# Crypto 7

**Note: this material is not intended to replace the live lecture for students.**

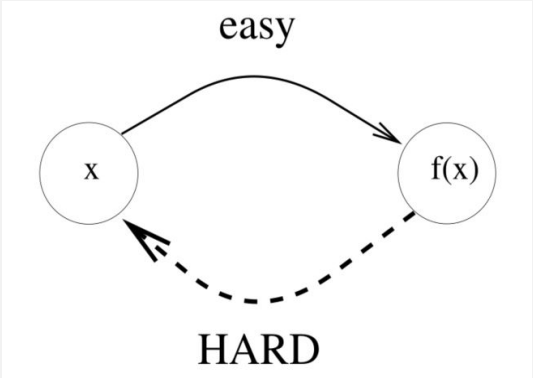
**Contents**



## 7.1 One-way functions & trapdoors

### 7.1.1 Preimage resistance or one-way property

A function $f : D \to T$ is one-way if for $x \in D$ the value $f(x)$ is can be computed efficiently but for any $y \in T$ it is not computationally feasible to find $x \in D$ such that $f(x) = y$:




**NOTE 7.1.2**

One-way functions $f$ can be obtained by using a secure block-cipher $(\mathsf{Enc}, \mathsf{Dec})$.

Example 1) :

$$f : \mathcal{K} \to \mathcal{C}$$

from the key space to the ciphertext space is defined as follows: pick a random plaintext $\mathrm{P}_0$ and put

$$f(\mathsf{k}) := \mathsf{Enc}_{\mathsf{k}}(\mathrm{P}_0)$$

So, since encryption should be efficient we get that computing $f(\mathsf{k})$ is easy. But finding $\mathsf{k}$ for a given C is cryptanalysis which should be hard.

Example 2) :

$$f : \mathcal{P} \to \mathcal{C}$$

from plaintext space to ciphertext space defined as

$$f(\mathrm{P}) := \mathsf{Enc}_{\mathsf{k}}(\mathrm{P})$$

here the key $\mathsf{k}$ is regarded as a **trapdoor** i.e. the special information which turns to easy inversion.

Well-known examples are RSA and Rabin functions $f(x) = x^e \pmod n$ and $f(x) = x^2 \pmod n$. The trapdoor is the factorization of $n$ i.e. the two prime numbers $p, q$ such that $n = p \cdot q$. For the Rabin's function keep in mind the role of CRT + Tonelli-Shanks' algorithm.

### 7.1.1 Discrete Logarithm

A typical example of one-way function is the power map $f : \mathbb{Z}_n \to \langle\alpha\rangle$:

$$\gamma \to \alpha^\gamma$$

where $\alpha$ has order $n$. Usually, the best way to solve $f(\gamma) = y$ is related to the Birthday's Paradox, an algorithm of complexity $O(\sqrt{n})$:

If $\beta = \alpha^\gamma$ and

$$\alpha^a \beta^b = \alpha^A \beta^B$$

then

$$\gamma = \frac{B-b}{a-A} \pmod n$$

https://en.wikipedia.org/wiki/Pollard%27s_rho_algorithm_for_logarithms

**NOTE 7.1.3**

This is the one-way function used by Diffie-Hellman in his key exchange cryptosystem.

### 7.1.2 Factorization & CRT

Another one-way is the map which takes two prime numbers $p, q$ and gets the product $N = p \cdot q$. Namely, it is computationally efficient to multiply $p \cdot q$ but is computationally expensive to compute the factors both factors $p$ and $q$ from $N$.

**NOTE 7.1.4**

This is the one-way function that together with the CRT it is used in RSA.

### 7.1.3 Computing with encrypted data

**Alice** has a particular value $x$ and wants to compute

$$f(x)$$

but either her computer is broken or it has not enough computational power. **Bob** is willing to compute $f(x)$ for her, but **Alice** isn't keen on letting **Bob** know her $x$.

