# Introduction to Cryptography - Summary

## Lecture 1 - Introduction

**Definition Confidentiality (Data Privacy):**

The assurance that data cannot be viewed by an unauthorised party.

**Definition Data Integrity:**

The assurance that data has not been modified in an unauthorised manner.

**Definition Data Origin Authentication:**

The assurance that a given entity was the *original source* of received data.

**Definition Entity Authentication:**

The assurance that a given entity is who they claim to be.

**Definition Non-Repudiation:**

The assurance that a person cannot deny a previous commitment or action. Often realized by contract, law or directive rather than cryptography.

Basic Data Confidentiality is to protect people's privacy, company assets, enforcing business model, PIN, password, **cryptographic keys**. This can be achieved by **encryption**. To achieve this the sender and receiver need to establish a shared secret key.
Encryption does not provide integrity so no authentication. To ensure integrity of a message a **Message Authentication Code (MAC)** is used. This is a lightweight cryptographic operation. Requires prover and verifier to establish a shared secret key. A signature: cryptographic counterpart of real-life signing. Verifier only needs the public key of the signer. Requires verified to authenticate signer's ownership of the public key. Reasons to use signature rather than MAC: (1) auth. of broadcast messages, e.g. software updates. (2) signature as evidence for a judge (non-repudiation). (3) if the verifier is not known in advance.

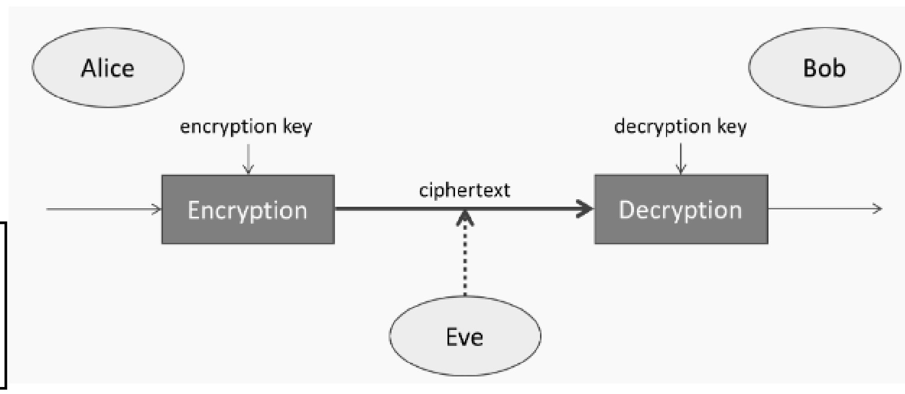
**Definition Freshness:**

Entity is there **now**. The received message was **recently** written**.** Mechanism: include **unpredictable challenge** in MAC/Signature computation. Unpredictable challenge must come from the verifier.

Protection against a replay attack: authenticated message was not just a copy from an earlier one. Mechanism: include **nonce** in MAC/Signature computation and the verifier must check the uniqueness of the nonce.
**Forward secrecy:** compromise of endpoint does not jeopardize confidentiality of old communications.
A **secure channel** is a cryptographically secured link between two entities. This provides data confidentiality and authentication. There is also session-level authentication (insertion, removal, shuffling of message). The secure channel can be one-directional or full-duplex. Also possible to have the secure channel online or store-and-forward. Can require freshness or just protection against replay attack.

Classical encryption use case:

| **Definition Adversary Model:** |
| --- |
| Specification of what we assume an adversary (eve) can do and access |




Understand security goals that a scheme/protocol should meet:

(1) Define the adversary model
   (a) What is the adversary's goal?
   (b) What is the adversary's power?
   (c) This defines the requirements the solution must meet
   (d) Verify that the adversary model fits the application
(2) Express a solution (protocol or scheme) that address the requirements
   (a) Use constructions and modes that allow to reduce the requirements on the construction to that of primitives
   (b) Show that an adversary cannot break the scheme without breaking the underlying primitive
   (c) Use primitives that are believed to satisfy those requirements

Trust in cryptographic primitive depends on:

- Reputation of designers
- Perceived simplicity
- Perceived amount of analytic effort inverted in it
- Reputation of cryptanalysts

| **Definition Security Claim:** |
| --- |
| Precise statement on expected security of a cryptographic primitive. |

Security claim serves as a challenge for cryptanalysts: if they break it, it means they performed an attack better than the claim. And serves as a security specification for users (as long as it is not broken). It's not about the scheme is impossible to break but rather about

- Success probability of breaking the primitive by an adversary with the following well-define resources:
    - **N**: amount of computation, in some well-specified unit
    - **M**: Amount of input/output commuted with the secret key

**Definition Security Strength:**

A cryptographic scheme offers security strength **s** (bits) if there are no attacks with (M+N)/p < $2^s$ with N and M the adversary's resources and p the success probability.

As reference:

- 56 bits: not secure
- 80 bits: lightweight
- 96 bits: solid
- 128 bits: secure for the foreseeable future
- 256 bits: for the clueless

**Definition N - amount of computation:**

- Computational complexity
- Time complexity (as it typically spends time on a CPU)
- Offline complexity (offline from attacked instance)

The only limit to **N** is the wealth of the attacker

**Definition M - amount of input/output computed with the secret key:**

- Data complexity (data as obtained from the attacked instance)
- Online complexity (online with attacked instance)

Can be limited by designing protocols in a smart way

Security strength often makes an abstraction of distinction between these two very different complexities.

# Lecture 2 - Stream Ciphers (part 1)

**Definition Integers:**

{...,-4, -3, -2, -1, 0, 1, 2, 3, 4,...} form the set of integers $\mathbb{Z}$

**Definition Modular equivalence of integers:**

A,b $\in \mathbb{Z}$ are congruent modulo n $\in \mathbb{N}$ if a - b is divisible by n (e.g. 1 ≡ 13 (mod 12))

**Reduction modulo** n of an integer returns its equival in the interval [0, n-1] (c <- a mod n where c is the remainder after division of a by n). **Addition modulo** n as an operation gives c <- a + b if c >= n, c <- c - n. The notation is a + b mod n or just a + b. **Multiplication modulo** n as an operation gives c <- a * b, do the result module n: c <- c mod n. The notation is a * b mod n or just a * b. Addition and multiplication is ℤ/nℤ.

The one-time pad add elements of ℤ/2ℤ mod 2. See the image below for an example. Here M is the message, K the key and C the encrypted message.

$$
\begin{array}{rccccccccccccccccl}
M = & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & \\
K = & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \bigoplus \\
\hline
C = & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 &
\end{array}
$$

One-time pad gives perfect secrecy if (1) the key **has the same length as all plaintext together** (2) the adversary **has no information about the key bits.**

| **Definition Stream Encryption:** |
|---|
| Encryption where a **keystream** is bitwise added to plaintext |

Often ℤ/26ℤ is used since the alphabet is 26 long. **The main point is that the encryption is a simple symbol-by-symbol operation.**

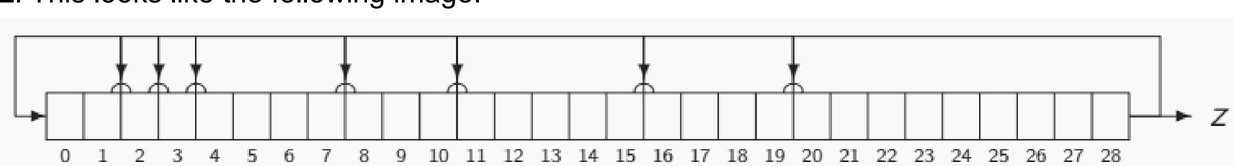
| **Definition Stream Cipher:** |
|---|
| Algorithm to convert a short key K into a long keysteram Z |

Vigenère cipher:
- K: a password e.g. LEMON
- Z: K repeated all over, LEMONLEMONLEMONLEMONL…
- Addition module 26 gives ciphertext.

This is compact and efficient but problems are:
- Knowledge of short plaintext sequence reveals full keystream: *known plaintext attack*
- Long ciphertext enciphered leak via letter frequencies: *ciphertext-only attack*

Therefore not used often anymore.

Linear feedback shift register (LFSR) its goal is to efficiently generate non-repeating sequence Z. This looks like the following image:




Mechanism:
- Circuit with state s that is regularly clocked
- Each cell contains a bit $s_i$

- Each clock cycle: cells move right $s_{i+1}$ <- $s_i$
- ...for some positions there are feedback taps: e.g. $s_{i+1}$ <- $s_i + 2_{28}$
- Rightmost cell is output: z <- $s_{28}$
- Cycle as long as your output has to be.

Maximum-length LFSR: if feedback taps are well chosen, cycle length is $2^n - 1$
LFSR features are that they are very simple to implement (just a shift and some XORs), keystream has good local statistical properties and bits of Z satisfy recurrence relation.

| **Distinction between algorithm and key:** |
|---|
| Public algorithms AKA cipher: LFSR length and tap positions. Security should be based on secrecy of K (Kerckhoffs principle). |

**Attacks on LFSR: Exhaustive Key Search:**
Setting: adversary gets C and C = P + Z with a P a meaningful plaintext (ciphertext-only attack).

- Make a guess K' for the value of K
- Generate the corresponding keystream Z'
- Compute P' = C + Z' and check if P' is meaningful
- If so, you're done. Otherwise try a different K'.

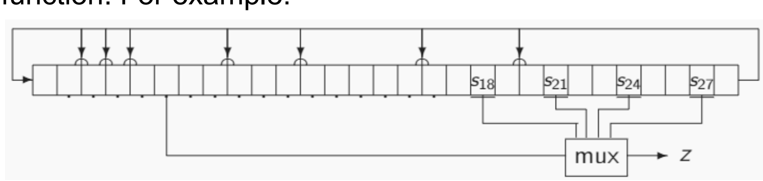
Implications: for k-bit key, probability to find key after N guesses: $N2^{-k}$. **Generically** true for any cipher if the adversary has >= k output bits.
**Security strength s of a cipher with a k-bit key is at most k.**

**Attacks on LFSR: state reconstruction using linear algebra:**
Setting: adversary can obtain n subsequent bits of keystream $z_t$ (known plaintext attack)
When you have n keystream bits this allows for reconstruction of the full state. Possible countermeasure: decimate the keystream. This also doesn't work due to linearity of LFSR.

| **Definition Linearity:** |
|---|
| A function f is linear (over $\mathbb{Z}/n\mathbb{Z}$) if f(x + y) = f(x) + f(y). If $f_1$ and $f_2$ are linear, so is $f_2 \circ f_1$. |

Important to realize changing to for example a Matrix for the LFSR, this also doesn't work since it's still linear. Because of linear algebra all of these can be broken. Therefore it holds that:
**Purely linear ciphers offer no security.**

**Filtered LFSR introduces a non-linear output function**. Instead of using a LFSR statebit as a keystream bit $z_t = s^t_{n-1}$ compute z as a function of statebits: $z = f(s_0,...s_{n-1})$ with f a nonlinear function. For example:

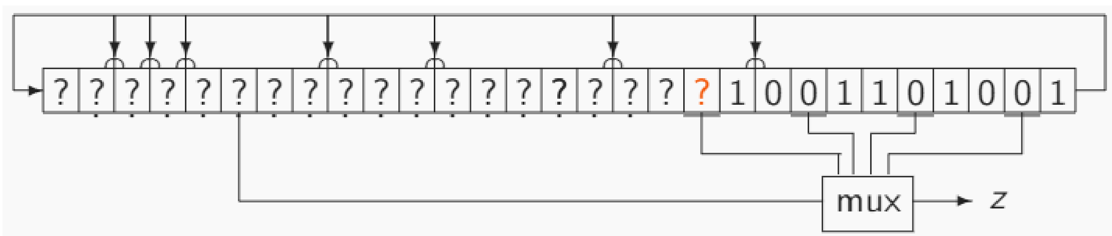



This gives uncertainty on where the output bit comes from and therefore complicates attacks. Attacks are still possible but require more sophistication.

