One-way functions

Daniel Bosk

School of Electrical Engineering and Computer Science,
KTH Royal Institute of Technology, Stockholm[1]

19th January 2026

[1] Part of the work was done while at the Department of Information and Communication Systems, Mid Sweden University, Sundsvall.

DD2520 Applied Cryptography
Lecture 3

Douglas Wikström
KTH Royal Institute of Technology
dog@kth.se

January 21, 2024

1 Introduction

2 One-way functions

3 Message-authentication codes

Example (Encrypt with OTP)

- 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 (Encrypt with OTP)

- 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 (Encrypt with OTP)

- 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 (Encrypt with OTP)

- 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 (Encrypt with OTP)

- 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 (Encrypt with OTP)

- 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?

Solution

- *We need redundancy.*
- *Let's try with a cipher.*
- *Alice and Bob share a verification key vk.*
- *Take the ciphertext c and generate a tag t:*

$$t = \mathrm{Enc}_{vk}(c).$$

- *Send (c, t) to Bob. Bob gets (c', t').*
- *Bob computes $t'' = \mathrm{Enc}_{vk}(c')$, checks $t'' \stackrel{?}{=} t'$.*

Question

- Any thoughts on this construction?

Solution

- *We need redundancy.*
- *Let's try with a cipher.*
- *Alice and Bob share a verification key vk.*
- *Take the ciphertext c and generate a tag t:*

$$t = \mathrm{Enc}_{vk}(c).$$

- *Send (c, t) to Bob. Bob gets (c', t').*
- *Bob computes $t'' = \mathrm{Enc}_{vk}(c')$, checks $t'' \stackrel{?}{=} t'$.*

Question

- Any thoughts on this construction?

idea

- We also want compression.
- Can't make things double in size.

1 Introduction

2 One-way functions

- Hash functions
- Hash functions in practice

3 Message-authentication codes

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

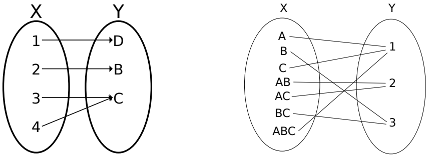
Idea

- 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?

Non-cryptographic Hash Functions

Recall how a hash table is constructed.

- An array D indexed by keys K.
- A huge set T of potential objects that may be stored in D.
- A **hash function** $h : T \to K$ that computes the key $h(t)$ of any object t to be stored.

Non-cryptographic Hash Functions

Recall that we need the following for a hash table to work as intended.

- **The function** h **must be exceptionally simple.** Thus, it may be easy to find a collision, and h is nothing like a randomly chosen function.

Non-cryptographic Hash Functions

Recall that we need the following for a hash table to work as intended.

- **The function h must be exceptionally simple.** Thus, it may be easy to find a collision, and h is nothing like a randomly chosen function.
- **The distribution of stored objects is not malicious.** Then h distributes objects "smoothly" over K.

Non-cryptographic Hash Functions

Recall that we need the following for a hash table to work as intended.

- **The function h must be exceptionally simple.** Thus, it may be easy to find a collision, and h is nothing like a randomly chosen function.
- **The distribution of stored objects is not malicious.** Then h distributes objects "smoothly" over K.
- **The size $|K|$ of the table is not large.** Thus, due to the birthday paradox we cannot expect to avoid collisions, but we do not care as long as they are evenly distributed.

If assumptions are violated we go from $O(1)$ to $O(\log n)$ lookups or memory is wasted.

Cryptographic Hash Function

A **(cryptographic) hash function** maps arbitrarily long bit strings into bit strings of fixed length.

The output of a hash function should be “unpredictable”.

Wish List

- Finding a pre-image of an output should be hard.
- Finding two inputs giving the same output should be hard.
- The output of the function should be “random”.

Definition (Preimage resistance)

Input hash function H, value y.

Output Any x such that $H(x) = y$.

Definition (Second preimage resistance)

Input hash function H, value x.

