Algebra & Cryptography CSE3230

Lecture 9 – Stream Ciphers

Stjepan Picek

This Lecture

- Review of Basic Concepts in Cryptography
- Symmetric key Cryptography – Stream ciphers
  Vernam Cipher
  Linear Feedback Shift Registers
  Stream ciphers models
  Cryptographic properties of Boolean functions
  The TRIVIUM stream cipher

References and reading material

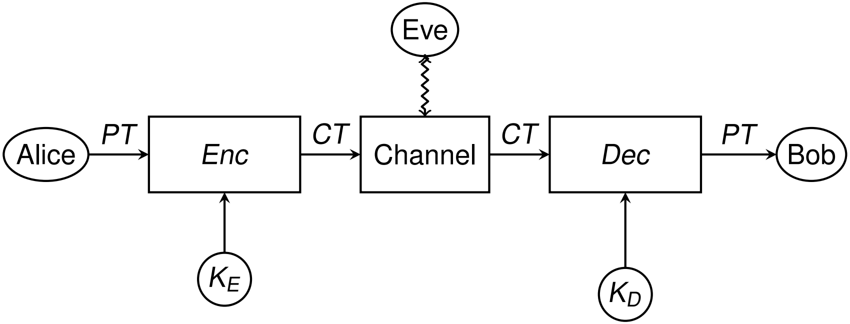
- Carlet, C.: Boolean functions for cryptography and error-correcting codes. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 257–397. Cambridge University Press, 2010 (available at: `https://www.math.univ-paris13.fr/~carlet/chap-fcts-Bool-corr.pdf`)
- Further reading: Stinson, D. Paterson, M.: Cryptography: theory and practice (4th ed.). CRC press, 2018

This Lecture

- Review of Basic Concepts in Cryptography

- Symmetric key Cryptography – Stream ciphers
  - Vernam Cipher
  - Linear Feedback Shift Registers
  - Stream ciphers models
  - Cryptographic properties of Boolean functions
  - The TRIVIUM stream cipher

Cryptosystem – Block scheme

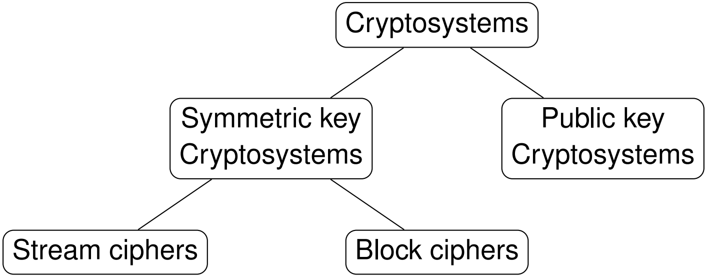


- PT: plaintext
- Enc: encryption function
- K_E: encryption key
- CT: ciphertext
- Dec: decryption function
- K_D: decryption key

Cryptosystem – Basic properties

- *Unique decodability*: given K_E and K_D, for all plaintext PT it must be the case that $Dec_{K_D}(Enc_{K_E}(PT)) = PT$
- For all PT, it must be *easy* for Alice to compute the ciphertext $CT = Enc_{K_E}(PT)$ by knowing K_E and Enc
- For all CT, it must be *easy* for Bob to recover the plaintext $PT = Dec_{K_D}(CT)$ by knowing K_D and Dec
- Given CT, it must be *extremely difficult* for Eve to recover PT *without* knowing the decryption key K_D
- It is always assumed that the encryption and decryption functions are known to Eve (*Kerchoff's principle*)

Classification of Cryptosystems



Symmetric and Public Key Cryptosystems

Symmetric key cryptosystems:

- The key used for encryption and decryption is the same
- Alice and Bob must agree on this key before the communication takes place
- Can be further classified in:
  - *Stream ciphers*: Encryption and decryption process single symbols of the plaintext and the ciphertext
  - *Block ciphers*: Encryption and decryption work over *blocks* of fixed length of symbols

Public key (or *asymmetric key*) cryptosystems:

- The keys used for encryption and decryption differ
- Alice uses Bob's *public key* K_E to encrypt, while Bob uses his own *private key* K_D to decrypt

This Lecture

- Review of Basic Concepts in Cryptography

- Symmetric key Cryptography – Stream ciphers
  Vernam Cipher
  Linear Feedback Shift Registers
  Stream ciphers models
  Cryptographic properties of Boolean functions
  The TRIVIUM stream cipher

This Lecture

- Review of Basic Concepts in Cryptography
- Symmetric key Cryptography – Stream ciphers
  - Vernam Cipher
  - Linear Feedback Shift Registers
  - Stream ciphers models
  - Cryptographic properties of Boolean functions
  - The TRIVIUM stream cipher

Vernam Cipher

- Invented by Gilbert Vernam in 1917
- The plaintext x is a string (or vector) over the binary alphabet $\Sigma = \mathbb{F}_2 = \{0,1\}$, that is $x = (x_1, x_2, \cdots, x_n) \in \mathbb{F}_2^n$
- The symmetric key k is a binary string *of the same length* of x, that is $k = (k_1, k_2, \cdots, k_n) \in \mathbb{F}_2^n$
- The ciphertext y is the *bitwise XOR* (i.e., addition modulo 2) of the plaintext x and the key k:

$$y = x \oplus k = (x_1 \oplus k_1, x_2 \oplus k_2, \cdots, x_n \oplus k_n)$$

- Since $a \oplus a = 0$ for all $a \in \mathbb{F}_2$, decryption is symmetrical:

$$x = y \oplus k = (y_1 \oplus k_1, y_2 \oplus k_2, \cdots, y_n \oplus k_n)$$

Vernam Cipher – Block Scheme

$$\begin{tikzcd}
& k \arrow[d] & \\
x \arrow[r] & \oplus \arrow[r] & y
\end{tikzcd}$$

(a) Encryption

$$\begin{tikzcd}
& k \arrow[d] & \\
y \arrow[r] & \oplus \arrow[r] & x
\end{tikzcd}$$

(b) Decryption

- $x \in \mathbb{F}_2^n$: plaintext
- $y \in \mathbb{F}_2^n$: ciphertext
- $k \in \mathbb{F}_2^n$: symmetric key
- $\bigoplus$: bitwise XOR

- The Vernam cipher is also called *One-Time Pad* (OTP): each plaintext is encrypted with a *different* key

Security of the Vernam Cipher

- The OTP can be proved to be unconditionally secure under known ciphertext attacks
- This property is also called *perfect secrecy*
- Assume that probability distributions $Pr[X = x]$ and $Pr[Y = y]$ are defined on $\mathcal{P}$ and $\mathcal{C}$. Then:

Definition

A cryptosystem $(\mathcal{P},\mathcal{C},\mathcal{K},\mathcal{E},\mathcal{D})$ has *perfect secrecy* if for any plaintext $x \in \mathcal{P}$ and ciphertext $y \in \mathcal{C}$

$$Pr[x|y] = Pr[x]$$

- In other words, knowing the ciphertext y gives *no information* on the plaintext x

Perfect Secrecy of OTP

Theorem (Shannon, 1949)

Let $(\mathcal{P},\mathcal{C},\mathcal{K},\mathcal{E},\mathcal{D})$ be a cryptosystem where $|\mathcal{K}| = |\mathcal{C}| = |\mathcal{P}|$. Then the cryptosystem has perfect secrecy if and only if:

- *Every key is used with equal probability $\frac{1}{|\mathcal{K}|}$*
- *For all $x \in \mathcal{P}$ and $y \in \mathcal{C}$, there is a unique key $k \in \mathcal{K}$ such that $Enc_k(x) = y$*

- The OTP satisfies the conditions of Shannon's theorem
- If Eve only knows the ciphertext y, each plaintext x is equally likely
- Hence, the OTP is KCA-unconditionally secure

Perfect Secrecy and Limitations of OTP

The OTP is unconditionally secure under known ciphertext attacks, but it suffers from three major drawbacks, namely:

- The key must be as long as the plaintext
- Keys cannot be reused (hence the name "One-time")
- It is unconditionally secure only against a known ciphertext attack, which is the weakest attack model

These drawbacks severely limits the use of the OTP in practice

Practical Stream Ciphers

- The biggest issue in the OTP is the length of the key
- If Alice and Bob already have a secure way to share keys as long as the plaintexts, then why don't they use it directly to transmit plaintexts?
- In practical stream ciphers, the key is significantly shorter than the plaintext
- By Shannon's theorem, such stream ciphers cannot be unconditionally secure
- Usually, the security level considered is *computational security*

Keystreams and Pseudorandom Generators

- Idea: stretch the short secret key into a *keystream* that is as long as the plaintext
- A *Pseudorandom Generator* (PRG) is usually employed to produce the keystream
- A PRG is a deterministic algorithm that takes in input a short random string and outputs a long pseudorandom string
- "Pseudorandom" means that the string *looks like* a random string to an adversary with bounded computational resources

Synchronous Stream Ciphers

Definition

A synchronous stream cipher is a 6-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{E}, \mathcal{D})$, together with a PRG g, such that:

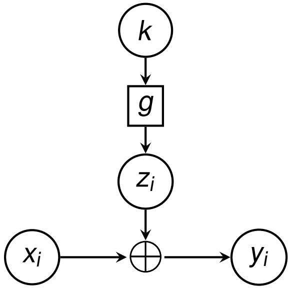
- $\mathcal{P}$ is a finite set of plaintexts symbols
- $\mathcal{C}$ is a finite set of ciphertexts symbols
- $\mathcal{K}$ is a finite set of possible keys
- $\mathcal{L}$ is a finite set called the keystream alphabet
- g is takes in input $k \in \mathcal{K}$, and generates an infinite keystream $z_1 z_2 \cdots$, where $z_i \in \mathcal{L}$ for all $i \in \mathbb{N}$.
- For each $z \in \mathcal{L}$, there is an encryption function $Enc_z \in \mathcal{E}$ and a decryption function $Dec_z \in \mathcal{D}$ such that $Dec_z(Enc_z(x)) = x$ for every plaintext symbol $x \in \mathcal{P}$

Vernam-like Stream Ciphers

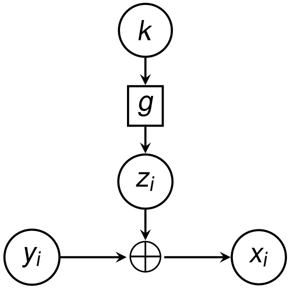
In a Vernam-like stream cipher, the protocol is the following:

1. Alice and Bob agree on a short secret key k
2. for each plaintext bit $x_i \in \mathbb{F}_2$:
   - Alice:
     - 2.1 Computes the next keystream bit z_i with the PRG g, giving in input the key k
     - 2.2 Computes the next ciphertext bit $y_i = x_i \oplus z_i$
     - 2.3 Sends y_i over the channel
   - Bob:
     - 2.1 Receives the ciphertext bit y_i
     - 2.2 Computes the next keystream bit z_i with the PRG g, giving in input the key k
     - 2.3 Recovers the next plaintext bit $x_i = y_i \oplus z_i$

Vernam-like Stream Cipher – Block Scheme



(a) Encryption



(b) Decryption

- k: secret key
- g: pseudorandom generator
- $z_i \in \mathbb{F}_2$: next keystream bit
- $\bigoplus$: XOR operator
- $x_i \in \mathbb{F}_2$: next plaintext bit
- $y_i \in \mathbb{F}_2$: next ciphertext bit

This Lecture

- Review of Basic Concepts in Cryptography
- Symmetric key Cryptography – Stream ciphers
  - Vernam Cipher
  - Linear Feedback Shift Registers
  - Stream ciphers models
  - Cryptographic properties of Boolean functions
  - The TRIVIUM stream cipher

Keystreams with Linear Recurrences (LR)

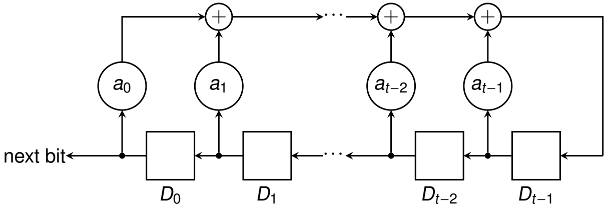
- Idea for keystream generator: use a *linear recurrence*
- Given $a_0, a_1, \cdots, a_{t-1} \in \mathbb{F}_2$, and $z_0, z_1, \cdots, z_{t-1} \in \mathbb{F}_2$, a LR of order t is defined for all $i \geq t$ as:

$$z_i = a_0 \cdot z_{i-t} \oplus a_1 \cdot z_{i-t+1} \oplus \cdots \oplus a_{t-1} \cdot z_{i-1}$$

  where $\cdot$ indicates multiplication over $\mathbb{F}_2$ (i.e., logical AND)
- The initial t values $z_0, z_1, \cdots, z_{t-1}$ are the bits of the secret key
- The next keystream bit z_i is computed with the LR equation

Linear Feedback Shift Registers (LFSR)

- LR can be efficiently implemented in hardware with *Linear Feedback Shift Registers* (LFSR)
- A LFSR is composed of t flip-flops $D_0, \cdots, D_{t-1}$, holding the t previous terms in the sequence



- At each clock cycle, the flip-flops shift their current value
- The output of D_0 is taken as the next keystream bit
- D_{t-1} is updated with the result of the LR

LFSR Properties

- Since the LFSR is described by a register of t bits, after at most 2^t cycles the keystream bits will repeat
- If $z_0 = z_1 = \cdots = z_{t-1} = 0$ the LFSR always outputs 0
- Hence, the period of a LFSR can be at most $2^t - 1$
- The *characteristic polynomial* of a LFSR of order t defined by the coefficients $a_0, a_1, \cdots, a_{t-1}$ is

$$a_0 + a_1 X + a_2 X^2 + \cdots + a_{t-1} X^{t-1} + X^t \in \mathbb{F}_2[X]$$

Theorem

A LFSR of order t has period $2^t - 1$ if and only if its characteristic polynomial is primitive (i.e. it is the minimal polynomial of a primitive element of $\mathbb{F}_{2^t}$)

LFSR Weaknesses

- A LFSR keystream generator of order t can be totally broken with a known-plaintext attack
- Suppose that Eve has a plaintext $(x_1, x_2, \cdots, x_n)$ and the corresponding ciphertext $(y_1, y_2, \cdots, y_n)$. The keystream is:

$$(z_1, z_2, \cdots, z_n) = (x_1 \oplus y_1, x_2 \oplus y_2, \cdots, x_n \oplus y_n)$$

- If $n \geq 2t$, Eve can recover the LFSR coefficients $a_0, \cdots, a_{t-1}$ by solving a system of t linear equations in t unknowns

$$(a_0, a_1, \cdots, a_{t-1}) = (z_{t+1}, z_{t+2}, \cdots, z_{2t}) \cdot \begin{pmatrix} z_1 & z_2 & \cdots & z_t \\ \vdots & \vdots & \ddots & \vdots \\ z_t & z_{t+1} & \cdots & z_{2t-1} \end{pmatrix}^{-1}$$

- With $a_0, a_1, \cdots, a_{t-1}$ Eve can reconstruct the whole keystream

This Lecture

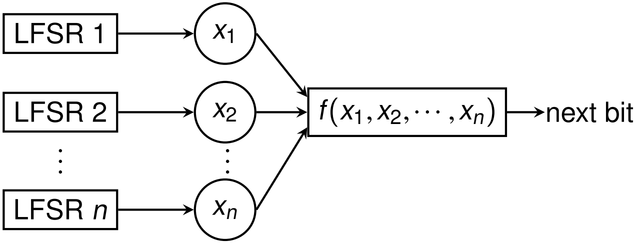
- Review of Basic Concepts in Cryptography
- Symmetric key Cryptography – Stream ciphers
  - Vernam Cipher
  - Linear Feedback Shift Registers
  - Stream ciphers models
  - Cryptographic properties of Boolean functions
  - The TRIVIUM stream cipher

Keystream generators models

- Keystream generators still use LFSR as components, due to their maximal periods and efficient hardware implementation
- The LFSR weaknesses are addressed by adding some nonlinear components, typically *Boolean functions*
- Common keystream generators models include:
  - The *Combiner Generator*
  - The *Filter Generator*

The Combiner Generator

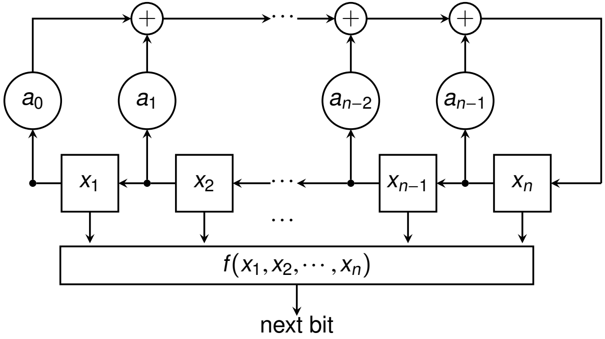
- Idea: use n LFSR in parallel, and combine their outputs with a *Boolean function* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ of n variables



- The period of the combiner is at most the least common multiple of the periods of the n LFSR
- The combiner function f must satisfy several properties to resist different types of attacks

The Filter Generator

- Idea: single LFSR of order n, but use the values of *all* flip-flops as inputs to a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$



- Equivalent to the combiner model with n copies of the same LFSR, but attacks work differently on the filter generator

This Lecture

- Review of Basic Concepts in Cryptography
- Symmetric key Cryptography – Stream ciphers
  - Vernam Cipher
  - Linear Feedback Shift Registers
  - Stream ciphers models
  - Cryptographic properties of Boolean functions
  - The TRIVIUM stream cipher

Boolean Functions in Stream Ciphers

- The security analysis of a stream cipher model can often be reduced to the study of the cryptographic properties of the involved Boolean functions

Some notation used in the following:

- $\mathbb{F}_2^n$: the set of *n*-tuples of elements over the finite field $\mathbb{F}_2$, endowed with a vector space structure
- Vector sum is bitwise XOR, denoted as $\oplus$
- Given two vectors $x, y \in \mathbb{F}_2^n$, their *scalar product* is:

$$x \cdot y = x_1 y_1 \oplus x_2 y_2 \oplus \cdots \oplus x_n y_n$$

- The *Hamming distance* of $x, y \in \mathbb{F}_2^n$ is the number of coordinates where they differ

Boolean Functions - Basic Definitions

- A *Boolean function* of n variables is a mapping $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$, most commonly represented by its *truth table*
- The truth table of f is the 2^n-bit vector Ω_f specifying the value of f for each input vector in lexicographic order:

$$\Omega_f = (f(0,\cdots,0,0),\ f(0,\cdots,0,1),\ \cdots,\ f(1,\cdots,1,1))$$

- Example with $n = 3$ variables:

(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
Ω_f	0	1	1	0	1	0	1	0

- *Support* of f: $supp(f) = \{x \in \mathbb{F}_2^n : f(x) \neq 0\}$
- *Hamming weight* of f: $HW(f) = |supp(f)|$

Algebraic Normal Form (ANF)

- Represents $f : \mathbb{F}_2^n \to \mathbb{F}_2$ as a multivariate polynomial in $\mathbb{F}_2[x_1, \cdots, x_n]/(x_1^2 \oplus x_1, \cdots, x_n^2 \oplus x_n)$:

$$f(x_1, \cdots, x_n) = \bigoplus_{a \in \mathbb{F}_2^n} h(a) \cdot \left(\prod_{i=1}^n x_i^{a_i}\right)$$

  where $h(a) = \bigoplus_{x \leq a} f(x)$ (*Möbius inversion*)
- Every coordinate x_i appears in this polynomial with exponents at most 1, because every bit in $\mathbb{F}_2$ equals its own square
- As such, since the inputs of the function take their values in $\mathbb{F}_2$, they must be considered modulo $X^2 + X$.
- It follows that any monomial of this polynomial is the product of some input variables.