In such situation **Bob** behaves as an **oracle** i.e. he answer questions.

#### 7.1.5 Computing with DL encrypted data

Let $p$ a prime number, let $g$ a generator and $f(x)$ be the corresponding discrete logarithm. Namely, if $x = g^e \pmod p$ then

$$f(x) = e$$

To get $e$ without revealing $x$ **Alice** generate a random $r \in \{1, \cdots, p-1\}$, computes $\tilde{x} = x \cdot g^r$ and ask **Bob** $f(\tilde{x}) = \tilde{e}$.

Then **Alice** recover $e$ from

$$e + r = \tilde{e} \pmod{p-1}$$

## 7.2 Hash functions

Hash functions-such as MD5, SHA-1, SHA-256, SHA-3, and BLAKE2 -comprise the cryptographer's Swiss Army Knife: they are used in digital signatures, public-key encryption, integrity verification, message authentication, password protection, key agreement protocols, and many other cryptographic protocols.

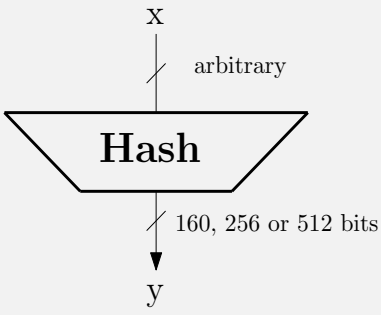
**[Aumasson18, Chapter 6]**

A funzione **Hash** has inputs in $\mathbb{Z}_2^*$ and output a *digest* of $n = 160, 256, 512$ bits:

$$\textbf{Hash} : \mathbb{Z}_2^* \to \mathbb{Z}_2^n$$

The value $y = \textbf{Hash}(x)$, is also called *hash value, hash code* . This hash value is usually regarded as finger print of the input $x$.

Figure 7.2.1: Hash function



Hash functions with full (co)domain or with a parameter for the length of the digest can be also useful in applications e.g. **SHAKE**128(M,d) and **SHAKE256**(M,d)..

Hash properties

**one-way**

**Collision Resistance**

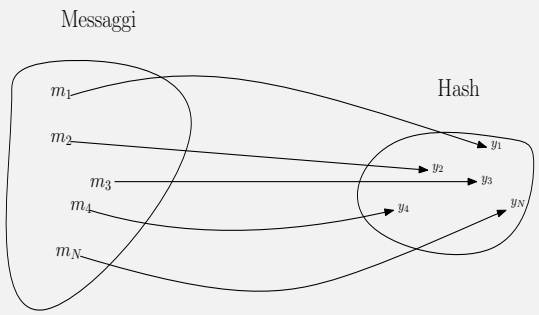
**Second Preimage resistance or weak collision**

**Exercise 7.2.2**

Go to http://www.sha1-online.com/ and compute the **SHA**-1 of the letters a, b e c. Notice that **SHA**-1(d) has nothing to do with **SHA**-1(a) or **SHA**-1(b) or **SHA**-1(c).

## Collisions: Birthday attack

Assume a digest of $n$ bit. With $N$ messages $x_1, \cdots x_N$ we can form $\frac{N(N-1)}{2}$ pairs which are candidates for a collision. So it is natural to guess that the probability $\lambda_N$ of a collision between two of these $\frac{N(N-1)}{2}$ pairs of messages would be related to the numbers $2^{\frac{n}{2}}$ and $2^n$.




