Crypto 7

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

Contents

Euler ϕ and Chinese Remainder Theorem

$\phi(24)$ = # of remainders invertibles

$\phi(n) = \#\mathbb{Z}_n^*$

$\mathbb{Z}_n^* \subset \mathbb{Z}_n$

$n = p \cdot q$

$\mathbb{Z}_n$	$\mathbb{Z}_p$ $\mathbb{Z}_q$
0	$(0, 0)$
1	$(1, 1)$
2	
i	$(x, \tilde{x})$

Consequence of Lagrange's Theorem: Let G be a group and $H \subset G$ be a subgroup then $\#H \mid \#G$

$\forall a \in \mathbb{Z}_n^*$

$a^{\phi(n)} \equiv 1 \pmod{n}$

↑ Theorem/fact about ϕ!

n: p is prime

$\forall a \in \mathbb{Z}_p^*$

$a^{p-1} \equiv 1 \pmod{p}$

Fermat's little theorem

$a^p = a \pmod{p}$

$\forall a \in \mathbb{Z}_p$

$\phi(p \cdot q) = \phi(p) \cdot \phi(q)$

x invertible only if $\tilde{x}$ and $\tilde{\tilde{x}}$ are invertibles

$r = \phi(n)$

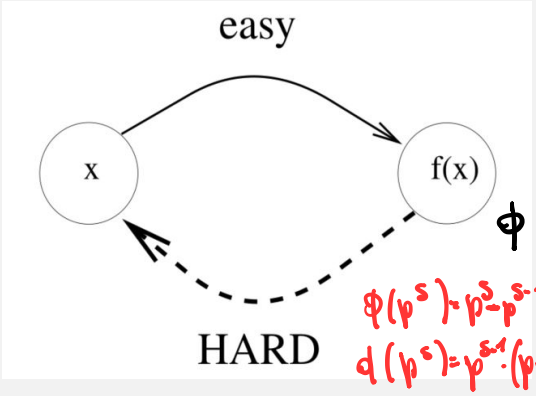
Notice $\phi(n) = \#\mathbb{Z}_n^*$

$H = \{a, a^2, \ldots, a^r = 1\}$

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 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 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

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

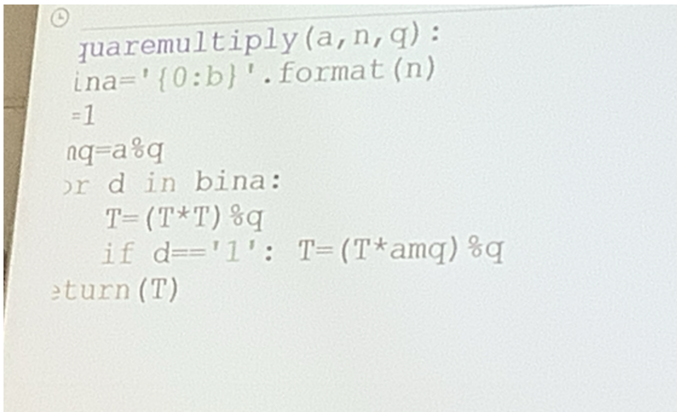
here the key k is regarded as a **trapdoor** i.e. the special information which turns to easy inversion.

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.

$g \in \mathbb{Z}_n$ g is invertible g is huge

$g, g^2, g^3, g^4, \dots, g^r = 1, g^{r+1} = g$

r is called order of g or period of g



```
juaremultiply(a,n,q):
ina='{0:b}'.format(n)
=1
nq=a%q
or d in bina:
    T=(T*T)%q
    if d=='1': T=(T*amq)%q
eturn(T)
```

$x = 1$

for $:1$ — for

$x = x \cdot q$ print(x)

print (x) $x = x^2$

q^{32}

32 iterations		6 iterations
q		q
q^2		q^2
q^3		$(q^2)^2$
q^4		$(q^4)^2$
$\vdots$		$(q^8)^2$
q^{32}		$(q^{16})^2 = q^{32}$

F.F.C.

