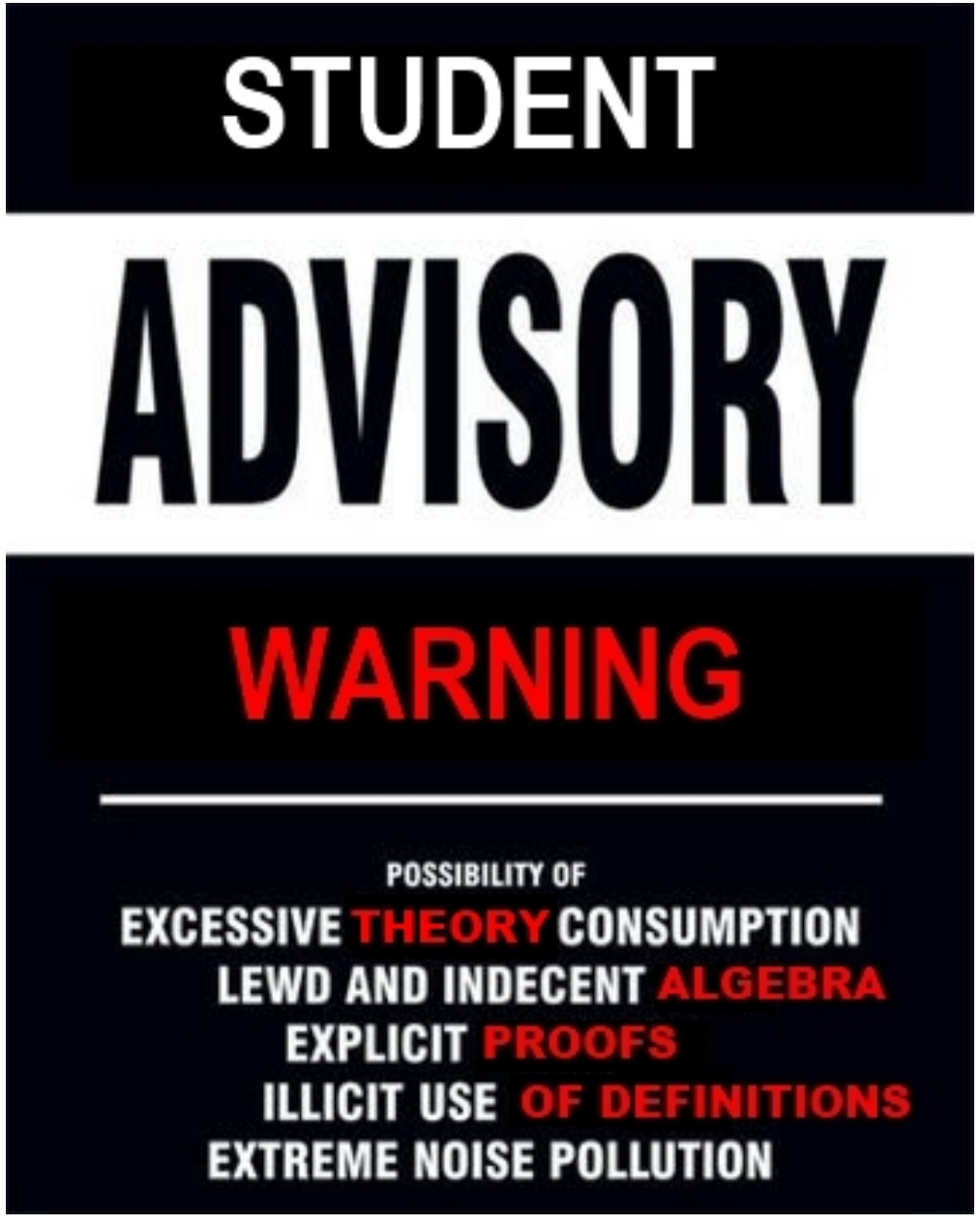
Introduction to
Cryptography
Lecture 01

©Michael Luttenberger

Chair for Foundations of Software Reliability and Theoretical Computer Science
Technische Universität München

2018/10/30, 13:27


STUDENT
ADVISORY
WARNING
POSSIBILITY OF
EXCESSIVE THEORY CONSUMPTION
LEWD AND INDECENT ALGEBRA
EXPLICIT PROOFS
ILLICIT USE OF DEFINITIONS
EXTREME NOISE POLLUTION

Outline of today's lecture

- Organization
- Fundamental goals of cryptography
- Overview on cryptographic schemes
- Content and goals of the lecture

Dates and time: Tue&Wed 16:00 in IMETUM

Tutorials: alternate with the lectures (no fixed alternation; goal: mostly on Wednesdays every second week)

Short tests: allow to obtain a bonus of "'0.3"; voluntarily; at the end of the tutorials; roughly 10 – 15 min; 10 points per test; bonus if at least 66% of all points obtained; bonus applicable to the two exams for this winterterm 2018/19 (endterm, retake)

Exams: depending on number of participants

either written or oral (will be decided after registration period has ended)

allowed auxiliary means: only a non-programmable calculator.

Slides:

slides won't change much compared to last year;

once in a while "password quizzes": details will be posted on the webpage

1. Lecture 1 – Introduction: goals and locks
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

1. Lecture 1 – Introduction: goals and locks
   - Central goals of cryptography
   - Cryptographic schemes ("locks")
   - Private-key vs. public-key
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

Introduction

Alice: I'm at Munich.

Alice: Hi, I'm at Munich.

Alice

Eve

Bob

"Cryptography is about communication in the presence of adversaries." (R. L. Rivest [9])

Alice: I'm at Munich.

Alice: Hi, I'm at Munich.

Alice

Eve

Bob

"We like to compare cryptography to locks [...]. The lock is just a small part of a much larger security system. The same goes for cryptography[...]." (Ferguson/Schneier [5])

Alice: I'm at Munich.

Alice: Hi, I'm at Munich.

Alice

Eve

Bob

"'*The new Snowden revelations are explosive. Basically, the NSA is able to decrypt most of the Internet. They're doing it primarily by cheating, not by mathematics. [...] Remember this: The math is good, but math has no agency. Code has agency, and the code has been subverted.*" (*Schneier on Snowden/NSA*)

Alice: I'm at Munich.

Alice: Hi, I'm at Munich.

Alice

Eve

Bob

- Single message m from Alice to Bob using a public channel.
  - ▷ $m \in \Sigma^+$, in general $\Sigma = \{0, 1\}$.
  - ▷ E.g.: m is the ASCII encoding of "Alice: I'm at Munich."
  - ▷ We don't (want to) care about the concrete encoding.
- Adversary Eve may eavesdrop, intercept, replace, forge, ... messages.
- **Remark**: We do not consider transmission errors in this lecture.

What is cryptography?

Alice: I'm at Munich.

Alice: Hi, I'm at Munich.

Alice

Eve

Bob

- Central goals of cryptography for this lecture:

Message privacy	Identity authentication
Message authenticity (and integrity)	Non-repudiation

Alice: I'm at Munich.

Alice: Hi, I'm at Munich.

Alice

Eve

Bob

- Message privacy: (also: confidentiality/secrecy)
  "A service used to keep the content of information from all but those authorized to have it." [8]
  - ▷ "Eve should not be able to extract any relevant information from intercepted messages."

Alice: Eve is not so nice.

Alice: Eve is nice.

Alice

Eve

Bob

- Message authenticity: (also: message origin authentication)
  "A service related to identification. Messages should be authenticated as to origin, date of origin, data content, time sent, etc." [8]
  - ▷ "Bob should be able to verify that a message he believes to come from Alice was indeed sent by her."

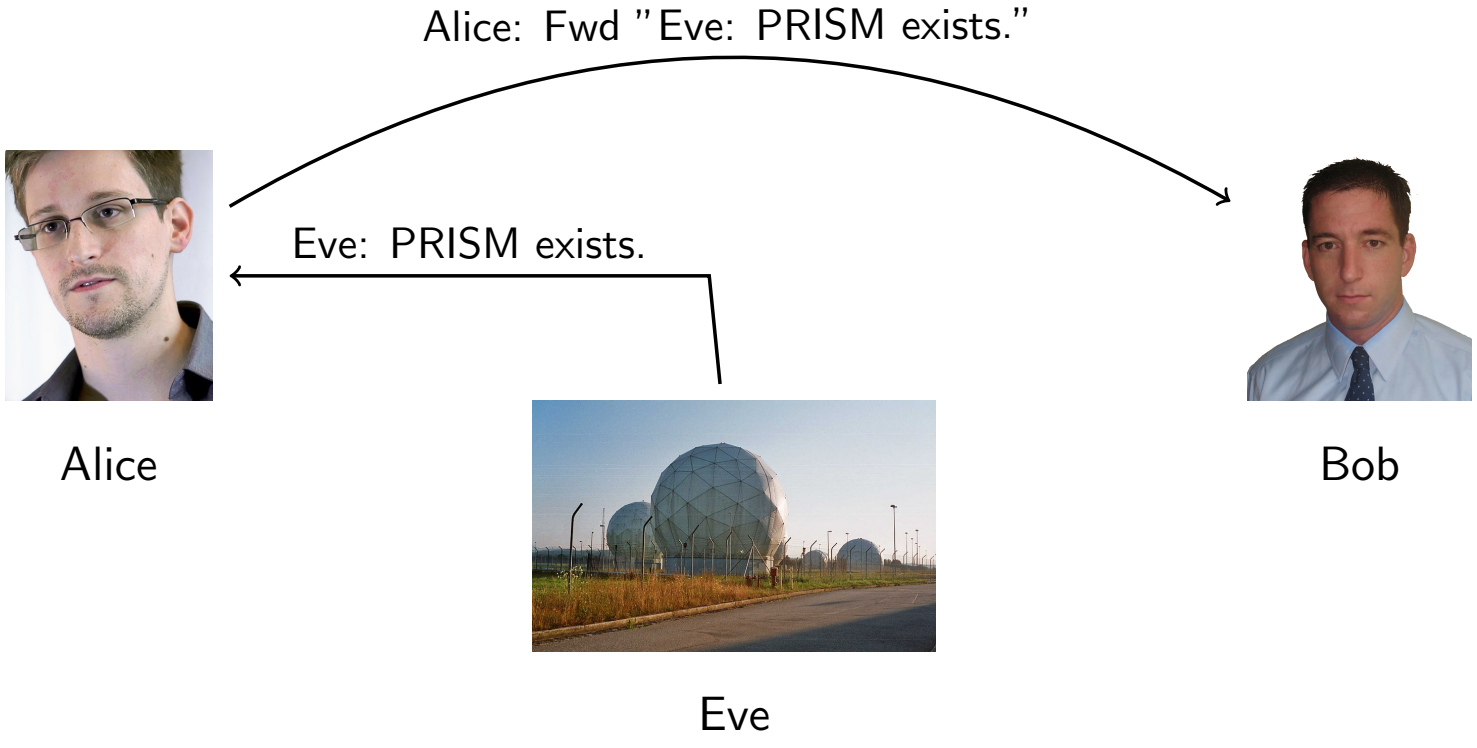
Alice

Eve

Bob

Alice: I'm Alice.

- Identity authentication:
  "Two parties entering into a communication should identify each other." [8]
  - ▷ "Eve shouldn't be able to trick Bob into believing she was Alice"
  - ▷ Important when Alice and Bob have never met before.



