A High-Level Overview of Cryptography

Daniel Bosk

School of Computer Science and Communication,
KTH Royal Institute of Technology, Stockholm

Department of Information and Communication Systems,
Mid Sweden University, Sundsvall

1st May 2017

- The word has its origin in greek[1]:

  κρυπτός (*kryptos*) meaning hidden[2].

  γράφος (*graphos*) meaning writing[3].

- The area has been around for ages.
- We should not confuse it with *steganography*.
- Steganography concerns hiding a message's *existence*.
- Cryptography concerns hiding a message's *contents*.

[1] 'cryptography, n.' In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/45374?redirectedFrom=cryptography&.

[2] 'crypto-, comb. form'. In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/45363.

[3] 'graphy-, comb. form'. In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/80855.

- The word has its origin in greek[1]:
  κρυπτός (*kryptos*) meaning hidden[2].
  γράφος (*graphos*) meaning writing[3].
- The area has been around for ages.
- We should not confuse it with *steganography*.
- Steganography concerns hiding a message's *existence*.
- Cryptography concerns hiding a message's *contents*.

[1] 'cryptography, n.' In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/45374?redirectedFrom=cryptography&.

[2] 'crypto-, comb. form'. In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/45363.

[3] 'graphy-, comb. form'. In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/80855.

- The word has its origin in greek[1]:
  κρυπτός (*kryptos*) meaning hidden[2].
  γράφος (*graphos*) meaning writing[3].
- The area has been around for ages.
- We should not confuse it with *steganography*.
- Steganography concerns hiding a message's *existence*.
- Cryptography concerns hiding a message's *contents*.

---

[1] 'cryptography, n.' In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/45374?redirectedFrom=cryptography&.

[2] 'crypto-, comb. form'. In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/45363.

[3] 'graphy-, comb. form'. In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/80855.

- Then it was an art, now it's a science.
- People used 'clever' constructions.
- These were thought to be secure: 'How can anyone figure this out?'
- Well, it turns out that there are always a lot of people with a lot of time and motivation . . .

- Then it was an art, now it's a science.
- People used 'clever' constructions.
- These were thought to be secure: 'How can anyone figure this out?'
- Well, it turns out that there are always a lot of people with a lot of time and motivation . . .

- Then it was an art, now it's a science.
- People used 'clever' constructions.
- These were thought to be secure: 'How can anyone figure this out?'
- Well, it turns out that there are always a lot of people with a lot of time and motivation . . .

A quote[4]

[A cryptosystem] should not require secrecy, and it should not be a problem if it falls into the enemy hands;

Kerckhoff's Principle

- No security-by-obscurity
- The key should be the only secret

[4] Auguste Kerckhoff. 'La cryptographie militaire'. In: *Journal des sciences militaires* 9 (1883), pp. 5–38, 161–191.

A quote[4]

[A cryptosystem] should not require secrecy, and it should not be a problem if it falls into the enemy hands;

Kerckhoff's Principle

- No security-by-obscurity
- The key should be the only secret

[4] Auguste Kerckhoff. 'La cryptographie militaire'. In: *Journal des sciences militaires* 9 (1883), pp. 5–38, 161–191.

Note

- This doesn't mean we must tell the adversary what we're using.
- But we shouldn't loose any security if we do.

Idea

- Alice and Bob share a (small) common secret.
- Alice takes a message, combines it with the secret, sends it to Bob.
- If Eve captures the whatever Alice sent, she shouldn't learn anything about the message.
- Bob combines what he received with the secret and gets the message.

Idea

- Alice and Bob share a (small) common secret.
- Alice takes a message, combines it with the secret, sends it to Bob.
- If Eve captures the whatever Alice sent, she shouldn't learn anything about the message.
- Bob combines what he received with the secret and gets the message.

Idea

- Alice and Bob share a (small) common secret.
- Alice takes a message, combines it with the secret, sends it to Bob.
- If Eve captures the whatever Alice sent, she shouldn't learn anything about the message.
- Bob combines what he received with the secret and gets the message.

Idea

- Alice and Bob share a (small) common secret.
- Alice takes a message, combines it with the secret, sends it to Bob.
- If Eve captures the whatever Alice sent, she shouldn't learn anything about the message.
- Bob combines what he received with the secret and gets the message.

Block-cipher encryption

Input A fixed-sized *key* k, a fixed-sized block of *plaintext* p.

Output A fixed-sized block of *ciphertext* c.

Notation $\text{Enc}_k(p) = c$

Block-cipher decryption

Input A fixed-sized *key* k, a fixed-sized block of *ciphertext* c.

Output A fixed-sized block of *plaintext* p.

Notation $\text{Dec}_k(c) = p$

Block-cipher encryption

Input A fixed-sized *key* k, a fixed-sized block of *plaintext* p.

Output A fixed-sized block of *ciphertext* c.

Notation $\mathrm{Enc}_k(p) = c$

Block-cipher decryption

Input A fixed-sized *key* k, a fixed-sized block of *ciphertext* c.

Output A fixed-sized block of *plaintext* p.

Notation $\mathrm{Dec}_k(c) = p$

Definition (Crypto system[5])

- A *crypto system* is a tuple $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where:
  - $\mathcal{M}$ is a finite set of *plaintexts* or messages,
  - $\mathcal{C}$ is a finite set of *ciphertexts*,
  - $\mathcal{K}$ is the *keyspace*, a finite set of keys.
  - $\mathcal{E}$ and $\mathcal{D}$ are the sets of encryption and decryption rules, respectively.
- For every $k \in \mathcal{K}$ there is a $\mathrm{Enc}_k \in \mathcal{E}$ and $\mathrm{Dec}_k \in \mathcal{D}$ such that
  - $\mathrm{Enc}_k : \mathcal{M} \to \mathcal{C}$ and $\mathrm{Dec}_k : \mathcal{C} \to \mathcal{M}$ are functions and
  - $\mathrm{Dec}_k(\mathrm{Enc}_k(m)) = m$ for all plaintexts $m \in \mathcal{M}$.

[5] Douglas R. Stinson. *Cryptography: Theory and Practice*. 3rd ed. Boca Raton: Chapman & Hall/CRC, 2006. ISBN: 1-58488-508-4 (Hardcover).

Definition (Crypto system[5])

- A *crypto system* is a tuple $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where:
  - $\mathcal{M}$ is a finite set of *plaintexts* or messages,
  - $\mathcal{C}$ is a finite set of *ciphertexts*,
  - $\mathcal{K}$ is the *keyspace*, a finite set of keys.
  - $\mathcal{E}$ and $\mathcal{D}$ are the sets of encryption and decryption rules, respectively.
- For every $k \in \mathcal{K}$ there is a $\mathrm{Enc}_k \in \mathcal{E}$ and $\mathrm{Dec}_k \in \mathcal{D}$ such that
  - $\mathrm{Enc}_k \colon \mathcal{M} \to \mathcal{C}$ and $\mathrm{Dec}_k \colon \mathcal{C} \to \mathcal{M}$ are functions and
  - $\mathrm{Dec}_k(\mathrm{Enc}_k(m)) = m$ for all plaintexts $m \in \mathcal{M}$.

[5] Douglas R. Stinson. *Cryptography: Theory and Practice*. 3rd ed. Boca Raton: Chapman & Hall/CRC, 2006. ISBN: 1-58488-508-4 (Hardcover).