Algebraic Normal Form (ANF)

- Each monomial can then be characterized by a subset of $I = \{1, ..., n\}$, i.e.,$\prod_{i \in I} x_i$, or equivalently by the n-bit vector u having I as support.
- The monomial $\prod_{i=1}^{n} x_i^{a_i}$ is often denoted as $x^u, \forall u \in \mathbb{F}_2^n$
- As such, we write $\mathbb{F}_2(x_1, \cdots, x_n) = \bigoplus_{u \in \mathbb{F}_2^n} a_u x^u$, with $a_u \in \mathbb{F}_2$
- $a_u = \bigoplus_{x \leq u} f(x)$ and $f(u) = \bigoplus_{x \leq u} a_x$
- $x \leq y$ if and only if $x_i \leq y_i, \forall 1 \leq i \leq n$
- Example with $n = 3$ variables:

(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
Ω_f	0	1	1	0	1	0	1	0
$h(a)$	0	1	1	0	1	0	1	0

$$\Downarrow$$

$$f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_1 x_2 \oplus x_3$$

Walsh-Hadamard Transform (WHT)

- Represents $f : \mathbb{F}_2^n \to \mathbb{F}_2$ in terms of *correlations* with the *linear* functions $a \cdot x$, for $a \in \mathbb{F}_2^n$:

$$W_f(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus a \cdot x}$$

- The WHT is useful to characterize several cryptographic properties of f
- Note that here, $\cdot$ represents the inner (scalar) product: $a \cdot b = (a_1 \ AND \ b_1) \ XOR \ (a_2 \ AND \ b_2) \ XOR \ \ldots XOR \ (a_n \ AND \ b_n)$.
- Example with $n = 3$ variables:

(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
Ω_f	0	1	1	0	1	0	1	0
$W_f(a)$	0	-4	0	4	0	4	0	4

Walsh-Hadamard Transform (WHT)

- Continuing with the previous example
- $W_f(0) = (-1)^{0 \oplus 0 \cdot 0} + (-1)^{1 \oplus 0 \cdot 1} + (-1)^{1 \oplus 0 \cdot 2} + (-1)^{0 \oplus 0 \cdot 3} + (-1)^{1 \oplus 0 \cdot 4} + (-1)^{0 \oplus 0 \cdot 5} + (-1)^{1 \oplus 0 \cdot 6} + (-1)^{0 \oplus 0 \cdot 7} =$
- $= (-1)^0 + (-1)^1 + (-1)^1 + (-1)^0 + (-1)^1 + (-1)^0 + (-1)^1 + (-1)^0 = 0$
- $W_f(1) = (-1)^{0 \oplus 1 \cdot 0} + (-1)^{1 \oplus 1 \cdot 1} + (-1)^{1 \oplus 1 \cdot 2} + (-1)^{0 \oplus 1 \cdot 3} + (-1)^{1 \oplus 1 \cdot 4} + (-1)^{0 \oplus 1 \cdot 5} + (-1)^{1 \oplus 1 \cdot 6} + (-1)^{0 \oplus 1 \cdot 7}$
- $= (-1)^0 + (-1)^0 + (-1)^1 + (-1)^1 + (-1)^1 + (-1)^1 + (-1)^1 + (-1)^1 = -4$
- …

Cryptographic Properties: Balancedness

- A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is *balanced* if its truth table Ω_f has the same number of 0 and 1, i.e. if $HW(\Omega_f) = 2^{n-1}$
- Walsh characterization: f is balanced $\Leftrightarrow W_f(0) = 0$
- Example with $n = 3$ variables:

(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
Ω_f	0	1	1	0	1	0	1	0
$W_f(a)$	0	-4	0	4	0	4	0	4

$$\Downarrow$$

f is balanced

- Unbalanced functions present a statistical bias that can be exploited in distinguishing attacks

Cryptographic Properties: Algebraic Degree

- The *algebraic degree* d of $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ is the degree of the multivariate polynomial representing the ANF of f

$$f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_2 x_3 \oplus x_3$$

$$\Downarrow$$

$$f \text{ has degree } d = 2$$

- *Affine functions* have degree 1, and correspond to linear functions $a \cdot x$ summed with a constant $b \in \mathbb{F}_2$
- Boolean functions of high degree make the attacks based on the *Berlekamp-Massey algorithm* less effective on the combiner model

Cryptographic Properties: Nonlinearity

- The *nonlinearity* of $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is the minimum Hamming distance of f from the set of all affine functions
- Walsh characterization:

$$nl(f) = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} \left\{ \left| W_f(a) \right| \right\}$$

(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
Ω_f	0	1	1	0	1	0	1	0
$W_f(a)$	0	-4	0	4	0	4	0	4

$$\Downarrow$$

$$nl(f) = 2^{3-1} - \frac{1}{2} \cdot 4 = 2$$

- Functions with low nonlinearity are vulnerable to *fast-correlation attacks* in the combiner model

Parseval's Relation

- For every function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, *Parseval's relation* holds:

$$\sum_{a \in \mathbb{F}_2^n} [W_f(a)]^2 = 2^{2n}$$

- To have high nonlinearity, the maximum absolute value of $W_f(a)$ must be as low as possible
- By Parseval's relation, this happens when 2^{2n} is spread uniformly over all 2^n WHT coefficients, that is:

$$W_f(a)^2 = \frac{2^{2n}}{2^n} = 2^n \Rightarrow W_f(a) = \pm 2^{\frac{n}{2}}$$

Covering Radius Bound and Bent Functions

- Thus, the nonlinearity of f is bounded above by

$$nl(f) \leq 2^{n-1} - 2^{\frac{n}{2}-1}$$

  which is also called the *covering radius bound* (CRB), due to its connection with Reed-Muller codes
- Functions satisfying the CRB are also called *bent*, and exist only when n is even
- Bent function has the highest nonlinearity possible, but they are unbalanced (in fact, $W_f(0) = \pm 2^{\frac{n}{2}} \neq 0$)
- As such, bent functions cannot be used directly in the design of stream ciphers

Cryptographic Properties: Correlation Immunity

- Function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is *t-correlation immune* if, by fixing any i variables with $1 \leq i \leq t$, the resulting restrictions all have the same Hamming weight
- WHT characterization: $W_f(a) = 0$ for all a s.t. $1 \leq HW(a) \leq t$

(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
Ω_f	0	1	1	0	1	0	1	0
$W_f(a)$	0	−4	0	4	0	4	0	4

$$\Downarrow$$

$$W_f(001) = -4 \Rightarrow f \text{ is NOT 1-correlation immune}$$

- In the combiner model, correlation immune functions of high order t can resist to *correlation attacks*

Cryptographic Properties: Algebraic Immunity

- Given $f, g : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$, let $f.g$ denote the *Hadamard product* of f and g, that is:

$$f.g(x) = f(x)g(x) \text{ for all } x \in \mathbb{F}_2^n$$

- g is an *annihilator* of f is $f.g \equiv \underline{0}$ is the constant function $\underline{0}$
- The *algebraic immunity* $AI(f)$ of f is the smallest algebraic degree among all annihilators of f
- In stream ciphers, the algebraic immunity of the involved Boolean functions f must be as high as possible to avoid *algebraic attacks*

Bounds and Trade-offs

In summary, $f : \mathbb{F}_2^n \to \mathbb{F}_2$ in a keystream generator model should:

- be balanced
- have high algebraic degree d
- have high nonlinearity nl
- be correlation immune of high order t
- have high algebraic immunity $AI(f)$

Most of these properties cannot be satisfied simultaneously:

- *Covering Radius bound*: $nl \leq 2^{n-1} - 2^{\frac{n}{2}-1}$
- *Siegenthaler's bound*: $d \leq n - t$ ($d \leq n - t - 1$ if f is balanced)
- *Sarkar's et al. bound*: $nl \leq 2^{n-1} - 2^{t+1}$
- *Algebraic Immunity bound*: $AI(f) \leq \left\lceil \frac{d}{2} \right\rceil$

Constructions of good Boolean Functions

- Number of Boolean functions of n variables: 2^{2^n}
- $\Rightarrow$ too huge for exhaustive search when $n > 5$!

In practice, one can resort to *algebraic constructions*, which are classified in:

- *Primary constructions*, where Boolean functions with certain properties are built from scratch
- *Secondary constructions*, where new Boolean functions are obtained from existing ones

Primary and Secondary Constructions

Example of primary construction: the *Maiorana-McFarland* class

- Given n even, let $\mathbb{F}_2^n = \left\{(x, y) : x, y \in \mathbb{F}_2^{\frac{n}{2}}\right\}$
- The function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ defined as

$$f(x, y) = x \cdot \pi(y) \oplus g(y)$$

where $\pi : \mathbb{F}_2^{\frac{n}{2}} \to \mathbb{F}_2^{\frac{n}{2}}$ is any permutation of $\mathbb{F}_2^{\frac{n}{2}}$ and $g : \mathbb{F}_2^{\frac{n}{2}} \to \mathbb{F}_2$ is any function of $n/2$ variables is bent

Example of secondary construction: *Rothaus's* construction

- If g, h, k and $g \oplus h \oplus k$ are bent on $\mathbb{F}_2^n$, then the following function is bent:

$$\begin{aligned} f(x_1, x_2, x) = {} & g(x)h(x) \oplus g(x)k(x) \oplus h(x)k(x) \oplus \\ & \oplus [g(x) \oplus h(x)]x_1 \oplus [g(x) \oplus k(x)]x_2 \oplus x_1 x_2 \end{aligned}$$

where $(x_1, x_2, x) \in \mathbb{F}_2^{n+2}$ with $x_1, x_2 \in \mathbb{F}_2$, $x \in \mathbb{F}_2^n$

This Lecture

- Review of Basic Concepts in Cryptography
- Symmetric key Cryptography – Stream ciphers

  Vernam Cipher

  Linear Feedback Shift Registers

  Stream ciphers models

  Cryptographic properties of Boolean functions

  The TRIVIUM stream cipher

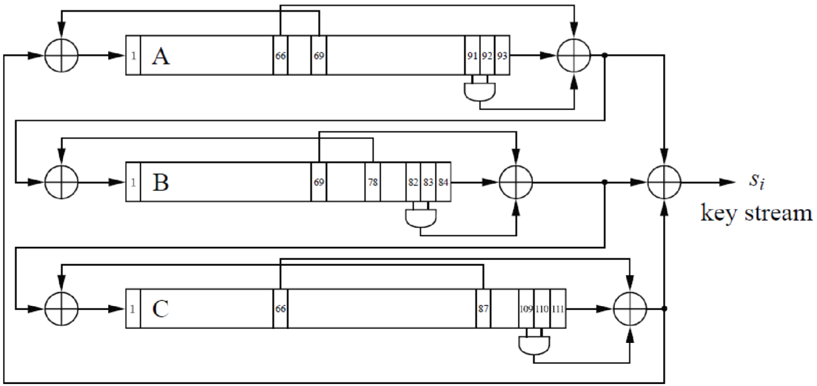
Concrete Stream Ciphers

- The keystream generators viewed so far are theoretical models
- They are useful to formalize and study attacks
- Concrete stream ciphers use more complex designs
- Examples include the finalists of eSTREAM, a competition completed in 2008 to select stream ciphers for two different profiles:
  - Stream ciphers for software applications with high throughput requirements
  - Stream ciphers for hardware applications with restricted resources such as limited storage, gate count, or power consumption.

Trivium

- TRIVIUM is an eSTREAM finalist, selected for the hardware profile
- It uses an 80-bit key and it is based on a combination of three shift registers.
- Even though these are linear feedback shift registers, there are nonlinear components used to derive the output of each register, unlike the LFSRs that we studied in the previous section

Block Scheme



Description

- Three shift registers, A, B and C
- The lengths of the registers are 93, 84 and 111
- The XOR-sum of all three register outputs forms the keystream bit s_i
- The output of each register is connected to the input of another register
- Thus, the registers are arranged in circle-like fashion
- The cipher can be viewed as consisting of one circular register with a total length of $93 + 84 + 111 = 288$

Description

- The input of each register is computed as XOR of two bits:
  1. The output of register A is part of the input of register B
  2. One register bit at a specific location is fed back to the input.
- The output of each register is computed as the XOR-sum of three bits:
  1. The rightmost register bit
  2. One register bit at a specific location is fed forward to the output
  3. The output of a logical AND function whose input is two specific register bits

Security

- Note that the AND operation is equal to multiplication in modulo 2 arithmetic
- If we multiply two unknowns, and the register contents are the unknowns that an attacker wants to recover, the resulting equations are no longer linear as they contain products of two unknowns
- Thus, the feedforward paths involving the AND operation are crucial for the security of Trivium as they prevent attacks that exploit the linearity of the cipher

Encryption

- Two input parameters: a key k and an initialization vector IV
- Initially, an 80-bit IV is loaded into the 80 leftmost locations of register A, and an 80-bit key is loaded in the 80 leftmost locations of register B
- All other register bits are set to zero with the exception of the three rightmost bits of register C, i.e., bits c_{109}, c_{110} and c_{111}, which are set to 1
- In the first phase, the cipher is clocked $4 \cdot 288 = 1152$ times. No cipher output is generated
- The bits produced hereafter, i.e., starting with the output bit of cycle 1153, form the keystream

Algebra & Cryptography CSE3230

Lecture 10 – Block Ciphers and S-Boxes

Stjepan Picek

Recap – Previous Lecture

- Review of Basic Concepts in Cryptography
- Vernam-like stream ciphers
- Linear feedback shift registers
- Combiner and Filter models for stream ciphers
- Cryptographic properties of Boolean Functions
- The TRIVIUM stream cipher

Takeaway: The design of a stream cipher requires to take into account several trade-offs of conflicting security properties, especially concerning the involved Boolean functions

This Lecture

- Basic concepts of Block Ciphers
- S-Boxes
  - Representations of S-boxes
  - Cryptographic Properties of S-Boxes

References and reading material

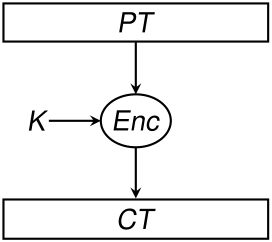
- Carlet, C.: Vectorial Boolean Functions for Cryptography. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methodsin Mathematics, Computer Science, and Engineering, pp. 257–397. Cambridge University Press, 2010 (available at: `https://www.math.univ-paris13.fr/~carlet/chap-vectorial-fcts-corr.pdf`)
- Further reading: Stinson, D. Paterson, M.: Cryptography: theory and practice (4th ed.). CRC press, 2018

This Lecture

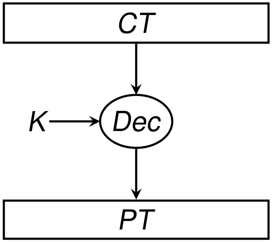
- Basic concepts of Block Ciphers
- S-Boxes
  - Representations of S-boxes
  - Cryptographic Properties of S-Boxes

Block Ciphers

- Recall: stream ciphers encrypt one plaintext symbol at a time
- *Block ciphers*, instead, directly encrypt fixed-size *blocks* of plaintext, and generate ciphertext blocks of the same length



(a) Encryption



(b) Decryption

- Longer messages are encrypted one block at a time

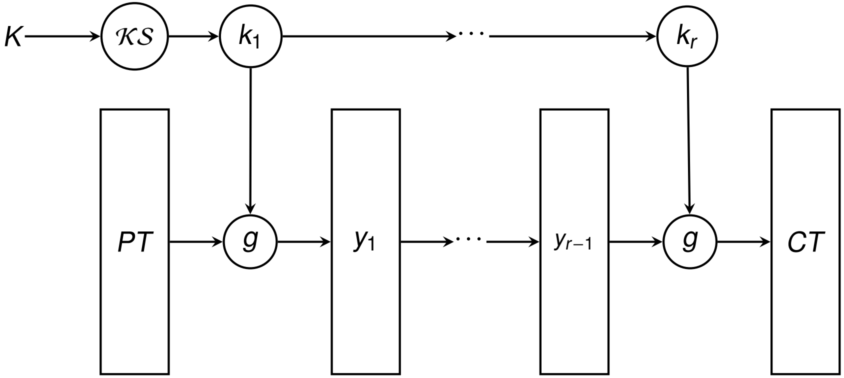
Confusion and Diffusion

- Design principles formulated by Shannon in 1949 for frustrating statistical attacks in symmetric cryptosystems
- *Confusion*: The relation between the ciphertext *CT* and the encryption key *K* should be as complex as possible
- *Diffusion*: The statistical structure of the plaintext *PT* should be "spread" over the ciphertext
- Thus, the encryption function should behave as a *mixing transformation* to achieve these two properties

Iterated Ciphers

- Common approach to design block ciphers encryption functions that result in mixing transformations
- Two main components:
  - *Key scheduling algorithm*: starting from the secret key K, derives R *round keys* $k_1, \cdots, k_r$
  - *Round function*: encrypts a block using a round key
- The round function is iteratively applied for r *rounds*. At the first round $i = 1$, the input block is the plaintext
- In the subsequent rounds $1 < i \leq r$, the round function encrypts the result of the previous round $i - 1$ using the key k_i
- The result of the round function at round r is the ciphertext

Iterated Ciphers – Block Scheme



- PT: plaintext
- CT: ciphertext
- K: symmetric key
- $\mathcal{KS}$: key scheduling algorithm
- g: round function
- k_i: round keys

Iterated Block Ciphers – Formal Definition

Definition

An *iterated block cipher* of *block size* b, key size l and r *rounds* is a cryptosystem $\langle \mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D} \rangle$ where:

- $\mathcal{P} = \mathcal{C} = \{0,1\}^b$, and $\mathcal{K} = \{0,1\}^l$
- Each encryption function $Enc_K \in \mathcal{E}$ is defined by a *key scheduling function* $KS : \{0,1\}^l \to \{0,1\}^l$ and by a *round function* $g : \{0,1\}^{b+l} \to \{0,1\}^b$ for all plaintext $x \in \{0,1\}^b$ and key $k \in \{0,1\}^l$ as follows:

$$Enc_K(x) = g^r(x, K) \enspace ,$$

where for all rounds $i \in \{1, \cdots, r\}$ the function g^i is defined as:

$$g^i(x,K) = \begin{cases} g(x, KS(K)) & \text{, if i=1} \\ g(g^{i-1}(x,K), KS(KS^{i-1}(K)) & \text{, if i>1} \end{cases}$$

Iterated Block Ciphers – Formal Definition

Definition

- Denoted as $h = g^{-1}$ the inverse round function, each decryption function $Dec_k \in \mathcal{D}$ is defined symmetrically for all key $K \in \{0,1\}^l$ and ciphertext $y \in \{0,1\}^b$ as follows:

$$Dec_K(y) = h^r(y, K) \ ,$$

with h^i being defined for all $i \in \{1, \cdots, r\}$ as:

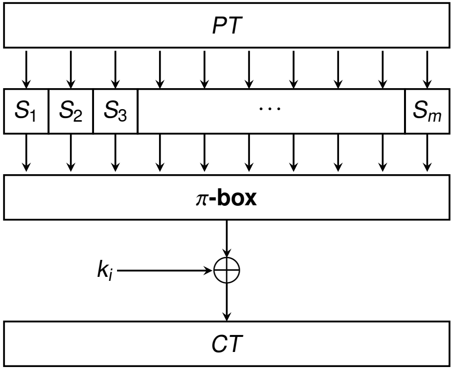
$$h^i(y,K) = \begin{cases} h(y, KS^r(K)) & \text{, if i=1} \\ h(h^{i-1}(y,K), KS^{r-i+1}(K))) & \text{, if i>1} \end{cases}$$

Substitution-Permutation Networks (SPN)

- One of the most common approaches to design the round function of an iterated cipher
- The input block is encrypted by a *confusion layer* followed by a *diffusion layer*
- The confusion layer is made of *m Substitution Boxes* (S-boxes) $S_i : \mathbb{F}_2^n \to \mathbb{F}_2^n$ applied in parallel, where $mn = b$. Each S-box encrypts a sub-block of n bits
- The diffusion layer is a *Permutation Box* (P-box) $\pi : \mathbb{F}_2^b \to \mathbb{F}_2^b$, which transposes the bits in the whole block
- The result is finally bitwise XORed with the round key k_i (we assume $l = b$, thus $\mathcal{P} = \mathcal{C} = \mathcal{K} = \{0,1\}^b$)

SPN Block Scheme

Round function of a SPN cipher:



- The P-box π and all S-boxes S_i must be *invertible* to ensure that decryption is possible

Feistel Networks (FN)

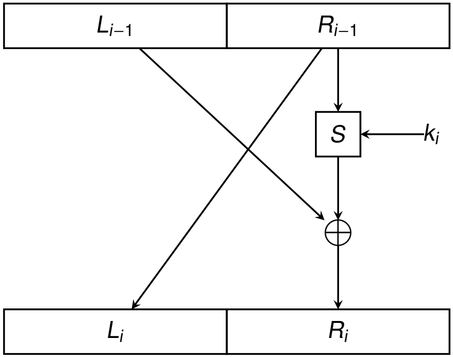
- Alternative approach to design the round function, invented by Horn Feistel for the *Lucifer* cipher in the '70s
- Given $b = 2t$, the plaintext $x \in \{0,1\}^b$ is divided in a left block $L \in \{0,1\}^t$ and a right block $R \in \{0,1\}^t$
- Given the round key k_i, the left and right block $L_i, R_i \in \{0,1\}^t$ for round i are defined as:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus S(R_{i-1}, k_i) \end{aligned}$$

where $S : \{0,1\}^{t+l} \rightarrow \{0,1\}^t$ is an *S-box* combining the previous right block with the round key

FN Block Scheme

Round function of a FN cipher:



- Remark that the S-box S does not need to be invertible for decryption, because of the XOR operation

This Lecture

Security of Block ciphers and S-boxes

- Similarly to Boolean functions in stream ciphers, several attacks in SPN and FN ciphers can be formalized on their underlying S-boxes
- To thwart these attacks, the involved S-boxes must satisfy several *cryptographic properties*
- Recall of notation (see extension fields in Lecture 6):
  - $\mathbb{F}_2^n$: the vector space of dimension n over the finite field with two elements $\mathbb{F}_2 = \{0,1\}$
  - Given two vectors $x, y \in \mathbb{F}_2^n$, their *scalar product* is:

$$x \cdot y = \bigoplus_{i=1}^{n} x_i y_i = x_1 y_1 \oplus x_2 y_2 \oplus \cdots \oplus x_n y_n$$