- Non-repudiation:
  "A service which prevents an entity from denying previous commitments or actions." [8]
  - ▷ "Bob should be able to convince other parties (e.g. a judge) that a message by e.g. Eve was indeed sent by Eve and not someone else."
  - ▷ Better example: webshop

1. Lecture 1 – Introduction: goals and locks
   - Central goals of cryptography
   - Cryptographic schemes ("locks")
   - Private-key vs. public-key
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

- Overview:

	symmetric setting	asymmetric setting
message privacy	private-key encryption scheme (ES)	public-key encrytion scheme (PKES)
message authenticity	message authentication code (MAC)	digital signature scheme (DSS)

- "Locks" are usually called cryptographic schemes.
- Symmetric aka. private key setting:
  A secret (key) shared by Alice and Bob.
- Asymmetric aka. public key setting:
  Each party holds its own secret (key) and broadcasts a public key.

- Alice wants to send m to Bob.

	ES/PKES	MAC/DSS
1.	B: $(\mathsf{ek}, \mathsf{dk}) \stackrel{r}{:=} \mathsf{Gen}$	A: $(\mathsf{sk}, \mathsf{vk}) \stackrel{r}{:=} \mathsf{Gen}$
2.	B: Gives ek to A	A: Gives vk to B
3.	A: $c \stackrel{r}{:=} \mathsf{Enc}_{\mathsf{ek}}(m)$	A: $t \stackrel{r}{:=} \mathsf{Mac}_{\mathsf{sk}}(m)$
4.	A: Sends c to B	A: Sends $m\|\|t$ to B
5.	B: $m := \mathsf{Dec}_{\mathsf{dk}}(c)$	B: Checks $\mathsf{Vrf}_{\mathsf{vk}}(m, t) = \mathsf{true}$

- "lock" interfaces:

$$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) \quad \text{resp.} \quad (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrf})$$

- "$\stackrel{r}{:=}$": output of a randomized algorithm.
- "Gives": secure way of passing the key.
- "Sends": public channel.

- Alice wants to send m to Bob.

	ES/PKES	MAC/DSS
1.	B: $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{r} \mathsf{Gen}$	A: $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{r} \mathsf{Gen}$
2.	B: Gives ek to A	A: Gives vk to B
3.	A: $c \xleftarrow{r} \mathsf{Enc}_{\mathsf{ek}}(m)$	A: $t \xleftarrow{r} \mathsf{Mac}_{\mathsf{sk}}(m)$
4.	A: Sends c to B	A: Sends $m\|\|t$ to B
5.	B: $m := \mathsf{Dec}_{\mathsf{dk}}(c)$	B: Checks $\mathsf{Vrf}_{\mathsf{vk}}(m, t) = \mathsf{true}$

- Private key setting: $\mathsf{ek} = \mathsf{dk}$ resp. $\mathsf{vk} = \mathsf{sk}$.
  - Keys only known to Alice and Bob.
  - Shared secret.

- Alice wants to send m to Bob.

	ES/PKES	MAC/DSS
1.	B: $(\mathsf{ek}, \mathsf{dk}) \stackrel{r}{:=} \mathsf{Gen}$	A: $(\mathsf{sk}, \mathsf{vk}) \stackrel{r}{:=} \mathsf{Gen}$
2.	B: Gives ek to A	A: Gives vk to B
3.	A: $c \stackrel{r}{:=} \mathsf{Enc}_{\mathsf{ek}}(m)$	A: $t \stackrel{r}{:=} \mathsf{Mac}_{\mathsf{sk}}(m)$
4.	A: Sends c to B	A: Sends $m\|\|t$ to B
5.	B: $m := \mathsf{Dec}_{\mathsf{dk}}(c)$	B: Checks $\mathsf{Vrf}_{\mathsf{vk}}(m, t) = \mathsf{true}$

- Public key setting: $\mathsf{ek} \neq \mathsf{dk}$ resp. $\mathsf{vk} \neq \mathsf{sk}$.
  - secret: dk (decryption) and sk (signing)
  - public: ek (encryption) and vk (verification)
  - Secret restricted to one entity.

- Alice wants to send m to Bob.

	ES/PKES	MAC/DSS
1.	B: $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{r} \mathsf{Gen}$	A: $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{r} \mathsf{Gen}$
2.	B: Gives ek to A	A: Gives vk to B
3.	A: $c \xleftarrow{r} \mathsf{Enc}_{\mathsf{ek}}(m)$	A: $t \xleftarrow{r} \mathsf{Mac}_{\mathsf{sk}}(m)$
4.	A: Sends c to B	A: Sends $m\|\|t$ to B
5.	B: $m := \mathsf{Dec}_{\mathsf{dk}}(c)$	B: Checks $\mathsf{Vrf}_{\mathsf{vk}}(m, t) = \mathsf{true}$

▷ Informal security assumption:

If Eve does not know dk resp. sk, then it is "infeasible" to her to

- obtain any relevant information on m from c resp.
- compute any "'valid" tag t' for any new message m'.

1. Lecture 1 – Introduction: goals and locks
   - Central goals of cryptography
   - Cryptographic schemes ("locks")
   - Private-key vs. public-key
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

- Public-key schemes seem to be superior:
- ▷ Exchanging the secret key not necessary.
  - ▷ But how to make sure that everyone gets the correct public key?
- ▷ The knowledge of the secret key identifies Alice (resp. Bob) which enables identity authentication and non-repudiation.
  - ▷ But what if someone steals this secret without Alice noticing it?
- ▷ Security of public-key schemes usually based on well researched long-standing mathematical problems.
  - Problems existed before being used for cryptography
  - Also studied by researchers not interested in cryptography.
  - In contrast: most private-key schemes rely on block ciphers or hash functions constructed to withstand all currently known attacks.

- The major disadvantage of public-key schemes: Speed

  "By comparison, DES (see Section 3.2) and other block ciphers are much faster than the RSA algorithm. DES is generally at least 100 times as fast in software and between 1,000 and 10,000 times as fast in hardware, depending on the implementation." (RSA labs.)

  - ▷ AES support in most recent CPUs, see e.g. wikipedia
- Solution: combine both (hybrid encryption)
  - ▷ See e.g. TLS.

Example: (private-key) encryption using a block cipher

- Modern private-key ES $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ are built from block ciphers
- **Definition**: Block cipher $B\colon \Sigma^n \to \mathsf{Prm}_{\Sigma,l}\colon k \mapsto B_k$
  - Σ: alphabet; if not stated otherwise: $\Sigma = \{0,1\}$
  - n: key length
  - l: block length
  - $\mathsf{Prm}_{\Sigma,l}$: set of all permutations of Σ^l

  where $B_k(w)$ and $B_k^{-1}(w)$ can be computed "fast".
- **Examples**:
  - Prior to computers: $\Sigma = \{a, b, \ldots, z\}$, $n = l$ (e.g. Ceasar: $l = 1$)
  - DES: $\Sigma = \{0,1\}$, $n = 56$, $l = 64$
  - AES-finalists: $\Sigma = \{0,1\}$, $n \in \{128, 192, 256\}$, $l = 128$
  - Threefish: $\Sigma = \{0,1\}$, $n \in \{256, 512, 1024\}$, $l = n$

- 1. Question: Choice of the key generator Gen?

▷ Often outputs a key k to be used directly in the block cipher, e.g. $k \in \{0,1\}^{128}$

▷ "Common" choices for Gen:

  - Output some 128bit encoding of

    password, 123456, qwerty, letmein, 654321, abc123, [birthday of some relative], ...

    Used quite often, but not that clever.

  - Better approach: Choose any 128bit string uniformly at random ("'truly random"). See also next slide.

▷ **Ex**: Compare the number of possible passwords consisting of at most 8 (16) alpha-numerical characters to $\left|\{0,1\}^{128}\right|$.

- **Ex**: "The power of uniform distribution"

  Let K be uniformly distributed over $\{0,1\}^n$.

  - I.e. $\Pr[K = x] = 2^{-n}$ for all $x \in \{0,1\}^n$.

  Let X be any random variable on $\{0,1\}^n$.

  - X might also be constant.

  Further, assume that K and X are independent. Show:

  - $\Pr[K = X] = 2^{-n}$

    e.g. let K be Alice's secret key, and X Eve's guess what K might be,

    then Eve's chance to guess K is minimized if Alice chooses K uniformly at random.

  - $K \oplus X$ is again uniformly distributed on $\{0,1\}^n$ where $\oplus$ is the bit-wise XOR operation (e.g. $001 \oplus 101 = 100$)

    e.g. let X be some message, and $K \oplus X$ Alice's encryption of X,

    then the encrypted text is as "random as it can be".

- 2. Question: Choice of Enc and Dec?

▷ Problem: AES e.g. accepts only blocks of 128bit as input.

  ▷ **Assumption**: $m \in \{0,1\}^*$ is already the encoding (ASCII,UTF8,...) of the actual message.

▷ Solution: Pad m to a length divisible by $l = 128$.

Many possibilities: (choose i so that l divides $|\mathsf{pad}(m)|$.)

- $\mathsf{pad}(m) = m||0^i$
- $\mathsf{pad}(m) = m||10^i$
- $\mathsf{pad}(m) = m||0^i||\lfloor|m|\rceil_l$ ($\lfloor|m|\rceil_l$: l-bit encoding of the length $|m|$)
- $\mathsf{pad}(m) = \lfloor|m|\rceil_l||m||0^i$

▷ **Ex**: How can you reverse each padding? Which padding is more space efficient?

- 2. Question: Choice of Enc and Dec? (cont'd)

▷ Assume some padding $\mathsf{pad}(m) = m^{(1)}|| \ldots ||m^{(s)}$ for now.

- **Notation**: $m^{(i)}$ is a block of $l = 128$ bits

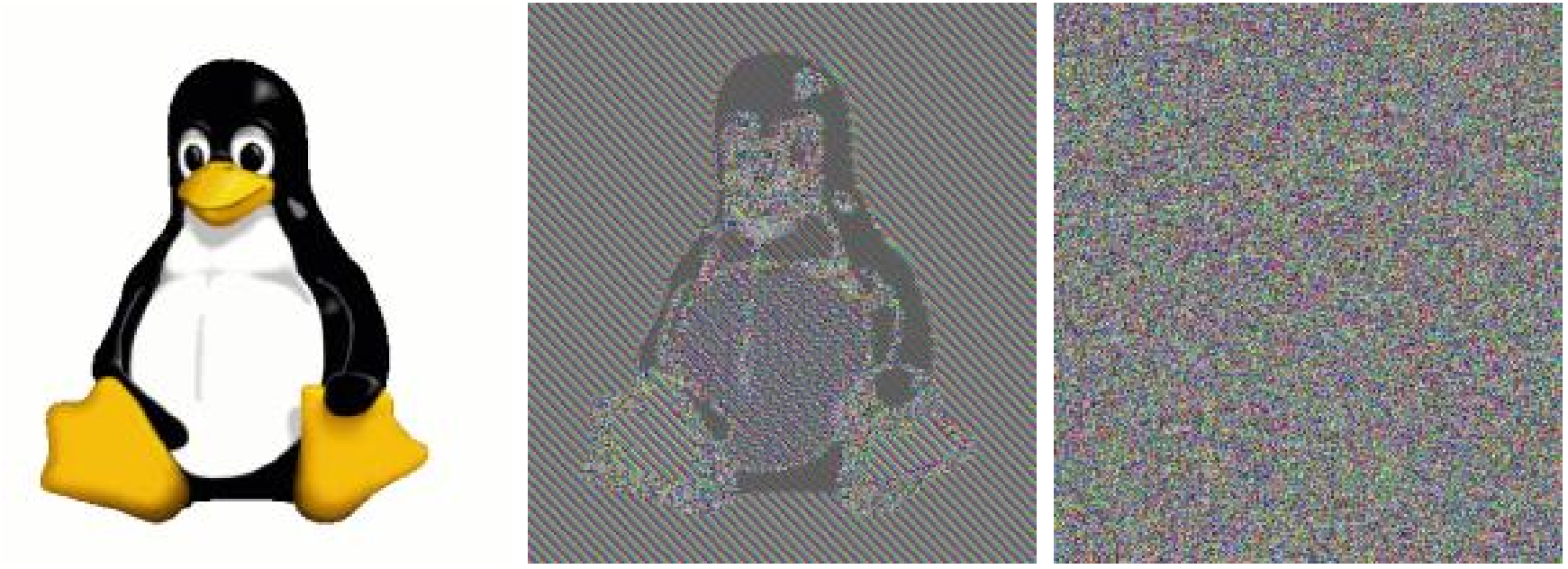
Many possibilities to process the blocks using the secret key k:

- $\mathsf{Enc}_k(m) := \mathsf{AES}_k(m^{(1)})|| \ldots ||\mathsf{AES}_k(m^{(s)})$
- $\mathsf{Enc}_k(m) := \rho||m^{(1)} \oplus \mathsf{AES}_k(\rho)|| \ldots ||m^{(s)} \oplus \mathsf{AES}_k^s(\rho)$
- $\mathsf{Enc}_k(m) := \mathsf{AES}_k(\rho)||m^{(1)} \oplus \mathsf{AES}_k(\rho)|| \ldots ||m^{(s)} \oplus \mathsf{AES}_k^s(\rho)$
- $\mathsf{Enc}_k(m) := \rho^{(1)}||m^{(1)} \oplus \mathsf{AES}_k(\rho^{(1)})|| \ldots ||\rho^{(s)}||m^{(s)} \oplus \mathsf{AES}_k(\rho^{(s)})$

▷ where: $\rho, \rho^{(i)} \stackrel{u}{\in} \{0,1\}^l$ chosen independently, uniformly at random for every m; AES_k^i means to apply AES_k i times; $\oplus$ denotes bitwise XOR.

▷ **Ex**: Specify Dec in each case. Compare the size of the ciphertext. Which do you think is "secure" (in what sense)?

Example: private-key ES



Taken from wikipedia

- Summary:
  - ▷ 1. Observation: block cipher $\neq$ encryption scheme.
    - Many different constructions: CTR, CBC, OFB, OCB, XTS, GCM, ...
  - ▷ 2. Observation: concrete block cipher unimportant for construction
    - Constructions parametrized on the block length l and the key length n.
  - ▷ 3. Observation: "good" block cipher $\not\Rightarrow$ "secure" encryption scheme
    - No matter what B is (AES,DES,Twofish,Serpent,...), this leaks $m^{(1)}$:

      $\mathsf{Enc}_k(m) := \mathsf{B}_k(\rho)||m^{(1)} \oplus \mathsf{B}_k(\rho)|| \ldots ||m^{(s)} \oplus \mathsf{B}_k^s(\rho)$
- "Modern" cryptography therefore distinguishes between:
  - What is a "good" block cipher?
  - Is AES a "good" block cipher?
  - Does a "good" block cipher exist at all?
  - Given a "good" block cipher, how to built a "secure" ES?

- Most important terms/definitions/notions used in cryptography.
- What a block cipher like AES should achieve.
- What a hash function like KECCAK should do.
- "provable security"
  - What we need to assume
  - What we can prove to be attainable based on these assumption.
- Better understanding of modern cryptographic schemes:
  - Which ES/PKES/MAC/DSS should be used.
  - Motivation for some of the planned changes in TLS 1.3. (AEAD, Enc-then-Mac)

- Concrete attacks on some specific block cipher, protocol, . . .
- Side-channel attacks.
  - See e.g. the talk by Adi Shamir here.
- Detailed study of the construction of block ciphers or hash functions.
- Design of cryptographic protocols (combining the locks) like TLS or SSH.

▷ See the courses offered by the Chair of IT Security (Prof. Eckert)

Outline

- Provable security and perfect secrecy
- Computational secrecy and pseudorandom generators
- Computation security beyond eavesdropping and pseudorandom functions/permutations
- DES and AES
- Algebra (and Rabin-Miller test)
- Pseudorandom generators from one-way functions
- Secure hash functions, Keccak/SHA3
- Public-key encryption schemes
- Digital signature schemes
- Elliptic curves

- Main goals of cryptography and corresponding schemes

  (names and interfaces)
- Interface of a block cipher
- Why "encrypted using AES" is a pointless statement
- Padding schemes
- The need to prove (reduce) that e.g.

  an ES is "secure" as the used block cipher is (hopefully) "secure".
- Why to use private-key schemes

1. Lecture 1 – Introduction: goals and locks
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

1. Lecture 1 – Introduction: goals and locks

2. Lecture 2 – Principles and randomized computation

   - Principles of (modern) cryptography
   - Randomized computation
   - Source of randomness

3. Lecture 3 – Perfect secrecy

4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

"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."

- Short: "The enemy knows the system." (Shannon's maxim)
- Main motivations:
  - Allows for standards (e.g. NIST standards DES, AES, SHA) and validation of implementations (CAVP).
    - ▷ See e.g. Dual_EC_DRBG.
  - People have more confidence in an (unbroken) open system.
  - Intuitively, a lock should be secure because only you have the key to it, i.e.

    security should only depend on the secret key.
  - A short key is easier to keep secret/store/exchange/replace.

"Any cryptographic scheme must have a key space that is not vulnerable to exhaustive search."

- Key space $\mathcal{K}$:

  Set of all keys generated by the key generator Gen

- Testing all keys in $|\mathcal{K}|$ should be impracticable for at least the near future.

▷ Problem: How to define what "impracticable for the near future" means?

  - What kind of computational power should we assume that Eve has?

  ▷ Does "Eve" only possess a standard pc? Or a supercomputer/botnet/cloud? Or perhaps a quantum computer?

  ▷ See also Timeline of quantum computing (wikipedia).

"Any cryptographic scheme must have a key space that is not vulnerable to exhaustive search."

- Assume a modern 8c/16t cpu running at 4 GHz that can test one key per thread and clock signal (independent of the actual ciphertext).
- ▷ Time needed for checking
  - ▷ a list with $500 \cdot 10^6$ stolen passwords: 0.0078125s

    `https://haveibeenpwned.com/Passwords`
  - ▷ all DES keys $\{0,1\}^{56}$: 13d
  - ▷ all AES-128 keys $\{0,1\}^{128}$: $169 \cdot 10^{18}$y
  - For comparison:

    Estimated age of the universe: $13.798 \cdot 10^9$y

    Hazel Hen (HLRS): 185088c/370176t

"Any cryptographic scheme must have a key space that is not vulnerable to exhaustive search."

- **BUT** if Eve had a quantum pc, she might be able to test all AES-128 keys within time $\mathcal{O}(2^{64})$ only.
  - See Grover's algorithm for details.
  - ▷ So perhaps better use 256bit keys?!
  - Problem remains: Our estimation resp. over-approximation of the capabilities of "Eve" is based on our knowledge and experience, and we might be wrong (See NSA/Snowden).

- Simple encrpytion scheme:
  - Alphabet: $\Sigma = \{A, B, C, \ldots, Z\}$
  - Key space: $\mathcal{K} := \mathsf{Prm}_{\Sigma,1}$, i.e. all permutations of Σ.
  - Encryption of a message $m = m_1 m_2 \ldots m_s \in \Sigma^*$:

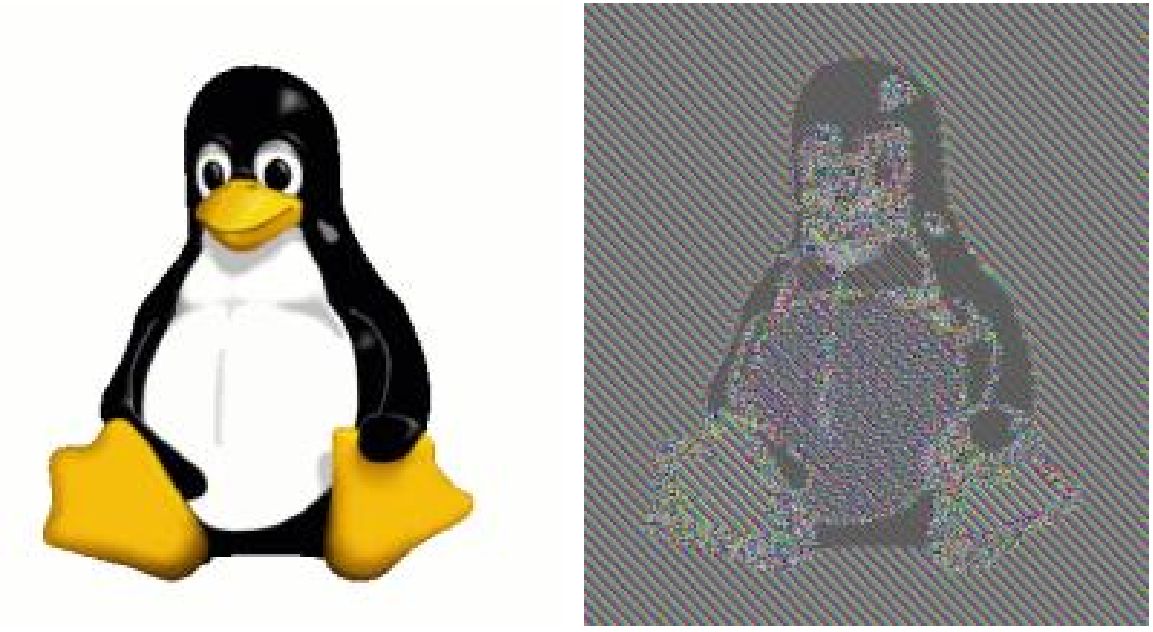
    $$\mathsf{Enc}_\pi(m) = \pi(m_1)||\pi(m_2)||\ldots||\pi(m_s)$$

    for $\pi \in \mathcal{K} = \mathsf{Prm}_{\Sigma,1}$.
  - ▷ Alternative view:

    The key π is essentially the block cipher (of block length 1) used in "ECB" mode.
  - Decryption: obvious.

▷ Size of key space: $26! \approx 2^{88}$ (on "our" 8c/16t-cpu: $200 \cdot 10^6$y).

▷ Should suffice for the near future!?



Taken from wikipedia

- Mono-alphabetic substitution/ECB with alphabet Σ (roughly spoken) RBG color values.
- Colors are only permuted, but the same unencrypted color is always on the same encrypted color, thus are large fraction of the edges are preserved.

- **But**: The key can easily be found using frequency analysis:

▷ Let $p_A(m)$ be the frequency of the letter A in the text $m \in \Sigma^*$.

Then $p_a(m) = p_{\pi(a)}(c)$ for all $a \in \Sigma$.

▷ For large texts ("$|m| \to \infty$"), frequency of a letter only depends on the language, e.g. for US-English:

$p_A^{\text{US}} \approx 0.082$, $p_B^{\text{US}} \approx 0.015$, $p_C^{\text{US}} \approx 0.028$, $p_D^{\text{US}} \approx 0.0042$, $p_E^{\text{US}} \approx 0.127$.