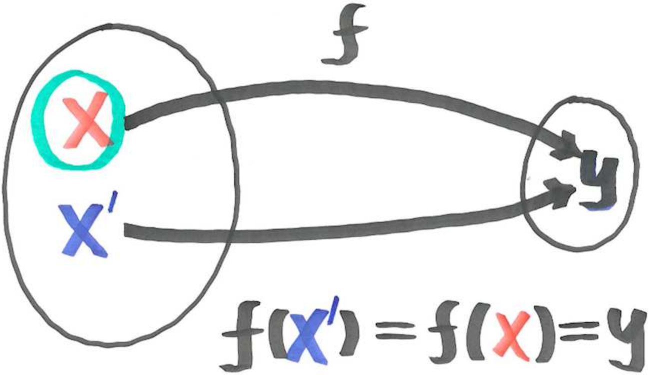
Output Any value x' such that $H(x) = H(x')$.

Definition (Collision resistance)

Input hash function H.

Output Any two x, x' such that $H(x) = H(x')$.

Pre-image Problem



RANDOM	SECRET	COMPUTED	PUBLIC

One-Wayness

Definition. A function f on bit strings is said to be **one-way**[1] if given $f(x)$ for a random x it is infeasible[2] to compute x' such that $f(x) = f(x')$.

[1] "Enkelriktad" på svenska **inte** "envägs".

[2] This means that we are convinced that it is impossible in practice.

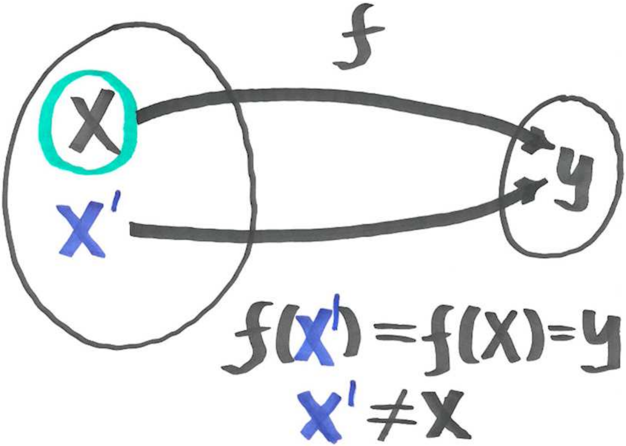
Definition (One-way function[2])

- 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

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

[2] Oded Goldreich. *Foundations of cryptography, Vol. 1: Basic tools*. Cambridge: Cambridge Univ. Press, 2001.

Second Pre-image Problem

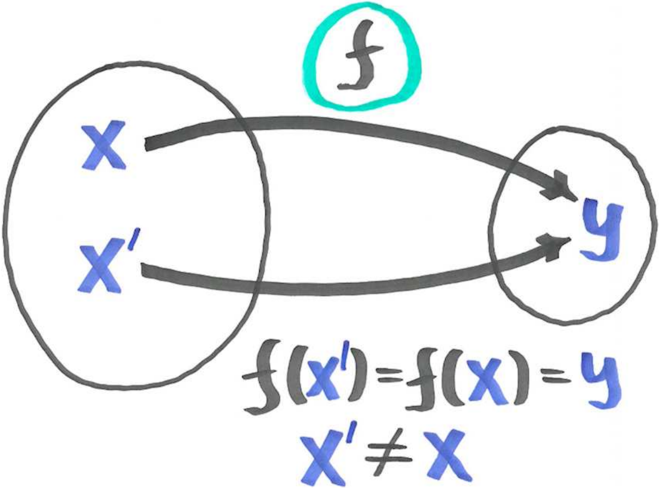


RANDOM	COMPUTED	PUBLIC

Second Pre-image Resistance

Definition. A function h on bit strings is said to be **second pre-image resistant** if given a random x it is infeasible to compute $x' \neq x$ such that $f(x') = f(x)$.

Collision Problem



RANDOM	COMPUTED	PUBLIC

Collision Resistance

Definition. Let $f = \{f_\alpha\}_\alpha$ be an ensemble of functions. The "function" f is said to be **collision resistant** if given a random α it is infeasible to compute $x \neq x'$ such that $f_\alpha(x') = f_\alpha(x)$.