**Filtered LFSR: guess-and-determine attack:**
Setting: adversary can obtain n subsequent bits of keystream $z_t$ (known plaintext attack)

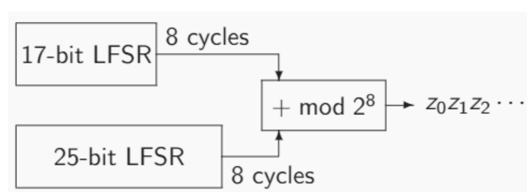
- Make a guess for a subset of the bits of the state
- Combined with output Z, this determines other statebits.

This will be faster than exhaustive key search.
**Recursive algorithm for the mux LFSR:**




- Starting for all possible values of rightmost 10 bits
- For two guesses of the bit in position indicated with “?”
    - Use output z to determine the statebit chosen by the mux
    - If contradiction, cut this branch
    - Else, fill in in the LFSR and repeat procedure
- Tree search where each node has at most two children
    - Only one child is the value of “?” is known
    - No children if contradiction
- LFSR state with all bits known and no contradiction: ready!

**Combiner LFSR:** non-linear output function taking bits from several LFSRs.
**Divide-and-conquer attack:**
Setting: adversary has Z (known plaintext attack).




- Guess state of top LFSR
- Each byte $z_i$ allows reconstructing output byte of bottom LFSR
- 4 output bytes $z_t$ give 32 output bits of bottom LFSR
- Should satisfy recurrence relationship
- Total complexity: some subtranctions module $2^8$ and checking recurrence relation for about $2^{16}$ guesses.

# Lecture 3 - Stream Ciphers (part 2)

**Modern stream ciphers**

Modern stream ciphers take a key K and a diversifier D as input. So the same cipher key can now be used to generate many keystreams.

Message encryption:

- Have a system that generates a unique diversifier D per message (e.g. data/time)
- Encipher message with keystream Z from K and D

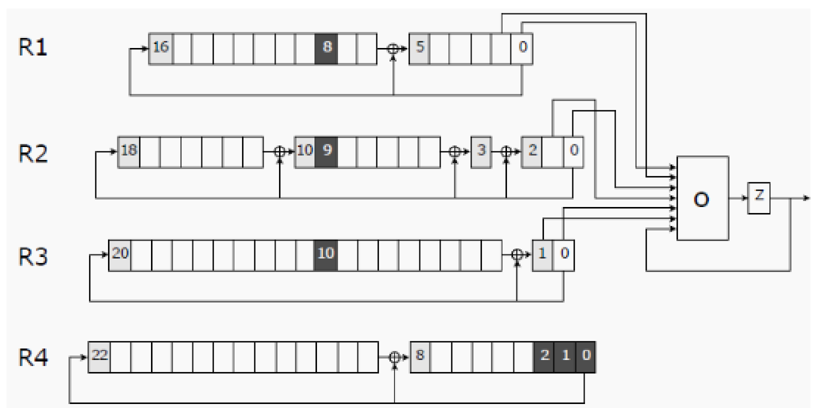
Stronger stream ciphers:

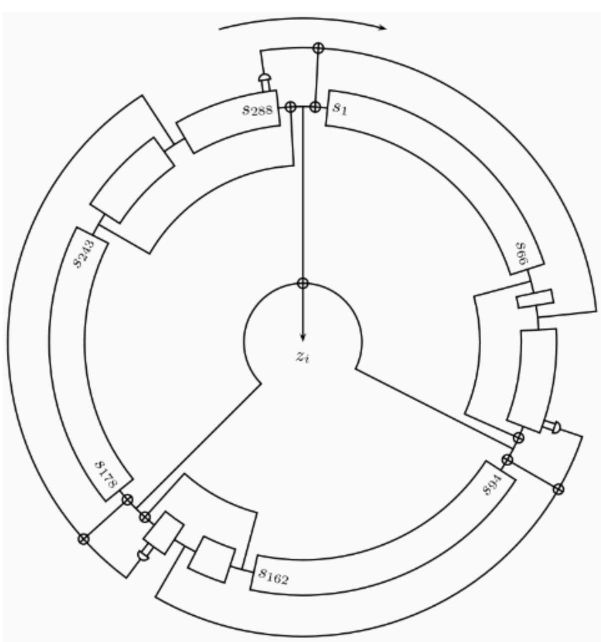
(1) Introduce non-linearity in state updating function
    (a) Irregular clocking: let # LFSR cycles depend on state bit values
    (b) Make recursion formula non-linear
(2) After writing D and K in state, do black cycles (no output)
    (a) Non-linearity from D and K to $s_t$ is weak for small t
    (b) But increases fast with growing t
    (c) note : requires state updating function to be non-linear
(3) Make output function stronger

Alternative approach: build stream cipher from a string cryptographic primitive (e.g. block cipher).

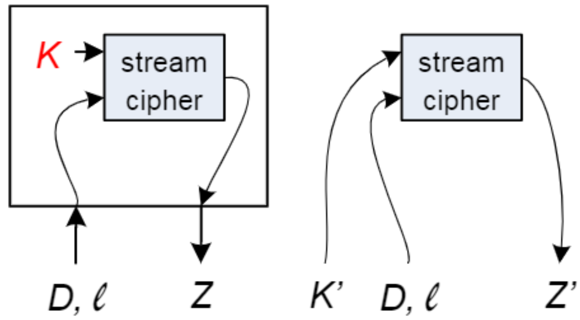
Irregular clocked LFSR: DECT Stream Cipher:

- 4 maximum-length LFSRs with coprime lengths
- Top 3 clocked 2 or 3 times in between time steps t
- Bottom LFSR determines clocking of top 3 ones
- Output function O with 1 bit of memory
- Practically broken with statistical key recovery attack




- Claims 80 bits of security
- 80-bit K and 80-bit D
- 288-bit state
- Linear output function
- Regularly clocked
- Non-linearity in update: only 3 AND gates
- Output period not known in advance but likely OK
- init. Takes 1152 cycles
- as yet unbroken




Adversary has query access to:

$SC_K$: stream cipher instance with unknown key K

$SC_k$' = stream cipher instance with chosen key K'

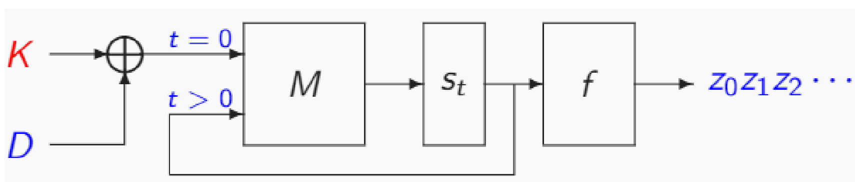
Can make queries Q

$Q_d$: queries to $SC_K$ with cost (e.g. total length) M

$Q_c$: queries to $SC_k$' with cost N

Express probability of success as function of M and N. Example: generic exhaustive key search: Pr(success) = $N2^{-|k|}$ with N number of key-trail queries to $SC_k$'.

**Iterative stream ciphers internal structure:**

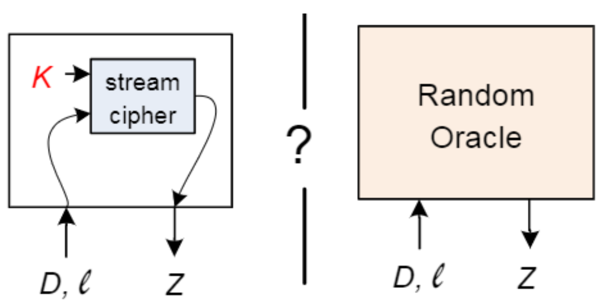



Operates on an evolving state $s_t$.
State update function: $s_t$ = M($s_t$-1).
Output function: $z_t$ = f($s_t$)

A Stream Cipher is considered secure if **no attacks with success probability above the one in claim.**

The ideal cryptographic function is the **Random Oracle (RO)**.

| **Definition Random Oracle:** |
| --- |
| A cryptographic function in which every input gives a different unique non related output. And for input m and m' where m=m', the output is the same. (Only used to reason about crypto functions) |




**Black box model:**

Adversary A has query access to a system that is either:

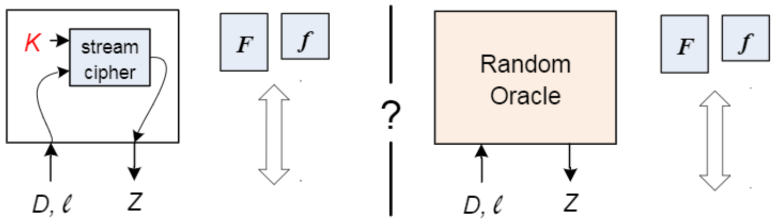
- $SC_k$: stream cipher with unknown key K
- RO: ideal stream cipher in form of a random oracle

A does not know which one and has to guess. A is actually an attack algorithm that returns either 1 if it estimates the system is $SC_k$ and 0 if it estimates the system is RO.