Definition (Shift Cipher)

- Let $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{29}$
- For each $k \in \mathcal{K}$ we define

$$\begin{aligned} \mathrm{Enc}_k(m) &= (m + k) \bmod 29, m \in \mathcal{M}, \text{ och} \\ \mathrm{Dec}_k(c) &= (c - k) \bmod 29, c \in \mathcal{C}\,. \end{aligned}$$

Example

- $\mathrm{Enc}_3(7) = 7 + 3 \bmod 29 = 10$ h→J
- $\mathrm{Enc}_3(4) = 4 + 3 \bmod 29 = 7$ e→G
- $\mathrm{Enc}_3(9) = 9 + 3 \bmod 29 = 12$ j→L

Definition (Shift Cipher)

- Let $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{29}$
- For each $k \in \mathcal{K}$ we define

$$\begin{aligned} \text{Enc}_k(m) &= (m + k) \bmod 29, m \in \mathcal{M}, \text{ och} \\ \text{Dec}_k(c) &= (c - k) \bmod 29, c \in \mathcal{C}\,. \end{aligned}$$

Example

- $\text{Enc}_3(7) = 7 + 3 \bmod 29 = 10$ h→J
- $\text{Enc}_3(4) = 4 + 3 \bmod 29 = 7$ e→G
- $\text{Enc}_3(9) = 9 + 3 \bmod 29 = 12$ j→L

Note

- The shift cipher is a classical cipher — also know as the Caesar Cipher.
- It's easily broken *by hand*!
- It's used here for illustrative purposes.

Exercise

- What do we have to do to set this up between two parties, say Alice and Bob?
- What problems do we have to solve?

Definition (Perfect secrecy[6])

- Cryptosystem $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$.
- Stochastic variables M, C.
- *Perfect secrecy* if and only if

$$\Pr(M = m \mid C = c) = \Pr(M = m)$$

for all $m \in \mathcal{M}$ and $c \in \mathcal{C}$.

Note

Equivalent to $H(M \mid C) = H(M)$, i.e. ciphertext does not reveal anything about plaintext.

[6] Claude E Shannon. 'Communication theory of secrecy systems'. In: *Bell system technical journal* 28.4 (1949), pp. 656–715.

Definition (Perfect secrecy[6])

- Cryptosystem $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$.
- Stochastic variables M, C.
- *Perfect secrecy* if and only if

$$\Pr(M = m \mid C = c) = \Pr(M = m)$$

for all $m \in \mathcal{M}$ and $c \in \mathcal{C}$.

Note

Equivalent to $H(M \mid C) = H(M)$, i.e. ciphertext does not reveal anything about plaintext.

[6] Claude E Shannon. 'Communication theory of secrecy systems'. In: *Bell system technical journal* 28.4 (1949), pp. 656–715.

Theorem (Shannon's theorem)

- *Assume cryptosystem* $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *such that* $|\mathcal{K}| = |\mathcal{C}| = |\mathcal{M}|$.
- *This provides perfect secrecy if and only if*
  1. *every key* $k \in \mathcal{K}$ *is used with equal probability* $1/|\mathcal{K}|$,
  2. *for every plaintext* $m \in \mathcal{M}$ *and* $c \in \mathcal{C}$ *there is a unique key such that* $\mathrm{Enc}_k(m) = c$.

Theorem (Shannon's theorem)

- *Assume cryptosystem* $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *such that* $|\mathcal{K}| = |\mathcal{C}| = |\mathcal{M}|$.
- *This provides perfect secrecy if and only if*
  1. *every key* $k \in \mathcal{K}$ *is used with equal probability* $1/|\mathcal{K}|$,
  2. *for every plaintext* $m \in \mathcal{M}$ *and* $c \in \mathcal{C}$ *there is a unique key such that* $\mathrm{Enc}_k(m) = c$.

Example (One-time Pad)

- Let n be a positive integer.
- Let $\mathcal{M} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^n$.
- For each key $k = (k_1, \ldots, k_n) \in \mathcal{K}$, plaintexts $m = (m_1, \ldots, m_n) \in \mathcal{M}$ and ciphertexts $c = (c_1, \ldots, c_n) \in \mathcal{C}$ we define

$$\text{Enc}_k(m) = (m_1 + k_1, \ldots, m_n + k_n)$$

- We also define $\text{Dec} = \text{Enc}$.
- $k \in \mathcal{K}$ must be chosen uniformly randomly for each encryption.

Example (One-time Pad)

- Let n be a positive integer.
- Let $\mathcal{M} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^n$.
- For each key $k = (k_1, \ldots, k_n) \in \mathcal{K}$, plaintexts $m = (m_1, \ldots, m_n) \in \mathcal{M}$ and ciphertexts $c = (c_1, \ldots, c_n) \in \mathcal{C}$ we define

$$\mathrm{Enc}_k(m) = (m_1 + k_1, \ldots, m_n + k_n)$$

- We also define Dec = Enc.
- $k \in \mathcal{K}$ must be chosen uniformly randomly for each encryption.

Example (One-time Pad)

- Let n be a positive integer.
- Let $\mathcal{M} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^n$.
- For each key $k = (k_1, \ldots, k_n) \in \mathcal{K}$, plaintexts $m = (m_1, \ldots, m_n) \in \mathcal{M}$ and ciphertexts $c = (c_1, \ldots, c_n) \in \mathcal{C}$ we define

$$\mathrm{Enc}_k(m) = (m_1 + k_1, \ldots, m_n + k_n)$$

- We also define Dec = Enc.
- $k \in \mathcal{K}$ must be chosen uniformly randomly for each encryption.

Example (One-time Pad)

- Let n be a positive integer.
- Let $\mathcal{M} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^n$.
- For each key $k = (k_1, \ldots, k_n) \in \mathcal{K}$, plaintexts $m = (m_1, \ldots, m_n) \in \mathcal{M}$ and ciphertexts $c = (c_1, \ldots, c_n) \in \mathcal{C}$ we define

$$\mathrm{Enc}_k(m) = (m_1 + k_1, \ldots, m_n + k_n)$$

- We also define Dec = Enc.
- $k \in \mathcal{K}$ must be chosen uniformly randomly for each encryption.

Definition (Pseudo-random permutation, PRP[7])

- Let $F\colon \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^n$.
- F is a PRP if
  1. for any $k \in \{0,1\}^s$, F is a bijection;
  2. for any $k \in \{0,1\}^s$, we can 'efficiently' evaluate $F_k(x)$;
  3. for all 'efficient' distinguishers D,

$$\left|\Pr[D^{F_k}(1^n) = 1] - \Pr[D^{f_n}(1^n) = 1]\right| < \epsilon(s)$$

  if we choose $k \in \{0,1\}^s$ and the random permutation f_n uniformly at random.

[7] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. 1st ed. Boca Raton: Chapman & Hall/CRC, 2008. ISBN: 9781584885511.

Definition (Pseudo-random permutation, PRP[7])

- Let $F\colon \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^n$.
- F is a PRP if
  1. for any $k \in \{0,1\}^s$, F is a bijection;
  2. for any $k \in \{0,1\}^s$, we can 'efficiently' evaluate $F_k(x)$;
  3. for all 'efficient' distinguishers D,

$$\left|\Pr[D^{F_k}(1^n) = 1] - \Pr[D^{f_n}(1^n) = 1]\right| < \epsilon(s)$$

     if we choose $k \in \{0,1\}^s$ and the random permutation f_n uniformly at random.

