Cryptography[1]

Hans-Wolfgang Loidl
http://www.macs.hw.ac.uk/~hwloidl

School of Mathematical and Computer Sciences
Heriot-Watt University, Edinburgh

[1] Based on Goodrich's textbook, HAC, and Aspinall's slides

Stream Ciphers

Question

Recall Caesar's cipher: Why is it so easy to crack?

We can improve the strength of the Caesar cipher by:

- Performing a more general substitution of characters, rather than simple rotation. Keyspace grows from 26 to $26! = 403291461126605635584000000 =\tilde{} 4 \cdot 10^{27}$

Question

Does such increased keyspace make Caesar's cipher more secure?

- Using **different keys** on different characters.
- Transforming groups of characters, rather than individual characters.

Stream Ciphers

- A **stream cipher** encrypts a message **character by character**.
- The transformation that is applied typically varies over time.
- Stream ciphers are usually faster than block ciphers.
- They can be used even if the full message is not available, i.e. good for internet-style streaming.
- In some cases, hardware accelerators have been developed for stream ciphers, to speed up en-/de-cryption further.
- Because the handle character-by-character, they have limited error propagation, and transmission errors are less disruptive.

One-time pads

A one-time pad, or **Vernam cipher**, is an unbreakable stream cipher:

- Invented by Joseph Mauborgne and Gilbert Vernam in 1917.
- It uses rotations, similar to the Caesar cipher, but a **different** key for each element in the plain text
- This cipher is **provable secure** if keys are never reused.
- Reportedly, it has been used during the Cold War for secure communication between Washington and Moscow.
- However, it is **impractical** because it needs a sequence of keys as long as the message length
- Reportedly, key re-use happened quite often in communication between Soviet spies, allowing the US to attack their communication.
- To get around the problem of key re-use, sometimes **pseudo-random sequences** are used, which generate sequences of numbers that appear to be random, from a much smaller, secret key.
- For efficient implementation, this cipher is often used over binary numbers, performing an XOR as operation, i.e. $\mathcal{M} = \mathcal{C} = \mathcal{A} = \{0, 1\}$

Applications to Wireless Networks

- The Wired Equivalent Privacy (WEP) protocol was included in the 802.11 standard for secure networking, to provide confidentiality, integrity and access control to a wireless network.
- WEP has several weaknesses and should be replaced with WPA (WPA Enterprise is considered secure).
- WEP uses a **stream cipher** with a **symmetric cryptosystem**, namely the **RC4** asynchronous stream cipher.
- The RC4 cipher generates a pseudo random number sequence, using a 256-bit seed value.
- The operation on each stream element is an XOR.
- Thus, the RC4 cipher is very fast.
- The seed is generated out of a 24-bit initialisation vector (IV) and a WEP key, that is shared between user and access point.
- The IV is sent together with the cipher-text, so that the access point can reconstruct the pseudo random number sequence, needed to decrypt the message.
- For integrity, a **CRC-32 checksum** is added to the plain-text.

Attacks against WEP

- Two protocols are used for wireless authentication:
  - **open system**, where the access point trusts the user in having acquired the WEP key, and encrypts all communication
  - **shared key**, where additionally the access point regularly sends a challenge text that must be encrypted and sent back
- In practice, shared key is **less** secure than open system, because plain-text and IV are sent unencrypted.
- Open systems can be attacked, if a lot of packets are obtained, exploiting weakness of the pseudo-random number generator.
- As of 2010, a WEP key can be cracked with a probability of 50%, if 40,000 packets have been obtained.
- Through ARP injection, an attacker can gain an arbitrary number of packets.

Attacks against WEP

- Another attack exploits weaknesses due to performing (CRC-32) hashing. The attacker can capture and truncate a packet. Then he guesses the truncated byte and sends it. Iff the access point responds, the byte was correct. This is repeated until an entire message has been reconstructed.
- Another attack sets up a honeypot, acting as a fake access point, which is easy since WEP doesn't check whether the access point has the WEP key (only the client is checked). The attacker then tricks the client into sending further messages, until enough messages have been received to crack the WEP key. Note, that this doesn't have to be done in the targeted network itself!

Block ciphers

- A **block cipher** encrypts groups of characters in a message.
- The transformation is the same for each group.
- Pure block ciphers are memoryless, i.e. earlier data in a message does not influence the encryption of a later part of the message.

Basic design concepts for good ciphers:

- **Confusion:** A good cipher should add confusion, obscuring the relationship between the key and the ciphertext.
- **Diffusion:** A good cipher should add diffusion, spreading out redundancy in the plaintext across the ciphertext.
- Typically, modern block ciphers
  - use **substitution** to add confusion;
  - use **transpositions** to add diffusion;
  - apply rounds consisting of substitution and transposition steps to improve both.
- **Note:** a large keyspace does not guarantee a strong cipher, as we have seen with the generalised Caesar's cipher.

Simple substitution ciphers

A simple substitution cipher is a block cipher for arbitrary block length t. It swaps each letter for another letter, using a permutation of the alphabet.

- Let $\mathcal{A}$ be an alphabet, $\mathcal{M}$ be the set of strings over $\mathcal{A}$ of length t, and $\mathcal{K}$ be the set of all permutations on $\mathcal{A}$.
- For each $e \in \mathcal{K}$ define E_e by applying the permutation e to each letter in the plaintext block:

$$E_e(m) = e(m_1)e(m_2)\cdots e(m_t) = c$$

  where $m \in \mathcal{M}$ and $m = m_1 m_2 \cdots m_t$.
- For each $d \in \mathcal{K}$ we define D_d in exactly the same way,

$$D_d(c) = d(c_1)d(c_2)\cdots d(c_t)$$

- Key pairs are permutations and their inverses, so $d = e^{-1}$, and

$$D_d(c) = e^{-1}(c_1)e^{-1}(c_2)\cdots e^{-1}(c_t) = m_1 m_2 \cdots m_t = m$$

Simple transposition ciphers

The simple transposition cipher is a block cipher with block-length t. It simply permutes the symbols in the block.

