CISPA - Saarland
Helmholtz Center for Information Security
Department of Cryptography

Script

Cryptography

Summer 2020

Lecturer:
Prof. Dr. Nico Döttling

Documentation:
Christian Schmidt
Guido Battiston

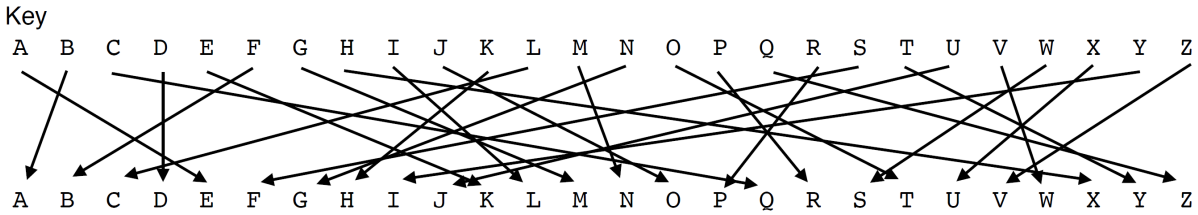
Contents

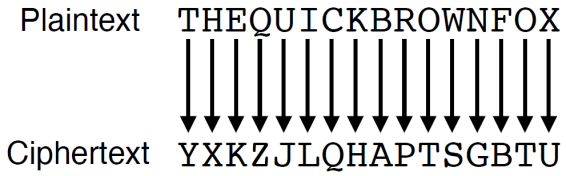
Part I

Basics

Historical Ciphers

1.1 Substitution Ciphers





- One of the oldest cipher in the world, used in the bible (Atbash)
- $\#Keys = 26! \gg 2^{86}$

1.2 Caesar Cipher

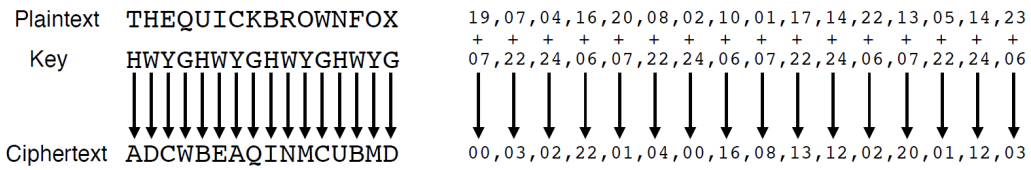
- Key is a fixed substitution table which shifts every letter by 3
- Encryption and decryption are as in the substitution cipher

Shift Cipher

- Generalization of Ceasar's Cipher: Variable Shift
- $\#Keys = 26$

1.3 Vigènere Cipher

- Variant of Caesar/Shift Cipher
- Polyalphabetic Cipher: Not same shift for every plaintext character
- $\#Keys = 26^n$

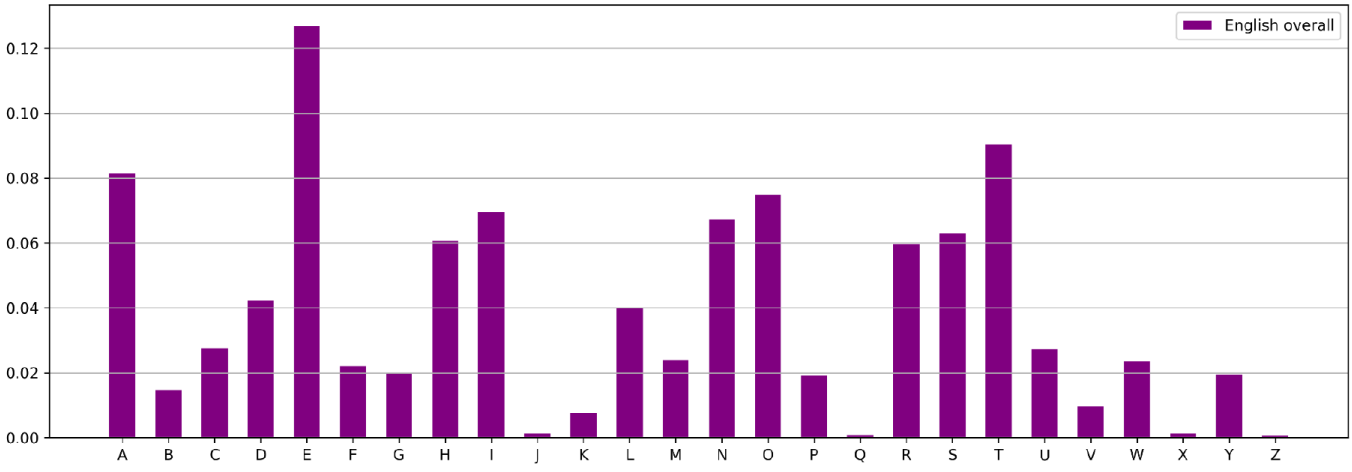


1.4 Cryptanalysis

- Scytale: Curvature of leather belt is give away
- Caesar Scheme: No key! Insecure once the method of encryption is known
- Shift Cipher: Key-space too small! Try all 26 keys
- What about the substitution and Vigènere ciphers?

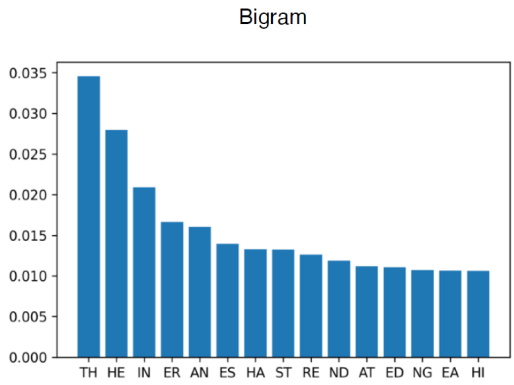
1.5 Frequency Analysis

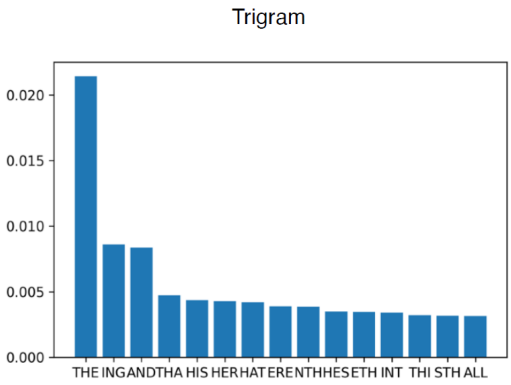
- Idea: Short piece of English text has similar statistics as English language as a whole
- Count letter frequencies in natural language
- Used for Vigènere Ciphers



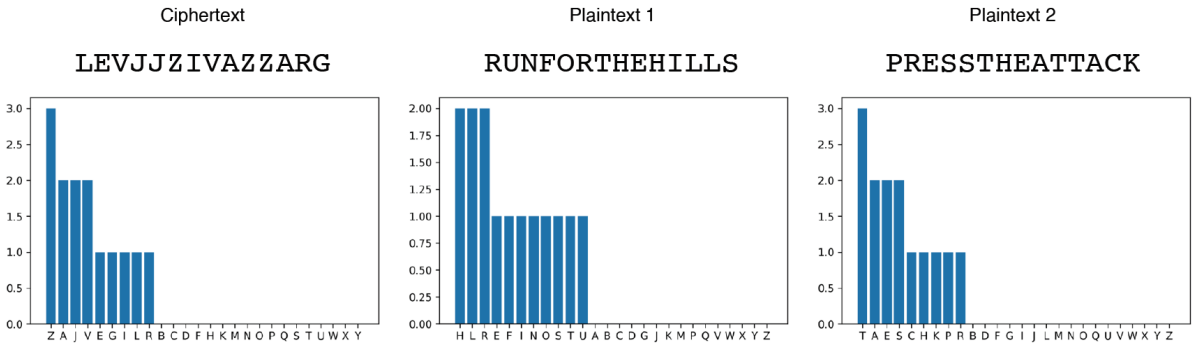
1.6 Cryptanalysis of the Substitution Cipher

- Natural language also has frequent character constellations
- Generalize character frequencies: Bigrams, trigrams … N-grams
- Plaintext recovery requires a lot of context information and guessing





- Plaintext recovery is strongest possible attack
- Requires context information
- More prior knowledge?
- Maximum context information: Ciphertext encrypts one of two messages
- Structural Property of Cipher: Same letters always encrypt to same letter



1.7 Summary

- Historical Ciphers were based on simple character substitution patterns
- Typically small keys spaces
- Can be broken with simple statistical analysis
- Security relied mostly on the fact the method of encryption was not known to the adversary

1.8 Enigma

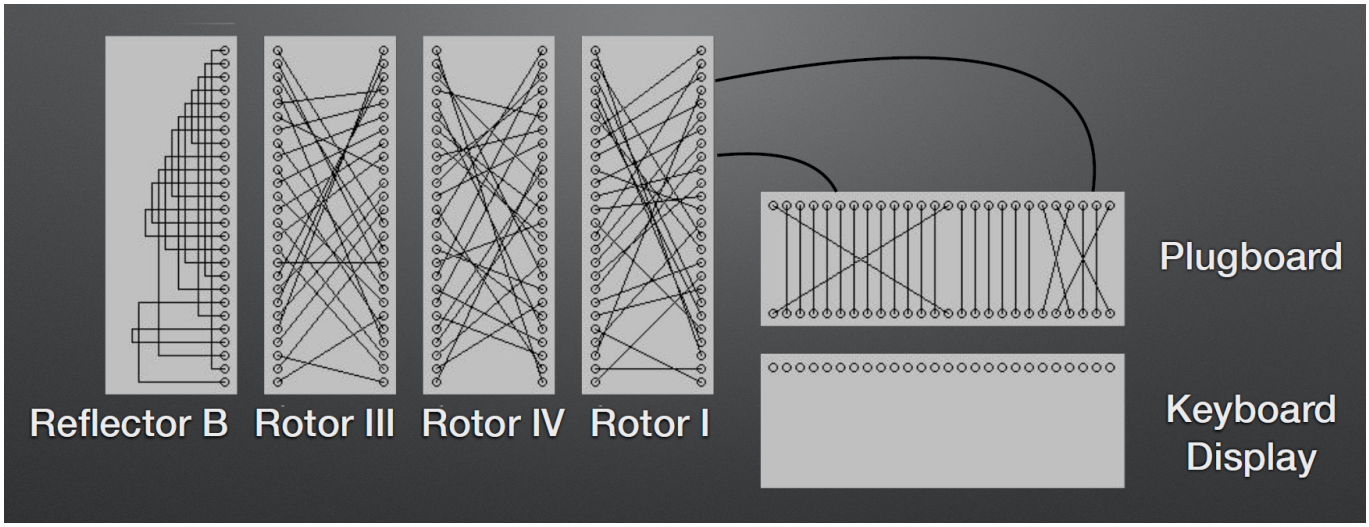
- Enigma was a family of cipher machines used before and during WWII
- Several of those were used by German forces during WWII
- Cryptanalysis of these machines focused huge Allies effort
- In particular, Turing and Blechtley park

1.8.1 General Architecture of Enigma

- Rotor machine with input keyboard and output device
- Includes several layers of transformations to perform encryption
- Encryption is performed letter by letter
- The transformations evolve with time to prevent direct frequency analysis
- The cipher was an involution

1.8.2 Basic principle

- Mix of mechanical and electrical technology
- With the light output display : electric circuit from keyboard to lights
- Goes through several stages into and out of the machine
- Mechanical evolution after every key is released



1.8.3 Key Size

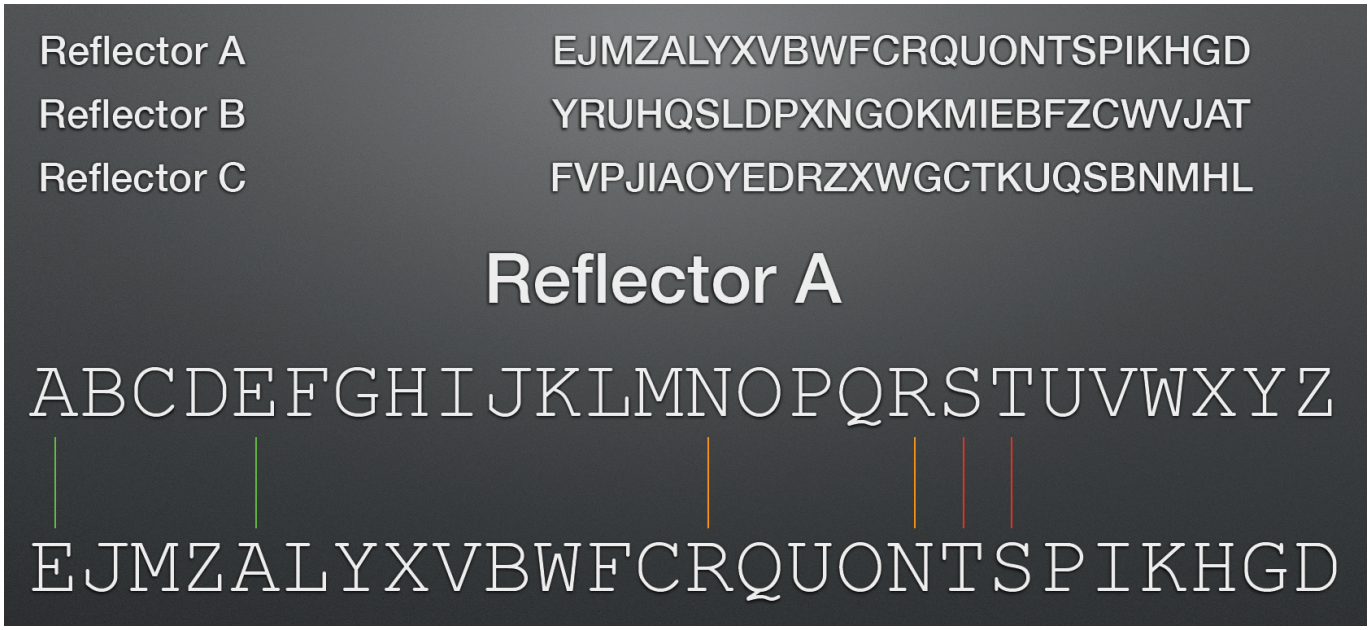
- Per message key:
  - 17.576 possibilities (with 3 rotors) ≈ 14 bits
  - 456.976 possibilities (with 4 rotors) ≈ 19 bits
- Daily key (4 rotor case)≈ 78 bits:
  - 2 choices of reflector
  - 2 choices of fourth rotor
  - 336 choices of first three rotors (in set of 8)
  - Ring setting : 456.976 possibilities
  - Arbitrary plugboard pairing (up to 13 plugs): 532.985.208.200.576 possibilities

Note: There is a dependency between daily and message key
Total key size ≈ 87.5 bits

1.8.4 Digital rewriting of Enigma

- View the encryption as a sequence of (evolving) permutations on letters
  - Step 1: Swap letters according to plugboard
  - Step 2, 3, 4, (5): Apply permutations corresponding to each rotor in order
  - Mid Step: Apply (involutive) permutation of the reflector
  - Step (-5), -4, -3, -2 : Apply inverse rotor permutations
  - Step -1: Swap letters according to plugboard
- Evolve Rotors state

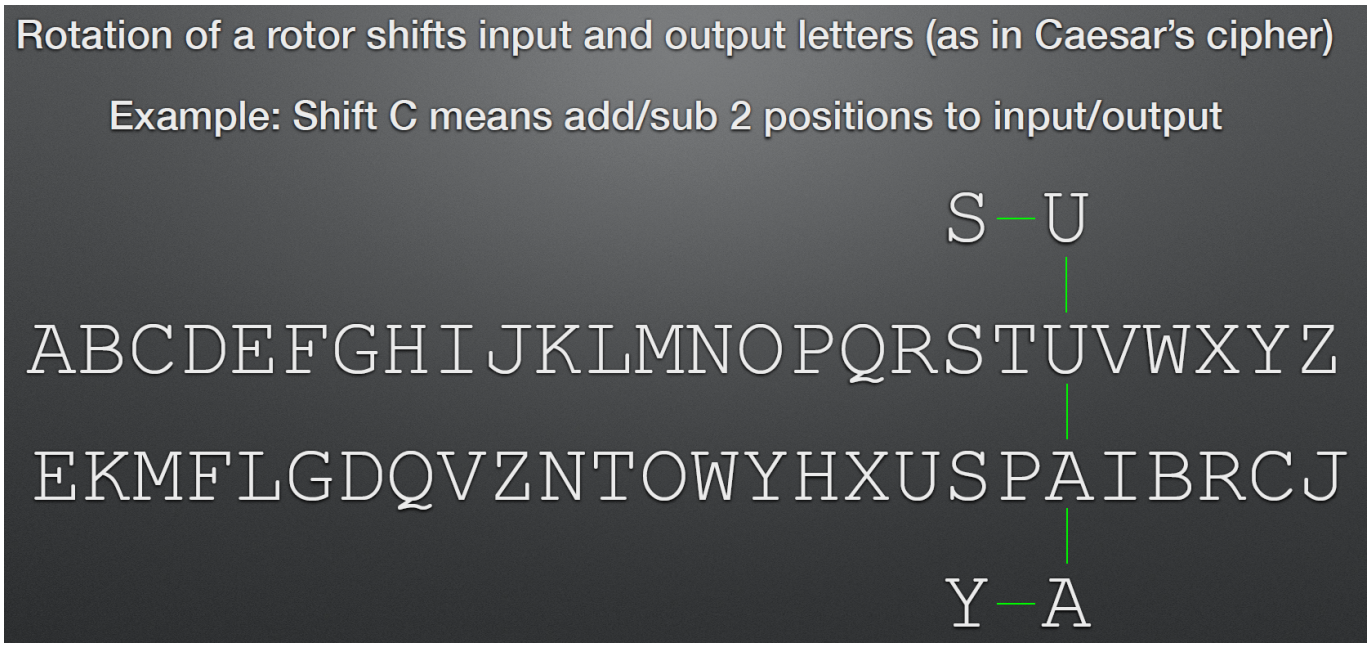
1.8.5 Reflector Permutations



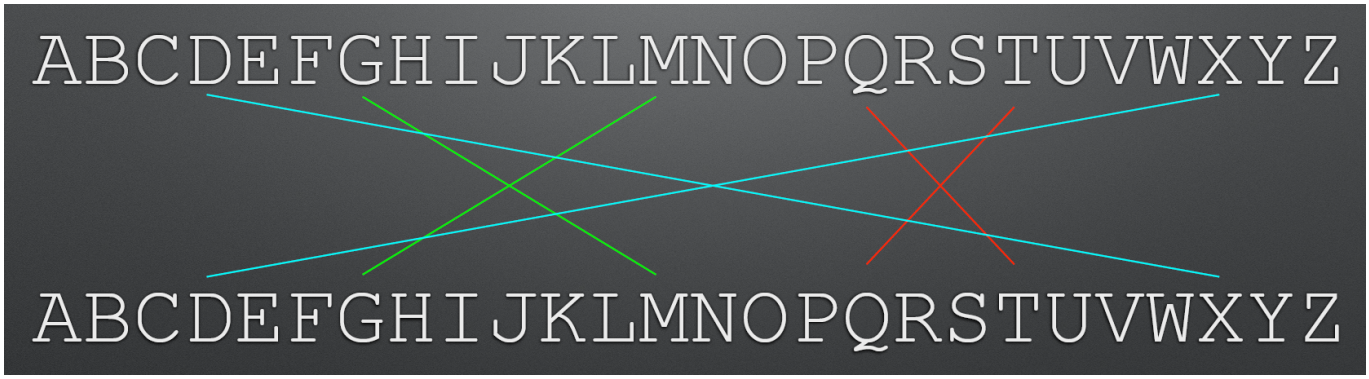
1.8.6 Rotor Permutations

		Notch
Rotor I	EKMFLGDQVZNTOWYHXUSPAIBRCJ	Q
Rotor II	AJDKSIRUXBLHWTMCQGZNPYFVOE	E
Rotor III	BDFHJLCPRTXVZNYEIWGAKMUSQO	V
Rotor IV	ESOVPZJAYQUIRHXLNFTGKDCMWB	J
Rotor V	VZBRGITYUPSDNHLXAWMJQOFECK	Z
Rotor VI	JPGVOUMFYQBENHZRDKASXLICTW	Z and M
Rotor VII	NZJHGRCXMYSWBOUFAIVLPEKQDT	Z and M
Rotor VIII	FKQHTLXOCBJSPDZRAMEWNIUYGV	Z and M
Rotor Beta	LEYJVCNIXWPBQMDRTAKZGFUHOS	None
Rotor Gamma	FSOKANUERHMBTIYCWLQPZXVGJD	None

1.8.7 Rotor permutations (after rotation)



1.8.8 Plugboard



1.8.9 Rotor Evolution

- Basic evolution: After every key release, rotate right rotor by 1 letter
  - State A goes to B, B to C, … Y to Z and Z to A
- Second rotor: If right rotor in notch pos. before change, rotate second rotor
- Third rotor: If second rotor in notch pos., rotate third AND second rotors
- Fourth rotor (optional): Never rotates

1.8.10 A simple property of Enigma

- At every time t, the current transformation is an involution with no fixed point
- Proof:

- By inspection, this is true for reflectors.
- Let π denote the reflector permutation and σ be the current permutation of rotors/plugboard. Then, the global permutation is $\sigma^{-1} \circ \pi \circ \sigma$.
- It is an involution since $(\sigma^{-1} \circ \pi \circ \sigma) \circ (\sigma^{-1} \circ \pi \circ \sigma) = \sigma^{-1} \circ \pi \circ \pi \circ \sigma = \sigma^{-1} \circ \sigma = Id$.
- No fixed point since $\sigma^{-1} \circ \pi \circ \sigma(X) = X \implies \pi \circ \sigma(X) = \sigma(X)$ (impossible for π).

1.8.11 Modern view on Enigma

- Trivially broken with modern standards
- Example attack 1 (Chosen plaintext distinguisher): Encrypt AAA….AAA (100 letters): Enigma encryption never contains A while random string of the same length has A with prob > 98
- Example attack 2 : The message is malleable letter by letter
- Example attack 3 (Chosen plaintext attack): To decrypt challenge cipher text, simply feed it as plaintext to an encryption blackbox. Because of the involution property, get original plaintext back.
- A simple key recovery faster than exhaustive search
- Important : The cipher has a fixed part and a time varying part. More precisely at time t it can be written as $F^{-1} \circ V_t \circ F$.
  - The fixed part comprised the plugboard and the initial position of first rotor.
  - The varying part is the rotors, reflectors and varies with rotations.
- Key size for F is 26 x 532.985.208.200.576 possibilities $\approx$ 53.6 bits.
- Key size for V_t is 2 x 2 x 336 x 265 possibilities $\approx$ 33.9 bits.

1.8.12 Recovering V_t by "cheap" brute force

- We can ”fingerprint” V_t with no information on F
- Encrypt AAAAA...AAAAAA to $L_1 L_2 ... L_n$ where $L_i = F^{-1} \circ V_i \circ F(A)$
- Note that $L_i = L_j \Leftrightarrow V_i(X) = V_j(X)$ where $X = F(A)$
- Enumerate X and the key of the sequence V_t.
- Cost $\approx$ 38.6 bits (with 4 rotos) - At first glance, $+log_2(n)$ but avoidable

1.8.13 Practical Version

- Data collection: Encrypt a sequence of 46 consecutive A — discard first 3
- Fingerprint the 43 letters truncated encrypted sequence as follows:
  - To each of the last 12 letters:
    - Associate the position of the first letter in the first 31 that matches it
    - If no match, associate the value 0
  - This fingerprint has 60 bits and doesn't depend on F (except for $F(A)$)
  - Enough to identify the varying part V (almost) uniquely
  - Once V is obtained, recovering F fully is easy

Principles

2.1 Lessons from Historic Ciphers

- Ciphers without keys or small key-space cannot recover from compromise
- Ciphertext should not reveal statistical information about plaintexts
- Ciphertexts should not reveal any (efficiently computable) information about the plaintext
- Further thoughts: Many historic ciphers immediately broken given a single message-ciphertext pair
- Ciphers were designed with limited imagination
- **How do you defend against an adversary who is smarter than you?**

2.2 Kerkhoff's Principle

The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience.

- Designs which violate Kerkhoff's principle hope to achieve Security by Obscurity
- Today this is considered bad cryptographic design
- It is easier to switch a compromised key than a compromised design

2.3 Principles of Modern Cryptography

- **Formal Definitions/Models**
  - What constitutes a successful attack?
  - What is the goal of an attack?
- **Precise Assumptions**
  - Underlying hard problem?
- **Formal Proofs**
  - Show that a successful attack actually violates the assumptions

2.4 Formal Models

- Example: Encryption
- We will generally define security by defining what it means that a scheme is insecure.
  - What is the computational power of the adversary?

  - What information is given to the adversary?
    * Only Ciphertext
    * A message and a ciphertext encrypting this message
    * A ciphertext encrypting a message chosen by the adversary?
    * Access to a “decryption oracle”
  - What is the goal of the adversary?
    * Recover the key
    * Recover the message
    * Recover parts of the message
    * Distinguish encryptions of two messages

2.5 Precise Assumptions

- Should it be impossible to break the cipher only hard?
- How can we mathematically define hard problems?
  - Complexity Theory!
  - Example: Factor large numbers
- A good assumption should be simple to state but hard to break

Precise Assumptions let us

- Falsify an assumption by cryptanalysis
- Build confidence by lack of cryptanalysis
- Compare schemes by their underlying assumptions
- Understand the necessity of the assumption

2.6 Summary

- Kerkhoff’s principle: Only the key should be secret, not the scheme itself
- Modern cryptography evolved into a discipline following scientific principles
- It lets us design schemes secure against adversaries smarter than ourselves!

Probability Theory

3.1 Why Randomness and Probability?

Probability Theory: Mathematical framework for systems which behave in unpredictable or random ways

- **Most Sciences:** Systems too complex for exact modeling, lack of precise measurements
- **Computer Science:** Randomness is a resource
- **Cryptography:** Impossible without randomness!

3.2 How can randomization lead to efficient algorithms?

- Example: Election Winners
- You have an election with 40 million voters and want to know quickly who is the (likely) winner
- You want to know who the winner is without counting all votes
- Idea: Take a **random sample** of 1000 votes determine the winner!

3.3 Probability Spaces

A finite probability space consists of two components:

1. A finite set Ω called the sample space.
2. A probability function $Pr : \Omega \to \mathbb{R}$ satisfying
   - For all $\omega \in \Omega$: $0 \leq Pr[\omega] \leq 1$
   - $\sum_{\omega \in \Omega} Pr[\omega] = 1$

- The $\omega \in \Omega$ are called elementary events or outcomes
- A set $E \subseteq \Omega$ is called event
- For an event E define

$$Pr[E] = \sum_{\omega \in E} Pr[\omega]$$

  where $Pr[\emptyset] = 0$

3.4 Set Operations on events

- The intersection $E_1 \cap E_2$ of two events E_1 and E_2 is interpreted as the logical "and" of two events. Depending on the context, we will use the following notations

$$Pr[E_1 \cap E_2] =: Pr[E_1, E_2] =: Pr[E_1 \wedge E_2] =: Pr[E_1 \; and \; E_2]$$

- The union $E_1 \cup E_2$ of two events E_1 and E_2 is interpreted as the logical "or" of two events. Thus, we will use the following notation

$$Pr[E_1 \cup E_2] =: Pr[E_1 \vee E_2] =: Pr[E_1 \; or \; E_2]$$

- The complement $\bar{E} = \Omega \backslash E$ of an event E is interpreted as the logical "negation" of E. We will use the notation

$$Pr[\bar{E}] =: Pr[\neg E] =: Pr[not \; E]$$

- If $A \subseteq B$ we say that the event A implies the event B and also write $A \Rightarrow B$

3.5 Properties of Probability Spaces

- For all events $A, B \subseteq \Omega$:
  1. $Pr[A \cup B] = Pr[A] + Pr[B] - Pr[A \cap B]$
  2. $Pr[\bar{A}] = 1 - Pr[A]$
  3. If $A \subseteq B$ then $Pr[A] \leq Pr[B]$
- Item 1 implies the **union-bound**:
  For all $A, B \subseteq \Omega$ it holds $Pr[A \cup B] \leq Pr[A] + Pr[B]$

3.6 Conditional Probabilities and Independence

- For two events A and B we define the conditional probability

$$Pr[A|B] = \frac{Pr[A \cap B]}{Pr[B]}, \; Pr[B] > 0$$

- Two events $A, B \subseteq \Omega$ are called independent if

$$Pr[A \cap B] = Pr[A] \cdot Pr[B]$$

- This is equivalent to $Pr[A|B] = Pr[A]$
- By symmetry also to $Pr[B|A] = Pr[B]$

3.7 Bayes' Rule

- Bayes' rule relates the conditional probabilities $Pr[A|B]$ and $Pr[B|A]$
- This is especially useful when analyzing larger processes composed of smaller ones

Bayes' Rule
For all $A, B \subseteq \Omega$, it holds for $Pr[A], Pr[B] > 0$, that

$$Pr[A|B] = \frac{Pr[B|A] \cdot Pr[A]}{Pr[B]}$$

Proof. $Pr[A|B] = \frac{Pr[A \cap B]}{Pr[B]} \cdot \frac{Pr[A]}{Pr[A]} = \frac{Pr[A \cap B]}{Pr[A]} \cdot \frac{Pr[A]}{Pr[B]} = \frac{Pr[B \cap A]}{Pr[A]} \cdot \frac{Pr[A]}{Pr[B]} = \frac{Pr[B|A] \cdot Pr[A]}{Pr[B]}$ □

3.8 Law of Total Probability

- Sometimes we want to infer the probability of an event given conditional probabilities of this event
- This will also be useful to compute/bound probabilities by a case analysis

Law of Total Probability
Let $B_1, ..., B_n$ be events that partition the probability space Ω, i.e. $B_1 \cup ... \cup B_n = \Omega$ and $B_i \cap B_j = \emptyset$ for $i \neq j$. Then it holds for every event A that

$$\sum_{i=1}^{n} Pr[A \cap B_i] = Pr[A], \text{ with } Pr[A \cap B_i] = Pr[A|B_i] \cdot Pr[B_i]$$

Proof. Preconditions:

1. $Pr[E_1 \cup E_2]Pr[E_1] + Pr[E_2]$ if $E_1 \cap E_2 = \emptyset$, and
2. $Pr[E_1 \cup ... \cup E_n] = \sum_{i=1}^{n} Pr[E_i]$ if $E_i \cap E_j = \emptyset$ for $i \neq j$.
3. $Pr[A \cap B_i] = Pr[A|B_i] \cdot Pr[B_i]$

We know that $A = A \cap \Omega = A \cap \Big(\bigcup_{i=1}^{n} B_i \Big) = \bigcup_{i=1}^{n} A \cap B_i$.
So it follows $Pr[A] = Pr\Big[\bigcup_{i=1}^{n} A \cap B_i \Big] \overset{(2.)}{=} \sum_{i=1}^{n} Pr[A \cap B_i] \overset{(3.)}{=} \sum_{i=1}^{n} Pr[A|B_i] \cdot Pr[B_i]$. □

3.9 Example: Bayes and LotP in Action

- Assume 2 in 1000 people in a population have a certain disease
- There exists a test for this disease but it is not flawless
- If you have the disease, then there is a 95% chance that the test returns positive
- Moreover, there is a 3% chance that the test returns positive even if you don t have the disease
- Imagine, you do a test and it returns positive, what is the probability that you actually have the disease?

Remarks:

- The probability 2 1000 = 0.002 is your prior of having the disease, i.e. if you do not get the additional information of test, this is your best guess
- We are interested in the posterior probability

Solution:

- Let s model this with a probability space
- Let A be the event that you have the disease. We know that $Pr[A] = 0.002$
- Let B be the event that the test returns positive. We know that $Pr[B|A] = 0.95$
- We also know that $Pr[B|\bar{A}] = 0.03$
- We want to know the posterior $Pr[A|B]$

$Pr[A|B] = \frac{Pr[B|A] \cdot Pr[A]}{Pr[B]} = \frac{Pr[B|A] \cdot Pr[A]}{Pr[B|A] \cdot Pr[A] + Pr[B|\bar{A}] \cdot Pr[\bar{A}]} = \frac{Pr[B|A] \cdot Pr[A]}{Pr[B|A] \cdot Pr[A] + Pr[B|\bar{A}] \cdot (1 - Pr[A])} \approx 0.06$

If you do a test and it returns positive, the probability that you actually have the disease is 6%!