Collision Resistance

Definition. Let $f = \{f_\alpha\}_\alpha$ be an ensemble of functions. The "function" f is said to be **collision resistant** if given a random α it is infeasible to compute $x \neq x'$ such that $f_\alpha(x') = f_\alpha(x)$.

An algorithm that gets a small "advice string" for each security parameter can easily hardcode a collision for a fixed function f, which explains the random index α.

Birthday Paradox and Hash Functions

Suppose that the range of a function f is R and let $r = |R|$. Then the probability that there is at least one collision for k random inputs is equal to

$$1 - \left(1 - \frac{1}{r}\right)\left(1 - \frac{2}{r}\right)\cdots\left(1 - \frac{k-1}{r}\right) \approx 1 - e^{-k^2/r} \ .$$

Birthday Paradox and Hash Functions

Suppose that the range of a function f is R and let $r = |R|$. Then the probability that there is at least one collision for k random inputs is equal to

$$1 - \left(1 - \frac{1}{r}\right)\left(1 - \frac{2}{r}\right) \cdots \left(1 - \frac{k-1}{r}\right) \approx 1 - e^{-k^2/r} \quad .$$

When $k = \Omega(\sqrt{r})$ we should **expect** a collision for **any** function!

Relations for Compressing Hash Functions

- If a function is not second pre-image resistant, then it is not collision-resistant.

Relations for Compressing Hash Functions

- If a function is not second pre-image resistant, then it is not collision-resistant.
  1. Pick random x.
  2. Request second pre-image $x' \neq x$ with $f(x') = f(x)$.
  3. Output x' and x.

Relations for Compressing Hash Functions

- If a function is not second pre-image resistant, then it is not collision-resistant.
  1. Pick random x.
  2. Request second pre-image $x' \neq x$ with $f(x') = f(x)$.
  3. Output x' and x.

- If a function is not one-way, then it is not second pre-image resistant.

Relations for Compressing Hash Functions

- If a function is not second pre-image resistant, then it is not collision-resistant.
    1. Pick random x.
    2. Request second pre-image $x' \neq x$ with $f(x') = f(x)$.
    3. Output x' and x.

- If a function is not one-way, then it is not second pre-image resistant.
    1. Given random x, compute $y = f(x)$.
    2. Request pre-image x' of y.
    3. Repeat until $x' \neq x$, and output x'.

Random Oracles

Random Oracle As Hash Function

A random oracle is simply a randomly chosen function with appropriate domain and range.

A random oracle is the **perfect** hash function. Every input is mapped **independently** and **uniformly** in the range.

Let us consider how a random oracle behaves with respect to our notions of security of hash functions.

Pre-Image of Random Oracle

We assume with little loss that an adversary always "knows" if it has found a pre-image, i.e., it queries the random oracle on its output.

Theorem. Let $H : X \to Y$ be a randomly chosen function and let $x \in X$ be randomly chosen. Then for every algorithm A making q oracle queries

$$\Pr[A^{H(\cdot)}(H(x)) = x' \wedge H(x) = H(x')] \leq 1 - \left(1 - \frac{1}{|Y|}\right)^q \enspace .$$

Pre-Image of Random Oracle

We assume with little loss that an adversary always "knows" if it has found a pre-image, i.e., it queries the random oracle on its output.

Theorem. Let $H : X \to Y$ be a randomly chosen function and let $x \in X$ be randomly chosen. Then for every algorithm A making q oracle queries

$$\Pr[A^{H(\cdot)}(H(x)) = x' \wedge H(x) = H(x')] \leq 1 - \left(1 - \frac{1}{|Y|}\right)^q \quad .$$

Proof. Each query x' satisfies $H(x') \neq H(x)$ independently with probability $1 - \frac{1}{|Y|}$.

Remark

- We can prove similar results for second-preimage and collision resistance.
- Collision resistance implies the other two.