Diffie - Hellman

g $\mathbb{Z}_p = \mathbb{F}_p$

$\begin{cases} sk \in \{0, 1, \dots, p-1\} \\ pk \quad g^{sk} \bmod p \end{cases}$

It's hard to recover sk from $g^{sk} = pk \pmod p$

$\Downarrow$

ONE-WAY FUNCTION:

$f(x) = g^x \pmod p$

f is ONEWAY

$f(x) = C$ (given) to find x such that

Discrete logarithm problem ⟶ $3^x = 581240126 \pmod{p}$

$p = 999345181$

$x = \log_3(\;) \pmod{p}$

32 bits

2^{32}

$\sqrt{2^{32}} = 2^{\frac{32}{2}} \approx 2^{16}$

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

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

$$\alpha^{a} \cdot \alpha^{xb} = \alpha^{A} \cdot \alpha^{xB} \longleftarrow \alpha^{a}\beta^{b} = \alpha^{A}\beta^{B}$$

then

$$\alpha^{a+bx} = \alpha^{A+xB} \pmod{p}$$

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

$$a + bx = A + xB \pmod{n}$$

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.

It is important to notice that the computation of α^{γ} requires $O(\log_2(\gamma))$ iterations:

Let $(d_t d_{t-1} d_{t-2} \cdots d_0)_2$, $d_t = 1$ be the binary representation of $\gamma \in \mathbb{N}$, notice $t = \lfloor \log_2(\gamma) \rfloor$.

7.1.4 Square-multiply

```
T=γ
For i=t-1 downto i=0
    T = T · T
    if d_i = 1
       T = T · γ
return(T)
```

The above algorithm is based on the following identity:

$$\alpha^{\gamma} = (((\alpha^2 \cdot \alpha^{d_{t-1}})^2 \cdot \alpha^{d_{t-2}})^2 \cdots)^2 \cdot \alpha^{d_0}$$

Here another versions:

discrete log problem

You know g, b and p such that:

$$g^x = \beta \pmod{p}$$

$\Rightarrow$ Problem: find "x"

Solution of (1) by using a collision

$$g^a \cdot \beta^b = g^A \cdot \beta^B \pmod{p} \quad (c) \qquad a, b, A, B \text{ such that}$$

1) How to find a, b, A, b

Collision

2) How to find "x"

Notice that c is:

$$F(a, b) = F(A, B) \leftarrow \text{is (c)}$$

where $F(X, Y) = g^X \cdot g^Y$

proceeds on pag 9

Exercise 7.1.5

Check that α^γ is the output of the following:

```
Y=α
X=1
For i=0 to i=t-1
    if d_i = 1
       X = X · Y
    Y = Y · Y
return(X · Y)
```

Hint:

$$\alpha^{2^t} \cdot (\alpha^{d_{t-1}2^{t-1}} \cdot (\cdots (\alpha^{d_2 2^2} \cdot (\alpha^{d_1 2} \cdot (\alpha^{d_0})))$$

The Montgomery ladder [Mo87] approach computes α^γ in a fixed amount of time. This can be beneficial when timing or power consumption measurements are exposed to an attacker performing a side-channel attack.

7.1.6 Montgomery's Ladder

```
X=α
Y=X^2
For i=t-1 downto i=0
    if d_i = 0 then
       Y = X · Y
       X = X · X
    else
       X = X · Y
       X = Y · Y
return(X)
```

Exercise 7.1.7

Find the page of Montgomery's paper Speeding the Pollard and Elliptic Curve Methods of Factorization with the explanation of the *Montgomery's Ladder* .

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 7.1.8

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.