3.10 Summary 1

- Probability spaces model randomized processes
- Conditional probabilities tell us a posteriori probabilities given that some event has happened
- Bayes' rule and the law of total probability simplify working with conditional probabilities

3.11 Random Variables

- In general, any function $X : \Omega \to D$ for some domain D is called a (D-valued) **random variable**
- We can now define events with respect to a random variable.
- Let $x \in D$ define the event $E_x = \{\omega \in \Omega : X(\omega) = x\} \subseteq \Omega$
- We write the shorthand $X = x$ for E_x
- We call the function $F(x) = Pr[E_x] = Pr[X = x]$ the probability distribution of X

3.11.1 Independence of random variables

- Consider two random variables $X \in D_X$ and $Y \in D_Y$, which means $X : \Omega \to D_X$ and $Y : \Omega \to D_Y$
- We say that X and Y are **independent**, if it holds for all $x \in D_X$ and $y \in D_Y$ that the events $X = x$ and $Y = y$ are independent
- I.e. $Pr[X = x, Y = y] = Pr[X = x] \cdot Pr[Y = y]$

3.11.2 Examples

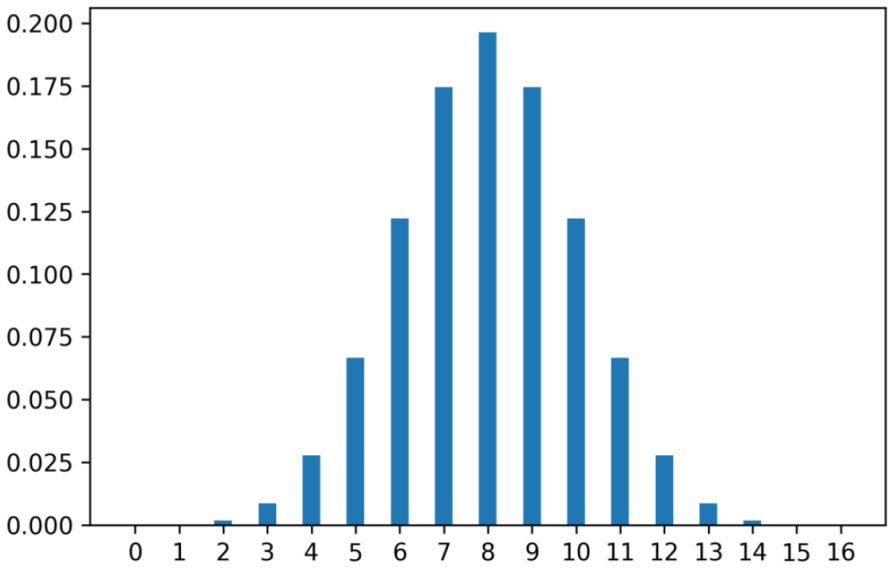
- **Identity function:**
  $X : \Omega \to \Omega$. In this case $Pr[X = \omega] = Pr[\omega]$.
- **Projections:**
  Let $\Omega \subseteq D_1 \times D_2$, then $X : \Omega \to D_1$, $(x, y) \mapsto x$ and $Y : \Omega \to D_2$, $(x, y) \mapsto y$ are projections.
- **Reverse direction:**
  Given two sample spaces Ω_1 and Ω_2, we can form a new sample space $\Omega = \Omega_1 \times \Omega_2$.
  Then the projections $X : \Omega \to \Omega_1$ and $Y : \Omega \to \Omega_2$ are independent random variables.
- We say that a random variable $X : \Omega \to D$ is **uniform** on D or an **uniform distribution**, if for every $x \in D$ it holds $Pr[X = x] = \frac{1}{|D|}$
- **Sum of two dice rolls:**
  Let $\Omega = \{1, ..., 6\} \times \{1, ..., 6\}$, then $S : \Omega \to \mathbb{Z}$ with $(x, y) \mapsto x + y$ is a sum of two dice rolls.
- **Indicator random variable:**
  For an event E we can define the random $X_E : \Omega \to \{0, 1\}$ such that $X_E(\omega) = 1$ if $\omega \in E$ and $X_E(\omega) = 0$ if $\omega \notin E$

3.11.3 Defining Probability Spaces via Random Variables

- From now on we will omit the function notation for random variables
- Moreover, from now on we will also omit defining the sample space Ω explicitly
- Instead, we implicitly define Ω via a set of random variables
- General idea: The probability space is defined via a few independent "base" random variables
- All other random variables of interest depend on these deterministically
- When constructing a system that uses randomization, we do not need to keep track of the sample space, it is define "on the fly" as we introduce new random variables
- If X is chosen uniformly random from some domain D, we will use the notation $X \leftarrow_\$ D$ instead of the function notation

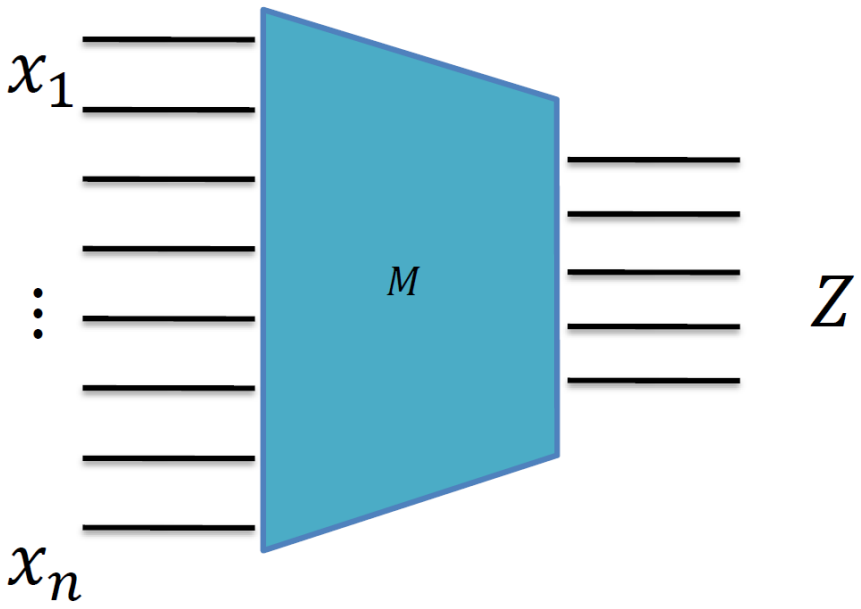
3.11.4 Examples

- Let $X_1, ..., X_n$ be independent random variables such that each X_i is uniform on $\{0, 1\}$. We say $X_1, ..., X_n$ are **iid uniform on** $\{0, 1\}$ for short
- Then $X = (X_1, ..., X_n)$ is uniform on $\{0, 1\}^n$
- $Y = X_1 + ... + X_n$ follow a distribution which is called the Bernoulli distribution



3.11.5 Sampling Algorithms

- In general: We say a random variable Z is **sampleable**, if there exists an (efficient) algorithm M such that $Z = M(X_1, ..., X_n)$, where $X_1, ..., X_n$ are iid uniform on $\{0, 1\}$
- $X_1, ..., X_n$ are called the **random coins** or just **coins** of M



3.12 Real-Valued Random Variables and Expectations

- We call random variables $X : \Omega \to \mathbb{R}$ **real-valued**
- We call $Supp(X) = X(\Omega) \subseteq \mathbb{R}$ the **support** of X
- Note that $Supp(X)$ is finite as Ω is finite
- For a real-valued random variable X, we can define an **expectation**

$$E[X] = \sum_{x \in Supp(X)} x \cdot Pr[X = x]$$

3.12.1 Linearity of Expectation

- Expectations are characteristic of random variables
- Powerful tool to analyze random variables constructed from other random variables

Linearity of Expectation
Let X and Y be two real-valued random variables and $\alpha \in \mathbb{R}$ be a constant.
Then it holds that

1. $E[\alpha \cdot X] = \alpha \cdot E[X]$
2. $E[X + Y] = E[X] + E[Y]$

Note: X and Y don't need to be independent

Proof. Proof of the linearity properties of expectation

1. To show: $E[\alpha \cdot X] = \alpha \cdot E[X]$

$$E[\alpha \cdot X] = \sum_{z \in Supp(\alpha \cdot X)} z \cdot Pr[\alpha \cdot X = z], \text{ with } \alpha \neq 0$$

   Now we make the following substitution: $z = \alpha \cdot z'$

$$\begin{aligned} E[\alpha \cdot X] &= \sum_{z'} \alpha \cdot z' \cdot Pr[\alpha \cdot X = \alpha \cdot z'] \\ &= \alpha \cdot \sum_{z'} \cdot Pr[X = z'] \\ &= \alpha \cdot E[X] \end{aligned}$$

2. To show: $E[X + Y] = E[X] + E[Y]$

$$\begin{aligned} E[X] &= \sum_{x \in Supp(x)} x \cdot Pr[X = x] \\ &= \sum_{x \in S} x \cdot Pr[X = x], \text{ with } Supp(X), Supp(Y), Supp(X + Y) \leq S \end{aligned}$$

$$
\begin{aligned}
E[X+Y] &= \sum_{z \in Supp(X+Y)} z \cdot Pr[X+Y=z] \\
&= \sum_{y \in Supp(Y)} \sum_{z \in Supp(X+Y)} z \cdot Pr[X = z-y \wedge Y = y] \\
&= \sum_{y \in S} \sum_{x \in S} (x+y) \cdot Pr[X=x, Y=y], \text{ with } z = x+y \\
&= \sum_{y \in S} \sum_{x \in S} x \cdot Pr[X=x, Y=y] + \sum_{y \in S} \sum_{x \in S} y \cdot Pr[X=x, Y=y] \\
&= \sum_{x \in S} x \cdot \sum_{y \in S} Pr[X=x, Y=y] + \sum_{y \in S} y \cdot \sum_{x \in S} Pr[X=x, Y=y] \\
&= \sum_{x \in S} x \cdot Pr[X=x] + \sum_{y \in S} y \cdot Pr[Y=y] \\
&= E[X] + E[Y]
\end{aligned}
$$

□

3.12.2 Example

- Let $X_1, ..., X_n$ be independently distributed random variables on $\{0,1\}$ such that for all i it holds $Pr[X_i = 1] = p$ for a constant $p \in [0,1]$
- What is the expectation of $Y = X_1 + ... + X_n$?
  $E[Y] = E[X_1 + ... + X_n] = E[X_1] + ... + E[X_n] = n \cdot p$
  $Pr[X_i = 1] = p \Rightarrow E[X_i] = \sum_{b \in \{0,1\}} b \cdot Pr[X_i = b] = Pr[X_i = 1] = p$

3.13 A little detour to Algebra: Finite Groups

3.13.1 Groups

A set G together with a binary operation $\circ : G \times G \to G$ is called a group, if the following conditions are met:

1. **Associativity:** For all $f, g, h \in G$ it holds that $(f \circ g) \circ h = f \circ (g \circ h)$
2. **Identity Element:** There exists a special element $e \in G$ such that for all $g \in G$ it holds $e \circ g = g \circ e = g$
3. **Inverses Element:** For every $g \in G$ there exists an element $g^{-1} \in G$ such that $g \circ g^{-1} = e$

- A group $(G, \circ)$ is called **commutative (abelian) group** if for all $g, h \in G$ it holds $g \circ h = h \circ g$
- We say a group $(G, \circ)$ is **finite** if G is finite as a set

3.13.2 Examples of Groups

- The integers under addition: $(\mathbb{Z}, +)$
- Modular integers under addition: $(\mathbb{Z}/n\mathbb{Z}, + \ mod \ n)$
- The non-zero rational numbers under multiplication: $(\mathbb{Q}\backslash\{0\}, \cdot)$
- The binary set under XOR: $(\{0,1\}, \oplus)$
- Binary vectors under XOR: $(\{0,1\}^n, \oplus)$ with $\oplus$ bitwise

3.13.3 Random Variables in Groups

- Let U be uniformly distributed on a **finite group** G
- I.e. for all $g \in G$ it holds $Pr[U = g] = \frac{1}{|G|}$
- Fix an $z \in G$, what is the probability distribution of $U \circ z$?
- $\Rightarrow$ If U is uniformly distributed on G and Z is a random variable supported on G and independent of U, then $U \circ Z$ is also uniformly distributed on G
- In particular for $G = \{0, 1\}$ and $\circ = \oplus$, $U \oplus Z$ is distributed uniformly random.

3.14 Summary 2

- Random variables are "the right way" to define and work with probability spaces
- In CS, we usually define all variables depending on some uniform and independent "base variables"
- For real valued random variables we can compute expectations
- Expectations are linear
- XOR-ing a uniformly random and independent variable onto something else results again in a uniform distribution.

Perfect Secrecy

4.1 Historical Context

- Until World War 2 the design principles of ciphers were military secrets and not subject to open research
- During World War 2 especially the allies recruited many (civil) mathematicians and engineers in their cryptanalytic efforts
- It was realized that the central weaknesses of contemporary ciphers was that ciphertexts preserved certain properties/invariants of plaintext

  -> E.g. Substitution Cipher: Ordered histogram of character frequencies was preserved

  -> E.g. Enigma: We can rule out that a ciphertext encrypts a certain message
- After WW2: Beginning of Information Age; Seminal works of Shannon '48,'49

4.2 Encryption Schemes (Shannon)

An encryption scheme consists of three algorithms: $(KeyGen, Enc, Dec)$ and three sets $\mathcal{K}, \mathcal{M}, \mathcal{C}$

- $KeyGen$: A randomized algorithm which outputs a key $K \in \mathcal{K}$
- $Enc(K, m)$: A randomized algorithm which takes a key $K \in \mathcal{K}$ and message $m \in \mathcal{M}$ as input and outputs a ciphertext $c \in \mathcal{C}$
- $Dec(K, c)$: A deterministic algorithm which takes as a key $K \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ as input and outputs a message $m \in \mathcal{M}$

4.3 Encryption Schemes: Correctness

- An encryption scheme is called correct if it holds for all messages $m \in \mathcal{M}$ that

$$Pr[Dec(K, Enc(K, m)) = m] = 1, \text{ where } K \leftarrow KeyGen()$$

- This probability is taken over all random choices involved, namely over the choice of the coins of $KeyGen$ and Enc

4.4 Security

- How do we define security?
- **Goal:** Security definition should guarantee that if the adversary sees a ciphertext, he shouldn't learn anything about the message
- **Idea:** Ciphertext should be independent of the message.
- What should the distribution of the message be? Any distribution!

4.5 Perfect Secrecy: Part I

Definition 1 (Perfect Screcy 1)**.** *An encryption scheme* $(KeyGen, Enc, Dec)$ *is perfectly secret, if it holds for every random variable* M *supported on the message space* $\mathcal{M}$ *that* M *and* $C = Enc(K, M)$ *are independent, where* K *is chosen by* $K \leftarrow KeyGen()$*.*

I.e. it holds for all messages $m \in \mathcal{M}$ *and every ciphertext* $c \in \mathcal{C}$ *with* $Pr[Enc(K, M) = c] > 0$ *that*

$$Pr[M = m \mid Enc(K, M) = c] = Pr[M = m]$$

4.5.1 One-Time Pad (OTP)

$$\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^n$$

- $KeyGen()$: Choose $K \leftarrow_\$ \{0,1\}^n$ and output K
- $Enc(K, m)$: Output $c \leftarrow K \oplus m$
- $Dec(K, c)$: Output $m \leftarrow K \oplus c$
- Note: Key as long as message!

Correctness. $Dec(K, c) = Dec(K, Enc(K, m)) = K \oplus K \oplus m = 0^n \oplus m = m$ □

4.5.2 One-Time Pad: Perfect Secrecy

- $K \leftarrow_\$ \{0,1\}^n$ and $K \oplus m$ is uniformly random
- M supported of $\mathcal{M}$, with M and K are independent
- $M \oplus K$ is distributed uniformly random

Fix any distribution M, and show that for all m, c:

$$Pr[M = m \mid Enc(K, M) = c] = Pr[M = m]$$

Proof.

$$\begin{aligned} Pr[M = m \mid Enc(K, M) = c] &= Pr[M = m \mid K \oplus M = c] \\ &= \frac{Pr[K \oplus M = c \mid M = m] \cdot Pr[M = m]}{Pr[K \oplus M = c]} \text{(Baye's Rule)} \end{aligned}$$

Let's consider $Pr[K\oplus \mid M = m]$:

$$\begin{aligned} Pr[K\oplus \mid M = m] &= Pr[K \oplus M = c] \\ &= \sum_{m'} Pr[K \oplus M = c \mid M = m'] \cdot Pr[M = m'] \\ &= \sum_{m'} \frac{1}{2^n} \cdot Pr[M = m'] \\ &= \frac{1}{2^n} \cdot \sum_{m'} Pr[M = m'] = \frac{1}{2^n} \end{aligned}$$

So it follows that

$$Pr[M = m \mid K \oplus M = c] = \frac{\frac{1}{2^n} \cdot Pr[M = m]}{\frac{1}{2^n}} = \frac{\frac{1}{2^n}}{\frac{1}{2^n}} \cdot Pr[M = m] = Pr[M = m]$$

□

4.5.3 One-Time Pad in any Finite Group

$$\mathcal{K} = \mathcal{M} = \mathcal{C} = G$$

- $KeyGen()$: Choose $K \leftarrow_{\$} G$ and output K
- $Enc(K, m)$: Output $c \leftarrow K \oplus m$
- $Dec(K, c)$: Output $m \leftarrow K^{-1} \oplus c$

4.5.4 Why "One-Time Pad"?

- Invented by Vernam in 1917
- Proven secure by Shannon in 1949
- The proof of security only holds if the key is used at most once
- If the key is reused, this essentially becomes a Vigenere cipher!
- Thus, the OTP consumes large amounts of key material
- Can we do better?

4.6 Summary 1

- We defined the syntax and the correctness property for encryption schemes
- We provided a definition of perfect security which captures the intuition that ciphertexts should reveal no information about the plaintext
- We showed that the one time pad is perfectly secret

4.7 Discussion: Perfect Secrecy

- This definition (Definition 1) is **not easy to work with** when designing larger protocols
- Also, it does not immediately generalize to "weaker" security notions that allow for more efficient schemes.
- Another Approach: In the context of historic ciphers, we also have seen distinguishing attacks
- The adversary knows that a ciphertext encrypts one out of two messages and has to find out which one it is
- Is this a minimal attack goal?

4.8 Perfect Secrecy: Part II

Definition 2 (Perfect Secrecy 2)**.** *An encryption scheme* $(KeyGen, Enc, Dec)$ *is perfectly secret, if it holds for all* $m, m' \in \mathcal{M}$ *and all* $c \in \mathcal{C}$ *that*

$$Pr[Enc(K, m) = c] = Pr[Enc(K, m') = c]$$

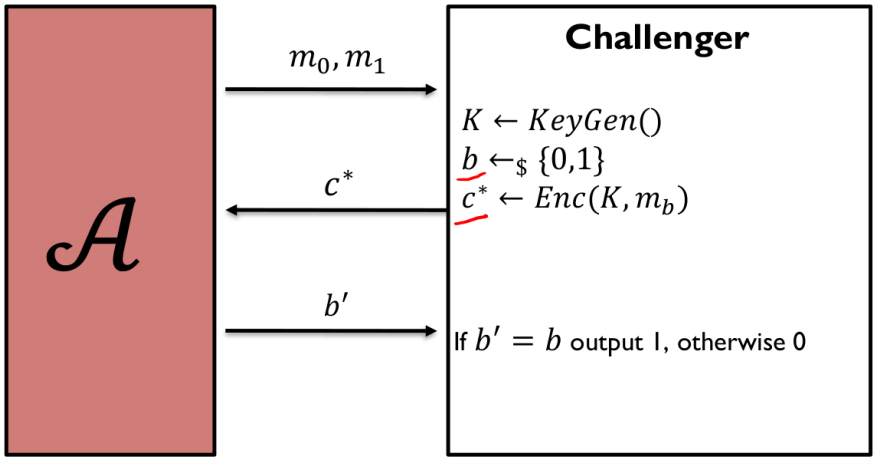
where $K \leftarrow KeyGen()$

4.8.1 Discussion

- The definitions of perfect secrecy provided so far (1 & 2) only concerned the distributions of messages and ciphertexts
- There is no explicit mention of an adversary in these definitions
- Recall from Lecture 1.3 that we want to study security notions against adversaries with different resources
- Let's make the adversary explicit

Definition 3 (Perfect Screcy 3)**.** *We first define some expressions:*

- *Let* $(KeyGen, Enc, Dec)$ *be an encryption scheme*
- *Let* $\mathcal{A}$ *be a (possibly unbounded) randomized algorithm, called the adversary*
- *Let* $IND_{\mathcal{A}}$ *be a random variable which describes the output of the challenger at the end of the experiment*
- *Consider this experiment:*



- *We call this the ciphertext indistinguishability experiment*

An encryption scheme $(KeyGen, Enc, Dec)$ *is perfectly secret, if it holds for* ***every adversary*** $\mathcal{A}$ *that*

$$Pr[IND_{\mathcal{A}} = 1] = \frac{1}{2}$$

where the probability is taken over all random choices of the experiment.

Theorem 1. *Definition 1, Definition 2 and Definition 3 of perfect secrecy are equivalent.*

Proof. Proof concept: $((2) \Rightarrow (1)) \rightarrow ((3) \Rightarrow (2)) \rightarrow ((1) \Rightarrow (3))$

(2) $\Rightarrow$ (1) For all m, m', c show

$$Pr[Enc(K, m) = c] = Pr[Enc(K, m') = c] \Rightarrow \delta_c := Pr[Enc(K, m) = c]$$

Let M be a distribution of messages. Need to show for all m, c that

$$Pr[M = m \mid Enc(K, M) = c] = Pr[M = m]$$

With Baye's rule it follows, that

$$Pr[M = m \mid Enc(K, M) = c] = \frac{Pr[Enc(K, M) = c \mid M = m] \cdot Pr[M = m]}{Pr[Enc(K, M) = c]}$$

Now we consider the changed conditional probability

$$Pr[Enc(K, M) = c \mid M = m] = Pr[Enc(K, m) = c \mid M = m] = Pr[Enc(K, m) = c] = \delta_c$$

The denominator can be formed as follows

$$\begin{aligned} Pr[Enc(K, M) = c] &= \sum_{m' \in Supp(M)} Pr[Enc(K, m') = c \mid M = m'] \cdot Pr[M = m'] \\ &= Pr[Enc(K, m') = c] \cdot \sum_{m' \in Supp(M)} Pr[M = m'] \\ &= \delta_c \cdot \sum_{m' \in Supp(M)} Pr[M = m'] = \delta_c \end{aligned}$$

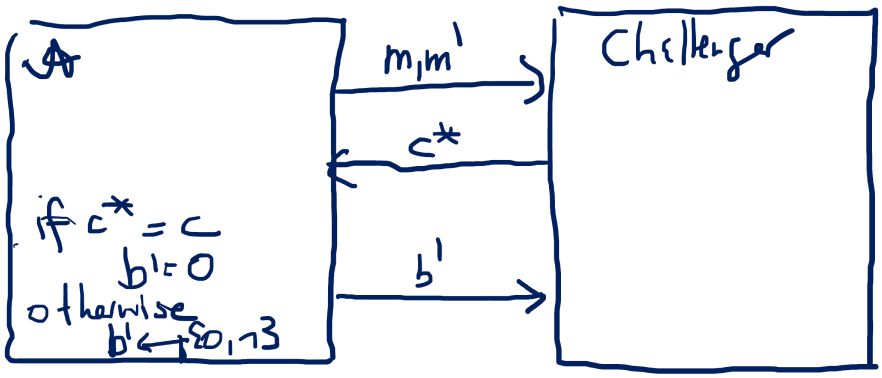
So it follows

$$Pr[M = m \mid Enc(K, M) = c] = \frac{\delta_c \cdot Pr[M = m]}{\delta_c} = Pr[M = m]$$

(3) $\Rightarrow$ (2) Via contraposition:
Assume Definition 2 does not hold, i.e. there exist m, m', c so that

$$Pr[Enc(K, m) = c] > Pr[Enc(K, m') = c]$$

Strategy: Construct $\mathcal{A}$ so that $Pr[IND_{\mathcal{A}} = 1] > \frac{1}{2}$



with $Pr[c^* = c] > 0$

$$Pr[Enc(K, m) = c] > Pr[Enc(K, m') = c] \geq 0$$

With the "law of total probability" it follows that

$$Pr[IND_{\mathcal{A}} = 1] = Pr[IND_{\mathcal{A}} = 1 \mid c^* = c] \cdot Pr[c^* = c] + Pr[IND_{\mathcal{A}} = 1 \mid c^* \neq c] \cdot Pr[c^* \neq c]$$

Now we need to show that $Pr[IND_{\mathcal{A}} = 1 \mid c^* = c]$:
If $c^* = c$ it holds that $IND_{\mathcal{A}} = 1$ if and only if $b = 0$
Wherever $c^* = c$ adversary $\mathcal{A}$ will guess $b' = 0$ (by construction)
$\Rightarrow$ Under the condition $c^* = c$ the events $IND_{\mathcal{A}} = 1$ and $b = 0$ are equivalent.

$$Pr[IND_{\mathcal{A}} = 1 \mid c^* = c] = Pr[b = 0 \mid c^* = c]$$

We will compute $Pr[b = 0 \mid c^* = c]$ using Baye's rule and LOTP

$$Pr[b = 0 \mid c^* = c] = \frac{Pr[c^* = c \mid b = 0] \cdot Pr[b = 0]}{Pr[c^* = c]}$$

We have to compute two things, first

$$Pr[c^* = c \mid b = 0] = Pr[Enc(K, m) = c]$$

and then

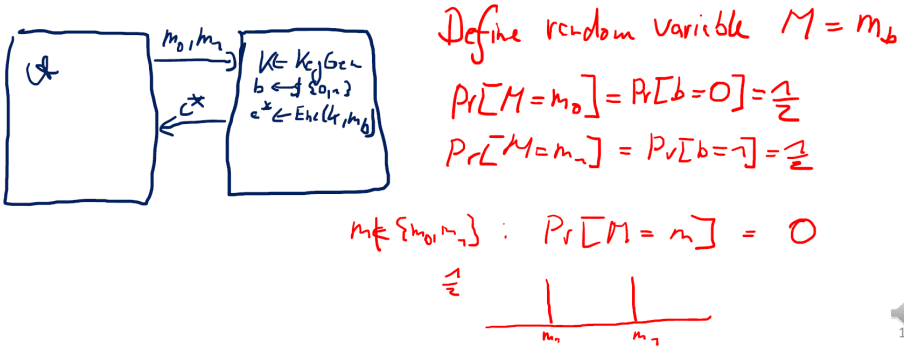
$$\begin{aligned} Pr[c^* = c] &= Pr[c^* = c \mid b = 0] \cdot Pr[b = 0] + Pr[c^* = c \mid b = 1] \cdot Pr[b = 1] \\ &= Pr[Enc(K, m) = c] \cdot \frac{1}{2} + Pr[Enc(K, m') = c] \cdot \frac{1}{2} \\ &= \frac{1}{2} \cdot (Pr[Enc(K, m) = c] + Pr[Enc(K, m') = c]) \end{aligned}$$

Now we want to use the results:

$$\begin{aligned} Pr[b = 0 \mid c^* = c] &= \frac{Pr[Enc(K, m) = c] \cdot \frac{1}{2}}{\frac{1}{2} \cdot (Pr[Enc(K, m) = c] + Pr[Enc(K, m') = c])} \\ &= \frac{1}{1 + \frac{Pr[Enc(K,m')=c]}{Pr[Enc(K,m)=c]}} > \frac{1}{2} \end{aligned}$$

with $\frac{Pr[Enc(K,m')=c]}{Pr[Enc(K,m)=c]} < 1$

(1) $\Rightarrow$ (3) Let $\mathcal{A}$ be an adversary for the IND experiment.
For now assume that $\mathcal{A}$ is deterministic. $\mathcal{A}$ always uses some messages m_0, m_1



So $c^* = Enc(K, m_b) = Enc(K, M)$
Idea: Use Definition 1 to show that c^* is independent of M and therefore of b
First: For $\beta \in \{0, 1\}$ the events $M = m_\beta$ are equivalent.
For all ciphertexts c it holds:

$$Pr[b = \beta \mid c^* = c] = Pr[M = m_\beta \mid Enc(K, M) = c] = Pr[M = m_\beta] = \frac{1}{2}$$

If we recall that $\mathcal{A}$ is deterministic, we can follow that $b' = b'(c^*)$ is a deterministic function of c^*.

$$\begin{aligned} Pr[IND_{\mathcal{A}} = 1] &= Pr[b = b'(c^*)] \\ &= \sum_{c \in \mathcal{C}} Pr[b = b'(c^*) \mid c^* = c] \cdot Pr[c^* = c], \text{ with LOTP} \\ &= \sum_{c \in \mathcal{C}} Pr[b = b'(c) \mid c^* = c] \cdot Pr[c^* = c] \\ &= \sum_{c \in \mathcal{C}} \frac{1}{2} \cdot Pr[c^* = c] = \frac{1}{2} \cdot \sum_{c \in \mathcal{C}} Pr[c^* = c] \\ &= \frac{1}{2} \cdot 1 = \frac{1}{2} \end{aligned}$$

But what about randomized $\mathcal{A}$?
$\mathcal{A}$ takes as input additional random coins $r \leftarrow \{0,1\}^l$ and write as $\mathcal{A}(r)$ to make this explicit.
Idea: For every fixed $\delta \in \{0,1\}^l$, $\mathcal{A}(\delta)$ is a deterministic algorithm.
That means $Pr[IND_{\mathcal{A}(\delta)} = 1] = \frac{1}{2}$ (*), and it follows with the LOTP that

$$\begin{aligned} Pr[IND_{\mathcal{A}(r)} = 1] &= \sum_{\delta \in \{0,1\}^l} Pr[IND_{\mathcal{A}(\delta)} = 1 \mid r = \delta] \cdot Pr[r = \delta] \\ &= \sum_{\delta \in \{0,1\}^l} \frac{1}{2} \cdot Pr[r = \delta], \text{ because of (*)} \\ &= \frac{1}{2} \cdot \sum_{\delta \in \{0,1\}^l} Pr[r = \delta] = \frac{1}{2} \cdot 1 = \frac{1}{2} \end{aligned}$$

□

- For the one time pad, keys are as large as the message and can only be use once.
- Can we construct perfectly secret encryption with short(er) keys? ← No!

Theorem 2 (Shannon's Theorem)**.** *Let $(KeyGen, Enc, Dec)$ be an encryption scheme with key space $\mathcal{K}$,message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. If it is perfectly secure, then*

$$|\mathcal{K}| \geq |\mathcal{M}|$$

Proof. Assume $|\mathcal{K}| < |\mathcal{M}|$
Strategy: Show there exist m_0, m_1 and ciphertext c so that

$$Pr[Enc(K, m_0) = c] > 0$$

$$Pr[Enc(K, m_1) = c] = 0$$

This contradicts Definition 2 of perfect secrecy.

Fix an arbitrary $m_0 \in \mathcal{M}$, a key $K_0 \in \mathcal{K}$, and set

$$c = Enc(K_0, m_0)$$

Define the set

$$S = \{Dec(k, c) \mid k \in \mathcal{K}\} \subseteq \mathcal{M}$$

$\Rightarrow |S| \leq |\mathcal{K}| < |\mathcal{M}|$
$\Rightarrow \mathcal{M} \backslash S \neq \emptyset$, i.e. there exist $m_1 \in \mathcal{M} \backslash S$

We claim there exists no key $K \in \mathcal{K}$ s.t. $c = Enc(K, m_1)$.
If there was such a K, then by correctness of the scheme it holds

$$Dec(K, c) = Dec(K, Dec(K, m_1)) = m_1$$

$\Rightarrow m_1 \in S$
Contradiction to the given condition that $m_1 \in \mathcal{M} \backslash S$ □

4.9 Summary 2

- Two alternate definitions for perfect secrecy.
- All 3 definitions are equivalent.
- Perfectly secret schemes have keys as large as the message which can only be used once.

Part II

Private Key Encryption

Basics of Private Key Encryption

5.1 Limitations of Perfectly Secret Encryption

- Thm of Shannon: In perfectly secret encryption key must be as long as the message and cannot be reused
- But: **Perfect Secrecy is an overkill!**
- Relaxation: Only consider adversaries with realistic computational power
- Also: **Is secrecy alone sufficient? What about other attacks?**

5.2 Computational Security: Encryption

- Back to the problem of secrecy with short keys
- **Idea:** Limit the runtime of adversaries
  - We will relax the notion of perfect secrecy by considering only **computationally bounded** adversaries $\mathcal{A}$
  - Assume that the runtime of adversaries is upper bounded by a realistic upper bound of T operations, in practice often $T = 2^{80}$

5.3 Computationally Secure Encryption

Let's modify the definition of perfect secrecy to only account for T-bounded adversaries

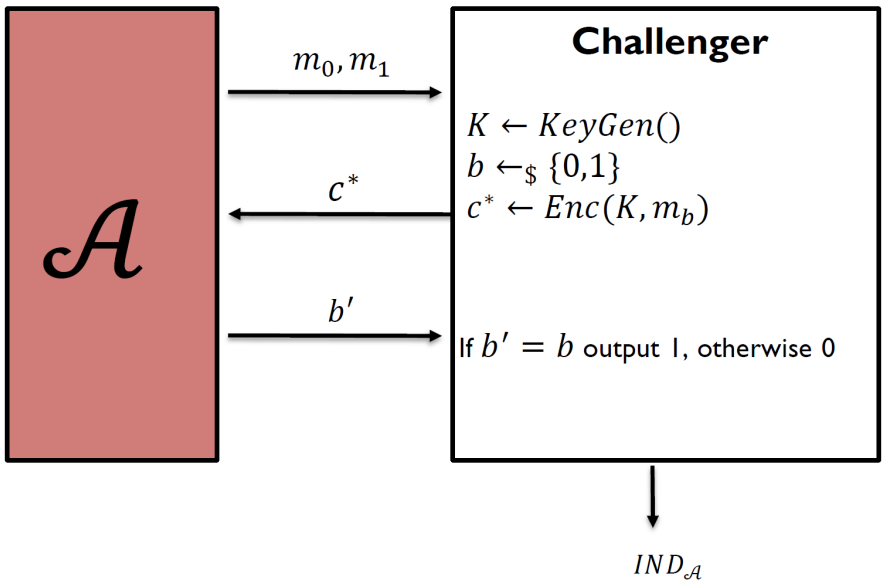
Definition 4 (Computationally Secure Encryption: Attempt 1)**.** *An encryption scheme* $(KeyGen, Enc, Dec)$ *is* $T - IND$ ***secure****, if it holds for* ***every*** T***-bounded adversary*** $\mathcal{A}$ *that*

$$Pr[IND_{\mathcal{A}} = 1] = \frac{1}{2}$$

Proper Definition: T-bounded adversaries with advantage $< \epsilon$

Definition 5 (Computationally Secure Encryption: Concrete Indistinguishability Security)**.** *An encryption scheme* $(KeyGen, Enc, Dec)$ *is* $(T, \epsilon) - IND$ ***secure****, if it holds for* ***every*** T***-bounded adversary*** $\mathcal{A}$ *that*

$$Pr[IND_{\mathcal{A}} = 1] < \frac{1}{2} + \epsilon$$



5.4 Concrete Security

- How do we determine if a scheme is (T, ϵ)-secure?
- In practice: Choose parameters of scheme such that **best known attack** with time complexity T has advantage less than ϵ
- In practice, often suggested parameters are $T = 2^{80}$ and $\epsilon = 2^{-60}$
- What is the unit of T?
  - Milliseconds?
  - Elementary CPU operations?
  - CPU cycles?
- What about other models of computation?
- Concrete security is inherently technology dependent!
- How can we argue about security in any *reasonable* computational model?

5.5 Asymptotic Complexity

- Computational Complexity Theory!
- Recall: In complexity theory problems consist of an infinite number of instances
- The complexity of an algorithm solving a problem is measured as a function of the size of an instance, or more generally a *size-parameter*
- E.g. sorting: Merge sort has complexity $\mathcal{O}(n \cdot log(n))$ to sort a list of n elements
- Recall that we say that an algorithm is efficient, if it runs in time $poly(n)$

5.6 Asymptotic Security

- **Idea of asymptotic security:** Introduce a security parameter λ which characterizes the hardness of breaking a scheme
- We only care about what happens when $\lambda \to \infty$
- $KeyGen$ now takes as explicit input security parameter λ, typically written in unary as 1^λ
- Efficient computation: **probabilistic polynomial time (PPT) algorithms**, i.e. randomized algorithms running in time $poly(\lambda)$
- Thus efficient adversaries = PPT machines
- What about the advantage ϵ?
- Exponentially small?
- Definitely smaller than $\frac{1}{poly(\lambda)}$ for every polynomial and sufficiently large λ!

Definition 6 (Negligible Functions)**.** *A function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is called negligible, if for every polynomial p there exists an $N \in \mathbb{N}$ such that for all $n > N$ it holds that*

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

Remark:

- *It suffices to look at polynomials $p(n) = nc$ for constants c.*
- *We will write $negl(n)$ for an unspecified negligible function.*

5.6.1 Examples

- $f_1(n) = 2^{-n} \to$ NEGLIGIBLE
  because $2^{-n} < n^{-c} \Leftrightarrow -n < -c \cdot log(n) \Leftrightarrow n > c \cdot log(n)$
- $f_2(n) = n^{-100} \to$ NOT NEGLIGIBLE
  because $n^{-c} = \frac{1}{n^c}$
- $f_3(n) = 2^{-log(n)} \to$ NOT NEGLIGIBLE
  because $2^{-log(n)} = \frac{1}{n}$
- $f_4(n) = 2^{-(log(n))^2} \to$ NEGLIGIBLE
  because $2^{-(log(n))^2} < n^{-c} \Leftrightarrow -(log(n))^2 < -c \cdot log(n) \Leftrightarrow (log(n))^2 > c \cdot log(n)$
- $f_5(n) = n^{-log(log(n))} \to$ NEGLIGIBLE
  because $n^{-log(log(n))} < n^{-c} \Leftrightarrow log(log(n)) \cdot log(n) > c \cdot log(n)$
- **Simple Rule:** f is negligible iff $-log(f(n)) \geq \omega(log(n))$

Lemma 1 (Properties of negligible functions)**.** *If v_1, v_2 are negligible functions and p is a polynomial, then*

1. *$v_1 + v_2$ is negligible.*
2. *$p \cdot v_1$ is negligible.*

Proof. Let q be any poly

1. There is at least one N_1, N_2 so that:

$$\forall n > N_1 : v_1(n) < \frac{1}{2 \cdot q(n)} \text{ and } \forall n > N_2 : v_2(n) < \frac{1}{2 \cdot q(n)}$$

   Let $N = max\{N_1, N_2\}$ be the maximum. So it holds

$$\forall n > N : v_1(n) + v_2(n) < \frac{1}{2 \cdot q(n)} + \frac{1}{2 \cdot q(n)} = \frac{1}{q(n)}$$

   So it follows that $v_1 + v_2$ is negligible!

2. Let $q'(n) = p(n) \cdot q(n)$ with p is a polynomial. Then there is at least one N so that:

$$\forall n > N : v_1(n) < \frac{1}{p(n) \cdot q(n)} \Leftrightarrow p(n) \cdot v_1(n) < \frac{1}{q(n)}$$

   So it follows that $p \cdot v_1$ is negligible!