[7] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. 1st ed. Boca Raton: Chapman & Hall/CRC, 2008. ISBN: 9781584885511.

Definition (Pseudo-random permutation, PRP[7])

- Let $F\colon \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^n$.
- F is a PRP if
  1. for any $k \in \{0,1\}^s$, F is a bijection;
  2. for any $k \in \{0,1\}^s$, we can 'efficiently' evaluate $F_k(x)$;
  3. for all 'efficient' distinguishers D,

$$\left|\Pr[D^{F_k}(1^n) = 1] - \Pr[D^{f_n}(1^n) = 1]\right| < \epsilon(s)$$

     if we choose $k \in \{0,1\}^s$ and the random permutation f_n uniformly at random.

[7] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. 1st ed. Boca Raton: Chapman & Hall/CRC, 2008. ISBN: 9781584885511.

Definition (Pseudo-random permutation, PRP[7])

- Let $F\colon \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^n$.
- F is a PRP if
  1. for any $k \in \{0,1\}^s$, F is a bijection;
  2. for any $k \in \{0,1\}^s$, we can 'efficiently' evaluate $F_k(x)$;
  3. for all 'efficient' distinguishers D,

$$\left|\Pr[D^{F_k}(1^n) = 1] - \Pr[D^{f_n}(1^n) = 1]\right| < \epsilon(s)$$

     if we choose $k \in \{0,1\}^s$ and the random permutation f_n uniformly at random.

[7] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. 1st ed. Boca Raton: Chapman & Hall/CRC, 2008. ISBN: 9781584885511.

Idea

- We want a function which we can efficiently compute.
- However, it shouldn't be possible to find its inverse.

Example

Easy $f(x) = y$

Hard $f^{-1}(y) = x$

Idea

- We want a function which we can efficiently compute.
- However, it shouldn't be possible to find its inverse.

Example

Easy $f(x) = y$

Hard $f^{-1}(y) = x$

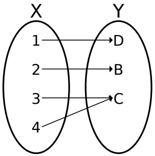
Idea

- We want a function which we can efficiently compute.
- However, it shouldn't be possible to find its inverse.

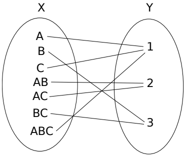
Example

Easy $f(x) = y$

Hard $f^{-1}(y) = x$



(a)
$h\colon X \to Y$



(b) $h'\colon X \to Y$

Figure: Two non-injective, surjective functions h and h'.

Exercise

Could either of these two functions be one-way functions?

Definition (One-way function[8])

- Let $h\colon \{0,1\}^* \to \{0,1\}^*$.
- h is *one-way* if
  1. there exists an efficient algorithm A such that $A(x) = h(x)$;
  2. for every efficient algorithm A', every positive polynomial $p(\cdot)$ and all sufficiently large n's

$$\Pr\left[A'(h(x), 1^n) \in h^{-1}(h(x))\right] < \frac{1}{p(n)}$$

[8] Oded Goldreich. *Foundations of cryptography, Vol. 1: Basic tools.* Cambridge: Cambridge Univ. Press, 2001. ISBN: 0-521-79172-3.

Example (Implementations you might've heard of)

- MD5
- SHA1
- SHA256 (SHA-2)
- SHA-3

Example (Applications)

- Verifying file content integrity
- Digital signatures
- Protect passwords

Note

- One-wayness returns as a useful property in many situations.
- Encryption also has the one-wayness property:
  - Easy: Given k, m, compute $c \leftarrow \text{Enc}_k(m)$.
  - Hard: Given c, compute either of k, m.
- However, encryption is bijective, hash functions are generally not.

Example

- Let $\text{Enc}_k(\cdot) = \text{Dec}_k(\cdot) = \cdot \oplus k \bmod 2$.
- Alice and Bob share k.
- Alice sends $\text{Enc}_k(m) = c$ to Bob.
- Eve intercepts c, she cannot get to m.
- Eve computes $c' = c \oplus m_E$ and passes c' to Bob.
- Bob computes $\text{Dec}_k(c') = \text{Dec}_k(c \oplus m_E) = m \oplus k \oplus m_E \oplus k = m \oplus m_E$.

Exercise

How can we solve this? Bob needs to know that Eve modified the message!

Example

- Let $\text{Enc}_k(\cdot) = \text{Dec}_k(\cdot) = \cdot \oplus k \bmod 2$.
- Alice and Bob share k.
- Alice sends $\text{Enc}_k(m) = c$ to Bob.
- Eve intercepts c, she cannot get to m.
- Eve computes $c' = c \oplus m_E$ and passes c' to Bob.
- Bob computes $\text{Dec}_k(c') = \text{Dec}_k(c \oplus m_E) = m \oplus k \oplus m_E \oplus k = m \oplus m_E$.

Exercise

How can we solve this? Bob needs to know that Eve modified the message!

Example

- Let $\text{Enc}_k(\cdot) = \text{Dec}_k(\cdot) = \cdot \oplus k \bmod 2$.
- Alice and Bob share k.
- Alice sends $\text{Enc}_k(m) = c$ to Bob.
- Eve intercepts c, she cannot get to m.
- Eve computes $c' = c \oplus m_E$ and passes c' to Bob.
- Bob computes $\text{Dec}_k(c') = \text{Dec}_k(c \oplus m_E) = m \oplus k \oplus m_E \oplus k = m \oplus m_E$.

Exercise

How can we solve this? Bob needs to know that Eve modified the message!

Example

- Let $\text{Enc}_k(\cdot) = \text{Dec}_k(\cdot) = \cdot \oplus k \bmod 2$.
- Alice and Bob share k.
- Alice sends $\text{Enc}_k(m) = c$ to Bob.
- Eve intercepts c, she cannot get to m.
- Eve computes $c' = c \oplus m_E$ and passes c' to Bob.
- Bob computes $\text{Dec}_k(c') = \text{Dec}_k(c \oplus m_E) = m \oplus k \oplus m_E \oplus k = m \oplus m_E$.

Exercise

How can we solve this? Bob needs to know that Eve modified the message!

Example

- Let $\mathrm{Enc}_k(\cdot) = \mathrm{Dec}_k(\cdot) = \cdot \oplus k \bmod 2$.
- Alice and Bob share k.
- Alice sends $\mathrm{Enc}_k(m) = c$ to Bob.
- Eve intercepts c, she cannot get to m.
- Eve computes $c' = c \oplus m_E$ and passes c' to Bob.
- Bob computes $\mathrm{Dec}_k(c') = \mathrm{Dec}_k(c \oplus m_E) = m \oplus k \oplus m_E \oplus k = m \oplus m_E$.

Exercise

How can we solve this? Bob needs to know that Eve modified the message!

Example

- Let $\text{Enc}_k(\cdot) = \text{Dec}_k(\cdot) = \cdot \oplus k \bmod 2$.
- Alice and Bob share k.
- Alice sends $\text{Enc}_k(m) = c$ to Bob.
- Eve intercepts c, she cannot get to m.
- Eve computes $c' = c \oplus m_E$ and passes c' to Bob.
- Bob computes $\text{Dec}_k(c') = \text{Dec}_k(c \oplus m_E) = m \oplus k \oplus m_E \oplus k = m \oplus m_E$.