7.1.9 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]

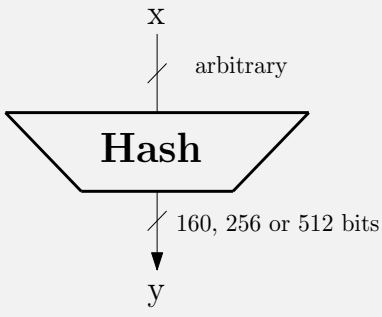
A funzione **Hash** has inputs in $\mathbb{Z}_2^*$ and output a *digest* of $n = 160, 256, 512$ bits:

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

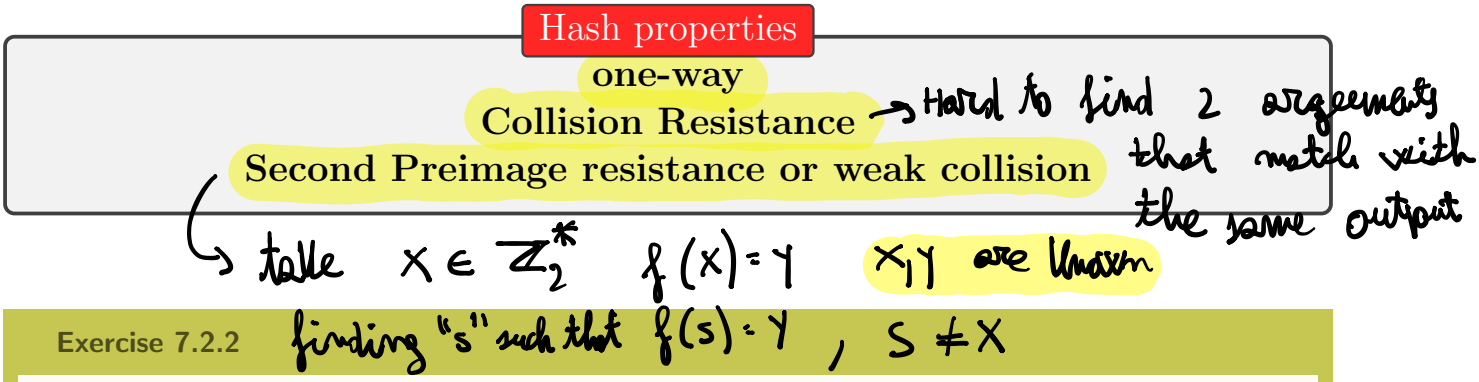
$\mathbb{Z}_2^*$ = all possible finite length string of bits

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)..

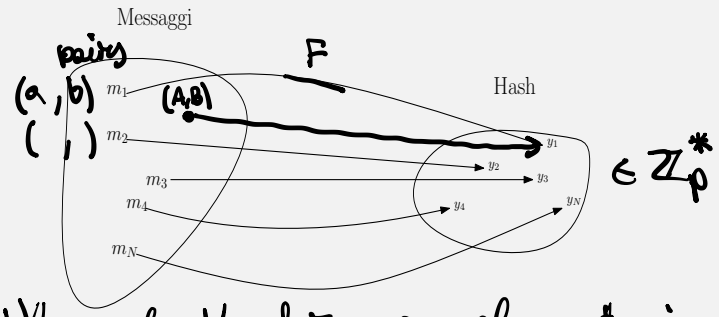


Exercise 7.2.2

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 λ_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.



We look for 2 elements in message having the same image in hash

Here the equation relating λ_N and the digest length:

n = # of bits of p $\quad \sqrt{p} \simeq N \approx 2^{\frac{n+1}{2}} \cdot \sqrt{\ln\left(\frac{1}{1-\lambda_N}\right)}$ $\quad n = \log_2 p, \; p \simeq 2^n$ $\quad \rightarrow$ Taylor dev. of exp. function

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:

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

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

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 ℓth bit of the input selects between messages of type 0 or type 1, and the ith bit (for $1 \le i \le \ell - 1$) selects between options for the ith 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.

