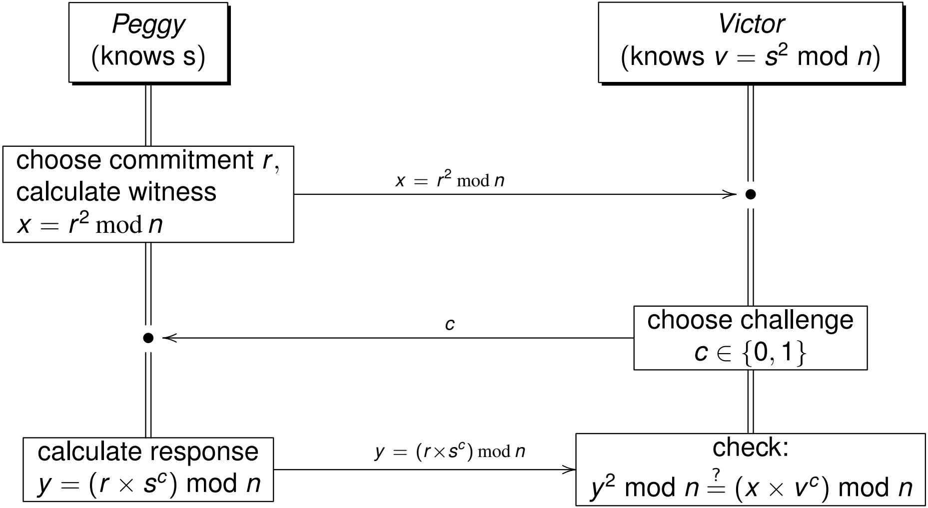
- Verification of Peggy by Victor then proceeds in 4 steps:
  1. Peggy chooses a random number r between 0 and $n-1$.
     r is called the **commitment**.
     Peggy then calculates the **witness** $x = r^2 \bmod n$ and sends it to Victor.
  2. Victor sends the **challenge** c to Peggy, where c is either 0 or 1.
  3. Peggy calculates the **response** $y = (r \times s^c) \bmod n$ and sends it to Victor to show that she knows her private key s modulo n.
     She claims to be Peggy.

- Verification of Peggy by Victor then proceeds in 4 steps:
  1. Peggy chooses a random number r between 0 and $n-1$.
     r is called the **commitment**.
     Peggy then calculates the **witness** $x = r^2 \bmod n$ and sends it to Victor.
  2. Victor sends the **challenge** c to Peggy, where c is either 0 or 1.
  3. Peggy calculates the **response** $y = (r \times s^c) \bmod n$ and sends it to Victor to show that she knows her private key s modulo n.
     She claims to be Peggy.
  4. Victor calculates $y^2 \bmod n$ and $(x \times v^c) \bmod n$.

- Verification of Peggy by Victor then proceeds in 4 steps:
  1. Peggy chooses a random number r between 0 and $n - 1$.
     r is called the **commitment**.
     Peggy then calculates the **witness** $x = r^2 \bmod n$ and sends it to Victor.
  2. Victor sends the **challenge** c to Peggy, where c is either 0 or 1.
  3. Peggy calculates the **response** $y = (r \times s^c) \bmod n$ and sends it to Victor to show that she knows her private key s modulo n.
     She claims to be Peggy.
  4. Victor calculates $y^2 \bmod n$ and $(x \times v^c) \bmod n$. If these values are congruent, then Peggy either knows the value of s (she is honest) or she has calculated the value of y in some either ways (dishonest) because in modulo n arithmetic we actually have that

$$y^2 =_n (r \times s^c)^2 =_n r^2 \times s^{2c} =_n r^2 \times (s^2)^c =_n x \times v^c$$

- The 4 steps constitute a **round**.
- The verification is repeated several times with the value of c equal to 0 or 1, chosen randomly.
- Peggy must pass the test in each round to be verified: if she fails one single round, the process is aborted.



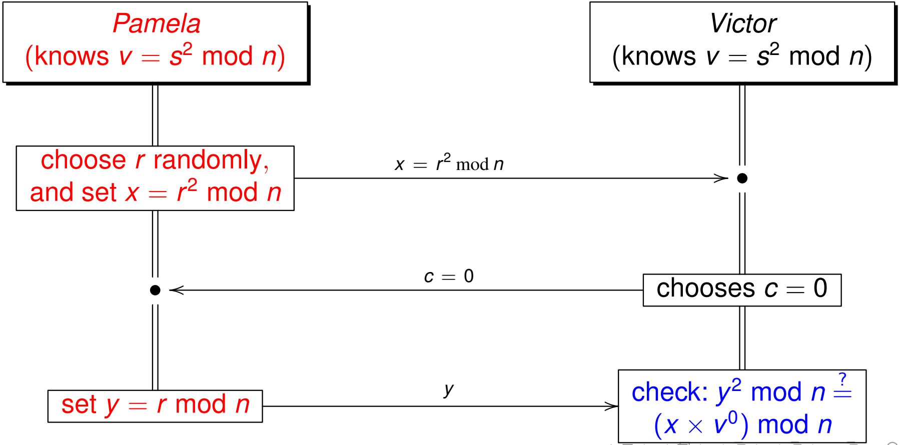
If the check

$$y^2 =_n (r \times s^c)^2 =_n r^2 \times s^{2c} =_n r^2 \times (s^2)^c =_n x \times v^c$$