Exercise

How can we solve this? Bob needs to know that Eve modified the message!

Idea: MACs

- Alice and Bob need something that Eve doesn't know how to modify.
- If that something is tied to the message, then a modified message would be detectable.

Exercise

Any ideas on how we can construct such a thing?

Idea: MACs

- Alice and Bob need something that Eve doesn't know how to modify.
- If that something is tied to the message, then a modified message would be detectable.

Exercise

Any ideas on how we can construct such a thing?

Idea: MACs

- Alice and Bob need something that Eve doesn't know how to modify.
- If that something is tied to the message, then a modified message would be detectable.

Exercise

Any ideas on how we can construct such a thing?

Example

- Let h be a one-way function.
- If we use $h(c) = t$, then Eve can also compute the hash function: $h(c') = t'$.
- A secret hash function would violate Kerckhoff's principle, so that's not an option.
- If we instead use the message, rather than the ciphertext.
- Then $h(m) = t$ and
  - $\text{Dec}_k(c') = m' = m \oplus m_E, h(m') \neq t$.
  - $\text{Dec}_k(c) = m, h(m) = t$.
- Eve cannot compute the hash function, she doesn't have m!
  - Bob: But neither do I!

Example

- Let h be a one-way function.
- If we use $h(c) = t$, then Eve can also compute the hash function: $h(c') = t'$.
- A secret hash function would violate Kerckhoff's principle, so that's not an option.
- If we instead use the message, rather than the ciphertext.
- Then $h(m) = t$ and
  - $\text{Dec}_k(c') = m' = m \oplus m_E, h(m') \neq t$.
  - $\text{Dec}_k(c) = m, h(m) = t$.
- Eve cannot compute the hash function, she doesn't have m!
  - Bob: But neither do I!

Example

- Let h be a one-way function.
- If we use $h(c) = t$, then Eve can also compute the hash function: $h(c') = t'$.
- A secret hash function would violate Kerckhoff's principle, so that's not an option.
- If we instead use the message, rather than the ciphertext.
- Then $h(m) = t$ and
  - $\mathrm{Dec}_k(c') = m' = m \oplus m_E, h(m') \neq t$.
  - $\mathrm{Dec}_k(c) = m, h(m) = t$.
- Eve cannot compute the hash function, she doesn't have m!
  - Bob: But neither do I!

Example

- Let h be a one-way function.
- If we use $h(c) = t$, then Eve can also compute the hash function: $h(c') = t'$.
- A secret hash function would violate Kerckhoff's principle, so that's not an option.
- If we instead use the message, rather than the ciphertext.
- Then $h(m) = t$ and
  - $\mathrm{Dec}_k(c') = m' = m \oplus m_E, h(m') \neq t$.
  - $\mathrm{Dec}_k(c) = m, h(m) = t$.
- Eve cannot compute the hash function, she doesn't have m!
  - Bob: But neither do I!

Example

- Let h be a one-way function.
- If we use $h(c) = t$, then Eve can also compute the hash function: $h(c') = t'$.
- A secret hash function would violate Kerckhoff's principle, so that's not an option.
- If we instead use the message, rather than the ciphertext.
- Then $h(m) = t$ and
  - $\text{Dec}_k(c') = m' = m \oplus m_E, h(m') \neq t$.
  - $\text{Dec}_k(c) = m, h(m) = t$.
- Eve cannot compute the hash function, she doesn't have m!
  - Bob: But neither do I!

Example

- Let h be a one-way function.
- If we use $h(c) = t$, then Eve can also compute the hash function: $h(c') = t'$.
- A secret hash function would violate Kerckhoff's principle, so that's not an option.
- If we instead use the message, rather than the ciphertext.
- Then $h(m) = t$ and
  - $\text{Dec}_k(c') = m' = m \oplus m_E, h(m') \neq t$.
  - $\text{Dec}_k(c) = m, h(m) = t$.
- Eve cannot compute the hash function, she doesn't have m!
  - Bob: But neither do I!

Solution

- *Let s be a secret shared between Alice and Bob.*
- $h(c \parallel s) = t$, *Eve doesn't know* s.
- *Bob can immediately check* $h(c' \parallel s) \neq t$.

Note

- It requires even a bit more than this!
- But the idea is correct.

Solution

- *Let s be a secret shared between Alice and Bob.*
- $h(c \parallel s) = t$*, Eve doesn't know* s*.*
- *Bob can immediately check* $h(c' \parallel s) \neq t$*.*

Note

- It requires even a bit more than this!
- But the idea is correct.

Solution

- *Let s be a secret shared between Alice and Bob.*
- $h(c \parallel s) = t$, *Eve doesn't know* s.
- *Bob can immediately check* $h(c' \parallel s) \neq t$.

Note

- It requires even a bit more than this!
- But the idea is correct.

Solution (Hash-based message-authentication code, HMAC[9])

- *Let h be a one-way function.*
- *Let c be the ciphertext, s our MA secret.*
- *Then tag $t = \mathrm{HMAC}_s(c)$, where*

$$\mathrm{HMAC}_s(c) = h\left[(s \oplus p_o) \parallel h\left[(s \oplus p_i) \parallel c\right]\right],$$

and p_i, p_o are inner and outer pads, respectively.

Note

This is proven secure in [9]!

[9] Mihir Bellare, Ran Canetti and Hugo Krawczyk. 'Keying Hash Functions for Message Authentication'. In: *Advances in Cryptology — CRYPTO '96: Prooceedings of the 16th Annual International Cryptology Conference.* Ed. by

Solution (Hash-based message-authentication code, HMAC[9])

- *Let h be a one-way function.*
- *Let c be the ciphertext, s our MA secret.*
- *Then tag $t = \mathrm{HMAC}_s(c)$, where*

$$\mathrm{HMAC}_s(c) = h\left[(s \oplus p_o) \parallel h\left[(s \oplus p_i) \parallel c\right]\right],$$

and p_i, p_o are inner and outer pads, respectively.

Note

This is proven secure in [9]!

[9] Mihir Bellare, Ran Canetti and Hugo Krawczyk. 'Keying Hash Functions for Message Authentication'. In: *Advances in Cryptology — CRYPTO '96: Prooceedings of the 16th Annual International Cryptology Conference*. Ed. by

Solution (Hash-based message-authentication code, HMAC[9])

- *Let h be a one-way function.*
- *Let c be the ciphertext, s our MA secret.*
- *Then tag $t = \mathrm{HMAC}_s(c)$, where*

$$\mathrm{HMAC}_s(c) = h\left[(s \oplus p_o) \parallel h\left[(s \oplus p_i) \parallel c\right]\right],$$

and p_i, p_o are inner and outer pads, respectively.

Note

This is proven secure in [9]!

[9] Mihir Bellare, Ran Canetti and Hugo Krawczyk. 'Keying Hash Functions for Message Authentication'. In: *Advances in Cryptology — CRYPTO '96: Prooceedings of the 16th Annual International Cryptology Conference*. Ed. by

Idea

- It's difficult to have to exchange keys in advance.
- What if we could securely exchange keys at a distance?
- If we could do it just before we use them?

Idea

- It's difficult to have to exchange keys in advance.
- What if we could securely exchange keys at a distance?
- If we could do it just before we use them?