- Let $\mathcal{K}$ be the set of all permutations on the set $\{1, 2, ..., t\}$.
- For each $e \in \mathcal{K}$, the encryption function is defined by

$$E_e(m) = (m_{e(1)}, m_{e(2)}, \ldots, m_{e(t)})$$

- The corresponding decryption key is the inverse permutation.

Product ciphers

- It's easy to combine encryption functions using **composition**, because the composition of two bijections is again a bijection.
- A product cipher is defined as the composition of N encryption transformations, $E_e^1, E_e^2, \ldots, E_e^N$, for $n \geq 0$.
- The overall encryption function composes the parts:

$$E_e = E_e^{(1)} \circ E_e^{(2)} \circ \cdots \circ E_e^{(N)}$$

  where $\circ$ denotes function composition in the diagramatic order.
- The overall decryption function composes the decryptions:

$$D_d = D_d^{(N)} \circ \cdots D_d^{(2)} \circ D_d^{(1)}$$

- **Involutions** (functions that are their own inverse) are particularly useful in constructing product ciphers. The favourite is XOR: $f(x) = x \oplus c$.

DES

Data Encryption Standard (DES) is a symmetric block cipher:

- DES is a block cipher based on Feistel's principle in order to provide effective confusion and diffusion. Block-size is 64 bits, key-size 56 bits (+8 parity bits). Invented by IBM in 1970s, tweaked by NSA. Still widely used, esp. in financial sector. Much analysed.
- The **Feistel principle** gives a way of constructing a cipher so that the same circuit is used for both encryption and decryption.
- Main threat isn't cryptanalytic, but (slightly optimised) exhaustive search in small key-space. Remedied by 3DES (triple DES), 3 keys:

$$C = E_{k_3}(D_{k_2}(E_{k_1}(P))) \quad P = D_{k_1}(E_{k_2}(D_{k_3}(C))).$$

  Security of 3DES is not obvious: repeated encryption may not gain security (one-step DES is not closed, so it in fact does), and new attacks may be possible (meet-in-the-middle attack). With 3 independently chosen keys, security is roughly the same as expected with 2 keys.
- Several other DES variants, including DESX, using whitenening keys k_1, k_2 as $C = E_k(P \oplus k_1) \oplus k2$. (Used in Win2K encrypting FS).

The AES (or Rijndael) symmetric cipher

- Rijndael is a block cipher with a fixed block length of 128 bits.
- Since 2001 it is the AES standard for symmetric encryption set by the U.S. National Institute for Standards and Technology (NIST)
- It replaces the older DES cipher, which has a proven weakness.
- Rijndael can be used with key lengths of 128, 192 or 256-bits.
- Rijndael satisfied a number of requisite criteria for the AES:
  - **Security:** mathematical, cryptanalytic resistance; randomness;
  - **Efficiency:** time/space, hardware and software;
  - **Flexibility:** block sizes 128 bits, key sizes 128/192/256 bits.
  - **Intellectual property:** unclassified, published, royalty-free.
- Rijndael is built as a network of linear transformations and substitutions, with 10, 12 or 14 rounds, depending on key size.

Structure of the Rijndael (or AES) algorithm

- Initialise the state, by performing an XOR of plain-text and key
- Perform **10 rounds** of modifying the state
- The result is the state after these 10 rounds.

Each round consists of the following steps:

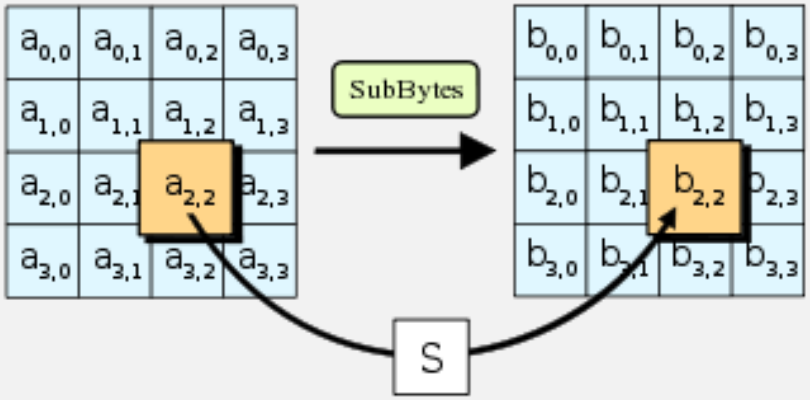
- **SubByte step:** perform an S-box substitution
- **ShiftRows step:** perform a permutation
- **MixColumns step:** a matrix multiplication step
- **AddRoundKey step:** an XOR operation with the round key, derived from the key

Note: Each step is invertible, so de-crypting amounts to running the algorithm in the reverse order.
Note: All steps operate on a **state**, which is a 4×4 matrix of the (8-bit) bytes of a block

SubByte Step

- **Input:** 4×4 matrix A
- **Output:** 4×4 matrix B, by performing a **component-wise operation** S-Box transformation S one each element of A.
- S computes for one byte of the input matrix the **multiplicative inverse** over the finite field $GF(2^8)$ combined with an invertible affine transformation.
- Importantly, this operation assures non-linearity of the transformation.
- This operation S can be performed efficiently, by performing a lookup in a 16×16 table.