  - $\mathbb{F}_{2^n}$: the finite field with 2^n elements (recall that $\mathbb{F}_{2^n}$ is isomorphic to $\mathbb{F}_2^n$)

This Lecture

- Basic concepts of Block Ciphers
- S-Boxes
  - Representations of S-boxes
  - Cryptographic Properties of S-Boxes

S-Boxes: General definitions

- Let $n, m \in \mathbb{N}$. A *Substitution Box* (S-box) is a *vectorial Boolean function* $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, also called a (n, m)-*function*
- The output of a (n, m)-function is defined by *m coordinate functions* $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$
- Hence, an S-box $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ can be represented by a $m \times 2^n$ *truth table*, where row *i* is the truth table of f_i
- Example: $n = m = 3$ (the 3-Way S-box)

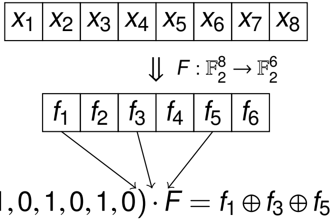
(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
$dec(x_1, x_2, x_3)$	0	1	2	3	4	5	6	7
$F(x_1, x_2, x_3)$	0	5	6	1	3	2	4	7
$f_1(x_1, x_2, x_3)$	0	1	1	0	0	0	1	1
$f_2(x_1, x_2, x_3)$	0	0	1	0	1	1	0	1
$f_3(x_1, x_2, x_3)$	0	1	0	1	1	0	0	1

Component Functions

- Several cryptographic properties of S-boxes are defined in terms of their *component functions*
- Given $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ and a vector $v \in \mathbb{F}_2^m$, the component function $v \cdot F$ is defined for all $x \in \mathbb{F}_2^n$ as:

$$v \cdot F(x) = \bigoplus_{i=1}^{m} v_i f_i(x)$$

- Example with $n = 8$, $m = 6$ and $v = (1, 0, 1, 0, 1, 0)$:



x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
$\Downarrow\ F : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^6$
f_1 f_2 f_3 f_4 f_5 f_6
$(1, 0, 1, 0, 1, 0) \cdot F = f_1 \oplus f_3 \oplus f_5$

- Component functions are thus *linear combinations* of coordinate functions

Walsh-Hadamard Transform (WHT)

- The Walsh-Hadamard Transform (WHT) of a (n,m)-function is the WHT of all its component functions $v \cdot F$, that is

$$W_F(a,v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus a \cdot x} \text{ , for all } a \in \mathbb{F}_2^n, v \in \mathbb{F}_2^m$$

- Example: $n = m = 3$ (the 3-WAY S-box)

(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
$F(x)$	000	101	110	001	011	010	100	111
$W_F(a,000)$	8	0	0	0	0	0	0	0
$W_F(a,001)$	0	4	0	−4	0	4	0	4
$W_F(a,010)$	0	0	0	0	4	−4	4	4
$W_F(a,011)$	0	4	0	4	−4	0	4	0
$W_F(a,100)$	0	0	4	4	0	0	−4	4
$W_F(a,101)$	0	−4	4	0	0	4	4	0
$W_F(a,110)$	0	0	−4	4	4	4	0	0
$W_F(a,111)$	0	4	4	0	4	0	0	−4

Algebraic Normal Form (ANF)

- The ANF can be easily generalized to an S-box $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ through the ANF of its coordinate functions
- In particular, the ANF of F is a multivariate polynomial over $\mathbb{F}_2^m[x_1, \cdots, x_n]/[x_1^2 \oplus x_1, \cdots, x_n^2 \oplus x_n]$:

$$F(x) = \bigoplus_{a \in \mathbb{F}_2^n} h(a) \cdot \left(\prod_{i=1}^n x_i^{a_i} \right)$$

where $h(a) \in \mathbb{F}_2^m$ is again obtained by *Möbius inversion*:

$$h(a) = \bigoplus_{x \preceq a} F(x) \quad , x \in \mathbb{F}_2^n$$

- Taking the *i*-th coordinate in each vector of the ANF of F we have the ANF of the coordinate function f_i

Univariate Polynomial Representation

- When $n = m$, an S-Box $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ can be uniquely represented as a univariate polynomial over the finite field $\mathbb{F}_{2^n}$
- Recall the multiplicative and additive notations for extension fields (see Lecture 8). Given $p(X) \in \mathbb{F}_2[X]$ an primitive polynomial of degree n, let α be a root of $p(X)$
- $\mathbb{F}_{2^n}$ is identified with $\mathbb{F}_2^n$ by the linear combinations of the basis $\{1, \alpha, \alpha^2, \cdots, \alpha^{n-1}\}$ in additive notation, and with the powers $\{1, \alpha, \alpha^2, \cdots, \alpha^{2^n-2}\}$ in multiplicative notation
- Example: $n = 3$ and $p(X) = 1 + X + X^3$, and α is such that $p(\alpha) = 0$. Thus, $\alpha^3 = \alpha + 1$, and the basis is $\{1, \alpha, \alpha^2\}$. The truth table of the 3-Way S-box becomes:

α^i	0	1	α	α^2	α^3	α^4	α^5	α^6
lin. comb.	0	1	α	α^2	$\alpha + 1$	$\alpha^2 + \alpha$	$\alpha^2 + \alpha + 1$	$\alpha^2 + 1$
(x_1, x_2, x_3)	000	001	010	100	011	110	111	101
$F(x)$	000	101	110	011	001	100	111	010
$F(\alpha^i)$	0	α^6	α^4	α^3	1	α^2	α^5	α

Univariate Polynomial Representation

Theorem

Let $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ be a function from $\mathbb{F}_{2^n}$ to itself. Then, there exists a unique polynomial $\sum_{i=0}^{2^n-1} b_i X^i$ such that

$$F(x) = \sum_{i=0}^{2^n-1} b_i x^i \text{ , for all } x \in \mathbb{F}_{2^n}$$

where $b_0 = F(0)$, $b_{2^n-1} = \sum_{x \in \mathbb{F}_{2^n}} F(x)$ and for all $1 \le i \le 2^n - 2$ the value of b_i is given by the Mattson-Solomon transform:

$$b_i = \sum_{j=0}^{2^n-2} F(\alpha^j)\alpha^{-ij}$$

Power Functions

- *Power functions* are S-boxes $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ of the form $F(x) = x^d$ in univariate polynomial representation
- Power functions are interesting for designing block ciphers since they have low implementation costs
- The S-box used in AES is a composition of an affine function with a power function over $\mathbb{F}_{2^8}$
- $\mathbb{F}_{2^8}$ is identified with $\mathbb{F}_2^8$ via the irreducible polynomial

$$1 + x + x^3 + x^4 + x^8$$

- The power function in the AES S-box with this isomorphism corresponds to the *inversion* over $\mathbb{F}_{2^8}$, that is

$$F(x) = \begin{cases} x^{-1} = x^{254} & \text{, if } x \neq 0 \\ 0 & \text{, if } x = 0 \end{cases}$$

This Lecture

- Basic concepts of Block Ciphers
- S-Boxes
  - Representations of S-boxes
  - Cryptographic Properties of S-Boxes

Balancedness

- $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ is *balanced* if $|F^{-1}(y)| = 2^{n-m}$ for all $y \in \mathbb{F}_2^m$
- In other words, each output vector $y \in \mathbb{F}_2^m$ must appear the same number of times 2^{n-m} in the truth table of F
- F is balanced iff for all $v \in \mathbb{F}_2^m \setminus \{0\}$, the component function $v \cdot F$ is balanced
- Balanced functions with $m = n$ are *invertible* (or bijective) S-boxes, since $|F^{-1}(y)| = 2^{n-n} = 1$
- Example: $n = m = 3$, the 3-Way S-box

(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
$F(x)$	000	101	110	001	011	010	100	111

$$\Downarrow$$

F is balanced (bijective)

Algebraic Degree

- The algebraic degree of an (n, m)-function $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ is the degree of its ANF
- Equivalently, the degree of F is the maximum degree of the ANF of all its coordinate functions
- S-boxes of high degree thwart *higher-order differential attacks*
- Example: $n = m = 3$, the 3-Way S-box

$$\begin{aligned} f_1(x_1, x_2, x_3) &= x_1 \oplus x_1 x_2 \oplus x_3 \Rightarrow deg(f_1) = 2 \\ f_2(x_1, x_2, x_3) &= x_2 \oplus x_2 x_3 \oplus x_1 \Rightarrow deg(f_2) = 2 \\ f_3(x_1, x_2, x_3) &= x_3 \oplus x_3 x_1 \oplus x_2 \Rightarrow deg(f_3) = 2 \end{aligned}$$

$\Rightarrow$ the degree of F is 2

Nonlinearity

- Given $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, recall that the Walsh-Hadamard transform for component $v \cdot F$ is, for all $a \in \mathbb{F}_2^n$:
$$W_f(a, v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus a \cdot x}$$
- Hence, the nonlinearity of component $v \cdot F$ is:
$$nl(v \cdot F) = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} \left\{ \left| W_F(a, v) \right| \right\}$$
- The *nonlinearity* of a S-box F is defined as the *minimum nonlinearity* among all its component functions $v \in \mathbb{F}_2^m \setminus \{0\}$:
$$nl(F) = min_{v \in \mathbb{F}_2^m \setminus \{0\}} \{ nl(v \cdot F) \}$$
- S-boxes with high nonlinearity allow to resist to *linear cryptanalysis* attacks

Nonlinearity – Example

- Example: $n = m = 3$, nonlinearity of the 3-Way S-box

(x_1,x_2,x_3)	000	001	010	011	100	101	110	111	nl
$F(x)$	000	101	110	001	011	010	100	111	
$W_F(a,001)$	0	4	0	−4	0	4	0	4	2
$W_F(a,010)$	0	0	0	0	4	−4	4	4	2
$W_F(a,011)$	0	4	0	4	−4	0	4	0	2
$W_F(a,100)$	0	0	4	4	0	0	−4	4	2
$W_F(a,101)$	0	−4	4	0	0	4	4	0	2
$W_F(a,110)$	0	0	−4	4	4	4	0	0	2
$W_F(a,111)$	0	4	4	0	4	0	0	−4	2

$$\Downarrow$$

Nonlinearity of F: $nl = 2$

Covering Radius Bound for S-boxes

- The *Covering Radius Bound* also holds for S-boxes:

$$nl(F) \leq 2^{n-1} - 2^{\frac{n}{2}-1}$$

- S-boxes satisfying ("=") the CRB (upper bound) are again called *bent*, and they exist only for n even and when $m \leq \frac{n}{2}$
- A bent function F also has the property that all its *derivatives*

$$D_a F(x) = F(x) \oplus F(x \oplus a)$$

are balanced for all $a \in \mathbb{F}_2^n \setminus \{0\}$. For this reason, bent functions are also called *Perfect Nonlinear* (PN)

Sidelnikov-Chabaud-Vaudenay Bound

- When $n = m$, a tighter bound exists for the nonlinearity of S-boxes, i.e. the *Sidelnikov-Chabaud-Vaudenay* (SCV) bound:

$$nl(F) \leq 2^{n-1} - 2^{\frac{n-1}{2}}$$

- S-boxes reaching this bound are called *Almost Bent* (AB)
- AB functions exist only when n is odd
- The SCV bound holds also when $m < n$, however it has a more complicated form:

$$nl(F) \leq 2^{n-1} - \frac{1}{2}\sqrt{3 \cdot 2^n - 2 - 2\frac{(2^n - 1)(2^{n-1} - 1)}{2^m - 1}}$$

Differential Uniformity

- Given $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, the *delta difference table* of F with respect to $a \in \mathbb{F}_2^n \setminus \{0\}$ and $b \in \mathbb{F}_2^m$ is:

$$\Delta_F(a,b) = \left\{x \in \mathbb{F}_2^n : D_a F(x) = b\right\}$$

  where $D_a F(x) = F(x) \oplus F(x \oplus a)$ is the derivative of F in direction a introduced before for PN functions
- Let $\delta_F(a,b) = |\Delta_F(a,b)|$. The *differential uniformity* of F is:

$$\delta_F = \max_{\substack{a \in \mathbb{F}_2^n \setminus \{0\} \\ b \in \mathbb{F}_2^m}} \delta_F(a,b)$$

- S-boxes should have low differential uniformity to resist *differential cryptanalysis attacks*

Differential Uniformity – Example

- Example: $n = m = 3$, nonlinearity of the 3-Way S-box

(x_1, x_2, x_3)	000	001	010	011	100	101	110	111
$F(x)$	000	101	110	001	011	010	100	111

$$\Downarrow$$

$\delta_F(a,b)$	000	001	010	011	100	101	110	111
001	0	2	0	2	0	2	0	2
010	0	0	0	0	2	2	2	2
011	0	2	0	2	2	0	2	0
100	0	0	2	2	0	0	2	2
101	0	2	2	0	0	2	2	0
110	0	0	2	2	2	2	0	0
111	0	2	2	0	2	0	0	2

$\Rightarrow$ differential uniformity of F: $\delta_f = 2$

Difference Distribution Table

- The sum of each row/column is 2^n.
- All the values are even.
- If we would also calculate DDT for $a = 0$, then the top left corner value ($a = b = 0$) is always 2^n.
- Ideal S-box would have all values with probability of occurring some differential would be $1/2^n$.

Bounds on Differential Uniformity

- Remark that δ_f is always even, since if x is a solution of

$$F(x) \oplus F(x \oplus a) = b$$

then also $x \oplus a$ is a solution. Consequently,

$$\delta_F \geq 2$$

- S-boxes with $\delta_f = 2$ are also called *Almost Perfect Nonlinear* (APN) functions
- Every AB function is also APN, but the converse is not true
- APN functions have optimal resistance against differential attacks

S-boxes (Optional)

- Equivalently, a function F is APN if the system of equations

$$\begin{cases} x + y + z + t = 0 \\ F(x) + F(y) + F(z) + F(t) = 0 \end{cases}$$

  has for solutions only quadruples (x, y, z, t) whose elements are not all distinct (i.e., are pairwise equal).
- Changing d into one of its conjugates $2^j d$ corresponds to changing $F(x)$ into a linearly equivalent APN function, which preserves APNness.

Algebra & Cryptography CSE3230

Lecture 11 – The AES block cipher and modes of operations

Stjepan Picek

Recap – Previous Lecture

- Basic concepts of block ciphers
- Substitution Permutation Networks (SPN)
- Feistel Networks (FN)
- Cryptographic properties of S-boxes

Takeaway: S-Boxes are an important primitive in the design of both SPN and FN ciphers. Similarly to Boolean functions for stream ciphers, several attacks on block ciphers are formalizable in terms of the cryptographic properties of S-boxes

This Lecture

- S-boxes
  Known classes of S-boxes
- Real-world Block Ciphers
  Legacy Block Ciphers: DES and 3DES
  The AES Block Cipher
  Modes of Operation

References and reading material

- Carlet, C.: Vectorial Boolean Functions for Cryptography. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methodsin Mathematics, Computer Science, and Engineering, pp. 257–397. Cambridge University Press, 2010 (available at: `https://www.math.univ-paris13.fr/~carlet/chap-vectorial-fcts-corr.pdf`)
- Further reading: Stinson, D. Paterson, M.: Cryptography: theory and practice (4th ed.). CRC press, 2018
- For curiosity: The Real World Cryptographers podcast recently featured an episode with Vincent Rijmen and Joan Daemen on the history of AES: `https://www.buzzsprout.com/1393207/8080601`

Outline of this Lecture

- S-boxes
  Known classes of S-boxes
- Real-world Block Ciphers
  Legacy Block Ciphers: DES and 3DES
  The AES Block Cipher
  Modes of Operation

Construction of good S-boxes

To be used in a block cipher, an S-box $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ should:

- be invertible when used in a SPN cipher with $n = m$ (for FN ciphers, balanced S-boxes are desirable)
- have high algebraic degree
- have high nonlinearity (ideally, be an AB function)
- have low differential uniformity (ideally, be an APN function, or otherwise have $\delta_f = 4$)

The number of (n, m)-functions is $(2^m)^{2^n}$

- Exhaustive search already unfeasible for $n = m = 4$
- The search space can be reduced by *affine equivalence*
- Power functions give several constructions of AB and APN functions

Known AB and APN functions

- Most of the known AB (Almost Bent - lecture 10) and APN (Almost Perfect Nonlinear - lecture 10) functions are defined by *power functions* which we introduced earlier:

$$F : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n} \text{ , } F(x) = x^d \ \ \forall x \in \mathbb{F}_{2^n}$$

- APNness is characterized by conditions on the exponent d
- Some of the main classes are characterized by these exponents:

Function	Exponent d	Conditions
Gold	$2^i + 1$	$\gcd(i,n) = 1$
Kasami	$2^{2i} - 2^i + 1$	$\gcd(i,n) = 1$
Welch	$2^t + 3$	$n = 2t + 1$
Niho	$\begin{cases} 2^t + 2^{\frac{t}{2}} - 1 & \text{, t even} \\ 2^t + 2^{\frac{3t+1}{2}} - 1 & \text{, t odd} \end{cases}$	$n = 2t + 1$
Inversion	$2^{2t} - 1$	$n = 2t + 1$

Bijectivity of APN power functions

Theorem (Dobbertin)

If a power function $F(x) = x^d$ is APN, then

$$\gcd(d, 2^n - 1) = \gcd(3, 2^n - 1)$$

Consequences:

- *If n is odd, F is bijective*
- *If n is even, F is 3-to-1 (except at 0)*

- The question whether there exist APN permutations not defined by power functions is open for $n \geq 8$
- For $n = 4$, no APN permutations exist
- For $n = 6$, an APN permutation was found relatively recently in 2009 by Dillon et al.

Outline of this Lecture

- S-boxes
  Known classes of S-boxes

- Real-world Block Ciphers
  Legacy Block Ciphers: DES and 3DES
  The AES Block Cipher
  Modes of Operation

Concrete Block Ciphers

- Block ciphers are the most commonly used symmetric cryptosystems in practice, since they are more versatile than stream ciphers
- Under certain *modes of operations* block ciphers can act also as stream ciphers, and they can provide message integrity and authentication
- Typical block sizes and key lengths vary between 128 and 256 bits, to avoid *brute-force attacks*
- Older block ciphers used smaller key lengths and are nowadays insecure, although they are based on a solid design
- An example is the *Data Encryption Standard* (DES)

Data Encryption Standard (DES)

- Developed by IBM in the '70s as an improvement of the *Lucifer* cipher, standardized by NIST in 1977
- Feistel cipher with $r = 16$ rounds and block size of $b = 64$ bit
- The confusion layer is made of 8 S-boxes $S_i : \mathbb{F}_2^6 \to \mathbb{F}_2^4$, followed by a permutation
- Key length of $l = 56$ bit, with 8 parity check bits added to match the block length
- Reasonable keyspace for the '70s, but could be brute-forced with specialized hardware already in the '90s
- Although insecure, it is still used in some legacy applications

3DES

- Temporary replacement to address the shortcomings of DES, standardized in 1995
- Applies DES three times on the plaintext block, using three 56-bit keys k_1, k_2, k_3:
- The plaintext is first encrypted with DES using k_1, then decrypted with k_2, and finally re-encrypted with k_3
- Different keying options:
  - Option 1: All three keys are different (key length of 168 bits)
  - Option 2: $k_1 = k_3$ (key length of 112 bits)
  - Option 3: $k_1 = k_2 = k_3$ (key length of 56 bits, option used only for backward compatibility with DES)
- Declared deprecated by NIST in 2017

Outline of this Lecture

The AES Competition

- In 1997, the NIST called for a competition for selecting a new block cipher to supersede DES
- The winner cipher would have later been adopted as the *Advanced Encryption Standard* (AES)
- 15 candidates ciphers were submitted, out of which 5 finalists were selected:
  - RIJNDAEL
  - SERPENT
  - TWOFISH
  - RC6
  - MARS
- After a further selection round, RIJNDAEL was selected

Rijndael – The AES algorithm

- Designed by Joan Daemen and Vincent Rijmen
- SPN cipher with block size of 128 bits, support key lengths of 128, 192 and 256 bits
- The number of rounds is determined as follows:

Key length	Rounds
128 bits	10
192 bits	12
256 bits	14

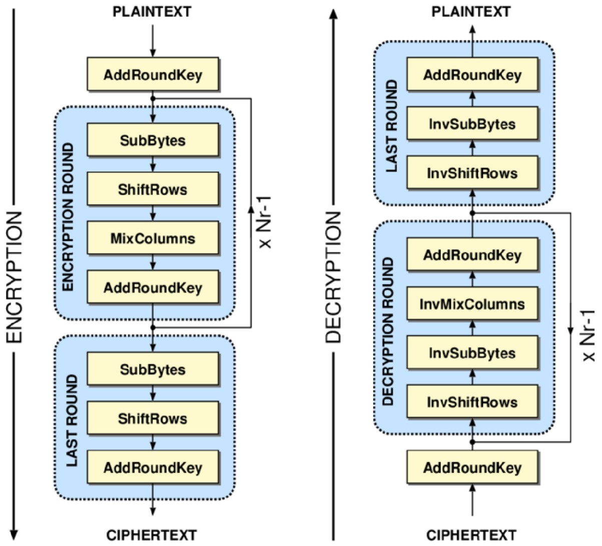
- Byte-oriented designed, which results in efficient software implementations
- Operations in the confusion and diffusion layer are done in $\mathbb{F}_{2^8}$