Here the equation relating $\lambda_N$ and the digest length:

$$N \approx 2^{\frac{n+1}{2}} \cdot \sqrt{\ln\left(\frac{1}{1-\lambda_N}\right)}$$

The above equation follows from the following discussion.
A collision between the $N$ messages is produced when the restriction of the **Hash** function to the set of $N$ messages is no more injective. So the probability $1-\lambda_N$ of **no collision** is the quotient between the number of injective functions and the number of all possible functions:

$$1-\lambda_N = \frac{N! \cdot \binom{2^n}{N}}{(2^n)^N} = \frac{2^n!}{(2^n-N)! \cdot (2^n)^N} = \frac{2^n \cdot (2^n-1) \cdot (2^n-2) \cdots (2^n-(N-1))}{(2^n)^N} =$$

hence

$$\lambda_N = 1 - (1-\frac{1}{2^n}) \cdot (1-\frac{2}{2^n}) \cdots \left(1-\frac{(N-1)}{2^n}\right)$$

by using $e^{-x} \approx 1-x$ for $x$ close to zero we get:

$$\lambda_N \approx 1 - e^{-\frac{1}{2^n}} \cdot e^{-\frac{2}{2^n}} \cdots e^{-\frac{N-1}{2^n}} = 1 - e^{-\frac{N \cdot (N-1)}{2 \cdot 2^n}}$$

and from this we get:

$$N \approx 2^{\frac{n+1}{2}} \cdot \sqrt{\ln\left(\frac{1}{1-\lambda_N}\right)}$$

**Exercise 7.2.3**

We consider three different hash functions which produce outputs of lengths 64, 128 and 160 bit. After how many random inputs do we have a probability of $\lambda = 0.5$ for a collision? After how many random inputs do we have a probability of $\lambda = 0.1$ for a collision?

**Finding meaningful collisions.** The algorithm just described may not seem amenable to finding meaningful collisions since it has no control over the elements sampled. Nevertheless, we show how finding meaningful collisions is possible. The trick is to find a collision in the right function!

Assume, as before, that Alice wishes to find a collision between messages of two different "types," e.g., a letter explaining why Alice was fired and a flattering letter of recommendation that both hash to the same value. Then, Alice writes each message so that there are $\ell - 1$ interchangeable words in each; i.e., there are $2^{\ell-1}$ messages of each type. Define the one-to-one function $g : \{0,1\}^\ell \to \{0,1\}^*$ such that the $\ell$th bit of the input selects between messages of type 0 or type 1, and the $i$th bit (for $1 \le i \le \ell - 1$) selects between options for the $i$th interchangeable word in messages of the appropriate type. For example, consider the sentences:

0: Bob is a *good/hardworking* and *honest/trustworthy worker/employee.*
1: Bob is a *difficult/problematic* and *taxing/irritating worker/employee.*

Define a function $g$ that takes 4-bit inputs, where the last bit determines the type of sentence output, and the initial three bits determine the choice of words in that sentence. For example:

$g(0000)$ = Bob is a good and honest worker.
$g(0001)$ = Bob is a difficult and taxing worker.
$g(1010)$ = Bob is a hardworking and honest employee.
$g(1011)$ = Bob is a problematic and taxing employee.

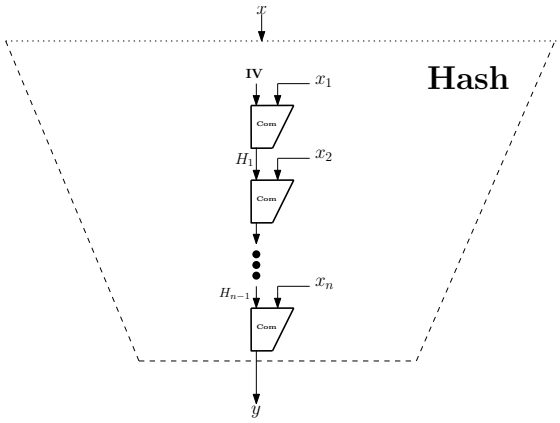
Now define $f : \{0,1\}^\ell \to \{0,1\}^\ell$ by $f(x) \stackrel{\text{def}}{=} H(g(x))$. Alice can find a collision in $f$ using the small-space birthday attack shown earlier. The point here is that any collision $x, x'$ in $f$ yields two messages $g(x), g(x')$ that collide under $H$. If $x, x'$ is a random collision then we expect that with probability $1/2$ the colliding messages $g(x), g(x')$ will be of different types (since $x$ and $x'$ differ in their final bit with that probability). If the colliding messages are not of different types, the process can be repeated again from scratch.