1 Introduction

2 One-way functions
- Hash functions
- Hash functions in practice

3 Message-authentication codes

Example (Implementations you might've heard of)

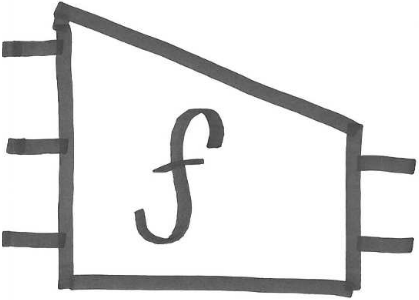
- MD5
- SHA1
- SHA256 (SHA-2)
- SHA-3

Example (Applications)

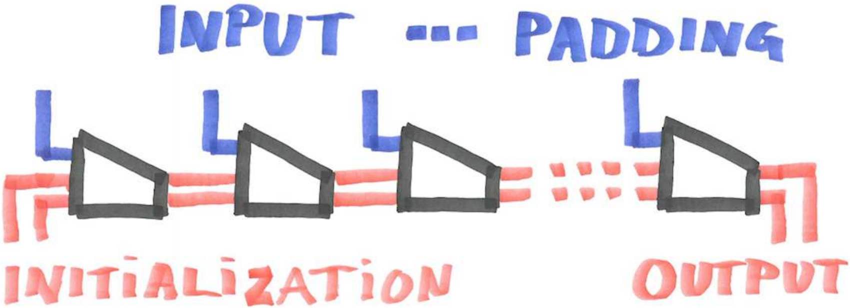
- Verifying file content integrity
- Digital signatures
- Protect passwords

Iterated Hash Functions

Compression Function



Merkle-Damgård (1/4)



Merkle-Damgård (2/4)

Suppose that we are given a collision resistant hash function

$$f : \{0,1\}^{n+t} \rightarrow \{0,1\}^n \ .$$

How can we construct a collision resistant hash function

$$h : \{0,1\}^* \rightarrow \{0,1\}^n$$

mapping any length inputs?

Merkle-Damgård (3/4)

Construction.

1. Let $x = (x_1, \ldots, x_k)$ with $|x_i| = t$ and $0 < |x_k| \leq t$.
2. Let x_{k+1} be the total number of bits in x.
3. Pad x_k with zeros until it has length t.
4. $y_0 = 0^n$, $y_i = f(y_{i-1}, x_i)$ for $i = 1, \ldots, k+1$.
5. Output y_{k+1}

Here the total number of bits is bounded by $2^t - 1$, but this can be relaxed.

Merkle-Damgård (4/4)

Suppose A finds collisions in Merkle-Damgård.

- If the number of bits differ in a collision, then we can derive a collision from the last invocation of f.
- If not, then we move backwards until we get a collision. Since both inputs have the same length, we are guaranteed to find a collision.

Standardized Hash Functions

Despite that theory says it is impossible, in practice people simply live with **fixed** hash functions and use them as if they are randomly chosen functions.

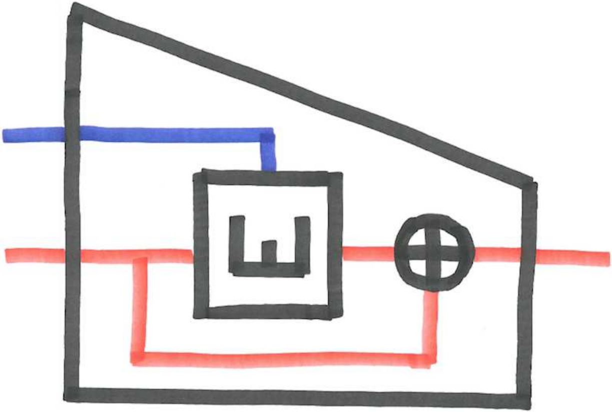
SHA-0,1,2

- Secure Hash Algorithm (SHA-0,1, and the SHA-2 family) are hash functions standardized by NIST to be used in, e.g., signature schemes and random number generation.
- SHA-0 was **weak** and withdrawn by NIST. SHA-1 was **withdrawn** 2010. SHA-2 family is based on similar ideas but seems safe so far...
- All are **iterated** hash functions, starting from a basic **compression function** essentially derived from an encryption function E

$$f(y_{i-1}, x_i) = \mathrm{E}_{x_i}(y_{i-1}) \oplus y_{i-1}$$

and you know how to build a cipher :-)