High level description of AES

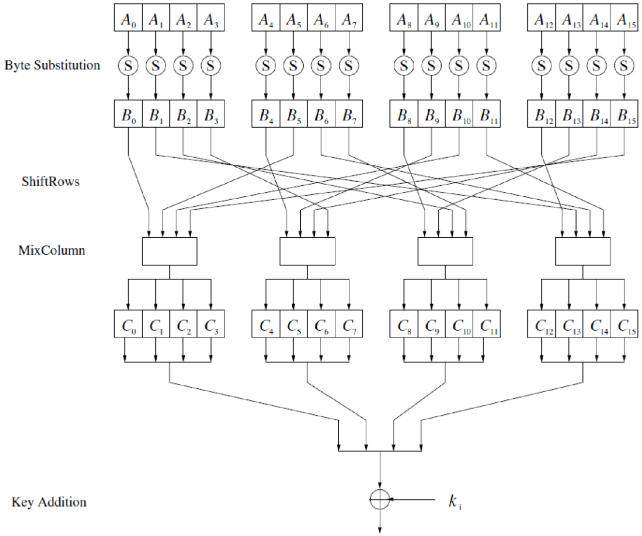
Given a plaintext x and an encryption key k:

1. Set the state $s \leftarrow x$, and run the *key scheduling algorithm* KeyExpansion to derive $r+1$ round keys $k_1, \cdots, k_{r+1}$
2. Perform AddRoundKey on s with k_1
3. For all rounds $2 \leq i \leq r$ do these operations on s:
   - 3.1 SubBytes
   - 3.2 ShiftRows
   - 3.3 MixColumns
   - 3.4 AddRoundKey with k_i
4. For the final round, perform SubBytes, ShiftRows and AddRoundKey with k_{r+1} on s
5. Return s as the ciphertext y

High-level Description – Block Scheme



AES Round Function – Block Scheme

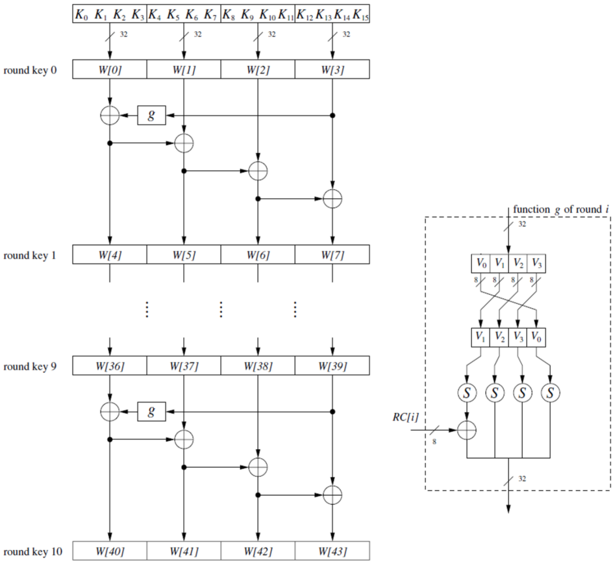


A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11} A_{12} A_{13} A_{14} A_{15}
Byte Substitution
S
B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9 B_{10} B_{11} B_{12} B_{13} B_{14} B_{15}
ShiftRows
MixColumn
C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10} C_{11} C_{12} C_{13} C_{14} C_{15}
Key Addition
k_i

Key Scheduling Algorithm – KeyExpansion

- Let's consider the case of a 128-bit key k (for the other key lengths there are minor differences)
- KeyExpansion is *word*-oriented (1 word $\equiv$ 4 bytes), i.e. it represents the round keys as sequences of 4 words
- The concatenation of the 11 round keys is a sequence of 44 words (called the *expanded key*), denoted as $W[0], \cdots W[43]$
- Starting from the first 4 words k, the KeyExpansion uses the following operations to derive the next words:
  - RotWord: perform a cyclic shift of the bytes in the word
  - SubWord: applies the AES S-box on the bytes of the word
  - XOR with *round constants*

KeyExpansion – Block Scheme



AddRoundKey

- Initialization: The 128-bit state s is represented as a 4×4 bytes matrix and initialized with the bytes of the plaintext

$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|}\hline x_0 & x_4 & x_8 & x_{12} \\ \hline x_1 & x_5 & x_9 & x_{13} \\ \hline x_2 & x_6 & x_{10} & x_{14} \\ \hline x_3 & x_7 & x_{11} & x_{15} \\ \hline \end{array}$$

- Each byte of s is then bitwise-XORed with the corresponding byte of the round key $w_{4i}, w_{4i+1}, w_{4i+2}, w_{4i+3}$ of round i
- *Whitening*: AddRoundKey is the first and the last operation performed in AES encryption
- Whitening prevents an attacker to perform even a single encryption or decryption operation

SubBytes

- Constitutes the *confusion layer* of AES
- Applies an 8×8 S-box indepedently on each byte of the state
- Composition of an *affine transformation* and a *nonlinear transformation*
- Nonlinear transformation: the inversion power function in $\mathbb{F}_{2^8}$
- Each byte is an element of $\mathbb{F}_{2^8}$, represented as a polynomial in $\mathbb{F}_2[x]/(1+x+x^3+x^4+x^8)$

$$F(x) = \begin{cases} \frac{1}{x} = x^{254} & \text{, if } x \neq 0 \\ 0 & \text{, if } x = 0 \end{cases}$$

- F has nonlinearity 112, and differential uniformity 4

SubBytes

- After the inversion function has been applied on the byte $x \in \mathbb{F}_{2^8}$, the following affine transformation is performed:

$$A \cdot F(x) + c$$

where:

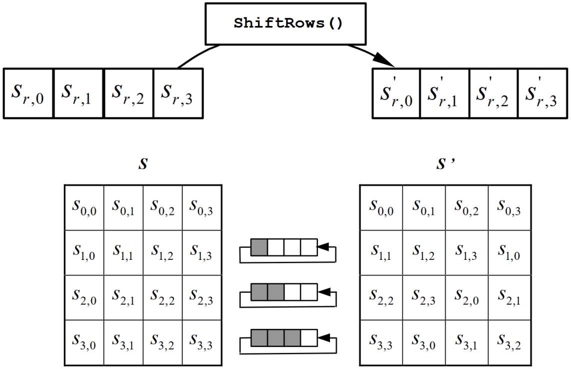
$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

The AES S-box – Lookup table

	Y															
X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

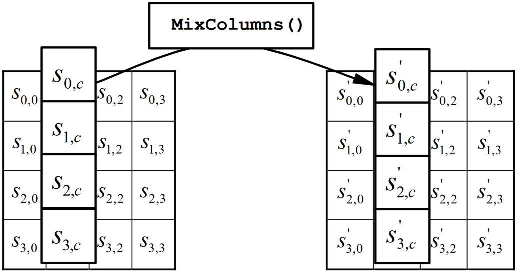
ShiftRows

- Constitutes the first part of the diffusion layer in AES
- ShiftRows applies a cyclic shift on the 4 rows of the state *s* (except the first one), respectively of 1, 2 and 3 positions


ShiftRows()
$s_{r,0}$ $s_{r,1}$ $s_{r,2}$ $s_{r,3}$
$s'_{r,0}$ $s'_{r,1}$ $s'_{r,2}$ $s'_{r,3}$
s
s'
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

MixColumns

- Constitutes the second part of the diffusion layer in AES
- MixColumn mixes the bytes of a 4-byte column, and it is applied independently on each of the 4 columns of *s*



MixColumns()
$s_{0,0}$ $s_{0,c}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,c}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,c}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,c}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,c}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,c}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,c}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,c}$ $s'_{3,2}$ $s'_{3,3}$

MixColumns

- The MixColumns operation can be formalized in terms of polynomials over $\mathbb{F}_{2^8}$
- Each word is represented by a 4-term polynomial $p(X) \in \mathbb{F}_{2^8}[X]$

$$a(X) = a_3X^3 + a_2X^2 + a_1X + a_0$$

  where a_3, a_2, a_1, a_0 are bytes represented as elements of $\mathbb{F}_{2^8}$
- Sum between two 4-term polynomials $p(X), q(X)$ is the usual polynomial sum: the result is still a 4-term polynomial
- Multiplication of $p(X)$ and $q(X)$ does not yield a 4-term polynomial in general, thus $p(X) \cdot q(X)$ is reduced modulo the irreducible polynomial $X^4 + 1$, and denoted as $p(X) \otimes q(X)$

MixColumns

- Each column s_c in the state s is multiplied with the following 4-term polynomial and reduced modulo $X^4 + 1$:

$$a(X) = \{03\}X^3 + \{01\}X^2 + \{01\}X + \{02\}$$

- The operation can also be expressed in terms of matrix multiplication: given $s'_c(X) = a(X) \otimes s_c(X)$, we have

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix}$$

Decryption

- Each layer needs to be inverted, using the inverse operations:
  - InvSubBytes
  - InvShiftRows
  - InvMixColumns
- AddRoundKey equals its own inverse, since it is a bitwise XOR with the round key bytes
- The key scheduling order must be reversed
- Since the last encryption round does not perform the *MixColumns* operation, the first decryption round also does not contain the corresponding inverse layer

AES security

- As of now, AES is secure against all known attacks
- Some attacks have been discovered by considering *reduced-round* variants of AES
- Best attack on full-round AES: *biclique attack*
- However, the biclique attack only gives a slight advantage over brute-force, reducing the keyspace by a factor of 4 or 5 depending on the variant

Outline of this Lecture

- S-boxes
  - Known classes of S-boxes
- Real-world Block Ciphers
  - Legacy Block Ciphers: DES and 3DES
  - The AES Block Cipher
  - Modes of Operation

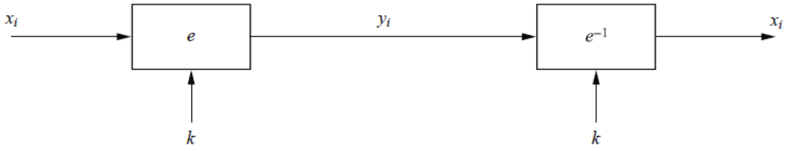
Modes of Operation

- How to encrypt a plaintext longer than the block length?
- There are five standard *confidentiality modes of operation*:
  - Electronic Code Book mode (ECB),
  - Cipher Block Chaining mode (CBC),
  - Cipher Feedback mode (CFB),
  - Output Feedback mode (OFB),
  - Counter mode (CTR).
- All five modes have one goal: they encrypt data and thus provide confidentiality for a message sent from Alice to Bob.
- There exist also other modes for providing *authenticated encryption* (e.g. GCM mode), but we do not cover them here

Modes of Operation

- The ECB and CFB modes require that the length of the plaintext be an exact multiple of the block size of the cipher used, e.g., a multiple of 16 bytes in the case of AES.
- If the plaintext does not have this length, it must be padded.
- One possible padding method is to append a single “1” bit to the plaintext and then to append as many “0” bits as necessary to reach a multiple of the block length.
- If the plaintext length is a multiple of the block length, an extra block consisting only of padding bits is appended.

ECB (Electronic Codebook mode)



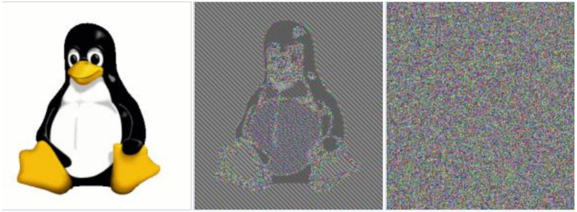
- The most straightforward way of encrypting a message.
- Let us assume that the block cipher encrypts (decrypts) blocks of size *b* bits.
- Messages which exceed *b* bits are partitioned into *b*-bit blocks. If the length of the message is not a multiple of *b* bits, it must be padded to a multiple of *b* bits prior to encryption.

ECB Advantages

- Block synchronization between the encryption and decryption parties Alice and Bob is not necessary, i.e., if the receiver does not receive all encrypted blocks due to transmission problems, it is still possible to decrypt the received blocks.
- Bit errors, e.g., caused by noisy transmission lines, only affect the corresponding block but not succeeding blocks.
- Block ciphers operating in ECB mode can be parallelized, e.g., one encryption unit encrypts (or decrypts) block 1, the next one block 2, and so on.

ECB Disadvantages

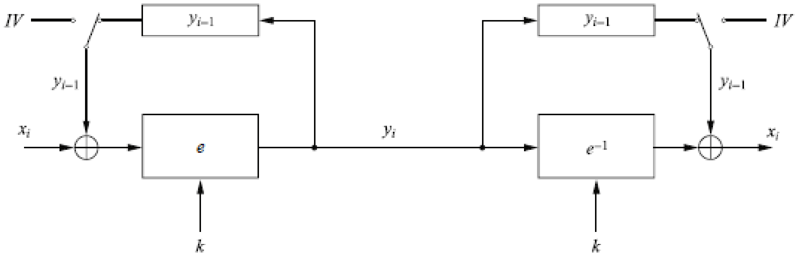
- The main problem of the ECB mode is that it encrypts highly deterministically.
- Identical plaintext blocks result in identical ciphertext blocks, as long as the key does not change.
- An attacker can recognize if the same message has been sent twice simply by looking at the ciphertext.
- If an attacker reorders the ciphertext blocks, this might result in valid plaintext and the reordering might not be detected.


Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

CBC (Cipher Block Chaining mode)

- Two main ideas behind Cipher Block Chaining (CBC):
  1. The encryption of all blocks is “chained together” so that ciphertext y_i depends on all previous plaintext blocks.
  2. The encryption is randomized using an initialization vector (IV).
- The ciphertext y_i corresponding to the plaintext block x_i is XORed with the succeeding plaintext block x_{i+1}.
- This XOR sum is then encrypted, yielding the next ciphertext y_{i+1}, which can then be used for encrypting x_{i+2}, etc.
- For the first plaintext and IV is use, which also makes each CBC encryption nondeterministic.

CBC (Cipher Block Chaining mode)



- The first ciphertext y_1 depends on plaintext x_1 (and the IV).
- The second ciphertext depends on the IV, x_1 and x_2.
- The last ciphertext is a function of all plaintext blocks and IV.

CBC (Cipher Block Chaining mode)

- When decrypting a ciphertext block y_i in CBC mode, we have to reverse the two operations done on the encryption side.
- First, we have to reverse the block cipher encryption by applying the decryption function.
- After this we have to undo the XOR operation by again XORing the correct ciphertext block.

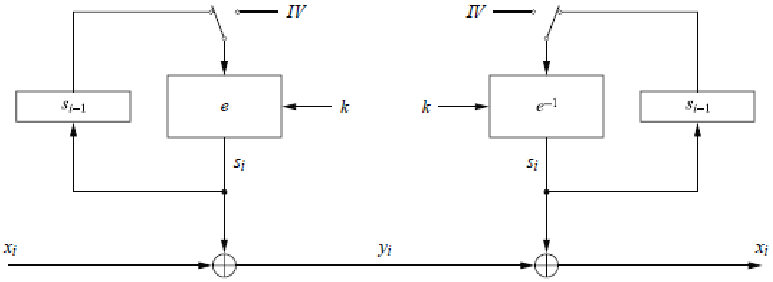
CBC (Cipher Block Chaining mode)

- If we choose a new IV every time we encrypt, the CBC mode becomes a probabilistic encryption scheme.
- If we encrypt a string of blocks $x_1, \ldots, x_t$ once with a first IV and a second time with a different IV, the two resulting ciphertext sequences look completely unrelated to each other for an attacker.
- In most cases, we want the IV to be a nonce, i.e., a number used only once.

OFB (Output Feedback mode)

- In the Output Feedback (OFB) mode a block cipher is used to build a stream cipher encryption scheme.
- Note that in OFB mode the keystream is not generated bitwise but instead in a blockwise fashion.
- The output of the cipher gives us *b* keystream bits, where *b* is the width of the block cipher used, with which we can encrypt *b* plaintext bits using the XOR operation.
- We start with encrypting an IV with a block cipher.
- The cipher output gives us the first set of *b* keystream bits.
- The next block of keystream bits is computed by feeding the previous cipher output back into the block cipher and encrypting it.

OFB (Output Feedback mode)



OFB (Output Feedback mode)

- The OFB mode forms a synchronous stream cipher as the keystream does not depend on the plain or ciphertext.
- Using the OFB mode is quite similar to using a standard stream cipher such as RC4 or Trivium.
- Since the OFB mode forms a stream cipher, encryption and decryption are exactly the same operation.
- The receiver does not use the block cipher in decryption mode to decrypt the ciphertext.
- This is because the actual encryption is performed by the XOR function, and in order to reverse it, we have to perform another XOR function on the receiver side.

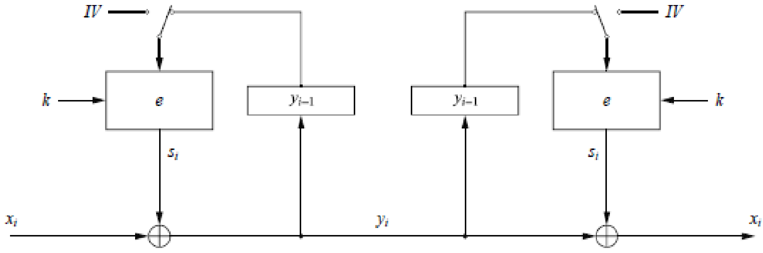
OFB (Output Feedback mode)

- This contrasts with ECB and CBC mode, where the data is encrypted and decrypted by the block cipher.
- Using an IV, OFB encryption is also nondeterministic: encrypting the same plaintext twice results in different ciphertexts.
- As in CBC mode, the IV should be a nonce.
- One advantage of the OFB mode is that the block cipher computations are independent of the plaintext.
- One can precompute one or several blocks of keystream.

CFB (Cipher Feedback mode)

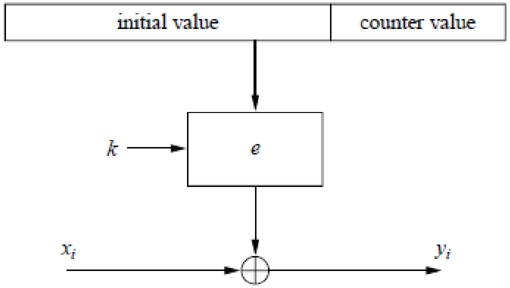
- The Cipher Feedback (CFB) mode also uses a block cipher as a building block for a stream cipher.
- It is similar to the OFB mode but instead of feeding back the output of the block cipher, the ciphertext is fed back.
- As in the OFB mode, the keystream is not generated bitwise but instead in a blockwise fashion.
- For the first keystream block we encrypt an IV. For all subsequent blocks, we encrypt the previous ciphertext.
- Since the CFB mode forms a stream cipher, encryption and decryption are exactly the same operation.

CFB (Cipher Feedback mode)



- CFB is an example of an asynchronous stream cipher since the stream cipher output is also a function of the ciphertext.
- Using an IV, CFB mode is also nondeterministic: encrypting the same plaintext twice results in different ciphertexts.
- As for CBC and OFB modes, the IV should be a nonce.

CTR (Counter mode)



- Another mode which uses a block cipher as a stream cipher.
- As in the OFB and CFB modes, the keystream is computed in a blockwise fashion.
- The input is a counter taking a different value every time the block cipher computes a new keystream block.

CTR (Counter mode)

- We have to be careful how to initialize the block cipher input.
- We must prevent using the same input value twice.
- If an attacker knows one of the two plaintexts that were encrypted with the same input, the keystream block can be computed and thus the ciphertext can be decrypted.
- A common approach is to use *IV* and *CTR*.
- One attractive feature of the CTR mode is that it can be parallelized because, unlike the OFB or CFB mode, it does not require any feedback.