**[KatLin15, page 168]**

### 7.2.1 Compression & Merkle-Damgård

Merkle–Damgård is a method to construct a **Hash** from a compression function **Com**. An $IV$ initialization vector is necessary.

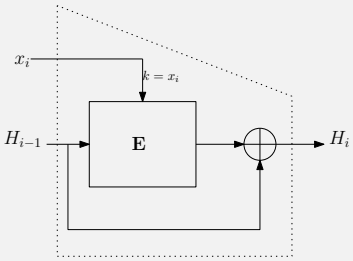
After a padding, the input $x$ is divided in blocks $x = x_1 x_2 \cdots x_n$ it goes through the loop:




The compression function can come from a block cipher:

**Davies-Meyer construction of Com**

Un block-cipher Enc is used to get a compression function **Com**:




Observe that the blocks $x_i$ are used as keys and the previous $H_i$'s as blocks to be ciphered.
**Notice the XOR at the end:** $\mathsf{Enc}_{x_i}(H_{i-1}) \oplus H_{i-1}$ **(Cf. Salsa20 & Chacha20)**

**Exercise 7.2.4**

Show that the David-Meyer function **Com** has fixed points. Namely, for any $x_i$ there are $H$ such that

$$H = \mathsf{Enc}_{x_i}(H) \oplus H$$

**Exercise 7.2.5**

Here we construct a hash function by using a toy compression function.

Let $\mathsf{Com} : \{0,1\}^5 \times \{0,1\}^5 \to \{0,1\}^5$ the function defined as follows

$$([a_9a_8a_7a_6a_5, a_4a_3a_2a_1a_0]) = [b_4b_3b_2b_1b_0]$$

where $b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$ is the remainder of the division of $a_9x^9 + a_8x^8 + \cdots + a_0$ by $G(x) = x^5 + x^2 + 1$ in $\mathrm{Z}_2[x]$.

Let $x \in \{0,1\}^*$ a message of length $|x|$. Consider the padding $\mathrm{padd}(x) = x||1||0||0^*||1$, where $0^*$ means to append as many 0 so $|\mathrm{padd}(x)|$ is multiple of 5. For example,

$$\mathrm{padd}('') = 10001 \ , \ \mathrm{padd}('11111') = 1111110001 \ , \ \mathrm{padd}('111') = 1111000001$$

Then parse $\mathrm{padd}(x) = x_1x_2\cdots x_n$ with blocks of 5 bits and hash $x$ by using Merkle-Damgård scheme, by setting $IV = 01010$.

1) hash $x =''$, $x =' 0'$ and $x =' 00'$.

2) Find collisions.