□

Definition 7 (Encryption Schemes - Full Definition)**.**

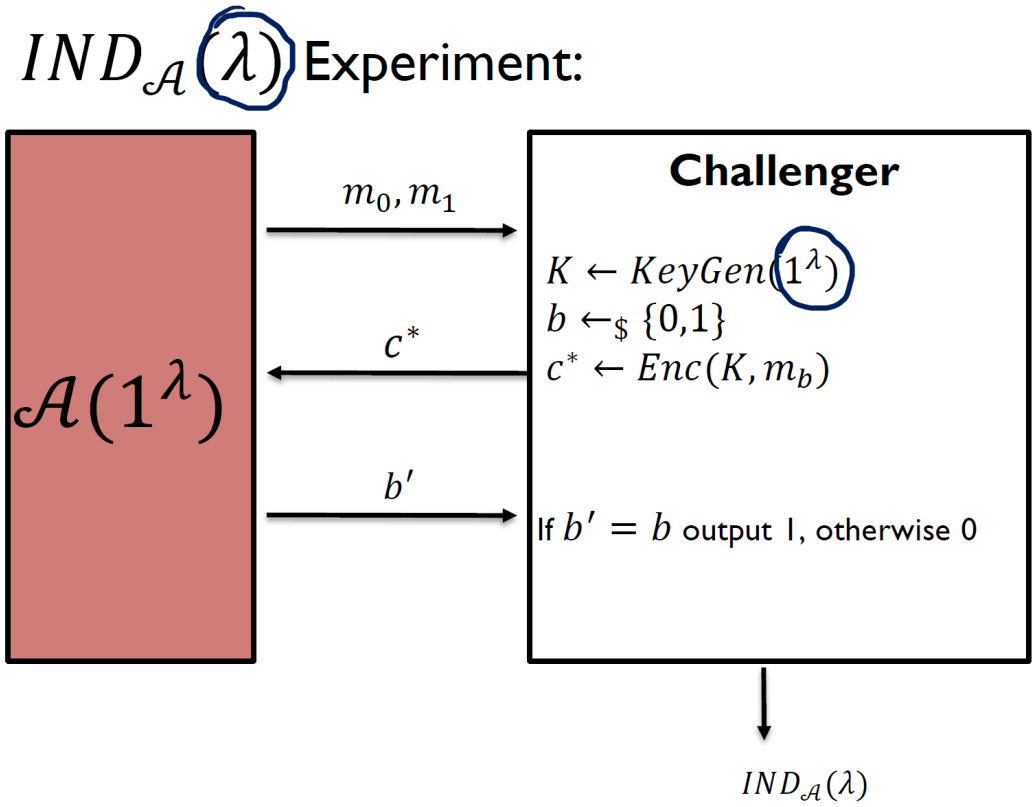
- ***Syntax:***
  An encryption scheme consists of three PPT algorithms: $(KeyGen, Enc, Dec)$
  - $KeyGen(1\lambda)$*: A randomized algorithm which takes as input the security parameter* 1λ *(encoded in unary) and outputs a key* K*.*
  - $Enc(K, m)$*: A randomized algorithm which takes a key* K *and a message* m *as input and outputs a ciphertext* c*.*
  - $Dec(K, c)$*: A deterministic algorithm which takes as a key* K *and a ciphertext* c *as input and outputs a message* m*.*
- ***Correctness:***
  It holds for all $\lambda \in \mathbb{N}$ *and all messages* m *that* $Pr[Dec(K, Enc(K, m)) = m] = 1$*, where* $K \leftarrow KeyGen(1\lambda)$*.*

5.7 Computationally Secure Encryption: Asymptotic Indistinguishability Security

PPT adversaries with negligible advantage

Definition 8 (Asymptotic Indistinguishability Security)**.** *An encryption scheme* $(KeyGen, Enc, Dec)$ *is* IND*-secure, if it holds for every PPT-bounded adversary* $\mathcal{A}$ *there exists a negligible function* v *s.t. for all* $\lambda \in \mathbb{N}$*:*

$$Pr[IND_{\mathcal{A}}(\lambda) = 1] < \frac{1}{2} + v(\lambda)$$



5.8 Discussion

- Asymptotic security allows us to base security of schemes on **standard assumptions** via reductions
- Shortcomings: Asymptotic security does not tell us how to instantiate schemes concretely
- But: Asymptotic security typically is a guarantee that a scheme has no design flaws.
- Attempt at reconciliation: *Tight Security*, i.e. security reductions which provide concrete quantitative guarantees

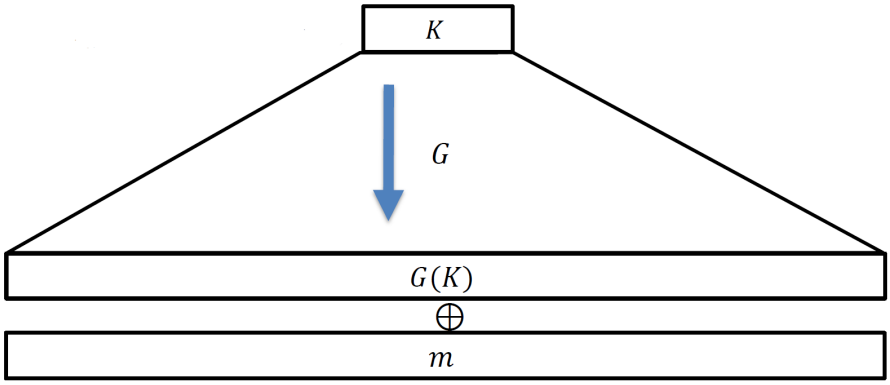
5.9 Summary

- Perfect security is both an overkill and insufficient
- Computational Security: Only consider efficient adversaries, and require their advantage to be small
- Asymptotic security: efficient adversaries = **PPT algorithms**, small advantage = **negligible advantage**

Stream Ciphers

Definition 9 (Stream Ciphers)**.**

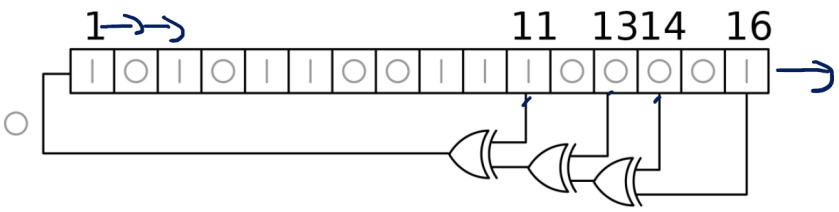
- *Basic Idea: Take a short key, expand it into a long key and use as one-time pad.*



- $KeyGen(1^\lambda)$*: Choose* $K \leftarrow_\$ \{0,1\}^\lambda$*, output* K
- $Enc(K, m)$*: Compute and output* $c \leftarrow G(K) \oplus m$
- $Dec(K, m)$*: Compute and output* $m \leftarrow G(K) \oplus c$

6.1 Pseudorandomness against Simple Statistical Tests

- Idea: $G(K)$ should *behave* like a uniform distribution
- Linear Feedback Shift Registers (LFSR)
- Un biased: On average same number of 0's and 1's in output
- **Runs:** On average same number of 0-runs 000000 as 1-runs 111111
- What other *efficient* statistical test should we account for? All of them!



Definition 10 (Pseudorandom Generators)**.**

- *$G(K)$: deterministic polynomial time algorithm, takes as input $K \in \{0,1\}^\lambda$ and outputs $G(K) \in \{0,1\}^l$, where $l = poly(\lambda)$.*
- *u is uniformly distributed over $\{0,1\}^l$*
- *A distinguisher is an algorithm which outputs a single bit.*
- *G is a pseudorandom generator, if it holds for every PPT distinguisher $\mathcal{D}$ that*
  - *$|Pr[\mathcal{D}(G(k)) = 1] - Pr[\mathcal{D}(u) = 1]| \leq negl(\lambda)$*
  - *where probability is over $K \leftarrow \{0,1\}^\lambda$, $u \leftarrow \{0,1\}^l$ and the random coins of $\mathcal{D}$.*

6.2 Examples

- Modular Congruence generator:
  - P a prime
  - $\mathbb{Z}_P = \{0, ..., P-1\}$
  - $a, b \leftarrow_\$ \mathbb{Z}_P$
  - $x_0 \leftarrow_\$ \mathbb{Z}_P$
  - $s = (a, b, x_0)$
  - $G(s)$: Compute a sequence $x_0, x_1, ..., x_l$ via $x_{i+1} \leftarrow a \cdot x_i + b \ mod \ P$
- Is this a pseudorandom generator?
  - No!
  - Sequence in completely determined by x_0, x_1, x_2

$$a = \frac{x_2 - x_1}{x_1 - x_0} \ mod \ P, \ b = x_1 - a \cdot x_0 \ mod \ P$$

$$x_1 = a \cdot x_0 + b \ mod \ P$$

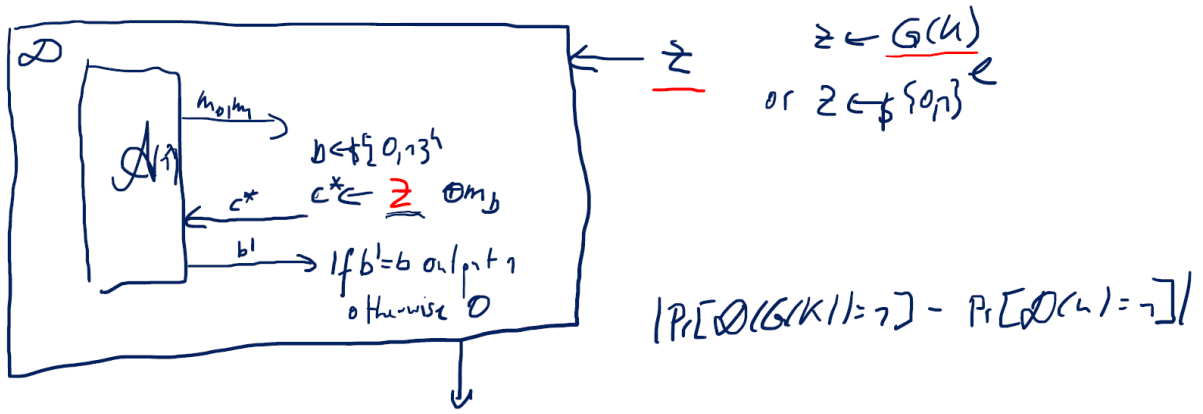
$$x_2 = a \cdot x_1 + b \ mod \ P$$

  - **Distinguishing attack:** Compute a, b from x_0, x_1, x_2 and test for all $i > 2$ if $x_i = a \cdot x_{i-1} + b \ mod \ P$
  - For a uniform sequence this happens only with probability $P^{-(l-3)}$
  - Distinguishing advantage: $1 - P^{-(l-3)}$

Theorem 3 (Security of Stream Ciphers)**.**
If G is a pseudorandom generator, then $(KeyGen, Enc, Dec)$ is IND-secure.

Proof. Proof by Contraposition:
Assume $(KeyGen, Enc, Dec)$ is not IND-secure
$\Rightarrow$ There is at least one PPT $\mathcal{A}$ and a non-negligible ϵ so that

$$Pr[IND_{\mathcal{A}}(\lambda) = 1] = \frac{1}{2} + \epsilon$$



$$Pr[\mathcal{D}(G(K)) = 1] = Pr[IND_{\mathcal{A}}(\lambda) = 1] = \frac{1}{2} + \epsilon$$

But $Pr[\mathcal{D}(u) = 1] = \frac{1}{2}$ and so it follows, that

$$|Pr[\mathcal{D}(G(K)) = 1] - Pr[\mathcal{D}(u) = 1]| = \frac{1}{2} + \epsilon - \frac{1}{2} = \epsilon$$

is non-negligible. Therefore G is not a PRG and so the theorem is proven. □

6.3 Summary

- Cryptographic pseudorandom generators (PRGs) need to fool all statistical tests, not just a few we chose
- Pseudorandom Generators generate distributions that are far from uniform but cannot be distinguished from uniform by any PPT algorithm
- PRGs imply IND secure stream ciphers, for which the key is shorter than the message

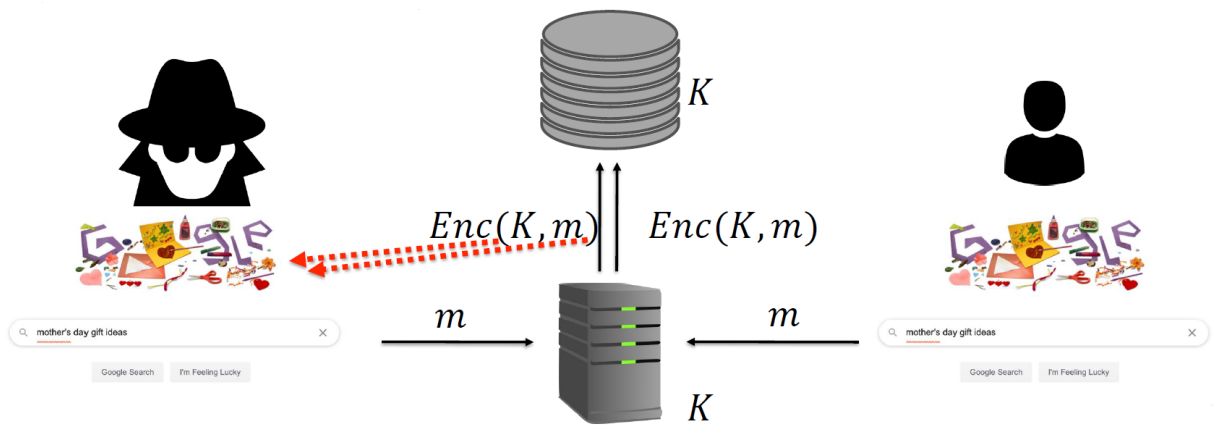
Chosen Plaintext Security

7.1 Reusing Keys

- Stream ciphers have shorter keys than the OTP
- But still keys can only be used once
- How can we make keys reusable?
- First: How should this be modeled in the security definition?

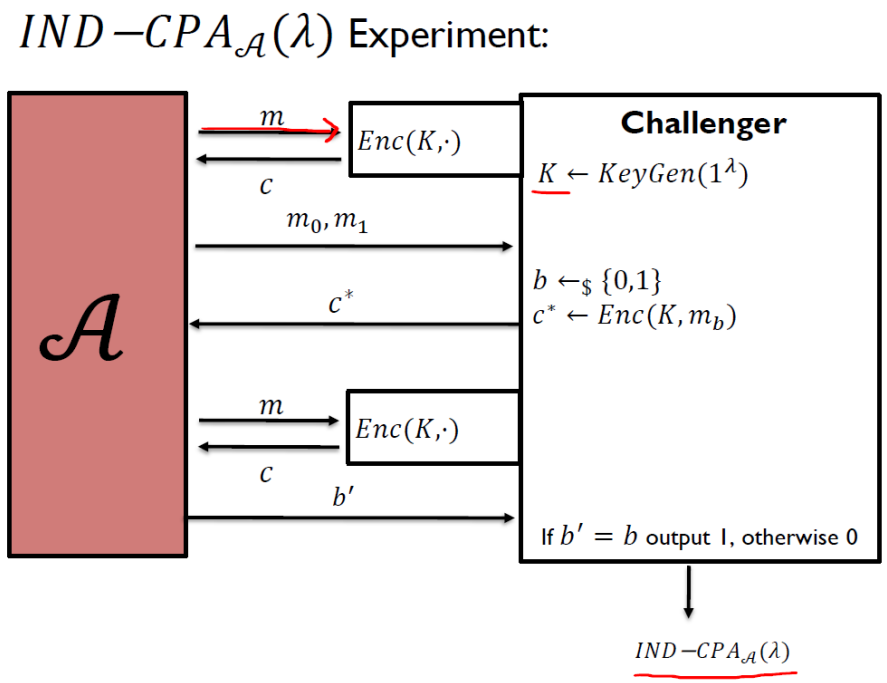
7.2 Chosen Plaintext Attacks

- Key-Reuse: Adversary sees many encryptions under the same key
- Not all of them might be unpredictable
- Real Real world scenario: Adversary can influence what honest party encrypts
- Conservative Approach: Let the adversary choose the messages of which he sees encryptions!
- Idea: Give the adversary an **encryption oracle**



Definition 11 ($IND-CPA$-secure)**.** *An encryption scheme* $(KeyGen, Enc, Dec)$ *is* $IND-CPA$*-secure, if it hol;ds for every PPT-adversary* $\mathcal{A}$ *there exists a negligible function* v *s.t. for all* $\lambda \in \mathbb{N}$

$$Pr[IND - CPA_{\mathcal{A}}(\lambda) = 1] < \frac{1}{2} + v(\lambda)$$



7.3 Constructing IND-CPA secure encryption

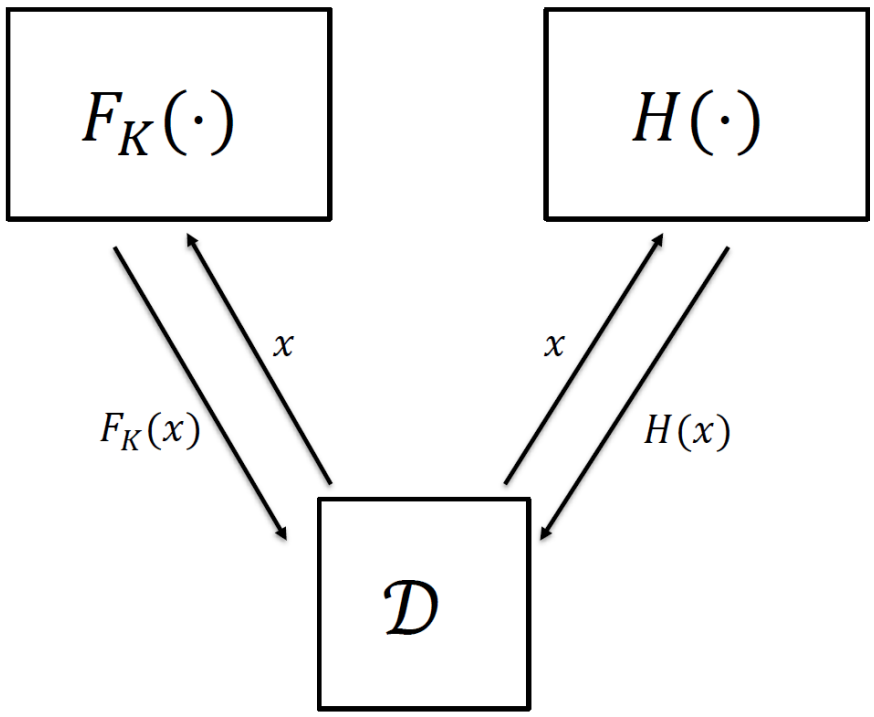
- Observation: IND CPA secure encryption cannot be deterministic
- Encryption needs to use randomness
- To achieve this stronger security notion, we need stronger ingredient
- PRGs: Small amount of randomness ⇒ Larger amount of pseudorandomness
- Needed: Small amount of randomness ⇒ (Essentially) unlimited pseudorandomness

Definition 12 (Pseudorandom Functions)**.**

- *Efficiently computable keyed function (ensemble)* $F_K : \{0,1\}^\lambda \to \{0,1\}^l$*, where* $K \in \{0,1\}^\lambda$
- *Uniformly random function* $H : \{0,1\}^\lambda \to \{0,1\}^l$
- *I.e. for every* $x \in \{0,1\}^\lambda$ *it holds that* $H(x)$ *is independently uniformly random on* $\{0,1\}^l$
- F *is a pseudorandom function, if for every oracle PPT distinguisher* $\mathcal{D}$ *it holds*

$$|Pr[\mathcal{D}^{F_K(\cdot)}(1^\lambda) = 1] - Pr[\mathcal{D}^{H(\cdot)}(1^\lambda) = 1]| < negl(\lambda)$$

where probability is over choice of $K \leftarrow_\$ \{0,1\}^\lambda$*,* H *and random coins of* $\mathcal{D}$*.*



7.3.1 Examples

Assume F, F' are PRFs:

- $F^*_{(K,K')}(x) = F_K(x) \oplus F'_{K'}(x)$
  $F^*_{(K,K')}(x)$ is a PRF, because

  $$\mathcal{D}^{*(F_K \oplus F'_{K'})(\cdot)} \approx \mathcal{D}^{*(H \oplus F'_{K'})(\cdot)} \approx \mathcal{D}^{*H'(\cdot)}$$

  with

  $$\mathcal{D}^{*(F_K \oplus F'_{K'})(\cdot)} \text{ computes } F_K(x) \oplus F'_{K'}(x), \text{ and } \mathcal{D}^{*(H \oplus F'_{K'})(\cdot)} \text{ computes } H(x) \oplus F'_{K'}(x).$$

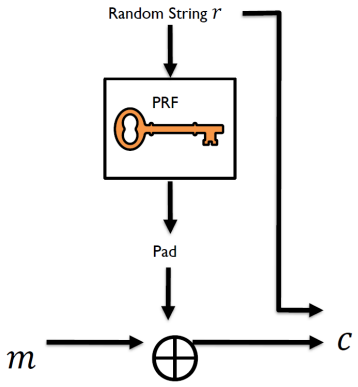
  Let $H(x)$ be an uniformly random function.
  If we XOR a function with an uniformly random function, the result is an uniformly random function too. So, $H'(x) := H(x) \oplus F'_{K'}(x)$ is an uniformly random function.
  (The proof of reduction was omitted)

- $F^*_K(x) = (F_K(x), F_K(x \oplus 1^n))$
  $F^*_K(x)$ is not a PRF, because

  $$F^*_K(x \oplus 1^n) = (F_K(x \oplus 1^n), F_K((x \oplus 1^n) \oplus 1^n)) = (F_K(x \oplus 1^n), F_K(x \oplus 1^n \oplus 1^n)) = (F_K(x \oplus 1^n), F_K(x))$$

  has the same output as $F^*_K(x)$ if we change the positions of the tupel.
  Therefore the outputs for the inputs x and $x \oplus 1^n$ are correlated.

7.4 Encryption from Pseudorandom Functions



- $KeyGen(1^\lambda)$: Choose $K \leftarrow_\$ \{0,1\}^\lambda$
- $Enc(K, m)$: Choose $r \leftarrow_\$ \{0,1\}^\lambda$, compute and output $c \leftarrow (r, F_K(r) \oplus m)$
- $Dec(K, c)$: Parse $c = (r, y)$, compute and output $m \leftarrow F_K(r) \oplus y$
- **Correctness:**

$$Dec(K, Enc(K, m)) = Dec(K, (r, F_K(r) \oplus m)) = F_K(r) \oplus F_K(r) \oplus m = m$$

Theorem 4 (IND-CPA Security)**.**

$$F \textit{ is a pseudorandom function} \Rightarrow (KeyGen, Enc, Dec) \textit{ is IND-CPA secure}$$

Proof. Assume towards contradiction there exists a PPT $\mathcal{A}$ and a non-negligible ϵ so that

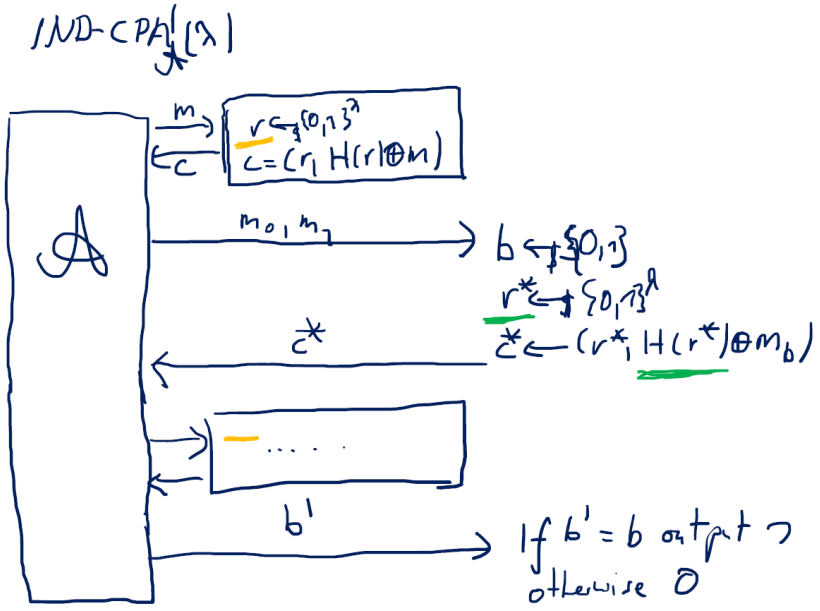
$$Pr[IND - CPA_{\mathcal{A}}(\lambda) = 1] \geq \frac{1}{2} + \epsilon$$

Show: This implies distinguisher $\mathcal{D}$ against PRF F with non-negligible advantage.
First, a thought experiment. What if we use instead of an encryption scheme with a PRF

$$Enc(K, m) = (r, F_K(r) \oplus m),$$

a modified encryption scheme with an uniformly random function

$$Enc'(m) = (r, H(r) \oplus m).$$



How many r? $\mathcal{A}$ is PPT-machine. It runs in polynomial time.
$\Rightarrow$ Let $q = q(\lambda) = poly(\lambda)$ is an upper bound on the number of queries to enc-oracle by $\mathcal{A}$.
Let $R \subseteq \{0,1\}^\lambda$ be the set of strings r used by enc-oracle during the interaction with $\mathcal{A}$.
$\Rightarrow |R| \leq q \Rightarrow Pr[r^* \in R] = \frac{|R|}{2^\lambda} \leq \frac{q}{2^\lambda}$
With the LOTP it follows that

$$\begin{aligned} Pr[IND - CPA'_{\mathcal{A}}(\lambda) = 1] = Pr[IND - CPA'_{\mathcal{A}}(\lambda) = 1 \mid r^* \in R] \cdot Pr[r^* \in R] \\ + Pr[IND - CPA'_{\mathcal{A}}(\lambda) = 1 \mid r^* \notin R] \cdot Pr[r^* \notin R] \end{aligned}$$

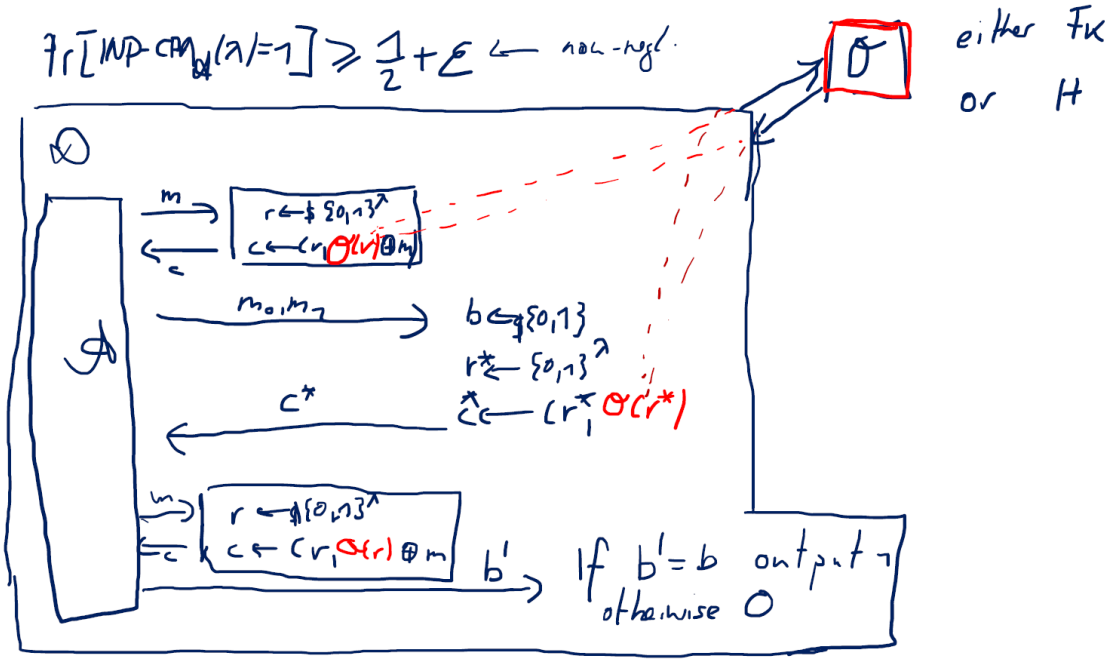
Let's consider the single components:

- $Pr[IND - CPA'_{\mathcal{A}}(\lambda) = 1 \mid r^* \in R] \leq 1$
- $Pr[r^* \in R] \leq \frac{q}{2^\lambda}$
- $Pr[IND - CPA'_{\mathcal{A}}(\lambda) = 1 \mid r^* \notin R] = \frac{1}{2}$
- $Pr[r^* \notin R] \leq 1$

Therefore we have

$$Pr[IND - CPA'_{\mathcal{A}}(\lambda) = 1] \leq \frac{1}{2} + \frac{q}{2^\lambda}$$

with $\frac{q}{2^\lambda}$ negligible.



Case distinction:

1. If $\mathcal{O} = F_K$ then $\mathcal{D}$ simulates $IND - CPA_{\mathcal{A}}(\lambda)$ experiment:

$$Pr[\mathcal{D}^{F_K(\cdot)}(1^\lambda) = 1] = Pr[IND - CPA_{\mathcal{A}}(\lambda) = 1] \geq \frac{1}{2} + \epsilon$$

2. If $\mathcal{O} = H$ then $\mathcal{D}$ simulates $IND - CPA'_{\mathcal{A}}(\lambda)$ experiment:

$$Pr[\mathcal{D}^{H(\cdot)}(1^\lambda) = 1] = Pr[IND - CPA'_{\mathcal{A}}(\lambda) = 1] \leq \frac{1}{2} + \frac{q}{2^\lambda}$$

Now we can follow

$$Pr[\mathcal{D}^{F_K(\cdot)}(1^\lambda) = 1] - Pr[\mathcal{D}^{H(\cdot)}(1^\lambda) = 1] \geq \frac{1}{2} + \epsilon - \frac{1}{2} - \frac{q}{2^\lambda} = \epsilon - \frac{q}{2^\lambda}$$

We know that ϵ is negligible and $\frac{q}{2^\lambda}$ is non-negligible, so it follows that the difference is not negligible! $\square$

7.5 Summary

- Key reuse requires a stronger security notion: $IND-CPA$ security
- $IND-CPA$ secure encryption schemes cannot be deterministic!
- $IND-CPA$ secure encryption can be constructed from pseudorandom functions (PRFs)

Block Ciphers

Definition 13 (Block Ciphers)**.**
Let k, n be positive integers. A block cipher with key length k and block length n is an efficient keyed permutation

$$F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$$

Remark:

- *The function $F_K := F(K, \cdot)$ is a bijection (its inverse is denoted F_K^{-1}).*
- *Both F_K and F_K^{-1} are efficiently computable given the key K.*
- *A secure block cipher should "behave as a pseudorandom permutation".*

8.1 A Note on the Concrete Security Setting

The key length and block length of block ciphers are fixed → no varying "security parameter". In practice:

- actual (not asymptotic) complexity of adversaries is considered.
- a block cipher is considered secure as long as no attack significantly faster than exhaustive key search exists.

8.2 Security Definition - Indistinguishability from a Random Permutation

Definition 14 ((q, t, ϵ)-prp secure)**.**
A block cipher F with key length k and block length n is (q, t, ϵ)-prp secure if, for every probabilistic adversary D that runs in time at most t and deals at most q oracle queries, one has

$$Adv_F^{prp}(D) := |Pr[D^{F_K(\cdot)} = 1] - Pr[D^{f(\cdot)} = 1]| \leq \epsilon,$$

where the first probability is taken over the uniformly random draw of K and the second one over the uniformly random draw of the permutation $f : \{0,1\}^n \to \{0,1\}^n$.

Definition 15 ((q, t, ϵ)-sprp secure)**.**
A block cipher F with key length k and block length n is (q, t, ϵ)-sprp secure if, for every probabilistic adversary D that runs in time at most t and deals at most q oracle queries, one has

$$Adv_F^{sprp}(D) := |Pr[D^{F_K(\cdot), F_K^{-1}(\cdot)} = 1] - Pr[D^{f(\cdot), f^{-1}} = 1]| \leq \epsilon,$$

where the first probability is taken over the uniformly random draw of K and the second one over the uniformly random draw of the permutation $f : \{0,1\}^n \to \{0,1\}^n$.

Remark:
F is considered "secure" as long as $Adv_F^{sprp}(D) \leq c_1 \frac{t/T_F}{2^k} + c_2 \frac{q}{2^n}$, where c_1 and c_2 are small constants and T_F is an upper bound on the time required to evaluate F.

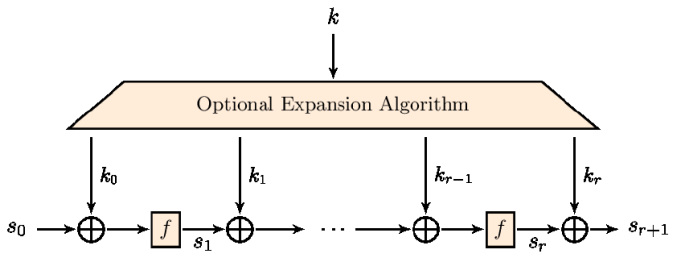
8.3 A word on Provable Security

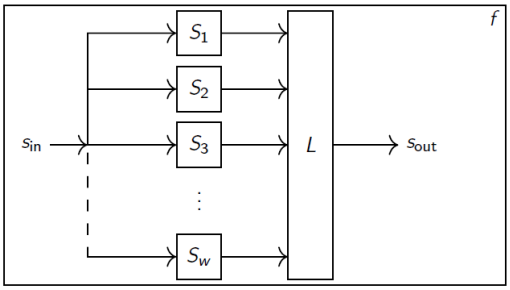
- Typically, the security of block ciphers is not provable.
- The pseudorandomness of a block cipher is actually used as an assumption in security proofs for block cipher-based private-key algorithms.
- To build confidence in the pseudorandomness of a block cipher:
  - decades of cryptanalysis;
  - heuristical arguments of security against particular classes of attacks (generic attacks, differential or linear cryptanalysis, ...).

8.4 Design Principles

- Block ciphers should behave as pseudorandom permutations: a 1-bit change in the input should affect every bit of the output (avalanche effect)!
- But block ciphers should be efficient and have a short description.
- Confusion-diffusion paradigm: build a pseudorandom permutation from small random (or random-looking) permutations.
- In practice, the following process, called a round, is applied several times to the input block:
  1. divide the block in small chunks;
  2. apply small random-looking permutations (called S-boxes) to each chunk of data (confusion);
  3. mix the bits of the intermediate value to spread the local changes to the whole block (diffusion); this step can be a simple reordering of the bits or the application of a more complex (invertible) linear function.

8.5 Substitution-Permutation Networks





- Practical instantiation of the confusion-diffusion paradigm.
- The S-boxes, key expansion algorithm and linear layer L are public.
- The key is expanded to several subkeys that are mixed with the intermediate values using a bitwise XOR.
- Subkeys are added before each round and after the last one.
- The construction is invertible since both the S-boxes and the linear mixing layer are invertible.

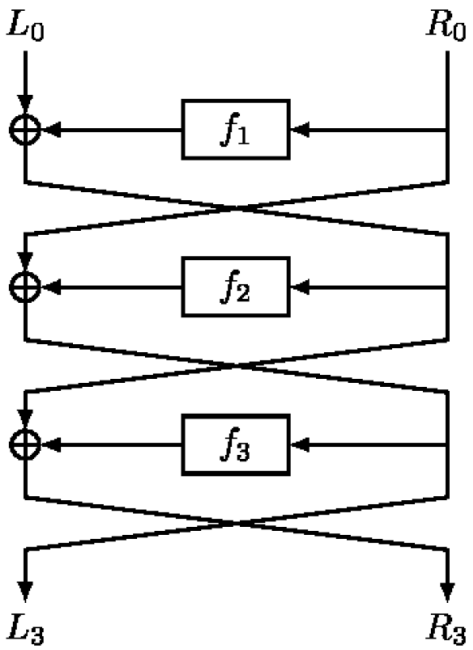
8.6 The Avalanche Effect

- Example of design criterion to get the avalanche effect:
  - the S-boxes are chosen so that changing a single bit of the input changes at least two bits of the output.
  - the mixing permutations (in that case simple bit reordering) are chosen so that the output bits of any given S-box are used as input of multiple S-boxes in the next round.
- Consequence:
  - a single bit difference between input blocks results in a difference of two bits after one round;
  - the second condition ensures that the inputs of at least two S-boxes of the second round will differ by one bit;
  - after the second round: at least 4 bits differ between the two blocks;
  - in the best case, 2^r bits will be affected after r rounds (actually less);
  - this gives a lower bound for the number of rounds of an SPN:

$$r \geq \lceil \log_2(n) \rceil$$

8.7 Feistel Networks

- Generic iterated structure to build a PRP from round functions.
- Typically, round functions are also constructed from (possibly non-invertible) S-boxes and linear mixing layers.
- Description of a round:
  - break the block in two equal halves L_i and R_i;
  - output (L_{i+1}, R_{i+1}) defined as $L_{i+1} = R_i$ and $R_{i+1} = L_i \oplus f_i(R_i)$, where f_i denotes the i-th (public) round function instantiated with a (secret) key K.
- Inherently invertible: $R_i = L_{i+1}$ and $L_i = R_{i+1} \oplus f_i(L_{i+1})$.