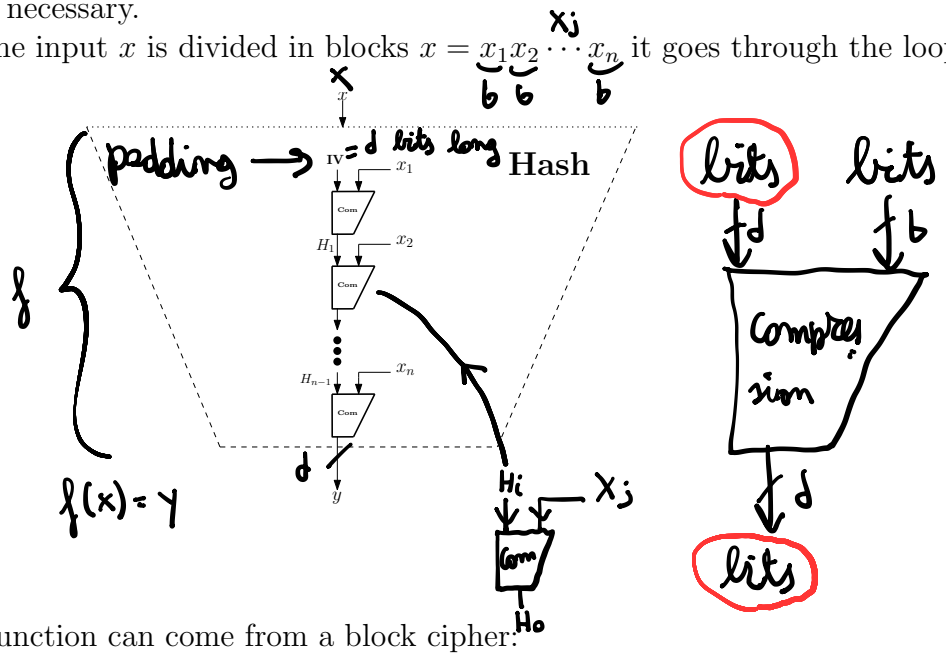
Now define $f : \{0,1\}^\ell \to \{0,1\}^\ell$ by $f(x) \stackrel{\text{def}}{=} H(g(x))$. Alice can find a collision in f using the small-space birthday attack shown earlier. The point here is that any collision x, x' in f yields two messages $g(x), g(x')$ that collide under H. If x, x' is a random 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.

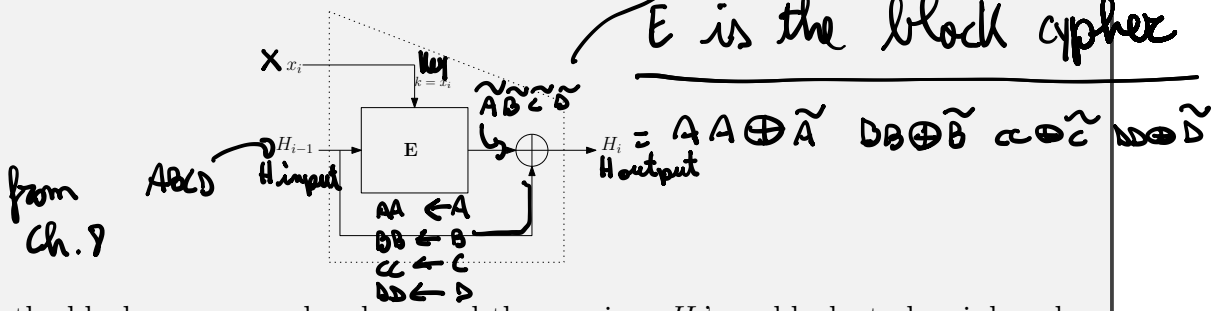
After a padding, the input x is divided in blocks $x = \underbrace{x_1}_{b}\underbrace{x_2}_{b}\cdots\underbrace{x_n}_{b}$ it goes through the loop:



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

* both injective and surjective

7.2.2 Permutations & Sponge

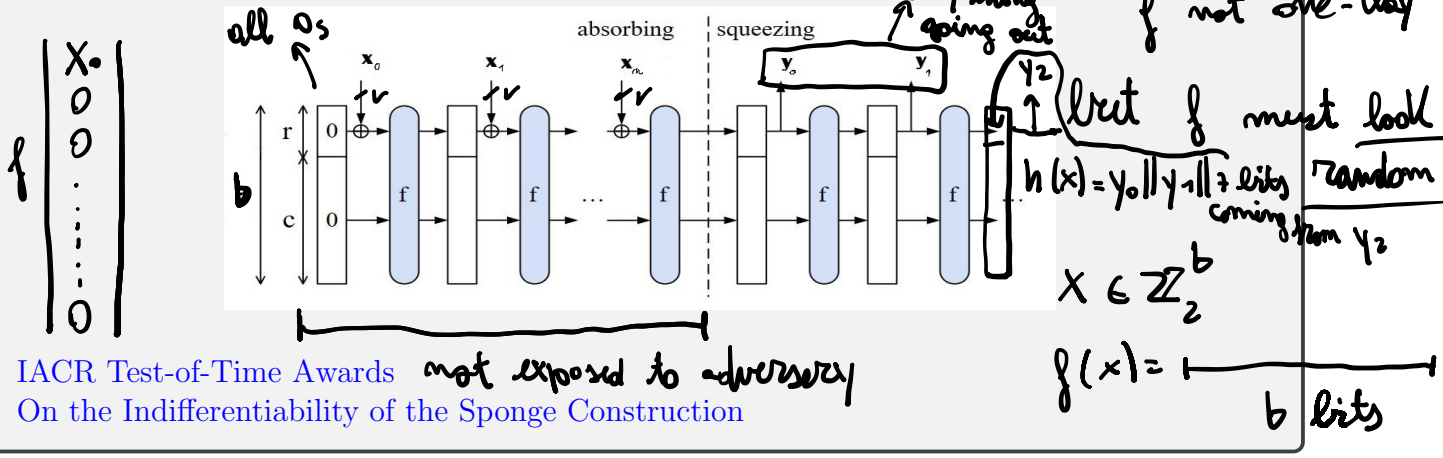
> In this approach, one designs a permutation f on $b = r + c$ bits and uses it in the sponge construction to build the sponge function F. In addition, one makes a flat sponge claim on F with a claimed capacity equal to the capacity used in the sponge construction, namely $c_{\text{claim}} = c$. In other 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]

$f: \mathbb{Z}_2^b \to \mathbb{Z}_2^b$ $\mathbb{Z}_2^b$ is the set of all strings of b bits.
A permutation is a bijection*
- $f: \mathbb{Z}_2^b \to \mathbb{Z}_2^b$
- f fast to compute

Here the Sponge construction:

Hash: Sponge construction



IACR Test-of-Time Awards
On the Indifferentiability of the 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.

$h(x)$ h is sponge based on the permutation f

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

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

$f\left(\begin{bmatrix}\cdot\\ \cdot\\ \cdot\end{bmatrix} \oplus x_1\right) \Rightarrow f\left(\oplus [x_i]\right)$

$h(x)$ x ↓ $x_0\ x_1\ x_2 \cdots x_m$ ↓ $h(x) =$ r_1 r_2

absorbing: the input r-bits of block x_i's are xored with the first 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 NEVER 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.

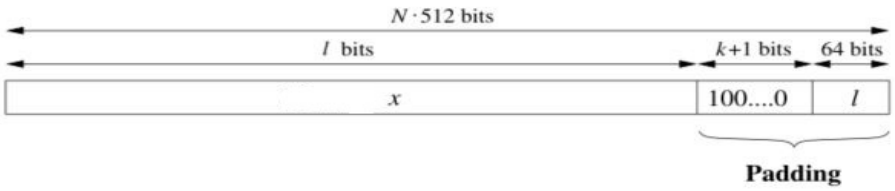
2) Find two preimages of the digest $[001]$.

Exercise 7.2.8

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

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

and using the sponge with $b = 1024, r = 512$ and the padding:



SHA1 Padding

b = 512

Same padding as MD4

MD4

1) Find a collision.