SubByte Step

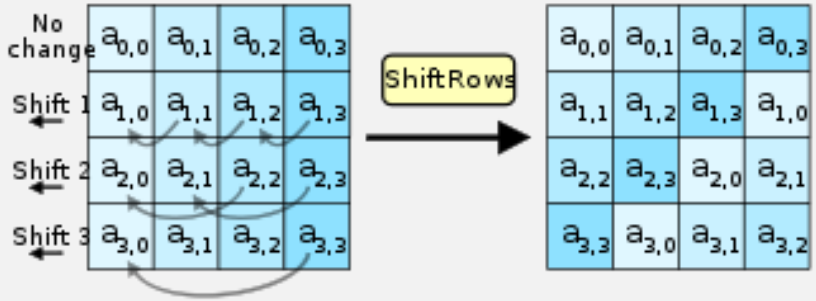


[1] From Wikipedia: Advanced Encryption Standard

ShiftRows Step

- **Input:** 4×4 matrix A
- **Output:** 4×4 matrix B, by performing a **permutation** on the elements of A.
- The permutation is done row-wise, **shifting** the 0-th row by 0, 1-st row by one etc.

ShiftRows Step

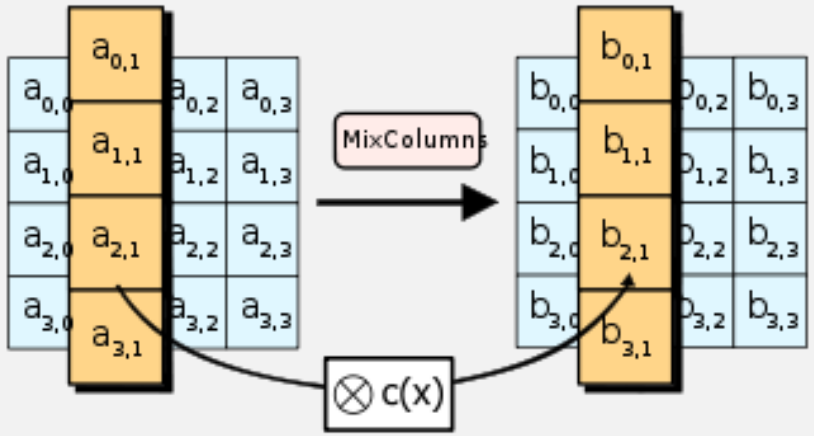


[1] From Wikipedia: Advanced Encryption Standard

MixColumns Step

- **Input:** 4×4 matrix A
- **Output:** 4×4 matrix B, by performing a Hill-cipher **matrix multiplication** of a fixed matrix E with A.
- The matrix multiplication uses an XOR operation to add two bytes, and uses a polynomial product, modulo a fixed base, for multiplication.
- Although this seems complicated, it can be implemented very efficiently.

MixColumns Step

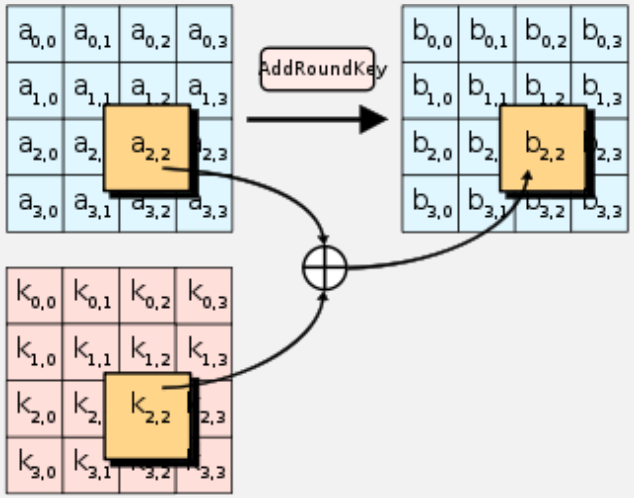


[1] From Wikipedia: Advanced Encryption Standard

AddRoundKey Step

- **Input:** 4×4 matrix A
- **Output:** 4×4 matrix B, by performing a **component-wise XOR** with a key set S.
- The key set S is a 4×4 matrix of keys derived from the main key.
- This derivation starts with the main key, and generates a **pseudo-random-number** sequence with it as seed value

AddRoundKey Step



[1] From Wikipedia: Advanced Encryption Standard

Properties of AES

To summarise, the main notable properties of AES are:

- It is a **symmetric block cipher**, with possible key lengths of 128, 192 or 256-bits.
- It is an iterative cipher, which applies the same sequence of operations in 10, 12 or 14 rounds.
- It performs permutations in order to prevent statistics-based attacks.
- All operations are invertible, so that for decryption the algorithm is run with the same key in reverse order.
- All operations are designed to be efficiently implementable.
- It is probably the “best” symmetric cipher today.

Attacks on AES

- As of 2010, the only known practical attacks on AES are side channel attacks, in particular timing attacks.
- A **side channel attack** observes the behaviour of the machine that performs en-/de-cryption.
- A **timing attack** observes the **time** the algorithm takes.
- Because the main operations in AES are table lookups, and because lookups have largely varying time when performed in memory or in cache, by timing the runtime of the algorithm on known plaintext, ciphertext pairs gives a possibility to learn the key.
- To defend against this weakness, the implementation of AES should provide constant time memory lookup.
- This can be done by disabling the cache, but this slows down en-/de-cryption a lot.

Related ciphers

A symmetric block cipher, similar to AES, is **Twofish**, by Bruce Schneier.

- It uses a similar overall structure of rounds as AES, but uses a 16-round Feistel network
- Distinctive features are
  - pre-computed key-dependent S-boxes, and
  - a more complex key schedule than AES
- It was one of five finalists in the process of defining the AES standard.
- Its reference implementation has been placed in the public domain
- It is supported by most encryption libraries such as OpenSSL.
- Some libraries support only **blowfish**, which is a weaker predecessor of twofish.