Theorem 5 (Feistel Networks)**.**
Assuming the round functions are uniformly random and independent functions from $\{0,1\}^n$ *to* $\{0,1\}^n$*, then:*

- *the* 3*-round Feistel construction is* $(q, \infty, \frac{q^2}{2^n})$*-prp secure;*
- *the* 4*-round Feistel construction is* $(q, \infty, \frac{q^2}{2^n})$*-sprp secure;*
- *the* 6*-round Feistel construction is* $(q, \infty, \frac{9q}{2^n})$*-sprp secure secure as long as* $q \leq 2^{n-7}$*.*

Interpretation of the theorem:

- *actual block ciphers have simple round functions that are not random or pseudorandom;*
- *however, the result justifies the soundness of the Feistel structure;*
- *it provides lower bounds for the number of rounds that have to be used by an actual block cipher.*

8.8 Building Blocks of a Block Cipher

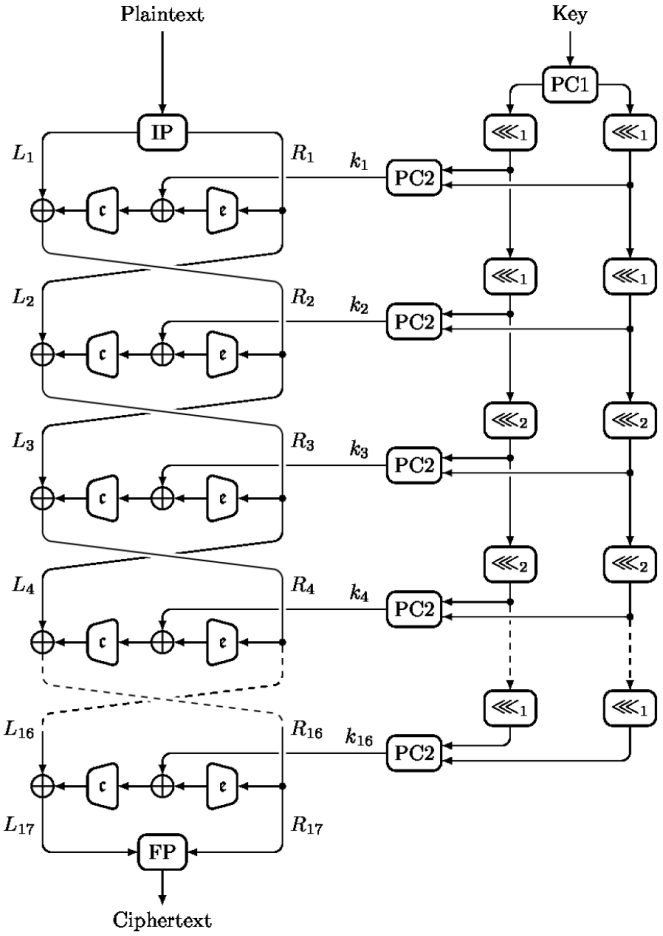
To summarize, the specification of a block cipher generally contains the following ingredients:

- a high-level (iterated) structure (SPN, Feistel scheme,. . .);
- a key-schedule algorithm to derive sub-keys from a master key;
- small S-boxes that must be non-linear;
- an efficient linear layer to properly spread local changes from the application of the S-boxes.

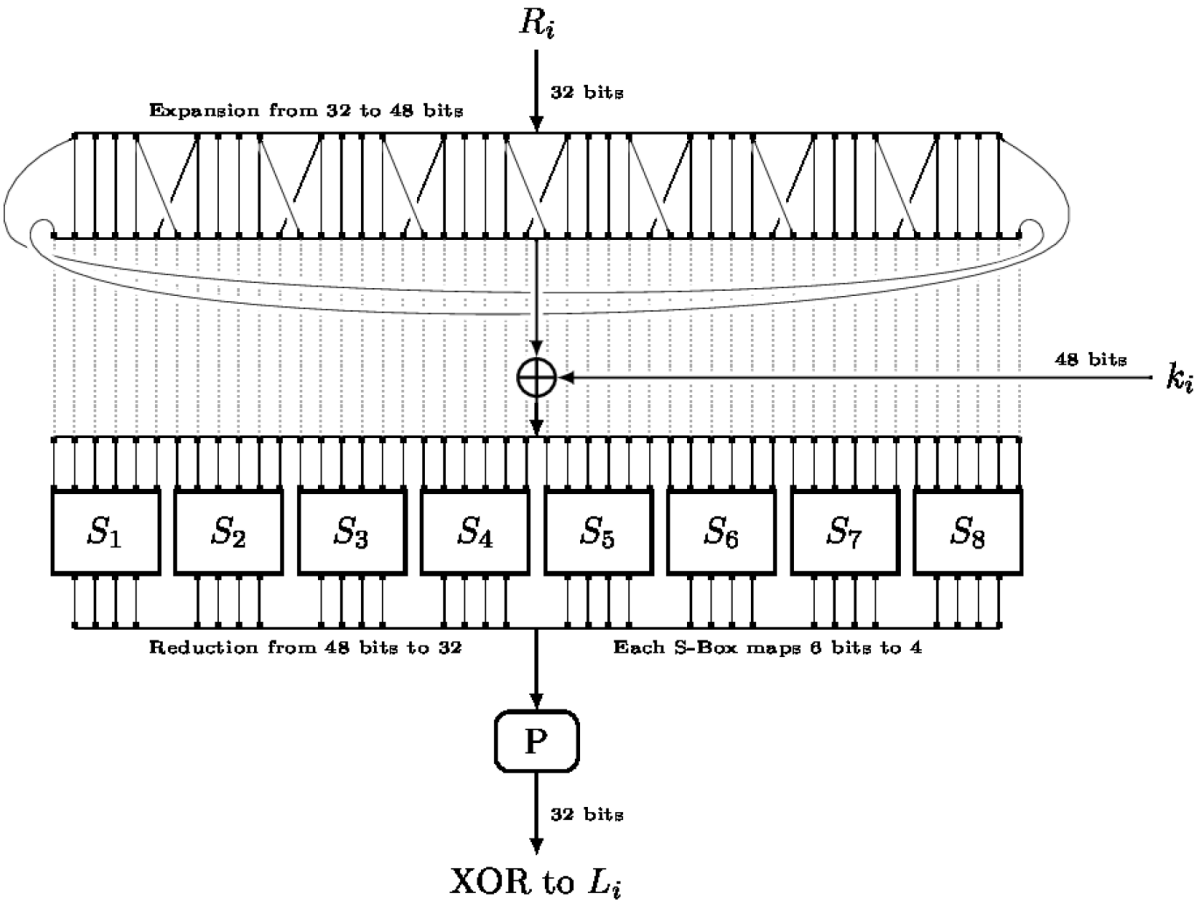
8.9 Examples

8.9.1 Example 1: The DES Block Cipher

- DES was developed in the 1970s by IBM (+ some help from the NSA).
- Key length: 56 bits and block length: 64 bits.
- Although key length is too small for our current standards, it has been impressively resilient to decades of cryptanalysis.
- Best cryptanalysis: linear cryptanalysis using 2^{43} known plaintext/ciphertext pairs and time around 2^{40} DES evaluations.



- Generic structure: 16 round Feistel network.
- Key schedule:
  1. PC1 splits the key in two 28-bit halves;
  2. both halves are rotated to the left before each round (number of places depends on the round);
  3. PC2 then extracts 48 key bits.
- IP and FP reorder the bits, but do not depend on any secret information (no impact on security).
- Round function:
  1. e extends its 32-bit input to 48 bits;
  2. c contracts its 48-bit input back to 32 bits.



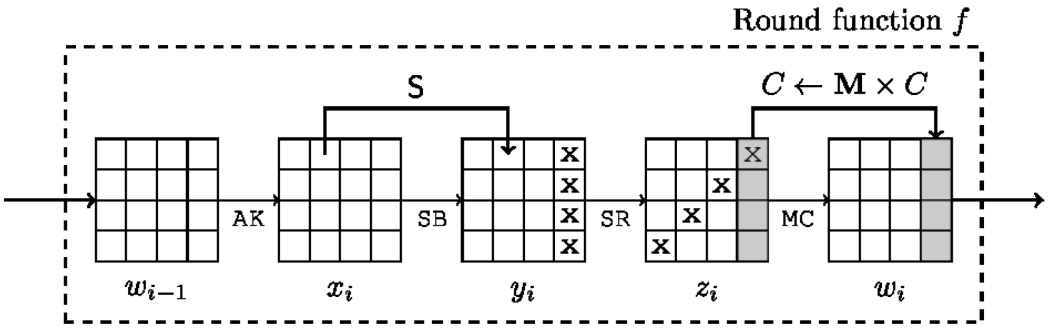
- The S-boxes $S_1, ..., S_8$ are 6-bit to 4-bit functions such that:
  1. each S-box is 4-to-1;
  2. changing one bit of any input to an S-box always changes at least two bits of the output.
- the mixing layer P is a simple reordering of the input bits such that the 4 output bits of any S-box will affect the input to 6 S-boxes in the next round (thanks to the linear expansion layer e).

The DES avalanche effect:

1. Assume (L_0, R_0) and (L'_0, R'_0) are two inputs to the cipher such that $R_0 = R'_0$, L_0 and L'_0 differ by exactly one bit.
2. After the first round, since $R_0 = R'_0$, then $L_1 = L'_1$, R_1 and R'_1 differ by exactly one bit.
3. Since $L_1 = L'_1$, R_2 and R'_2 differ by at least 2 bits, while L_2 and L'_2 differ by exactly one bit.
4. Thanks to P, the 2 different bits of R_2 and R'_2 appear in at least 2 S-boxes, which means that the new intermediate values will differ in at least 4 positions. This results in at least 2 different bits between L_3 and L'_3 and at least 4 different bits between R_3 and R'_3.
5. We observe a similar exponential effect as in the Substitution-Permutation Network case.

8.9.2 The AES Block Cipher

- United States National Institute of Standards and Technology (NIST) standard adopted in 2000 after an open competition.
- Based on the winner of the competition: Rijndael (designed by Vincent Rijmen and Joan Daemen).
- Actually a set of 3 algorithms with block length 128 bits and key length 128, 192 or 256 bits.
- The length of the key affects the number of rounds (10, 12 or 14) and the key schedule.
- Modern processors support instruction sets to accelerate the computation of AES in hardware: around 4.5GB/sec for AES-128 in CTR mode on an Intel(R) Core(TM) i7-3740QM CPU @ 2.70GHz (Fedora 31, openssl 1.1.1d)
- Best cryptanalysis: there exist attacks that recover the AES key with a computational complexity of 2^{126} operations for AES-128, 2^{189} operations for AES-192, and 2^{254} operations for AES-256. No practical threat to the security of the cipher.
- Better attacks exist if the adversary is given more power (the ability to XOR any constant of its choice to the secret key), but this does not impact the security of AES when used in standard modes of operation.
- Generic structure: 10, 12 or 14-round Substitution-Permutation Network.
- Key schedule: relies on linear operations and on the AES S-box, and outputs one 128-bit sub-key for each round, plus a final one.
- The AES round permutation consists in four steps:
  - AddRoundKey: the bitwise XOR of the round key;
  - SubBytes: the application of a single invertible S-box to each of the 16 bytes of the input;
  - ShiftRows: a reordering of the bytes of the state;
  - MixColumns: the application of a linear operation to the state.
- In the last round, the MixColumns operation is skipped, and replaced with the XOR of the last key.



- The block is seen as a 4×4 matrix of bytes.
- SubBytes applies the AES S-box to each element of the matrix.
- ShiftRows cyclically shifts row r of the matrix by r positions to the left for $r = 0, 1, 2, 3$.
- MixColumns can be seen as the multiplication of each column of the matrix by a fixed invertible 4×4 matrix. This transformation has the property that, if two inputs differ in $b > 0$ bytes, then the outputs will differ in at least $5 - b$ bytes.

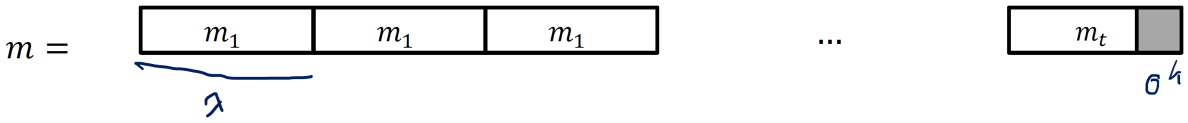
The AES avalanche effect:

1. Assume X_0 and X_0' are two inputs to the cipher that differ in exactly one byte.
2. After the first ShiftRows step, they will still differ in exactly one byte. However, at the end of the first round, all the bytes of the corresponding column will differ between both intermediate values, while the remaining bytes will stay equal.
3. After the second ShiftRows step, there will be exactly one byte per column that will differ between both intermediate values.
4. At the end of the second round, all bytes will be different between both intermediate values.

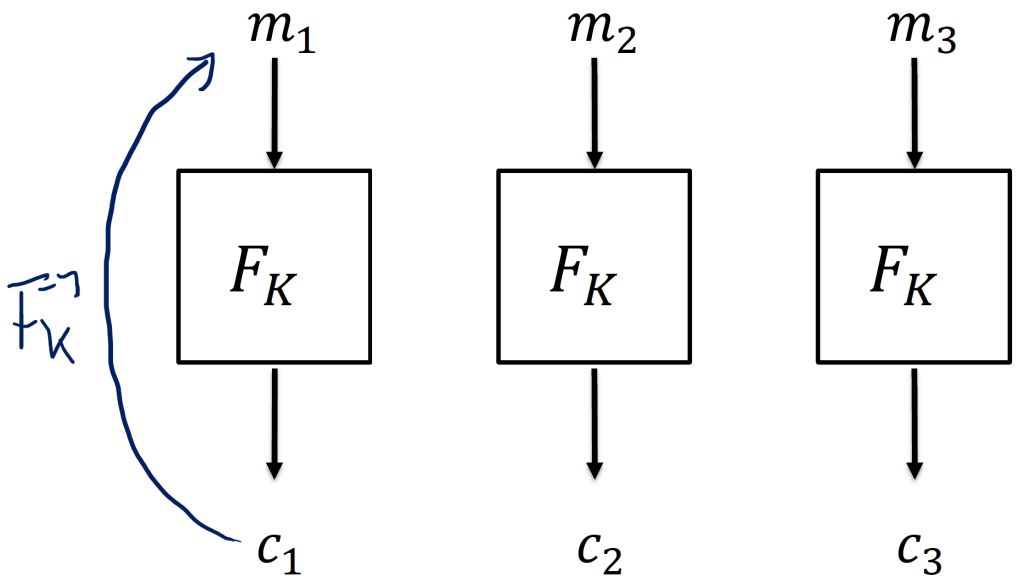
8.10 Blockcipher Modes of Operation

8.10.1 Using Block-Ciphers

- Block ciphers/PRPs are a *primitive*
- How do we construct encryption from PRPs?
- Constructions of encryption from block ciphers are called **modes of operation**
- Assume messages can be chopped into blocks of size λ, otherwise use padding

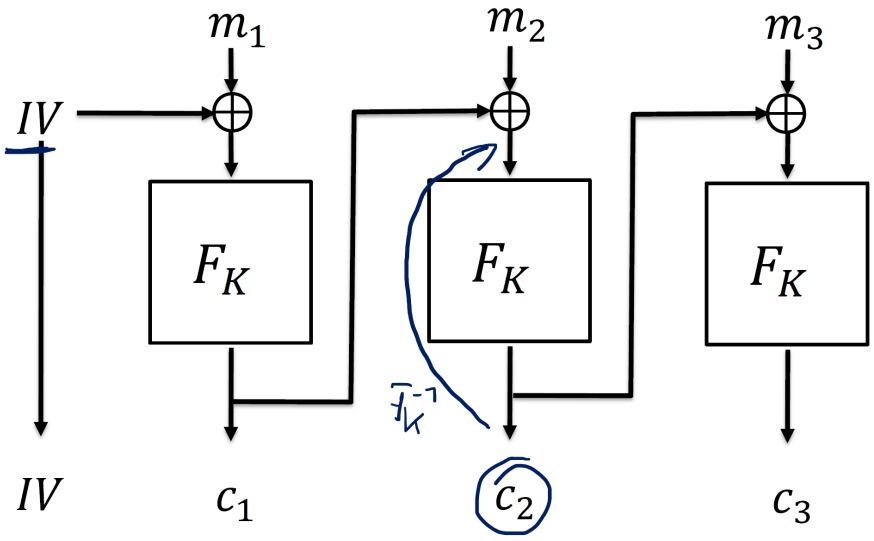


8.10.2 Electronic Codebook Mode (ECM)



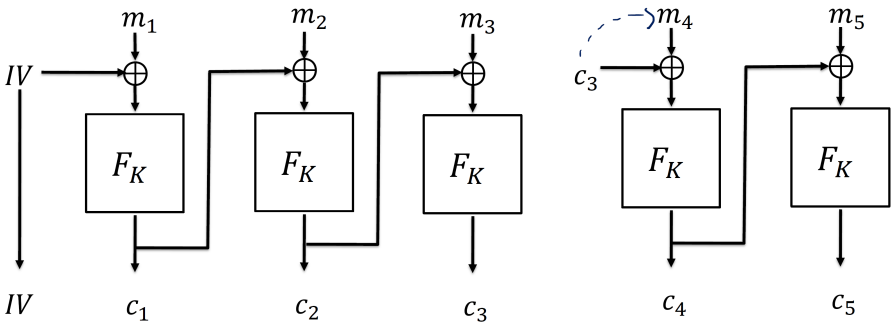
- Not IND-CPA secure!
- Not even IND secure!
- Should not be used!

8.10.3 Cipher Block Chaining (CBC) Mode



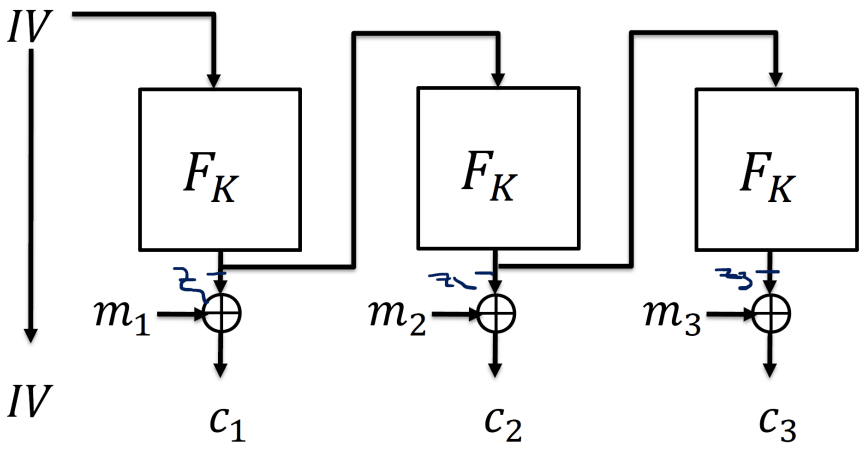
- IND CPA secure for uniformly random IV if is PRP
- Encryption must be performed sequentially
- Decryption is local

8.10.4 Chained CBC Mode



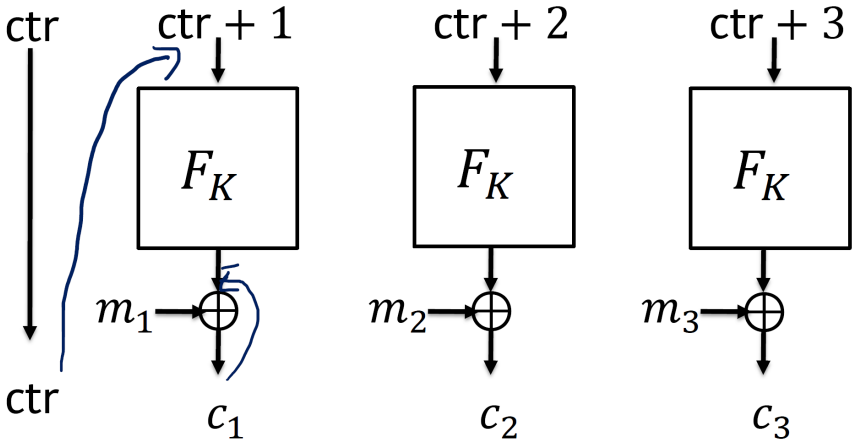
- **Not** IND-CPA secure!

8.10.5 Output Feedback (OFB) Mode



- IND-CPA secure for uniformly random IV if F is PRF
- Encryption and Decryption must be performed sequentially!
- Pad can be precomputed

8.10.6 Counter (CTR) Mode



- IND-CPA secure for uniformly random ctr if F is PRF
- Encryption and Decryption are local/can be parallelized/precomputed
- Not reason not to use this one!

8.11 Summary

- Blockcipher modes of operation provide a way to encrypt long messages
- ECM provides very little security → should only be used in very special scenarios
- CBC is ok!
- Chained CBC is not ok!
- OFM is ok!
- Use CTR whenever possible!

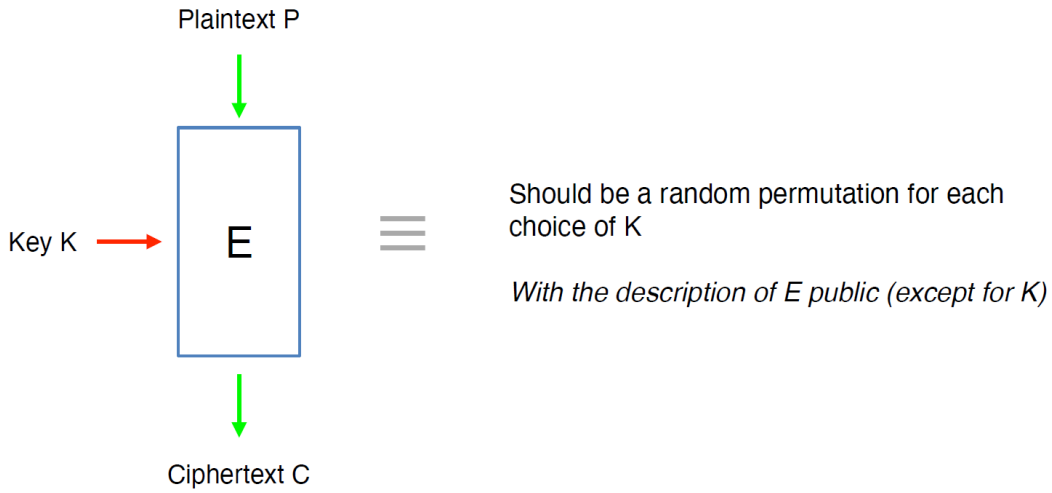
Cryptanalysis

9.1 Security goals viewed by Cryptanalysis

- In general, security of crypto primitives can be hard to define
- « Extreme definitions » don't work, exemple:
  - Secret key should be impossible to find
    ⇒ Too weak
  - A Block cipher should behave in all respects like a random permutation
    ⇒ Impossible to achieve
- Security proofs need formal definition. Cryptanalysts don't!
- Basically, any « non generic » property might become a weakness
  - Some generic properties might also be weaknesses showing bad parameters

A first remark is that security notions are not intuitive in the field of cryptology. Even the meaning of the word « attack » is relative and shifts with time or with respect to the point of view of the speaker. In particular, the security notions involved in security proofs can be very different from the one we see in cryptanalysis. For security proofs, a formal definition is required; for cryptanalysts, an attack scenario with a potential weakness that could be exploited is a better justification. The choice of scenario has greatly evolved with time. In the early days of historical crypto, ciphertext-only attacks were the reference. Nowadays, a wide range of interactive attacks are considered.
For this reason, just asking for the secret-key to be impossible to recover by an attacker isn't enough. It might not prevent them from deciphering messages or from performing unauthorised modifications. At the other extreme, some definitions can be too strong, simply because they are impossible to achieve. We give an example in the next slide.
The current consensus among secret-key cryptographers is that any explicit, non-generic, property of a cryptographic primitive is a potential weakness (at least, as long as it can be made explicit with an efficient enough computation). In some cases, generic weaknesses simply show that some desired primitive is just impossible to construct in a secure way.

9.2 Block Ciphers



A first idea would be to request that a block cipher should behave like a random permutation in all respects for every choice of the key K. Furthermore, following Kerckoffs' principle, we still want this to be true when the description of the cipher itself is public and only the key remains secret.

- This informal definition is impossible to achieve
  - Indeed, a random permutation doesn't generally have a compact description
- Generic attacks
  - Exhaustive key search
  - Plaintext block collisions are visible on Ciphertext block

⇒ Key size and block size should be large enough. Today's standard:

- At least 128 bits for key size, 192 or 256 are better when possible
- At least 128 bits for block size
  (64 bits still appears in legacy applications)

Unfortunately, this definition cannot be achieved. Indeed, by construction the block cipher has a compact description (in the form of its program), which is impossible for a truly random permutation.
Instead, for security proofs, one turns to the notion of random permutation family which we will not revisit during this lecture.
Since a block cipher is a permutation acting on plaintext-block values and chosen in a large but finite family by instantiating the key, some generic attacks arise. First, since there are only finitely many keys, an attacker could (in principle) try all of them to recover the correct one. Due to Shannon's information theory, even a few blocks of data are enough to uniquely characterise the correct key. To make this attack infeasible, cryptographers use extremely large sets of possible keys. The current choice of key size is between 128 and 256 bits. As a consequence, exhaustive search is infeasible even for attackers possessing tremendous computing powers and willing to use them for years or even decades.
Another generic property is that encrypting the same block of plaintext twice yields the same ciphertext. This can lead to devastating attacks. As a consequence, block-ciphers always need to be used in a way that prevents such collisions between plaintext values from occurring. This is taken into account when constructing « modes of operation », which are basically recipes for using block ciphers. Furthermore, the block size should be large enough to prevent collisions from appearing when blocks are randomly chosen. For this reason, modern block ciphers operate on 128 bits at a time.

9.3 DES

9.3.1 DES: an outdated but interesting example

- DES = Data Encryption Standard
- Block Cipher developed in the 70s at IBM
- NIST standard from 1976 to 2005
- DES is a Feistel cipher with a 56-bit key and 64-bit blocks
- Even in 1976, the key size was on the low side

During this lecture, we are going to use DES as an example to present some cryptanalytic results. Despite being outdated, this algorithm has motivated a lot of research in cryptanalysis and led to several seminal breakthroughs. In addition, since it was a NIST standard from 1976 to 2005, it is still often encountered in legacy applications. This cipher has 56-bit keys and 64-bit blocks. Note that even when it is was introduced, its key size was already considered too small by many people in academia.

9.3.2 How to have longer keys?

- Modify the algorithm to change key size
  - In the long run, it led to AES (Advanced Encryption Standard)
  - In the early days, FEAL tried to improve on DES
- Incorporate the algorithm in a bigger structure:
  - Multiple DES (Introduced by Diffie-Hellman 77)
  - DES-X (Rivest'84, unpublished)

Because of DES small key size, the question of increasing the key size quickly arose. One obvious avenue is to redesign a new cipher with bigger keys. However, the crypto community soon discovered that this is harder than it seems and many attempts were broken which helped slowly building cryptanalytic expertise within academia. Eventually, NIST even became confident that this expertise was very strong and asked the community to design the successor of DES in the open AES competition.
Another avenue is to assume that DES is well-designed, despite its small key, and to integrate it into a bigger construction to obtain larger keys.

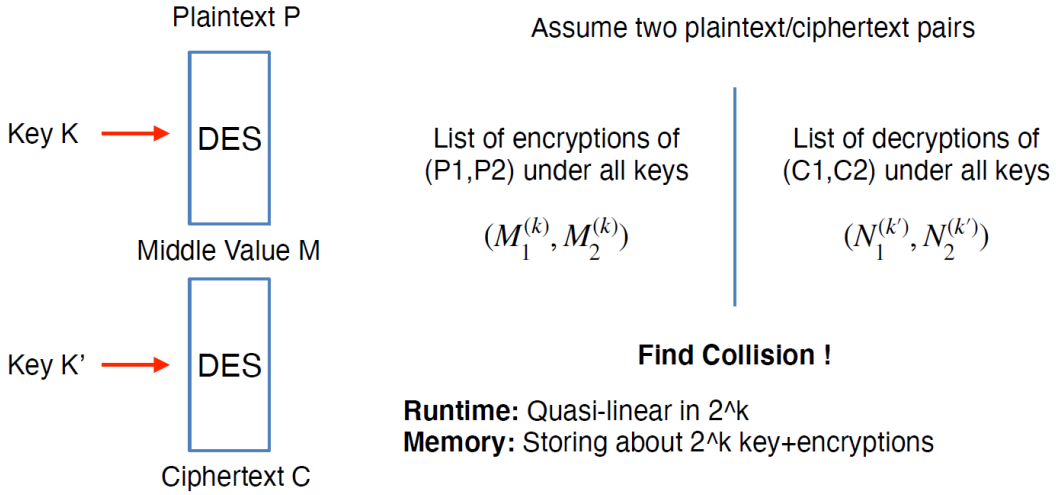
9.3.3 Multiple DES - Diffie-Hellman 77

Another way to obtain a variable length key would be to use the currently proposed standard, but to encipher m times with m independent 56-bit keys. This would hopefully yield a $56m$-bit key, but additional analysis is required. For example, enciphering twice with two monoalphabetic ciphers is equivalent to enciphering only once with a third monoalphabetic cipher. If cipher one carries A to F and cipher two carries F to C, then the overall effect is to carry A to C. This is because monoalphabetic ciphers form a semi-group under composition. It is highly doubtful that the proposed standard possesses this property.

A first method, proposed by Diffie and Hellman in 1977 is simply to encrypt several times with independent keys. They remarked that if DES was hiding a group-like structure, this approach would be doomed. It took some time for the community to rule out this possibility.

9.3.4 Double DES - Diffie-Hellman 77

After introducing multiple DES, they show that m=2 isn't secure



In the same paper, Diffie and Hellman show that encrypting twice isn't good enough and doesn't provide the security level that one should expect from 112-bit keys. To see that, assume that we know the encryption of two plaintext blocks. Indeed, one isn't enough to characterise the key of double DES. Remark that if we encrypt the plaintext with the first half of the key and decrypt the ciphertext with the second half, we obtain the same middle values. Moreover, for wrong keys, this event is really unlikely.

As a consequence, we construct two lists, the first with all encryptions under all first half-keys and the second with all decryptions. We expect very few collisions between these lists and every collision give a candidate key. The right key is, of course, within this small set and it is easy to test it (for example by decrypting some extra text). This works because collisions can be found efficiently.

9.4 Finding Collisions in a list or between lists

- This is a major algorithmic tool in cryptanalysis
- Can be done in quasi-linear time in the lists' sizes

- Basic idea for a single list:
  - Sort the list of size N (takes $O(N \cdot log(N))$ comparisons)
  - Read it in order. Collisions will be between consecutive elements!
  - Any ordering of the elements work (a natural ordering isn't needed)

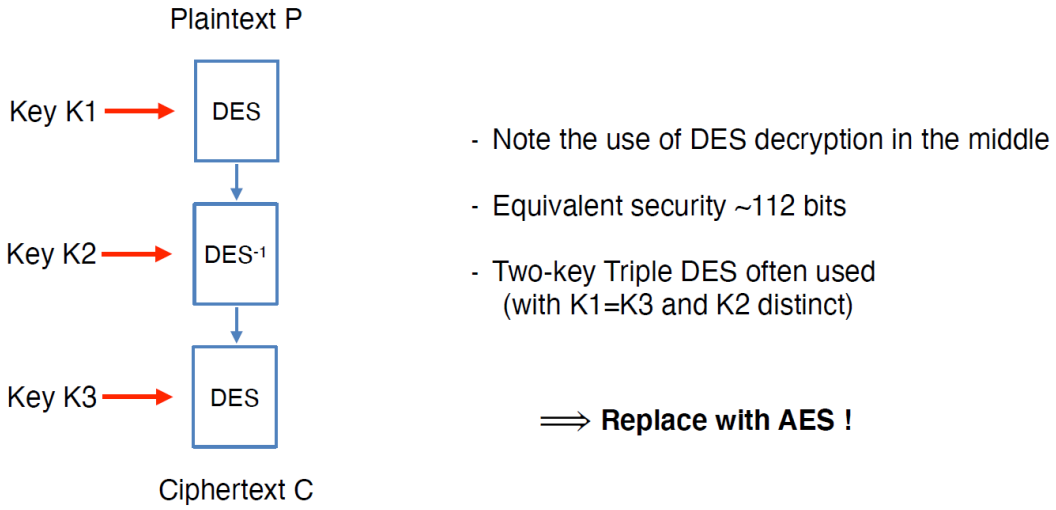
- With two lists:
  - Sort both
  - Start at the beginning of both lists, compare elements, advance the smallest
  - Iterate until a collision is found

Extra care needed to get **all** collisions

In fact, searching for collisions within two lists or inside of a single list is a very important tool in cryptanalysis. We will use it many times throughout the crypto course.
The simplest idea is to test all pairs. However, this yields an algorithm whose time is quadratic in the lists sizes. In the double DES example, with two lists containing 2^{56} elements, we would have to try 2^{112}

pairs. As a consequence, this wouldn't be an improvement on exhaustive search at all.
To outperform this, the basic idea is to start by sorting the list (or lists). Thanks to fast algorithms like quicksort, this can be done in quasi-linear time. After sorting, equal elements become neighbours in a single list. So to discover them, it suffices to read the sorted list in order.
With two lists, reading out the collisions is slightly more difficult. The idea is to start by pointing to the first elements of the two lists and compare them. Then advance the pointer corresponding to the smallest element to the next element in the same list. Compare again and repeat.
Note that if there are many collisions and we want to find all of them, we need to take extra care. Typically, if after sorting a single list we have ten consecutive elements which are equal, we have to build all pairs of elements among the ten and to construct 45 collisions. In many applications like double DES for example, this is usually not needed.

9.5 Triple DES - NIST standard (deprecated, disallowed after 2023)



Because of the meet-in-the-middle attack on double DES, it turns out that to really get more security using multiple DES, we need to use triple DES. In fact, triple DES still exists as a NIST standard, but will be disallowed after 2023. Of course, we can still use this attack on triple DES to reduce its security to 112 bits (even if the overall key size is 168 bits). Note that the attack now construct unbalanced lists, one of size 2^{56} (corresponding to the first key) and the other of size 2^{112} (corresponding to the last two keys). Because of these unbalanced sizes, it is worth noting that there is a way to do the collision search that only requires a memory of the order of 2^{56} blocks. Indeed, once the first list is constructed and sorted, we can build the second list one element at a time and use a dichotomy search to look for each element in the first list.
Because the security can't achieve 168 bits, the standard contains what is called two-keys triple DES, where the first and last keys are set to be identical. Note that in the standard the middle invocation of DES decrypts with the second key rather than encrypt. It can be seen as a weakness. For example, with two-key triple DES, if the first two keys are equal, the overall construction degenerates to simple DES. In fact, this was done on purpose by NIST, to allow triple-DES equipment to perform simple DES by using this degenerate keys.
You might still encounter triple DES in legacy applications, but it should be replaced by AES.

9.6 Linear and Differential Cryptanalysis

- The effort on studying DES (and other block ciphers like FEAL) led to:
  - Differential Cryptanalysis (Biham/Shamir 1990)
  - Linear Cryptanalysis (Matsui/Yamagishi 1992, Matsui 1993)

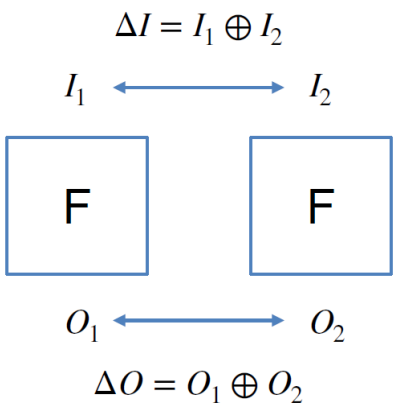
- Both techniques consider probabilistic properties of the encryption:
  - Found by examining building blocks
  - Under the heuristic assumptions that events behave independently

- These probabilistic techniques have been extended to other attacks
  - Linear and differential remain essential for evaluating ciphers

Two of the most fruitful methods that appeared by studying DES and some of the other block ciphers it inspired are the differential and the linear cryptanalysis techniques. The first was invented by Biham and Shamir in 1990, the second two years later by Matsui and Yamagishi.
These techniques share an important feature. Indeed, both of them rely on observing probabilistic relations that are satisfied in the target block cipher more often than they would be in a random permutation. Because of the large block sizes (and key sizes), such relations cannot be predicted by a brute force method. Instead, they are first constructed on small building blocks of the cipher and then composed into a property of the block cipher (or of a reduced version of the cipher for some attacks). The composition techniques that are used are usually heuristic and rely on the assumption that successive rounds behave independently.
They are many more recent attacks that share a lot of these features. To name a few, we can cite the square attack, the boomerang attack or the impossible differential attack. However, the linear and differential attacks remain essential to evaluate block ciphers and it is important to have some basic knowledge about them.