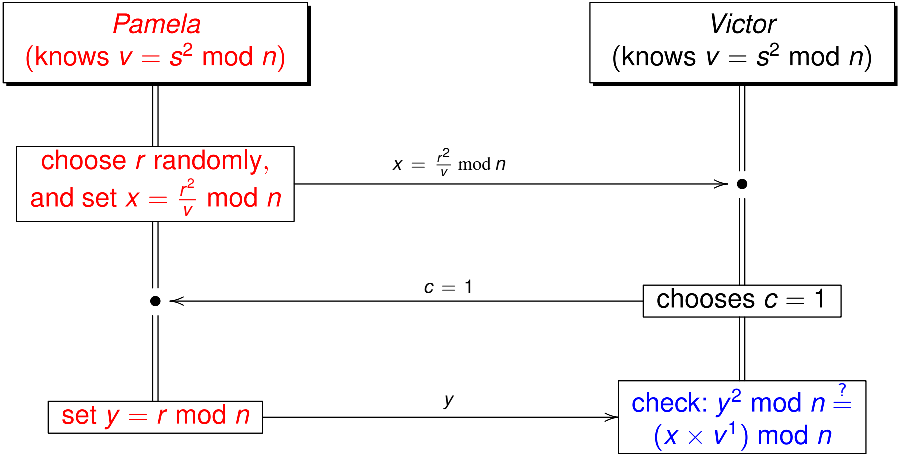
returns a yes, then verification is probable; otherwise, the process is aborted.

Fiat-Shamir: Trying to Cheat

Pamela does not know the secret s, but tries to prove its knowledge.
Pamela guesses that Victor is going to choose $c = 0$ (if she guesses wrong, then she loses).



Pamela guesses that Victor is going to choose $c = 1$ (if she guesses wrong, then she loses).



So, Pamela must find numbers x and y such that $x \times v =_n (x \times s^2)$.
Choose y randomly and then set $\frac{r^2}{v}$ mod n (division modulo n is also easy!).

Fiat-Shamir: Trying to Cheat

- Pamela has a strategy to cheat if $c = 0$:

Choose r randomly, set $x = r^2 \bmod n$.

Fiat-Shamir: Trying to Cheat

- Pamela has a strategy to cheat if $c = 0$:

Choose r randomly, set $x = r^2 \bmod n$.

- Pamela has a strategy to cheat if $c = 1$:

Choose r randomly, set $x = \frac{r^2}{v} \bmod n$.

Fiat-Shamir: Trying to Cheat

- Pamela has a strategy to cheat if $c = 0$:

Choose r randomly, set $x = r^2 \bmod n$.

- Pamela has a strategy to cheat if $c = 1$:

Choose r randomly, set $x = \frac{r^2}{v} \bmod n$.

- If $c \in \{0, 1\}$ is randomly chosen, Pamela has thus a chance of $\frac{1}{2}$ to cheat.

Fiat-Shamir: Trying to Cheat

- Pamela has a strategy to cheat if $c = 0$:

Choose r randomly, set $x = r^2 \bmod n$.

- Pamela has a strategy to cheat if $c = 1$:

Choose r randomly, set $x = \frac{r^2}{v} \bmod n$.

- If $c \in \{0, 1\}$ is randomly chosen, Pamela has thus a chance of $\frac{1}{2}$ to cheat.
- If Victor accepts only after n successful rounds, the chance to cheat is only $\frac{1}{2^n}$.

Fiat-Shamir: Trying to Cheat

- Pamela has a strategy to cheat if $c = 0$:

Choose r randomly, set $x = r^2 \bmod n$.

- Pamela has a strategy to cheat if $c = 1$:

Choose r randomly, set $x = \frac{r^2}{v} \bmod n$.

- If $c \in \{0, 1\}$ is randomly chosen, Pamela has thus a chance of $\frac{1}{2}$ to cheat.
- If Victor accepts only after n successful rounds, the chance to cheat is only $\frac{1}{2^n}$.
- We can conclude that

Pamela has no strategy to cheat for unpredictable c.

Fiat-Shamir: Curious Victor

Victor would like to learn the secret $x\ldots$
but we can conclude that

Fiat-Shamir: Curious Victor

Victor would like to learn the secret x...
but we can conclude that

Zero-Knowledge Property

Victor learns nothing except the proved statement.

Bibliography

Most of the figures in this lecture are taken from:

- William Stallings. *Cryptography and Network Security*. Fifth Edition, Prentice Hall, 2010.

Other interesting sources:

- The International PGP Home Page: `http://www.pgpi.org/`
- SDSI/SPKI (and PKI and PGP): `http://world.std.com/~cme/html/spki.html`
- Dieter Gollmann. *Computer Security*. Wiley, 2000.
- Bruce Schneier. *Applied Cryptography*. Wiley, 1996.
- Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Available online.
- Arthur E. Hutt, Seymour Bosworth, Douglas B. Hoyt. *Computer Security Handbook*. Wiley, 1995.