Algebra & Cryptography CSE3230

Lecture 12 – One-Way Functions and Pseudorandom Generators

Stjepan Picek

Recap – Previous Lecture

- Construction of APN functions
- Legacy block ciphers: DES and 3DES
- The AES block cipher
- Block ciphers modes of operation

Takeaway: Due to the flexibility granted by modes of operation, block ciphers (in particular AES) are the most used symmetric ciphers in real-world settings

This Lecture

- One-Way Functions and Pseudorandom Generators
  - Asymptotic Approach to Computational Security
  - One-Way Functions and Permutations
  - Construction of PRG from One-Way Permutations

References and reading material

- Katz, J., Lindell, Y.: Introduction to modern cryptography. CRC press, 2020.
- Further reading: Stinson, D., Paterson, M.: Cryptography: theory and practice (4th ed.). CRC press, 2018
- Further reading (more advanced, only for your curiosity): Goldreich, O.: Foundations of cryptography: volume 1, basic tools. Cambridge university press, 2007

Outline of this Lecture

- One-Way Functions and Pseudorandom Generators
  - Asymptotic Approach to Computational Security
  - One-Way Functions and Permutations
  - Construction of PRG from One-Way Permutations

Unconditional vs. Computational Security

- Recall that the unconditional security paradigm considers adversaries with unbounded computational resources
- Although it is the strongest definition of security, it is also hardly usable in practice: remember that the OTP must have a key of the same length of the plaintext
- Modern ciphers use keys shorter than the plaintext, and fall back on the concept of *computational security*

Kerckhoffs's First Principle

- The concept of computational security was informally defined by Kerckhoffs already in 1883:

  “A cryptosystem must be practically, if not mathematically, indecipherable"

- *Mathematically indecipherable* ⇒ unconditional security
- *Practically indecipherable* ⇒ computational security
- "*Practically*" means that the adversary can't succeed in a "reasonable" time with a "reasonable" probability of success
- In modern cryptography, the "reasonableness" requirement on the adversary is quantified through *computational complexity*

Efficient Adversaries

- The "reasonable" amount of time requirement is modeled by assuming that the adversary is *efficient*, i.e. that it runs in *probabilistic polynomial time*:

Definition

A randomized algorithm A runs in *Probabilistic Polynomial Time* (PPT) if there exists a polynomial p such that for all inputs $x \in \{0,1\}^*$, A terminates after at most $p(|x|)$ steps (where $|x|$ denotes the length of x)

- In particular, by *randomized* algorithm we assume that A has access to a source of uniformly distributed random bits to perform its computation

Negligible Success Probability

- The "reasonable" probability of success is modeled by the concept of *negligible functions*:

Definition

A function $f : \mathbb{N} \rightarrow \mathbb{R}$ is *negligible* if for every polynomial p there exists an $N \in \mathbb{N}$ such that, for all $n \geq N$, it results that

$$f(n) < \frac{1}{p(n)}$$

- In other words, a function is negligible if it grows slower than any inverse polynomial
- A negligible probability of adversarial success is so small that we can consider the scheme practically secure

Asymptotic Approach to Computational Security

- Summing up the two previous requirements, we obtain the following definition for computationally secure cryptosystems:

Definition

A cryptosystem is secure if, for every PPT adversary A performing a particular attack, and for every polynomial p, there exists an integer N such that the probability that A succeeds is negligible:

$$Pr[A \text{ succeeds}] < \frac{1}{p(n)} \text{, for all } n \geq N$$

- This is also called *asymptotic security*, since it holds only for sufficiently large n
- n, in this context, is a *security parameter* of the cryptosystem (e.g., the key length)

Cryptography and Complexity Theory

- Given sufficient time (for attacks), a computationally secure cryptosystem can be trivially broken (e.g. by bruteforcing the keyspace)
- Unconditionally proving that a cryptosystem is computationally secure would entail to prove that it can't be broken by *any* polynomial time (bounded) algorithm
- ... But this would also imply a major breakthrough in complexity theory, namely that $\mathcal{P} \neq \mathcal{NP}$!
  - $\mathcal{P}$: the class of problems that can be solved by polynomial time algorithms
  - $\mathcal{NP}$: the class of problems whose solutions can be verified by a polynomial time algorithm
- The $\mathcal{P}$ vs. $\mathcal{NP}$ question is the most important open problem in theoretical computer science

Provable Security

- Since unconditional proof of computational security is beyond reach at the moment, one usually falls back to the concept of *provable* (or *reductionist*) security
- Idea: assume that some *low-level* problem P is *hard*, and then prove the cryptosystem to be secure under this hardness assumption
- In other words, finding a PPT adversary A that breaks the cryptosystem with non-negligible probability implies that A can be used to solve P with non-negligible probability
- Low-level hard problems $\Rightarrow$ *one-way functions*

Outline of this Lecture

One-Way Functions

- Informally: A function $f : \{0,1\}^* \to \{0,1\}^*$ is *one-way* if it is *easy* to compute, but *hard* to invert
- Given an algorithm A and a parameter n, we formalize the *inversion experiment* as follows:

  $\textsc{Inv}_{A,n}(f)$

  1. Choose an input $x \in \{0,1\}^n$ and compute $y = f(x)$
  2. A is given n and y as input, and computes a candidate preimage $x' \in \{0,1\}^n$
  3. Return 1 if $f(x') = y$, and 0 otherwise

One-Way Functions – Formal Definition

Definition

A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is *one-way* if:

- *Easy Computation*: there exists a polynomial-time algorithm P_f that computes f, that is $P_f(x) = f(x)$ for all inputs $x \in \{0,1\}^*$
- *Hard Inversion*: for every PPT algorithm A, the probability that the inversion experiment $\textsc{Inv}_{A,n}$ returns 1 is negligible, that is, there exists a negligible function $negl : \mathbb{N} \rightarrow \mathbb{R}$ such that

$$Pr[\textsc{Inv}_{A,n}(f) = 1] < negl(n)$$

- Remark: given n, any one-way function can be trivially broken by trying all strings $x \in \{0,1\}^n$
- This strategy however runs in exponential time (there are 2^n bitstrings of length n)

One-Way Permutations

- A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is *length-preserving* if, for all $x \in \{0,1\}^n$, the length of x and of its output $f(x)$ are the same

Definition

Let $f : \{0,1\}^* \rightarrow \{0,1\}^*$ be a length-preserving one-way function, and for all $n \in \mathbb{N}$ denote by $f_n : \{0,1\}^n \rightarrow \{0,1\}^n$ the restriction of f to the inputs of length n. Then, f is called a *one-way permutation* if f_n is bijective for all $n \in \mathbb{N}$

- One-way permutations are fundamental to prove the existence of several cryptographic primitives, including *pseudorandom generators*

One-Way Functions and $\mathcal{P}$ vs. $\mathcal{NP}$

- Remark: if $\mathcal{P} = \mathcal{NP}$, then no one-way functions exist, and no computationally secure crypto is possible!
- On the other hand: if a one-way function exists, then $\mathcal{P} \neq \mathcal{NP}$
- If $\mathcal{P} \neq \mathcal{NP}$, it could still be that *no* one-way function exists: $\mathcal{NP}$ captures the problems that are hard to solve in the *worst case*
- But a one-way function must be hard to invert for almost all instances
- Since we do not know how to prove the (non-)existence of one-way functions, we simply assume that they exist
- We now show some examples of *candidate* one-way functions

Product of Natural Numbers

- The best known example of candidate one-way function is the *product of natural numbers*, i.e. $f_{prod}(x, y) = x \cdot y$ for all $x, y \in \mathbb{N}$
- Inverting f_{prod} corresponds to the *factorization* problem, i.e. determining its prime factors
- Notice that if the length $|x|$ and $|y|$ of x and y are unrestricted, it is easy to invert f_{prod}: with high probability $x \cdot y$ will be even, thus one can return $(2, xy/2)$ as factors
- One way to address this issue is to restrict the domain of f_{prod} to *prime numbers* having the same binary length

Factorization

- Given two equal-length prime numbers x and y, let $n = f_{prod}(x, y)$. Then, the length of n is $m = \log_2 n$
- A trivial algorithm to factorize n is to try divide n for all natural numbers from 2 to $\sqrt{n}$, but the required time is

$$\mathcal{O}(\sqrt{n}) = \mathcal{O}(\sqrt{2^m}) = \mathcal{O}(2^{\frac{m}{2}})$$

  which is exponential in m
- There are other specialized algorithms for factoring (e.g. quadratic sieve, Pollard-rho, ...), but in the general case they still run in exponential time
- Nowadays, it is not known whether a polynomial time algorithm exists for factoring integers

Rabin's Function

- Let $n = pq$ where p and q are two prime numbers. Then, the *Rabin's function* f_R of $x \in \mathbb{N}$ is defined as the squaring of x modulo n:
$$f_R(x) = x^2 \bmod n$$
- It can be shown that extracting square roots modulo n (i.e., inverting f_R) is computationally equivalent to factoring n
- Hence, assuming the hardness of factorization, f_R is also a one-way function

Modular Exponentiation

- Another example of candidate one-way function is *modular exponentiation*
- Let p be a prime number, and consider the multiplicative group of the prime field $\mathbb{Z}_p$, that is $\mathbb{Z}_p^* = \{1, 2, \cdots, p-1\}$
- Recall that $\mathbb{Z}_p^*$ is a cyclic group of order $p-1$, which means that there exists a generator $g \in \mathbb{Z}_p^*$ such that

$$\mathbb{Z}_p^* = \{g^0, g^1, \cdots, g^{p-2}\}$$

- Modular exponentiation is the function f_{exp} defined as follows:

$$f_{exp}(x) = g^x \bmod p \text{ , for all } x \in \mathbb{N}$$

- Every modular exponentiation produces a (supposedly one-way) permutation of the elements of $\mathbb{Z}_p^*$

Modular Exponentiation – Example

- For $p = 7$, the multiplicative group of $\mathbb{Z}_7$ is $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$
- It is easy to see that 3 is a generator for $\mathbb{Z}_7^*$. Indeed:

i	0	1	2	3	4	5
3^i	1	3	9	27	81	243
$3^i \mod 7$	1	3	2	6	4	5

- The modular exponentiation function f_{exp} on $\mathbb{Z}_7^*$ can thus be defined for all $x \in \mathbb{N}$ as:

$$f_{exp}(x) = 3^x \mod 7$$

Modular Exponentiation – Naive algorithm

- Let $x \in \{0, \cdots p-2\}$, and assume that $(x_{n-1}, \cdots, x_0)$ is the binary representation of x, where $n = \log_2 p$. Then:

$$x = \sum_{i=0}^{n-1} x_i 2^i$$

- A naive algorithm to compute $f_{exp}(x)$ is:

```
Mod-Exp-Naive($p, g, x$)
    $res \leftarrow 1$
    while $x > 0$ do
        $res \leftarrow res \cdot g \mod p$
        $x \leftarrow x - 1$
    end while
    return $res$
```

Modular Exponentiation – Naive Algorithm

- The problem of the naive algorithm is that it takes exponential time in $n = \log_2 p$, since it iterates over all powers $g^0, g^1, \cdots, g^x$
- Remark: since it holds

$$g^x = g^{\sum_{i=0}^{n-1} x_i 2^i} = \prod_{i=0}^{n-1} g^{x_i 2^i} = \prod_{i=0}^{n-1} (g^{2^i})^{x_i} \bmod p$$

  an idea would be to compute the values g^{2^i} for $0 \leq i \leq n-1$, and multiply only those where $x_i = 1$
- This can be done in polynomial time using the *Square-and-Multiply* algorithm

Modular Exponentiation – Square-and-Multiply Algorithm

- The *Square-and-Multiply* algorithm computes $g^x \mod p$ in polynomial time wrt n
- The idea is to iteratively square the result, and multiply it by the generator only if the corresponding exponent bit is 1

```
SQUARE-AND-MULTIPLY($p, g, x$)
    $res \leftarrow 1$
    for $i \leftarrow n-1$ down to 0 do
        $res \leftarrow res^2 \mod p$
        if $x_i = 1$ then
            $res \leftarrow (res \cdot g) \mod p$
        end if
    end for
    return $res$
```

The Discrete Logarithm Problem

- The inverse operation of modular exponentiation is the *discrete logarithm*
- Formulation of the problem: given a prime number $p \in \mathbb{N}$, a generator g of $\mathbb{Z}_p^*$ and a value $y \in \mathbb{Z}_p^*$, compute $x \in \{0, 1, \cdots, p-2\}$ such that

$$g^x \equiv y \bmod p$$

- Computing discrete logarithms in $\mathbb{Z}_p^*$ seems to be an intractable problem: no algorithm that runs in polynomial time wrt to $n = \log_2 p$ is known to solve it
- Hence, modular exponentiation can be considered as a candidate one-way permutation

Outline of this Lecture

- One-Way Functions and Pseudorandom Generators
  - Asymptotic Approach to Computational Security
  - One-Way Functions and Permutations
  - Construction of PRG from One-Way Permutations

Pseudorandom Generators (PRG)

- We already saw that *randomness* is essential in several cryptographic primitives
- However, generating *true* random bits is hard, since random sources are rare in nature
- For this reason, cryptographers resort to the notion of *Pseudorandom Generators* (PRG)
- In the past lectures, we saw some practical methods to define PRGs to be used as keystream generators in stream ciphers
- We now formalize PRGs in the asymptotic security context

PRG – Formal Definition

- Let l be a polynomial and G be a deterministic polynomial-time algorithm such that for all $n \in \mathbb{N}$ and $x \in \{0,1\}^n$ G outputs a string of length $l(n)$. Then:

Definition

G is called a *pseudorandom generator* (PRG) if:

1. For all $n \in \mathbb{N}$, it holds that $l(n) > n$
2. For all PPT algorithms D, and for all $r \in \{0,1\}^{l(n)}$ and $s \in \{0,1\}^n$ sampled uniformly at random, the probability that D *distinguishes* r from $G(s)$ is negligible, i.e. there exists a negligible function $negl : \mathbb{N} \to \mathbb{R}$ such that:

$$\Big|Pr[D(r) = 1] - Pr[G(s) = 1]\Big| \leq negl(n)$$

Remarks on PRG

- A PRG is secure as long as there are no PPT algorithms that are able to distinguish between a *true* random string and a string produced by G with non-negligible probability
- If we do not restrict to PPT distinguishers, any PRG G can be trivially broken by simply brute-forcing all possible *seeds* $s \in \{0, 1\}^n$ and computing G on them
- For this reason, the initial seed must be kept secret from the distinguisher
- The existence of PRGs can be proved by assuming the existence of *one-way permutations*

Constructing PRG

- *Question*: if we want to build a PRG $G : \{0,1\}^n \to \{0,1\}^{l(n)}$, do we need to define a different function for each length $l(n)$?
- Luckily, no: if we have a PRG $H : \{0,1\}^n \to \{0,1\}^{n+1}$ (i.e. it expands only by one bit), we can build any other PRG $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ as follows: given $x_0 \in \{0,1\}^n$

$$\begin{aligned} x_1\sigma_1 &= H(x_0) \\ x_2\sigma_2 &= H(x_1) \\ \vdots \quad & \vdots \quad \vdots \\ x_{l(n)}\sigma_{l(n)} &= H(x_{l(n)-1}) \end{aligned}$$

  and finally output $(\sigma_1, \cdots, \sigma_{l(n)})$ as the output of G
- In other words, we apply iteratively H on the first n bits of the output of the previous step

Hard-core Predicates

- Hence, it suffices to construct a PRG that stretches the seed only by one bit: this can be done with *hard-core predicates* of one-way permutations:

Definition

A function $hc : \{0,1\}^* \to \{0,1\}$ is a *hard-core predicate* for a one-way permutation $f : \{0,1\}^* \to \{0,1\}^*$ if:

1. *hc* can be computed by a polynomial time algorithm
2. For all PPT algorithm A there is a negligible function $negl : \mathbb{N} \to \mathbb{R}$ such that

$$Pr[A(f(x)) = hc(x)] \leq \frac{1}{2} + negl(n)$$

   where x is sampled with uniform probability from $\{0,1\}^n$

Hard-core Predicates

- Informally, the output bit of a hard-core predicate cannot be predicted with probability significantly better than $\frac{1}{2}$
- Given a one-way permutation f and a hard-core predicate hc for p, we can define a PRG $H : \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ as:

$$H(x) = f(x) || hc(x) \text{ , for all } x \in \{0,1\}^n$$

  that is, we simply *concatenate* the output of f with that of hc
- Example: given a generator g of $\mathbb{Z}_p^*$ and $y = g^x \bmod p$, Blum and Micali proved that the *most significant bit* of x ($msb(x)$) is a hard-core predicate for the modular exponentiation
- Thus a PRG can be built from modular exponentiation as:

$$H(x) = (g^x \bmod p) || msb(x)$$

Hard-core Predicates for any One-Way Permutation

- The previous result by Blum and Micali is specific to modular exponentiation
- Is there a generic way to obtain a hard-core predicate from any one-way permutation?

Theorem (Goldreich-Levin, 1989)

Let $f : \{0,1\}^* \rightarrow \{0,1\}^*$ be a one-way permutation, and let g be defined as $g(x,y) = f(x) \| y$, where $|x| = |y| = n$. Then g is a one-way permutation, and the scalar product

$$gl(x,y) = \bigoplus_{i=1}^{n} x_i y_i$$

is a hard-core predicate for g.

Proving Pseudorandomness

- Question: how to prove that an algorithm G is a PRG?
- One idea is to perform *statistical tests*: if the strings produced by G do not pass a particular test, then G can be discarded
- The problem is that the number of statistical tests is infinite, so no algorithm G can be proved to be a PRG with this approach!
- However, Yao proved that there exists a universal test:

Definition

An algorithm $G : 0,1^n \to \{0,1\}^{l(n)}$ passes the *next-bit test* if for all $i \in \{1, \cdots l(n)\}$ and for all PPT algorithms A, there exists a negligible function $negl : \mathbb{N} \to \mathbb{R}$ such that

$$Pr[(\sigma_1, \cdots, \sigma_{l(n)}) = G(x);\ A(\sigma_1, \cdots, \sigma_{i-1}) = \sigma_i] = \frac{1}{2} + negl(n)$$

where x is sampled uniformly at random from $\{0,1\}^n$

Proving Pseudorandomness

- Informally, G passes the next-bit test if, given the first $i-1$ bits of the sequence, it is unfeasible to guess the i-th bit
- By "guess" we mean with probability significantly better than $\frac{1}{2}$, i.e. higher than the sum of $\frac{1}{2}$ and any negligible function

Theorem (Yao, 1985)

An algorithm $G : \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$ is a PRG if and only if it passes the next-bit test

The BBS Generator

- Proposed in 1986 by Lenore Blum, Manuel Blum and Michael Shub, the BBS generator is based on the Rabin's function
- Hence, its security relies on the assumption that factoring integers is a computationally hard problem
- Let p and q be two (large) prime numbers such that

$$p \equiv q \equiv 3 \mod 4$$

- Given $n = pq$, the initial random seed s is chosen so that

$$\gcd(s, n) = 1$$

- The algorithm simply consists in iterating Rabin's function over s and taking the least significant bit (lsb) of the result

The BBS Generator

- The pseudocode for the generator is the following:

```
BBS($s, n$)
    $x_0 \leftarrow s^2 \mod n$
    $i \leftarrow 1$
    **while** true **do**
        $x_i \leftarrow x_i^2 \mod n$
        $b_i \leftarrow x_i \mod 2$
        $i \leftarrow i + 1$
    **end while**
```

- The bitstring $b_0, b_1, \cdots$ is the pseudorandom sequence produced by the generator
- It can be proved that the BBS generator passes the next-bit test, and thus it is a PRG

Algebra & Cryptography CSE3230

Lecture 13 – Public key cryptography, Diffie Hellmann key exchange and RSA

Stjepan Picek

Recap – Previous Lecture

- Asymptotic approach to computational security
- PPT algorithms, negligible probability of success
- One-way functions and permutations
- Pseudorandom generators from one-way permutations

Takeaway: One-way functions (OWF) are fundamental to all computationally secure crypto. Although it is not known how to unconditionally prove their existence, we have likely candidates for them (e.g. integer multiplication, modular exponentiation)

This Lecture

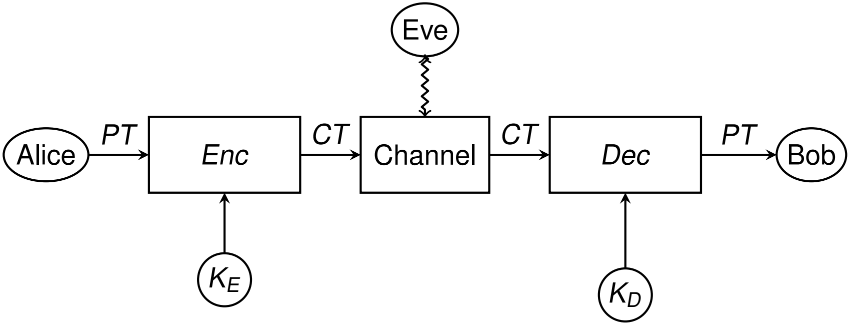
- Public-Key Cryptography
  - Basic Concepts on Public-Key Cryptography
  - The Diffie-Hellman Key Exchange Protocol
- The RSA Cryptosystem
  - Description of RSA
  - Proof of Correctness
  - Basic Attacks to RSA
  - Randomized RSA

References and reading material

- Smart, N.: Cryptography: an introduction. Vol. 3. New York: McGraw-Hill, 2003. `https://homes.esat.kuleuven.be/~nsmart/Crypto_Book/`
- Katz, J., Lindell, Y.: Introduction to modern cryptography. CRC press, 2020.
- Further reading: Stinson, D., Paterson, M.: Cryptography: theory and practice (4th ed.). CRC press, 2018

Outline of this Lecture

The Basic Scenario – Recall



- PT: plaintext
- Enc: encryption function
- K_E: encryption key
- CT: ciphertext
- Dec: decryption function
- K_D: decryption key

Symmetric key cryptosystems assume that $K_E = K_D = K$

How to Share a Symmetric Key?