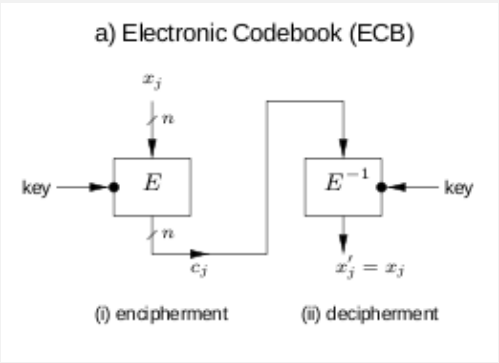
Related ciphers (cont'd)

Another related symmetric block cipher is **IDEA**.

- IDEA uses 64-bit blocks and 128-bit key.
- Its implementation is very efficient: uses XOR, addition and multiplication operations.
- Patented for commercial use.
- It is used in the PGP infrastructure for secure email.

ECB: electronic codebook mode

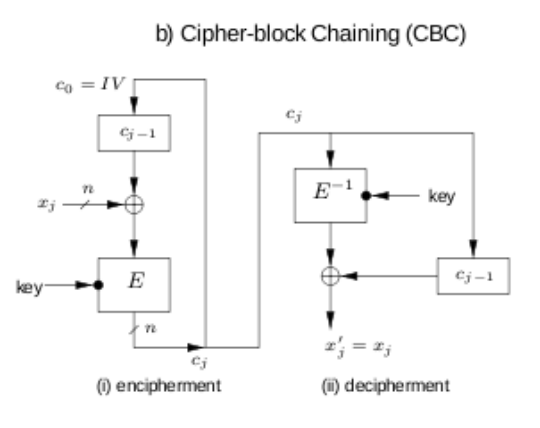
ECB: electronic codebook mode. Each block of plaintext x_j is **enciphered independently**: $c_j = E_k(x_j)$



[1] From: HAC, Chapter 8, p 229

CBC: cipherblock chaining mode

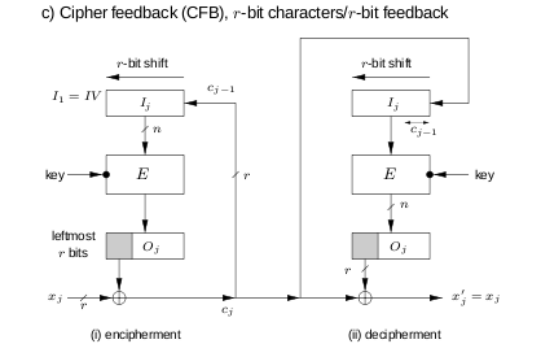
CBC: cipherblock chaining mode. Each plaintext block x_j is XORed with the previous ciphertext c_{j-1} block before encryption. An initialization vector (IV) (optionally secret, fresh for each message) is used for c_0: $c_j = E_k(x_j \oplus c_{j-1})$



[1] From: HAC, Chapter 8, p 229

CFB cipher-feedback mode

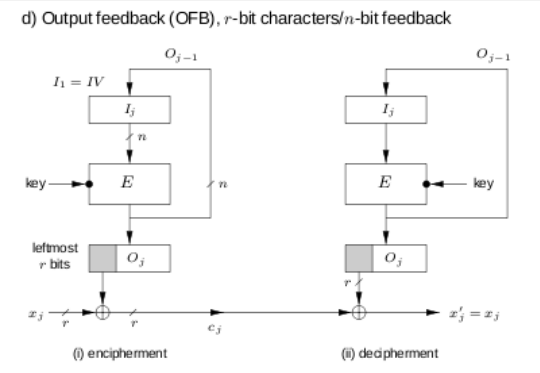
CFB cipher-feedback mode. Encryption function of block cipher used as self-synchronizing stream cipher for symbols of size up to block size, i.e. $c_j = E_k(c_{j-1}) \oplus x_j$



[1] From: HAC, Chapter 8, p 229

OFB: output-feedback mode

OFB: output-feedback mode. Conceptually, this mode works like a one-time pad, but with the sequence of blocks that are generated with the block cipher. Note, that the encryption function is only directly applied to the vectors, not directly to the plain-text:
$c_j = O_j \oplus x_j$ where $O_j = E_k(O_{j-1})$



[1] From: HAC, Chapter 8, p 229

CTR counter mode

CTR counter mode is similar to CFB mode in structure. Each element in the one-time-pad is computed directly from a seed value. Therefore, the entire sequence can be computed in parallel:
$c_j = O_j \oplus x_j$ where $O_j = E_k(s + j - 1)$

Summary of modes of symmetric key encryption

Different **modes** apply block-level encryption in different ways.

- **ECB: electronic codebook mode.** Each block of plaintext x_j is **enciphered independently**.

Question

Can you see a problem with this mode?

- **CBC: cipherblock chaining mode.** Each plaintext block x_j is XORed with the previous ciphertext c_{j-1} block before encryption. An initialization vector (IV) (optionally secret, fresh for each message) is used for c_0 .
- **CFB cipher-feedback mode.** Encryption function of block cipher used as self-synchronizing stream cipher for symbols of size up to block size.
- **OFB: output-feedback mode.** Block cipher encryption function used as synchronous stream cipher (internal feedback).

Exercises (optional)

Exercise

Symmetric encryption

- *Read Section 7.2.2. of HAC, for details on the different modes of block ciphers.*
- *Check Section 2.1 of Goodrich for details on symmetric encryption.*
- *Familiarise yourself with the `openssl` toolset installed on the Linux lab machines. These will be used in the next coursework.*

Modular Arithmetic

- We define $\mathbb{Z}_p = \{0, \ldots, p-1\}$, i.e. the set of all positive integer numbers modulo p.
- Basic arithmetic on these numbers is always *modulo p*, i.e. we take the remainder of dividing the result by p.
- Example: Multiplying 4 with 3 in $\mathbb{Z}_{11}$ gives

$$4 \cdot 3 \mod 11 = 12 \mod 11 = 1$$

- The inverse of x in $\mathbb{Z}_p$, written x^{-1}, is a number such that

$$x \cdot x^{-1} \mod p = 1$$

- Note: If p is a prime number, every element in $\mathbb{Z}_p$ has an inverse.
- Example: The above example shows the 3 is the inverse of 4 in $\mathbb{Z}_{11}$.
- Modular exponentiation, x^y *mod* p, is repeated multiplication, as usual, i.e. $\underbrace{x \cdots x}$ *mod* p.