Solution (Requirements)

- *We need a problem that is easy for Alice and Bob.*
- *It should be hard for Eve.*

Definition (Discrete Logarithm Problem, DLP)

- Let $\mathbb{Z}_p^*$ be the multiplicative group of residues modulo $p \in \mathbb{N}$, where p is a prime.

$$\text{Given } g, g^x \in \mathbb{Z}_p^* \quad \text{Find } x.$$

- I.e. compute $\log_{g \in \mathbb{Z}_p}(g^x)$.

Definition (Discrete Logarithm Problem, DLP)

- Let $\mathbb{Z}_p^*$ be the multiplicative group of residues modulo $p \in \mathbb{N}$, where p is a prime.

  Given $g, g^x \in \mathbb{Z}_p^*$
  Find x.

- I.e. compute $\log_{g \in \mathbb{Z}_p}(g^x)$.

Definition (Diffie-Hellman Problem, DHP[10])

Given $g, g^x, g^y \in \mathbb{Z}_p^*$

Find g^{xy}

Definition (Decisional Diffie-Hellman Problem, DDH)

Given $g, g^x, g^y, g^z \in \mathbb{Z}_p^*$

Decide $z \stackrel{?}{=} xy$

[10] Whitfield Diffie and Martin E Hellman. 'New directions in cryptography'. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

Definition (Diffie-Hellman Problem, DHP[10])

Given $g, g^x, g^y \in \mathbb{Z}_p^*$

Find g^{xy}

Definition (Decisional Diffie-Hellman Problem, DDH)

Given $g, g^x, g^y, g^z \in \mathbb{Z}_p^*$

Decide $z \stackrel{?}{=} xy$

[10] Whitfield Diffie and Martin E Hellman. 'New directions in cryptography'. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

- If we can solve DLP, then we can solve DHP and DDH too.
- Maybe DHP and DDH can be solved without DLP.
- We don't know yet.
- We usually assume DLP, DHP and DDH are hard.

- If we can solve DLP, then we can solve DHP and DDH too.
- Maybe DHP and DDH can be solved without DLP.
- We don't know yet.
- We usually assume DLP, DHP and DDH are hard.

- If we can solve DLP, then we can solve DHP and DDH too.
- Maybe DHP and DDH can be solved without DLP.
- We don't know yet.
- We usually assume DLP, DHP and DDH are hard.

Exercise

- Diffie and Hellman[11] used DHP to create a key-exchange protocol.
- Take some time to figure out how we can use these problems to achieve what we want.

Reminder

- Alice and Bob want to exchange a secret key.
- Then they can use the key to encrypt their communications.

[11] Whitfield Diffie and Martin E Hellman. 'New directions in cryptography'. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

Exercise

- Diffie and Hellman[11] used DHP to create a key-exchange protocol.
- Take some time to figure out how we can use these problems to achieve what we want.

Reminder

- Alice and Bob want to exchange a secret key.
- Then they can use the key to encrypt their communications.

[11] Whitfield Diffie and Martin E Hellman. 'New directions in cryptography'. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

Definition (Diffie-Hellman key-exchange)

- Let $g \in \mathbb{Z}_p^*$ (publicly known, e.g. RFC, standard dots).
- Alice generates random $0 < x < |\mathbb{Z}_p^*|$.
- She send g^x to Bob.
- Bob generates random $0 < y < |\mathbb{Z}_p^*|$.
- He sends g^y to Alice.
- Alice has x and g, g^y.
- Bob has g, g^x and y.
- They both compute $g^{xy} = (g^y)^x = (g^x)^y$.
- Eve has g, g^x, g^y.
- By DHP she cannot compute g^{xy}.

Definition (Diffie-Hellman key-exchange)

- Let $g \in \mathbb{Z}_p^*$ (publicly known, e.g. RFC, standard dots).
- Alice generates random $0 < x < |\mathbb{Z}_p^*|$.
- She send g^x to Bob.
- Bob generates random $0 < y < |\mathbb{Z}_p^*|$.
- He sends g^y to Alice.
- Alice has x and g, g^y.
- Bob has g, g^x and y.
- They both compute $g^{xy} = (g^y)^x = (g^x)^y$.
- Eve has g, g^x, g^y.
- By DHP she cannot compute g^{xy}.

Definition (Diffie-Hellman key-exchange)

- Let $g \in \mathbb{Z}_p^*$ (publicly known, e.g. RFC, standard dots).
- Alice generates random $0 < x < |\mathbb{Z}_p^*|$.
- She send g^x to Bob.
- Bob generates random $0 < y < |\mathbb{Z}_p^*|$.
- He sends g^y to Alice.
- Alice has x and g, g^y.
- Bob has g, g^x and y.
- They both compute $g^{xy} = (g^y)^x = (g^x)^y$.
- Eve has g, g^x, g^y.
- By DHP she cannot compute g^{xy}.

Definition (Diffie-Hellman key-exchange)

- Let $g \in \mathbb{Z}_p^*$ (publicly known, e.g. RFC, standard dots).
- Alice generates random $0 < x < |\mathbb{Z}_p^*|$.
- She send g^x to Bob.
- Bob generates random $0 < y < |\mathbb{Z}_p^*|$.
- He sends g^y to Alice.
- Alice has x and g, g^y.
- Bob has g, g^x and y.
- They both compute $g^{xy} = (g^y)^x = (g^x)^y$.
- Eve has g, g^x, g^y.
- By DHP she cannot compute g^{xy}.

Definition (Diffie-Hellman key-exchange)

- Let $g \in \mathbb{Z}_p^*$ (publicly known, e.g. RFC, standard dots).
- Alice generates random $0 < x < |\mathbb{Z}_p^*|$.
- She send g^x to Bob.
- Bob generates random $0 < y < |\mathbb{Z}_p^*|$.
- He sends g^y to Alice.
- Alice has x and g, g^y.
- Bob has g, g^x and y.
- They both compute $g^{xy} = (g^y)^x = (g^x)^y$.
- Eve has g, g^x, g^y.
- By DHP she cannot compute g^{xy}.

Idea

- Fine, we can use g^{xy} as a key in a cipher.
  - $\mathrm{Enc}_{g^{xy}}(m)$, where Enc is a symmetric cipher.
- But shouldn't we be able to include a message directly?

Definition (ElGamal Encryption Scheme[12])

Set-up:

- Let $g \in \mathbb{Z}_p^*$, randomly choose $0 < x < |\mathbb{Z}_p^*|$.
- Alice publishes $\mathbb{Z}_p^*, g, g^x$ to everyone.

Encryption:

- Bob chooses random $0 < y < |\mathbb{Z}_p^*|$ and computes g^y.
- Bob's message $m \in \mathbb{Z}_p^*$.
- He sends $(g^y, m(g^x)^y)$ to Alice.

Decryption:

- Alice computes $(g^y)^x$ and $m(g^x)^y((g^y)^x)^{-1} = m$.

[12] Taher ElGamal. 'A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms'. In: *Advances in Cryptology: Proceedings of CRYPTO 84*. Ed. by George Robert Blakley and David Chaum. Berlin,

Idea

- Sure, if Bob sends a message to Alice, he's sure she's the only one who can decrypt it.
- Can't we turn this around?
  - Can't Alice use the same system to ensure Bob knows the message came from Alice?

Exercise

- Look at the ElGamal encryption scheme for a bit.
- Try to find a way to 'run it backwards'.