- Pr(A = 1 | $SC_k$): probability that A return 1 in case of $SC_k$
- Pr(A = 1 | $SC_k$): probability that A return 1 in case of RO

Advantage of an adversary A: **$Adv_A$ = |Pr(A = 1 | $SC_k$) - Pr(A = 1 | $SC_k$)|** ($Adv_A$ is interval [0...1])

Black box model fails to model that F and f are public thus for a more accurate $Adv_A$ we can model query complexity in two parts:




- M: called online or data complexity
- N: called offline or computational complexity

We can express $Adv_A$ as $\epsilon(M,N)$

| **$\epsilon(M,N)$ Indistinguishability claim for a stream cipher SC:** |
|---|
| There exists no attack algorithm A that distinguishes $SC_K$, with K a uniformly chosen unknown key, from a random oracle with $Adv_A > \epsilon(M,N)$. (This is a very powerful type of claim) |

| **Implications of a $\epsilon(M,N)$ Indistinguishability claim:** |
|---|
| It claims for any imaginable attack: Pr(success of attack on $SC_k$ <= $\epsilon(M,N)$ + Pr(success of attack on RO) |

**Problems with Stream Encryption:**

(1) Diversifier collisions are fatal and avoiding them is seen as difficult
   (a) Taking a counter for D, implies keeping state in between messages which is problematic in some architectures.
   (b) Generating D randomly, is difficult because high quality randomness is hard and there remains the risk of collisions
   (c) Date/time as D requires reliable clocks

(2) It does not protect integrity of the plaintext
   (a) Adversary can flip individual bits in ciphertext which flips the corresponding bits in plaintext.

# Lecture 4 - Block Ciphers

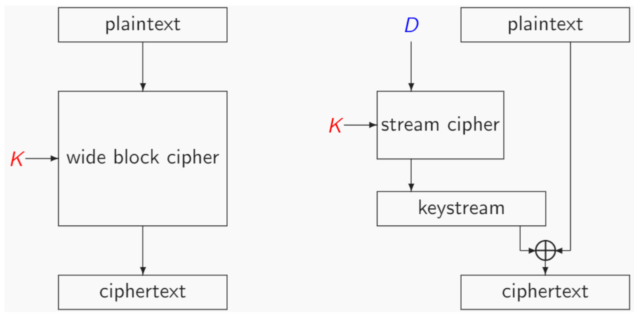
**Block Encryption, ideally:**

- Encryption as scrambling recipe
    - Transforming the full plaintext by sequence of operations
    - (Some of) these transformations depend on a secret key K
    - It must be **invertible**: there must be a recipe for decryption
    - Ciphertext is as long as the plaintext (.. or a little longer)

Such a recipe is called a **wide block cipher** and is considered secure if:

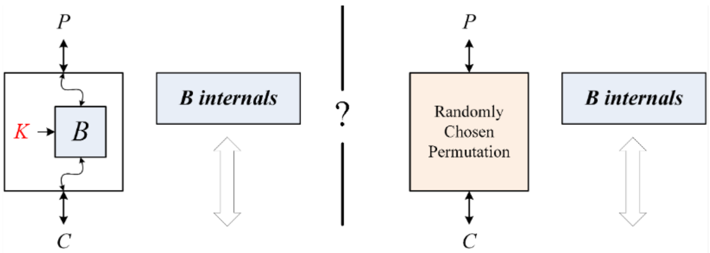
- It maps similar plaintexts to seemingly unrelated ciphertexts (and vice versa)
- This map is completely different for different keys K

Building a wide block cipher may be hard. The established block ciphers have fixed length (DES 8-byte plaintexts and AES 16-byte). Longer plaintexts require splitting in blocks, padding and modes. By fixing length, the advantages of block encryption evaporate.




**Definition Block Cipher:**

Permutation $B_K$ operating on $\{0,1\}^b$ with b the block length (Parameterized by secret key $B_K$ with an inverse $B_K^{-1}$ that should be efficient). Computing $C = B_K(P)$ or $P = B_K^{-1}(C)$ should be efficient knowing the secret key K but infeasible otherwise. Dimensions block length b and key length |K|

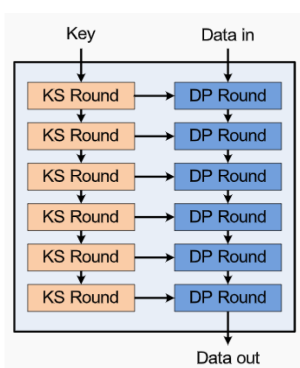



- Infeasibility to distinguish $B_K$ from a randomly chosen
permutation
- Adversary can make encryption queries to $B_K$ or RCP

An SPRP upper bound is also valid for PRP (but not vice versa). By default: a block cipher is considered secure if **SPRP Avd = $\mathbf{N2^{-|K|}}$**

| | Advantage ϵ(M,N) for Pseudorandom Permutation Security (PRS) | Advantage ϵ(M,N) for Strong Pseudorandom Permutation Security (SPRS) |
|---|---|---|
| **M** | $Q_S$ to $B_K$ or RCP | $Q_S$ to $B_K$ and $B_K^{-1}$ or RCP and $RCP^{-1}$ |
| **N** | $Q_c$ to B internals | $Q_c$ to B internals |

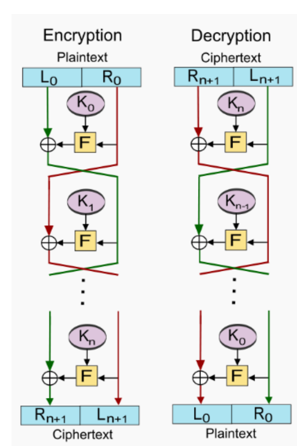
**Iterative block ciphers:**




- Data path (right): transforms input data to output data
  Iteration of a non-linear round function (depends on a round key)
- Key schedule (left): generates round keys from cipher key K

**The feistel structure:**

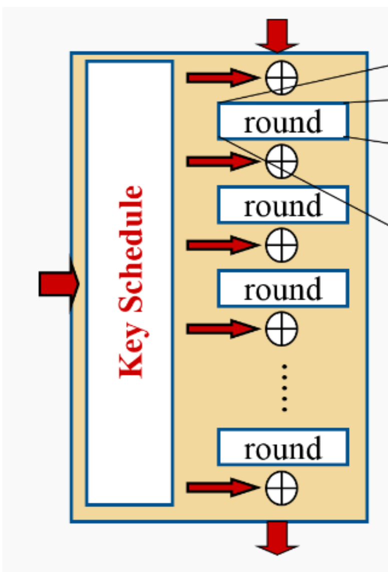
- State: left helf L and right half R
- Apply F to $R_i$ and add to $L_i$, swap left and right. Omit swap in the last round. $B^{-1}$ similar to B. No need for $F^{-1}$. FS is used in DES.

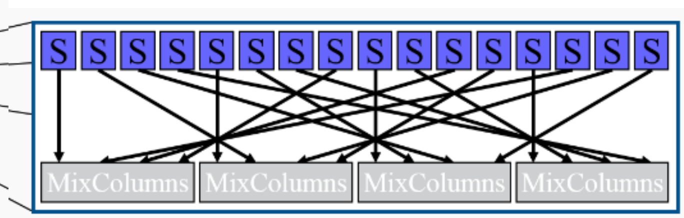



**DES:** block length: 64 bits, key length: 56 bit. 16-round Feistel. Initial IP and final FP permutations.

Key schedule: 8 bits thrown away in permuted choice 1 (PC1), remaining 56- bit split in two 28-bit strings (rotated for each round over 1 or 2 bits). 48-bit round key obtained with PC2 of these 56 bits. **Each round key bit is just a cipher key bit.**
**The problem with DES is the short key.**
Triple DES (double DES allows meet-in-the-middle attacks). TDES has three options:
- 3-key: 168-bit key
- 2-key: 112-bit key by taking $K_3 = K_1$
- 1-key: 56-bit key by taking $K_3 = K_2 = K_1$

**AES:**




10 rounds for 128-bit key, 12 for 192-bit key and 14 for 256-bit key. Last round has no MixColumns so that the inverse is similar to the cipher.

# Lecture 5 - Block Cipher Modes

**Symmetric:**
- Same key for encryption and decryption
- Same key for MAC generation and verification

**Basic Operations:**
- Reduce problem of securing (big) data to a problem of securing (small) keys

A secure solution requires secrecy of keys.
Different attacks:
- Exhaustive key search:
  - Giving some plaintext and corresponding ciphertext (M=1), trying all different keys (N)
- Single-target attack: one particular k-bit key K
  - Success prob. After N trials: $N2^{-k}$, expected effort $N = 2^{k-1}$. Security claim: this should be the best attack so a k-bit key limits security strength to k bits.
- Multi-target attack:

- Attack is happy if she finds one key out of n keys $K_i$, relevant in many cases. (E.g. if keys $K_i$ are on badges giving access to a building)

| **Definition Security Erosion:** |
| --- |
| Security strength is smaller than key if multi-target attacks are possible |

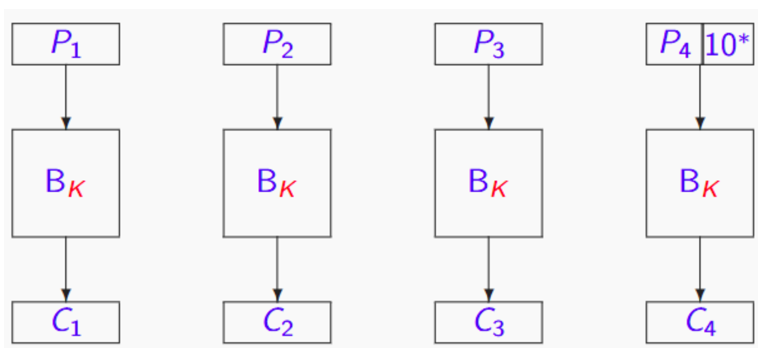
To encipher messages longer than the block ciphers, block encryption or stream encryption is used. **Block encryption modes** split the message in block, after padding the last incomplete block if needed, the permutation $B_K$ is applied to block in some way. **Stream encryption** modes build a stream cipher with a block cipher as updating function F or output function f.

**Padding:**

- Simplest padding: append zeros
    - Up to length multiple of block length
    - Shortest possible padding as such not for out purposes because it is not injective
- Decryption of cryptogram gives padded message, recovering message requires removing padding (send along message or padding length with cryptogram)
- Simplest reversible padding: a single 1 and then zeros (extends message in all cases)
- Padding with exotic requirements like random-length padding: to hide message length or random-padding: to add entropy.

**Block encryption modes:**

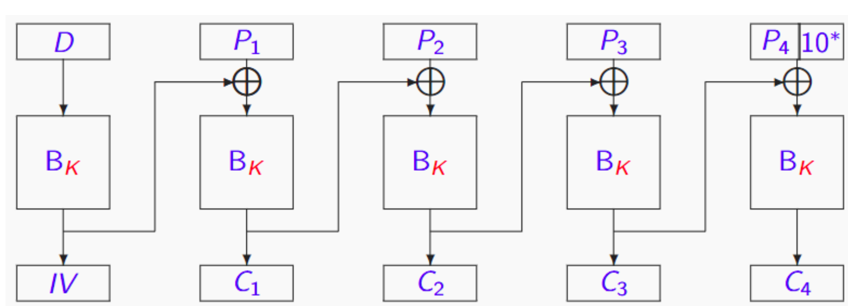
Electronic Code Book (ECB) mode




- (Only 16-byte message are considered)
- Longer messages are split in 16-byte block
- Shorter messages padded to 16 bytes
- Same for last incomplete block
- Advantages: simple and parallelizable. Limitations: equal plaintext block -> equal ciphertext blocks: likely to happen in low-entropy messages. Problem in padded last block, that can be a single byte.