Basic Number Theory

Let $n \in \mathbb{Z}, z \in \mathbb{Z}_n$. We define $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid \gcd(x, n) = 1\}$.
We define the *Euler totient function*, $\Phi(n) = \mid \mathbb{Z}_n^* \mid$.

Fact (Euler's Theorem)

If $n \in \mathbb{Z}, x \in \mathbb{Z}_n, \gcd(x, n) = 1$, *then* $x^{\Phi(n)} \mod n = 1$

Corollary

If $n \in \mathbb{Z}, x \in \mathbb{Z}_n, \gcd(x, n) = 1$, *then* $x^y \mod n = x^{y \mod \Phi(n)} \mod n$

Proof.

Let $d, r \in \mathbb{Z}, 0 \leq r < \Phi(n)$, such that $y = d \cdot \Phi(n) + r$.

$$\begin{aligned}
x^y \mod n &= x^{d \cdot \Phi(n) + r} \mod n && \text{def. of } y, d, r \\
&= x^{d \cdot \Phi(n)} \cdot x^r \mod n && \text{arithm.} \\
&= \left(x^{\Phi(n)}\right)^d \cdot x^r \mod n && \text{arithm.} \\
&= 1^d \cdot x^r \mod n && \text{Euler's theorem} \\
&= x^r \mod n && \square
\end{aligned}$$

Basic Number Theory

Another corollary from Euler's Theorem let's us compute the inverse of a given number $x \in \mathbb{Z}_n$, by computing $x^{-1} = x^{\Phi(n)} \mod n$, because:

Corollary

If $n \in \mathbb{Z}, x \in \mathbb{Z}_n, \gcd(x, n) = 1$, *then*
$x \cdot x^{\Phi(n)-1} \mod n = x^{\Phi(n)} \mod n = 1$

Since $\Phi(p) = p - 1$, if p is a prime number, we have the following specialised theorem:

Fact (Fermat's Little Theorem)

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

Computing in Modular Domains

Definition

The greatest common divisor (gcd) of two positive integers, $x, y \in \mathbb{Z}, x, y > 0$, is the largest positive integer $z \in \mathbb{Z}$, that divides both x and y, written $z \mid x$ and $z \mid y$.

Fact

*The greatest common divisor (*gcd*) of two positive integers,* $x, y \in \mathbb{Z}, x, y > 0$, *is the smallest positive integer* $z \in d$, *such that there exist* $i, j \in \mathbb{Z}$ *with* $z = i \cdot x + j \cdot y$.

Corollary (Extended Euclidean Algorithm)

$$\begin{array}{lcl} \gcd(x, 0) & = & x \\ \gcd(x, y) & = & \gcd(y, x \mod y) \end{array}$$

Note: We can use this algorithm to compute the inverse of a x in $\mathbb{Z}_p$, if x and p are relative prime: Since $\gcd(x, p) = i \cdot x + j \cdot p = 1$, we have $i \cdot x + j \cdot p \mod p = i \cdot x \mod p = 1$, and therefore i is the inverse of x in $\mathbb{Z}_p$.

Computing in Modular Domains

For efficient exponentiation, we break down the exponent into its components, using the following property:

Fact (Exponentiation by repeated squaring)

$x^{2n} \mod p = (x^n)^2 \mod p$

Note: We can perform efficient exponentiation, by repeated squaring, employing a divide-and-conquer strategy [2].

[2] See: *"De Bello Gallico"*, Gaius Julius Caesar.

Finite Fields

- $(\mathbb{Z}_p, +, \cdot)$ is a finite field.
- A number g is called a *generator*, if any element in $\mathbb{Z}_p$ can be constructed as g^n for some $n \in \mathbb{N} \mod p$.
- We know, if p is a prime number, then every $(\mathbb{Z}_p, +, \cdot)$ has $\Phi(\Phi(p)) = \Phi(p - 1)$ generators.
- A number $g \in \mathbb{Z}_p$ is a generator, iff $g^{\frac{p-1}{p_i}} \mod p \neq 1$ for all prime factors of p_i of $\Phi(p) = p - 1$.

Further Reading on Mathematical Background

- Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *"Handbook of Applied Cryptography"*, CRC Press, 2001. ISBN 0-8493-8523-7.
  On-line: `http://www.cacr.math.uwaterloo.ca/hac/`
  Chapter 2.
- Nigel Smart, *"Cryptography: An Introduction"*,
  On-line:
  `http://www.cs.bris.ac.uk/~nigel/Crypto_Book/`
  Chapter 11, 12, 13.

Limitations of symmetric encryption

- Symmetric encryption is fine to assure privacy of your own data, i.e. only you should be able to read the data
- In this case it's natural to have just one secret key
- But what if you want to securely transmit data and you want to assure that only the intended recipient should be able to read the message?
- You could use a secret key, known only to both of you.
- But what if you want to securely communicate with many recipients?

Question

If n participants want to communicate securely, making sure that only the 2 participants can read a message, how many secret keys are needed in total?

The idea of public-key encryption

Public-key encryption takes a radically different approach:

- Every participant in a communication has two keys
  - a **private key**, which is kept secret
  - a **public key**, which can be published e.g. on the web
- For safe communication, use the persons **public key** to **encrypt** the data.
- The recipient uses his **private key** to **decrypt**.
- Since the private key is kept secret, only he can read the message.
- This technology relies on the fact, that the private key cannot be efficiently computed from just knowing the public key (and the crypto algorithm).
- This idea was a **major research breakthrough** in the area of cryptography.