https://en.bitcoin.it/wiki/RIPEMD-160

### 7.2.2 Permutations & Sponge

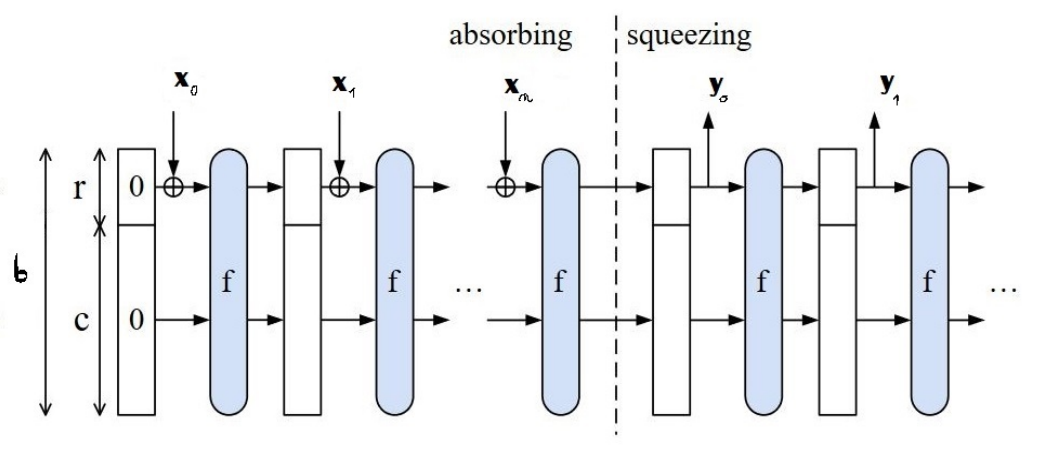
In this approach, one designs a permutation $f$ on $b = r + c$ bits and uses it in the sponge construction to build the sponge function $F$. In addition, one makes a flat sponge claim on $F$ with a claimed capacity equal to the capacity used in the sponge construction, namely $c_{\text{claim}} = c$. In other words, the claim states that the best attacks on $F$ must be generic attacks. Hence, $c_{\text{claim}} = c$ means that any attack on $F$ with expected complexity below $2^{c/2}$ implies a structural distinguisher on $f$, and the design of the permutation must therefore avoid such distinguishers.

In the hermetic sponge strategy, the capacity determines the claimed level of security, and one can trade claimed security for speed by increasing the capacity $c$ and decreasing the bitrate $r$ accordingly, or vice-versa.

**Security Level [BDPA11, page 9]**

Here the Sponge construction:

**Hash: Sponge construction**




L'input $x$ is divided in blocks $x_i$ of $r$ bits (there is a padding).

The number $b = r + c$ is called *width* , sum of he *capacity* $c$ and the *bitrate* $r$. The state of the sponge is the content of a register of $b$ bits.

The $f$ is a permutation of the state, i.e. a $\mathbb{Z}_2^b$.

There are two phases: **absorbing** followed of **squeezing**.

**absorbing:** the input $r$-bits of block $x_i$'s are xored bit the $r$ bits of the state. Then $f$ is used to permute the state. This is done until all blocks are "absorbed".

**squeezing:** the output is formed $y = y_0||y_1||\cdots$ by using the $r$ bits $y_i$'s of the state.

The last $c$ bits of the state are not altered by the blocks $x_i$'s are NOT part of the output.

**Exercise 7.2.6**

The permutation $f$ is not required to be one-way. So why hashing with a sponge should be preimage resistant? Assume that $r = 512$, $b = 1600$ and that after padding your message $M$ is $x_0$. So the hash digest is $y_0$. Write a loop to find a second preimage of $y_0$. How many cycles are expected?

**Exercise 7.2.7**

Let **Hash** be the hash function with digest of 3 bits, obtained via the sponge by using the following permutation $f : \mathbb{Z}_2^b \to \mathbb{Z}_2^b$:

$$f(b_1, b_2, \cdots, b_5) = (b_5, b_1, b_2, b_3, b_4)$$

So the register has $b = 5$ bits. Assume the bit rate $r = 3$ and that the padding is trivial i.e. the zeros bits are appended to $M$ to make its length a multiple of the bit rate 3 even in case the length of $M$ is already multiple of 3. For example, if $M = [101]$ then $M$ is padded to $[101000]$.

1) Find a collision.

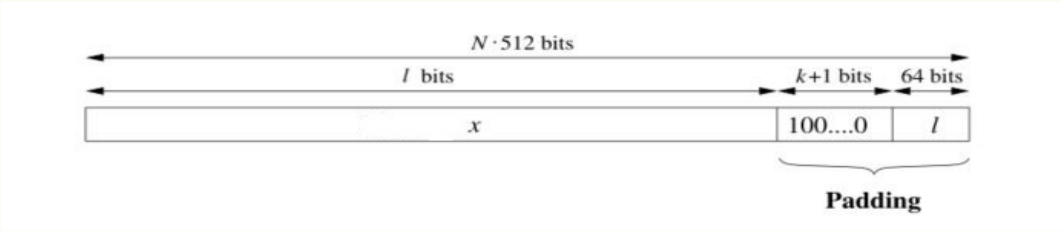
2) Find two preimages of the digest $[001]$.