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

7.3 Commitment schemes

7.3.1 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 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 Com; so c is computed by b and some random value r.
- ("opening/reveal phase") **Alice** send to **Bob** the former secret b and r, so **Bob** can check $c = \mathsf{Com}(b, r)$.

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.

7.3.2 Pedersen's Commitment (by using a Schnorr group)

I thanks to A. Guggino who asked me Pedersen's Commitment.

[P92, Section 3]

3 The Commitment Scheme

This section describes a commitment scheme, which is very similar to that of [BCP]. The only difference is in the choice of g and h.

Let g and h be elements of G_q such that nobody knows $\log_g h$. These elements can either be chosen by a trusted center, when the system is initialized, or by (some of) the participants using a coin-flipping protocol.

The committer commits himself to an $s \in \mathbb{Z}_q$ by choosing $t \in \mathbb{Z}_q$ at random and computing

$$E(s,t) = g^s h^t.$$

Such a commitment can later be opened by revealing s and t. The following theorem is very easy to prove and shows that $E(s,t)$ reveals no information about s, and that the committer cannot open a commitment to s as $s' \neq s$ unless he can find $\log_g(h)$.

Theorem 3.1
For any $s \in \mathbb{Z}_q$ and for randomly uniformly chosen $t \in \mathbb{Z}_q$, $E(s,t)$ is uniformly distributed in G_q.
If $s, s' \in \mathbb{Z}_q$ satisfies $s \neq s'$ and $E(s,t) = E(s',t')$, then $t \neq t' \bmod q$ and

$$\log_g h = \frac{s - s'}{t' - t} \bmod q.$$

Even though it will not be used in the following we mention that it is quite easy to prove one's ability to open two commitments as the same value without revealing this value. Let namely

$$\beta = E(s,t) \quad \text{and} \quad \beta' = E(s,t')$$

where $t \neq t'$. Anyone who knows an r such that $\beta/\beta' = h^r$ can open β as s if and only if he can also open β' as s. By revealing $r = t - t'$ it is therefore possible to prove equality of the contents of two commitments. Furthermore, $t - t'$ does not contain any information about s. It is not clear how to prove efficiently, that commitments to two different values really do contain different values. In particular, the proof of [BCC88] that two blobs contain different bits given a method of proving equality does not generalize to this commitment scheme.

Finally consider the efficiency of the commitment scheme. If p and q are constructed by first choosing q and then determining p as the first prime congruent to 1 mod q, heuristics show that $p \leq q(\log q)^2$ (see [Wag79]). Thus a commitment to $|q|$ bits requires at most $|q| + 2\log|q|$ bits. Furthermore, by first computing the product gh a commitment to s can be done in less than $2|q|$ multiplications modulo p or less than two multiplications pr. bit of s. Thus the commitment scheme is quite efficient with respect to the size of commitments as well as the computation required.

Both the hiding and binding properties follows form the hardness of DLP on the subgroup G_q . Indeed, if Bob can get any s from $E(s,t)$ then he can solve DLP $y = g^x$ getting x from $E(x,0) = y$. Alice can not cheat otherwise she would we able to construct a collision $E(s,t) = E(s',t')$ hence to compute the DL k of $h = g^k$.

https://crypto.stackexchange.com/questions/64437/what-is-a-pedersen-commitment

NOTE 7.3.3

The subgroup $G_q \subset \mathbb{Z}_p^*$ of order q is a so called Schnorr group. Namely, a large subgroup of $\mathbb{Z}_p^*$ where $p = qr + 1$ with p, q primes. Notice $\mathbb{Z}_p^*$ has order $p - 1 = qr$. Usually $r = 2$.
A generator $g \neq 1 \pmod p$ of G_q of the form $g = h^r \pmod p$ is constructed by searching h in the range $1 < h < p$. Then such g has order q.

Exercise 7.3.4

Why g has order q ?