Features of public-key cryptosystems

Features of public key (PK) encryption:

- PK encryption provides privacy, ie. prevents unauthorised persons from reading the cipher-data
- PK encryption **does not** provide data origin authentication or data integrity
- A public key infrastructure (PKI) is used to distribute keys
- PKIs need separate techniques to ensure data origin authentication

Features of public-key cryptosystems (cont'd)

Advantages of public key encryption:

- No secret shared key is needed
- Needs only n keys (the public keys) for secure communication between n recipients.

Disadvantages of public key encryption:

- Encryption is significantly slower than symmetric key encryption
- The key length for public key cryptosystems is typically one order of magnitude larger than for symmetric cryptosystems (typically 2048-bit as opposed up to 256-bit for AES)

Therefore, PK encryption is mainly used to encrypt **session keys**:

- Alice randomly generates a strong, symmetric key (a *session key*).
- Alice uses Bob's public key to encrypt and send the session key.
- Alice encrypts all plaintext for one session using the session key, and sends it.
- Bob uses his secret key to decrypt the session key.
- Bob now can decrypt all messages from Alice that use the same session key.

RSA encryption: a public-key cryptosystem

RSA encryption, a milestone in cryptography:

- RSA is the most commonly used public-key encryption systems
- It is named after the inventors: R. Rivest, A. Shamir, L. Adleman (**Turing Award 2002**).
- Its security is based on the intractability of **integer factorisation**.
- It is conceptually very simple, and founded on strong, mathematical properties.
- It can be used for encryption and for digital signatures.

RSA encryption in a nutshell

Alice wants to securely communicate with Bob, using RSA. First Bob generates a key-pair as follows:

- Bob picks two large, random **prime numbers** p, q. Let $n = pq$.
- Bob picks a number e that is **relatively prime** to $\Phi(n)$.
- Bob computes the **inverse of e**, modulo $\Phi(n)$, i.e. $d = e^{-1} \mod \Phi(n)$
- Bob's public key is (e, n). He publishes this key, e.g. on the web.
- Bob's private key is d.

Now, Alice can use Bob's public key as follows:

- Alice encrypts the plaintext **M**, using Bob's public key, by computing $\mathbf{C} = \mathbf{E}_{(e,n)}(\mathbf{M}) = \mathbf{M}^{e} \mod \mathbf{n}$.
- **Note:** Encryption amounts to a single, modular exponentiation.
- Alice sends the ciphertext **C** to Bob.
- Bob decrypts the ciphertext **C**, using his private key, by computing $\mathbf{D}_{d}(\mathbf{C}) = \mathbf{C}^{d} \mod \mathbf{n}$.
- **Note:** Decryption amounts to a single, modular exponentiation.

RSA: Why does this work?

Theorem (RSA Correctness)

Forall e, d, p, q, n, m, *if* $n = pq \wedge e \perp \Phi(n) \wedge d = e^{-1} \mod \Phi(n)$ *then* $D_d(E_e(M)) = M$

In other words, if we pick the appropriate domain parameters and generate a public-/private-key pair (e, d) out of them, then we can obtain the original message by decrypting the cipher-text, generated by the encryption function.

RSA: Why does this work?

Proof.

Assume, M is relatively prime to $n = pq$:

$$\begin{aligned} C^d \mod n &= (M^e)^d \mod n && \text{def. of D and E} \\ &= M^{ed} \mod n && \text{arithm.} \\ &= M^{ed \bmod \Phi(n)} \mod n && \text{Euler's Theorem} \\ &= M^1 \mod n && \text{def. of } d \\ &= M \end{aligned}$$

Assume, without loss of generality, there is an i such that $M = ip$ **(1)**

Since $ed = 1 \mod \Phi(n)$, we know $ed = k\Phi(n) + 1$ for some k **(2)**

Since $\Phi(n) = \Phi(p)\Phi(q)$, we know $M^{\Phi(n)} \mod q = M^{\Phi(p)\Phi(q)} \mod q = 1$.

Therefore, $M^{k\Phi(n)} \mod q = 1$, and we can write $M^{k\Phi(n)} = 1 + hq$ **(3)**

$$\begin{aligned} C^d \mod n &= M^{ed} \mod n && \text{def. of D and E, arithm.} \\ &= M^{k\Phi(n)+1} \mod n && \text{by (2)} \\ &= (M + Mhq) \mod n && \text{by (3) and arithm.} \\ &= (M + iphq) \mod n && \text{by (1)} \\ &= (M + (ih)pq) \mod n && \text{by assoc. and comm.} \\ &= (M + (ih)n) \mod n && \text{def. of } n \\ &= M && \text{arithm.} \end{aligned}$$

RSA Remarks

- RSA is an example of a reversible public-key encryption scheme. This is because e and d are symmetric in the definition. RSA digital signatures make use of this.
- RSA is a **deterministic** algorithm, i.e. if messages $M_1 = M_2$ then the cipher texts $C_1 = C_2$. Thus, RSA is often used with randomisation (e.g., salting with random appendix) to prevent chosen-plaintext and other attacks.
- RSA is the most popular and cryptanalysed public-key algorithm. Largest modulus factored in the (now defunct) RSA challenge is 768 bits (232 digits), factored using the Number Field Sieve (NFS) on 12 December 2009.
  - It took the equivalent of 2000 years of computing on a single core 2.2GHz AMD Opteron. On the order of 267 instructions were carried out.
  - Factoring a 1024 bit modulus would take about 1000 times more work (and would be achievable in less than 5 years from now).
  - Thus, key lengths of **at least 2048 bits** are recommended.