Idea

- Sure, if Bob sends a message to Alice, he's sure she's the only one who can decrypt it.
- Can't we turn this around?
  - Can't Alice use the same system to ensure Bob knows the message came from Alice?

Exercise

- Look at the ElGamal encryption scheme for a bit.
- Try to find a way to 'run it backwards'.

Idea

- Sure, if Bob sends a message to Alice, he's sure she's the only one who can decrypt it.
- Can't we turn this around?
  - Can't Alice use the same system to ensure Bob knows the message came from Alice?

Exercise

- Look at the ElGamal encryption scheme for a bit.
- Try to find a way to 'run it backwards'.

Definition (ElGamal Signature Scheme[13])

Set-up:

- Let $g \in \mathbb{Z}_p^*$ and $\boxed{h \text{ be a one-way function}}$.
- Alice publishes $\mathbb{Z}_p^*, g, g^x$ to everyone.

Signing $m \in \mathbb{Z}_p^*$:

- $\boxed{\text{Alice}}$ chooses random $0 < y < |\mathbb{Z}_p^*|$ and computes $r = g^y \in \mathbb{Z}_p^*$.
- She computes $\boxed{s = (h(m) - xr)y^{-1} \pmod{|\mathbb{Z}_p^*|}}$.
- She sends (r, s) to Bob.

Verification:

- $\boxed{\text{Bob checks if } g^{h(m)} \stackrel{?}{=}_{\mathbb{Z}_p^*} (g^x)^r r^s} =_{\mathbb{Z}_p^*}$

$(\ \ x)^{g^y}(\ \ y)^{(h(m)-xg^y)y^{-1}} \quad \ldots \quad {}^{xg^y+h(m)-xg^y}$

Note

- It works without the hash.
- But then we can multiply two messages and still get a valid signature.

Definition (Homomorphism)

A *homomorphism* is a map (function) that preserves structure between two algebraic structures.

Example

- Let $G_1 = (\mathbb{R}, \cdot)$ and $G_2 = (\mathbb{R}, +)$ be groups.
- $g_1, g_1' \in G_1$ and $g_2, g_2' \in G_2$.
- Consider $\log \colon G_1 \to G_2$.
- $\log(g_1 \cdot g_1') = g_2 + g_2'$.

Definition (Homomorphism)

A *homomorphism* is a map (function) that preserves structure between two algebraic structures.

Example

- Let $G_1 = (\mathbb{R}, \cdot)$ and $G_2 = (\mathbb{R}, +)$ be groups.
- $g_1, g_1' \in G_1$ and $g_2, g_2' \in G_2$.
- Consider $\log \colon G_1 \to G_2$.
- $\log(g_1 \cdot g_1') = g_2 + g_2'$.

Definition (Homomorphism)

A *homomorphism* is a map (function) that preserves structure between two algebraic structures.

Example

- Let $G_1 = (\mathbb{R}, \cdot)$ and $G_2 = (\mathbb{R}, +)$ be groups.
- $g_1, g_1' \in G_1$ and $g_2, g_2' \in G_2$.
- Consider $\log\colon G_1 \to G_2$.
- $\log(g_1 \cdot g_1') = g_2 + g_2'$.

Definition (Homomorphism)

A *homomorphism* is a map (function) that preserves structure between two algebraic structures.

Example

- Let $G_1 = (\mathbb{R}, \cdot)$ and $G_2 = (\mathbb{R}, +)$ be groups.
- $g_1, g_1' \in G_1$ and $g_2, g_2' \in G_2$.
- Consider $\log\colon G_1 \to G_2$.
- $\log(g_1 \cdot g_1') = g_2 + g_2'$.

Exercise

The encryption (decryption) function of the ElGamal cryptosystem is a homomorphism, what structure does it preserve?

Example (ElGamal's homomorphism)

- Messages m, m', ciphertexts $(g^y, m \cdot g^{xy}), (g^{y'}, m' \cdot g^{xy'})$.
- Remember: private key x, hence the same.
- Create ciphertext

$$(g^y g^{y'}, m \cdot g^{xy} \cdot m' \cdot g^{xy'}) = (g^{y+y'}, m \cdot m' \cdot g^{xy+xy'})$$
$$= (g^{y+y'}, m \cdot m' \cdot g^{x(y+y')}).$$

- Decryption: take $g^{y+y'}$, compute $(g^{y+y'})^x = g^{x(y+y')}$.
- Decryption thus yields $m \cdot m'$.

Example (ElGamal's homomorphism)

- Messages m, m', ciphertexts $(g^y, m \cdot g^{xy}), (g^{y'}, m' \cdot g^{xy'})$.
- Remember: private key x, hence the same.
- Create ciphertext

$$(g^y g^{y'}, m \cdot g^{xy} \cdot m' \cdot g^{xy'}) = (g^{y+y'}, m \cdot m' \cdot g^{xy+xy'}) = (g^{y+y'}, m \cdot m' \cdot g^{x(y+y')}).$$

- Decryption: take $g^{y+y'}$, compute $(g^{y+y'})^x = g^{x(y+y')}$.
- Decryption thus yields $m \cdot m'$.

Example (ElGamal's homomorphism)

- Messages m, m', ciphertexts $(g^y, m \cdot g^{xy}), (g^{y'}, m' \cdot g^{xy'})$.
- Remember: private key x, hence the same.
- Create ciphertext

$$\begin{aligned}(g^y g^{y'}, m \cdot g^{xy} \cdot m' \cdot g^{xy'}) &= (g^{y+y'}, m \cdot m' \cdot g^{xy+xy'}) \\ &= (g^{y+y'}, m \cdot m' \cdot g^{x(y+y')}).\end{aligned}$$

- Decryption: take $g^{y+y'}$, compute $(g^{y+y'})^x = g^{x(y+y')}$.
- Decryption thus yields $m \cdot m'$.

Note

- We use a hash function in the signature scheme to counter the homomorphic property.
- $h(m) \cdot h(m') \neq h(m \cdot m')$.
- Without the hash function we could create a valid signature for a new message *without knowing the signature key!*

Note

- We use a hash function in the signature scheme to counter the homomorphic property.
- $h(m) \cdot h(m') \neq h(m \cdot m')$.
- Without the hash function we could create a valid signature for a new message *without knowing the signature key!*

Note

- There are many schemes with different homomorphic properties.
- There is even *fully homomorphic encryption* [12].

Example (Yao's millionaires' problem)

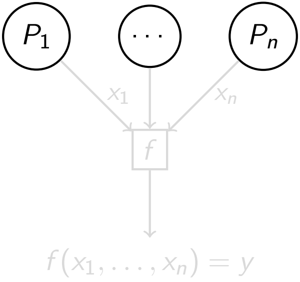
- Two millionaires meet in the street.
- They want to find out who is the richer.
- However, they don't want to reveal how many millions they each have.

Example (Yao's millionaires' problem)

- Two millionaires meet in the street.
- They want to find out who is the richer.
- However, they don't want to reveal how many millions they each have.

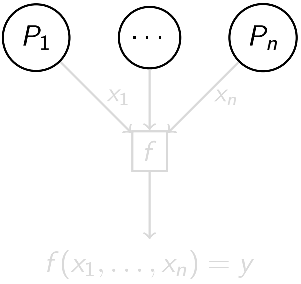
Idea

- We have n participants $P_1, \dots, P_n$.
- Each person has a *secret* input value x_j for $1 \leq j \leq n$.
- But they desperately want to know $y = f(x_1, \dots, x_n)$.