Cipher Block Chaining (CBC) mode

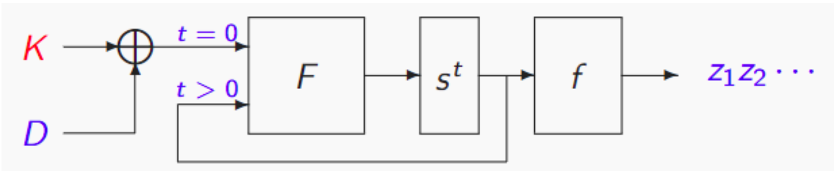
- ECB randomized wit what's available
- Requires also split in 16-byte block and padding
- First plaintext block randomized with random initial value (IV). Solves leakage in ECB (partially): equal plaintext blocks do not lead to equal ciphertext blocks. Requires randomly generating and



transferring IV. Replace this IV with D nonce requirement: IV = $B_K(D)$

- CBC properties: Encryption strictly serial, IV or diversifier D must be managed and transferred. Decryption can be done in parallel.

**Stream encryption with block ciphers**



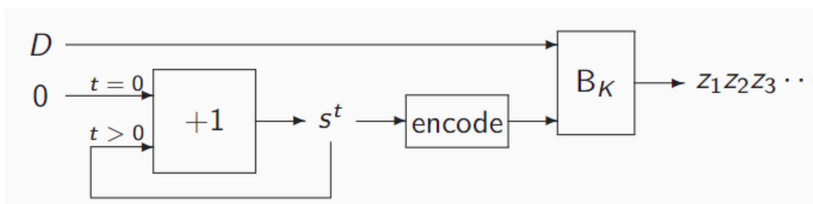

- State update function $s^t = F(s^{t-1})$
- Output function $z_t = f(s^t)$
- Uses a block cipher for F or f

Output FeedBack mode (OFB)

- $F = B_K$, so $s_t \leftarrow B_K(s_{t-1})$
- f is identity $z_t \leftarrow s^t$
- Init: storage of K and $s^0 \leftarrow$ D (often called IV)
- Properties: strictly serial, cycle lengths not knowing in advance, no need for $B^{-1}_K$ (valid for all stream encryption)

Counter mode

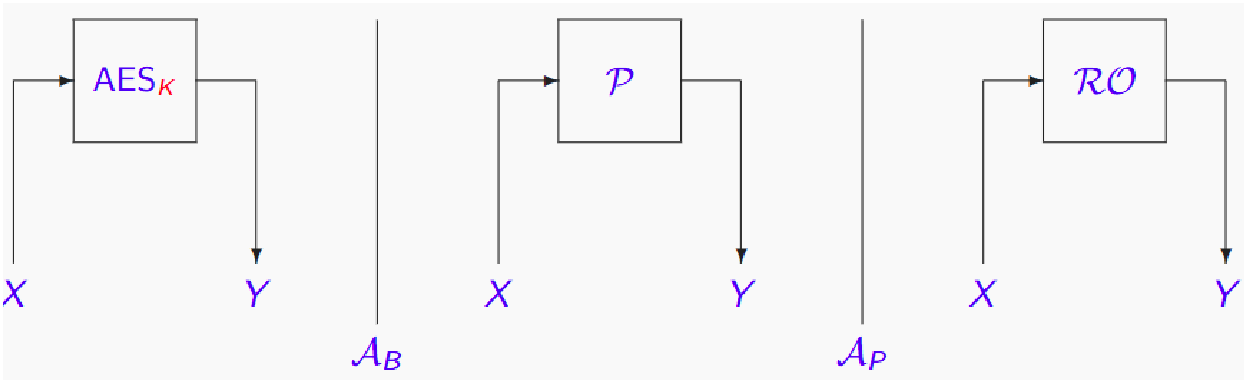
- F: interpret $s^t$ as integer and add 1: $s^t = s^{t-1} + 1$
- $f = B_K$(D || encoding of $s^t$)
- Init: storage of K and $s^0$ <- 0




- Properties: fully parallelizable, l = |Z| for given D limited to $2^{2b-|D|}$ block. No risk of short cycles.

| | ECB | CBC | OFB | Counter |
|---|---|---|---|---|
| **Parallel encryption** | Yes | No | No | Yes |
| **Parallel decryption** | Yes | Yes | No | Yes |
| **Random access** | Yes | Yes | No | Yes |
| **$B^{-1}$ free** | No | No | Yes | Yes |
| **Padding free** | No | No | Yes | Yes |
| **Bit errors C -> P limited** | No | No | Yes | Yes |
| **Nonce-violation tolerant** | n.a. | Yes | No | No |

Random access: fast decryption of bits anywhere in the message
Bit errors limited: bitflips in C do not spread out in P




**Triangle inequality: $Adv_{A'}(AES_K, RO) <= Adv_{AB}(AES_K, P) + Adv_{AP}(P, RO)$** where P is a random permutation and Adversary AB distinguishes between $AES_K$ and P and adversary AP distinguishes between P and RO.

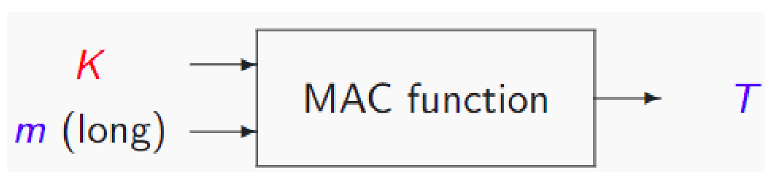
- Advantage of the primitive
    - PRP security of AES
    - Domain of cryptanalysis
    - Cannot be proven, only assumed, claimed and challenged
- Advantage of the mode assuming ideal component
    - CTR mode of a random permutation P
    - Domain of provable security
    - Bounds can be proven using probability theory.

Difference in behaviour between P and RO. P returns uniformly random responses, with restriction that they don't collide. RO returns uniformly random responses. This implies that AP can distinguish P from RO if and only if she is speaking to RO and RO returns colliding outputs. Pr(coll. | RO) = $M^2 2^{-129}$ so advantages get close to 1 when M = $2^{64}$ (birthday bound).

**Message authentication code (MAC) functions**

- MAC: cryptographic checksum
    - Input: key K and arbitrary-length message m
    - output : tag (aka MAC) T with some length l

**T <- $MAC_K$(m)**




Two types:

- Generation: give m and get T <- $MAC^k$(m)
- Verification: give (m,T) and get 1 if T = $MAC_K$(m) and else 0
- 

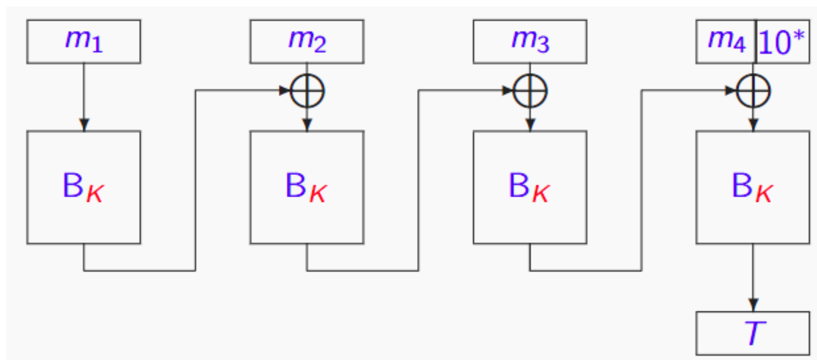
| **Definition MAC forgery:** |
| --- |
| Generating a couple (m,T) such that tag verification returns 1 without knowing K and without querying tag generation with m. |

| **Definition Pseudorandom function (PRF) security of a MAC function** |
| --- |
| MAC() is PRF-secure if $MAC_K(m)$ is hard to distinguish from RO (same security concept as for stream cipher) |

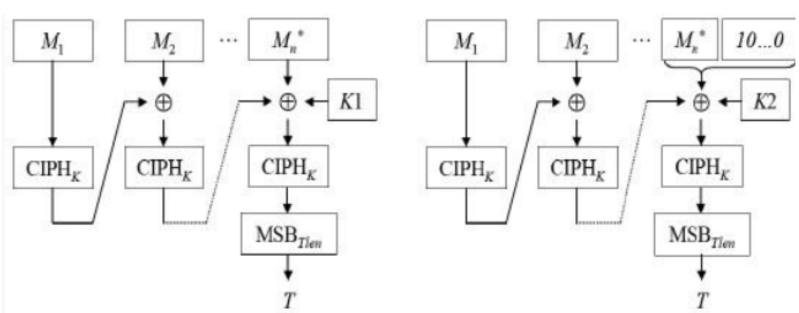
| **Definition PRF-advantage of a MAC function** |
| --- |
| $Adv_A(MAC_K, RO) = \|Pr(A=1 \mid MAC_K) - Pr(A = 1 \mid RO)\|$ |

**An advantage $Adv_A(MAC_K, RO) <= \epsilon(M,N)$**

**Cipher Block Chaining MAC mode (CBC-MAC)**
CBC-MAC using T as the tag.
CBC-MAC weakness: length extension
Distinguishing from RO is two queries:




Query $m_1$ returns $T = B_K(m_1)$
Query $m_1||m_2$ with $m_2 = m_1 + T$
returns $B_K(m_2 + B_K(m1)) = B_K(m_1 + T + B_K(m_1)) = B_K(m_1) = T$
A RO will give two completely unrelated tags.
Fix: **C-MAC:**




Avoid length-extension problem by doing something different at the end: finalization
Addition of subkey before last application of $B_K$.