RSA remarks (cont'd)

- In practice, RSA is used to **encrypt symmetric keys** (session keys), not messages
- Like most public key algorithms, the RSA key size is larger, and the computations are more expensive (compared to AES, for example)
- This is believed to be a necessary result of the key being publicly available
- With regard to attack complexity based upon an n-bit key
  - A worst-case attack algorithm on a symmetric cipher would take $O(2^n)$ work (exponential).
  - A worst-case attack algorithm for RSA is dependent upon the complexity of factoring, and thus would take $O(e^{o(n)})$ (sub-exponential)

Security of RSA

- The security of RSA is based on the difficulty of finding the private key d, from the public key (e, n).
- If an attacker knew $\Phi(n)$ it would be easy to compute d, because $d = e^{-1} \mod \Phi(n)$ which can be computed using the extended Euclidean algorithm.
- **Note:** p, q, with $n = pq$, and therefore $\Phi(pq)$ are only needed in the key generation. They can, and should, be destroyed thereafter.
- But, if the attacker can factor $\Phi(pq)$, he can compute $p - 1$ and $q - 1$, and from that d.
- Thus, conceptually the security of RSA is tied to the complexity of factoring a large integer number.

Cryptographic Reference Problems

FACTORING Integer factorisation. Given positive n, find its prime factorisation, i.e., distinct p_i such that

$$n = p_1^{e_1} \cdots p_n^{e_n} \text{ for some } e_i \geq 1$$

SQRROOT Given a such that $a = x^2 \mod n$, find x.

RSAP RSA inversion. Given n such that $n = pq$ for some odd primes $p \neq q$, and e such that $gcd(e, (p-1)(q-1)) = 1$, and c, find m such that $m^e = c \mod n$.

DLP Discrete logarithm problem. Given prime p, a generator g of $\mathbb{Z}_p$, and an element $a \in \mathbb{Z}_p$, find the integer x, with $0 \leq x \leq p-2$ such that $g^x = a \mod p$.

Note: SQRROOT $=_P$ FACTORING and RSAP $\leq_P$ FACTORING
$A \leq_P B$ means there is a polynomial time (efficient) reduction from problem A to problem B. $A =_P B$ means $A \leq_P B$ and $B \leq_P A$

Efficient implementation of RSA

Efficient implementation of RSA needs efficient algorithms for

- **Primality testing** is needed during **key generation**, when picking p and q.
- **GCD computation**: computing the greatest common divisor (GCD) is needed during **key generation**, when picking e relatively prime to $\Phi(n)$.
- **Modular inverse computation** is needed during **key generation**, when computing $d = e^{-1} \mod \Phi(n)$.
- **Modular exponentiation** is needed during **encryption and decryption**.

Further Reading on Efficient Implementations

Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*,
CRC Press, 2001. ISBN 0-8493-8523-7.
Chapter 14: Efficient Implementation
Very detailed discussion of implementation issues of all cryptosystems presented here.

Nigel Smart, *Cryptography: An Introduction*,
Chapter 15: Implementation Issues
Good general discussion of the main implementation issues.

Donald Knuth, *The Art of Computer Programming: Volume 2: Seminumerical Algorithms*,
Addison-Wesley, 1975.
The foundations for most of the efficient algorithms.

Further Reading on Efficient Implementations

R.L. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*,
In *Communications of the ACM*, Vol 21, No 2, Feb 1978, pp. 120-126.
The original paper discussing the RSA public-key cryptosystem.

W. Diffie, M.E. Hellman, *New Directions in Cryptography*,
In *IEEE Transactions on Information Theory*, Vol 22, Nov 1976, pp. 644-654.
The original paper discussing public-key infrastructures and secure key exchange.

P.L. Montgomery, *Modular Multiplication without Trial Division*,
In *Mathematics of Computation*, Vol 44, No 170, April 1985, pp. 519-521.
The original paper describing efficient multiplication in a modular domain.

C.K. Koc, T. Acar, B.S. Kaliski, *Analyzing and Comparing Montgomery Multiplication Algorithms*, In *IEEE Micro*, Vol 16, No 3, 1996, pp 26-33.
Survey of different ways to perform multiplication in a modular domain, and thus how to perform exponentiation (the core of RSA en-cryption).

Rabin public-key encryption

Rabin encryption is a **provably secure** public-key encryption scheme in the sense that it has been formally proven that recovering the plaintext from the ciphertext is computationally equivalent to factoring.

Key generation:

- Generate two large prime numbers p, q.
- Compute $n = pq$.
- The public key is n.
- The private keys is (p, q).

Rabin public-key encryption

Encryption:

- Obtain the recipient's public key n.
- Represent the message as an integer m in the range of $\{0, 1, \ldots, n-1\}$.
- **Compute:** $c = m^2 \mod n$
- Send the cipher text c.
- In summary: the encryption function is $E_n(m) = m^2 \mod n$

Decryption:

- Find the four square roots m_1, m_2, m_3, m_4 of c modulo n.
- The plain text message is one of m_1, m_2, m_3, m_4.
- In summary: the decryption function is $D_n(m) = select(\sqrt{c} \mod n)$.
- For details on the Rabin cryptosystem, see HAC, Section 8.3 or Smart, Section 11.3

ElGamal Cryptosystem

- ElGamal is a public-key cryptosystem, which uses **randomisation** so that independent encryptions of the same plain-text give different cipher-texts.
- Blocks of input text are considered as numbers.
- En-/De-cryption is done by performing arithmetic on these numbers.
- Bob chooses a prime number p such that we can easily find a generator g for $\mathbb{Z}_p$.
- Bob chooses a number between x between 1 and $p-2$, and computes $y = g^x \mod p$.
- Bob's **private key** is x.
- Bob's **public key** is (p, g, y).