9.7 Differential Cryptanalysis

9.7.1 Fundamentals



Notation: $\Delta I \rightarrow_p \Delta O$

Rule 1: $\Delta I = 0 \rightarrow_{p=1} \Delta O = 0$

Rule 2: For linear transformations, all probs are 1

Rule 3 (heuristic):

$\Delta I \rightarrow_p \Delta M$ and $\Delta M \rightarrow_q \Delta O$

$\Longrightarrow \quad \Delta I \rightarrow_{pq} \Delta O$

In the differential attack, we look at pairs of blocks, I_1 and I_2, that are encrypted into outputs O_1 and O_2. The name differential comes from the fact that we study the relation between the difference of the inputs and the difference of the outputs. Remember that, in our context, every block is formed of bit strings. As a consequence, it is quite natural to study a bit-by-bit difference. Moreover, when working with bits, operations modulo 2 are the most common choice. Because of that, the difference of blocks is simply their Exclusive-OR.
More precisely, for every possible input difference ΔI, we count how many times it leads to an output difference, ΔO. After dividing by the number of possible pairs, we obtain the probability that ΔI leads to ΔO. We use the Notation at the top of the right column to indicate that ΔI leads to ΔO with probability p. These probabilistic arrows are usually called differential characteristics.
For example, for any block cipher (or any component of a block cipher), a zero difference in input always produce a zero difference in the output. Similarly, for a linear transformation (on bits), the output

difference is always the transformation by the linear map of the input difference.
To be able to construct differential characteristics on a block cipher with many rounds, we need a way to compose probabilistic arrows. For this, we use the third heuristic rule that says that we can combine two arrows where the output difference of the first is equal to the input difference of the second. Moreover, the probability of the composition is the product of the individual probabilities, under the heuristic independence assumption.

9.7.2 Small example

Consider the ADD function with 2 input bits and 2 output bits:

	ADD
00	00
01	01
10	01
11	10

DIFF	00	01	10	11
00	x4			
01		x2		x2
10		x2		x2
11	x2		x2	

On this small example, we show a full table of differential counts for a function with two bits of input and two bits of output. The chosen function is the addition of the two input bits, written in binary.
The table on the right tells how many times each output difference appears for a given input difference and empty cells indicate that the corresponding output difference is not possible.

9.7.3 Principle of the attack

- Assume we have $\Delta I \rightarrow_p \Delta O$ for a block cipher (with $p >> 2^{-n}$)
  - Encrypt many pairs with input difference
  - For the studied cipher ΔO is observed with probability close to p
  - For a random permutation ΔO is observed with probability close to 2^{-n}
- This yields a chosen plaintext distinguisher from a random permutation!
- For key recovery, use the distinguisher on a restricted num of rounds
- Together with exhaustive key search on (part of) the removed rounds

Once a differential characteristic is known for the full block cipher, we can build a distinguishing attack. Namely, we are given access to an encryption device and want to determine whether it implements the target block cipher (with an unknown key) or a random permutation. We simply encrypt many pairs with the prescribed input difference and observe how often the corresponding output occurs. For the block cipher, the proportion is close to the probability p. By contrast, for a random permutation, it is close to 2^{-n}.
The attack can also be used for key recovery. For this, we start from a characteristic on a reduced number of rounds and query plenty of pairs with the prescribed input difference. Then, we perform exhaustive search for the last round key, decrypt the output for one round and observe the difference. For the correct guess, the prescribed output will occur more frequently. There are subtleties to implement the attack when the key for the last round is too large for exhaustive search. Indeed, in that case, the cryptanalysis needs to find a way to implement the attack using exhaustive search on a reduced part of the key.
The details of how this can be done can vary depending on the exact specification of the block cipher we are attacking.

9.8 Linear Cryptanalysis

9.8.1 Fundamentals

Choose masks (m_I, m_O)

Compute $(m_I \odot I) \oplus (m_O \odot F(I))$ for all I

Obtain N_0 zeroes and N_1 ones

We have $N_0 + N_1 = 2^n$. Focus on $N_0 - N_1$

If $|N_0 - N_1|$ is large, linear comb. is biased

Bias: $\varepsilon_{(m_I,m_O)} = \dfrac{N_0 - N_1}{2^n}$

We have $(m_I \odot I) \oplus (m_O \odot F(I)) = 0$ with prob $(1 + \varepsilon_{(m_I,m_O)})/2$

Notation: $m_I \mapsto_\varepsilon m_O$

Rule 1: $m_I = 0 \mapsto_{\varepsilon=1} m_O = 0$

Rule 2: For an affine transformation L, $m_I \mapsto m_O$ has biais -1, 0 or 1

Rule 3 (heuristic):

$m_I \mapsto_{\varepsilon_1} m_M$ and $m_M \mapsto_{\varepsilon_2} m_O$

$\Longrightarrow \quad m_I \mapsto_{\varepsilon_1\varepsilon_2} m_O$

15

In linear cryptanalysis, we study the relation between inputs and outputs in a different way. We choose two bit strings m_I and m_O (that we will call input and output mask) and, for input/output pairs $I - O$, we study the distribution of the quantity (m_I scalar I) XOR (m_O scalar O). In other words, we mask the input bits with the mask m_I and the output bits with m_O, we count the number of '1' in the masked bit strings and compute the parity of this number, namely we reduce the count modulo 2. Over all inputs, let say that we observe a zero result N_0 times and a one result N_1 times. Of course, $N_0 + N_1$ is equal to 2^n, the total number of possible inputs. Thus, the difference $N_0 - N_1$ encodes all the information we need about N_0 and N_1. We now focus on this difference.

When its absolute value is small, the result is balanced, that is zeroes and ones occur roughly as often as each other. When it is large, there are much more zeroes or ones, depending on the sign of the difference. It is convenient to normalize the difference and consider the bias epsilon of (m_I, m_O) which is defined as the difference divided by 2^n. Then, the probability to observe a zero is equal to (the bias +1) divided by two. Using a notation reminiscent of the one from differential cryptanalysis, we will write an arrow from m_I to m_O with the bias epsilon written as a subscript. To avoid confusion, we use a different arrow in the notation.

As before, there are some important rules that always hold. First, for input and output masks equal to zero, we don't observe any of the input bits or output bits. As a consequence, the number of ones in this empty observation is always equal to zero and its parity is always zero. Thus, the bias of this 0 to 0 arrow is equal to 1.

For linear (and also affine) transformations, all arrows have bias equal to either 0, 1 or -1. Showing this is a good exercise to become more familiar with the notion.

The third rule is heuristic and consider a chain of two arrows, where the output mask of the first arrow is equal to the input mask of the second. Then we have a composed arrow whose bias is the product of the biases of the initial ones.

These rules can be used to construct linear characteristics for block ciphers built from many sub-components.

9.8.2 Small example

	ADD
00	00
01	01
10	01
11	10

Input Mask

Output Mask	00	01	10	11
00	4	0	0	0
01	0	0	0	4
10	2	2	2	-2
11	-2	2	2	2

$N_0 - N_1$

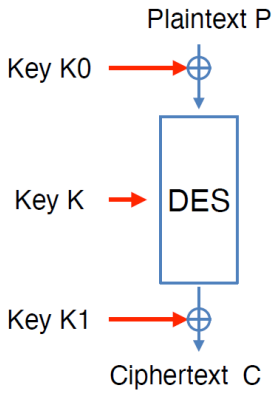
We now revisit our small example for linear cryptanalysis. The first line is greyed out since the case of a zero output mask never provide any useful information because, in that case, we are not observing any of the output bits. On the other hand, the first column where we do not observe the input bit can yield interesting information. For exemple, the -2 at the bottom of the first column tells us that the XOR of the two output bits of an addition is more often equal to 1 than to 0. The red 4 tells us that there is a linear relationship for input mask 11 and output mask 01. Indeed, the low order bit of a sum is equal to the XOR of the two input bits.

9.8.3 Principle of attack

- Assume we have $m_1 \mapsto_\epsilon m_0$ for a block cipher (with $\epsilon >> 2^{-n}$)
  - Encrypt with inputs I with output O, compute $(m_I \odot I) \oplus (m_O \odot O)$
  - For the studied cipher, large difference between num of zeroes and ones
  - For a random permutation, close to balance between zeroes and ones
- This yields a known plaintext distinguisher from a random permutation!
- For key recovery, we can take advantage of key mixed in using XORs

The principle of using linear cryptanalysis as a distinguisher is basically the same as the one we saw for differential cryptanalysis.
However, for key recovery attacks, there is a very interesting twist. This comes from the fact that in most block cipher, the key is incorporated by XOR it with intermediate values every now and then during the encryption process. If we know a linear characteristic for a part of the cipher and want to XOR a key after this part, we can observe that this changes the value of the output scalar product by a constant which is the parity of the number of ones in the corresponding key masked with the output mask. This also composes well and as a consequence, the absolute value of bias of the global linear characteristic is unaffected by this specific way of introducing the key. Only the sign will change (or not) depending on the value of one parity bit of the expanded key (coming from the key schedule).
Of course, it is also possible to use the partial exhaustive search technique we described for differential cryptanalysis.

9.9 Related Key attacks : Example of DES-X



Attack Model
The attacker can ask for encryptions with different but related keys.

Easy DES-X distinguisher

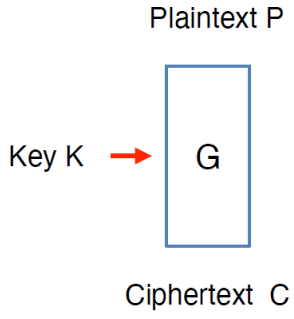
$$\text{DES-X}(K_0, K, K_1, P) = \text{DES-X}(K_0 \oplus 1, K, K_1, P \oplus 1)$$

Also on DES (or triple DES)

$$\overline{\text{DES}(K, P)} = \text{DES}(\bar{K}, \bar{P})$$

As a last example of block-cipher attacks, we now consider a new method called the related key attack. It is a different attack model where the attacker can not only query the block cipher with its basic key but also with different but related keys obtained by XORing the basic key with values which are arbitrarily chosen by the attacker. It is also possible to replace XOR by another operation such as addition but care needs to be taken when defining the attack model.
Related-key distinguishers arise naturally with some block ciphers. A first example is the DES-X cipher proposed by Rivest to enlarge the key size of DES. It consists in a sandwich structure where before and after DES encryption the plaintext and ciphertext blocks are XORed with two whitening keys K0 and K1. By itself it is an interesting technique that led to the Even-Mansour construction. However, with DES-X, a related-key distinguisher can easily be built. For example, it is clear that XORing any constant (let say 1) with both the plaintext block and the first whitening key while leaving everything else unchanged doesn't change the ciphertext.
Interestingly, there also is a simple related key distinguisher on DES itself. Indeed, negating every bit of the plaintext and of the key at the same time creates a ciphertext which is the negation of the initial one. Related-key attacks may or may not be a problem in applications, it really depends on how the block-cipher is used. However, block-cipher designers usually try to avoid them in order to prevent bad interactions in applications that could be vulnerable otherwise.

9.10 Arbitrary related key attack model is too strong



Assume we can do related key queries with both XOR relations and ADD relations.
Then:

$$K^{(0)} = 0 \implies G(K+1, P) = G(K \oplus 1, P)$$

This yields a generic bit-by-bit key recovery

Attack model is too strong !

As mentioned in the previous slide, some care needs to be taken when considering related-key attacks. Indeed, if the attacker is allowed to use arbitrary transforms of the initial key, there is an easy generic key recovery attack that becomes possible. To show that assume that the adversary is allowed to either XOR the initial key with a constant of its choice or to ADD a chosen constant to the key (modulo 2^k where k is the key size).
Remark that XORing the constant 1 to a key K and adding 1 to the same key K give the same result if and only if the lowest order bit of the key is a zero. Moreover, testing if two keys are equal is easy to do by encrypting any plaintext block with both keys and comparing the resulting ciphertext blocks. Thus, the adversary learns the first bit of the key. By adding and XORing 2, 4, 8 and so on, he can also learn the remaining bits (with the exception of the top bit). Of course, this exception is not a problem since once all the bits of the key except the last one are known, it suffices to test which of the two remaining possibilities is the correct key.
As a consequence, when using related key attacks, one should always check that the model hasn't been pushed in an extreme regime where this kind of generic attacks are possible.

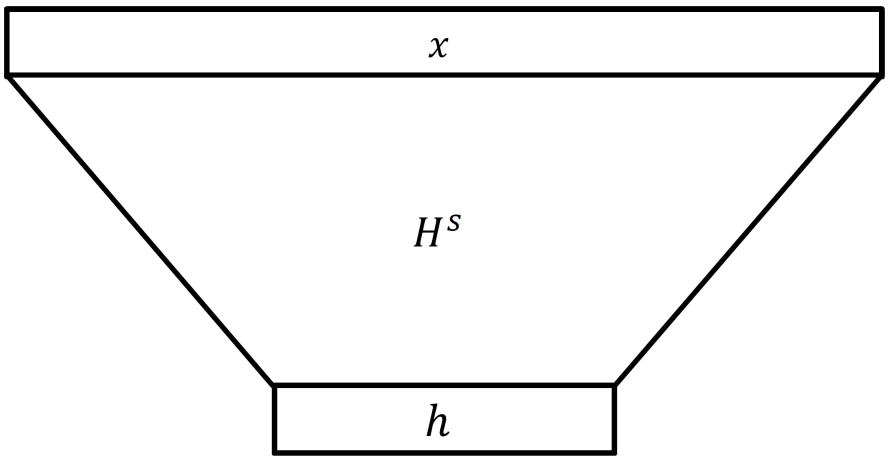
Hash Functions

10.1 Fingerprinting

- Problem: You have a large file/object and want to compare to objects in a database
- E.g. check if you object is in the database
- Object too large to send to the server
- $\Rightarrow$ Need a small unique identifier/digest of your object
- **Hashing!**
- Compress objects into a small fingerprint
- $\Rightarrow$ Speed up comparisons/membership tests, smaller communication cost

Definition 16 (Hash Functions)**.**
Syntax: *A family of hash functions is given by a pair of algorithms* (Gen, H)

- $Gen(1^\lambda)$*: A probabilistic algorithm which on input* 1^λ *outputs a key* S
- $H^S(x)$*: A deterministic algorithm which on input a key* S *and a string* $x \in \{o,1\}^*$ *outputs a hash value* $h \in \{0,1\}^l$

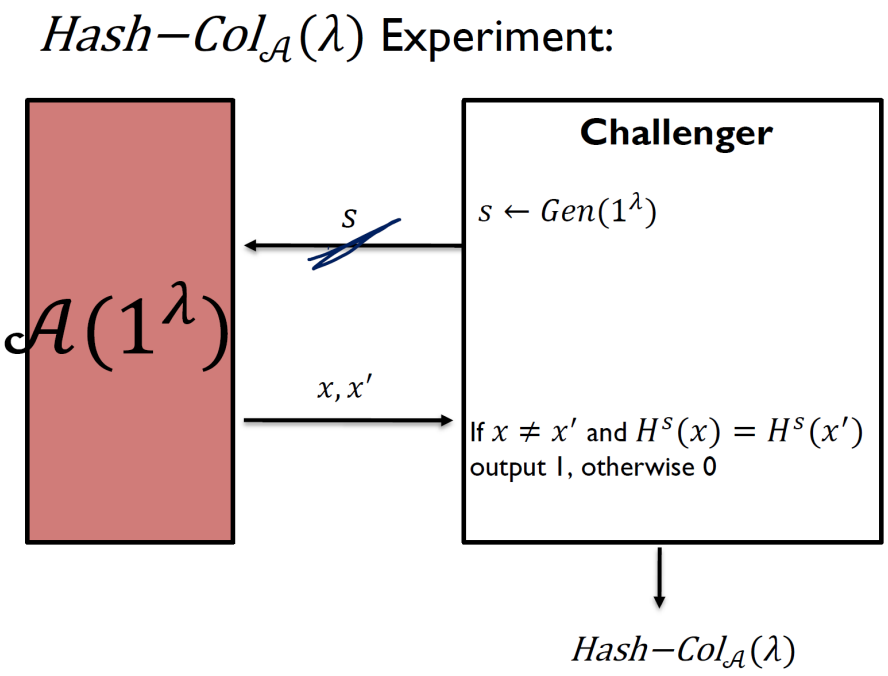


Remarks:

- *The output length* $l = l(\lambda)$ *only depends on* λ
- *If* H^S *is only defined on inputs of length* $l' > l$ *we call* H *a fixed-length hash function*
- *If we don't specify Gen it just chooses uniformly random* $s \leftarrow_\$ \{0,1\}^\lambda$

Definition 17 (Collision Resistance)**.** *Let $H : \{0,1\}^{l'} \to \{0,1\}^{l}$ with $l' > l$.*
A hash function (Gen, H) is called collision-resistant, if for every PPT-bounded adversary $\mathcal{A}$ there exists a negligible function v s.t. for all $\lambda \in \mathbb{N}$

$$Pr[Hash - Col_{\mathcal{A}}(\lambda) = 1] < v(\lambda)$$



10.1.1 Examples

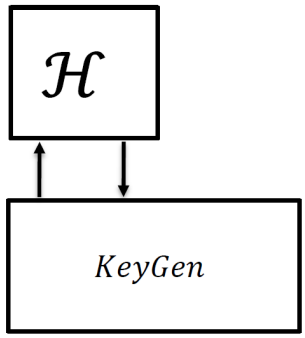
- Assume H is a collision resistant hash function
- Is H' given by $H'^S(x) = H^S(x||0^\lambda)$ also collision resistant?
  - Yes!
  - Idea: Collision x, x' for H' yields collision $x||o^\lambda$, $x'||0^\lambda$ for H
- Is H'' given by $H''^S(x_1...x_n) = H^S(x_1...x_{n-1})$ collision resistant?
  - No!
  - 0...00 and 0...01 hash to the same value

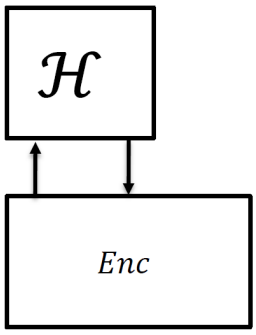
10.2 Idealized Hash Functions

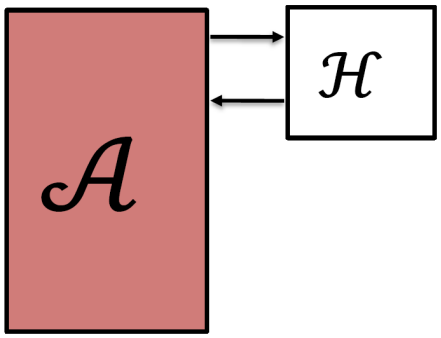
- Sometimes collision resistance isn't enough…
- Sometimes we need hash functions for which it is hard to find arbitrary correlations
- E.g. for a fixed function f it should be hard to find an input x with $H^S(x) = f(x)$
- How should an ideal hash function behave?
- Like a random function!

10.3 The Random Oracle Model

- Random Oracle (RO): Uniformly random function $\mathcal{H} : \{0,1\}^l \to \{0,1\}^\lambda$ which can only be accessed via oracle access/blackbox access
- Random oracle is uniform on positions it has not been queried on
- Models a hash function whose code no-one knows
- Rationale: The only way of learning something about a hash function which is modeled by RO is to evaluate it
- To evaluate $\mathcal{H}$ adversary needs to provide input explicitly to random oracle
- Idea: In security proofs reduction controls RO!



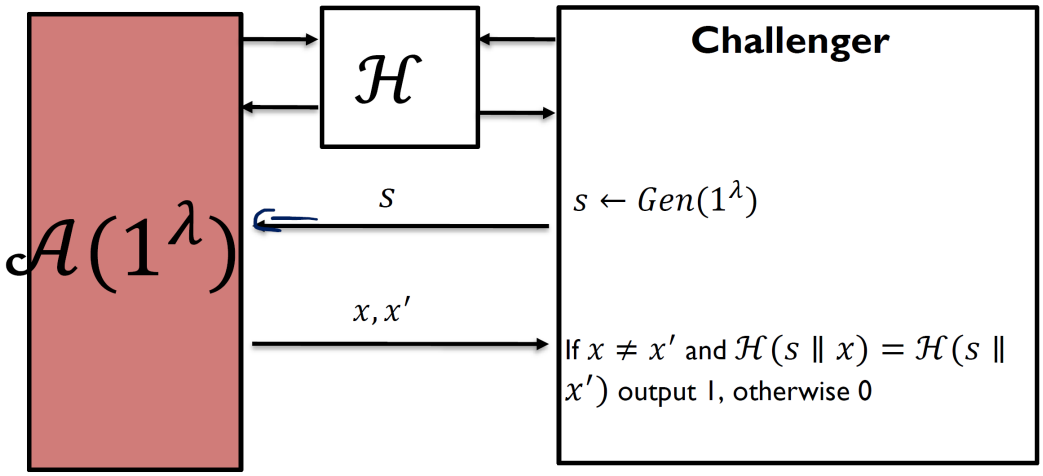




- Obviously, real hash functions are not random oracles
- E.g. real hash functions have small description (e.g. program)
- Sometimes, we only know how to prove constructions secure if $\mathcal{H}$ is modeled as random oracle
- Two step approach:
  - Model hash function as a RO in the security proof
  - In the real world, instantiate with actual hash function
- Better than no proof at all!
- Proofs in RO model are heuristic
- If a scheme is secure in the RO model, but insecure in real world, adversary must do something non trivial and interesting with hash function

10.3.1 Examples

Random Oracle yields collision resistant hash function $H^S(x) = \mathcal{H}(s||x)$



Proof. Assume $\mathcal{A}$ makes at most $q = poly(\lambda)$ queries $x_1, ..., x_q$ to $\mathcal{H}$.
$\mathcal{H}$ is uniformly random function, i.e. all function values are uniform and independent.
Probability of single collision:

$$Pr[\mathcal{H}(s||x_i) = \mathcal{H}(s||x_j)] = 2^{-\lambda}$$

It follows that

$$Pr[\exists i, j : \mathcal{H}(s||x_i) = \mathcal{H}(s||x_j)] \leq q^2 2^{-\lambda} = negl(\lambda) \text{(union bound)}$$

□

10.4 Summary

- Hash functions compress objects into short digests
- Digests are *computationally* unique
- This is captured in the collision resistance property
- Random oracles model ideal hash functions with stronger properties
- RO model allows for easy proofs!
- Random oracle model is a heuristic: Proofs in the random oracle model don't necessarily hold in the real world.

Definition 18 (Security Definition)**.**
The collision-finding experiment $Hash - coll_{\mathcal{A},(Gen,H)}(n)$

- *On input* s, $\mathcal{A}$ *outputs* x *and* x'.
- $Hash - coll_{\mathcal{A},???(05-02,3)}(n) = \begin{cases} 1 & x \neq x' \text{ and } H^S(x) = H^S(x') \\ 0 & otherwise \end{cases}$

In the above experiment,

- $\mathcal{A}$ *is polynomial time bounded;*
- *if no efficient adversary can find a collision except with negligible probability, then* $(Gen; H)$ *is collision resistant.*

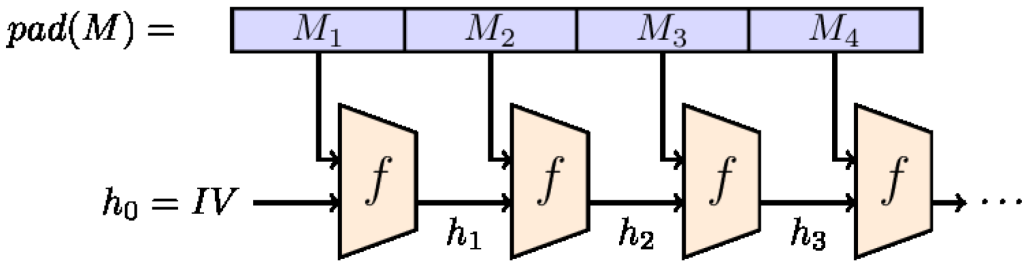
Some weaker security notions are also considered:

- *second-preimage resistance: given* s *and a random* x*, it is infeasible for a probabilistic polynomial-time adversary to find* $x' \neq x$ *such that* $H^S(x') = H^S(x)$.
- *preimage resistance: given* s *and a the hash* $y = H^S(x_0)$ *of a random* x_0*, it is infeasible for a probabilistic polynomial-time adversary to find* x *such that* $H^S(x) = y$.

10.5 A word on Provable Security

- Hash functions in the real world are generally unkeyed and have a fixed output length.
- Problematic from a theoretical standpoint.
- Current hash functions are still collision resistant since no colliding pair is known, and would be computationally difficult to find.
- A hash function with output length n is considered secure as long as:
  - no known algorithm can find a colliding pair in time much smaller than $2^{n/2}$;
  - no known algorithm can find a preimage or second preimage in time much smaller than 2^n.
- Provable security for hash functions follows the same patterns as provable security for block ciphers.

10.6 The Merkle-Dåmgard construction



Source: TikZ for Cryptographers `https://www.iacr.org/authors/tikz/`

- (Gen, f) is a fixed-length hash function for inputs of length $2n$ and outputs of length n.
- (Gen, F) (as defined above) operates on strings of length $L < 2^n$.
- To compute $pad(M)$: pad the message M with 0 until its length is a multiple of n, then add a final block that corresponds to L encoded as an n-bit string.

- IV is called the *initialization vector*, and is an arbitrary constant.

Theorem 6. *If (Gen, f) is collision resistant, so is (Gen, F).*

Proof. Proofsketch:

- Assume you know $x \neq x'$ such that
  - $F^S(x) = F^S(x')$;
  - x and x' are of respective length L and L' (in bits);
  - $M = pad(x)$ and $M' = pad(x')$ are of length B and B' (in blocks).
- Case $L \neq L'$: one has $f^S(h_B||L) = f^S(h'_{B'}||L')$, which gives a collision under f^S.
- Case $L = L'$:
  - let $l_i = h_{i-1}||M_i$, $l'_i = h'_{i-1}||M'_i$, and $I_{B+2} = F^S(x) = F^S(x') = I'_{B+2}$;
  - let i_0 be the largest index for which $l_{i_0} \neq l'_{i_0}$. Necessarily, $i_0 \leq B + 1$.
  - by maximality of i_0, $l_{i_0} \neq l'_{i_0+1}$, thus $f^S(l_{i_0}) = f^S(l'_{i_0})$, and $l_{i_0} \neq l'_{i_0}$, which gives a collision under f^S.

□

Is the previous theorem sufficient in practice?

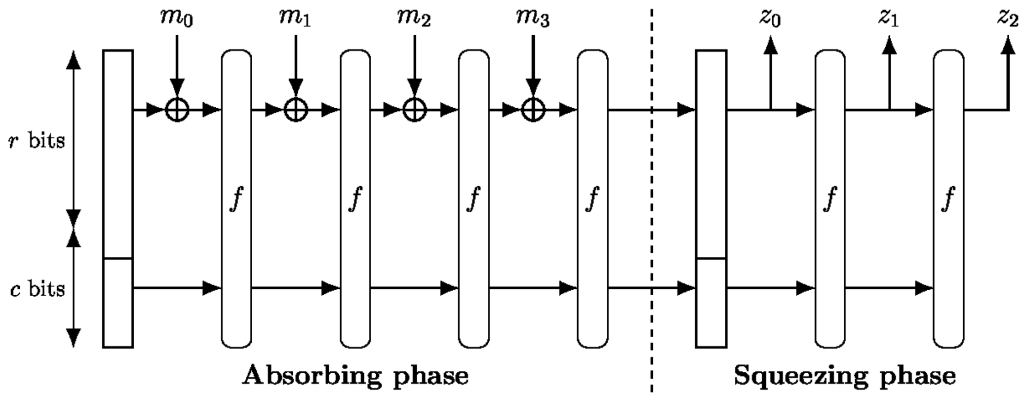
- Second preimages of (very) long messages can be found much faster than exhaustive search (but it still requires finding collisions for the hash function).
- Length extension: given the hash of an unknown message x, it is trivial to compute the hash of $pad(x)||y$ for any message y.
- Length extension attacks have actually been used in practice: one has to be careful when using a hash function based on the Merkle-Dåmgard construction in practice.

10.7 The Design of Compression Functions

General structure of the compression functions that we are going to deal with in the examples:

- The function operates over w-bit words.
- Its input is divided in two parts: a state, which stores the output of the previous call to the compression function, and a block of n_c words of padded data.
- Compression functions are generally iterative: a simple round is repeated r times.
- A linear function extends the n_c words of data into r words of extended data.
- At each round, the state is updated with a nonlinear function, and a round constant and a word of extended data are absorbed in the state.

10.8 Sponge Functions



Source: TikZ for Cryptographers `https://www.iacr.org/authors/tikz/`

- Generic structure to build a hash function from a public and unkeyed function f.
- f is usually a permutation.
- The b-bit function is repeatedly applied to a state consisting in:
  - a r-bit outer state, where r is called the rate;
  - a c-bit inner state, where $c = b - r$ s called the capacity.
- Padding of a message can be done by appending a single bit 1, followed by bits 0 until the length of the result is a multiple of r.

10.9 A word on Provable Security

How to justify the soundness of the sponge construction?

- The function f is fixed and public. To study the structure, we model it as a uniformly random function (or permutation). The resulting construction is called a random sponge.
- How close is the random sponge from a random oracle?

Definition 19. *Let D be a probabilistic algorithm that deals at most q oracle queries. Its distinguishing advantage is defined as*

$$\mathcal{A}^{dist}_{Sponge}(D) :- |Pr[D^{Sponge[f](\cdot)} = 1] - Pr[D^{H(\cdot)} = 1]|,$$

where the first probability is taken over the uniformly random draw of $f : \{0,1\}^b \to \{0,1\}^b$ and the second one over the uniformly random draw of the function H.

- However, for a concrete hash function, f is public!
- An adversary has access to any intermediate value.
- Hence, we have to give access to f when the adversary is interacting with $Sponge[f]$.
- In the random function case, a simulator is defined. It is a probabilistic polynomial time algorithm with access to the random oracle, whose goal is mimic the internal function of a random sponge.

Definition 20. *Let D be a probabilistic algorithm that deals at most q oracle queries. Its differentiating advantage is defined as*

$$\mathcal{A}^{diff,S}_{Sponge}(D) :- |Pr[D^{Sponge[f](\cdot),f(\cdot)} = 1] - Pr[D^{H(\cdot),S[H](\cdot)} = 1]|,$$

where the first probability is taken over the uniformly random draw of $f : \{0,1\}^b \to \{0,1\}^b$ and the second one over the uniformly random draw of the function H.

Theorem 7. *There exists a simulator S such that, for every a probabilistic algorithm D whose queries require at most $q < 2^c$ calls to f, one has*

$$\mathcal{A}^{diff,S}_{Sponge}(d) \leq 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^c}) \lessapprox \frac{q(q+1)}{2^{c+1}}$$

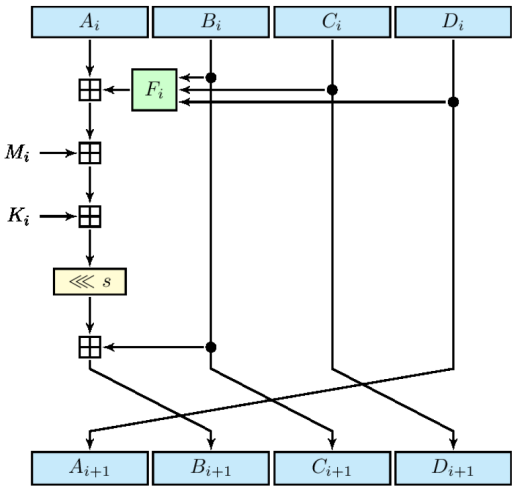
Interpretation of theorem 7:

- It does not provide actual security guaranties for any concrete hash functions.
- But it does justify the soundness of the structure.
- It also provides a lower bound on the capacity: $c \geq 2n$, where n is the output length.

10.10 Examples

10.10.1 Example 1: The MD5 Hash Function

- Designed by Rivest in 1991.
- Produces a 128-bit hash.
- Built using the The Merkle-Dåmgard construction.
- Considered obsolete: collisions can be found in 2^{18} operations.
- Compression function takes as input the concatenation of the previous 128-bit output and the next 512-bit of padded message.

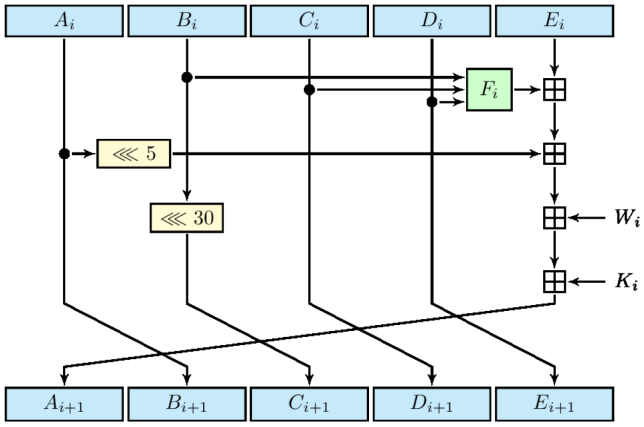


Source: TikZ for Cryptographers `https://www.iacr.org/authors/tikz/`

- MD5 compression function consists in 4 rounds of 16 MD5 operations.
- A_i, B_i, C_i, D_i are the four 32-bit words that form the state.
- F_i is a non-linear function that uses bitwise NOT, XOR, AND and OR operations. It depends on the round number.
- The "square with cross"-symbol denotes addition modulo 2^{32}.
- At each operation, a 32-bit word M_i of data and a constant K_i are absorbed in the state.

10.10.2 Example 2: The SHA-1 Hash Function

- Designed by the United States National Security Agency in 1995.
- Closely related to the MD5 hash function.
- Considered obsolete: collisions are known, and can be found in around $2^{63.1}$ SHA1 evaluations.
- Produces a 160-bit hash.
- Compression function takes as input the concatenation of the previous 160-bit output and the next 512-bit of padded message.

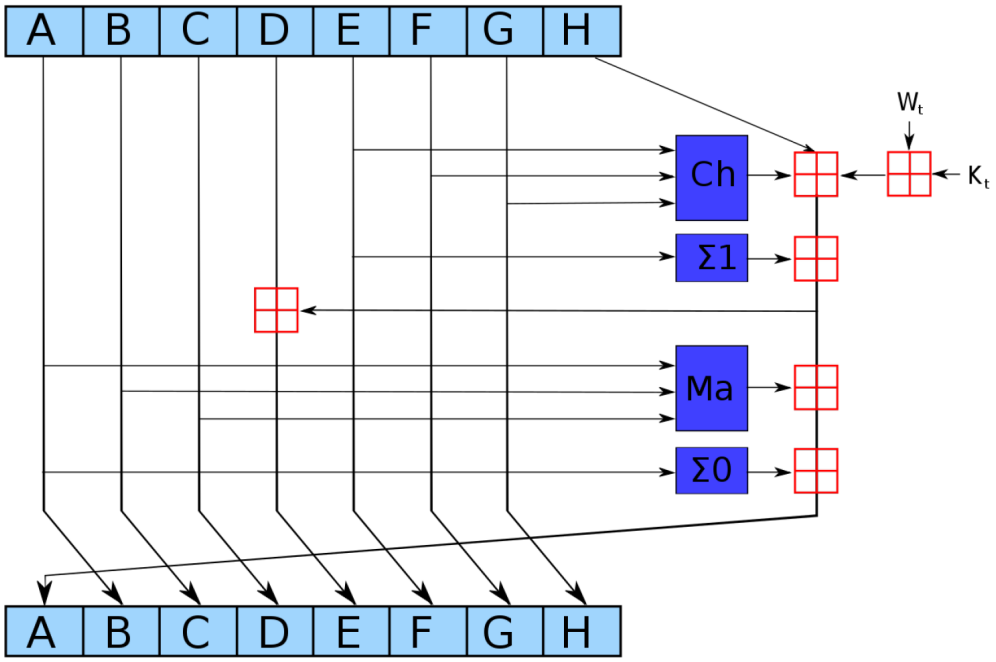


Source: TikZ for Cryptographers `https://www.iacr.org/authors/tikz/`

- SHA-1 compression function consists in 80 rounds. The new 512 bits of data are linearly extended to 80 32-bit words $(W_i)_{1 \leq i \leq 80}$.
- A_i, B_i, C_i, D_i are the five 32-bit words that form the state.
- F_i is a non-linear function that uses bitwise NOT, XOR, AND and OR operations. It depends on the round number.
- At round i, a 32-bit word W_i of data and a constant K_i are absorbed in the state.

10.10.3 Example 3: The SHA-2 family of Hash Functions

- Designed by the United States National Security Agency in 2001.
- Evolution of the SHA-1 hash function.
- Consists in a family of 6 hash functions with output lengths of 224, 256, 384 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.
- Actually, only 2 hash functions: SHA-256 and SHA-512, the rest are truncated versions with different initial values.
- Both use similar compression functions. They differ by the size of the words on which they operate (32 bits and 64 bits), the number of rounds (64 and 80), and the round constants.
- Still considered secure nowadays.