Consider CBC-MAC with finalization $B'_K$ e.g. C-MAC.
Distinguishing this form a RO:

- Query for many 3-block input $m^{(i)}$ of the form $m_1m_2m_3$
- $m_1$ and $m_2$ different in each query, $m_3$ always the same

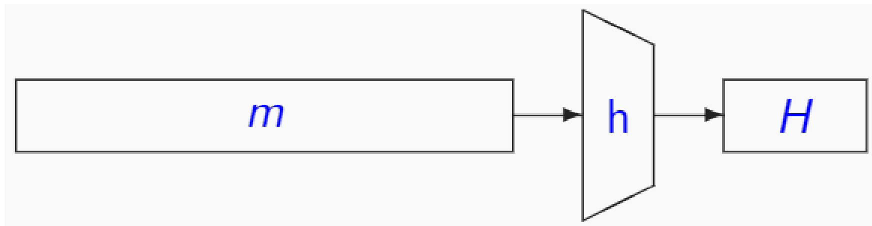
Collision for $i \neq j$ at input of $B'_K$ gives colliding tags

- Probability = $M^2 2^{-(b+1)}$ with M number of queries
- Detect internal collision by tag collision plus some check queries
- Then all m': $m^{(i)}$ || m' gives same tag as $m^{(j)}$ || m'

RO has no internal collisions.

# Lecture 6 - Hashing

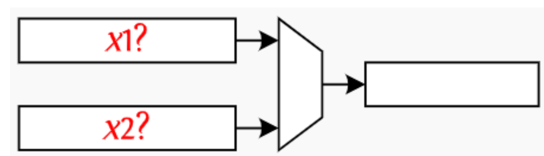
A **Hash Function** is a function h from $\{0,1\}^*$ to $\{0,1\}^l$. No dedicated key input, input m has arbitrary length and output H, called digest of just hash has fixed length l. Secure if it behaves as a RO without output truncated to l bits.




Signing m with private key PrK: sign h(m) instead. Identification of a file m with its hash h(m). These rely on h(m) being unique

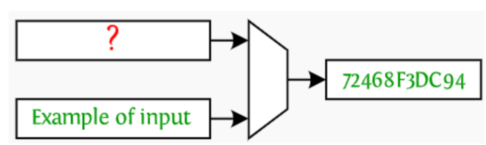
**Definition Collision-resistance**

Hard to find x1 ≠ x2 such that h(x1) = h(x2). For RO: Pr(Success) ≈ $N^2/2^{l+1}$ with N: # calls h(.). Expected cost of generating collision about $2^{l/2}$, collision resistance security strength <= l/2. This is the birthday bound on the digest length l.



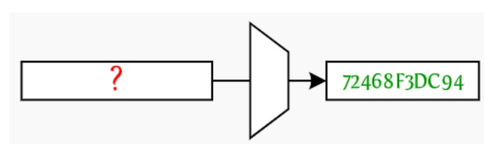

**Definition 2nd preimage resistance**

Given m and h(m), find m' ≠ m such that h(m') = h(m). General attack (on RO) has success probability $N/2^l$. Security strength limited to l instead of l/2.




**Definition Collision-resistance**

Given y, find any m such that h(m) = y. Security strength <= l.



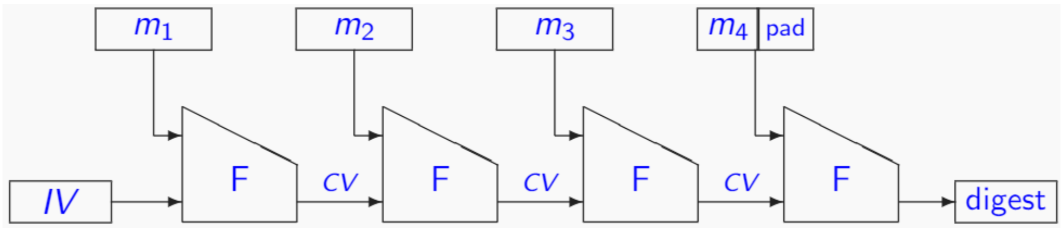

Storage of hashed passwords on servers: h(password||salt). MAC computation: h(K||m) = T. Stream cipher: h(K||D||i) = $z_i$. Key derivation: h(MasterK||"Bob") = $K_{bob}$. Different diversifier values give independent subkeys. Knowledge of $K_i$ shall not reveal MK.

The PRF security is the same notion as for stream cipher and MAC functions.

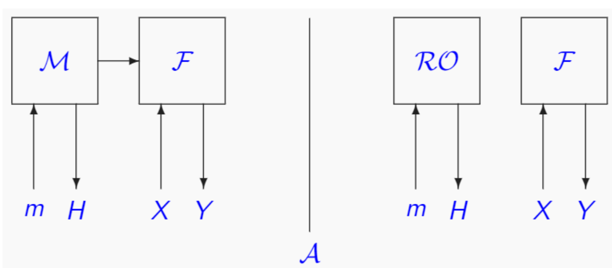
MAC function: forgery success probability h(K||.) sum of: (1) probability of guessing a random l-bit tag correctly $2^{-l}$ and (2) advantage of distinguishing h(K||.) from RO.

| **Definition Domain Separation** |
|---|
| Some applications need multiple independent hash functions. This can be done with a single h using domain separation. Output of h(m‖0) and h(m‖1) are independent (unless h has a cryptographic weakness). Generalization to $2^w$ functions with D a w-bit diversifier. $h_D(m) = h(m\|\|D)$. |

**Merkle-Damgård:**




Mode of use of a fixed-input-length compression function F. Collision-resistance preserving: Collision in hash function implies collision in F, reduces hash function design to fixed-input-length compression function design, implies fixing initial value (IV) of changing value (CV) and conditions on the padding. MD is used in MD5 and standards SHA-1 and SHA-2.

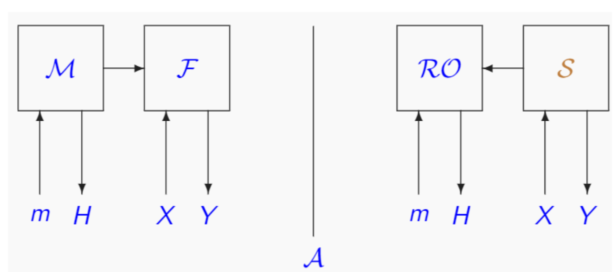



We give adversary access to F in the real and ideal world. M can be distinguished in a few queries: (1) adversary queries h (M(F) or RO) with m, (2) adversary simulation mode M(F) by making calls to F herself. (M(F), F) will behave M-consistently. (RO,F) both return random responses so not likely M-consistent. (Keyed models do not have this problem: unknown key K prevents simple M-inconsistency check).

Concept for hashing:

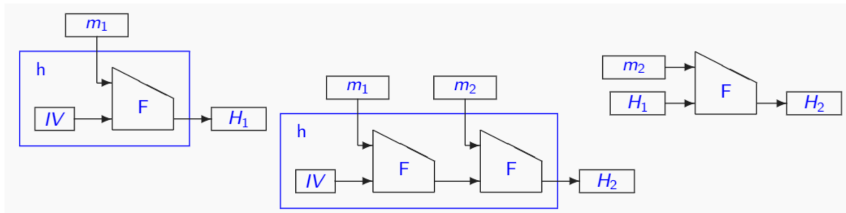
- Adversary gets access to F in the real world, and introduces a counterpart in the ideal world: simulator S.

Methodology for proving bounds on the advantage: build S that makes left/right distinguishing difficult, prove bound of advantage given this simulator S, S may query RO for acting M-consistently: S(RO).




**Merkle-Damgård weakness: Length extension**

- Take the indifferentiability setup with M = MD. Distinguish (M(F), F) from (RO, S(RO)) in 3 queries (see image on the next page):
    - Query h with $m_1$ resulting in $H_1$
    - Query h with $m_1\|\|m_2$ resulting in $H_2$
    - Query F with $H_1\|\|m_2$ resulting in H'




For (M(F), F) we have H’ = $H_2$. Simulator cannot enforce this because it doesn’t know $m_1$ to ask RO. This is called length extension weakness (major problem for MAC function h(K||.)

This problem can be fixed by dedicating a bit in F input to indicate final/non-final. Add 0 at the end of the F(.) query for first, and intermediate blocks. Add 1 at the end of the F(.) query for the last block.

**Limit of iterative hashing: internal collisions**

- There exists input m ≠ m* leading to the same CV. Messages m||X and m*||X always collide for any string X. This effect does not occur in RO. Security strength is upper bounded by birthday bound in CV length.

**Distinguishing iterative hashing modes from RO:**

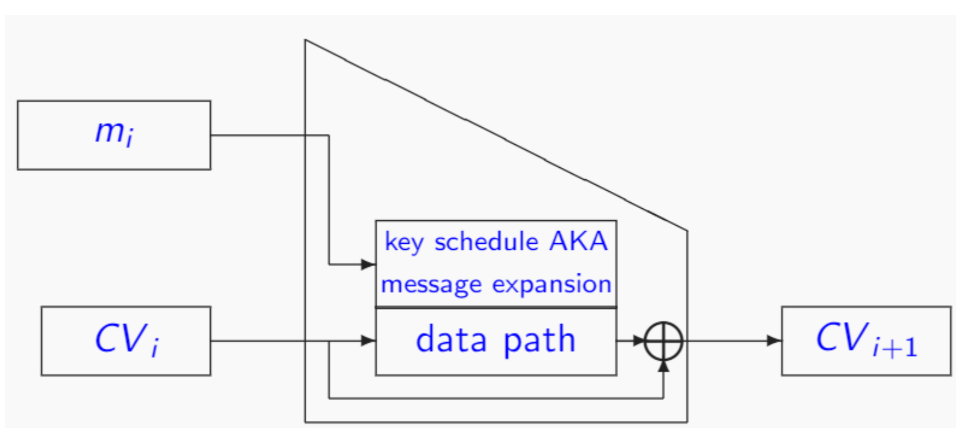
- Send N queries to RO/M(F) of form $m^{(i)}$||X with X always the same.
    - If there is no collision, say RO
    - Otherwise, we have one or more collisions for some i ≠ j
    - For each, query $m^{(i)}$||X’ and $m^{(j)}$||X’ for some X’ ≠ X
    - IF equal: say M(F), otherwise say RO
- Adv ≈ $N^2 2^{-(|CV|+1)}$ (security strength of iterative hashing <= |CV|/2. Truncating output to l < |CV| does not affect advantage).
- Attack success probability on hashing mode with ideal F at most: success probability of that attack on RO + Adv($N^2 2^{-(|CV|+1)}$)

**MD5 and standards SHA-1 and SHA-2**

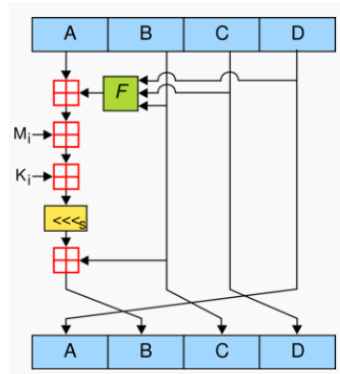
- MD5: based on MD4 that was an original design. 128-bit digest
- SHA-1: inspired by MD5, designed at NSA. 160-bit digest
- SHA-2: reinforced versions of SHA-1, designed at NSA. 6 functions with 224-,256-,384- and 512-bit digest

Internally they use MD iteration mode. F based on a block cipher in Davies-Meyer mode.