**Exercise 7.2.8**

Let **Hash** be the hash function with digest of 512 bit obtained by using the permutation $f : \mathbb{Z}_2^{1024} \to \mathbb{Z}_2^{1024}$ by using the sponge with $b = 1024, r = 512$:

$$f(b_1, b_2, b_3, \cdots, b_{1024}) = (b_{1024}, b_1, b_2, \cdots, b_{1023})$$

Here is the padding:




1) Find a collision.

2) Find two preimages of the zero digest $[000 \cdots 0]$.

## 7.3 Bit Commitment

[Schneier15, page 86]

Stockbroker Alice wants to convince investor Bob that her method of picking winning stocks is sound.

BOB: "Pick five stocks for me. If they are all winners, I'll give you my business."
ALICE: "If I pick five stocks for you, you could invest in them without paying me. Why don't I show you the stocks I picked last month?"
BOB: "How do I know you didn't change last month's picks after you knew their outcome? If you tell me your picks now, I'll know that you can't change them. I won't invest in those stocks until after I've purchased your method. Trust me."
ALICE: "I'd rather show you my picks from last month. I didn't change them. Trust me."

Alice wants to commit to a prediction (i.e., a bit or series of bits) but does not want to reveal her prediction until sometime later. Bob, on the other hand, wants to make sure that Alice cannot change her mind after she has committed to her prediction.

A **commitment** scheme is a protocol for **Alice** and **Bob** with two different phases and algorithm $\mathsf{Com}$.

- ("commitment phase") **Alice** has a secret $b$ to commit and send to **Bob** the commitment value $c$ (commitment for $b$). The value $c$ is the output $c = \mathsf{Com}(b, r)$ of the algorithm $\mathsf{Com}$; $r$ indicates that $c$ is computed by $b$ and some random value $r$.
- ("opening/reveal phase") **Alice** send to **Bob** the former secret $b$ so **Bob** can check $c = \mathsf{Com}(b, r)$ for $r$.

A commitment scheme is secure if:

1. ("hiding property") At the end of the first phase, **Bob** (even dishonest) does not has any information about $b$ .
2. ("binding property") For a given $c$ , **Alice** (even dishonest) there is a unique value of $b$ that convince **Bob**.

**Exercise 7.3.1**

(cryptographic coin flipping) By using a commitment scheme for a single bit ($b \in \{0,1\}$) construct a protocol for the problem of "flipping a coin by telephone". That is to say, **Alice** and **Bob** want to flip a coin by telephone. (They have just divorced, live in different cities, want to decide who gets the car.) **Bob** would not like to tell **Alice** HEADS and hear **Alice** (at the other end of the line) say "Here goes...I'm flipping the coin... You lost !"

Hint: start with some naive protocol e.g. **Alice** choose a random bit $b_A$ and send it to **Bob**. **Bob** do the same and choose a random bit $b_B$ and send it to **Alice**. The output bit for both is $b = b_A \oplus b_B$.

**Exercise 7.3.2**

Construct a commitment algorithm Com by using a **Hash** random oracle.

## 7.4 MAC : Message Authentication Codes

A MAC is a symmetric algorithm that allows to check the authenticity of the message.

### 7.4.1 Message Authentication Code (MAC)

A MAC consists of three algorithms:

- $\mathsf{Gen}(n)$: the input $n$ is a security parameter and the output is a key $\mathsf{k}$ of $n$ bits (*key-generation*).
- $\mathsf{Mac}_{\mathsf{k}}(m)$: input a message $m$ and a key $\mathsf{k}$. The output is a tag $\mathbf{t}$ (*tag-generation*).
- $\mathsf{Vrfy}_{\mathsf{k}}(m, \mathbf{t})$: input a message $m$, a key $\mathsf{k}$ and a tag $\mathbf{t}$; output 1 for "valid tag" or 0 per "invalid tag" (*verification*).