Source: Wikipedia `https://en.wikipedia.org/wiki/SHA-2`

- $A, ..., H$ are the 8 words of the state (block length is still 16 words).
- $Ch(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$
- $Ma(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$
- $\sum_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$
- $\sum_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$
- Bitwise rotation constants differ between both round functions (here SHA-256).

10.10.4 Example 4: The SHA-3 family of Hash Functions

- United States National Institute of Standards and Technology (NIST) standard adopted in 2015 after an open competition.
- Based on the winner of the competition: Keccak (designed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche).
- Based on the sponge construction with the $10*1$ padding. All instances use the same Keccak-f[1600] permutation.
- Actually a set of 4 algorithms:

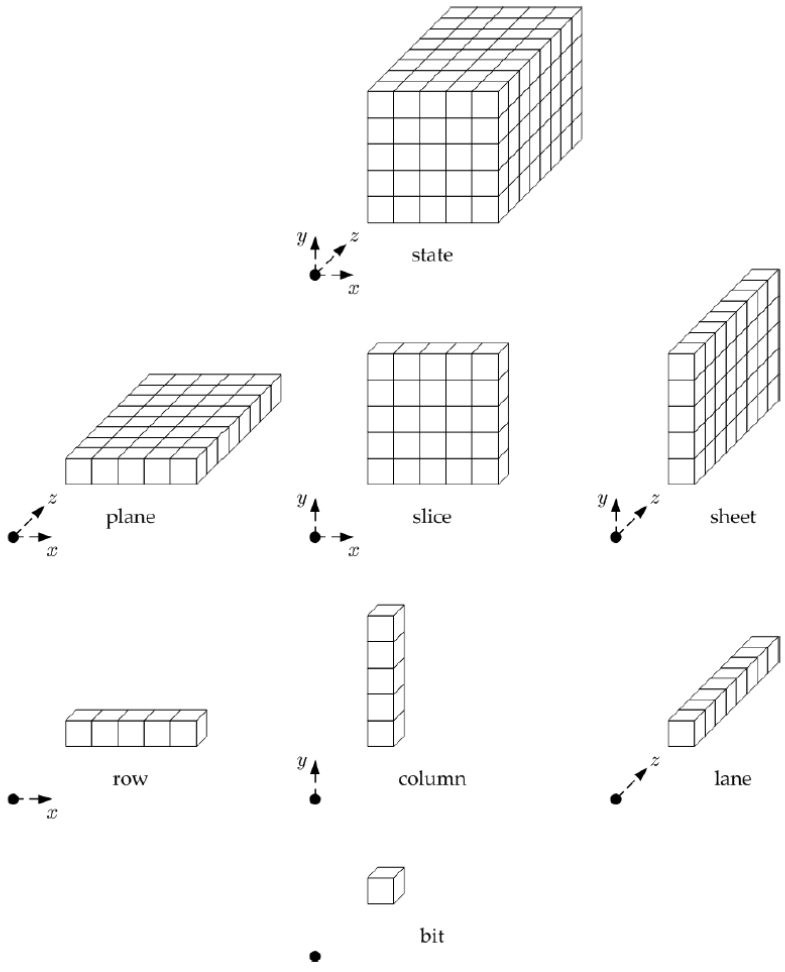
Instance	Output size	Rate	Capacity	Definition
SHA3-224(M)	224	1152	448	Keccak[448](M || 01, 224)
SHA3-256(M)	256	1088	512	Keccak[512](M || 01, 256)
SHA3-384(M)	384	832	768	Keccak[768](M || 01, 384)
SHA3-512(M)	512	576	1024	Keccak[1024](M || 01, 512)

- Keccak[c]$(\cdot, d)$ denote the Keccak sponge, with capacity c, and output size d.
- Current standard, considered highly secure.

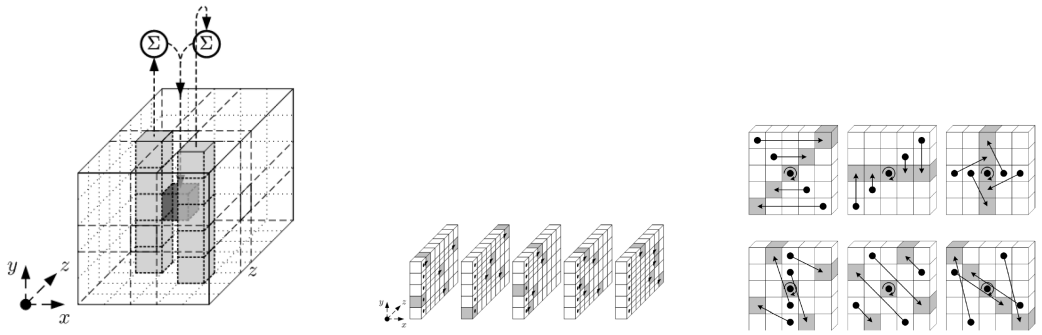
```
Keccak-f[b](A) {
  for i in 0,...,n-1
    A = Round[b](A, RC[i])
  return A
}

Round[b](A,RC) {
  # θ step
  C[x] = A[x,0] xor A[x,1] xor A[x,2] xor A[x,3] xor A[x,4],    for x in 0,...,4
  D[x] = C[x-1] xor rot(C[x+1],1),                             for x in 0,...,4
  A[x,y] = A[x,y] xor D[x],                                 for (x,y) in (0...4,0...4)
  # ρ and π steps
  B[y,2*x+3*y] = rot(A[x,y], r[x,y]),                       for (x,y) in (0...4,0...4)
  # χ step
  A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y]),        for (x,y) in (0...4,0...4)
  # ι step
  A[0,0] = A[0,0] xor RC
  return A
}
```

- 7 different variants for $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
- The state consists in 25 words of size $w = \frac{b}{25}$.
- The number n of rounds is computed as $n = 12 + 2l$ where $w = 2^l$.

10.10.4.1 The Keccak-f[200] Permutation



Source: https://keccak.team/figures.html



(a) The θ step. (b) The ρ step. (c) The π step.

Source: `https://keccak.team/figures.html`

- Linear mixing layer of the Keccak-f[200] permutation.
- Operates similarly for the other variants.
- Goal: ensure good diffusion. Similar to the linear layer in block cipher design.

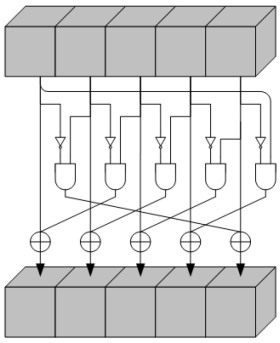


Figure: The χ step. Source: `https://keccak.team/figures.html`

- Non-linear layer of the Keccak permutation. Operates at the lanes level, but operations are actually bitwise.
- Followed by the ι step: the addition of a round constant to the first lane.
- Without ι, the permutation would commute with bitwise rotation of the lanes, and all the rounds would be equal.

10.10.4.2 The avalanche effect of the Keccak-f[1600] Permutation:

- Diffusion property of the linear layer: a 1-bit difference is propagated to 11 different bits then to the input of 11 χ functions (at the bit level).
- Property of the non-linear step: a 1-bit difference in the input of χ gives at least a 1 bit difference in its output.
- Experimental numbers: given a 1-bit difference in input, around half the bits of the state are different at the end of the third round.

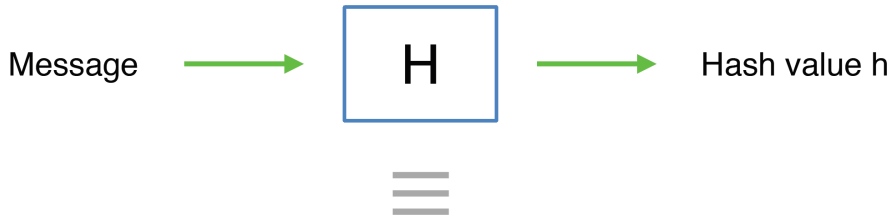
10.11 Cryptanalysis on Hash Functions

10.11.1 Attacker's goals against hash functions

- Recall that a hash function is a map $H : \{0,1\}^* \mapsto \{0,1\}^n$
- Possible goals to achieve:
  - Collisions: Find distinct bit strings x, y such that $H(x) = H(y)$
  - Pre-image: Given $h \in \{0,1\}^n$, find x such that $H(x) = h$
  - Second pre-image: Given x, find $y \neq x$ such that $H(x) = H(y)$
  - One-wayness: given a set S of size at most 2^n and $h = H(s)$ for $s \in S$. Find a $s' \in S$ such that $H(s') = h$
- Of all these goals, collisions is usually the easiest to attack

Hash functions are widely used in cryptography. So much that they are sometimes referred to as the cryptographic swiss-knife. Because of this, they need to satisfy several essential properties. From the point of view of the attacker, this gives more attack targets. The most well-known target is the search for collisions where the attackers looks for two different strings that hash to the same value. Then we find the search for pre-images where he needs to find a message that hashes to a given target. There are two frequent variations on this. Second pre-images, where given a message the attacker needs to find a distinct message with the same hash value. And one-wayness, where given the hash of a message from a relatively small set, he has to find a message from the same set hashing to the prescribed value. In that case, the original message is the desired target but any other message that collides with it is also accepted.
Among these goals, finding collisions is usually the easiest, and it will be our main topic during the lecture.

10.11.2 Random Oracle Model



Should behave like a random function

A frequently encountered idealisation of hash functions is the random oracle model where the output value is chosen uniformly at random for every new message. (Of course, since hash functions are deterministic, the answer to an already asked question should remain identical). In other words, the hash function should behave like a random function. This is impossible to achieve, since a random function almost never has a short description as a computer program, while a hash function does by design.
As a consequence of this it is possible to construct cryptographic protocols that are secure when using a random oracle but become insecure as soon as it is replaced by any concrete hash function. Admittedly, these examples are contrived and the random oracle model remains a good way to create sound designs.

10.11.3 Generic Collisions

- The key ingredient is the *birthday paradox*
- If we have a set of size N and random elements:

- Collision guaranteed after picking $N + 1$ (pigeonhole principle)
- Collision with constant probability after $\mathcal{O}(\sqrt{N})$
- Collision with overwhelming probability after $\mathcal{O}(\log N \cdot \sqrt{N})$

Even with idealised hash functions, collisions are possible. Indeed, the set of possible messages is infinite while the set of hashed values is finite, as a consequence, collisions must exist. In fact, if the size of the set of hashed values is N, a collision is guaranteed as soon as we have hashed $N + 1$ messages. This is the pigeonhole principle and it also holds for randomly chosen values.
However, and may be surprisingly, it is possible to do much better. If we only want a collision to occur with constant probability or even with overwhelming probability, we only need to consider a number of messages of the order of square-root of N or slightly larger.
This fact is usually known as the birthday paradox because it applies when looking for people with common birthday in a relatively small group.

10.11.4 Explanation of birthday paradox

- Picking s elements in N
  - Number of ordered choices (with collision): N^s
  - Number of ordered choices (without collisions): $\frac{N!}{(N-s)!}$
- Example for $N = 365$:

S	22	23	60
Total	$2.35 \cdot 10^{56}$	$8.57 \cdot 10^{58}$	$5.46 \cdot 10^{153}$
No collision	$1.23 \cdot 10^{56}$	$4.22 \cdot 10^{58}$	$3.21 \cdot 10^{151}$
Ratio	**52 %**	**49 %**	**0.6 %**

The reason for this is combinatorial in nature. Assume that we are picking s elements in a group of N. The number of possible ways of choosing is N to the s, if duplicates are permitted. If they are not, they are fewer options, namely N factorial over $(N - s)$ factorial. Let's do the concrete computations for $N = 365$, i.e., for the case of birthdays (omitting leap years). For $s = 22$, we see that about 52% of the cases are without collisions and for $s = 23$, only 49% are without collisions. As a consequence, the probability of getting a collision is higher than one half. Furthermore, for $s = 60$, the probability is higher than 99% (of getting a collision).

- We can simplify the probability as:

$$p(s) = \prod_{i=0}^{s-1} \frac{N - i}{N}$$

- Remember that

$$ln(1 - x) \leq -x \text{ (for all } x < 1)$$

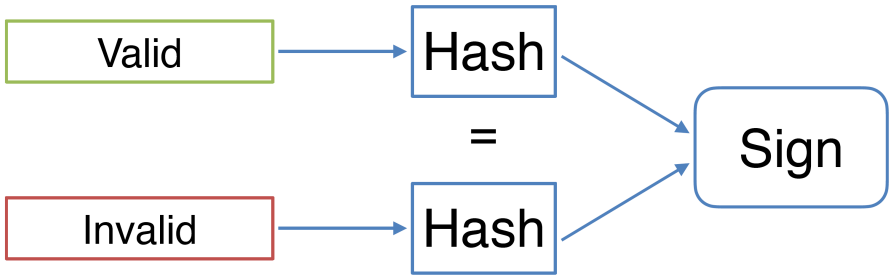
- Taking logarithm we have:

$$ln(p(s)) \leq -\sum_{i=0}^{s-1} \frac{i}{N} = -\frac{s(s-1)}{2N}$$

To show this, we can rewrite the probability of not having a collision as N factorial over $(N-s)$ factorial divided by N to the s. Rearranging terms, we get the product of value $N-i$ over N for i ranging from 0 to $s-1$. Taking log, we turn this into a sum. Remembering that the log of $(1-x)$ is always smaller than $-x$, we can upper bound the log of the probability of not having a collision by $-s(s-1)$ over $2N$. When s becomes bigger than square root of N, this bound becomes a negative constant and the probability of not having a collision is bounded away from 1. Furthermore, if s is of the form square root of N times $log(N)$, the bound tends to minus infinity and the probability of not having a collision becomes as close to 0 as we desire when N grows. In other words, the probability of having a collision is overwhelming.

10.11.5 An example scenario for exploiting collisions: digital signature

- In the standard Hash-and-sign paradigm



Hash collision can thus lead to signed unwanted messages

Collisions can, in particular, be used to forge digital signatures. Indeed, if an attacker can prepare two messages with the same hash value, one of which can be accepted as valid by the signer while the other would be rejected, then (in the classical hash-and sign paradigm) he can use the signature obtained for the valid message and pretend that it is a signature of the other one. This is clearly a forgery.

10.11.6 More details

Valid	Invalid
Alice Wonderland owes Bob Sponge 50 euros	*Alice Wonderland owes Bob Sponge 500000 euros*

Attacker introduces variations in the messages

E.g. Alice Wonderland vs. Wonderland Alice (or Bob)

euros, Euros, euro, Euro, €

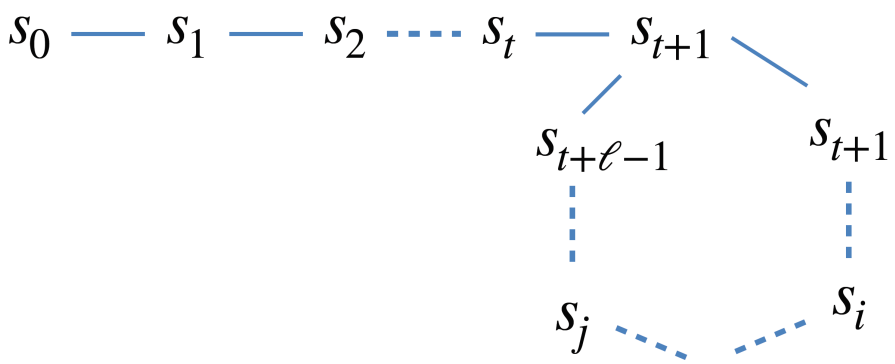
Add punctuation, change sentence

Even with moderate length messages, plenty of possibilities

In practice, preparing messages for the attack relies on the fact that given a message in a natural language, it is possible to create plenty of variations of the message without altering its meaning. In our example, Alice is ready to sign a letter saying that she owes Bob 50 euros. However, Bob want to change that to half a million. We show on the slide a few ways of changing the sentence without changing its meaning. The good thing for the attacker is that changes are mostly independent which means that the number of variation are multiplied. In particular, identifying 40 places in the message where 4 variations are possible is enough to get 2^{80} variations. By the birthday paradox, this would be enough to be able to find collisions for 160 bit hash functions. Of course, the computational cost for Bob to perform this generic attack would be high.
This size of 160 bits was the standard for a long time and was used in the SHA-1 hash function. Nowadays, 256 bits or even 512bits are preferred.

10.11.7 Memory-less collision finding

- Also a major algorithmic tool in cryptanalysis
- Consider a random function $f : S \mapsto S$
- We want a collision $f(s) = f(s')$
- Pick random and build sequence $s_{i+1} = f(s_i)$
- Since is finite, the sequence is ultimately periodic



We saw in the block cipher cryptanalysis lecture that collisions can be found quickly using fast sorting techniques. However, this approach requires a lot of memory and this memory cost is usually the limiting factor for the attacker. We now consider the problem of finding collisions without using large amounts of memory.
We assume that f is a random function from a set to itself and we search for a collision $f(s) = f(s')$.
The trick is to start from a random point s_0 and consider the sequence defined by the recursive formula $s_(i + 1) = f(s_i)$. Of course, since the sequence is infinite, the values it takes must cycle back at some point. Note that, there is no reason, in general, to cycle back to s_0 and most of the time, the sequence looks that the picture on the slide. There is a tail, followed by a cycle and the sequence is ultimately periodic.
It is clear that the entry point in the cycle has two distinct pre-images, one in the cycle and one in the tail. Thus, there is a collision at this location.
We now need an algorithmic technique to efficiently find this entry point and compute the collision.

- We want to find the entry points to the cycle
- Without storing all intermediate computations
- Many methods, we consider Floyd's cycle finding technique

- Idea: Compute in parallel s_n and s_{2n}
  - At some point, we will get a collision $s_n = s_{2n}$, with $l|n$
  - We thus obtain a multiple of l

- Then, we use it to reconstruct the entry points.

There are several algorithms to do that, but they all share the same principle. First, identify the length of the cycle (or a multiple of it) and then recover the entry points. We only consider Floyd's algorithm that find the cycle's length by computing the sequences s_n and s_{2n} in parallel.
Note that this can be done using only a small amount of memory. Indeed, s_n is computed just by applying f to the previous value and s_{2n} by applying f twice to $s_{2*(n-1)}$. We will refer to them as the slow and fast sequences.
Keeping in mind the picture from the previous slide, we see that the fast sequence rushes ahead and enter

the cycle much earlier than the slow one. Of course, once in the cycle, the sequence stays there and waits for the slow sequence to also enter the cycle. Once they are both in it, we can view the fast sequence as being behind the slow one and we see that at each moment in time it gets closer to it by one step. Thus, at some point, the fast sequence catches the slow one and we have $s_{2n} = s_n$. When this happen, n is necessarily a multiple of the cycle length.
In the next slide, we show how to use the length to get the collision itself.

- Given a multiple L of l, construct the entry point
  - Step 1: Compute s_L
  - Step 2: Compute in parallel s_i and s_{L+i} until $s_i = s_{L+i}$
  - Then, $f(s_{i-1}) = f(s_{L+i-1})$ is the desired entry point collision

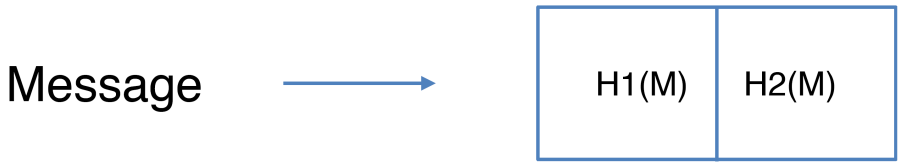
Once we have a multiple of the length of the cycle (call it L), we first compute S_L then maintain in parallel the two sequences s_i and s_{L+i}. At some point, precisely when i is the length of the tail, the two sequences collide and we discover a collision for f.
Analysing these cycle finding techniques isn't a trivial matter, however, the conclusion is they also have a running time of the order of the square of the size of the set S. As a consequence, whenever possible, this method should be preferred to collision finding with memory.

10.11.8 Generic pre-images, second pre-images and one wayness

- Basically, use exhaustive search.
- For Pre-images the expected time is the size of the output set
- For One-wayness it is the minimum of the size of output and reference sets

For pre-images or one-wayness, generic attack basically rely on exhaustive search. It simply consists of trying distinct messages until the desired target is reached. For pre-images, the expected time is the size of the output set. For one-wayness, assuming the initial message whose hash value is given belongs to a smaller set, it suffices to try all the messages in that smaller set to succeed.

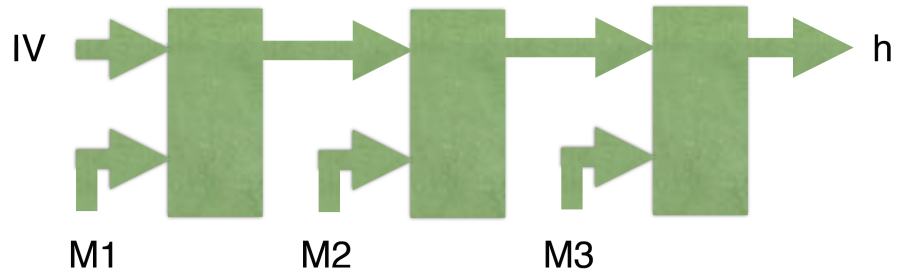
10.11.9 Parallel Hash extension



For random oracles, it is a double-sized hash

We now move into the realm of non generic attacks and show a construction that despite being generically secure can fail utterly in practice. This construction is the parallel hash extension method. Assume that we have two hash functions H_1 and H_2 on n bits each and look at the function that given a message M put together $H_1(M)$ and $H_2(M)$. It is easily shown that if H_1 and H_2 are random oracles on n bit each then this parallel composition (also called concatenation) is a random oracle on $2n$ bits. As a consequence, this technique was long consider to be a nice way to build large hash functions from smaller ones. Of course, it is clear that there are bad choices such as $H_1 = H_2$ and people guessed that having H_1 and H_2 too similar wouldn't be a good idea.
We are now going to show why this fails as soon as H_1 is a Merkle-Damgard hash function.

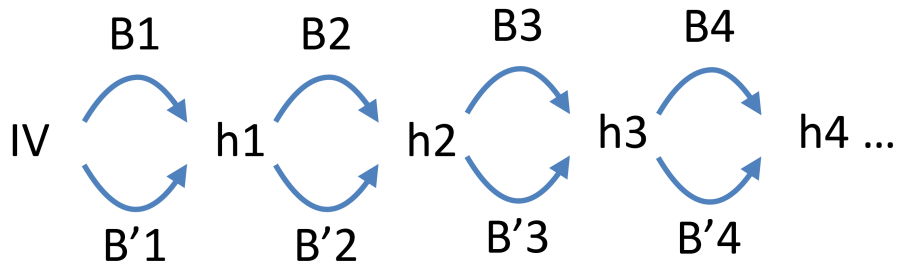
10.11.10 Merkle-Damgard Hash construction



Merkle (79) Damgard (89)

We briefly recall that a Merkle-Damgard hash is based on a compression function and computes the hash block by block starting from a fixed initial value and compressing at each round the current value and a message block into a new value. The final value is the hash result.
Note that the message is pre-formatted by using padding techniques to make its length a multiple of the block size.
For simplicity, we assume in the sequel that the blocks are of size at least comparable to the inner values.

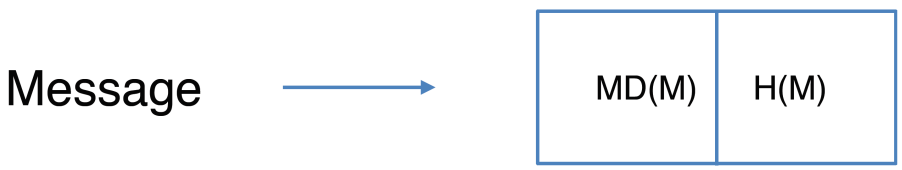
10.11.10.1 Multicollisions on Merkle-Damgard construction



Plenty of messages with the same hash !

Since the blocks are large enough and thanks to the birthday paradox, there should be two blocks B_1 and B_1' such that the first compression step collides, giving an internal value h_1. Similarly we can find two block B_2 and B_2' such that the compression of h_1 and the block collides giving h_2.
If we continue like this, we can create plenty of distinct messages with the same hash. On our example with four blocks, we have 16 distinct messages built by taking an arbitrary mix of the four B/B' blocks. Working with t blocks instead of 4, we would obtain a set of 2^t colliding messages for t times the cost of one collision.
Such structures are called multi collisions on the hash function and they are extremely hard to construct on random oracles.

10.11.10.2 Parallel Hash extension with a Merkle-Damgard hash



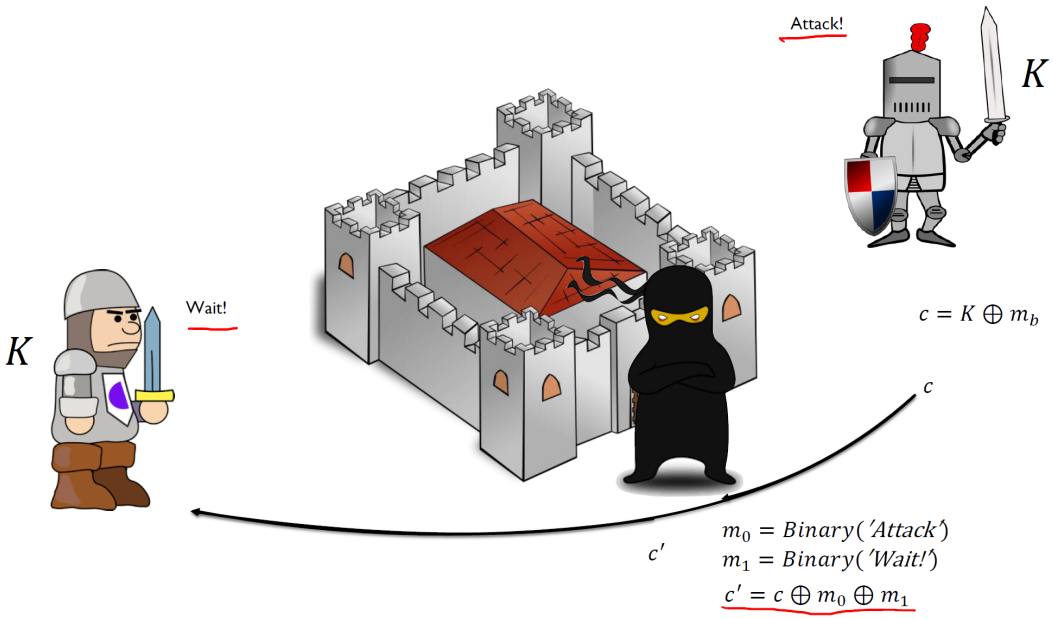
For the MD part, construct a multicollision with $2^{\frac{n}{2}}$ messages. Then, by birthday paradox, find a collision on H. Global cost much closer to $2^{\frac{n}{2}}$ than to 2^n.

We now come back to the parallel extension where H_1 is a Merkle-Damgard hash function. We first construct a multi collision of size $2^{\frac{n}{2}}$ on this hash. Among these $2^{\frac{n}{2}}$ messages which all collide on the first hash, we expect (thanks to the birthday paradox) to find a pair that also collides on the second hash function.
As a consequence, we create a collision on the $2n$-bit concatenated hash for a cost of the order of n times $2^{\frac{n}{2}}$. Thus the construction is only marginally more secure than a single n-bit hash function.
Multicollisions have plenty of other cryptanalytic applications. To avoid them many recent hash constructions avoid the Merkle-Damgard method in favour of other techniques. For example the sponge technique to just name one.

Before concluding the lecture, let us also mention that differential cryptanalysis can also be used to construct collisions on hash functions. In particular, it was successful in breaking older hash functions such as MD5, SHA0 or SHA1.

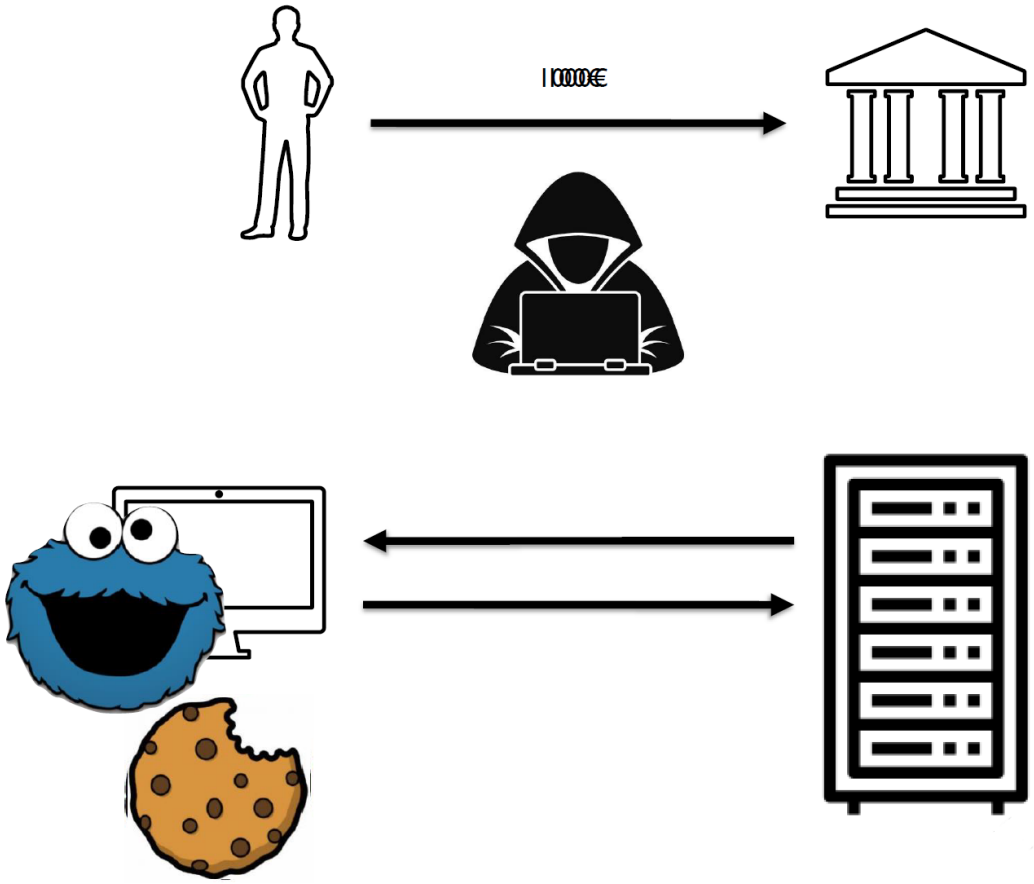
Authentication

11.1 An Attack that doesn't break Secrecy



11.2 Message Integrity

- Need a mechanism to ensure that messages are authentic!
- Authentic messages should not be *tamperable*
- More examples:
  - Bank Transaction
  - Cookies
- Idea: you need to be in possession of a secret to be able to authenticate messages



11.3 Message Authentication Codes

Syntax: An message authentication code consists of three PPT algorithms: $(Gen, Mac, Verify)$

- $Gen(1^\lambda)$: A randomized algorithm which takes as input the security parameter 1^λ (encoded in unary) and outputs a key K
- $Mac(K, m)$: A (possibly randomized) algorithm which takes a key K and message m as input and outputs authentication tag t
- $Verify(K, m, t)$: A deterministic algorithm which takes as a key K, a message m and a tag t as input and outputs a bit b

Correctness: It holds for all $\lambda \in \mathbb{N}$ and all messages m that $Pr[Verify(K, m, Mac(K, m)) = 1] = Pr[Verify(K, m, t) = 1] = 1$, where $K \leftarrow Gen(1^\lambda)$

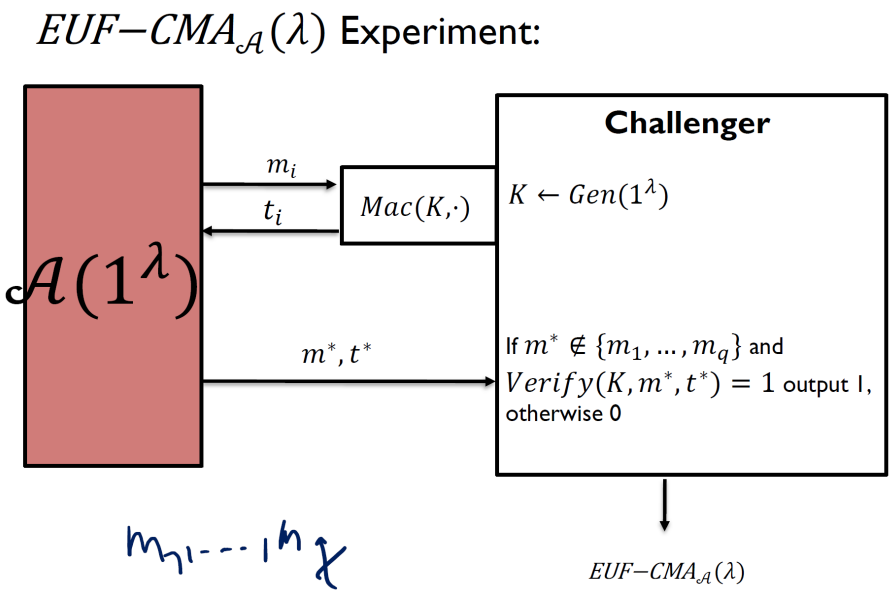
11.4 Security of Message Authentication Codes

- How can we formalize security of MACs?
- What resources are given to the adversary?
  - Real world adversary sees authenticated messages

  $\Rightarrow$ Let adversary choose authenticated messages (as in CPA security)
- What is the goal of the adversary?
  - Real world: Authenticate a new meaningful message
  - Better: Authenticate any new message

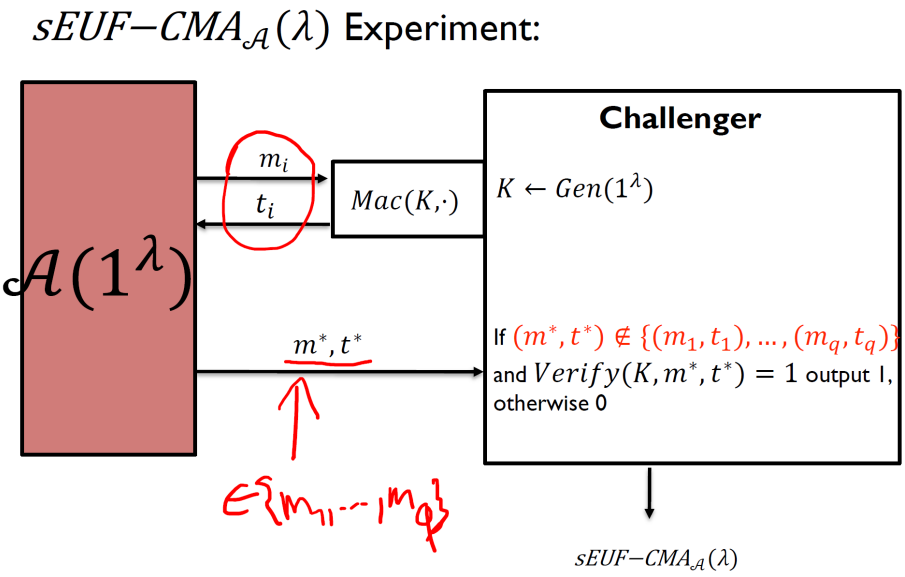
Definition 21 ($EUF-CMA$-secure)**.** *A message authentication code* $(Gen, Mac, Verify)$ *is existentially unforgeable under adaptive chosen message attacks, or* $EUF-CMA$-***secure****, if it holds for* ***every PPT-bounded adversary*** $\mathcal{A}$ *there exists a negligible function* v *s.t. for all* $\lambda \in \mathcal{N}$

$$Pr[EUF-CMA_{\mathcal{A}}(\lambda) = 1] < v(\lambda)$$

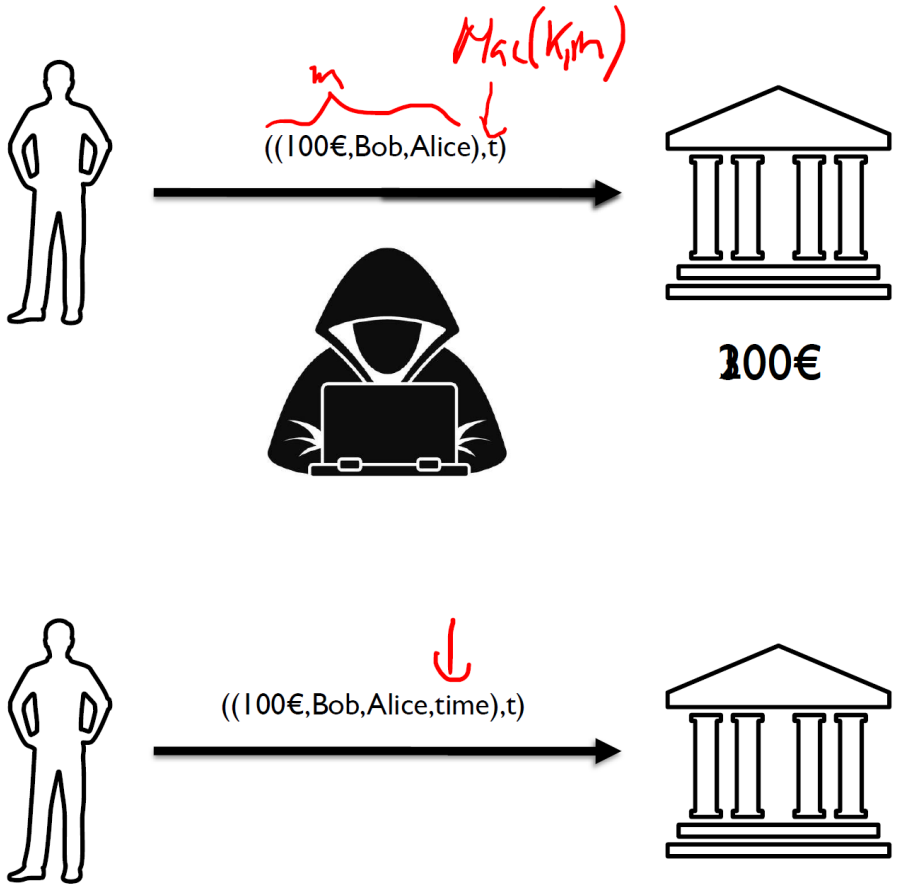


Definition 22 ($sEUF-CMA$-secure)**.** *A message authentication code* $(Gen, Mac, Verify)$ *is* ***strongly*** *existentially unforgeable under adaptive chosen message attacks, or* ***strongly*** $EUF-CMA$-***secure****, if it holds for* ***every PPT-bounded adversary*** $\mathcal{A}$ *there exists a negligible function* v *s.t. for all* $\lambda \in \mathcal{N}$

$$Pr[sEUF-CMA_{\mathcal{A}}(\lambda) = 1] < v(\lambda)$$



11.5 Using Message Authentication



- Message Authentication codes do not prevent replay attacks
- Need additional measures, e.g. timestamp

11.6 Summary 1

- Message secrecy and message integrity are fundamentally different goals
- Encryption does not guarantee integrity
- Message Authentication Codes protect integrity of messages
- They don't protect against replay attacks!