Elgamal Cryptosystem (cont'd)

- To encrypt a message m, Alice chooses a number k between 1 and $p-2$, and performs this operation:

$$E_{(p,g,y)}(M) = (g^k \mod p, My^k \mod p)$$

- To decrypt a cipher-text, which is a pair of the form (a, b), Bob performs this operation:

$$D_x(a, b) = b(a^x)^{-1} \mod p$$

- Note, that g^{kx} does not need to be transmitted. It acts as a random, one-time pad.
- Note, that, as in any public-key cryptosystem, Alice does not need to know Bob's secret key.
- Pragmatically, the security of the Elgamal cryptosystem depends on choosing a new random number k for every transmission.
- Fundamentally, the security of the Elgamal cryptosystem depends on the difficulty of the **discrete logarithm** problem.
- A similar method can be used for secure key exchange, as defined by the **Diffie-Hellman key exchange protocol**.

Correctness of ElGamal Cryptosystem

Theorem (Correctness of ElGamal)

For all $p \in \mathbb{Z}$, p is prime, $g \in \mathbb{Z}_p$, g is a generator of $\mathbb{Z}_p$, $y = g^x \mod p$, $x \in \mathbb{Z}_p$, $1 \leq x \leq p-2$, messages $M \in \mathcal{M}$, $D_x(E_{(p,g,y)}(M)) = M$.

Proof.

$$\begin{aligned} b(a^x)^{-1} \mod p &= My^k((g^k)^x)^{-1} \mod p && \text{def. of } a \text{ and } b \\ &= M(g^x)^k(g^{-kx}) \mod p && \text{def. of } y \text{ \& arithm.} \\ &= Mg^{kx}(g^{-kx}) \mod p && \text{arithm.} \\ &= M \mod p && \text{arithm.} \\ &= M \end{aligned}$$

□

Diffie-Hellman key exchange protocol

Problem: In order to use a symmetric cipher, how is the shared key exchanged in the first place?

Definition (Key Exchange Protocol)

A *Key Exchange Protocol* is a cryptographic approach to establishing a shared secret key by communicating solely over an insecure channel, without any previous private communication.

The **Diffie-Hellman protocol** is such a key exchange protocol that is secure against passive adversaries.

Diffie-Hellman key exchange protocol

The **Diffie-Hellman protocol** is based on an ElGamal (asymmetric) cipher, using the public parameters are (p,g).

1. Alice chooses a random number x in $\mathbb{Z}_p$ and sends:
$$X = g^x \mod p$$
2. Bob chooses a random number y in $\mathbb{Z}_p$ and sends:
$$Y = g^y \mod p$$
3. Alice computes the secret key as
$$K_1 = Y^x \mod p$$
4. Bob computes the secret key as
$$K_2 = X^y \mod p$$
5. Note, that both Alice and Bob now have the same, secret key g^{xy}:

$$K_1 = Y^x \mod p = (g^y)^x \mod p = (g^x)^y \mod p = X^y \mod p = K_2$$

Diffie-Hellman key exchange protocol

The Diffie-Hellman protocol is secure against passive adversaries, because it is infeasible to compute the secret values x and y from the public parameters (p,g) and the messages $X = g^x$ and $Y = g^y$.

An attacker would have to compute x, y from $g^{xy} \mod p$
$\Longrightarrow$ **discrete logarithm problem**.

The Diffie-Hellman protocol is vulnerable against man-in-the-middle attacks (active adversaries):

- The attacker chooses s, t in $\mathbb{Z}_p$
- The attacker uses s for a key exchange with Alice
- The attacker uses t for a key exchange with Bob
- Now, the attacker can decode all intercepted messages (using s or t) and re-encode them with his own fake key (t or s)

Suggested Exercises (optional)

Exercise

Modular arithmetic:

- *Generate a multiplication table for $\mathbb{Z}_7$. What are the inverses of $3, 5, 6$?*
- *How can you find the inverse for 6 without producing (part of) this table?*
- *Argue, based on this table, that multiplication in $\mathbb{Z}_7$ is commutative.*
- *Using this table, compute 3^5. Explain the steps, and explain why this is better than computing $3^5 \mod 7$.*
- *Describe an algorithm, that performs exponentiation in $\mathbb{Z}_p$.*
- *What is the complexity of this algorithm?*

Exercises

Exercise

RSA encryption:

1. *Perform RSA encryption of the message 4, using the public key $(5, 91)$.*
2. *In order to, crack the public key $(5, 91)$, which concrete computation do you have to perform.*
3. *Use the cracked private key to decrypt the message produced in (1).*

Summary

- We can distinguish between
  - **stream ciphers**, that encrypt character-by-character, and
  - **block ciphers**, that encrypt block-wise.
- For block ciphers, we can distinguish between
  - **symmetric (private-key)** encryption, that uses the same key for en- and de-cryption, and
  - **asymmetric (public-key)** encryption, that uses separate keys for en- and de-cryption.
- Private-key encryption, e.g. DES or AES, is typically faster.
- Public-key encryption, e.g. RSA or ElGamal, is easier to manage.
- In practice, a combination of both is used for secure network communication:
  - Public-key encryption is used to encode a *session key*.
  - Private-key encryption is used with this session to encrypt the data.

References


Michael T. Goodrich and Roberto Tamassia,
"Introduction to Computer Security",
Addison Wesley, 2011. ISBN: 0-32-151294-4.
Chapter 2: Cryptography.


Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone,
"Handbook of Applied Cryptography",
CRC Press, 2001. ISBN 0-8493-8523-7.
Chapter 7: Block Ciphers, Chapter 8: Public-Key Encryption


Nigel Smart,
"Cryptography: An Introduction".
On-line Version:
`http://www.cs.bris.ac.uk/~nigel/Crypto_Book/`