▷ Consider only keys π' which satisfy $p_a^{\text{Eng}} \approx p_{\pi'(a)}(c)$ for all $a \in \Sigma$.

▷ Drastically reduces the search space, i.e. effective key space is much smaller than 2^{88}.

(Attack already known in 9th century; attributed to Al-Kindi.)

▷ **Note**: "Sufficient key space principle" is necessary, but not sufficient for a scheme to be secure.

- **Ex**: Consider the following slight adaption:

  $\mathsf{Enc}_\pi(m) = \rho_1 || \pi(\rho_1) \oplus m_1 || \ldots || \rho_l || \pi(\rho_l) \oplus m_s$

  where

  - Each ρ_i is chosen uniformly at random from $\Sigma = \{A, B, C, \ldots, Z\}$.
  - Addition on Σ is defined by means of the bijection $\Sigma \to \mathbb{Z}_{26}$ where $A \mapsto 0, B \mapsto 1, \ldots, Z \mapsto 25$ and the addition modulo 26.

▷ Aadapt the "frequency analysis"-attack to this encrytion scheme.

How does $|\Sigma|$ influence the speed of your attack?

- Above construction works for any block cipher of block length l:

  Replace π by B_k, break the message up into blocks of length l.

▷ Does your attack still work? How is it influenced by l?

▷ Is above scheme insecure because of the used construction, the used block cipher, or both?

- **Goal**:

  We would like to assess how hard it is to break a cryptographic scheme, i.e. determine its "effective key space".

  Problem:

  The "effective key space" depends on our knowledge.

  - E.g. so far the best way to "break" AES seems to be brute-force.
    - For now let "break" mean: Given $x, y \in \{0,1\}^{128}$ find a key k s.t. $\mathrm{AES}_k(x) = y$.

    But there is no proof that this is true.

  "Practical" cryptographic schemes are thus based on some conjecture.

  - Conjecture: "An unproven assumption which is reasonable given our current knowledge and/or experience."
  - It can be shown: $\mathbf{P} \neq \mathbf{NP}$ is necessary for practical schemes.

Partial solution: Principle of provable security.

1. Formalize a definition of security including adversarial model:

   The adversarial model defines exactly Eve's capabilities and knowledge.

   The definition of security states when Eve successfully breaks the given scheme.

2. State the assumption (the cryptographic primitive that has to exist) underlying the construction.

3. Prove that the constructed scheme satisfies the definition using only the stated assumption. (prove-by-reduction)

- Adversarial model: What Eve knows and what she is allowed to do.
- ▷ In particular, how she can interact with the cryptographic scheme (type of attack), e.g.:
  - Eavesdropping (passive listener): Eve can only copy the information sent over the public channel.
  - Chosen-plaintext attack (CPA) (subsumes eavesdropping): Eve can further influence what is sent over the public channel.
  - Chosen-ciphertext attack (CCA) (subsumes CPA): Eve may further ask for decryptions of ciphertexts created by her.
  - Side-channel attack (not discussed): Eve can measure the resources consumed by the en/decryption e.g. on a smart card (time/energy).

  and her computational power, e.g.:
  - Model of computation: deterministic (classical), probabilistic, quantum
  - Bounds on the time or messages: none, asymptotic, concrete

- Definition of security: What Eve (as described by the adversarial model) tries to do,

  i.e. what the cryptographic scheme should prevent her from doing.

  - ▷ Usually this means a bound on the probability that she succeeds.
    - As Eve might simply be lucky and guess the secret key.
  - ▷ But when is a definition of security "good"?
    - You still have to decide if a definition of security is the right one for your concrete setting/application.
    - ▷ We will study the basic definitions of security used today:

      perfect secrecy, IND-CPA, IND-CCA, secure MACs/DSS

- Assumption: Everything we need to assume in order to prove that the scheme satisfies the given definition of security.
  - ▷ Usually computationally hard ("infeasible") problems.
  - ▷ Cryptographic primitives: Standard assumptions.
    - So far, their existence has to be conjectured.
    - ▷ An (unconditional) proof of their existence would also show: $\mathbf{P} \neq \mathbf{NP}$.
    - Allows to separate the conjecture

      "At this point of time DES/AES is a secure block cipher"

      from the weaker conjecture

      "There is some secure block cipher B".
    - Construct schemes based on the assumption that there is some secure block cipher B; swap candidates (AES for DES) if necessary.
    - What is the weakest assumption for a given level of security?

- Proof of security: A formal proof that the constructed scheme satisfies the stated definition of security using only the stated assumption.
  - The proof reduces the security of the scheme to the assumption.
  - ▷ Usually works by showing the contraposition:

    If Eve could break the scheme as specified by the definition of security,

    then she could also show that our assumption is false.
  - ▷ More precisley such a proof shows

    how a successful attack on the proposed scheme can be transformed into an efficient algorithm for the problem assumed to be computationally hard/infeasible.
  - ▷ For instance: An efficient attack on the Rabin encryption scheme can be used to efficiently factorize integers.

- What do we gain from this?
  - ▷ Without specifying what we expect of a cryptographic scheme (definition of security), it's hard to tell if the scheme does what it is supposed to do in the first place.
  - ▷ Studying the cryptographic primitives gives us a better understanding of what cryptography can and can not achieve.
  - ▷ We can choose the construction which uses the weaker primitive.
  - ▷ From two constructions based on the same primitive, we can choose the one which is faster/attains the better security guarantee/etc.
  - ▷ We can keep the construction and simply swap one conjectured implementation of a primitive for another one.

    E.g.: Transition from DES to AES.

1. Lecture 1 – Introduction: goals and locks

2. Lecture 2 – Principles and randomized computation

   Principles of (modern) cryptography

   Randomized computation

   Source of randomness

3. Lecture 3 – Perfect secrecy

4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

- Most cryptographic schemes rely on randomization of some sort.
  - ▷ For instance, for key generation (Gen) but also encryption.
  - ▷ Randomized computation:

    Standard algorithm

    - Given as Turing machine, as RAM, as code in any modern programming language, ...

    extended by the access to a function $\mathrm{fair\_coin}$.
- **Assumption**: $\mathrm{fair\_coin}$ implements a fair coin toss

  I.e. the result of every function call is 0 with prob. $1/2$, and 1 with prob. $1/2$ independently of any previously call.

- Randomized polynomial time computation:

  Algorithm always stops in time polynomial in the input size no matter the outputs of $\mathrm{fair\_coin}$.

- Important applications:

▷ Choosing a key or a prime uniformly at random.

▷ Fast primality tests

  Randomized polynomial-time algorithms (e.g. Miller-Rabin (1980), Solovay-Strassen (1977)) prefered to deterministic polynomial-time test (Agrawal–Kayal–Saxena (2002)) because of speed.

- **Example**: Generating large random odd integers.

▷ Input: Natural number $n > 1$.

▷ Output: A random odd number in $[2^{n-1}, 2^n - 1]$.

▷ Algorithm **M**:

1. Set $x_0 = 1$, $x_{n-1} = 1$.
2. For i in 1 to $n - 2$: Set $x_i := \text{fair\_coin}()$.
3. Let x be the integer represented by the bit string $x_0 x_1 \dots x_{n-1}$ with the least-significant bit first (LSBF-rep).

- **Ex**: Determine the distribution of the output of $\mathbf{M}(n)$.

  What is the prob. that the output number is a prime?

  - Use that within $[2^{n-1}, 2^n - 1]$ there are at least $\frac{2^{n-1}}{n}$ primes.

- Sometimes it is easier to view a randomized algorithm **M** as deterministic algorithm which takes an additional input:

▷ Instead of calling fair_coin within the algorithm,

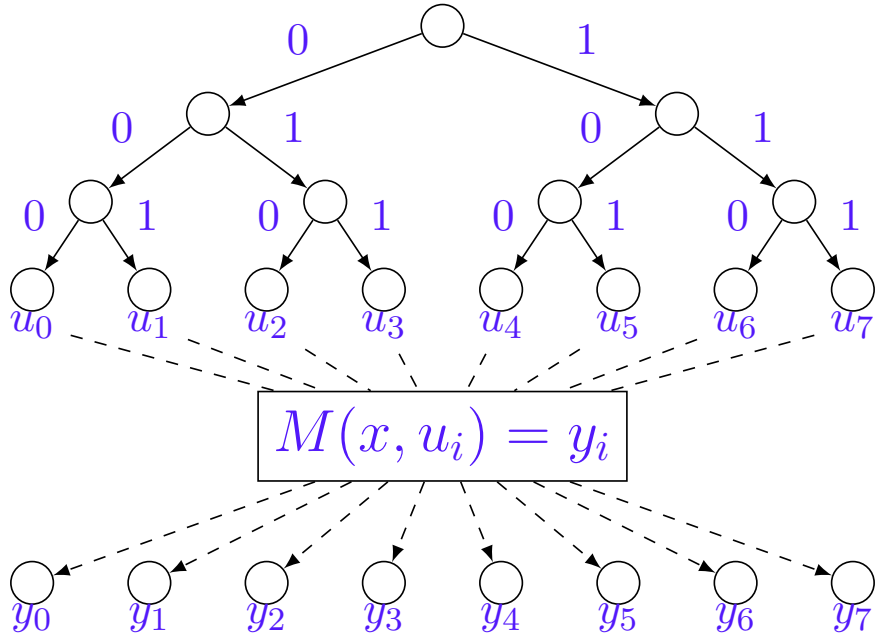
call fair_coin a sufficient number of times before running **M**

and pass the obtained bit string as an additional input to **M**.

▷ In our previous example:

First generate the $n-2$ random bits $y = y_1 y_2 \ldots y_{n-1}$ and replace

$x_i := \text{fair\_coin}()$ by $x_i := y_i$.



- Fixing all inputs (here x) of an randomized algorithm except for the coin tosses (here u) yields a tree of possible computations.
  - ▷ This defines a probability distribution over the possible outpus for the given inputs.
  - ▷ Randomized algorithms with fixed inputs are random variables/experiments.

1. Lecture 1 – Introduction: goals and locks

2. Lecture 2 – Principles and randomized computation

   Principles of (modern) cryptography

   Randomized computation

   Source of randomness

3. Lecture 3 – Perfect secrecy

4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

- Sources of randomness:
  - ▷ Physical processes which are by their mathematical model random like quantum phenomena:

    Photons travelling through a semi-transparent mirror, nuclear decay, shot noise
  - ▷ Other physical sources used:

    Thermal noise, clock drift
  - ▷ Other sources conjectured to yield randomness:

    Interrupts by keyboard or hard disk
  - ▷ Some free randomness sources:

    QRNG (Photons), Hotbits (Decay), random.org (Noise)
- Problem: These sources usually are biased, and the bias variates with time in general.

- In order to obtain random bit streams adequate for cryptography the bias has to be removed.
- These algorithms are called Randomness extractor.

▷ **Example**: "von Neumann extractor"

Assume a biased coin with constant bias $\Pr[C = 1] = p > \Pr[C = 0]$ and let $b_1 b_2 b_3 \ldots$ be a sequence of results by independent coin tosses.

For $i \geq 0$ take $b_{2i} b_{2i+1}$ and define $b'_i := 0$ if $b_{2i} b_{2i+1} = 01$; $b'_i := 1$ if $b_{2i} b_{2i+1} = 10$; and $b'_i := \varepsilon$ otherwise, e.g.:

$$01||00||10||00||11 \rightsquigarrow 0||\varepsilon||1||\varepsilon||\varepsilon = 01$$

▷ **Ex**: Show that in the resulting sequence bits are uniformly distributed.