11.7 Constructing Message Authentication Codes

- How should we construct MACs?
- Consider the constraints: Should not be possible to find an authentication tag for a new message
- What tools/primitives have we seen so far?
- We need a function which is unpredictable on new inputs
- ...the workhorse of symmetric key cryptography...
- ...pseudorandom functions!

11.8 Message Authentication Codes for fixed length messages

Assume for now we want to authenticate messages of fixed bit length n **Construction 6.1:** Let $PRF : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^\lambda$ be a pseudorandom function

- $Gen(1^\lambda)$: Choose key $K \leftarrow_\$ \{0,1\}^\lambda$ for PRF
- $Mac(K,m)$: Compute $t \leftarrow PRF(K,m)$
- $Verify(K,m,t)$: If $t = PRF(K,m)$ output 1, otherwise 0

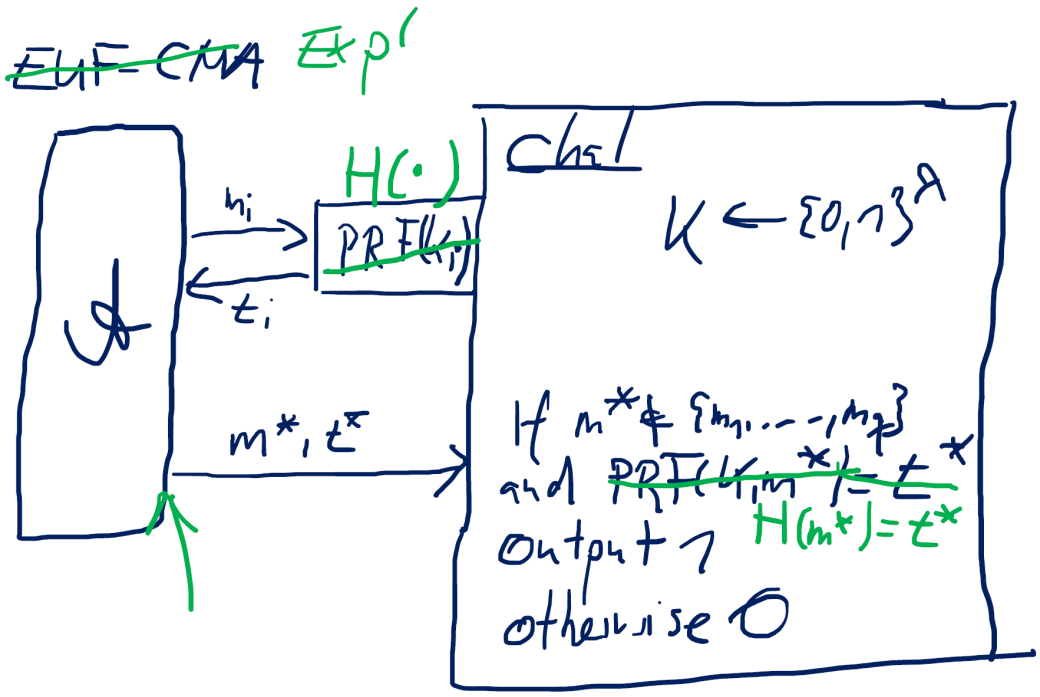
Correctness: Canonic

11.8.1 Security

Theorem 8. *If PRF is a pseudorandom function, then Construction 6.1 is a $sEUF-CMA$-secure MAC*

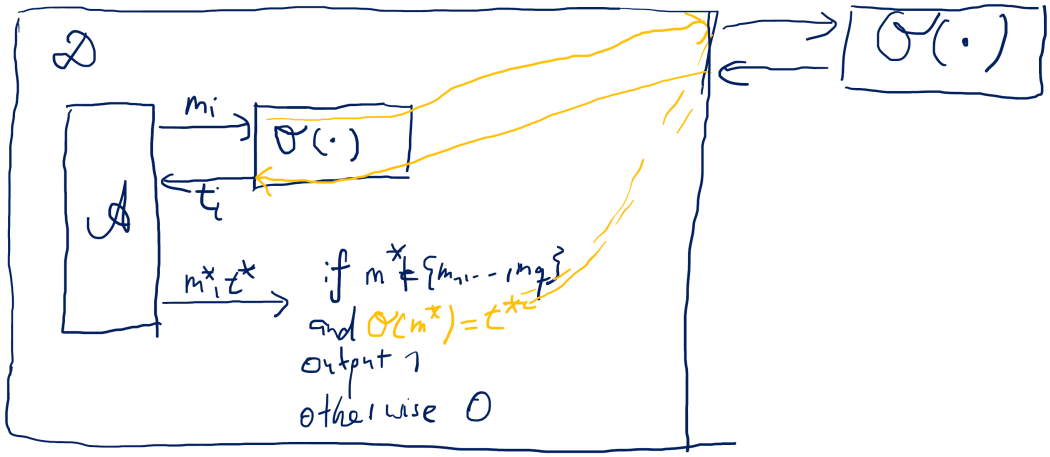
Proof. $MAC = (Gen, Mac, Verify)$ is not $EUF-CMA$-secure $\Rightarrow$ PRF is not pseudorandom
Let $\mathcal{A}$ be a PPT adversary and ϵ a non-negligible function so that

$$Pr[EUF-CMA_{\mathcal{A}}(\lambda) = 1] > \epsilon$$



Assume $m^* \notin \{m_1, ..., m_q\}$:

- H is a truly random function
- $H(m^*)$ is uniformly random
- $Pr[H(m^*) = t^*] = 2^{-\lambda} (= \frac{1}{|\{0,1\}^\lambda|})$
- $Pr[Exp' = 1] 2^{-\lambda} \Rightarrow$ negligible!



Case 1: $\mathcal{O}(\cdot) = PRF(K, \cdot)$
$\Rightarrow \mathcal{D}$ perfectly simulates $EUF - CMA_{\mathcal{A}}(\lambda)$
$\Rightarrow Pr[\mathcal{D}^{PRF(K,\cdot)}(1^\lambda) = 1] = Pr[EUF - CMA_{\mathcal{A}}(\lambda) = 1] > \epsilon$
Case 2: $\mathcal{O}(\cdot) = H(\cdot)$
$\Rightarrow \mathcal{D}$ perfectly simulates Exp'
$\Rightarrow Pr[\mathcal{D}^{H(\cdot)}(1^\lambda) = 1] = Pr[Exp'_{\mathcal{A}}(\lambda) = 1] = 2^{-\lambda}$

$$|Pr[\mathcal{D}^{PRF(K,\cdot)}(1^\lambda) = 1] - Pr[\mathcal{D}^{H(\cdot)}(1^\lambda) = 1]| > \epsilon - 2^{-\lambda}$$

$\Rightarrow$ non-negligible! □

11.9 Message Authentication Codes for long Messages

Construction 6.2: Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function and $MAC = (Gen, Mac, Verify)$ be a message authentication code for messages of length λ.

- $Gen'(1^\lambda)$: Generate $K' \leftarrow Gen(1^\lambda)$ and $s \leftarrow_\$ \{0,1\}^\lambda$. Output $K = (s, K')$.
- $Mac'(K, m)$: Parse $K = (s, K')$. Compute and output $t \leftarrow MAC(K', H^s(m))$.
- $Verify'(K, m, t)$: If $t = MAC(K', H^s(m))$ output 1, otherwise 0.

Correctness: Canonic

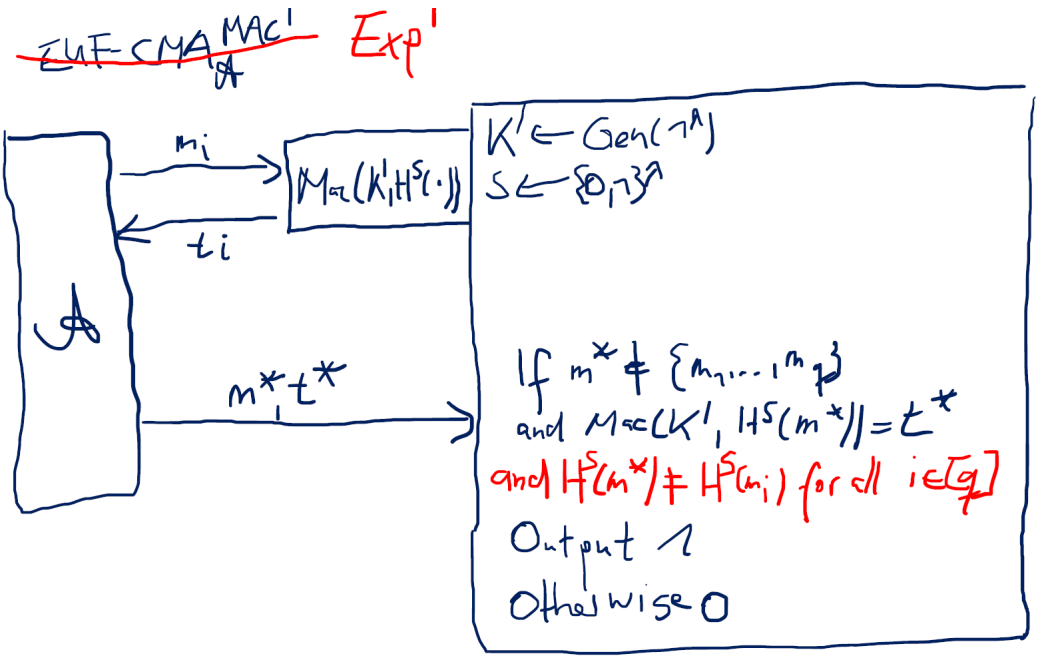
11.9.1 Security

Theorem 9. *If H is a collision resistant hash function and $MAC = (Gen, Mac, Verify)$ as $sEUF - CMA$-secure MAC, then $MAC' = (Gen', Mac', Verify')$ is a $sEUF - CMA$-secure MAC.*

Proof. **To show:** Assume MAC' is not $EUF - CMA \Rightarrow$ Either H is not collision resistant or MAC is not $EUF - CMA$-secure
Let $\mathcal{A}$ be a PPT adversary and ϵ a non-negligible function so that

$$Pr[EUF - CMA_{\mathcal{A}}^{MAC'}(\lambda) = 1] > \epsilon$$

Set $Exp = EUF - CMA_{\mathcal{A}}^{MAC'}$



Claim:

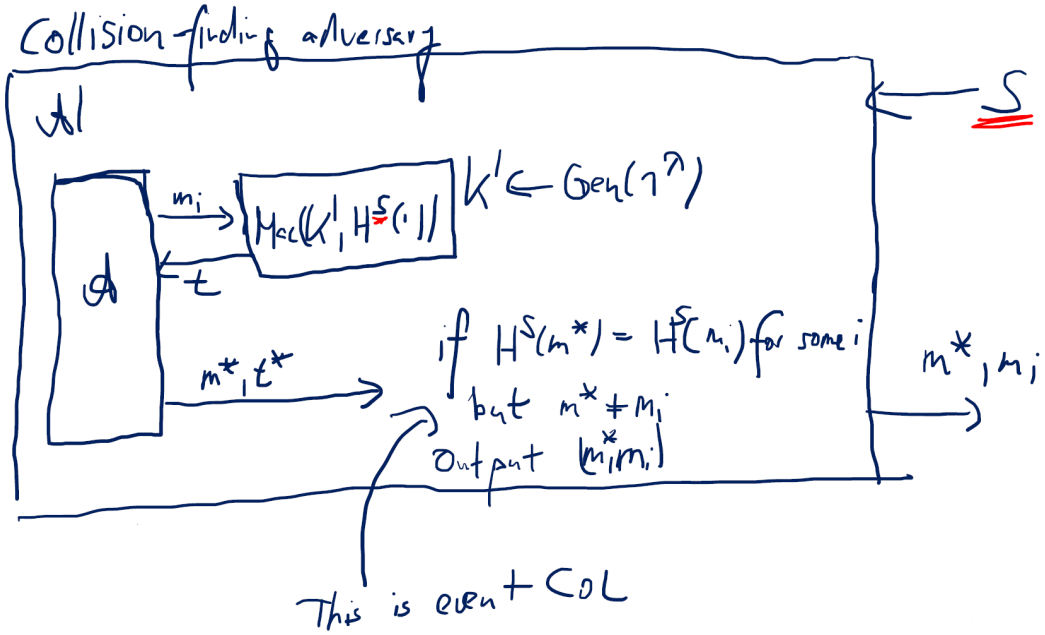
$$|Pr[Exp = 1] - Pr[Exp' = 1]| < negl.$$

Otherwise it is not collision resistant
Proof via contraposition:
Assume $|Pr[Exp = 1] - Pr[Exp' = 1]| > \epsilon'$ where ϵ' is non-negligible.
Observation: If $H^s(m^*)$ does not collide with any of the $H^s(m_i)$ then the two experiments are identical.
Let Col be the event that $H^s(m^*) = H^s(m_i)$ for some $i \in [q]$ $m^* \neg m_i$
With the LOTP it follows that

1. $Pr[Exp = 1] = Pr[Exp = 1 \mid Col] \cdot Pr[Col] + Pr[Exp = 1 \mid \bar{Col}] \cdot Pr[\bar{Col}]$
2. $Pr[Exp' = 1] = Pr[Exp' = 1 \mid Col] \cdot Pr[Col] + Pr[Exp' = 1 \mid \bar{Col}] \cdot Pr[\bar{Col}]$

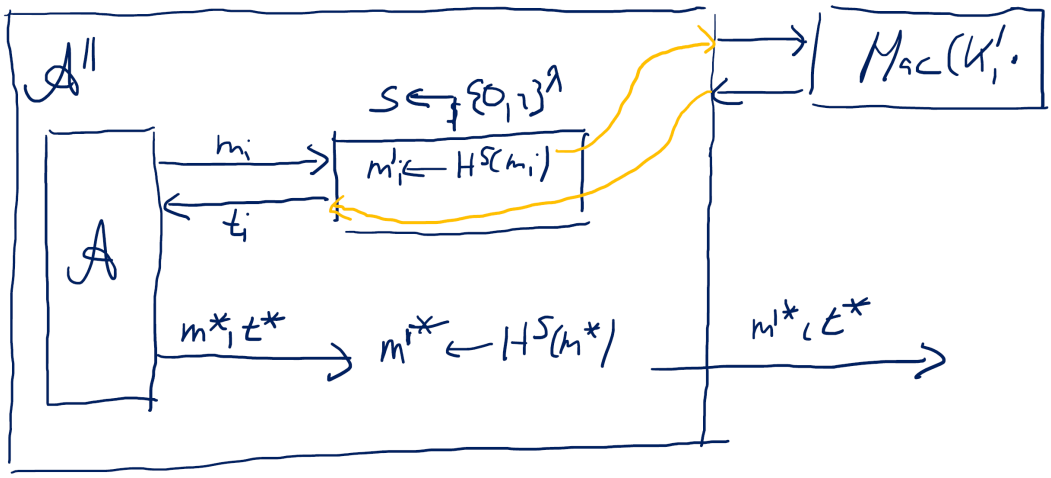
With $Pr[Exp' = 1 \mid Col]$ and $Pr[Exp = 1 \mid Col] < 1$ we can consider

$$\epsilon < Pr[Exp = 1] - Pr[Exp' = 1] = Pr[Exp = 1 \mid Col] \cdot Pr[Col] < Pr[Col]$$



$$Pr[Hash - Col_{\mathcal{A}}(\lambda) = 1] = Pr[Col] > \epsilon'$$

ϵ' is non-negligible. (Proves the Claim)
Now assume $Pr[Exp = 1] - Pr[Exp' = 1] < negl.$ with $Pr[Exp = 1] > \epsilon$
$\Rightarrow Pr[Exp' = 1] > \epsilon - negl. = \epsilon'$ which is non-negligible.
We will construct a PPT adversary $\mathcal{A}''$ against MAC



From the view of $\mathcal{A}$, $\mathcal{A}''$ simulates Exp' perfectly

$$Pr[EUF - CMA^{MAC}_{\mathcal{A}''}(\lambda) = 1] = Pr[Exp'_{\mathcal{A}} = 1] > \epsilon''$$

which is non-negligible with $H^s(m^*) \neg H^s(m_i)$
$\Rightarrow m'^*$, t^* is valid forge for MAC □

11.10 Message Authentication Codes from Random Oracles

Construction 6.3: Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function (modelled as a random oracle).

- $Gen(1^\lambda)$: Generate $K \leftarrow_\$ Gen(1^\lambda)$. Output K.
- $Mac(K, m)$: Compute and output $t \leftarrow H(K||m)$.
- $Verify(K, m, t)$: If $t = H(K||m)$ output 1, otherwise 0.

Correctness: Canonic
Security: $sEUF - CMA$ secure, proof essentially as in Theorem 8.

- However: Problematic when used with many practical hash functions!
- Merkle Damgård: $H(m_1||m_2||m_3) = h(h(m_1||m_2)||m_3)$
- $Mac(K, (m_1, m_2)) = H(K||m_1||m_2) = h(h(K||m_1)||m_2)$

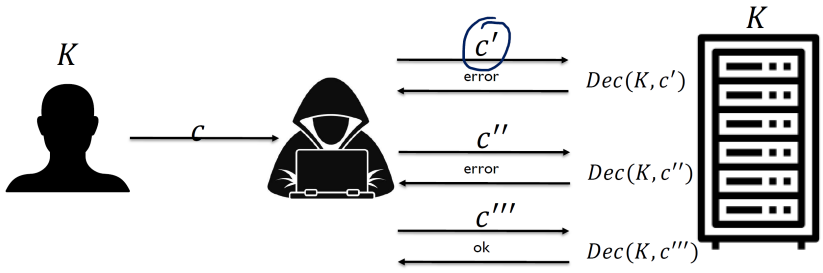
11.11 Summary 2

- MACs for fixed length messages can be constructed from PRFs
- Hash functions can be used to MAC messages of arbitrary length
- Random Oracles yield a simple MAC construction
- Be careful when a random oracle is replaced with a real hash function (Merkle Damgård)

Chosen Ciphertext Security

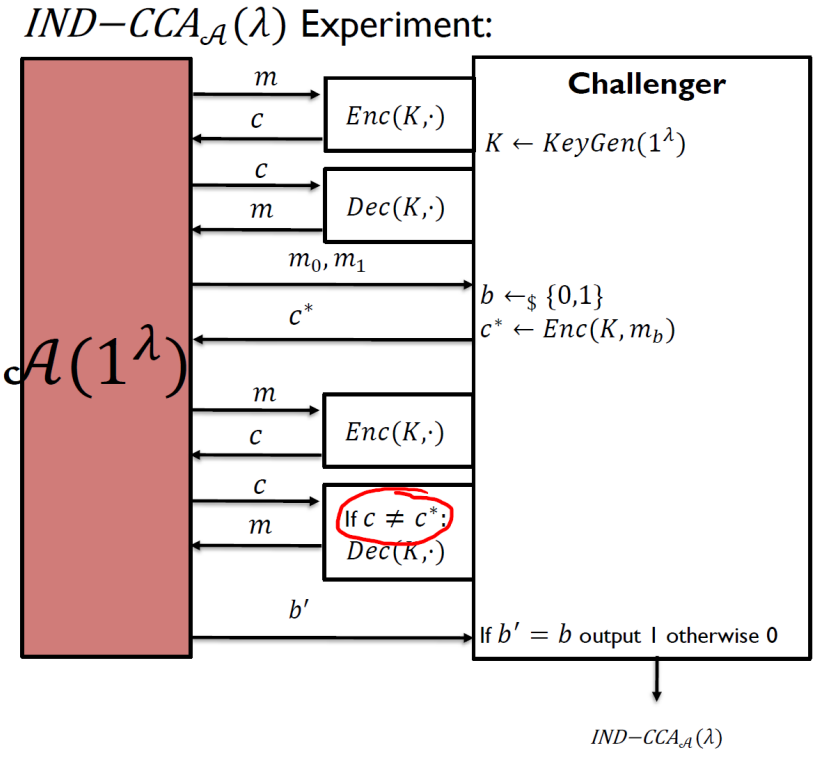
12.1 Encryption and Authentication

- ...are different goals...
- ...not always!



Definition 23 ($IND-CCA$-secure)**.** *An encryption scheme* $(KeyGen, Enc, Dec)$ *is indistinguishable under chosen ciphertext attacks, or* $IND-CCA$***-secure****, if it holds for* ***every PPT-bounded adversary*** $\mathcal{A}$ *there exists a negligible function* v *s.t. for all* $\lambda \in \mathbb{N}$

$$Pr[IND-CCA_{\mathcal{A}}(\lambda) = 1] < \frac{1}{2} + v(\lambda)$$



12.2 Constructing $IND-CCA$ secure encryption

- Idea: Use authentication
- But how?
- 3 Options:
  1. Authenticate and Encrypt: $Enc(K_1, m), Mac(K_2, m)$
  2. Authenticate then Encrypt: $Enc(K_1, (m||Mac(K_2, m)))$
  3. Encrypt then Authenticate: $Enc(K_1, m), Mac(K_2, Enc(K_1, m))$
- Evaluation of the 3 options:
  1. Authenticate and Encrypt: $Enc(K_E, m), Mac(K_M, m)$
     $\Rightarrow$ Is bad: $Mac(K_M, m) = (Mac'(K_M, m), m)$
  2. Authenticate then Encrypt: $Enc(K_E, (m||Mac(K_M, m)))$
     $\Rightarrow$ Also bad: $Enc(K_E, m) = (Enc'(K_E, m), r)$
  3. Encrypt then Authenticate: $Enc(K_E, m), Mac(K_M, Enc(K_E, m))$
     $\Rightarrow$ Winner!

Construction 6.4: Let $(KeyGen, Enc, Dec)$ be an encryption scheme and $(Gen, Mac, Verify)$ a MAC.

- $KeyGen'(1^\lambda)$: Compute $K_E \leftarrow KeyGen(1^\lambda)$ and $K_M \leftarrow Gen(1^\lambda)$ and output $K \leftarrow (K_E, K_M)$.
- $Enc'(K, m)$: Parse $K = (K_E, K_M)$. Compute $c \leftarrow Enc(K_E, m)$ and $t \leftarrow Mac(K_M, c)$. Output $c' \leftarrow (c, t)$.
- $Dec'(K, c')$: Parse $K = (K_E, K_M)$ and $c' = (c, t)$.
  If $Verify(K_M, c, t) = 1$ output $m \leftarrow Dec(K_E, c)$, otherwise $\perp$

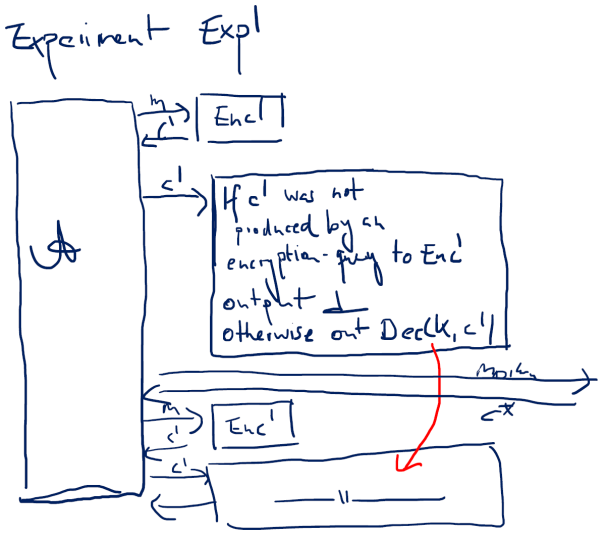
Correctness: Follows from correctness of $(KeyGen, Enc, Dec)$ and $(Gen, Mac, Verify)$.

Theorem 10. *If $(KeyGen, Enc, Dec)$ is an $IND-CPA$ secure encryption scheme and $MAC = (Gen, Mac, Verify)$ is a $sEUF-CMA$ secure message authentication code, then $(KeyGen', Enc', Dec')$ is an $IND-CCA$ secure encryption scheme.*

Proof. Assume $(KeyGen', Enc', Dec')$ is not an $IND-CCA$ secure encryption scheme!
$\Rightarrow$ $(KeyGen, Enc, Dec)$ is not $IND-CPA$ secure or $(Gen, Mac, Verify)$ is not $sEUF-CMA$ secure.
Thus, assume there is a PPT-adversary $\mathcal{A}$ and a non-negligible ϵ so that

$$Pr[IND-CCA_{\mathcal{A}}(\lambda) = 1] > \frac{1}{2} + \epsilon$$

We call Exp the $IND-CCA_{\mathcal{A}}(\lambda)$ experiment.



Claim:

$$|Pr[Exp = 1] - Pr[Exp' = 1]| > \epsilon'$$

for a non-negligible ϵ'
$\Rightarrow (Gen, Mac, Verify)$ is not $sEUF - CMA$ secure.

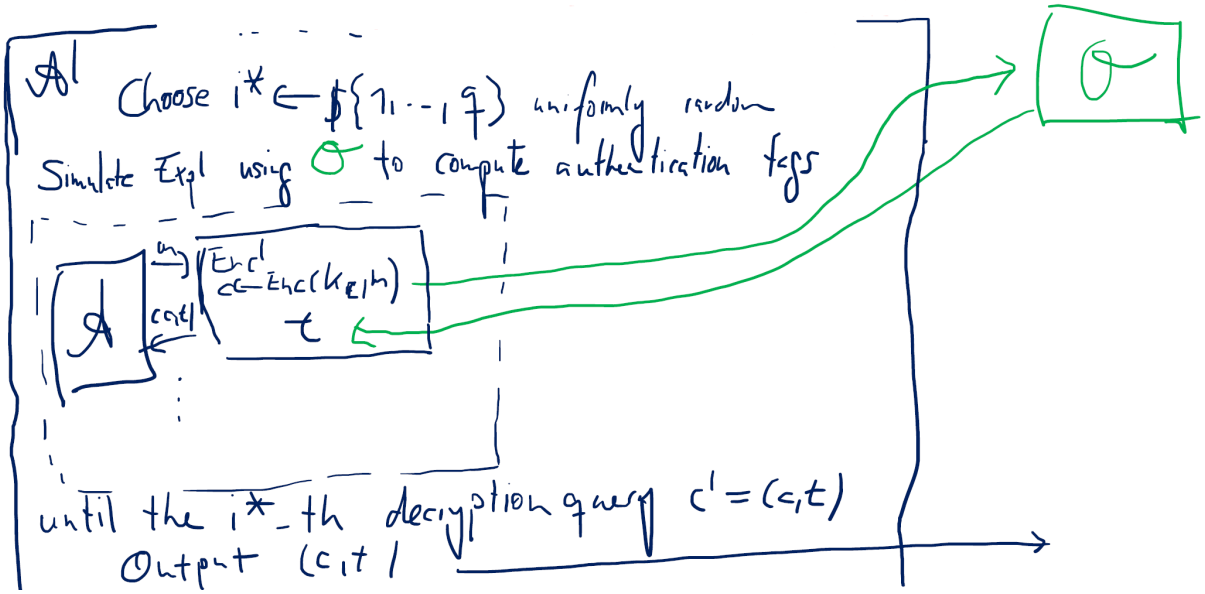
Proof of Claim: Let $ValidQuery$ (short: VQ) be the event that $\mathcal{A}$ sends a decryption query $c' = (c, t)$ with $Verify(K, m, c, t) = 1$ but c' was not obtained by an encryption query.
Conditioned on $\overline{ValidQuery}$ (short: $\overline{VQ}$) $\Rightarrow Exp$ and Exp' behave identically.
So it follows with LOTP that

1. $Pr[Exp = 1] = Pr[Exp = 1 \mid VQ] \cdot Pr[VQ] + Pr[Exp = 1 \mid \overline{VQ}] \cdot Pr[\overline{VQ}]$
2. $Pr[Exp' = 1] = Pr[Exp' = 1 \mid VQ] \cdot Pr[VQ] + Pr[Exp' = 1 \mid \overline{VQ}] \cdot Pr[\overline{VQ}]$

With $Pr[Exp = 1 \mid \overline{VQ}] \cdot Pr[\overline{VQ}] = Pr[Exp' = 1 \mid \overline{VQ}] \cdot Pr[\overline{VQ}]$ it follows that

$$\epsilon' < |Pr[Exp = 1] - Pr[Exp' = 1]| = |Pr[Exp = 1 \mid VQ] - Pr[Exp' = 1 \mid VQ]| \cdot Pr[VQ] \leq Pr[VQ],$$

because $|Pr[Exp = 1 \mid VQ] - Pr[Exp' = 1 \mid VQ]| \leq 1$.
Construct an adversary $\mathcal{A}'$ against $sEUF - CMA$ security of $(Gen, Mac, Verify)$ with non-negligible success probability.
$\mathcal{A}$ makes at most q decryption queries.



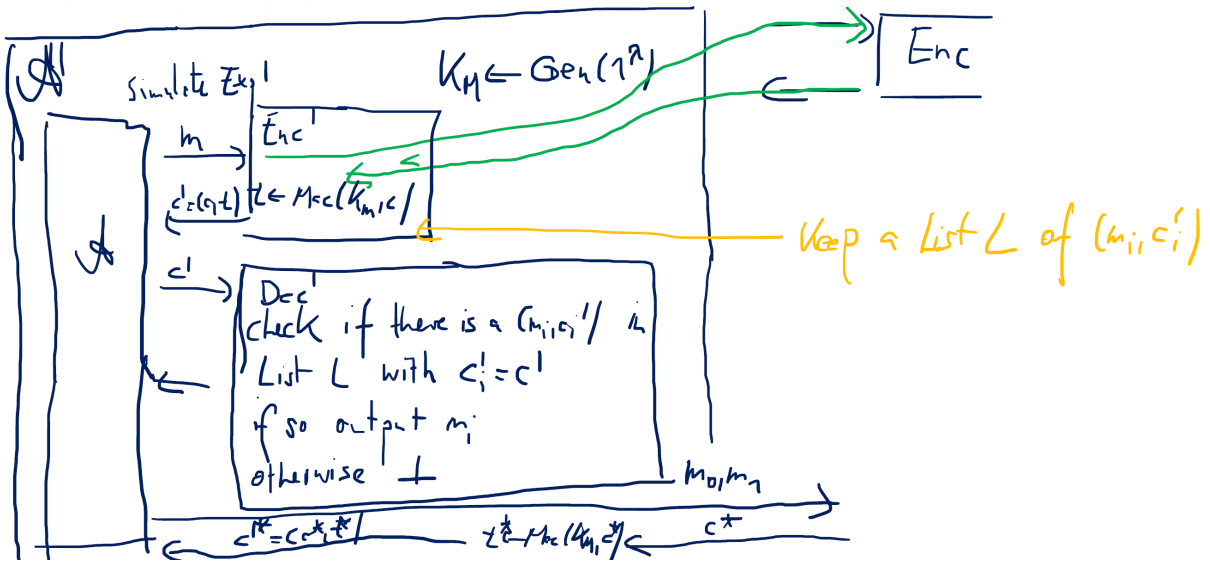
- From view of $\mathcal{A}$, Exp' and $\mathcal{A}'s$ simulation are identical.
- If $ValidQuery$ occurs, then there is an index $i \in \{1, ..., q\}$ s.t. the i-th decryption query $c' = (c, t)$ was not produced by the encryption oracle and $Verify(K_M, c, t) = 1$.
- Since i^* is chosen uniformly random from $\{1, ..., q\}$ it holds $Pr[i^* = i] = \frac{1}{q}$.
- If $ValidQuery$ happens and $i^* = i$ then $\mathcal{A}'$ outputs a new forge for the message c.

$\Rightarrow Pr[sEUF - CMA_{\mathcal{A}'}(\lambda) = 1] \geq Pr[i^* = i \text{ and } VQ] = Pr[i^* = i] \cdot Pr[VQ] \geq \frac{1}{q} \cdot \epsilon' = \frac{\epsilon'}{q}$ non-negligible.
But this contradicts the $sEUF - CMA$ security of MAC! Thus

$$|Pr[Exp = 1] - Pr[Exp' = 1]| < negl \Rightarrow Pr[Exp' = 1] > \frac{1}{2} + \epsilon - negl$$

with $\epsilon'' = \epsilon - negl$ which is non-negligible!

Construct a PPT adversary $\mathcal{A}''$ against $IND-CPA$ security of $(KeyGen, Enc, Dec)$ with advantage ϵ''



From view of $\mathcal{A}$, Exp' and simulation by $\mathcal{A}'$ are identically distributed

-> Enc' oracles behave identically.

-> Dec' oracles behave identically given that $(KeyGen, Enc, Dec)$ is correct.

$\Rightarrow Pr[IND-CPA_{\mathcal{A}''}(\lambda)=1] = Pr[Exp'=1] > \frac{1}{2}+\epsilon''$
$\Rightarrow$ This contradicts $IND-CPA$ security of $(KeyGen, Enc, Dec)$! □

12.3 Summary

- Active adversaries try to break secrecy by manipulating messages and observing behavior other parties
- $IND-CCA$ security provides security against active adversaries
- Modeled by giving the adversary access to a decryption oracle
- $IND-CCA$ encryption from $IND-CPA$ encryption and MAC via encrypt-then-mac!