Separation data path and message expansion (key schedule) and feedforward due to MD proof: collision-resistance preservation. Why a block cipher: we don’t know how to design a decent compression function from scratch.

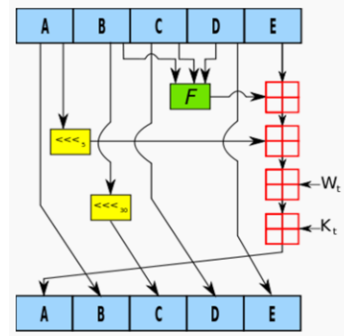


**MD5 internals:**



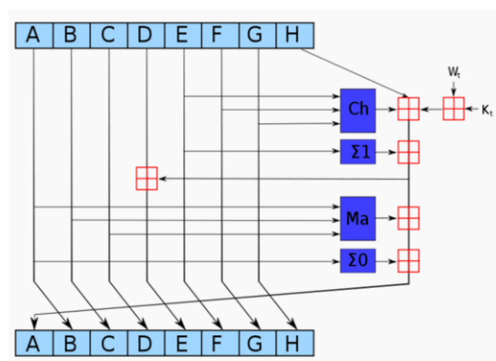

- Software oriented with 32-bit words
- 4-word CV and datapath
- 16-word message block
- 64 rounds, ech taking one message word
- Hoped strength by combining arithmetic, rotation and XOR (ARX)

**SHA-1 internals**




- Same as MD5 but 5-word state and 80 rounds
- Round i takes a word w[i] of the expanded message
- Message expansion
    - i < 16: w[i] = m[i]
    - i >- 16: w[i] = (w[i-3]$\oplus$w[i-8]$\oplus$w[i-14]$\oplus$w[i-16]) <<< 1
- Similar to AES key schedule

**SHA-2 internals**




- 8-word state and nonlinear message expansion
- 6 versions:
    - SHA-256 and SHA-224: 32-bit words and 64 rounds
    - SHA-512, SHA-384, SHA-512/256 and SHA-512/224: 64-bit words and 80 rounds.

MD (length-extension weakness), MD5 (collisions, F shown weak) and SHA-1 (collision attack in effort $2^{61}$) are broken. SHA-2 not yet except length extension.

# Lecture 7 - Sponge Functions

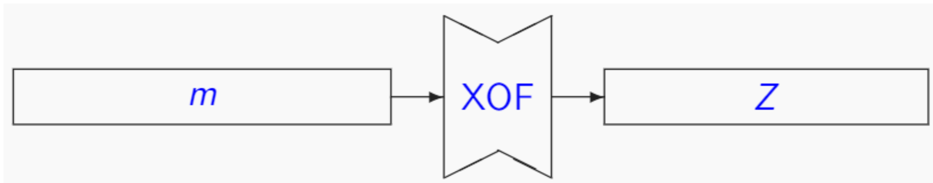
**Keccak (SHA-3):**

- A hashing mode that is sound and simple, with birthday-bound RO-differentiating advantage, calling a primitive that we known how to design
- Block cipher as a primitive: round function design
- No need for separation between data path and key schedule so merged: an (iterative) permutation.
- This is called a sponge construction

Many use cases of hashing require outputs longer or shorter than some nominal digest length: Z = XOF(m,n)

**Definition Extendable Output Function (XOF)**

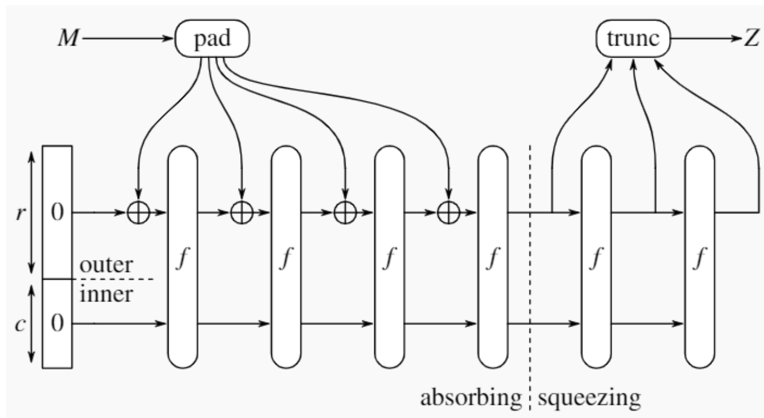
User specifies output length n when calling the function. Name introduced in SHA-3. Secure if it behaves as a RO.




STrength specified in terms of (internal) parameter capacity c.

**Sponge:**

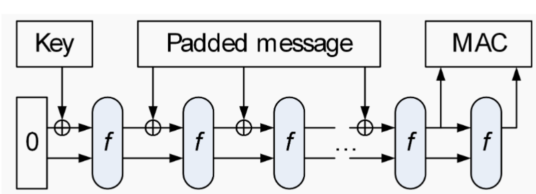
Builds a XOF from a b-bit **permutation** f, with b = r + c.
r bits of rate and c bits of capacity.
RO-differentiating advantage = $N^2 2^{-(c+1)}$. Due to collisions in the c-bit inner part. Super-tight it is the birthday bound in c.



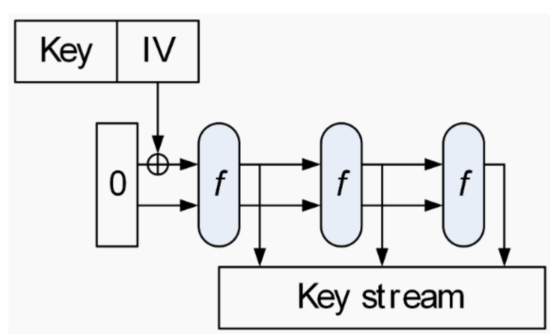

Random sponge: sponge construction with a random permutation P. Success probability of attack on random sponge upper bounded by: success probability of that attack on RO + differentiating advantage of random sponge from RO. Classical attacks on random sponge with output truncated to n bits:

- Collision: $N^2 2^{-(n+1)}+N^2 2^{-(+1)}$
- (first) preimage: $N2^{-n}+N^2 2^{-(c+1)}$
- 2nd preimage: $N2^{-n}+N^2 2^{-(c+1)}$

Security strength of random sponge truncated to n bits: collision resistance: min(c/2, n/2). 1st or 2nd preimage resistance: min(c/2,n). These are bounds for generic attacks (those that do not exploit specific properties of f).




- $MAC_K(m) = XOF(K||m,n)$
- $KDF_K(D) = XOF(K||D,n)$



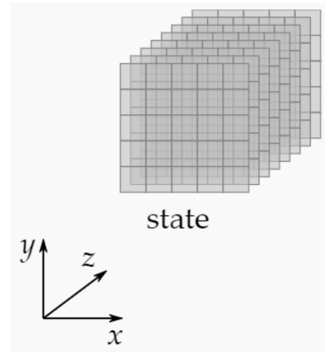

Stream cipher mode (image on the right).

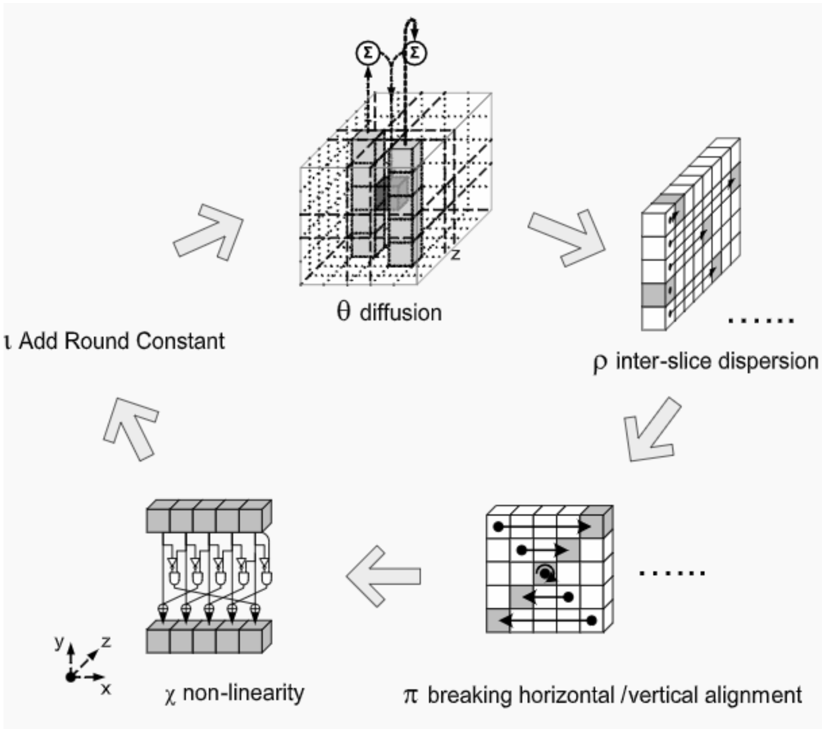
- Many output blocks per D: similar to OFB.
- 1 output block per D: similar to counter mode.

(note: figure indicates diversifier by IV)

Keccak is a sponge function using permutation Keccak-f. Keccak-f operates on a 3D state:

- 5 x 5 lanes, each containing $2^l$ bits (1, 2, 4, 8, 16, 32 or 64)
- (5 x 5)-bit slices, $2^l$ of them




Bit-oriented highly-symmetric wide-trail design.
Keccak-f has 7 permutations: b є {25, 50, 100, 200, 400, 800, 1600}
SHA-3 instance SHAKE128: r = 1344 and c = 256, permutation width: 1600 and security strength 128.
Security status:
Best attack on the hash function covers the 6-round version. # rounds range from 18 for b = 200 to 24 for b = 1600.

Four drop-in replacements for SHA-2 and two XOFs. All use Keccak-f with b = 1600. Domain separated from each other:

- Padding rule ensures separation between different capacities c
- XOF inputs end in **11**, drop-in inputs end in **01**
- XOF Tree-hashing ready: Sakura encoding

| **XOF** | **SHA-2 drop-in replacements** |
|---|---|
| Keccak[c = 256](m\|11\|**11**) | |
| | First 224 bits of Keccak[c = 448](m\|\|**01**) |
| Keccak[c = 512](m\|\|11\|**11**) | First 256 bits of Keccak[c = 512](m\|\|**01**) |
| | First 384 bits of Keccak[c = 768](m\|\|**01**) |
| | First 512 bits of Keccak[c = 1024](m\|\|**01**) |
| **SHAKE128** and **SHAKE256** | **SHA3-224** to **SHA3-512** |

# Lecture 8 - Intro to Public Key

**Cryptography does not fully solve problems, but only reduces them.**

| **Definition Trusted Third Party** |
| --- |
| Alice and Bob both trust a TTP and both share a secret key with it so they can communicate securely with that TTP. |

| **Public-Key Crypto Functionality** |
| --- |
| PKC requires a **key pair** per user. A private key PrK (never to be revealed to the outside world) and a public key PK (to be published and distributed freely). |

**Signatures Schemes:**

- Alice uses $PrK_A$ for signing message: $m$, $Sign_{PrKA}(m)$
- Anyone can use $PK_A$ for verifying Alice's signature
- $PrK_A$ is also called a signing key and $PK_A$ verification key.