- In practice, hash functions are used as randomness extractors, too.

- Cryptography crucially depends on the assumption that true randomness ($\mathrm{fair\_coin}()$) is available.
- For instance, C's rand() is not suitable for cryptography.
- Many OS provide better alternatives
  - Linux: /dev/random or Entroy Gathering Daemon
  - Windows (MSVC): CryptGenRandom resp. CNG (Cryptography API: Next Generation)

  but these are still inferior to a true randomness generator:
  - See [3] resp. [4] for a description and short comings of /dev/random resp. CryptGenRandom (Windows 2000).

- Essentially, both use internal OS statistics/timers/counters
  - e.g. cpu cycles, clock ticks, memory usage

  as sources for randomness, and use combinations of hash functions and ciphers to produce the actual output.
  - /dev/random uses SHA1
  - CryptGenRandom uses RC4 and SHA1.
- Another alternative: Fortuna [6] by Ferguson and Schneier
  - Has a better heuristic for combining OS statistics [2].
  - Also simpler description.
  - Based on AES-256 and SHA-256.
- We will assume in the following that we have access to true randomness (fair_coin()).

- Kerckhoff's principle/Shannon's maxim and why it is reasonable
- Sufficiently large key space is necessary but not sufficient
- Cryptographic primitives: block ciphers, hash funct., math problems
  - We have to conjecture that a primitive cannot be "broken"
  - Conjecture depends on our current knowledge & technical possibilities
- Provable security:
  - Assume "unbreakable" primitive, then show that "breaking" the scheme would yield a method to "break" the primitive
  - Need to define what "breaking" the scheme means, and what an attacker is allowed to do
- Randomized computation "=" standard programs plus $\mathrm{fair\_coin}()$
  - generating truly random primes/keys, testing primality

1. Lecture 1 – Introduction: goals and locks
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

1. Lecture 1 – Introduction: goals and locks
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
   - Perfect secrecy
   - One-time pad
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

- Assumption: Gen, Enc, Dec are randomized algorithms ($\mathrm{fair\_coin}()$) with finite running time, but not necessarily efficient.
  - They always terminate!
- Gen outputs a random key k chosen from a finite key space $\mathcal{K}$.
  - W.l.o.g. we may always assume $k \overset{u}{\in} \{0,1\}^n$ for some (large) n.
  - ▷ **Ex**: Proof this! Recall how to turn a randomized alg. into a deterministic one.
- Enc takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$,
  - the message space $\mathcal{M}$ is assumed to be finite (but arbitrarily large),

  and outputs a possibly random ciphertext $c \overset{r}{:=} \mathsf{Enc}_k(m)$.
  - $\mathcal{C} := \{\mathsf{Enc}_k(m) \mid k \in \mathcal{K}, m \in \mathcal{M}\}$ ciphertext space.
- Dec takes as input a ciphertext $c \in \mathcal{C}$ and a key $k \in \mathcal{K}$, and outputs a plaintext $\mathsf{Dec}_k(c)$ where $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$ for all $m \in \mathcal{M}, k \in \mathcal{K}$.
  - We assume that Dec is deterministic, i.e. not randomized.

- Standard setup: Alice&Bob, Eve, public channel
- Alice&Bob:
  - Have chosen an ES $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ (as on the previous slide).
  - Share a secret key k generated by Gen.
  - ▷ Hence, Alice&Bob can do randomized computations.
  - Exchange a single ciphertext $c := \mathsf{Enc}_k(m)$.

- Standard setup: Alice&Bob, Eve, public channel
- Eve: (Adversarial model)
  - Does not know the secret key used by Alice and Bob.
  - May only eavesdrop the c over the public channel.
  - May influence the choice of the messages.
  - Knows ("the source code of") the ES used by Alice&Bob.
  - Eve can also do randomized computations.
  - Eve's attack is a sequence of repeatable steps,
  - ▷ I.e. Eve's attack is a randomized algorithm $\mathcal{A}$.
  - ▷ In essence: Eve "=" $\mathcal{A}$.
  - ▷ We assume $\mathcal{A}$ terminates, but no further restrictions.

- How Eve can interact with Alice&Bob and the chosen ES:

  Presented as an experiment or game

  0. (Eve analyses $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and chooses $\mathcal{A}$ accordingly.)
  1. Eve chooses two admissible plaintexts $m_0, m_1 \in \mathcal{M}$ by running $\mathcal{A}$.
  2. Alice secretly chooses m uniformly at random from $\{m_0, m_1\}$, generates k using Gen, and finally sends $c \stackrel{r}{:=} \mathsf{Enc}_k(m)$ to Bob, i.e. Alice&Bob simply give c to Eve.
  3. Eve runs $\mathcal{A}$ on c to decide whether Alice chose m_0 or m_1.

- Eve's goal resp. what Eve must not be able to do:

  In this game Eve ($\mathcal{A}$) must not succeed with prob. better or worse than $1/2$,

  - i.e. Eve cannot do better than simply guessing,

- Note: As soon as all algorithms $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathcal{A}$ are chosen, the experiement becomes a randomized algorithm itself.

- **Definition**: An ES $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is prefectly secret if

  there is no attack (i.e. randomized algorithm) $\mathcal{A}$

  which wins with prob. strictly greater than $1/2$ in the following "eavesdropping game":

  1. $(m_0, m_1) \stackrel{r}{:=} \mathcal{A}.m()$;
  2. $k \stackrel{r}{:=} \mathsf{Gen}()$;
  3. $b \stackrel{r}{:=} \mathrm{fair\_coin}()$ (short: $b \stackrel{u}{\in} \{0,1\}$);
  4. $c \stackrel{r}{:=} \mathsf{Enc}_k(m_b)$;
  5. $r \stackrel{r}{:=} \mathcal{A}.r(c)$;

  ▷ $\mathrm{cout} << \text{"Eve"} << (b = r \ ? \ \text{"wins"} \ : \ \text{"loses"} \ )$;

▷ "Perfectly indistinguishable encryptions"

- **Ex**: Show that
  - Eve can always win with prob. exactly $1/2$.
  - Eve can win with probability strictly less than $1/2$ iff she can win with prob. strictly greater than $1/2$.
  - If $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is perfectly secret, there is no randomized algorithm which, on input a ciphertext $c \stackrel{r}{:=} \mathsf{Enc}_k(m)$, outputs the first bit of m with prob. strictly greater than $1/2$ (with $m \stackrel{u}{\in} \{0,1\}^n \subseteq \mathcal{M}$ chosen uniformly at random).

- The game specifies how Eve may interact with Alice&Bob:
- Eve knows the ES and the game, and can accordingly choose her algorithm (attack) $\mathcal{A}$ before the start of the game.

▷ As soon as the game starts, $\mathcal{A}$ (Eve's behaviour) is fixed.

▷ The behaviour (output) of $\mathcal{A}$ is completely determined by

(i) its own coin tosses ($\mathrm{fair\_coin}()$), and

(ii) the input ciphertext c.

- Once $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and $\mathcal{A}$ are fixed,

  we can simulate the game in several ways, e.g.:

$$\begin{array}{l} (m_0, m_1) \stackrel{r}{:=} \mathcal{A}.m() \\ k \stackrel{r}{:=} \mathsf{Gen}() \\ b \stackrel{u}{\in} \{0,1\} \\ c \stackrel{r}{:=} \mathsf{Enc}_k(m_b) \\ r \stackrel{r}{:=} \mathcal{A}.r(c) \end{array} \quad \left\| \quad \begin{array}{l} k \stackrel{r}{:=} \mathsf{Gen}() \\ (m_0, m_1) \stackrel{r}{:=} \mathcal{A}.m() \\ b \stackrel{u}{\in} \{0,1\} \\ c \stackrel{r}{:=} \mathsf{Enc}_k(m_b) \\ r \stackrel{r}{:=} \mathcal{A}.r(c) \end{array} \right\| \quad \begin{array}{l} b \stackrel{u}{\in} \{0,1\} \\ k \stackrel{r}{:=} \mathsf{Gen}() \\ (m_0, m_1) \stackrel{r}{:=} \mathcal{A}.m() \\ c \stackrel{r}{:=} \mathsf{Enc}_k(m_b) \\ r \stackrel{r}{:=} \mathcal{A}.r(c) \end{array}$$

  ▷ All programs give the same distribution of r as

  causally independent computations may be rearranged.

- **Remark**: The underlying probability space
  - The coin tosses made by the algorithms $(\mathrm{Gen}, \mathrm{Enc}, \mathcal{A})$ and by Alice and Bob (b) define a discrete probability space:

    If T is the maximal run time of the complete game, we need at most T random bits. Elementary events: $\{0,1\}^T$

  - ▷ Define random variables:

    B: Result of Alice and Bob's coin toss

    $R_{\mathcal{A}}$: Final reply by $\mathcal{A}$.

  - ▷ Then: $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ is perfectly secret iff

    $\Pr[R_{\mathcal{A}} = B] = 1/2$ for every randomized algorithm $\mathcal{A}$.

1. Lecture 1 – Introduction: goals and locks
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
   - Perfect secrecy
   - One-time pad
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

- Fix some finite alphabet Σ identified with $\mathbb{Z}_{|\Sigma|}$.
- Let $\oplus, \ominus$ be the addition/subtraction modulo $|\Sigma|$.
  - Extended to Σ^l by $a_1 \dots a_l \oplus b_1 \dots b_l := (a_1 \oplus b_1) || \dots || (a_l \oplus b_l)$
- **Definition**: (Σ, l)-one-time pad (OTP)
  - $\mathcal{M} = \mathcal{C} = \mathcal{K} = \Sigma^l$ ("fixed-length encryption scheme")
  - Gen generates uniformly at random a key $k = k_1 k_2 \dots k_l \overset{u}{\in} \Sigma^l$

    (See Venona project.)
  - $\mathrm{Enc}_k(m) := m \oplus k$.
  - $\mathrm{Dec}_k(c) := c \ominus k$.
- **Theorem**: The one-time pad is perfectly secrect.

(Requires only a random bit generator/fair coins.)

- **Ex**: Let $\langle G, \circ, 1 \rangle$ be a group. Define a one-time pad with $\Sigma = G$, and show that your "(G, l)-one-time pad" is perfectly secure.

▷ **Ex**: Show for any given $m, c \in \Sigma^l$, but random key $k \overset{u}{\in} \Sigma^l$

$$\Pr[m \oplus k = c] = \frac{1}{|\Sigma|^l}.$$

In words: No matter what m is, by adding ($\oplus$) a truly random k onto m the resulting string is uniformly distributed over all possible strings.

- That is, any $\mathcal{A}$ can only win with prob. exactly $1/2$ vs the OTP:

▷ **Proof**

$(m_0, m_1) \stackrel{r}{:=} \mathcal{A}().m()$	$\|\|$	$\|\|$
$k \stackrel{u}{\in} \Sigma^l$	$\|\|$	$\|\|$
$b \stackrel{u}{\in} \{0,1\}$	$\|\|$	$\|\|$
$c \stackrel{r}{:=} m_b \oplus k$	$\|\|$	$\|\|$
$r \stackrel{r}{:=} \mathcal{A}.r(c)$	$\|\|$	$\|\|$
$r \stackrel{?}{=} b$	$\|\|$	$\|\|$

▷ Left: The original eavesdropping game.