Part III

Public Key Encryption

Algebra

13.1 Basics of Algebra

13.1.1 Ring of integers $\mathbb{Z}$

A ring $(R, +, \cdot)$ is a set R equipped with two binary operations $+$ and $\cdot$. A ring is satisfying the following three axioms (ring axioms):

1. R is an **abelian group** under addition $(+)$, meaning that for all $a, b, c \in R$:
   - **Associative** under addition: $(a + b) + c = a + (b + c)$
   - **Commutative** under addition: $a + b = b + a$
   - Additive **identity**: There is an element $0 \in R$ such that $a + 0 = a$
   - Additive **inverse**: For each a there exists an $-a \in R$ such that $a + (-a) = 0$
2. R is an **monoid** under multiplication $(\cdot)$, meaning that for all $a, b, c \in R$:
   - **Associative** under multiplication: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
   - Multiplicative **identity**: There is an element $1 \in R$ such that $a \cdot 1 = 1 \cdot a$
3. Multiplication is distributive with respect to addition, meaning that for all $a, b, c \in R$:
   - **Left distributivity**: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
   - **Right distributivity**: $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$

13.1.1.1 Divisibility

Division operations are essential for the work in modern cryptography. We define a *divides* b, written $a|b$ as follow:

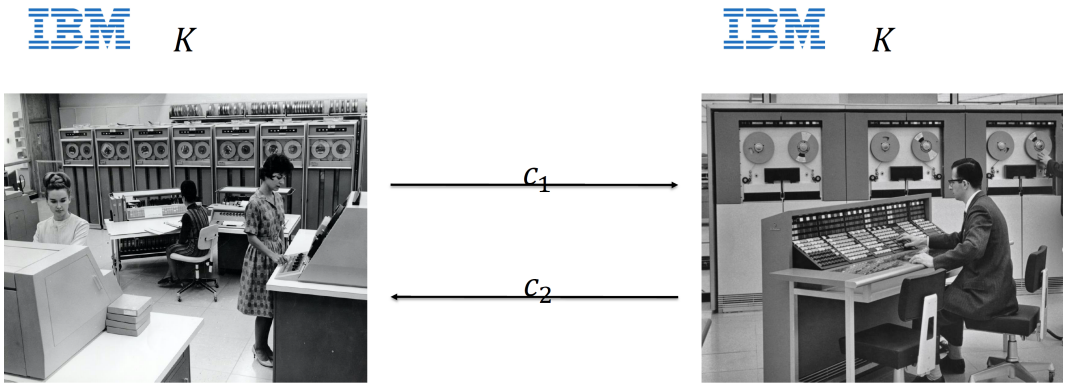
$$a|b :\Leftrightarrow \exists \lambda \in \mathbb{Z} \text{ so that } b = \lambda \cdot a$$

If a does not divide b, we write $a \nmid b$.

13.1.1.2 Euclidean division

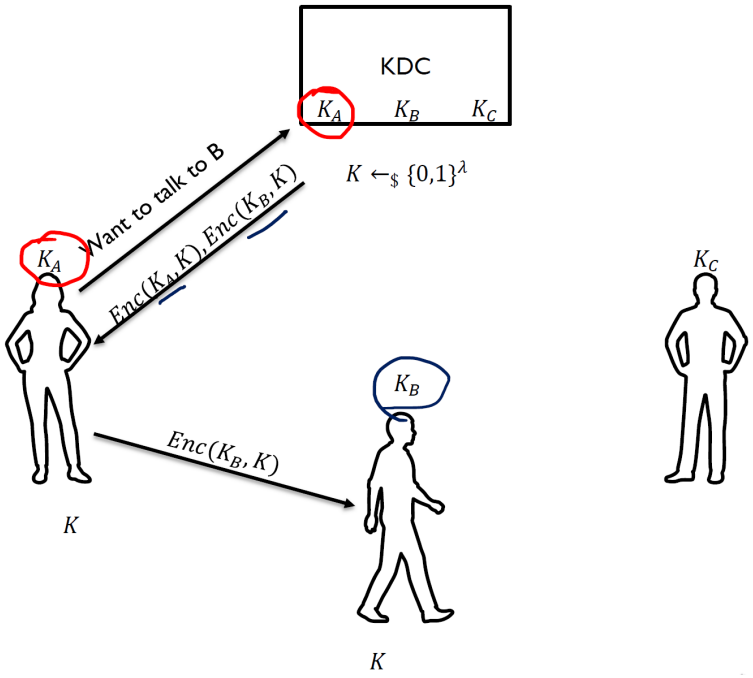
Key Distribution and Key Exchange

14.1 Private Key Cryptography



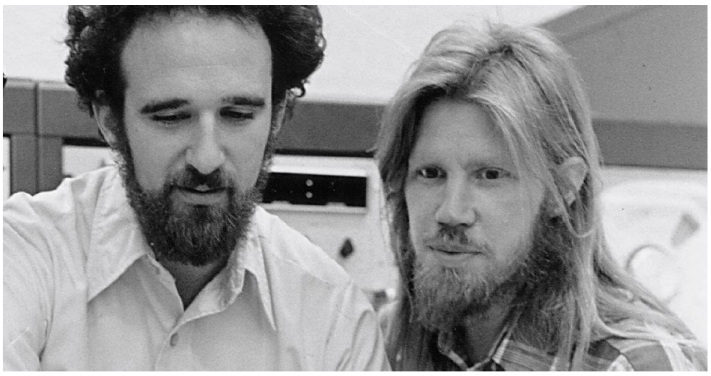
14.2 A modest Proposal: Key Distribution Centers

- Central Entity the manages and distributes keys
- Each user has a joint key with key distribution center (KDC)
- All keys can be computed pseudorandomly, KDC then only needs to store PRF key
- A establishes secure channel to B by requesting session key from KDC
- Major problem: KDC is single point of failure
- Compromising KDC compromises all other channels in the network



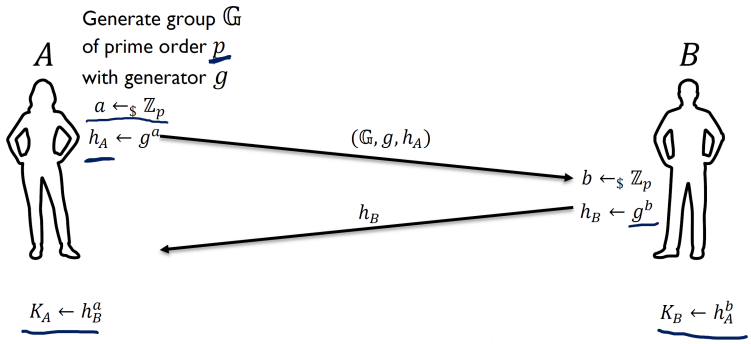
14.3 New Directions in Cryptography

- KDCs don't solve key distribution problemin open systems
- Observation of Diffie and Hellman: Many physical processes are asymmetric
- E.g. Padlock, shattering glass
- "Easy" in one direction, "hard" in the other
- Idea: exploit a computational phenomenon with this property to remotely establish a key



14.4 Diffie-Hellman Key Exchange

- Recall: Mechanism to generate cryptographic group $\mathcal{G}$ of order p with generator g
- $(\mathcal{G}, \cdot)$



$$K_A = h_B^a = (g^b)^a = g^{ab} = (g^a)^b = h_A^b = K_B$$

14.5 Formal Definitions for Key-Exchange Protocols

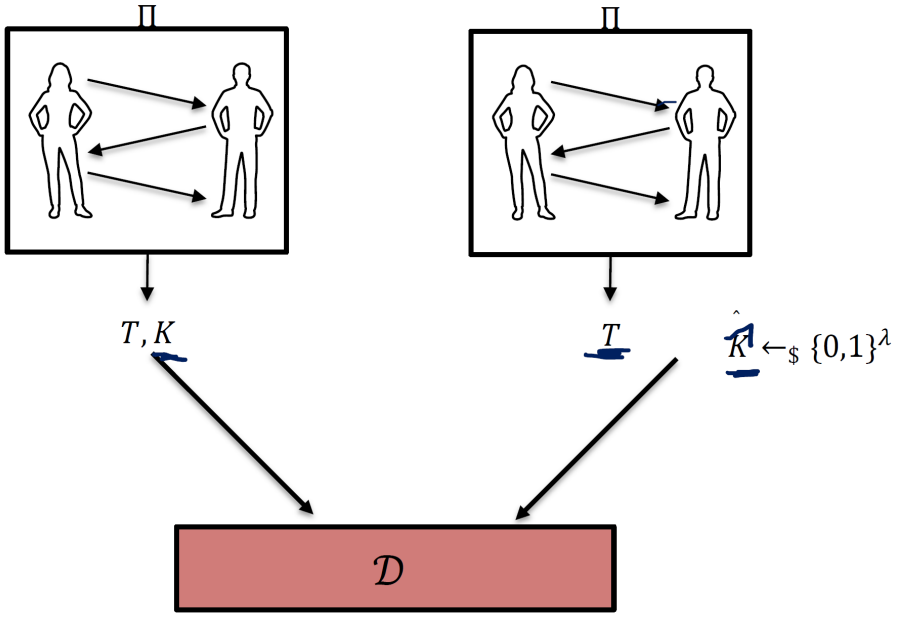
- Key exchange protocol Π consists of algorithms/instructions for Alice and Bob which output private state and output-message
- Let T denote the public transcript of the protocol
- Write $(T, K) \leftarrow \Pi(1^\lambda)$ to denote a run of the protocol Π (with uniformly random coins) which results in a transcript T and a key K
- E.g. for Diffie-Hellman key-exchange: $\Pi = (Alice_1, Bob, Alice_2)$
  - $Alice_1(1^\lambda) \to ((\mathbb{G}, a), (\mathbb{G}, g, h_a))$ with $h_A = g^a$
  - $Bob(\mathbb{G}, g, h_A) \to (K_B, K_B)$
  - $Alice_2((\mathbb{G}, a), h_B) \to K_A$
  - $T = (\mathbb{G}, g, h_A, h_B)$
- **Correctness:** $Pr[K_A = K_B] = 1$

14.6 Security Definition for Key Exchange

- Let Π be a key-exchange protocol.
- $(T, K) \leftarrow \Pi(1^\lambda)$
- Let $\hat{K} \leftarrow_\$ \{0,1\}^\lambda$ be chosen uniformly random
- We say that Π is *secure against eavesdropping*, if ith holds for every PPT distinguisher $\mathcal{D}$ that

$$|Pr[\mathcal{D}(T, K) = 1] - Pr[\mathcal{D}(T, \hat{K} = 1]| \leq negl(\lambda)$$

- Probability taken over the choice of T, K and $\hat{K}$



14.7 The Decisional Diffie-Hellman (DDH) Assumption

- DDH Assumption: The Difiie-Hellman Key-Exchange protocol is secure against eavesdropping
- $a, b, r \leftarrow_{\$} \mathbb{Z}_p$

$$(\mathbb{G}, g, g^a, g^b, g^{ab}) \approx (\mathbb{G}, g, g^a, g^b, g^r)$$

We assume the group $\mathbb{G}$ is fixed, so

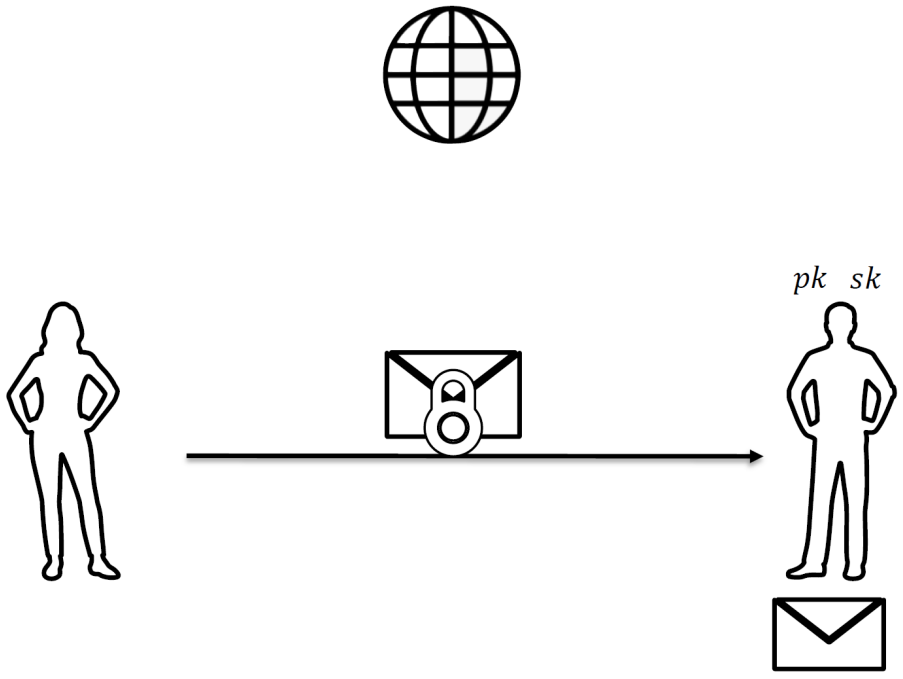
$$(g, g^a, g^b, g^{ab}) \approx (g, g^a, g^b, g^r)$$

14.8 Summary

- Private key encryption is useful in closed systems with pre shared keys
- In open systems we need a key distribution mechanism
- KDCs offer a partial solution
- Key exchange protocols offer a scalable solution

Public Key Encryption

- Key Exchange requires interaction
- To send mail we don t want to interact with the receiver
- In private key encryption same key K is used to encrypt and decrypt
- In public key encryption, we have a dedicated public key pk which is used to encrypt
- As the name suggests, this key is made public
- Anyone can encrypt using pk
- A secret key sk to decrypt



15.1 Encryption Schemes

Syntax: A public key encryption scheme consists of three PPT algorithms: $(KeyGen, Enc, Dec)$

- $KeyGen(1^\lambda)$: A randomized algorithm which takes as input the security parameter 1^λ (encoded in unary) and outputs a pair of **public** and **secret** keys (pk, sk)
- $Enc(pk, m)$: A randomized algorithm which takes a public key pk and message m as input and outputs a ciphertext c

- $Dec(sk, c)$: A deterministic algorithm which takes as a secret key sk and a ciphertext c as input and outputs a message m

Correctness: It holds for all $\lambda \in \mathbb{N}$ and all messages m that $Pr[Dec(sk, Enc(pk, m)) = m] = 1$, where $(pk, sk) \leftarrow KeyGen(1^\lambda)$

15.2 First Proposal

- Rivest, Shamir and Adleman provided the first proposal of a public key encryption scheme 1978
- Commonly referred to as **Textbook RSA** today
  - $KeyGen(1^\lambda)$: Choose two random λ-bit primes P, Q, set $N \leftarrow P \cdot Q$ and $\Phi(N) = (P-1)\cdot(Q-1)$, choose random $e \leftarrow_\$ \mathbb{Z}_{\Phi(N)}$ with $gcd(e, \Phi(N)) = 1$ and compute d s.t. $ed \equiv 1 \ mod \ \Phi(N)$. Output $pk \leftarrow (N, e)$ and $sk \leftarrow (N, d)$.
  - $Enc(pk, m \in \mathbb{Z}_N)$: Compute and output $c \leftarrow m^e \ mod \ N$.
  - $Dec(sk, c)$: Compute and output $m \leftarrow c^d \ mod \ N$.

Correctness:

$$\begin{aligned} Dec(sk, Enc(pk, m)) &= m^{e \cdot d} \ mod \ N \\ &= m^{1 + b \cdot \Phi(N)} \ mod \ N \\ &= m \cdot \underbrace{m^{b \cdot \Phi(N)}}_{=1} \ mod \ N \end{aligned}$$

15.3 What about Security?

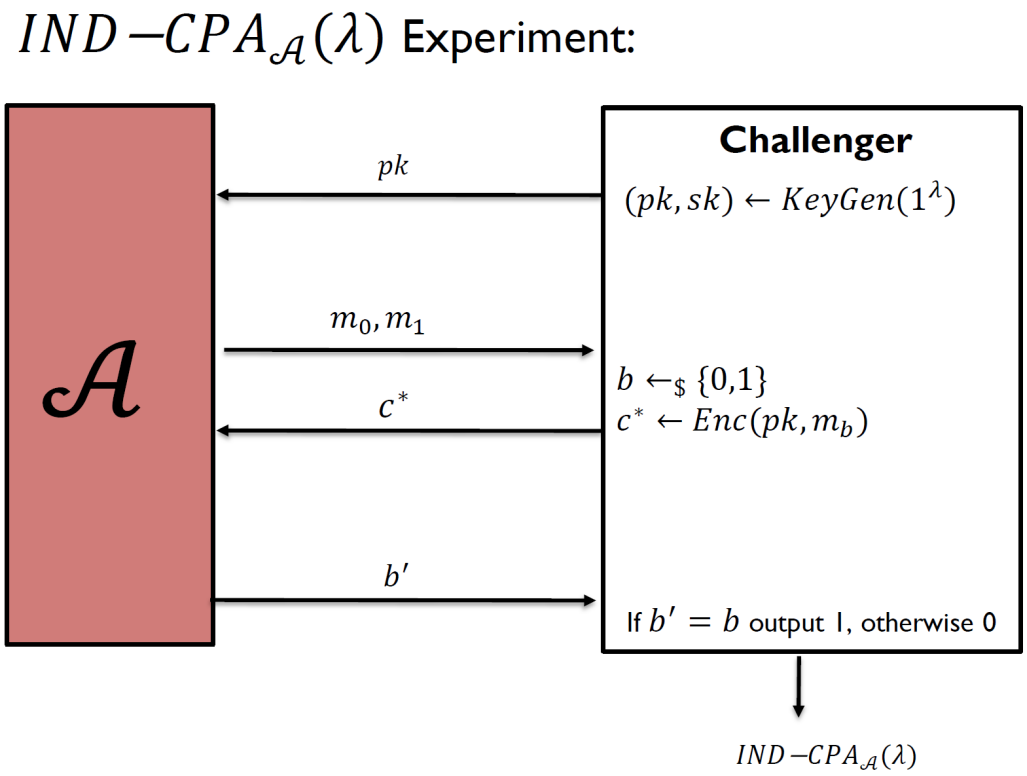
- Many problems with textbook RSA as encryption scheme
- Main problem: Encryption is deterministic
  - We can test whether a ciphertext c encrypts a message m by testing whether $c = Enc(pk, m)$
- Many other problems (features?) with textbook RSA:
  - E.g. if $c = Enc(pk, m)$, then $2^e \cdot c \ mod \ N = Enc(pk, 2m)$

15.4 Defining Security

- We will define IND-CPA security of public key encryption analogously to IND-CPA security of private key encryption
- Adversary $\mathcal{A}$ additionally gets public key pk as input
- Bonus: We don t need encryption oracle as $\mathcal{A}$ can encrypt by itself using pk

Definition 24 (IND-CPA-secure)**.** *An encryption scheme* $(KeyGen, Enc, Dec)$ *is IND-CPA-**secure**, if it holds for **PPT-secure** $\mathcal{A}$ there exists a negligible function* v *s.t. for all* $\lambda \in \mathcal{N}$

$$Pr[IND - CPA_{\mathcal{A}} = 1] < \frac{1}{2} + v(\lambda)$$



15.5 The ElGamal Cryptosystem

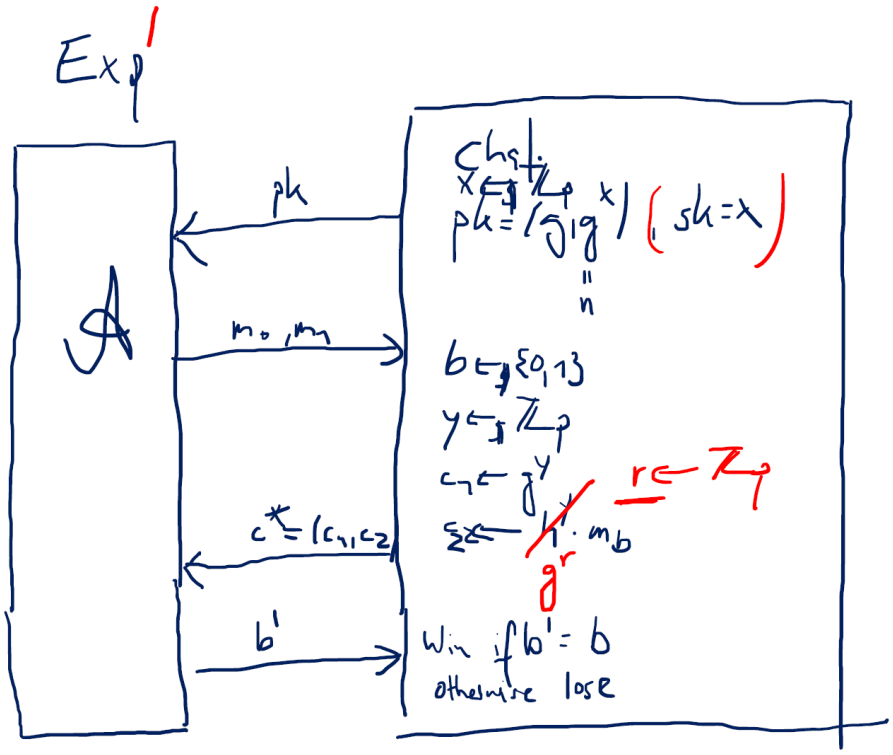
- We will now construct an encryption scheme based on the Diffie-Hellman key exchange
- The ElGamal encryption scheme $(KeyGen, Enc, Dec)$ is given as follows
  - $KeyGen(1^\lambda)$:
    - Generate cryptographic group $\mathbb{G}$ of prime order $p \approx 2^\lambda$ with generator g
    - Choose $x \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and set $h \leftarrow g^x$
    - Output $pk \leftarrow (g, h)$ and $sk \leftarrow x$
    - (implicity assume that pk include description of $\mathbb{G}$ and p)
  - $Enc(pk = (g, h), m \in \mathbb{G})$: Choose $y \leftarrow_\$ \mathbb{Z}_p$, compute and output $c \leftarrow (g^y, h^y \cdot m)$
  - $Dec(sk = x, c = (c_1, c_2))$: Compute and output $m \leftarrow c_2 \cdot c_1^{-x}$

Correctness: $(c_1, c_2) = Enc(pk, m)$, $pk = (g, h = g^x)$, $sk = x$, $c_1 = g^y$, $c_2 = h^y \cdot m$
$\Rightarrow Dec(sk = x, (c_1, c_2)) = c_2 \cdot c_1^{-x} = h^y \cdot m \cdot (g^y)^{-x} = g^{x \cdot y} \cdot m \cdot g^{-x \cdot y} = m$

Theorem 11.

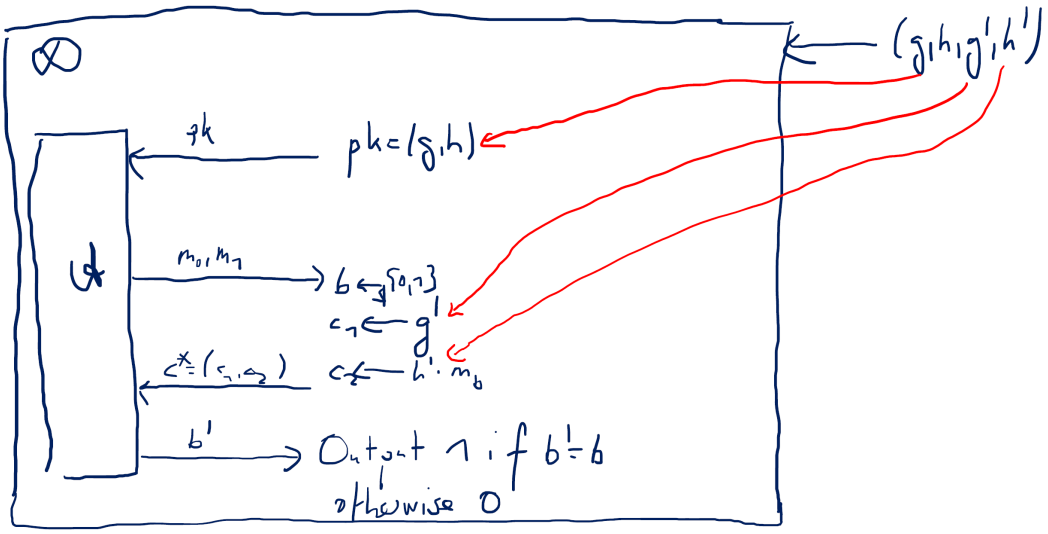
- *Assume that the DDH-assumption holds in the group* $\mathbb{G}$
- *Then the ElGamal encryption scheme* $(KeyGen, Enc, Dec)$ *is IND-CPA secure*

Proof. Assume towards contradiction there exists a PPT adversary $\mathcal{A}$ and a non-negligible ϵ s.t. $Pr[IND-CPA_{\mathcal{A}}(\lambda) = 1] \geq \frac{1}{2} + \epsilon$
Show: This implies a PPT-distinguisher $\mathcal{D}$ against DDH, contradicting the DDH assumption.



g^r is uniform in $\mathbb{G} \Rightarrow Pr[Exp' = 1] = \frac{1}{2}$

Construct PPT-distinguisher $\mathcal{D}$ against DDH



Case 1: $(g, h, g', h') = (g, g^x, g^y, g^{xy})$, $x, y \leftarrow_\$ \mathbb{Z}_p$
$\Rightarrow$ In this case $\mathcal{D}$ faithfully simulates the IND-CPA experiment
$h = g^x$, $c_1 = g^y$, $c_2 = g^{xy} \cdot m_b = h^y \cdot m_b$
$\Rightarrow Pr[\mathcal{D}(g, g^x, g^y, g^{xy}) = 1] = Pr[IND - CPA_{\mathcal{A}}(\lambda) = 1] \geq \frac{1}{2} + \epsilon$

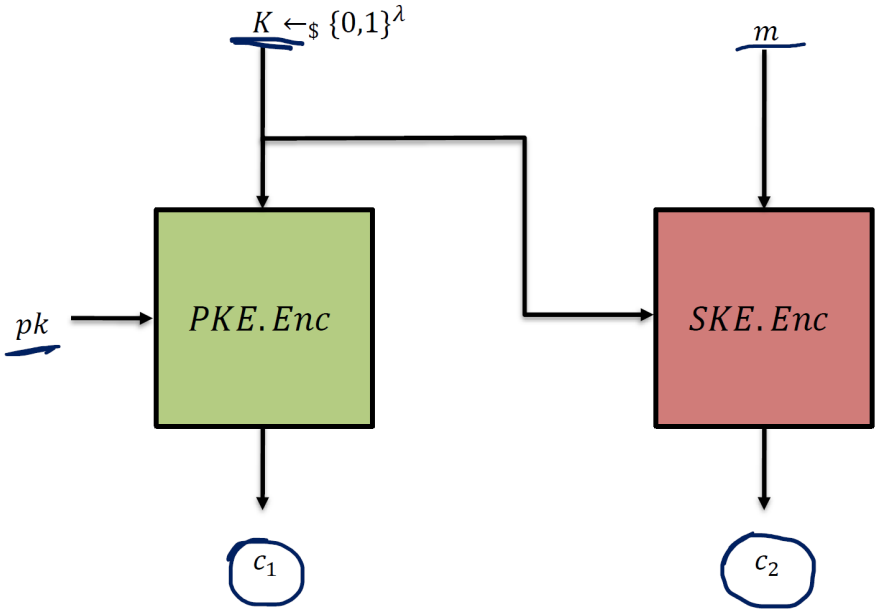
Case 2: $(g, h, g', h') = (g, g^x, g^y, g^r)$, $x, y, r \leftarrow_\$ \mathbb{Z}_p$
$\Rightarrow$ In this case $\mathcal{D}$ faithfully simulates Exp'
$\Rightarrow Pr[\mathcal{D}(g, g^x, g^y, g^r) = 1] = Pr[Exp' = 1] = \frac{1}{2}$
$\Rightarrow |Pr[\mathcal{D}(g, g^x, g^y, g^{xy}) = 1] - Pr[\mathcal{D}(g, g^x, g^y, g^r) = 1]| \geq \frac{1}{2} + \epsilon - \frac{1}{2} = \epsilon$ which is non-negligible!
$\Rightarrow \mathcal{D}$ distinguishes DDH-problem with advantage ϵ $\square$

15.6 Summary

- Public key encryption allows for secure communication without prior interaction
- The textbook RSA scheme is not an encryption scheme by modern standards (...but a useful building block)
- The standard security notion of public key encryption is IND CPA security, adapted from the private key definition
- The ElGamal encryption scheme is IND CPA secure under the Decisional Diffie Hellman (DDH) assumption

Hybrid Encryption

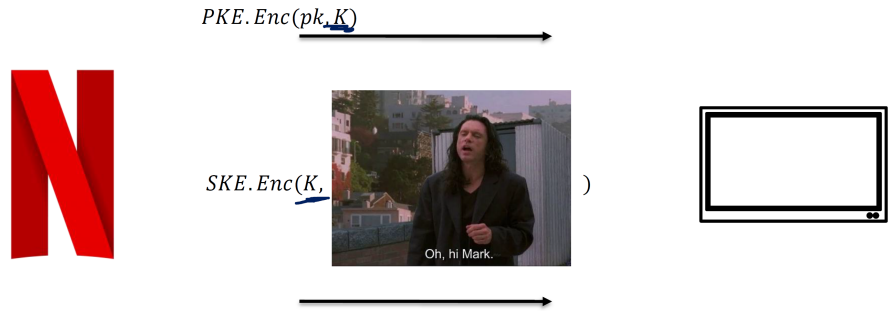
- Public Key Encryption is expensive
- 2-3 orders of magnitude slower than private key encryption
- AES Encryption rate: $\sim 700MB/s$ per core
- RSA Encryption rate: $\sim 1.5MB/s$ per core
- How can we efficiently encrypt large amounts of data?
  $\Rightarrow$ Combine public and private key encryption to get the best of both!
- Idea: For every encryption, generate a fresh K key for a private key encryption scheme
- Use K to encrypt the payload message
- Encrypt K using a public key encryption scheme



16.1 Advantages of Hybrid Encryption

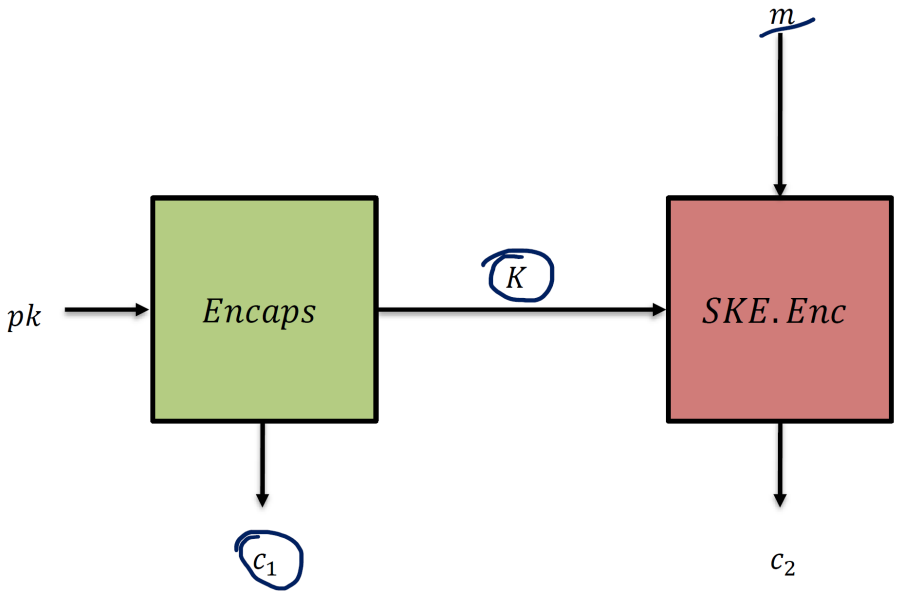
- Key K is short, e.g. 256 bits
- Thus expensive public key operation only performed once for a very short message
- The bulk of work is performed by the *fast* private key encryption scheme
- Also minimizes the size of the ciphertext for large messages $\rightarrow$ saves bandwidth
- For private key encryption $|c_2| \approx |m|$

- Thus $\frac{|c|}{|m|} \approx \frac{|c_1|+|m|}{|m|} = \frac{|c_1|}{|m|} + 1 \approx 1$



16.2 Key Encapsulation

- Using public key encryption this way, the ciphertext c_1 merely *encapsulates* a randomly chosen key K
- Key Encapsulation Mechanism (KEM):
  Replace (public key) encryption algorithm by an algorithm which generates a key K together with a ciphertext c which encapsulates K
- This allows for small (e.g. factor ~ 2) size and efficiency improvement, as ciphertext doesn't need to encode arbitrary user-chosen messages



16.3 Key Encapsulation Mechanisms

Syntax: A key encapsulation mechanism (KEM) consists of three PPT algorithms: $(KeyGen, Encaps, Decaps)$

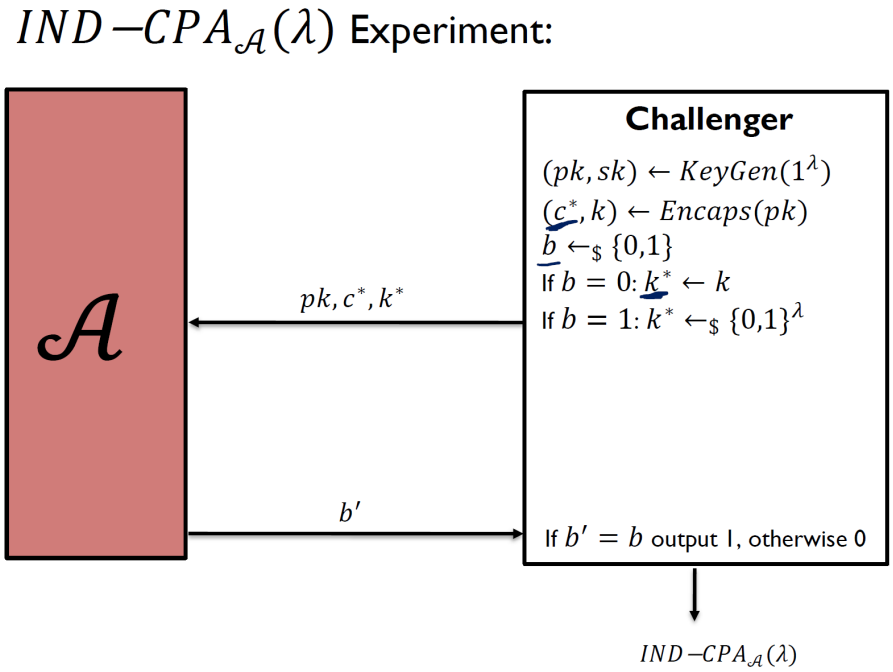
- $KeyGen(1^\lambda)$: A randomized algorithm which takes as input the security parameter 1^λ (encoded in unary) and outputs a pair of **public** and **secret** keys (pk, sk)
- $Encaps(pk)$: A randomized algorithm which takes a public key pk and outputs a ciphertext c and a key K
- $Decaps(sk, c)$: A deterministic algorithm which takes as a secret key sk and a ciphertext c as input and outputs a key K

Correctness: It holds for all $\lambda \in \mathbb{N}$ that $Pr[Dec(sk, c) = K] = 1$, where $(pk, sk) \leftarrow KeyGen(1^\lambda)$ and $(c, K) \leftarrow Encaps(pk)$

16.4 CPA Security for KEMs

Definition 25. *A KEM $(KeyGen, Encaps, Decaps)$ is $IND-CPA$-**secure**, if it holds for **every PPT-adversary** $\mathcal{A}$ there exists a negligible function v s.t. for all $\lambda \in \mathbb{N}$*

$$Pr[IND - CPA_{\mathcal{A}}(\lambda) = 1] < \frac{1}{2} + v(\lambda)$$



16.5 Examples of KEMs

- Trivial KEM: Use PKE to encrypt key K
  - $Encaps(pk)$: Choose $K \leftarrow_\$ \{0,1\}^\lambda$, compute $c \leftarrow Enc(pk, K)$ and output (c, K)
  - $Decaps(sk, c)$: Compute and output $K \leftarrow Dec(sk, c)$
- Every 2-message key exchange protocol immediately yields a KEM
- Example: Diffie-Hellman KEM
  - $KeyGen(1^\lambda)$: Choose $x \leftarrow_\$ \mathbb{Z}_p$, compute $h \leftarrow g^x$, outout $pk \leftarrow (g, h)$ and $sk \leftarrow x$
  - $Encaps(pk)$: Choose $y \leftarrow_\$ \mathbb{Z}_p$, compute and outout $c \leftarrow g^y$ and $K \leftarrow h^y$
  - $Decaps(sk, c)$: Compute and output $K \leftarrow c^x$
- Ciphertext is only one group element (ElGamal: 2 group elements)
- IND-CPA security follows immediately from eavesdropping security of Diffie-Hellman key exchange (the two experiments are identical)

16.6 Public Key Encryption from KEM + Private Key Encryption

- We will now show that the above construction actually yields an IND-CPA secure encryption scheme
- Let $(KeyGen, Encaps, Decaps)$ be a KEM and (Enc, Dec) be a private key encryption scheme (with uniformly random keys). The public key encryption scheme $(KeyGen', Enc', Dec')$ is given as follows
  - $KeyGen'(1^\lambda)$: Compute and output $(pk, sk) \leftarrow KeyGen(1^\lambda)$
  - $Enc(pk, m)$: Compute