A MAC is secure if an adversary who do not knows $\mathsf{k}$, but knows some valid tags $(m_1, \mathbf{t}_1) \dots (m_\ell, \mathbf{t}_\ell)$, it is not able to produce a valid pair $(m, \mathbf{t})$ where $\mathbf{t}$ is valid for the message $m$, with of course $m \neq m_i \; \forall \, i$.

**Exercise 7.4.2**

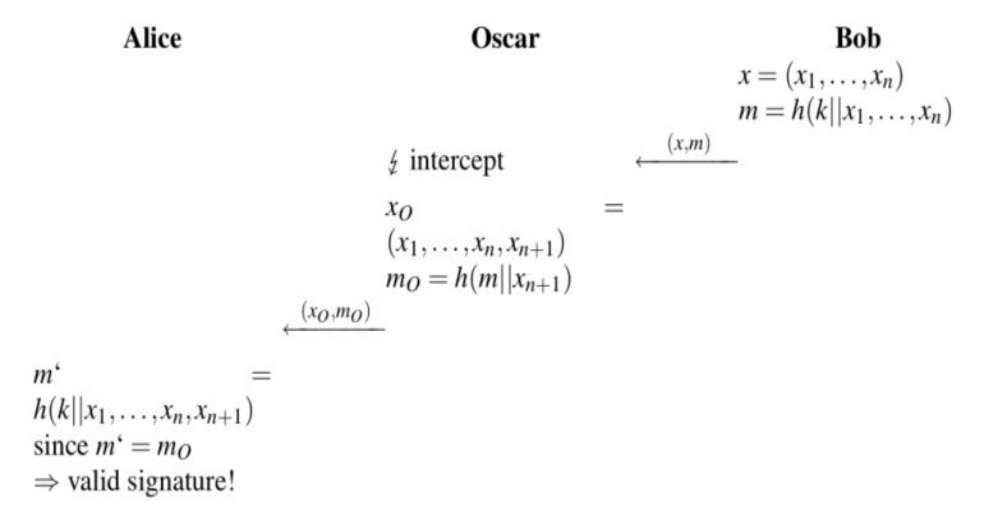
Let $\mathsf{Enc}_{\mathsf{k}}(B)$ be a block cipher. Consider the following MAC, are they secure? if not explain an attack.

1. The tag of the message $m = B_1||B_2$ with is $\mathbf{t} = \mathsf{Enc}_{\mathsf{k}}(B_1) \oplus \mathsf{Enc}_{\mathsf{k}}(B_2)$.
2. The tag of $m = B_1||B_2$ is $\mathbf{t} = \mathsf{Enc}_{\mathsf{k}}([1]||B_1) \oplus \mathsf{Enc}_{\mathsf{k}}([2]||B_2)$ where $[i]$ is the bit string of $i$ in base 2.
3. The tag for $m = B_1$ is $\mathbf{t} = (r||t')$ where $t' = \mathsf{Enc}_{\mathsf{k}}(r) \oplus \mathsf{Enc}_{\mathsf{k}}(B_1)$ where $r$ is a random bit string.

### 7.4.1 Attack: Length extensions & HMAC

**NOTE 7.4.3**

A naive way to get a MAC is by using a hash function: Given k and P a tag for the message P is just $\mathbf{t} = \mathbf{Hash}(\mathsf{k} \parallel \mathrm{P})$.