**Key establishment (setting up of a shared secret)**

- Key agreement
    - Bob uses $PrK_B$ and $PK_A$ to compute secret $K_{AB}$
    - Alice uses $PrK_A$ and $PK_B$ to compute same secret $K_{AB}$
- Key transport
    - Alice uses $PK_B$ to transfer secret $K_{AB}$ to Bob, that uses $PrK_B$

**Modular Arithmetic**

Notation:

- $\mathbb{Z}$: set of integers
- $a \in A$: this means that a is an element of set A
- $\forall$: for all or for every
- $\exists$: there exists
- $C = A \setminus B$: C contains elements of A that are not in B
- #A: the cardinality of a set, the number of elements it has
- $\mathbb{Z}/n\mathbb{Z}$: the set of residue classes modulo n (aka positive integers smaller than n, including zero)

**Finite groups**

Couple (A, ★) of a set A and an operation ★. The binary operation must satisfy the following properties:

| | | |
| --- | --- | --- |
| Closed | $\forall a,b \in A$ | $a ★ b \in A$ |
| Associative | $\forall a,b,c \in A$ | $(a ★ b) ★ c = a ★ (b ★ c)$ |
| Neutral element | $\exists e \in A, \forall a \in A$ | $a ★ e = e ★ a = a$ |

| | | |
|---|---|---|
| Inverse element | $\forall a \in A, \exists a' \in A$ | $a \bigstar a' = a' \bigstar a = e$ |
| Abelian (optional) | $\forall a,b \in A$ | $a \bigstar b = b \bigstar a$ |

Additive: (A, +) e = 0 a' = -a
Multiplicate: (A, *) e = 1 $a' = a^{-1}$

| Group Order |
|---|
| Order of a finite group (A, ★), denoted #A, is number of elements in A |

In a finite group (A, ★):
- $\forall a \in A$ this sequence is periodic
- Period of this sequence: order of a, denoted ord(a)

| Order of a group element |
|---|
| The order of a group element a, denoted ord(a), is the smallest integer k > 0 such that $a^k = 1$ (multiplicative) or [k]a = 0 (additive) |

**Cyclic groups**
Let $g \in$ (A, ★) and consider the set [0]g, [1]g, [2]g,. This is called cyclic group denoted:$\langle g \rangle$
- Composition law: [i]g + [j]g = [i + j mod ord(g)]g
- Neutral element [0]g
- Inverse of [i]g: [ord(g) - i)g

**Subgroups**: A subset B of A that is also a group (under the same operation): (B, ★) is subgroup of (A, ★) if:
- B is a subset of A
- $e \in B$
- $\forall a,b \in B$: $a \bigstar b \in B$
- $\forall a \in B$: the inverse of a is in B

| Lagrange's Theorem |
|---|
| If (B, ★) is a subgroup of (A, ★): #B divides #A |

In case of cyclic subgroup: $\forall a \in A\langle g \rangle$is a subgroup of (A, ★) and for any element $a \in A$: ord(a) divides #A.

Finding the prime number factorization is a **computationally hard problem** and one can base public-key cryptosystems on the hardness of this factoring.

**Greatest Common Divisor (gcd):** gcd(n,m) = greatest integer k that divides both n and m, greatest k with n = k * n' and m = k * m' for some n', m'.
If gcd(n.m) = 1, n and m are relatively prime or coprime.
**Euclidean Algorithm:**
Property: (assume n > m > 0):
- gcd(n,m) = gcd(m, n mod m)

Continue till one of the arguments in 0
**Extended Euclidean Algorithm:**
Returns a pair x,y ∈ ℤ with n * x + m * y = gcd(n,m)

The Extended Euclidean Algorithm

Example 1: $m = 65, n = 40$

Step 1: The (usual) Euclidean algorithm:

$$\begin{array}{llll} (1) & 65 & = 1 \cdot 40 & + \boxed{25} \\ (2) & 40 & = 1 \cdot \boxed{25} & + 15 \\ (3) & \boxed{25} & = 1 \cdot 15 & + 10 \\ (4) & 15 & = 1 \cdot 10 & + 5 \\ & 10 & = 2 \cdot 5 & \end{array}$$

Therefore: $\gcd(65, 40) = 5$.

Step 2: Using the method of back-substitution:

$$\begin{aligned} 5 &\overset{(4)}{=} 15 - 10 \\ &\overset{(3)}{=} 15 - (25 - 15) = 2 \cdot 15 - 25 \\ &\overset{(2)}{=} 2(40 - 25) - 25 = 2 \cdot 40 - 3 \cdot 25 \\ &\overset{(1)}{=} 2 \cdot 40 - 3(65 - 40) = 5 \cdot 40 - 3 \cdot 65 \end{aligned}$$

Conclusion: $65(-3) + 40(5) = 5$.

**Invertibility criterion**

m has multiplicative inverse modulo n iff gcd(m.n) = 1

**Corollary**

For p a prime, every non-zero m ∈ ℤ/pℤ has an inverse

**Multiplicative prime groups**

If p is prime, (ℤ/pℤ)* is cyclic group of order p - 1

**Square and multiply:**
Computing $g^e$ mod p in a naive way takes e - 1 multiplications but can be done faster with square-and-multiply.

Typical computation cost for $g^e$ mod p:
- |e| - 1 squaring, with |e| the bitlength of e
- 1 to |e| - 1 multiplications, depending on e and method

Example: computing $g^{43}$ with $g = 714, p = 1019$

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | $g^2$ | $= g \times g$ | $g^2 =$ | 296 | = | $714 \times 714$ |
| 100 | $g^4$ | $= g^2 \times g^2$ | $g^4 =$ | 1001 | = | $296 \times 296$ |
| 1000 | $g^8$ | $= g^4 \times g^4$ | $g^8 =$ | 324 | = | $1001 \times 1001$ |
| 10000 | $g^{16}$ | $= g^8 \times g^8$ | $g^{16} =$ | 19 | = | $324 \times 324$ |
| 100000 | $g^{32}$ | $= g^{16} \times g^{16}$ | $g^{32} =$ | 361 | = | $19 \times 19$ |

working it out:

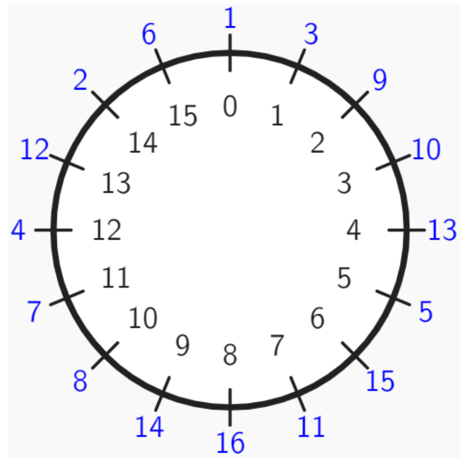
| | | | | | | |
|---|---|---|---|---|---|---|
| 11 | $g^3$ | $= g^2 \times g$ | $g^3 =$ | 411 | = | $296 \times 714$ |
| 1011 | $g^{11}$ | $= g^8 \times g^3$ | $g^{11} =$ | 694 | = | $324 \times 411$ |
| 101011 | $g^{43}$ | $= g^{32} \times g^{11}$ | $g^{43} =$ | 879 | = | $361 \times 694$ |

**Correspondence between⟨g⟩and ℤ/ord(g)ℤ**

For every A ∈⟨g⟩ there is a number a ∈ ℤ/ord(g)ℤ such that A = $g^a$

An example with $(\mathbb{Z}/17\mathbb{Z})^*$ and $\mathbb{Z}/16\mathbb{Z}$ is shown on the right. For each blue element $3^i \in \langle 3 \rangle$ we have a black element $i \in \mathbb{Z}/16\mathbb{Z}$.

- $C = A \times B = A * B \bmod 17$ maps to $c = a + b \bmod 16$
- $C = A^e \bmod 17$ maps to $c = a * e \bmod 16$




**Discrete log:**

- Given x, compute X such that $X = 3^x \bmod 17$: exponentiation
- Given X, compute x such that $X = 3^x \bmod 17$: discrete log
- Exponentiation is easy but discrete log is hard for many groups $\langle g \rangle$

# Lecture 9 - Diffie-Hellman & ElGamal

**Key agreement:** Alice and Bob exchange information over a public channel. After the protocol they share a secret.

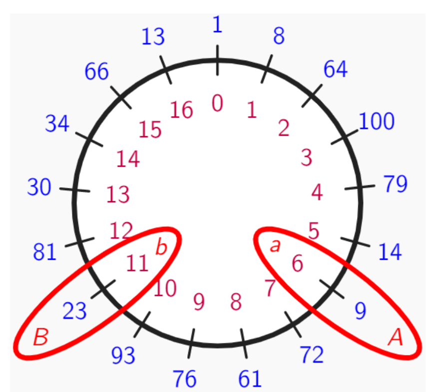
**Discrete-log based cryptography: key material**

Domain parameters : specification of cyclic group we work in

- Non- secret information that is common to all users:
    - $p \in N$ (natural numbers): prime modulus
    - $g \in (\mathbb{Z}/p\mathbb{Z})^*$: generator (and its order q)
- One always takes g with large prime order $ord(g) = q$
    q divides p-1 (due to Lagrange) so $\langle g \rangle \neq (\mathbb{Z}/p\mathbb{Z})^*$
- Key pairs
    - Private key PrK that Alice keeps for herself: $a \in \mathbb{Z}/p\mathbb{Z}$
    - Public key PK that Alice makes public: $A = g^a \in \langle g \rangle$

| **Key pair generation in discrete-log based crypto** |
|---|
| (1) Random selection of the private key: $a \leftarrow \mathbb{Z}/q\mathbb{Z}$<br>(2) Computation of the public key: $A \leftarrow g^a$ |

The image on the right shows the A and the B of Alice and Bob, which are both public keys. And the a and b of Alice and Bob which are both private keys.