- That is, any $\mathcal{A}$ can only win with prob. exactly $1/2$ vs the OTP:

▷ **Proof**

$(m_0, m_1) \stackrel{r}{:=} \mathcal{A}().m()$	$(m_0, m_1) \stackrel{r}{:=} \mathcal{A}.m()$
$k \stackrel{u}{\in} \Sigma^l$	$k \stackrel{u}{\in} \Sigma^l$
$b \stackrel{u}{\in} \{0,1\}$	$b \stackrel{u}{\in} \{0,1\}$
$c \stackrel{r}{:=} m_b \oplus k$	$c \stackrel{u}{\in} \Sigma^l$
$r \stackrel{r}{:=} \mathcal{A}.r(c)$	$r \stackrel{r}{:=} \mathcal{A}.r(c)$
$r \stackrel{?}{=} b$	$r \stackrel{?}{=} b$

▷ Left: The original eavesdropping game.

▷ Middle: Whatever the value of m_b,

$c = \mathsf{Enc}_k(m) = m \oplus k$ is uniformly distributed over Σ^l as $k \stackrel{u}{\in} \Sigma^l$

- That is, any $\mathcal{A}$ can only win with prob. exactly $1/2$ vs the OTP:

▷ **Proof**

$(m_0, m_1) \stackrel{r}{:=} \mathcal{A}().m()$	$(m_0, m_1) \stackrel{r}{:=} \mathcal{A}.m()$	~~$(m_0, m_1) \stackrel{r}{:=} \mathcal{A}.m()$~~
$k \stackrel{u}{\in} \Sigma^l$	$k \stackrel{u}{\in} \Sigma^l$	~~$k \stackrel{u}{\in} \Sigma^l$~~
$b \stackrel{u}{\in} \{0,1\}$	$b \stackrel{u}{\in} \{0,1\}$	$c \stackrel{u}{\in} \Sigma^l$
$c \stackrel{r}{:=} m_b \oplus k$	$c \stackrel{u}{\in} \Sigma^l$	$r \stackrel{r}{:=} \mathcal{A}.r(c)$
$r \stackrel{r}{:=} \mathcal{A}.r(c)$	$r \stackrel{r}{:=} \mathcal{A}.r(c)$	$b \stackrel{u}{\in} \{0,1\}$
$r \stackrel{?}{=} b$	$r \stackrel{?}{=} b$	$r \stackrel{?}{=} b$

▷ Left: The original eavesdropping game.

▷ Middle: Whatever the value of m_b,

$c = \mathsf{Enc}_k(m) = m \oplus k$ is uniformly distributed over Σ^l as $k \stackrel{u}{\in} \Sigma^l$

▷ Right: Get rid of k and m_0, m_1, not relevant for $r \stackrel{?}{=} b$; move choice of b to the end

- ▷ What we have just done is:
- ▷ We have constructed from an attack $\mathcal{A}$ for the game,
- ▷ an algorithm $\mathcal{P}$

$\mathcal{P}$:

$c \overset{u}{\in} \Sigma^l$
$r \overset{r}{:=} \mathcal{A}(c)$
return r

which predicts the value of b before it has been fixed!

- As $b \overset{u}{\in} \{0,1\}$ (i.e. $b := \text{fair\_coin}()$), the probability that b "hits" the output of $\mathcal{P}$ is always $1/2$.

Proof: OTP is prefectly secret*

- For comparison an alternative "standard" proof:
- Fix any (admissible) attack $\mathcal{A}$.
- Define the following random variables according to the eavesdropping game:
  - M_0, M_1: Plaintexts output by $\mathcal{A}$ in the first step.
  - B: Result of AliceBob's coin toss with $\Pr[B = 0] = \Pr[B = 1] = 1/2$.
  - K: Key generated by Gen ($K := \mathsf{Gen}$).
  - C: Ciphertext given to Eve, i.e. $C := \mathsf{Enc}_K(M_B) = K \oplus M_B$.
  - $R_{\mathcal{A}}$: Final output by $\mathcal{A}$ in the third step, i.e. $R_{\mathcal{A}} := \mathcal{A}(C) = \mathcal{A}(K \oplus M_B)$.
- Note by definition of the algorithms used in the game:
  - B is independent of M_0, M_1, K (but it influences C and thus $R_{\mathcal{A}}$).
  - K is independent of M_0, M_1, B (but again it influences C and $R_{\mathcal{A}}$).

- Formally, use the formula of total probability for all possible values of M_0, M_1 and that the choice of K is independent of M_0, M_1, B:

$$\begin{aligned}
& \Pr[C = c, B = b] \\
= & \sum_{m_0,m_1} \Pr[M_B \oplus K = c, M_0 = m_0, M_1 = m_1, B = b] \\
= & \sum_{m_0,m_1} \Pr[K = c \ominus m_b, M_0 = m_0, M_1 = m_1, B = b] \\
= & \sum_{m_0,m_1} \Pr[K = c \ominus m_b] \cdot \Pr[M_0 = m_0, M_1 = m_1, B = b] \\
= & \frac{1}{|\Sigma|^l} \cdot \sum_{m_0,m_1} \Pr[M_0 = m_0, M_1 = m_1, B = b] \\
= & \frac{1}{|\Sigma|^l} \cdot \Pr[B = b]
\end{aligned}$$

  (where $m_0, m_1 \in \Sigma^l$)

- **Ex**: Show that $\Pr[C = c] = \frac{1}{|\Sigma|^l}$. Conclude that C and B are independent.

- Use the formula of total probability for all values of B and C:

$$\begin{aligned} & \Pr[R_{\mathcal{A}} = B] \\ = & \sum_{b,c} \Pr[R_{\mathcal{A}} = B, B = b, C = c] \\ = & \sum_{b,c} \Pr[\mathcal{A}(C) = B, B = b, C = c] \\ = & \sum_{b,c} \Pr[\mathcal{A}(c) = b, B = b, C = c] \end{aligned}$$

- What is $\Pr[\mathcal{A}(c) = b, C = c, B = b]$?

  (That is, what is the probability that for fixed concrete values b, c, the attack $\mathcal{A}$ on input c outputs b, and Alice&Bob get in their coin toss b, and the resulting ciphertext is c.)

Proof: OTP is prefectly secret*

- Recall that $\mathcal{A}$ is a randomized algorithm.
- For a fixed ciphertext c, the computation $\mathcal{A}(c)$ only dependes on the internal coin tosses made by $\mathcal{A}$.
- By definition of randomized computation, these coin tosses are independent of all other coin tosses made by the other algorithms.
- Thus, the event that $\mathcal{A}(c)$ outputs the concrete value b is independent of the event that "$C = c$ and $B = b$":

$$\Pr[\mathcal{A}(c) = b, B = b, C = c] = \Pr[\mathcal{A}(c) = b] \cdot \Pr[B = b, C = c].$$

- As we have already seen that C is independent of B:

$$\begin{aligned} & \Pr[R_{\mathcal{A}} = B] \\ = \; & \textstyle\sum_{b,c} \Pr[\mathcal{A}(c) = b] \cdot \Pr[C = c] \cdot \Pr[B = b] \\ = \; & \tfrac{1}{2} \textstyle\sum_{c} \Pr[C = c] \left(\Pr[\mathcal{A}(c) = 0] + \Pr[\mathcal{A}(c) = 1]\right) \\ = \; & 1/2. \end{aligned}$$

- Shannon gave several (equivalent) characterizations of perfect secrecy.
- ▷ For instance:

  $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is perfectly secret if

  $\Pr[\mathsf{Enc}_K(m_0) = c] = \Pr[\mathsf{Enc}_K(m_1) = c]$ for all $m_0, m_1 \in \mathcal{M}, c \in \mathcal{C}$.
- **Ex**: Show that our definition of perfect secrecy is equivalent to the above one.
  - Property is sufficient: Have a look at the proof for the OTP.
  - Property is necessary: Give an attack otherwise.

- The OTP is impractical for most applications, as the key is just as long as the message.
- ▷ In order to encrypt 4GB of data Alice&Bob first have to meet and secretly exchange a secret of key of size 4GB.
- ▷ In fact, any perfectly secret ES has this disadvantages:

  Lemma: An ES with $|\mathcal{K}| < |\mathcal{M}|$ cannot be perfectly secret.

  Proof: **Ex**.

  - For any c, consider all possible decryptions $D_c = \{\mathsf{Dec}_k(c) \mid k \in \mathcal{K}\}$.
  - Argue that $|D_c| \leq |\mathcal{K}|$ and use the alternative definition of perfect secercy.
- ▷ For "computationally secret" ES with $|\mathcal{K}| \ll |\mathcal{M}|$, we will need to restrict the computational power of Eve.
  - E.g. she should not be able to compute D_c.

- Alternative presentation used in literature uses oracles.
- **Definition**: oracle access

  Let $f : \{0,1\}^* \to \{0,1\}^*$ be some function.

  Then $\mathbf{M}^f$ denotes an algorithm **M** with oracle access to f:

  $\mathbf{M}^f$ may call a black-box procedure, the oracle, to compute the value $f(x)$ within a single time step.

  - Black box: We can only sent queries to it and read its answer
  - We cannot break the black box open and look inside.
  - Although the oracle returns the value $f(x)$ within a single time step,

    we still take into account the time that the algorithm **M** needs to process the output $f(x)$ itself.
  - "**P** + SAT-oracle = **NP**"

Remark: Oracle view of perfect secrecy

- Oracle view of the "eavesdropping game"

  1. Alice and Bob set up two oracles (black boxes) $\mathcal{O}_0$ and $\mathcal{O}_1$,
     - Behaviour of oracle $\mathcal{O}_i$:
     - ▷ Initialization: Run Gen and store the obtained key k.
     - ▷ Query: On input $m_0, m_1 \in \mathcal{M}$, return $\mathrm{Enc}_k(m_i)$.

     flip a fair coin $b \overset{u}{\in} \{0,1\}$, and give $\mathcal{O}_b$ to Eve in a black box $\mathcal{O}$.
  2. Eve now runs $\mathcal{A}^{\mathcal{O}}$ to obtain the reply r
     - Restriction: $\mathcal{A}^{\mathcal{O}}$ must not query $\mathcal{O}$ more than once.

  ▷ Eve wins if $\mathcal{A}^{\mathcal{O}}$ correctly identifies the oracle, i.e. $r = b$.

▷ **Ex**: The ES $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ is perfectly secret iff

$$\underbrace{\left|\Pr\left[\mathcal{A}^{\mathcal{O}_1} = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_0} = 1\right]\right|}_{\text{"Advantage of } \mathcal{A}\text{"}} = 0.$$

- **Definition**: "Perfect secrecy w.r.t. q encryptions"

  An ES $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is prefectly secret under q encryptions if

  there is no randomized algorithm $\mathcal{A}$ which wins with prob. strictly greater than $1/2$ in the following game:

  1. Eve runs $\mathcal{A}$ to obtain message sequences

     $$\vec{m}_0 = ({m_0}^{(1)}, \ldots, {m_0}^{(q)}), \vec{m}_1 = ({m_1}^{(1)}, \ldots, {m_1}^{(q)}),$$

     and gives these messages to Alice&Bob (${m_i}^{(j)} \in \mathcal{M}$).