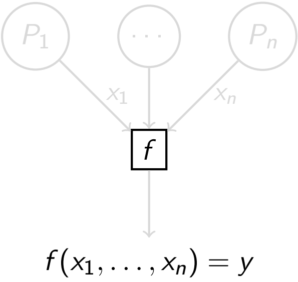
Idea

- We have n participants $P_1, \ldots, P_n$.
- Each person has a *secret* input value x_j for $1 \leq j \leq n$.
- But they desperately want to know $y = f(x_1, \ldots, x_n)$.



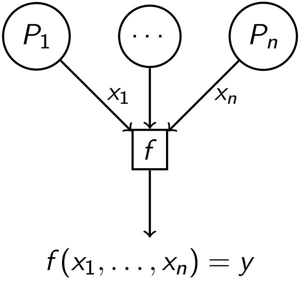
Idea

- We have n participants $P_1, \dots, P_n$.
- Each person has a *secret* input value x_j for $1 \leq j \leq n$.
- But they desperately want to know $y = f(x_1, \dots, x_n)$.



Idea

- We have n participants $P_1, \ldots, P_n$.
- Each person has a *secret* input value x_j for $1 \leq j \leq n$.
- But they desperately want to know $y = f(x_1, \ldots, x_n)$.



Example (Trivial solution)

- The n participants $P_1, \ldots, P_n$ agree on a trusted third-party (a TTP).
- Each participant give their secret to the TTP.
- The TTP trusted third-party performs the computation.
- Every participant receives the result from the TTP.

Definition (Secure multiparty computation, MPC)

- n participants $P_1, \ldots, P_n$.
- n secret inputs $x_1, \ldots, x_n$.
- A protocol π is executed by the participants.
- At the end of the protocol each participant learns $y = f(x_1, \ldots, x_n)$.
- The participants executing π should be *equivalent* to giving $x_1, \ldots, x_n$ to a TTP T who computes $f(x_1, \ldots, x_n) = y$ and returns y to each participant.

Note

Each participant P_i learns no more about x_j $(i \neq j)$ than what is revealed by y.

Definition (Secure multiparty computation, MPC)

- n participants $P_1, \ldots, P_n$.
- n secret inputs $x_1, \ldots, x_n$.
- A protocol π is executed by the participants.
- At the end of the protocol each participant learns $y = f(x_1, \ldots, x_n)$.
- The participants executing π should be *equivalent* to giving $x_1, \ldots, x_n$ to a TTP T who computes $f(x_1, \ldots, x_n) = y$ and returns y to each participant.

Note

Each participant P_i learns no more about x_j $(i \neq j)$ than what is revealed by y.

Definition (Secure multiparty computation, MPC)

- n participants $P_1, \ldots, P_n$.
- n secret inputs $x_1, \ldots, x_n$.
- A protocol π is executed by the participants.
- At the end of the protocol each participant learns $y = f(x_1, \ldots, x_n)$.
- The participants executing π should be *equivalent* to giving $x_1, \ldots, x_n$ to a TTP T who computes $f(x_1, \ldots, x_n) = y$ and returns y to each participant.

Note

Each participant P_i learns no more about x_j $(i \neq j)$ than what is revealed by y.

Definition (Secure multiparty computation, MPC)

- n participants $P_1, \ldots, P_n$.
- n secret inputs $x_1, \ldots, x_n$.
- A protocol π is executed by the participants.
- At the end of the protocol each participant learns $y = f(x_1, \ldots, x_n)$.
- The participants executing π should be *equivalent* to giving $x_1, \ldots, x_n$ to a TTP T who computes $f(x_1, \ldots, x_n) = y$ and returns y to each participant.

Note

Each participant P_i learns no more about x_j $(i \neq j)$ than what is revealed by y.

- In general this problem is solved.
- We can construct protocols for arbitrary functions f.
- Efficiency varies though.
- However, there are practically feasible protocols.
- Sometimes we can use homomorphisms.
- But we can construct rather complex functions too.

- In general this problem is solved.
- We can construct protocols for arbitrary functions f.
- Efficiency varies though.
- However, there are practically feasible protocols.
- Sometimes we can use homomorphisms.
- But we can construct rather complex functions too.

- In general this problem is solved.
- We can construct protocols for arbitrary functions f.
- Efficiency varies though.
- However, there are practically feasible protocols.
- Sometimes we can use homomorphisms.
- But we can construct rather complex functions too.

Example (Sugar beet auctions[14])

- Several thousand farmers produce sugar beets.
- These are sold to the monopoly Danisco, the sugar producer.
- Contracts are allocated via a nation-wide exchange, a double auction.
- A double auction contains multiple sellers and multiple buyers.
- The purpose is to find the *market clearing price*.

[14] Peter Bogetoft et al. 'Secure Multiparty Computation Goes Live'. In: *Financial Cryptography and Data Security: FC 2009*. Ed. by Roger Dingledine and Philippe Golle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–343. ISBN: 978-3-642-03549-4. DOI: 10.1007/978-3-642-03549-4_20. URL: http://dx.doi.org/10.1007/978-3-642-03549-4_20.

Example (Sugar beet auctions[14])

- Several thousand farmers produce sugar beets.
- These are sold to the monopoly Danisco, the sugar producer.
- Contracts are allocated via a nation-wide exchange, a double auction.
- A double auction contains multiple sellers and multiple buyers.
- The purpose is to find the *market clearing price*.

[14] Peter Bogetoft et al. 'Secure Multiparty Computation Goes Live'. In: *Financial Cryptography and Data Security: FC 2009*. Ed. by Roger Dingledine and Philippe Golle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–343. ISBN: 978-3-642-03549-4. DOI: 10.1007/978-3-642-03549-4_20. URL: http://dx.doi.org/10.1007/978-3-642-03549-4_20.

Example (Sugar beet auctions, continued)

- Each buyer places a bid specifying how much he is willing to buy *at each potential price*.
- Each seller says how much they are willing to sell at each given price.
- The auctioneer computes the total supply and demand for each price.
- We want to find where supply equals demand.
- When done, anyone who specified non-zero for this price may trade at this price.

Example (Sugar beet auctions, continued)

- Each buyer places a bid specifying how much he is willing to buy *at each potential price*.
- Each seller says how much they are willing to sell at each given price.
- The auctioneer computes the total supply and demand for each price.
- We want to find where supply equals demand.
- When done, anyone who specified non-zero for this price may trade at this price.

Example (Sugar beet auctions, continued)

- Each buyer places a bid specifying how much he is willing to buy *at each potential price*.
- Each seller says how much they are willing to sell at each given price.
- The auctioneer computes the total supply and demand for each price.
- We want to find where supply equals demand.
- When done, anyone who specified non-zero for this price may trade at this price.

Example

- Alice must prove her identity to Eve.
- Eve has Alice's public key, and knows it belongs to Alice.
- Alice wants to prove she is the owner of the private key belonging to the public key that Eve has.
- Eve asks Alice to sign the message m, if the signature verifies under the public key Eve believes Alice.

Gaaahh!

- Now Eve can show this message (chosen by Eve) with Alice's signature on it!
- What if Eve's chosen message was 'I give all my money to Eve'?