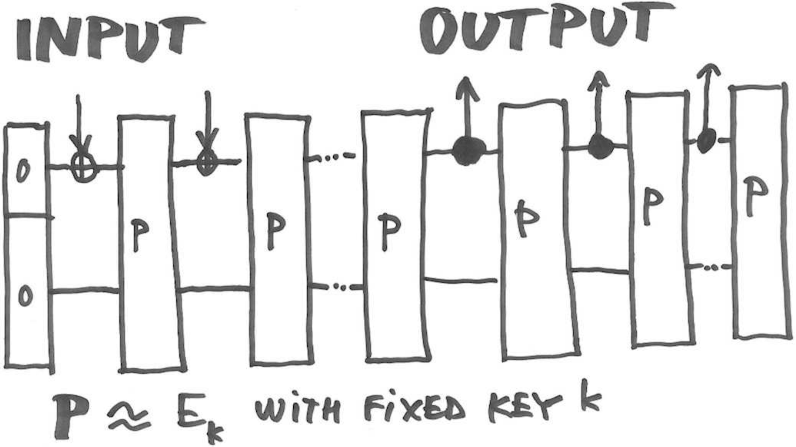
SHA-2



SHA-3

- NIST ran an open competition for the next hash function, named SHA-3. Several groups of famous researchers submitted proposals.
- Call for SHA-3 explicitly asked for "different" hash functions.
- It might be a good idea to read about SHA-1 for comparison.
- The competition ended October 2, 2012, and the hash function **Keccak was selected as the winner**.
- This was constructed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche,

SHA-3 Is a Sponge Function



Remark

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

Example (Encrypt with OTP, again)

- 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 (Encrypt with OTP, again)

- 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 (Encrypt with OTP, again)

- 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 (Encrypt with OTP, again)

- 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 (Encrypt with OTP, again)

- 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 (Encrypt with OTP, again)

- 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?

Remark

- Last time we used a cipher.

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?

Remark

- Last time we used a cipher.

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?

Remark

- Last time we used a cipher.

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 makes up m', she can compute $t' = h(m')$. Also lets Eve guess m.

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
  - $\operatorname{Dec} k(c') = m' = m \oplus m_E, h(m') \neq t$.
  - $\operatorname{Dec} k(c) = m, h(m) = t$.
- Eve makes up m', she can compute $t' = h(m')$. Also lets Eve guess m.

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.[3]
- 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 makes up m', she can compute $t' = h(m')$. Also lets Eve guess m.

[3] However, in theory, we should pick a random hash function. But, in practice, we use a fixed hash function. There aren't too many to choose from.

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 makes up m', she can compute $t' = h(m')$. Also lets Eve guess m.

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
  - $\operatorname{Dec} k(c') = m' = m \oplus m_E, h(m') \neq t$.
  - $\operatorname{Dec} k(c) = m, h(m) = t$.
- Eve makes up m', she can compute $t' = h(m')$. Also lets Eve guess m.

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 $h(c' \parallel s) \neq t$.*

Remark

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

Exercise

- Why is it fine for s but not for m (before)?

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 $h(c' \parallel s) \neq t$.*

Remark

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

Exercise

- Why is it fine for s but not for m (before)?

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* $h(c' \parallel s) \neq t$.

Remark

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

Exercise

- Why is it fine for s but not for m (before)?

Solution (Hash-based message-authentication code [2], HMAC[3])

- *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.

Remark

This is proven secure by Bellare, Canetti and Krawczyk [2]!

[3] 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.

Solution (Hash-based message-authentication code, HMAC[3])

- *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.

Remark

This is proven secure by Bellare, Canetti and Krawczyk [2]!

[3] 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.

Solution (Hash-based message-authentication code, HMAC[3])

- *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.

Remark

This is proven secure by Bellare, Canetti and Krawczyk [2]!

[3] 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.

Oded Goldreich. *Foundations of cryptography, Vol. 1: Basic tools*. Cambridge: Cambridge Univ. Press, 2001.

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.