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 k of n bits (*key-generation*).
- $\mathsf{Mac}_{\mathsf{k}}(m)$: input a message m and a key k. The output is a tag $\mathbf{t}$ (*tag-generation*).
- $\mathsf{Vrfy}_{\mathsf{k}}(m, \mathbf{t})$: input a message m, a key 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 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

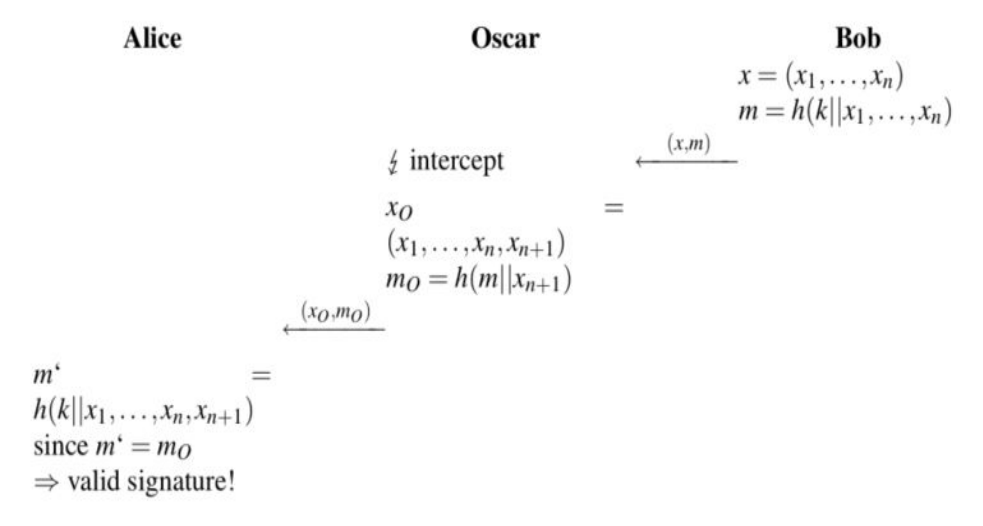
Let $\mathsf{Enc}_{\mathsf{k}}(B)$ be a block cipher. Consider the following MAC, are they secure? if not explain an attack.

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 $\mathbf{t} = \mathbf{Hash}(\mathsf{k} \parallel \mathrm{P})$.

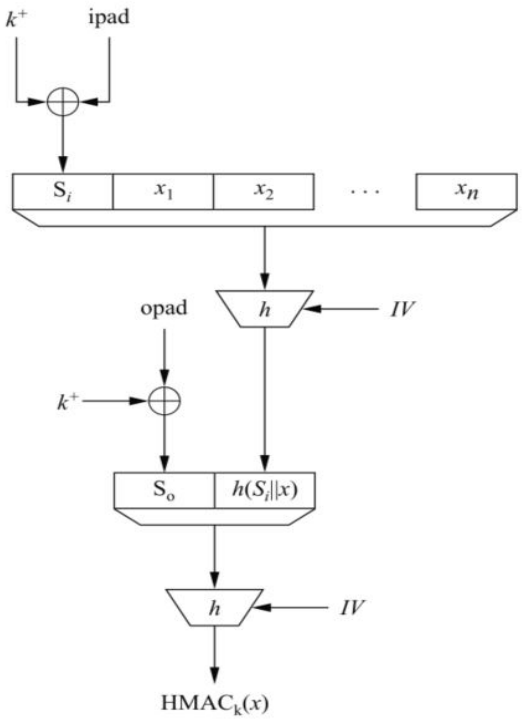


So the above naive MAC it is not secure if the **Hash** is designed with the Merkle–Damgård . This is a special case of a more general Lenght extension attack.

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.

[Mo87] Montgomery, Peter L. *Speeding the Pollard and elliptic curve methods of factorization*, Math. Comp. 48 (177): 243264. https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866113-7/

[P92] Pedersen, T.P. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*, In: Feigenbaum, J. (eds) Advances in Cryptology CRYPTO 91. CRYPTO 1991. Lecture Notes in Computer Science, vol 576. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-46766-1_9

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