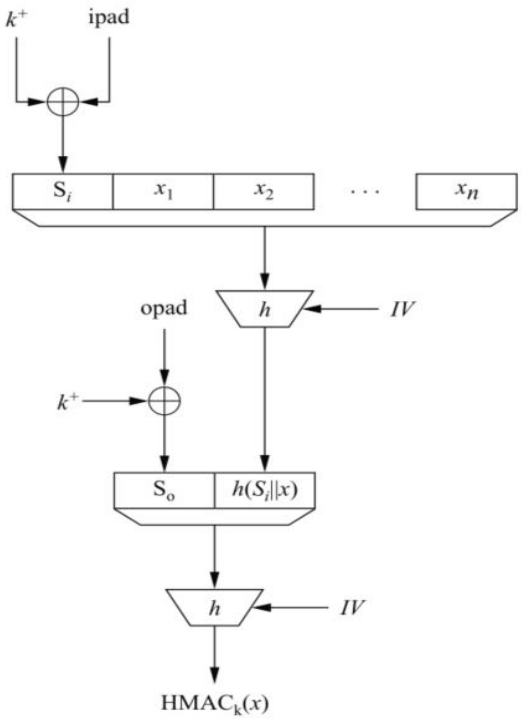



So the above naive MAC it is not secure if the **Hash** is designed with the Merkle–Damgård . This is a special case of a more general Lenght extension attack.

This design problem of a **Hash** is fixed by using HMAC.

HMAC

A hash-based message authentication code which does not show the security weakness described above is the HMAC construction proposed by Mihir Bellare, Ran Canetti and Hugo Krawczyk in 1996. The scheme consists of an inner and outer hash and is visualized in Figure 12.2.




where $k^+ = \underbrace{00\cdots00k}_{blocksize}$ and $\begin{cases} \text{ipad} = 0\text{x}36 \times \text{blocksize} \\ \text{opad} = 0\text{x}5c \times \text{blocksize} \end{cases}$ are two constants.
Here it is as an equation:

$$\mathsf{HMAC}_k(x) = h[k^+ \oplus \text{opad}||h[(k^+ \oplus \text{ipad})||x]]$$

A major advantage of the HMAC construction is that there exists a *proof of security* . This means that if the resulting MAC is not secure then the hash $h$ is not secure.

## 7.5 Bibliography

Books I used to prepare this note:

[Aumasson18] Jean-Philippe Aumasson, *Serious Cryptography: A Practical Introduction to Modern Encryption*, No Starch Press, 2018.

[KatLin15] Jonathan Katz; Yehuda Lindell, *Introduction to Modern Cryptography* Second Edition, Chapman & Hall/CRC, Taylor & Francis Group, 2015.

[Paar10] Paar, Christof, Pelzl, Jan, *Understanding Cryptography, A Textbook for Students and Practitioners*, Springer-Verlag, 2010.

[Schneier15] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C* ,Wiley; 20th Anniversary edition,2015.

Here a list of papers:

[BDPA11] Bertoni,G.;Daemen,J.;Peeters,M. and Van Assche, G.: *The Keccak reference*, https://keccak.team/files/Keccak-reference-3.0.pdf

[BDPA11b] Bertoni,G.;Daemen,J.;Peeters,M. and Van Assche, G.: *Cryptographic sponge functions*, https://keccak.team/files/CSF-0.1.pdf

[BeRo95] Bellare, Mihir and Rogaway, Phillip; *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, Proceedings of the First Annual Conference on Computer and Communications Security https://cseweb.ucsd.edu/~mihir/papers/ro.pdf

[Ber07] Daniel J. Bernstein; *The Salsa20 family of stream ciphers*, https://cr.yp.to/snuffle/salsafamily-20071225.pdf

[Blum83] Manuel Blum; *Coin Flipping by Telephone a Protocol for Solving Impossible Problems*, SIGACT News, vol 15, number 1, January, Winter-Spring 1983, ACM, pages 23–27. https://dl.acm.org/citation.cfm?id=1008911

[DH76] Diffie, W.; Hellman, M. *New directions in cryptography*, (1976). IEEE Transactions on Information Theory. 22 (6): 644654.

and some interesting links:

http://www.nicolascourtois.com/papers/ga18/Lecture%20-%20DLOG%20and%20Factoring%20Algorithms.pdf

http://passwords12.at.ifi.uio.no/Joan_Daemen_Passwords12.pdf