- The *key distribution problem*: how can Alice and Bob share the secret key K in a secure way?
- There are solutions based only on symmetric key primitives that rely on the use of a *trusted third party* (TTP)
- The TTP acts as a key distribution server (e.g. Kerberos)
- Key distribution servers may not be a scalable solution for open systems such as the Internet
- *Public-key cryptography* addresses this problem by dropping altogether the need for a secure sharing channel

Public-Key Cryptography: The Basic Idea

- *Objective*: allow Alice and Bob to communicate on a public channel, even if they never exchanged a secret key before
- *Idea:* the encryption and decryption keys differ, $K_E \neq K_D$
- K_D cannot be easily obtained by having only K_E
- Both Alice and Bob create their own key pair
  - The encryption key is made public (*public key*)
  - The decryption key is kept secret (*private key*)
- Hence, each user can encrypt with the recipient's public key, and decrypt with its own private key
- How to accomplish this in practice?

Boxes and Padlocks: First Protocol

Let us consider the following metaphor: Alice wants to sends a message *m* to Bob written on a sheet of paper

First protocol:

1. Alice puts the message *m* inside a box, and closes it with a *padlock* whose key is possessed only by her
2. Alice sends the box to Bob, and Bob applies a *second* padlock on it, whose key is possessed only by him
3. Bob sends back the box to Alice, and she removes her padlock using her key
4. Alice sends back the box to Bob, and he removes his padlock using his key. Bob can now open the box and read *m*

Boxes and Padlocks: Second Protocol

A drawback of the first protocol is that two rounds of communication are required: one to apply the padlocks, the other to remove them

Second protocol:

1. Bob sends to Alice a *padlock*, of which only he has the key to unlock it
2. Alice puts the message m inside a box, and closes it with the padlock received by Bob
3. Alice sends the box to Bob, and Bob removes the padlock with its own key. Bob can now open the box and read m

Formalization of the First Protocol

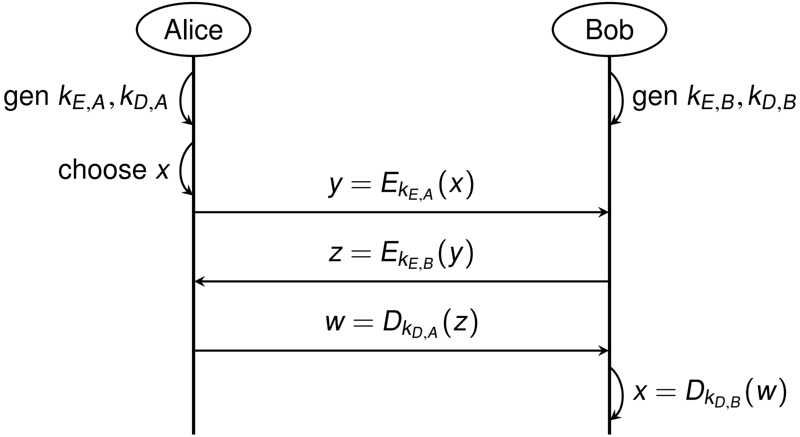
The first protocol can be formalized as follows:

1. Alice and Bob agree (publicly) on an encryption and decryption function, E and D
2. Alice and Bob generate their key pairs $k_{E,A}, k_{D,A}$ and $k_{E,B}, k_{D,B}$
3. Alice encrypts x as $y = E_{k_{E,A}}(x)$ and sends y to Bob
4. Bob encrypts y as $z = E_{k_{E,b}}(y)$ and sends z to Alice
5. Alice decrypts z as $w = D_{k_{D,A}}(z)$ and sends w to Bob
6. Bob recovers the message x by computing $x = D_{k_{D,B}}(w)$

Remark: the encryption and decryption functions must *commute*:

$$E_{k_{E,B}}(E_{k_{E,A}}(x)) = E_{k_{E,A}}(E_{k_{E,B}}(x))$$
$$D_{k_{D,B}}(D_{k_{D,A}}(x)) = D_{k_{D,A}}(D_{k_{D,B}}(x))$$

First Protocol – Scheme

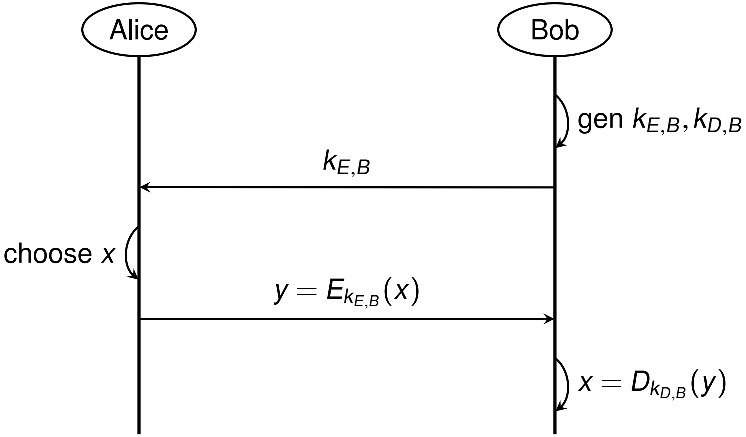


Formalization of the Second Protocol

The second protocol can be formalized as follows:

1. Alice and Bob agree (publicly) on an encryption and decryption function, E and D
2. Bob generates a key pair $k_{E,B}, k_{D,B}$, and sends $k_{E,B}$ to Alice
3. Alice encrypts the message x by computing $y = E_{k_{E,B}}(x)$ and sends y to Bob
4. Bob decrypts y by computing $x = D_{k_{D,B}}(y)$

Second Protocol – Scheme



Properties of the two Protocols

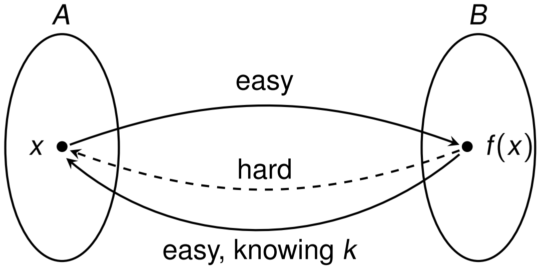
- Clearly, the following properties must hold:
  - For all plaintexts $x \in \mathcal{P}$, $D_{k_{D,A}}(E_{K_{E,A}}(x)) = x$
  - In the second protocol, it must be difficult (unfeasible) to obtain $k_{D,B}$ from $k_{E,B}$
  - In general, the intended use must be easy whereas all unintended uses must be very difficult
- Also remark that:
  - In the first protocol: the encryption keys $k_{E,A}$ and $k_{E,B}$ *can* be published, while $k_{D,A}$ and $k_{D,B}$ *must* be kept secret
  - In the second protocol: the encryption key $k_{E,B}$ *must* be published, while $k_{D,B}$ *must* be kept secret

One-Way Functions in Public-Key Cryptography

- As usual, "easy" and "difficult" are meant in terms of computational complexity:
  - Easy: there exists a (probabilistic) polynomial time algorithm that can be executed
  - Difficult: such an algorithm is not known to exist
- Idea: use the framework of one-way functions to implement the "padlock" for encryption
- But there is a problem: how to implement decryption, i.e. the padlock's key?
- One-way functions (or permutations) are by definition difficult to invert

Trapdoor One-Way Permutations

- A one-way permutation $f : A \to B$ has a *trapdoor* if:
  - Given x, it is easy to compute $f(x)$
  - Given y, by knowing a secret information k it is easy to compute x such that $f(x) = y$
  - Given y, it is difficult to compute x such that $f(x) = y$ if one does not know k
- The secret information k plays the role of decryption key



Outline of this Lecture

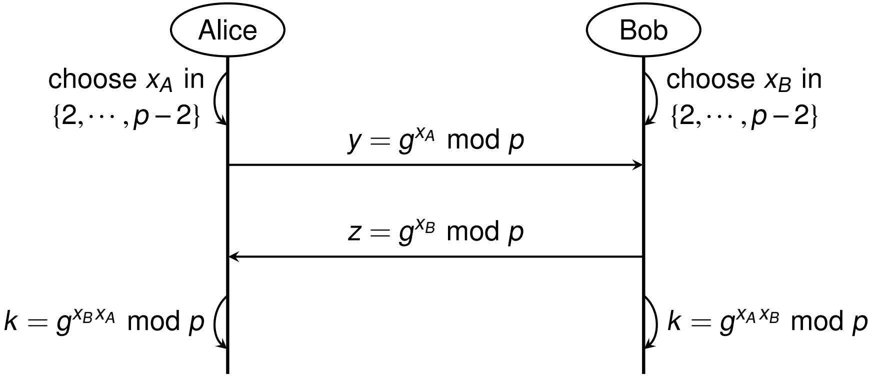
The Diffie-Hellman Protocol

- Introduced by Whitfield Diffie and Martin Hellman in 1976
- It is the first published example of public-key cryptographic protocol, although it is not an encryption scheme
- Rather, it is a *key agreement protocol*, where two parties can use a public channel to share a common secret key
- The agreed secret key is then used to encrypt messages with a symmetric cryptosystem
- It is based on modular exponentiation

The Diffie-Hellman Protocol

1. Alice and Bob choose (publicly) a prime number p, and a generator g of the cyclic group $\mathbb{Z}_p^*$.
2. Alice chooses a random secret integer $x_A \in \{2, \cdots, p-2\}$.
3. Bob chooses a random secret integer $x_B \in \{2, \cdots, p-2\}$.
4. Alice computes $y = g^{x_A} \bmod p$ and sends it to Bob.
5. Bob computes $z = g^{x_B} \bmod p$ and sends it to Alice.
6. Alice computes $z^{x_A} = (g^{x_B})^{x_A} = g^{x_A x_B} = k \bmod p$.
7. Bob computes $y^{x_B} = (g^{x_A})^{x_B} = g^{x_A x_B} = k \bmod p$.
8. k is the shared secret key.

Diffie-Hellman Protocol – Scheme


Alice
Bob
choose x_A in $\{2, \cdots, p-2\}$
choose x_B in $\{2, \cdots, p-2\}$
$y = g^{x_A} \bmod p$
$z = g^{x_B} \bmod p$
$k = g^{x_B x_A} \bmod p$
$k = g^{x_A x_B} \bmod p$

Diffie-Hellman Protocol – Example

1. Alice and Bob agree on the prime $p = 25307$ and $g = 2$ (which is a generator of $\mathbb{Z}^*_{25307}$)
2. Alice chooses $x_A = 3578$
3. Bob chooses $x_B = 19956$
4. Alice computes $g^{x_A} = 2^{3578} = 6113 \pmod{25307}$ and sends it Bob
5. Bob computes $g^{x_B} = 2^{19956} = 7984 \pmod{25307}$ and sends it to Alice
6. Bob computes $k = (g^{x_A})^{x_B} = 6113^{19956} = 3694 \pmod{25307}$
7. When receiving $g^{x_B} = 7984$ from Bob, Alice computes $(g^{x_B})^{x_A}$ = $7984^{3578} = 3694$ (mod 25307), which is the same value of k computed by Bob

Security of Diffie-Hellman Protocol

- Assuming that Eve is only *eavesdropping* the channel, she can observe only the following values:
  - The prime order p for $\mathbb{Z}_p^*$
  - The generator g of $\mathbb{Z}_p^*$
  - The result of the modular exponentiation $g^{x_A} \bmod p$
  - The result of the modular exponentiation $g^{x_B} \bmod p$
- Finding either x_A or x_B to recover k requires computing a *discrete logarithm*
- Alternatively, Eve could try to find $g^{x_A x_B}$ from g^{x_A} and g^{x_B}
- This is called the computational Diffie-Hellman Problem (CDH), and it is also considered to be computationally unfeasible

Outline of this Lecture

RSA

- Published in 1977, the name is the acronym of its three inventors: Ron Rivest, Adi Shamir and Leonard Adleman
- RSA is one of the most studied and used public-key cryptosystems today
- It is based on the (presumed) difficulty of factoring large integer numbers
- Contrary to cryptosystems and protocols based on the discrete logarithm problem (e.g. Diffie-Hellman), RSA actually exploits a *trapdoor* in its underlying one-way function

Outline of this Lecture

- Public-Key Cryptography
  Basic Concepts on Public-Key Cryptography
  The Diffie-Hellman Key Exchange Protocol
- The RSA Cryptosystem
  Description of RSA
  Proof of Correctness
  Basic Attacks to RSA
  Randomized RSA

Key Generation

Each user generates a keypair as follows:

1. Choose two *large* primes p, q of about the same size, but far apart from each other
2. Compute $n = pq$ (also called the *RSA module*) and compute Euler's function $\phi(n) = (p-1)(q-1)$
3. Randomly select d with $1 < d < \phi(n)$ and such that $\gcd(d, \phi(n)) = 1$ (that is, d is invertible in $\mathbb{Z}^*_{\phi(n)}$)
4. Using the *Extended Euclidean Algorithm*, compute the multiplicative inverse e of d modulo $\phi(n)$:

$$ed \equiv 1 \bmod \phi(n)$$

5. *Public key*: (n, e), *private key*: d

Key Generation

- Remark: the values of p, q and $\phi(n)$ must be kept secret
- Indeed, if we know $\phi(n)$, we can take the public key (n, e) and then we can compute d as:

$$d \equiv e^{-1} \bmod \phi(n)$$

- On the other hand, if we know (say) p, we can compute $q = \frac{n}{p}$, then obtain $\phi(n)$ as

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

and finally the decryption exponent d as above

Encryption and Decryption

- A plaintext $x \in \mathbb{Z}_n$ is encrypted with e as:

$$y = x^e \bmod n$$

  if $x \geq n$, then it needs to be split up in pieces, each $< n$
- The decryption of the ciphertext, given the private key d, is obtained as follows:

$$x = y^d \bmod n$$

- *Remark*: the modular exponentiation $x^e \bmod n$, with e and n chosen as above, is believed to be a one-way permutation
- In particular, the decryption exponent d is the *trapdoor*

Outline of this Lecture

Correctness of RSA

- We have to prove that the decryption works correctly, i.e. that $y^d \bmod n$ indeed maps to x. It holds that:

$$y^d \bmod n \equiv (x^e)^d \bmod n \equiv x^{ed} \bmod n$$

- Since $ed \equiv 1 \bmod \phi(n)$, by definition of congruence there is a $t \in \mathbb{Z}$ such that $ed = t\phi(n) + 1$, from which we derive

$$y^d \bmod n \equiv x^{t\phi(n)+1} \bmod n \equiv x \cdot (x^t)^{\phi(n)} \bmod n$$

- Assume now that $\gcd(x, n) = 1$. Then, by Euler's theorem we have $x^{\phi(n)} \equiv 1 \bmod n$, and thus:

$$y^d \bmod n \equiv x \cdot (1)^t \bmod n \equiv x \bmod n$$

which proves correctness of decryption

Correctness of RSA

- What if $\gcd(x,n) \neq 1$? Euler's theorem can't be applied here, but since $n = pq$ and p, q are prime, we can observe that

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

- Since $ed \equiv 1 \bmod \phi(n)$, it means that there are $t, t' \in \mathbb{Z}$ s.t.

$$ed - 1 = t(p-1) \; ; \; ed - 1 = t'(q-1)$$

- Considering the left equation, for $x \in \mathbb{Z}_n$ we obtain:

$$x^{ed} \equiv x^{ed-1+1} \bmod p \equiv x \cdot x^{ed-1} \bmod p \equiv x \cdot x^{t(p-1)} \bmod p$$

- Since p is prime, there are two cases: either $\gcd(x,p) = 1$ or $\gcd(x,p) = p$

Correctness of RSA

- Case 1: $\gcd(x,p) = 1$. In this case, by Fermat's Little Theorem we have that $x^{p-1} \equiv 1 \bmod p$
- Moreover, since x and p are coprime, we can raise each side of the congruence to any power, in particular t:

$$x^{t(p-1)} \equiv 1^t \bmod p \equiv 1 \bmod p$$

- Combining with the fact that $x^{ed} \equiv x \cdot x^{t(p-1)} \bmod p$, we get:

$$x^{ed} \equiv x \cdot x^{t(p-1)} \bmod p \equiv x \cdot 1 \bmod p$$

thus $x^{ed} \equiv x \bmod p$

Correctness of RSA

- Case 2: $\gcd(x, p) = p$. In this case, x is a multiple of p, and thus for any $k \in \mathbb{N}$, x^k is also divisible by p
- For $k = ed$, this means that $x^{ed} \equiv 0 \bmod p$. But since p also divides x, we have $x \equiv 0 \bmod p$, hence:

$$x^{ed} \equiv x \bmod p \text{ , for all } x \in \mathbb{Z}_n$$

- A similar argument can be applied to show also that

$$x^{ed} \equiv x \bmod q \text{ , for all } x \in \mathbb{Z}_n$$

Correctness of RSA

- Since the congruence relation is symmetric, we have thus proved that the following congruence system holds:

$$\begin{cases} x \equiv x^{ed} \bmod p \\ x \equiv x^{ed} \bmod q \end{cases}$$

- p and q are coprime and $n = pq$, so we obtain that

$$x \equiv x^{ed} \bmod n$$

- Finally, since $0 \leq x < n$, there will be a unique integer satisfying the congruence above, thus we get

$$x = x^{ed} \bmod n$$

Toy Example

- Assume that Bob chooses $p = 101$ and $q = 113$, from which $n_B = pq = 11413$ and $\phi(n_B) = (p-1)(q-1) = 11200$
- Now Bob must choose d_B, between 2 and $\phi(n_B) - 1 = 11199$, such that

  $$\gcd(d_B, \phi(n_B)) = \gcd(d_B, 11200) = 1$$

  Assume that he chooses $d_B = 6597$
- By using the Extended Euclidean Algorithm, Bob computes

  $$e_B = d_B^{-1} \bmod \phi(n_B) \equiv 3533 \bmod 11200$$

- Bob's public key is thus $(n_B, e_B) = (11413, 3533)$, and his private key is $d_B = 6597$

Toy Example

- Suppose Alice wants to send to Bob the message $x = 9726$. Thus, she computes the ciphertext as

$$y = 9726^{3533} \bmod 11413 = 5761$$

- Bob recovers x by computing:

$$y^{d_B} = 5761^{6597} \bmod 11413 = 9726$$

- Remark: this is a toy example – In real applications, the keys must be much larger!