  2. In secrecy, Alice&Bob run Gen to obtain the key k, toss a fair coin to obtain $b \overset{u}{\in} \{0,1\}$, and consecutively compute $c^{(1)} = \mathsf{Enc}_k({m_b}^{(1)})$ to $c^{(q)} = \mathsf{Enc}_k({m_b}^{(q)})$

     Finally, they give $c = (c^{(1)}, \ldots, c^{(q)})$ to Eve.

  3. Eve continues the computation of $\mathcal{A}$ with the additional input $\vec{c}$; eventually $\mathcal{A}$ outputs a reply $r \in \{0,1\}$.

  ▷ Eve wins if $b = r$.

- **Remark**: Frequency analysis for the one-time pad and $q = 2$:

$$c_1 \ominus c_2 = (m_1 \oplus k) \ominus (m_2 \oplus k) = m_1 \ominus m_2.$$

  Statistics on the frequency of $a \ominus b$ for $a, b \in \Sigma$ can also be used to recover m_1, m_2.

- **Ex**:

▷ Show that the one-time pad is not perfectly secret under $q = 2$ encryptions: Find an $\mathcal{A}$ which recovers k with prob. 1.

▷ Design an ES which is perfectly secret under q encryptions.

  Prove its security.

1. Lecture 1 – Introduction: goals and locks
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds

- Recap perfect secrecy:
  - ES is perfectly secret iff Eve can win the "eavesdropping game" only with prob. $1/2$.
  - Eve was given a single ciphtertext c.
  - Eve's attack was randomized algorithm without time or space bounds (she had to terminate though).
  - ▷ $|\mathcal{K}| \geq |\mathcal{M}|$ is necessary:

    If $|\mathcal{K}| < |\mathcal{M}|$, then for every $m \in \mathcal{M}, k \in \mathcal{K}$ and $c := \mathsf{Enc}_k(m)$ there is some $m' \notin D_c := \{\mathsf{Dec}_{k'}(c) \mid k' \in \mathcal{K}\}$. And (unbounded) Eve can find m, m' by simple brute-force computation of D_c.
- For a pratical ES we want keys much shorter than the actual messages: $|\mathcal{K}| \ll |\mathcal{M}|$ ($\ll$: much less than)

▷ Need to consider a weaker notion of secrecy where Eve is not capable of computing D_c. See next slide.

- We want to adapt the definition of perfect secrecy so that it is also satisfiable also for an ES with e.g. $\mathcal{K} = \{0,1\}^n$ and $\mathcal{M} = \{0,1\}^{3n}$.
- Let $c \in \mathcal{C}$ be some ciphertext and $D_c := \{\mathsf{Dec}_k(c) \mid k \in \mathcal{K}\}$.
  - ▷ Note $\frac{|D_c|}{|\mathcal{M}|} \leq \frac{2^n}{2^{3n}} = 2^{-2n}$, i.e. for $n \geq 64$ D_c is a tiny fraction of $\mathcal{M}$.
  - ▷ By computing D_c Eve now can rule out the majority of plaintexts.
    - In contrast to the OTP where we have $D_c = \mathcal{M} = \mathcal{K}$ for any $c \in \mathcal{C}$
  - ▷ Computing all of D_c requires to run $\mathsf{Dec}(c)$ for all possible keys in $\mathcal{K} = \{0,1\}^n$, i.e. requires time exponential in n.
  - ▷ Recall (see sufficient keyspace principle): For $n = 128$ testing all 2^{128} keys can be considered an impractical attack (at least for now).
  - ▷ First restriction: We will only consider practical attacks a threat from now on, where pratical amounts to efficient (e.g. PTIME) computations. I.e. any attack that is not an efficient algorithm will be disregarded. (More later.)

- Is this the only thing we have to change in the definition of perfect secrecy?
- ▷ No, Eve can still do better than $1/2$ when guessing b:

  Intuitively, as $|\mathcal{K}| \ll |\mathcal{M}|$ it is easier to guess the key and decrypt than to guess the message; so the chances improve from $\frac{1}{|\mathcal{M}|}$ to $\frac{1}{|\mathcal{K}|}$.
- ▷ See the next slides for a proof.
- ▷ Second restriction: Besides restricting the running time, we also will allow Eve to win with a probability "negligibly" better than $1/2$.
- Doing so, we will arrive with the definition of computational secrecy.
- Remaining question: Is there a computational secret ES?
- ▷ Perhaps!? See the following lectures.

- Why we cannot require that Eve only wins with prob. exaclty $1/2$:
- Consider "perfect secrecy" + "only efficient attacks".
  - ▷ Choose any ES with $\mathcal{K} = \{0,1\}^n$ and $\mathcal{M} = \{0,1\}^{3n}$.
    - Recall: wlog. Gen chooses $k \stackrel{u}{\in} \{0,1\}^n$.
  - ▷ Eve's attack $\mathcal{A}$:
    - $\mathcal{A}.m()$: output $m_0, m_1 \stackrel{u}{\in} \{0,1\}^{3n}$ (m_0, m_1 independent of each other).
    - $\mathcal{A}.r(c)$: choose $k' \stackrel{u}{\in} \{0,1\}^n$;

      if $\mathsf{Dec}_{k'}(c) = m_1$ return 1;

      else return $r \stackrel{u}{\in} \{0,1\}$.
- Running time of this $\mathcal{A}$: approx. $7n + T_{\mathsf{Enc}} + 2$, i.e. roughly as efficient as Enc.

- Eve's attack in the eavesdropping game:

$$
\begin{array}{l}
m_0, m_1 \stackrel{u}{\in} \{0,1\}^{3n} \\
k \stackrel{u}{\in} \{0,1\}^{n} \\
b \stackrel{u}{\in} \{0,1\} \\
c \stackrel{r}{:=} \mathsf{Enc}_k(m_b) \\
k' \stackrel{u}{\in} \{0,1\}^{n} \\
\text{if } \mathsf{Dec}_{k'}(c) = m_1\text{: } r := 1 \\
\text{else: } r \stackrel{u}{\in} \{0,1\} \\
\hline
\text{Eve wins iff } r = b
\end{array}
$$

- Probability of $\mathcal{A}$ to win in the game of perfect secrecy if $b = 1$:

$m_0, m_1 \overset{u}{\in} \{0,1\}^{3n}$
$k \overset{u}{\in} \{0,1\}^n$
~~$b \overset{u}{\in} \{0,1\}$~~ $b := 1$
$c \overset{r}{:=} \mathsf{Enc}_k(m_1)$
$k' \overset{u}{\in} \{0,1\}^n$
if $\mathsf{Dec}_{k'}(c) = m_1$: $r := 1$
else: $r \overset{u}{\in} \{0,1\}$

Eve wins iff $r = 1$

  - ▷ Eve wins with probability (as the value of r is independently chosen):

$$\Pr[\mathsf{Dec}_{k'}(c) = m_1] + \Pr[\mathsf{Dec}_{k'}(c) \neq m_1]\,\frac{1}{2} = \frac{1}{2} + \frac{1}{2} \cdot \Pr[\mathsf{Dec}_{k'}(c) = m_1]$$

  - ▷ We have $\mathsf{Dec}_{k'}(c) = m_1$ at least for $k' = k$, so:

$$\Pr[\mathsf{Dec}_{k'}(c) = m_1] \geq \Pr\left[k = k'\right] = \frac{1}{|\mathcal{K}|} = 2^{-n}$$

- Probability of $\mathcal{A}$ to win in the game of perfect secrecy if $b = 0$:

$m_0, m_1 \overset{u}{\in} \{0,1\}^{3n}$
$k \overset{u}{\in} \{0,1\}^n$
~~$b \overset{u}{\in} \{0,1\}$~~ $b := 0$
$c \overset{r}{:=} \mathsf{Enc}_k(m_0)$
$k' \overset{u}{\in} \{0,1\}^n$
if $\mathsf{Dec}_{k'}(c) = m_1$: $r := 1$
else: $r \overset{u}{\in} \{0,1\}$

Eve wins iff $r = 0$

  - ▷ Eve wins with probability (as the value of r is independently chosen):

$$\Pr[\mathsf{Dec}_{k'}(c) \neq m_1]\,\frac{1}{2} = \frac{1}{2} - \frac{1}{2}\Pr[\mathsf{Dec}_{k'}(c) = m_1]$$

  - ▷ Note: we can only decrypt c to m_1 if $m_1 \in D_c$.

    As m_1 is chosen independently of m_0 and, thus, c:

$$\Pr[\mathsf{Dec}_{k'}(c) = m_1] \leq \Pr[m_1 \in D_c] \leq \frac{|D_c|}{|\mathcal{M}|} \leq \frac{|\mathcal{K}|}{|\mathcal{M}|} \leq 2^{-2n}.$$

- Probability of $\mathcal{A}$ to win in the game of perfect secrecy:
- Both cases $b = 1$ and $b = 0$ happen with prob. $1/2$.
- So, $\mathcal{A}$ wins at least with prob.:

$$\begin{aligned}\Pr[\mathcal{A}.r(c) = b] &\geq \tfrac{1}{2} \cdot (\tfrac{1}{2} + \tfrac{1}{2} \cdot 2^{-n}) + \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot (1 - 2^{-2n}) \\ &= 1/2 + 1/2 \cdot 2^{-n} \cdot (1 - 2^{-n}) \\ &= 1/2 + 2^{-(n+1)} - 2^{-(2n+1)} \\ &> 1/2.\end{aligned}$$

▷ **Remark**: Functions like $\varepsilon(n) := 2^{-(n+1)} - 2^{-(2n+1)}$ will be called negligible – they drop to 0 faster than any polynomial fraction n^{-k} (for any constant $k \in \mathbb{N}$),

i.e. making n sufficiently large, Eve's advantage of $\varepsilon(n)$ becomes close 0 s.t. it can be considered negligible.

1. Lecture 1 – Introduction: goals and locks
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds
    - Notions of efficient computation
    - Negligible probability

- How to define "efficient"?
- In complexity theory: polynomial-time algorithm

  Algorithm **M** runs in polynomial time if there is a polynomial $p_{\mathbf{M}}$ such that:

  The computation $\mathbf{M}(x)$ stops after at most $p_{\mathbf{M}}(|x|)$ time steps.

▷ **Reminder**:

- $x \in \{0,1\}^*$ is some binary input and $|x|$ is the number of bits x consists of, i.e. the length of x.
- ▷ Example: Sorting an array of n integers each stored in m bits.

  Input size: $n \cdot m$; merge sort: $C_0 + C_1 \cdot n \cdot m \cdot \log_2 m \in \mathcal{O}((nm)^2)$.
- $\mathbf{P}$: the class of all decision problems (true or false) for which there is a polynomial-time algorithm.
- $p_{\mathbf{M}}$ might have arbitrarily large constants/coefficients (e.g. C_0, C_1)!

- In cryptography two slightly different approaches are used:
- Assume B is a block cipher of block length l and key length n.

  Task: Given $x, y \in \{0,1\}^{2l}$ try to find $k \in \{0,1\}^n$ s.t. $B_k(x) = y$.

- In cryptography two slightly different approaches are used:
- Assume B is a block cipher of block length l and key length n.

  Task: Given $x, y \in \{0,1\}^{2l}$ try to find $k \in \{0,1\}^n$ s.t. $B_k(x) = y$.

  - Not really interesting to analyse an attack for finding a key for inputs of length different from $2l$ as l is fixed by B.
  - ▷ The asymptotic running time for growing input length is not as useful as in the case of sorting.

- In cryptography two slightly different approaches are used:
- Assume B is a block cipher of block length l and key length n.