Example

- Alice must prove her identity to Eve.
- Eve has Alice's public key, and knows it belongs to Alice.
- Alice wants to prove she is the owner of the private key belonging to the public key that Eve has.
- Eve asks Alice to sign the message m, if the signature verifies under the public key Eve believes Alice.

Gaaahh!

- Now Eve can show this message (chosen by Eve) with Alice's signature on it!
- What if Eve's chosen message was 'I give all my money to Eve'?

Example

- Alice must prove her identity to Eve.
- Eve has Alice's public key, and knows it belongs to Alice.
- Alice wants to prove she is the owner of the private key belonging to the public key that Eve has.
- Eve asks Alice to sign the message *m*, if the signature verifies under the public key Eve believes Alice.

Gaaahh!

- Now Eve can show this message (chosen by Eve) with Alice's signature on it!
- What if Eve's chosen message was 'I give all my money to Eve'?

Example

- Alice must prove her identity to Eve.
- Eve has Alice's public key, and knows it belongs to Alice.
- Alice wants to prove she is the owner of the private key belonging to the public key that Eve has.
- Eve asks Alice to sign the message m, if the signature verifies under the public key Eve believes Alice.

Gaaahh!

- Now Eve can show this message (chosen by Eve) with Alice's signature on it!
- What if Eve's chosen message was 'I give all my money to Eve'?

Idea

- Alice wants to prove that she knows the discrete logarithm x of a value g^x.
- She will do this without revealing x to Eve.

Definition (Schnorr's protocol[15])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[15] C. P. Schnorr. 'Efficient signature generation by smart cards'. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174. ISSN: 1432-1378. DOI: 10.1007/BF00196725. URL: http://dx.doi.org/10.1007/BF00196725.

Definition (Schnorr's protocol[15])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[15] C. P. Schnorr. 'Efficient signature generation by smart cards'. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174. ISSN: 1432-1378. DOI: 10.1007/BF00196725. URL: http://dx.doi.org/10.1007/BF00196725.

Definition (Schnorr's protocol[15])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[15] C. P. Schnorr. 'Efficient signature generation by smart cards'. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174. ISSN: 1432-1378. DOI: 10.1007/BF00196725. URL: http://dx.doi.org/10.1007/BF00196725.

Definition (Schnorr's protocol[15])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[15] C. P. Schnorr. 'Efficient signature generation by smart cards'. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174. ISSN: 1432-1378. DOI: 10.1007/BF00196725. URL: http://dx.doi.org/10.1007/BF00196725.

Definition (Schnorr's protocol[15])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[15] C. P. Schnorr. 'Efficient signature generation by smart cards'. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174. ISSN: 1432-1378. DOI: 10.1007/BF00196725. URL: http://dx.doi.org/10.1007/BF00196725.

Proof outline.

- We need to prove *completeness*: for all (most) statements the verifier will accept.
- We need to prove *soundness*: for all (most) false statements the verifier will reject.
- We need to prove that it is zero-knowledge.

□

Proof outline.

- We need to prove *completeness*: for all (most) statements the verifier will accept.
- We need to prove *soundness*: for all (most) false statements the verifier will reject.
- We need to prove that it is zero-knowledge.

□

Proof outline.

- We need to prove *completeness*: for all (most) statements the verifier will accept.
- We need to prove *soundness*: for all (most) false statements the verifier will reject.
- We need to prove that it is zero-knowledge.

□

Zero-knowledge

- Transcript for protocol: (t, c, s).
- Probability for transcript occurring: $\frac{1}{|R|} \cdot \frac{1}{\deg g}$.
- Simulate protocol: randomly choose c, randomly choose s, compute t by $g^s y^c$.
- We see that we get the same probability distribution.
- Thus the simulated transcripts are indistinguishable from the real ones.

Zero-knowledge

- Transcript for protocol: (t, c, s).
- Probability for transcript occurring: $\frac{1}{|R|} \cdot \frac{1}{\deg g}$.
- Simulate protocol: randomly choose c, randomly choose s, compute t by $g^s y^c$.
- We see that we get the same probability distribution.
- Thus the simulated transcripts are indistinguishable from the real ones.

Zero-knowledge

- Transcript for protocol: (t, c, s).
- Probability for transcript occurring: $\frac{1}{|R|} \cdot \frac{1}{\deg g}$.
- Simulate protocol: randomly choose c, randomly choose s, compute t by $g^s y^c$.
- We see that we get the same probability distribution.
- Thus the simulated transcripts are indistinguishable from the real ones.

[1] 'cryptography, n.' In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/45374?redirectedFrom=cryptography&.

[2] 'crypto-, comb. form'. In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/45363.

[3] 'graphy-, comb. form'. In: *OED Online*. Hämtad den 5 april 2013. Oxford University Press, Mar. 2013. URL: http://www.oed.com/view/Entry/80855.

[4] Auguste Kerckhoff. 'La cryptographie militaire'. In: *Journal des sciences militaires* 9 (1883), pp. 5–38, 161–191.

[5] Douglas R. Stinson. *Cryptography: Theory and Practice*. 3rd ed. Boca Raton: Chapman & Hall/CRC, 2006. ISBN: 1-58488-508-4 (Hardcover).

[6] Claude E Shannon. ‘Communication theory of secrecy systems’. In: *Bell system technical journal* 28.4 (1949), pp. 656–715.

[7] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. 1st ed. Boca Raton: Chapman & Hall/CRC, 2008. ISBN: 9781584885511.

[8] Oded Goldreich. *Foundations of cryptography, Vol. 1: Basic tools*. Cambridge: Cambridge Univ. Press, 2001. ISBN: 0-521-79172-3.

[9] Mihir Bellare, Ran Canetti and Hugo Krawczyk. 'Keying Hash Functions for Message Authentication'. In: *Advances in Cryptology — CRYPTO '96: Prooceedings of the 16th Annual International Cryptology Conference*. Ed. by Neal Koblitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 1–15. ISBN: 978-3-540-68697-2. DOI: 10.1007/3-540-68697-5_1. URL: http://dx.doi.org/10.1007/3-540-68697-5_1.

[10] Whitfield Diffie and Martin E Hellman. 'New directions in cryptography'. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[11] Taher ElGamal. 'A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms'. In: *Advances in Cryptology: Proceedings of CRYPTO 84*. Ed. by George Robert Blakley and David Chaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 10–18. ISBN: 978-3-540-39568-3. DOI: 10.1007/3-540-39568-7_2. URL: http://dx.doi.org/10.1007/3-540-39568-7_2.

[12] Craig Gentry. 'A fully homomorphic encryption scheme'. PhD thesis. Stanford University, 2009. URL: https://crypto.stanford.edu/craig/craig-thesis.pdf.

[13] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach and Tomas Toft. 'Secure Multiparty Computation Goes Live'. In: *Financial Cryptography and Data Security: FC 2009*. Ed. by Roger Dingledine and Philippe Golle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–343. ISBN: 978-3-642-03549-4. DOI: 10.1007/978-3-642-03549-4_20. URL: http://dx.doi.org/10.1007/978-3-642-03549-4_20.

[14] C. P. Schnorr. 'Efficient signature generation by smart cards'. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174. ISSN: 1432-1378. DOI: 10.1007/BF00196725. URL: http://dx.doi.org/10.1007/BF00196725.