Outline of this Lecture

Key Sizes for RSA

- The security of RSA depends on the size of the module n, since if it is too small then it's easy to factorize it
- The largest number that has been factored using specialized factoring algorithms has 250 decimal digits (829 bits)[1]
- Common key sizes for RSA include 1024, 2048, 3072 and 4096 bits
- As of 2015, NIST recommends to use RSA keys of at least 2048 bits to be on the safe side

[1] see `https://en.wikipedia.org/wiki/RSA_Factoring_Challenge`, accessed on 24 Feb 2020

Attacks to RSA

- Recall that p, q and $\phi(n)$ must be kept secret, otherwise the attacker can compute the secret key d
- How hard is to compute $\phi(n)$? It is easy, if we know the factorization of n:

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

  where p ranges over all prime numbers that divide n
- It is hard, if we just start from the public module n without knowing its factorization
- As we said, from $\phi(n)$ we can compute the secret key of RSA
- So, breaking RSA seems to be equivalent to factorizing n

Attacks to RSA

Indeed, if Eve knows just $\phi(n)$, not only she is able to break RSA, but she can also factorize n:

- She knows that $n = p \cdot q$, and that

$$\phi(n) = (p-1)(q-1) = pq - (p+q) + 1 = n - (p+q) + 1$$

- Then, Eve computes $p + q = n - \phi(n) + 1$
- Since she now knows the sum and the product of p and q, she can compute p and q by solving the following quadratic equation:

$$x^2 - (p+q)x + pq = 0$$

- Beside this, there are several constraints to consider when choosing the prime numbers p and q

Attacks to RSA – *p* and *q* too close

- The first constraint is that p and q *should not be too close*
- Let $p > q$. If p and q are close, then $\frac{p-q}{2}$ is small, and $\frac{p+q}{2}$ is just a little bit larger than $\sqrt{n}$, due to the following equality:

$$\frac{(p+q)^2}{4} - n = \frac{(p-q)^2}{4}$$

- we thus deduce that $\frac{(p+q)^2}{4} - n$ is a *perfect square*
- Hence, we look for integers $\sqrt{x} > n$ such that $x^2 - n$ is a perfect square, which we denote by y^2
- from $x^2 - n = y^2$ we obtain $n = x^2 - y^2 = (x+y)(x-y)$, and thus p and q can be recovered as:

$$p = x + y \text{ and } q = x - y$$

Attacks to RSA – large $\gcd(p-1, q-1)$

- The second precaution is that $p-1$ and $q-1$ *should not have large common divisors*
- Suppose that $\gcd(p-1, q-1)$ is large. Then:

$$u = \text{lcm}(p-1, q-1) = \frac{(p-1)(q-1)}{\gcd(p-1, q-1)} = \frac{\phi(n)}{\gcd(p-1, q-1)}$$

  will be small with respect to $\phi(n)$
- Remark that if $d' \equiv e^{-1} \bmod u$, by Euler's theorem we can use d' instead of d to decrypt
- Since u is small, such d' can be found by trial and error

Attacks to RSA – same n for multiple users

- A third recommendation is that one *should not use the same modulo n for different users*
- Assume that a user has to send the same message x to two (or more) users
- Let e_1 and e_2 be the two encryption exponents. He computes:

$$y_1 = x^{e_1} \bmod n$$
$$y_2 = x^{e_2} \bmod n$$

- If $\gcd(e_1, e_2) = 1$ then Eve, by using the Extended Euclidean Algorithm, computes r, s such that $re_1 + se_2 = 1$
- Once she has determined r and s, Eve can finally compute:

$$y_1^r y_2^s \equiv x^{re_1} x^{se_2} \equiv x^{re_1 + se_2} \equiv x \bmod n$$

Attacks to RSA – same (small) e for different n

- A fourth precaution is *not to use the same (small) encryption exponent e for different RSA modules n*
- Let $e = 3$ and suppose that a user wants to send x to three other users A, B, C whose RSA modules are n_A, n_B and n_C.
- He thus computes the three ciphertexts as:

$$y_A = x^3 \bmod n_A$$
$$y_B = x^3 \bmod n_B$$
$$y_C = x^3 \bmod n_C$$

Attacks to RSA – same (small) e for different n

- If n_A, n_B and n_C are pairwise coprime, Eve can use the Chinese Remainder Theorem (CRT) and compute x such that:

$$\begin{cases} x \equiv y_A \bmod n_A \\ x \equiv y_B \bmod n_B \\ x \equiv y_C \bmod n_C \end{cases}$$

- In this way, Eve finds a unique solution x^* between 0 and $N-1$, where $N = n_A n_B n_C$
- Since x^3 is less than n_A, n_B and n_C, it follows that $x^3 < N$, that is, $x^* = x^3$
- Eve can recover x by computing the integer cube root of x^*

Outline of this Lecture

Hard-core Predicate for RSA

- Assume that we have to encrypt one bit at the time, using RSA
- Problem: for any choice of the public key (n, e), it holds:

  $$0^e \equiv 0 \bmod n$$
  $$1^e \equiv 1 \bmod n$$

  hence, $y = x$, for all $x \in \mathbb{Z}_n$!
- It can be proved that the least significant bit of the plaintext x ($lsb(x)$) is a *hard-core* predicate for the modular exponentiation $x^e \bmod n$
- *Idea*: we put the plaintext bit into $lsb(x)$, and we choose the other bits at random

Randomized RSA

Assume that Alice wants to send the bit $b \in \{0, 1\}$ to Bob. Then, she performs the following steps:

1. Take Bob's public key (n_B, e_B)
2. Choose at random an integer $x < n_B/2$ (hence, $2x < n_B$)
3. Send to Bob $y = (2x + b)^{e_B} \bmod n_B$

When receiving y, Bob does the following to decrypt it:

1. Compute $y^{d_B} \bmod n_B = 2x + b$
2. Takes the least significant bit of the result

Randomized RSA

- Remark: it is not known whether the other bits of x (in particular, how many of them, and which ones) are hard-core predicates for RSA
- Hence, to encrypt in a very secure way a plaintext message x, we can encrypt every bit of x with the above randomized version of RSA
- Cryptanalysis becomes very difficult
- However, if the message is long, this method is very inefficient

Algebra & Cryptography CSE3230

Lecture 14 – Public key cryptography, primality testing

Stjepan Picek

Recap – Previous Lecture

- Recap of basic concepts in public key cryptography
- Diffie-Hellmann key exchange
- RSA cryptosystem
- Attacks on RSA, Randomized RSA

Takeaway: Diffie-Hellmann is specifically designed to solve the agreement problem of a symmetric key between two parties. RSA is instead an encryption system, though in practice it is also used to exchange a symmetric key.

This Lecture

- Public-Key Cryptography
  - The Rabin's Cryptosystem
  - The ElGamal Cryptosystem
  - Hybrid Cryptosystems
- Primality Testing and Factorization Algorithms
  - Probabilistic Primality Tests
  - Fermat's Primality Test
  - Factorization Algorithms
  - Pollard's ρ Algorithm

References and reading material

- Smart, N.: Cryptography: an introduction. Vol. 3. New York: McGraw-Hill, 2003. `https://homes.esat.kuleuven.be/~nsmart/Crypto_Book/`
- Stinson, D., Paterson, M.: Cryptography: theory and practice (4th ed.). CRC press, 2018
- Further reading: Katz, J., Lindell, Y.: Introduction to modern cryptography. CRC press, 2020.

Outline of this Lecture

Outline of this Lecture

- Public-Key Cryptography
  The Rabin's Cryptosystem
  The ElGamal Cryptosystem
  Hybrid Cryptosystems

- Primality Testing and Factorization Algorithms
  Probabilistic Primality Tests
  Fermat's Primality Test
  Factorization Algorithms
  Pollard's ρ Algorithm

The Rabin's Cryptosystem

- Published in 1979 by Michael Rabin
- The encryption is based on Rabin's candidate one-way function, i.e. modular squaring
- Hence, the security of this cryptosystem is also related to the difficulty of factoring large integer numbers
- It can be proved that Rabin's cryptosystem is computationally secure against chosen-plaintext attacks

Key Generation

Each user generates a keypair as follows:

- Select two large prime numbers p, q such that

$$p \equiv q \equiv 3 \bmod 4$$

- Compute $n = p \cdot q$
- *Public key*: $K_E = n$
- *Private key*: $K_D = (p, q)$

Encryption and Decryption

- A plaintext $x \in \mathbb{Z}_n^*$ (if it is longer, it must be splitted in pieces less than n) is encrypted with n as:

$$y = x^2 \bmod n$$

- Given the private key formed by p and q, the decryption of the ciphertext y is obtained as follows:
  - Compute the square root of y mod p and q using the followings:

$$m_p = y^{(p+1)/4} mod\ p; m_q = y^{(q+1)/4} mod\ q.$$

  - Use the Extended Euclidean Algorithm to compute c_p and c_q so that $c_p \cdot p + c_q \cdot q = 1$.
  - Use the CRT to find the square roots of y mod n.
    $x_1 = (c_p \cdot p \cdot m_q + c_q \cdot q \cdot m_p) mod\ n$;
    $x_2 = n - x_1$;
    $x_3 = (c_p \cdot p \cdot m_q - c_q \cdot q \cdot m_p) mod\ n$;
    $x_4 = n - x_3$.
- However, there is a problem: decryption is not unique!

An Example of Rabin

Set $p = 7, q = 11$, and $m = 20$. We have the followings:

- Encryption: $y = 20^2 mod\ 77 = 15 mod\ 77$.
- Compute $m_p = 15^2 mod\ 7 = 1$; $m_q = 15^3 mod\ 11 = 9$;
  Use EEA to compute: $c_p = -3; c_q = 2$, so that
  $c_p p + c_q q = (-3 \cdot 7) + (2 \cdot 11) = 1$;
  Compute: $x_1 = (-3 \cdot 7 \cdot 9 + 2 \cdot 11 \cdot 1) mod\ 77 = 64$;
  $x_2 = 77 - 64 = 13$;
  $x_3 = 20$;
  $x_4 = 57$.

Quadratic Residues

Definition

Let p be an odd prime and $a \in \mathbb{Z}$. Then a is defined to be:

- A *quadratic residue* modulo p if $a \not\equiv 0 \bmod p$ and $y^2 \equiv a \bmod p$ has a solution $y \in \mathbb{Z}_p$
- A *quadratic non-residue* modulo p if $a \not\equiv 0 \bmod p$ and a is not a quadratic residue modulo p

Definition

Given an odd prime p and $a \in \mathbb{Z}$, the *Legendre symbol* $\left(\frac{a}{p}\right)$ is defined as follows:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{, if } a \equiv 0 \bmod p \\ 1 & \text{, if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{, if } a \text{ is a quadratic non-residue modulo } p \end{cases}$$

Unique Decryption

Theorem

Suppose that $n > 1$ is an odd integer having factorization $n = p_1^{e_1} \cdots p_l^{e_l}$ where the p_i are distinct primes and the e_i are positive integers. Suppose also that $\gcd(a, n) = 1$. Then the congruence $y^2 \equiv a \bmod n$ has 2^l solutions modulo n if $\left(\frac{a}{p_i}\right) = 1$ for all $i \in \{1, \cdots, l\}$, and no solutions otherwise.

- Hence, decryption in Rabin's cryptosystem yields exactly $2^2 = 4$ square roots of y modulo n, since $n = pq$
- Thus, Bob needs additional information in order to uniquely recover the plaintext x
- If the plaintext space $\mathcal{P}$ is a *sparse* subset of $\mathbb{Z}_n^*$, then one possibility would be to output the only root r among the obtained 4 such that $r \in \mathcal{P}$

Outline of this Lecture

- Public-Key Cryptography
  The Rabin's Cryptosystem
  The ElGamal Cryptosystem
  Hybrid Cryptosystems
- Primality Testing and Factorization Algorithms
  Probabilistic Primality Tests
  Fermat's Primality Test
  Factorization Algorithms
  Pollard's ρ Algorithm

The ElGamal Cryptosystem

- Published by Taher ElGamal in 1985
- Also widely used today in several protocols
- The difficulty of obtaining k_D from k_E, as well as the the difficulty of obtaining x from $y = E_{k_E}(x)$ derives from the (presumed) difficulty of computing discrete logarithms
- Contrary to RSA, it does not use a trapdoor for decryption
- As a matter of fact, up to now there are no known trapdoors that enable efficient computation of discrete logarithms

Key Generation

Each user generates a keypair as follows:

1. Select a random prime number p and consider the multiplicative group $\mathbb{Z}_p^*$
2. Choose a generator g of $\mathbb{Z}_p^*$
3. Select a random integer a such that $0 < a < p-1$, and compute the modular exponentiation g^a mod p
4. *Public key*: (p, g, g^a), *private key*: a

Remark: to obtain a from the public key, we should compute $\log g^a$

Encryption and Decryption

Encryption:

- Assume that Alice wants to send a message $x \in \mathbb{Z}_p^*$ to Bob (if x is longer than $\log_2 p$ bits, she divides it in blocks)
- Alice takes Bob's public key, $k_{E_B} = (p, g, g^a)$
- She chooses at random l, with $0 < l < p - 1$
- She computes $\gamma = g^l \bmod p$ and $\delta = x \cdot (g^a)^l \bmod p$
- She sends to Bob the ciphertext $y = (\gamma, \delta)$

Decryption:

- When Bob receives y, he uses his own private key $k_{D_B} = a$ to compute x as follows:

$$x = g^{-al} \cdot \delta = \gamma^{-a} \cdot \delta = \gamma^{p-1-a} \cdot \delta$$

Security Analysis

- The only information which is transmitted on the channel is $k_{E_B} = (p, g, g^a)$, $\gamma = g^l$ and $\delta = x \cdot (g^a)^l$
- if Eve could extract l from g^l then she could compute $(g^a)^l$ and finally

  $$x = \delta \cdot (g^{al})^{-1} = \delta \cdot g^{-al}$$

  however, this means computing a discrete logarithm
- If Eve could compute g^{al} from g^a and g^l, then she could compute $g^{-al} = (g^{al})^{-1}$, and finally

  $$x = \delta \cdot g^{-al}$$

- Hence, the security of the El Gamal cryptosystem is based on the same assumptions made for the Diffie-Hellman protocol

Outline of this Lecture

A Comparison Between Cryptosystems

- By using public key cryptosystems we can communicate without using private channels
- So, are symmetric cryptosystems useless?
- However, symmetric cryptosystems:
  - Are very fast
  - Do not base their security on unproven mathematical hypotheses
- Symmetric cryptosystems are faster but they require to share a secret key
- Public key cryptosystems, on the other hand, are slower but they do not require a secret key

Hybrid Cryptosystems

- In *hybrid cryptosystems* we use a public key cryptosystem to exchange a secret (*session*) key
- Then, we use a symmetric cryptosystem to encrypt / decrypt data with the established session key
- In this way, we use the slow public-key cryptosystem for communicating only a short amount of key bits (e.g. up to 256 bits for AES)
- Alternatively, one could use the Diffie-Hellman protocol to reach the same goal of key agreement
- This is the idea underlying cryptographic protocols such as *Transport Layer Security* (TLS)

Outline of this Lecture

- Public-Key Cryptography
  The Rabin's Cryptosystem
  The ElGamal Cryptosystem
  Hybrid Cryptosystems
- Primality Testing and Factorization Algorithms
  Probabilistic Primality Tests
  Fermat's Primality Test
  Factorization Algorithms
  Pollard's ρ Algorithm

Generating Large Prime Numbers

- When using public-key cryptosystems such as RSA, it is necessary to generate *very large prime numbers*
- Problem: test if an integer number $n > 2$ is prime or composite, in polynomial time with respect to its size $log_2 n$
- There are many PPT algorithms: Fermat's primality test, Solovay-Strassen, Miller-Rabin, ...
- There is only one deterministic algorithm, discovered by Agrawal, Kayal and Saxena in 2002 (AKS algorithm)
- In practice, only PPT algorithms are actually used for primality testing, since the complexity of AKS is still too high

Probabilistic Primality Tests

- A probabilistic algorithm can be wrong with probability p
- What does it mean for probabilistic primality algorithms?
- If the algorithm returns `TRUE`, then the tested number is certainly *not prime* (i.e., it is composite)
- If the algorithms returns `FALSE`, then the tested number *could still be* composite with probability p
- However, if we run m independent tests on the same number, then the probability of returning a wrong answer becomes p^m

Probabilistic Primality Tests

- How many independent tests *m* are necessary, on average?

Theorem (Prime Number Theorem)

Let $\pi(N)$ be the number of primes less than or equal to N. Then,

$$\pi(N) \sim \frac{N}{\ln N} \Leftrightarrow \lim_{N \to \infty} \frac{\pi(N) \cdot \ln N}{N} = 1$$

- Thus, a number *n* randomly sampled between 1 and *N* will be prime with probability $\frac{\pi(N)}{N} = \frac{1}{\ln N}$
- Example: suppose we generate a 1024-bit random number *n*
- Thus, $N = 2^{1024}$, and *n* is prime with probability $\frac{1}{\ln 2^{1024}} \approx \frac{1}{710}$
- Over $m = 710$ trials, on average one number will be prime

Probabilistic Primality Tests

- Factorizing an integer number seems to be much more difficult than determining whether it is prime or composite
- We can determine that a number is composite *without knowing* its prime factors
- We can apply several primality tests, one after the other, and consider as prime a number for which all tests return `FALSE`
- Alternatively, we use one probabilistic test, applied several times with an independent choice of the random bits

Outline of this Lecture

- Public-Key Cryptography
  - The Rabin's Cryptosystem
  - The ElGamal Cryptosystem
  - Hybrid Cryptosystems
- Primality Testing and Factorization Algorithms
  - Probabilistic Primality Tests
  - Fermat's Primality Test
  - Factorization Algorithms
  - Pollard's ρ Algorithm

Fermat's Primality Test

- Recall *Fermat's little theorem*: If p is prime and $a \in \mathbb{Z}_p$, then

$$a^{p-1} \equiv 1 \bmod p$$

- In this case, we say that a is a *witness* for the primality of p
- If p is composite, then it is not likely that the congruence holds
- If it holds, we say that p is a *pseudoprime* with respect to a
- In this case, a is a *false witness* for the primality of p

Theorem

All or at most half of the integers a such that $1 \leq a < p$ and $\gcd(a, p) = 1$ are witnesses for the primality of p

Fermat's Primality Test

Algorithm 1: Fermat Primality Test

```
Data: n: number to be tested for primality, k: number of times the
      test will be repeated for
Result: Composite or Pseudoprime
Function isPrime(n, k) is
    repeat $k$ times
        a ← uniform random number between 2 and n-1;
        if $a^{n-1}! = 1 \pmod{n}$ then
            return Composite;
        end
    end
    return Pseudoprime;
end
```

Algorithm 2: Calculating Residual with Modular Exponentiation

```
Data: x: exponentiation base, e: exponent, m: modulur operand
Result: Residual of $x^e \pmod{m}$
Function power(x, e, m) is
    result ← 1;
    while $e > 0$ do
        if e is odd then
            result ← (result × x) $\pmod{m}$;
            e ← e - 1;
        else
            x ← (x × x) $\pmod{m}$;
            e ← e/2;
        end
    end
    return result;
end
```

Fermat's Primality Test

- To check whether a number p is prime, we can do as follows:
  1. Randomly choose a such that $1 \leq a < p$
  2. If $\gcd(a, p) > 1$, then p is composite
  3. If $\gcd(a, p) = 1$, compute $u = a^{p-1} \bmod p$
- If $u \neq 1$ then p is composite, otherwise a is a witness for the primality of p
- By the previous theorem, if we have found k witnesses then the probability that p is composite is at most $\frac{1}{2^k}$...
- ... Unless we are in the unlucky case where *all* a are false witnesses for the primality of p
- In this case, p could still be composite (example: *Carmichael's numbers*, which are however very rare)

Outline of this Lecture

- Public-Key Cryptography
  The Rabin's Cryptosystem
  The ElGamal Cryptosystem
  Hybrid Cryptosystems

- Primality Testing and Factorization Algorithms
  Probabilistic Primality Tests
  Fermat's Primality Test
  Factorization Algorithms
  Pollard's ρ Algorithm

Factorization Algorithms

- As we have seen in the previous lecture, factorization is the most obvious way to attack RSA, but it is considered unfeasible (in PPT).
- Under some hypotheses, there are algorithms that factorize n without trying to divide by all integer numbers from 1 to n
- Example of such algorithms include:
  - Pollard's $p-1$ algorithm
  - Pollard's ρ algorithm
  - Elliptic curve factoring algorithm
  - Quadratic sieve
  - Number field sieve
- When the underlying hypotheses do not hold, the running time of these algorithms is exponential

Factorization Algorithms

- Recall that the trivial algorithm to factorize n divides n for all natural numbers from 2 to $\sqrt{n}$, with $m = \log_2 n$
- The required time is $\mathcal{O}(\sqrt{n}) = \mathcal{O}(\sqrt{2^m}) = \mathcal{O}(2^{\frac{m}{2}})$, which is exponential in m
- In general, a factorization algorithm attempts at factorizing n faster than the $\mathcal{O}(2^{\frac{m}{2}})$ upper bound of trivial division
- There are two types of factorization of n:
  - The complete factorization of n into primes
  - Finding any non-trivial factor d of n
- Since the RSA module n is the product of two primes, most factorization algorithms are of the second type

Outline of this Lecture

Pollard's ρ Algorithm

- Published in 1975 by John Pollard
- Let p be the smallest prime divisor of n
- Suppose that we find two values $x, x' \in \mathbb{Z}_n$ such that $x \neq x'$ and $x \equiv x' \bmod p$
- Then $p \leq \gcd(x - x', n) < n$, and thus we obtain a non-trivial divisor of n by computing $\gcd(x - x', n)$
- Pollard's ρ algorithm finds x and x'

Pollard's ρ Algorithm

- Pollard's ρ algorithm is based on the map $f(x) = x^2 + a$, where a is a small constant (for example, $a = 1$)
- It starts from $x_1 \in \mathbb{Z}_n$ chosen at random, and then iterates the map f by building the sequence $x_1, x_2, x_3, \cdots$, where:

$$x_j = f(x_{j-1}) \bmod n \text{ , for all } j \in \mathbb{N}$$

- We thus have to find $x_i \neq x_j$, with $i < j$, and such that $\gcd(x_i - x_j, n) > 1$
- It can be proved that if $x_i \equiv x_j \bmod p$ then $f(x_i) \equiv f(x_j) \bmod p$
- From here we obtain that for all $\delta \geq 0$, $x_{i+\delta} \equiv x_{j+\delta} \bmod p$

Pollard's ρ Algorithm

- If we build a *directed graph* whose vertices are in $\mathbb{Z}_p$, and whose edges go from x_i to x_{i+1}, we obtain an initial *tail*:

$$x_1 \to x_2 \to \cdots \to x_i$$

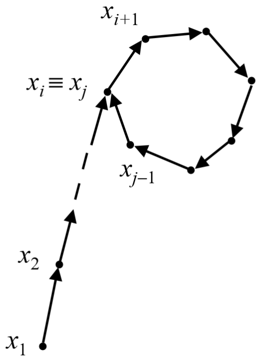
- Followed by a *cycle* of length $j - i$:

$$x_i \to x_{i+1} \to \cdots \to x_j \equiv x_i \bmod p$$

- This is where the algorithm's name comes from: the resulting graph has the shape of the Greek letter "ρ"
- The algorithm becomes simpler if we put $j = 2i$

Pollard's ρ Algorithm – Pseudocode

```
POLLARD-$\rho(n, x_0)$
    $x \leftarrow x_0$
    $x' \leftarrow f(x) \bmod n$
    $p = \gcd(x - x', n)$
    while $p = 1$ do
        $x \leftarrow f(x) \bmod n$
        $x' \leftarrow f(x') \bmod n$
        $p \leftarrow \gcd(x - x', n)$
        [note pay attention to the even position $j = 2i$.]
    end while
    if $p = n$ then
        return FAIL
    else
        return $p$
    end if
```



Pollard's ρ Algorithm

Pollard's Rho Method: Given an integer n which we assume has a small factor we choose some x_0 (often $x_0 = 2$), and we choose $f(x) = x^2 + 1$ (this is typical). We generate $x_1 = f(x_0)$ reduced $\bmod n$, $x_2 = f(x_1)$ reduced $\bmod n$, and so on. At each even subscript x_{2x} we calculate $\gcd(x_{2s} - x_s, n)$ and immediately upon obtaining a number greater than 1 we are done.