  Task: Given $x, y \in \{0,1\}^{2l}$ try to find $k \in \{0,1\}^n$ s.t. $B_k(x) = y$.

  - Not really interesting to analyse an attack for finding a key for inputs of length different from $2l$ as l is fixed by B.
  - ▷ The asymptotic running time for growing input length is not as useful as in the case of sorting.
  - Much more interesting: concrete bounds on the running time and the success probability of the attack.

    E.g. Attack will do exactly t computation steps and succeed (i.e. find a k) with probability $t \cdot 2^{-n}$.
  - ▷ Allows to assess how severe the attack is w.r.t. a concrete computation device.

- In cryptography two slightly different approaches are used:
- Assume B is a block cipher of block length l and key length n.

  Task: Given $x, y \in \{0,1\}^{2l}$ try to find $k \in \{0,1\}^n$ s.t. $B_k(x) = y$.

  - Not really interesting to analyse an attack for finding a key for inputs of length different from $2l$ as l is fixed by B.
  - ▷ The asymptotic running time for growing input length is not as useful as in the case of sorting.
  - Much more interesting: concrete bounds on the running time and the success probability of the attack.

    E.g. Attack will do exactly t computation steps and succeed (i.e. find a k) with probability $t \cdot 2^{-n}$.
  - ▷ Allows to assess how severe the attack is w.r.t. a concrete computation device.
  - ▷ Downside: How decide what concrete device the adversary is using?

Theorem 1 [Authenticity] *Fix OCB parameters n and τ. Let A be an adversary that asks q queries and then makes its forgery attempt. Suppose the q queries have aggregate length of σ blocks, and the adversary's forgery attempt has at most c blocks. Let $\bar{\sigma} = \sigma + 2q + 5c + 11$. Then*

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{OCB}[\mathrm{Perm}(n),\tau]}(A) \leq \frac{1.5\,\bar{\sigma}^2}{2^n} + \frac{1}{2^\tau}$$

(Reference [10])

- In cryptography two slightly different approaches are used:
- Let B be a block cipher of block length l and key length n.

  Set $\mathsf{Enc}_k(m) := \rho || m \oplus B_k(\rho)$ with $\rho \overset{u}{\in} \{0,1\}^l$.

  Task: Assess the security of Enc w.r.t. B.

- In cryptography two slightly different approaches are used:
- Let B be a block cipher of block length l and key length n.

  Set $\mathsf{Enc}_k(m) := \rho || m \oplus B_k(\rho)$ with $\rho \overset{u}{\in} \{0,1\}^l$.

  Task: Assess the security of Enc w.r.t. B.

  - For the construction of Enc the block length l and key length n are now parameters which are only fixed after we have chosen a block cipher.
  - Here, asymptotic bounds become useful again:

    How does the running time and the success probability of an attack behave when we change l, n? E.g. when we replace DES ($n = 56, l = 64$) by AES ($n \in \{128, 196, 256\}, l = 128$).

  ▷ Convenient: No need to fix the properties of Eve's computer.

- In cryptography two slightly different approaches are used:
- Let B be a block cipher of block length l and key length n.

  Set $\mathsf{Enc}_k(m) := \rho || m \oplus B_k(\rho)$ with $\rho \overset{u}{\in} \{0,1\}^l$.

  Task: Assess the security of Enc w.r.t. B.

  - For the construction of Enc the block length l and key length n are now parameters which are only fixed after we have chosen a block cipher.
  - Here, asymptotic bounds become useful again:

    How does the running time and the success probability of an attack behave when we change l, n? E.g. when we replace DES ($n = 56, l = 64$) by AES ($n \in \{128, 196, 256\}, l = 128$).

  ▷ Convenient: No need to fix the properties of Eve's computer.

  ▷ Downside: Asymptotics hide alot of detail (how small is $\mathcal{O}(2^{-n})$ for a specific n?), to really assess security need to use concrete bounds.

- We will use most of the time asymptotic definitions and bounds.
  - Less notation, more convenient.
  - Essentially, proofs and ideas are the same in both settings.
  - Sufficent for understanding cryptographic schemes, and primitives.
  - But not the right thing e.g. to talk about specific block ciphers like AES.
- If you want to see concrete bounds:
  - You can find them inside of the proofs we discuss in the lecture.
  - See Lecture notes by Bellare and Rogaway [1].
- For more detailed discussion of asymptotic vs. concrete bounds, see sections 3.1.1 und 3.1.2 of [7].

- **Assumption**:

  Alice, Bob, Eve use only probabilistic polynomial time (PPT)

  ▷ **Notation**: DPT for deterministic polynomial time.

- i.e. cryptographic schemes and attacks are randomized algorithms with polynomial time bounds w.r.t. their inputs.
- For technical reasons, Eve's attack $\mathcal{A}$ (sometimes) gets 1^n as additional input.
  - The so-called security parameter n is usually simply the key length.
  - This is to allow $\mathcal{A}$ to run in time polynomial in n.

▷ A PPT-attack $\mathcal{A}$ can asymptotically only

- generate messages of polynomial length w.r.t. n;
- interact with ("query") Alice&Bob at most a polynomial time w.r.t. n (e.g. eavesdrop a polynomial number of ciphertexts).

▷ Alternative to "PPT-Eve": nonuniform polynomial time.

1. Lecture 1 – Introduction: goals and locks
2. Lecture 2 – Principles and randomized computation
3. Lecture 3 – Perfect secrecy
4. Lecture 4 – From perfect to computation secrecy, asymptotic vs. concrete bounds
    - Notions of efficient computation
    - Negligible probability

- Recall that for $|\mathcal{K}| < |\mathcal{M}|$, Eve's probability of guessing the original plaintext improves from $|\mathcal{M}|^{-1}$ to $\geq |\mathcal{K}|^{-1}$.
  - ▷ In our example: from 2^{-3n} to 2^{-n}.
  - ▷ I.e. Eve has an advantage of $2^{-n} - 2^{-3n}$ to guess the plaintext.
- Computational secrecy thus allows Eve to have a negligible "advantage" ε compared to the "perfect" setting.
- But how to fix when this advantage is so small that is does not matter, i.e. when is the advantage negligible?

- When is a probability negligible?
  - ▷ Again: either fix a concrete value or give an asymptotic definition.
  - ▷ E.g. concrete bounds for side-effects of pharmaceuticals:
    - very rare: $< 10^{-4}$
    - rare: $< 10^{-3}$
    - on occasion: $< 10^{-2}$
    - often: $< 10^{-1}$
    - very often: $> 10^{-1}$
- Compare this to:
  - Jackpot in the German lottery: $< 10^{-9}$

- Asymptotic notion: Probability decreases "quickly" when increasing security parameter (key length) n.
  - ▷ Let the user choose n w.r.t. concrete bound he deems right.
- **Definition**: $\varepsilon(n)$ is negligible if

  for every $c \in \mathbb{N}$ there is an $N \in \mathbb{N}$ s.t. $\forall n \geq N : \varepsilon(n) < n^{-c}$.
- Some examples: $2^{-n}, \quad 2^{-\sqrt{n}}, \quad 2^{-(\log n)^2}$.
- Motivation: If $\mathcal{A}$ is a PPT-attack, than repeating this attack a polynomial number of times, yields again a polynomial attack $\mathcal{A}'$.

  If $\mathcal{A}$ only succeeds with negligible advantage, so thus $\mathcal{A}'$ (**Ex**).
- **Ex**: Show that for $\varepsilon_1(n), \varepsilon_2(n)$ negligible and $p(n)$ a polynomial, the following functions are also negligible: $\varepsilon_1(n) + \varepsilon_2(n), \quad p(n) \cdot \varepsilon_1(n)$.
  - ▷ **Ex**: But if $\varepsilon_1(n), \varepsilon_2(n), \ldots$ all are negiligible and $q(n)$ is a polynomial, $\sum_{i=1}^{q(n)} \varepsilon_i(n)$ does not need to be negligible anymore!

- From now on, we will only consider attacks a threat which
  - run in polynomial time w.r.t. the key length (security parameter) n, and
  - whose success probability is non-negligibly higher than simply guessing the correct value (like the chosen message, or the secret key).
- First example:

  Why polynomial-time attacks with negligible success probability/advantage are no problem (if you can increase the key length).
- Second example:

  Why superpolynomial-time attacks are no problem (if you can arbitrarily increase the key length).

- For simplicity, assume encrypting and decrypting takes time $10^6 \cdot n^2$ (measured in cpu cycles, for instance), while the attack $\mathcal{A}$ runs in time $10^8 \cdot n^4$ in order to succeed with prob. $2^{20} \cdot 2^{-n}$.

  Assuming a 1GHz cpu (i.e., 10^9 cycles per second) and a key length of $n = 50$, we obtain 2.5s for en/decryption, while the attack takes rougly one week in order to succeed with prob. 2^{-30}.

  Moving to a 16GHz cpu, these times decrease by a factor of sixteen. So, even if we the double the key length by moving from 50 bit keys to 100 bit keys, en/decryption again still runs four times faster, while the attack still requires roughly one week but now only succeeds with prob. 2^{-80}, i.e., the attack has become worse.

	en/decrypt	attack
$n = 50$ on 1GHz	$2.5s$	$\approx 7d$ with prob. 2^{-30}
$n = 100$ on 16GHz	$0.675s$	$\approx 7d$ with prob. 2^{-80}

- PKES based e.g. on the RSA problem can be broken by factoring a large intereger N
  - where $N = p \cdot q$ with p, q primes of length n the security parameter.

The best (classical) algorithm known for factoring integers runs in time $C \cdot e^{c \cdot n^{1/3} \cdot (\log n)^{2/3}}$ (for some constant $c > 1$), i.e., super-polynomial in n.

For this example, assume that $c = \log 2$ and $C = 10^8$ and that en/decryption still takes $10^6 \cdot n^2$ as in the previous example.

	en/decrypt	factoring
$n = 500$ on 1GHz	≈ 4.1min	$\approx 138d$
$n = 1000$ on 16GHz	≈ 1.0min	$\approx 16.4y$

Currently, ≥ 2048-bit primes are recommended for RSA by the NIST for security up to 2030 (see e.g. `www.keylength.com`).

[1] M. Bellare. *Lecture notes*. URL: http://cseweb.ucsd.edu/users/mihir/crypto-lecnotes.html.

[2] Y. Dodis et al. *How to Eat Your Entropy and Have It Too – Optimal Recovery Strategies for Compromised RNGs*. URL: http://eprint.iacr.org/2014/167.pdf.

[3] Y. Dodis et al. *Security Analysis of Pseudo-Random Number Generators with Input: /dev/random is not Robust*. URL: http://eprint.iacr.org/2013/338.pdf.

[4] L. Dorrendorf, Z. Gutterman, and B. Pinkas. *Cryptanalysis of the Random Number Generator of the Windows Operating System*. URL: http://eprint.iacr.org/2007/419.pdf.

[5] N. Ferguson and B. Schneier. *Practical cryptography*.

[6] N. Ferguson, B. Schneier, and T. Kohno. *Cryptography Engineering – Chapter 9*. URL: https://www.schneier.com/fortuna.pdf.

[7] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*.

[8] A. Menezes and P. van Oorschot. *Handbook of Applied Cryptography*. URL: http://www.cacr.math.uwaterloo.ca/hac/.

[9] R. Rivest. *Handbook of theoretical computer science*.

[10] P. Rogaway. *Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.9626&rep=rep1&type=pdf.