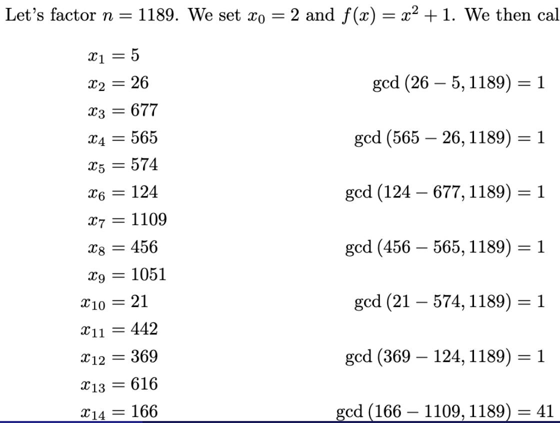
Note: The gcd we find is not necessarily our hypothesized p, however p is a divisor of it, and it is not uncommon to actually obtain a prime.

Example: Let's factor $n = 1111$. We set $x_0 = 2$ and $f(x) = x^2 + 1$. We then calculate:

$$\begin{aligned}
x_1 &= 5 & & \\
x_2 &= 26 & & \gcd(26 - 5, 1111) = 1 \\
x_3 &= 677 & & \\
x_4 &= 598 & & \gcd(598 - 26, 1111) = 11
\end{aligned}$$

Pollard's ρ Algorithm

How about $n = 1189$, $x_0 = 2$ and $f(x) = x^2 + 1$? What is the possible divisor of n? Tips: focus on even position and *gcd* for the subtraction between $2i, i$. Ans: 41.
And where is the cycle? Ans: in the positions: $x_{14} = x_7$, $x_{15} = x_8$, $x_{16} = x_9$......we will have equal values *mod* 41.



Let's factor $n = 1189$. We set $x_0 = 2$ and $f(x) = x^2 + 1$. We then calculate:

$$\begin{aligned}
x_1 &= 5 & & \\
x_2 &= 26 & \gcd(26-5, 1189) &= 1 \\
x_3 &= 677 & & \\
x_4 &= 565 & \gcd(565-26, 1189) &= 1 \\
x_5 &= 574 & & \\
x_6 &= 124 & \gcd(124-677, 1189) &= 1 \\
x_7 &= 1109 & & \\
x_8 &= 456 & \gcd(456-565, 1189) &= 1 \\
x_9 &= 1051 & & \\
x_{10} &= 21 & \gcd(21-574, 1189) &= 1 \\
x_{11} &= 442 & & \\
x_{12} &= 369 & \gcd(369-124, 1189) &= 1 \\
x_{13} &= 616 & & \\
x_{14} &= 166 & \gcd(166-1109, 1189) &= 41
\end{aligned}$$

Pollard's ρ Algorithm

- The average iterations required to find p are at most $\sqrt{p}$.
- There is a trade off between the probability of finding a prime factor and the time complexity of the algorithm.
- For a success probability of 0.5, the expected running time of the algorithm is $\mathcal{O}(n^{\frac{1}{4}})$. F.Y.I. If we compute $gcd(x_i - x_j, n)$ for every pair $1 \leq i < j \leq l$, then l is roughly $\sqrt{p}$. Since $p \leq \sqrt{n}$, then we have $l = n^{1/4}$.
- This is however a heuristic assessment, no rigorous proof of the time complexity of this algorithm have been provided yet.

Algebra & Cryptography CSE3230

Lecture 15 – Message integrity, digital signatures

Stjepan Picek

Recap – Previous Lecture

- Rabin’s and ElGamal’s cryptosystems
- Fermat’s little theorem as a probabilistic primality test
- Pollard’s ρ factorization algorithm

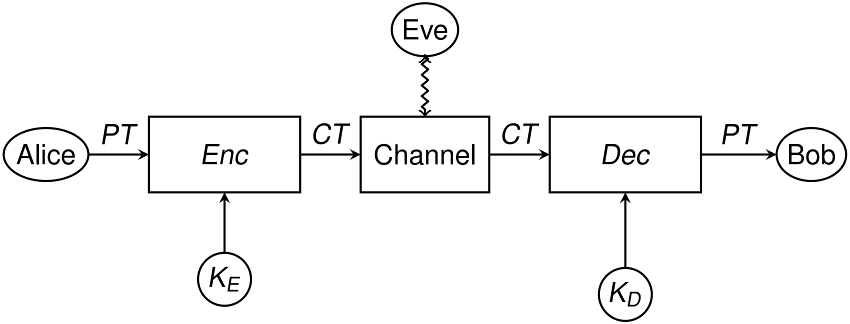
This Lecture

- Digital Signatures
  - Message Integrity and Digital Signatures
  - The RSA Digital Signature Scheme
  - The ElGamal Digital Signature Scheme

Outline of this Lecture

The Basic Scenario – Recall

- Up to now, we only considered a scenario where Eve is *listening* on the channel, i.e. she is a *passive adversary*



- What if Eve is also able to *inject* information in the channel?

Active Adversaries

- There are several possible attacks that Eve can mount as an active adversary:
  - *Alteration*: Eve modifies the information that Alice sends to Bob
  - *Impersonation*: Eve sends information to Bob in such a way that Bob thinks that the sender is Alice
  - *Deviation*: Eve diverts the information sent to Bob by Alice to a third party, Charlie
- How to ensure *message integrity* in light of such attacks?
- Remark that cryptosystems only protect the *confidentiality* of the message, not its integrity!
- There are two approaches to ensure integrity: *Message Authentication Codes* (MACs) in the symmetric-key setting and *digital signatures* in the public-key setting

Digital Signature Schemes

Definition

A *digital signature scheme* is a quintuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$, where:

- $\mathcal{P}$ is the set of all possible *messages* that can be signed
- $\mathcal{A}$ is the set of all possible *signatures*
- $\mathcal{K}$ is the set of all possible *keys*
- $\mathcal{S} = \{sig : \mathcal{P} \to \mathcal{A}\}$ is the set of *signature functions*
- $\mathcal{V} = \{ver : \mathcal{P} \times A \to \{\texttt{TRUE},\ \texttt{FALSE}\}\}$ is the set *verification functions*

For each $K \in \mathcal{K}$ there is $sig_K \in \mathcal{S}$ and $ver_K \in \mathcal{V}$ such that for all $x \in \mathcal{P}$ and $y \in \mathcal{A}$, it holds that

$$ver_K(x,y) = \begin{cases} \texttt{TRUE} & \text{, if } y = sig_K(x) \\ \texttt{FALSE} & \text{, if } y \neq sig_K(x) \end{cases}$$

Digital Signature Schemes

- The pair (x, y) is called a *signed message*
- The signing and verification functions sig_K and ver_K must be computable with polynomial-time algorithms
- Differences with respect to *autograph* (that is, hand made) signatures:
  - The signature y is separated from document x
  - The signature is not always the same: it depends upon the document, so it is different for different documents
- The digital signature *authenticates* the originator of the message: only who knows a certain secret information (the key K) can produce the signature of the message

Attack Types on Digital Signatures

Similarly to cryptosystems, attacks on digital signatures schemes are classified with respect to the information that Eve has:

- *Known Key Attack*: Eve only knows Alice's public key, that is, the verification function ver_K
- *Known Message Attack*: Eve possesses a list of messages previously signed by Alice, i.e. a set of pairs

$$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$$

  where the x_i are messages and y_i are signatures such that $ver_K(x_i, y_i) = \mathtt{TRUE}$ for all $i \in \{1, \cdots, n\}$
- *Chosen Message Attack*: Eve is able to choose a set of messages and to obtain from Alice the corresponding signatures

Adversarial Goals in Digital Signatures

Possible goals of Eve are the following:

- *Total break*: somehow, Eve manages to determine Alice's secret key K (and thus her signature function sig_K)
- *Selective forgery*: when observing a message x, Eve is able to create a signature y for x such that $ver_K(x, y) = \texttt{TRUE}$ with non-negligible probability
- *Existential forgery*: Eve is able to create a valid signature for at least one message x. In other words, she can determine a pair (x, y) such that $ver_K(x, y) = \texttt{TRUE}$

Digital Signatures from Public-Key Cryptosystems

- A public key cryptosystem may be used as a scheme for digital signature
- We sign by using the *private* key
- We verify by using the *public* key
- Hence, only who has the private key is able to sign
- Everybody can verify the signature of a message by using the public key of the presumed signer
- In this way, we can authenticate the originator of a message by using only a public key cryptosystem

Digital Signatures without Encryption

- Assume that Alice wants to send a signed message x to Bob which is *not* encrypted (so confidentiality here is not an issue)
- Alice has an encryption function E_A and a decryption function D_A: she keeps D_A secret, and makes E_A public
- To send the signed message x, Alice computes $D_A(x)$ and sends it to Bob
- To verify Alice's signature, Bob takes the public encryption algorithm E_A and checks that

$$E_A(D_A(x)) = x$$

  and he accepts the signature as valid if and only if this equality holds

Digital Signatures with Encryption

- Assume now that Alice wants to send a signed and encrypted message m to Bob
- Alice has a (public) encryption function E_A and a (private) decryption function D_A
- Bob also has (public) encryption function E_B and a (private) decryption function D_B
- To send the signed and encrypted message x, Alice could compute $D_A(E_B(x))$ and send it to Bob
- First Bob removes Alice's signature, by applying function E_A, and then decrypts, using his own decryption function D_B:

$$D_B(E_A(D_A(E_B(x)))) = x$$

- What is the problem with this approach?

Digital Signatures with Encryption

- Eve also knows the public encryption function E_A!
- She can apply E_A to $D_A(E_B(x))$, and then use her own private decryption D_E to impersonate Alice, sending to Bob $D_E(E_B(x))$
- The basic problem is that the channel is not *authenticated*
- The correct solution is to reverse the process (sign-then-encrypt)
- Alice first signs the message x with D_A and then she encrypts it for Bob: $E_B(D_A(x))$
- In this way, only Bob can extract $D_A(x)$, and Eve cannot substitute D_A with her own D_E

Outline of this Lecture

The RSA Signature Scheme

- The RSA encryption scheme can be straightforwardly used for digital signatures
- We define:
  - $n = p \cdot q$ where p, q are two large prime numbers
  - The messages and signatures space is $\mathcal{P} = \mathcal{A} = \mathbb{Z}_n$
  - The keyspace is $\mathcal{K} = \{(n, p, q, e, d) : ed \equiv 1 \bmod \phi(n)\}$
  - (n, e) is the public key, the values p, q and d are the private key
- Given $K = (n, p, q, e, d)$, Alice can sign a message $x \in \mathbb{Z}_n$ as

$$y = x^d \bmod n$$

- Bob accepts Alice's signature y on x if and only if:

$$x \equiv y^e \bmod n$$

Existential Forgery under Known Key Attack

- Remark: the RSA digital signature scheme is not randomized
- Any given message $x \in \mathbb{Z}_n$ maps to a unique signature y, since modular exponentiation is a permutation over $\mathbb{Z}_n$
- This allows Eve to easily perform an existential forgery as follows:
  1. Eve chooses a random $y' \in \mathbb{Z}_n$
  2. She then computes $x = (y')^e \bmod n$
- It is then easy to see that y' is a valid signature for x
- Note that, however, it does not seem feasible to *first* choose the message x and *then* the signature y'
- This would imply that RSA is insecure

Existential Forgery under Known Message Attack

- This attack is based on the property that the RSA encryption scheme is *homomorphic* with respect to multiplication:

$$Enc_{K_E}(x_1) \cdot Enc_{K_E}(x_2) = Enc_{K_E}(x_1 \cdot x_2)$$

- Suppose that Eve observed two message/signature pairs (x_1, y_1) and (x_2, y_2) sent by Alice to Bob
- Then, Eve can forge the third signature $y_3 = y_1 \cdot y_2$ for the message $x_3 = x_1 \cdot x_2$, since by homomorphism we have that

$$y_3 = y_1 \cdot y_2 = x_1^d \cdot x_2^d \bmod n = (x_1 \cdot x_2)^d \bmod n = x_3^d \bmod n$$

- Thus, the RSA digital signature scheme is also vulnerable to existential forgery under a known message attack

Outline of this Lecture

The ElGamal Signature Scheme

- Published in 1985 by Taher ElGamal
- It is different from the public-key cryptosystem published by the same researcher
- A modified version has been adopted as a standard (DSA - Digital Signature Algorithm) by NIST
- It is randomized: there are many valid signatures for a plaintext message
- The security is based on the assumption that computing discrete logarithms is unfeasible

Messages and Signatures Spaces

Formalization of the signature scheme:

- Let p be a prime number
- Let g be a generator of the multiplicative group $\mathbb{Z}_p^*$, i.e. $\mathbb{Z}_p^* = \{g^0 \cdots, g^{p-2}\}$
- The messages to be signed are elements of $\mathbb{Z}_p^*$, i.e. $\mathcal{P} = \mathbb{Z}_p^*$
- The signatures are pairs (γ, δ) with $\gamma \in \mathbb{Z}_p^*$ and $\delta \in \mathbb{Z}_{p-1}$. Thus, we have $\mathcal{A} = \mathbb{Z}_p^* \times \mathbb{Z}_{p-1}$

Key Generation and Signing

Assume that Alice wants to sign a message $x \in \mathbb{Z}_p^*$; first of all she generates her own pair of keys:

- Alice chooses a private key a, with $0 < a < p-1$
- She computes $\beta = g^a \bmod p$
- The public key is the triple (p, g, β)

To sign $x \in \mathbb{Z}_p^*$, she does the following:

- Alice chooses at random $k \in \mathbb{Z}_{p-1}^*$
- She computes $\gamma = g^k \bmod p$ and $\delta = (x - a\gamma)k^{-1} \bmod p-1$
- The pair (γ, δ) is the signature of x

Remark: from $\delta = (x - a\gamma)k^{-1} \bmod p-1$ we obtain that

$$x = a\gamma + k\delta \bmod p-1$$

Signature Verification

- Bob accepts the signature as valid if and only if $\beta^{\gamma}\gamma^{\delta} \equiv g^{x} \bmod p$
- In fact, assume that the signature has been produced correctly; then:

$$\beta^{\gamma}\gamma^{\delta} \equiv g^{a\gamma} \cdot g^{k\delta} \equiv g^{x} \bmod p$$

- The congruence $g^{a\gamma+k\delta} \equiv g^{x} \bmod p$ holds if and only if

$$a\gamma + k\delta \equiv x \bmod p-1$$

  that is, if and only if γ and δ have been computed as in the previous slide
- Hence, Bob accepts and rejects correctly

Example

- Suppose that $p = 467$ and $g = 2$ (which is a generator of $\mathbb{Z}_{467}^*$)
- Alice chooses $a = 127$ as private key. Then, β equals

$$\beta = g^a \bmod p = 2^{127} \bmod 467 = 132$$

- Assume now that Alice wants to sign $x = 100$, and she randomly chooses $k = 213$
- Remark that $\gcd(213, 466) = 1$ and $213^{-1} \bmod 466 = 431$
- γ and δ are computed as

$$\gamma = g^k \bmod p = 2^{213} \bmod 467 = 29$$
$$\delta = (x - a\gamma)k^{-1} \bmod p - 1 = (100 - 127 \cdot 29) \cdot 431 \bmod 466 = 51$$

- Hence, the signature for $x = 100$ is $(29, 51)$

Example

- Suppose that Bob wants to verify that the signature $(29, 51)$ for $x = 100$ has been produced by Alice
- Since he has p, g, β (which are public), he computes

$$\beta^{\gamma}\gamma^{\delta} \bmod p = 132^{29} \cdot 29^{51} \bmod 467 = 189$$

- Then, he finally verifies that

$$g^{x} \bmod p = 2^{100} \bmod 467 = 189$$

- Thus, Bob concludes that the signature is valid

Security Analysis

- Assume that Eve wants to produce a signature for the message x, without knowing the value of the private key a
- If Eve chooses γ and wants to compute the correspoding value of δ, she must compute the discrete logarithm $\log_\gamma g^x \beta^{-\gamma}$
- On the other hand, if Eve chooses δ and wants to compute γ, she must solve this congruence with respect to γ:

$$\beta^\gamma \gamma^\delta \equiv g^x \bmod p$$

- This problem is different from discrete logarithm, but still no polynomial time algorithm is known that solves it
- Also, if Eve chooses γ and δ and wants to compute a corresponding value for x, she must compute the discrete logarithm $\log_g \beta^\gamma \gamma^\delta$

Existential Forgery under Known-Key Attack

- Remark that Eve can compute γ, δ and x together
- Indeed, she can write $\gamma = g^i \beta^j \bmod p$, with $0 \leq i, j \leq p-2$
- The verification condition is then:

$$g^x \equiv \beta^\gamma \cdot (g^i \cdot \beta^j)^\delta \bmod p$$

  which is equivalent to:

$$g^{x-i\delta} \equiv \beta^{\gamma+j\delta} \bmod p$$

- If we impose $x - i\delta \equiv 0 \bmod p-1$ and $\gamma + j\delta \equiv 0 \bmod p-1$, then this congruence holds

Existential Forgery under Known Key Attack

- Given i and j, if $\gcd(j, p-1) = 1$ then it is easy to find values for x, γ and δ that satisfy $x - i\delta \equiv 0 \bmod p-1$ and $\gamma + j\delta \equiv 0 \bmod p-1$:

$$\gamma = g^i \cdot \beta^j \bmod p$$
$$\delta = -\gamma \cdot j^{-1} \bmod p-1$$
$$x = -\gamma \cdot i \cdot j^{-1} \bmod p-1$$

- By construction, (γ, δ) is a valid signature for x

Example

- Suppose again that $p = 467$, $g = 2$ and $\beta = 132$, and assume that Eve chooses $i = 99$ and $j = 179$
- Then, $j^{-1} \bmod p-1 = 151$, and she computes γ, δ, x as:

$$\gamma = 2^{99} \cdot 132^{179} \bmod 467 = 117$$
$$\delta = -117 \cdot 151 \bmod 466 = 41$$
$$x = -117 \cdot 99 \cdot 151 \bmod 466 = 331$$

- It is then easy to see that $(117, 41)$ is a valid signature for $x = 331$, since:

$$132^{117} \equiv 303 \bmod 467$$

- And finally

$$2^{331} \equiv 303 \bmod 467$$

Existential Forgery under Known Message Attack

- Suppose that Eve observes on the channel the message x and the corresponding signature (γ, δ) produced by Alice
- Eve chooses h, i, j such that $0 \leq h, i, j \leq p-2$ and $\gcd(h\gamma - j\delta, p-1) = 1$, and she computes:

$$\begin{aligned}
\lambda &= \gamma^h g^i \beta^j \bmod p \\
\mu &= \delta\lambda(h\gamma - j\delta)^{-1} \bmod p-1 \\
x' &= \lambda(hx + i\delta)(h\gamma - j\delta)^{-1} \bmod p-1
\end{aligned}$$

- It can then be verified that

$$\beta^\lambda \lambda^\mu \equiv g^{x'} \bmod p$$

- Thus, (λ, μ) is a valid signature for x'

Drawbacks of the Basic ElGamal Scheme

- Total break of the basic ElGamal scheme seems unfeasible since it involves computing discrete logarithms
- But we just showed how this scheme is vulnerable to two types of existential forgeries:
  - The first under a known-key attack
  - The second under a known-message attack
- There is also a practical drawback to this basic scheme, as well as all others based on public-key primitives:
- The signature has the same size as the message to sign!
- This problem, together with the existential forgeries in ElGamal scheme, can be addressed by using *hash functions*

Hash Functions

- *Hash functions* are an important primitives at the base of several cryptographic protocols
- They compute a *fingerprint* (or *message digest*) of the data given as input
- The fingerprint is a small string of fixed length, e.g.
  - 128 bits in MD5
  - 160 bits in SHA-1
  - 224-512 bits in SHA-2 and SHA-3
- They can be used to verify the integrity of messages
- They are also used in digital signatures: instead of directly signing the message x, we sign its fingerprint $h(x)$

Hash Functions in Digital Signatures

Assume that Alice wants to sign a message x using a public key cryptosystem. She can perform the following steps:

1. Compute $h(x)$, where h is a hash function
2. Compute $D_A(h(x))$, where D_A is her private decryption function
3. Sends to Bob the signed message $(x, D_A(h(x)))$

To verify the signature, Bob will:

1. Compute $x' = h(x)$
2. Compute $y = E_A(D_A(h(x)))$, where E_A is Alice's public encryption function
3. Accept x as valid if and only if $y = h(x)$