**Diffie-Hellman key agreement:**

Alice and Bob arrive at the sam shared secret $K_{A,B} = K_{B,A}$

$K_{A,B} = B^a = (g^b)^a = g^{b*a} = g^{a*b} = (g^a)^b = A^b = K_{B,A}$

Alice and Bob derive key(s) from secret: $K \leftarrow h(\text{“KDF”}; K_{A,B})$

K will be used to encipher and/or MAC their communication

| | Alice | | Bob |
|---|---|---|---|
| have in advance: | $p, g, q$ | | $p, g, q$ |
| | $a \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | $b \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ |
| | $A \leftarrow g^a$ | | $B \leftarrow g^b$ |
| | | $\xrightarrow{\text{Alice}, A}$ | |
| | | $\xleftarrow{\text{Bob}, B}$ | |
| | $K_{A,B} \leftarrow B^a$ | | $K_{B,A} \leftarrow A^b$ |

Man-in-the-middle attack:

| Alice | | Eve | | Bob |
|---|---|---|---|---|
| $p, g, q$ | | $p, g, q$ | | $p, g, q$ |
| $a \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | $e \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | $b \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ |
| $A \leftarrow g^a$ | | $E \leftarrow g^e$ | | $B \leftarrow g^b$ |
| | $\xrightarrow{\text{Alice}, A}$ | | $\xrightarrow{\text{Alice}, E}$ | |
| | | | $\xleftarrow{\text{Bob}, B}$ | |
| | $\xleftarrow{\text{Bob}, E}$ | | | |
| $K \leftarrow E^a$ | | $K \leftarrow A^e$ | | |
| | | $K' \leftarrow B^e$ | | $K' \leftarrow E^b$ |

Solution to MitM: Alice must verify B really comes from Bob and Bob must verify A really comes from Alice. **Public-key authentication is essential**

DH security against eavesdropping Eve:

- Eve needs either a or b compute $K_{A,B}$
- Given g, A and B, prediction $K_{A,B}$ should be hard: **(computational) Diffie-Hellman hardness assumption (CDH)**
- CDH seems as hard as discrete log problem but no proof of this

- Entity authentication can be done with challenge-response using a key derived from shared secret, along with encryption, message origin authentication.

**Mutual PK authentication**: both parties authenticate public keys

**Unilateral authentication:** only one party authenticates public key

Static DH: Alice and Bob have long-term key pairs (advantage: only needs to be authenticated once, disadvantage: $K_{A,B}$ is always the same and leakage of $K_{A,B}$, a or b allows decryption of all part messages)

| **Forward secrecy** |
|---|
| Forward secrecy is the property that the compromise of keys in a device does not compromise encrypted communication of the past |

**Ephemeral key pairs**: Alice and Bob generate fresh key paris per session/message
To still protect from MitM attack: both Alice and Bob have long-term signing keys they authenticate from each other.

**ElGamal Encryption:**

| Alice | | Bob |
|---|---|---|
| $p, g, (q), B$ | | $p, g, (q), b, B(= g^b)$ |
| $a \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | |
| $A \leftarrow g^a$ | | |
| $C \leftarrow M \times B^a$ | $\xrightarrow{\text{Alice},(C,A)}$ | $M' \leftarrow C \times A^{q-b}$ |

$$M' = C \times A^{q-b} = M \times B^a \times A^{-b} = M \times (g^b)^a \times (g^a)^{-b} = M \times g^{ba} \times g^{-ab} = M$$

Alice encrypts a message M to cryptogram (C,A) for Bob like the image above.

- Message M must be an element of $\langle g \rangle$
  - Requires encoding function mapping m to $M \in \langle g \rangle$
  - Note: must be efficiently decodable for Bob te decrypt
  - Existence of such a function depenso n the group$\langle g \rangle$
- As first step, Alice generates an ephemeral key pair (a,A)
  - For security a must be randomly generated for each encryption
  - Re-use leads to leakage like in one-time pad

ElGamal security:

- SEcure if one-time secret $B^a$ is indistinguishable from random element in $\langle g \rangle$
- Decisional Diffie-Hellman (DDH) security notion for $\langle g \rangle$
  - With what Eve knows, she cannot distinguish $B^a$ from an element randomly chosen from $\langle g \rangle$ that is: given ($g^a$, $g^b$, C) it is hard to determine whether $C = g^{ab}$

**IND-CPA security of ElGamal:**

- Security notion for (public-key) encryption: indistinguishability under chosen-plaintext attacks (IND-CPA).
- For $Enc_{PK}$ we play a game between challenger and adversary
- Adversary must guess which message was encrypted: $M_0$ or $M_1$ **(see image: page 28)**
- Secure if adversary has negligible advantage: which is the case if DDH problem is hard

| Challenger | | Adversary |
|---|---|---|
| $p, g, (q)$ | | $p, g, (q)$ |
| $b \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | |
| $PK \leftarrow g^b$ | $\xrightarrow{PK}$ | Repeat: $\mathrm{Enc}_{PK}(M)$ |
| | $\xleftarrow{M_0, M_1}$ | $M_0, M_1$ messages |
| $i \xleftarrow{\$} \{0, 1\}$ | | |
| $CT \leftarrow \mathrm{Enc}_{PK}(M_i)$ | $\xrightarrow{CT}$ | Repeat: $\mathrm{Enc}_{PK}(M)$ |

**Key encapsulation Mechanism (KEM) for ElGamal:**
KEM transport a key from Alice to Bob without interaction

- Allows sending arbitrary message in one transmission
  - Pk crypto is used to transport a shared secret
  - In same transmission, (symmetrically) encrypted message

| Alice | | Bob |
|---|---|---|
| $p, g, (q), B$ | | $p, g, (q), b, B(= g^b)$ |
| $a \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | |
| $A \leftarrow g^a$ | | |
| $K \leftarrow \mathrm{h}(\text{"KDF"}; B^a)$ | | |
| $CT \leftarrow \mathrm{Enc}_K(m)$ | $\xrightarrow{\text{Alice}, (A, CT)}$ | $K \leftarrow \mathrm{h}(\text{"KDF"}; A^b)$ |
| | | $m \leftarrow \mathrm{Dec}_K(CT)$ |

| **Problem:** | **Explanation** |
|---|---|
| Discrete log (DL) problem | Let $a \mapsfrom \mathbb{Z}/q\mathbb{Z}$ and $A \leftarrow g^a$<br>Given $\langle g \rangle$ and $A$, determine $a$ |
| Computational Diffie-Hellman (CDH) problem | Let $a,b \mapsfrom \mathbb{Z}/q\mathbb{Z}$, $A \leftarrow g^a$ and $B \leftarrow g^b$<br>Given $\langle g \rangle$ and $A$, $B$, determine $g^{ab}$ |
| Decisional Diffie-Hellman (DDH) problem | Let $a,b,c \mapsfrom \mathbb{Z}/q\mathbb{Z}$, $A \leftarrow g^a$ and $B \leftarrow g^b$<br>With probability ½, set $C \leftarrow g^c$ and otherwise $C \leftarrow g^{ab}$<br>Given $\langle g \rangle$ and $A$, $B$, $C$ determine whether $C = g^{ab}$ holds |

| **Assumption:** | **Explanation** |
|---|---|
| Computational hardness assumption | Let s be the security strength. A problem is computationally hard to solve with respect to s, if for all algorithms that solve ti with computational complexity N and success probability p, it holds that $N/p \geq 2^s$ |
| Indistinguishability hardness assumption | Let s be the security strength. An indistinguishability problem is hard with respect to s, if for all distinguishers A with computational complexity N and advantage $Adv_A$, it holds that $N/Adv_A \geq 2^s$ |

$Adv_A = |Pr(A=1 \mid C = g^{ab}) - Pr(A=1 \mid C = g^{c})$

**DDH is hard => CDH is hard => DL is hard**
**Strength: $\log_2(N/Pr(success))$ or $\log_2(N/Adv_A)$ with N workload.**

# Lecture 10 - Schnorr

| **Definition Completeness** |
|---|
| If the prover knows the secret, and prover and verifier run the protocol as specified, the protocol succeeds |

| **Definition Soundness** |
|---|
| If the prover does not know the secret, the protocol will only succeed with negligibly small probability |

**Chaum-Evertse-van de Graaf (CEG) protocol**

| | Alice | | Bob | |
|---|---|---|---|---|
| | $p, g, q, A, a$ | | $p, g, q$ (Alice: $A$) | |
| | $v \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | | |
| | $V \leftarrow g^v$ | $\xrightarrow{\text{Alice}, V}$ | $c \xleftarrow{\$} \{0,1\}$ | |
| | | $\xleftarrow{c}$ | | |
| if $(c = 0)$ | $r \leftarrow v$ | | | |
| else | $r \leftarrow v - a$ | $\xrightarrow{r}$ | if $(c = 0)$ | $V \stackrel{?}{=} g^r$ |
| | | | else | $V \stackrel{?}{=} g^r A$ |

Eve anticipates that challenge will be 0.
On a good day:

| Eve | | Bob |
|---|---|---|
| $p, g, q, A$ | | $p, g, q$, (Alice: $A$) |
| $r \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$, $V \leftarrow g^r$ | $\xrightarrow{\text{Alice}, V}$ | $c \xleftarrow{\$} \{0, 1\}$ |
| | $\xleftarrow{c(=0)}$ | |
| | $\xrightarrow{r}$ | $V \stackrel{!}{=} g^r$ |

On a bad day:

| Eve | | Bob |
|---|---|---|
| $r \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ , $V \leftarrow g^r$ | $\xrightarrow{\text{Alice}, V}$ | $c \xleftarrow{\$} \{0, 1\}$ |
| | $\xleftarrow{c(=1)}$ | |
| I'm outta here! | | |

Same thing but then reverse for the guess the challenge will be 1. If and only if she makes the right guess, the protocol succeeds, so her success **probability is ½.**

**Iterating the CEG protocol:**

| | Alice | | Bob | |
|---|---|---|---|---|
| | $p, g, q, A, a$ | | $p, g, q$ (Alice: $A$) | |
| For $i$ from 1 to $n$: | | | | |
| | $v_i \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | | |
| | $V_i \leftarrow g^{v_i}$ | $\xrightarrow{\text{Alice}, V_i}$ | $c_i \xleftarrow{\$} \{0, 1\}$ | |
| | | $\xleftarrow{c_i}$ | | |
| if $(c_i = 0)$ | $r_i \leftarrow v_i$ | | | |
| else | $r_i \leftarrow v_i - a$ | $\xrightarrow{r_i}$ | if $(c_i = 0)$ | $V_i \stackrel{?}{=} g^{r_i}$ |
| | | | else | $V_i \stackrel{?}{=} g^{r_i} A$ |

Note: these n iterations can be done in parallel, so with only 3 messages. Eve's success probability now shrinks to $2^{-n}$

**Definition Transcript**

A transcript of a protocol is the sequence of messages exchanged

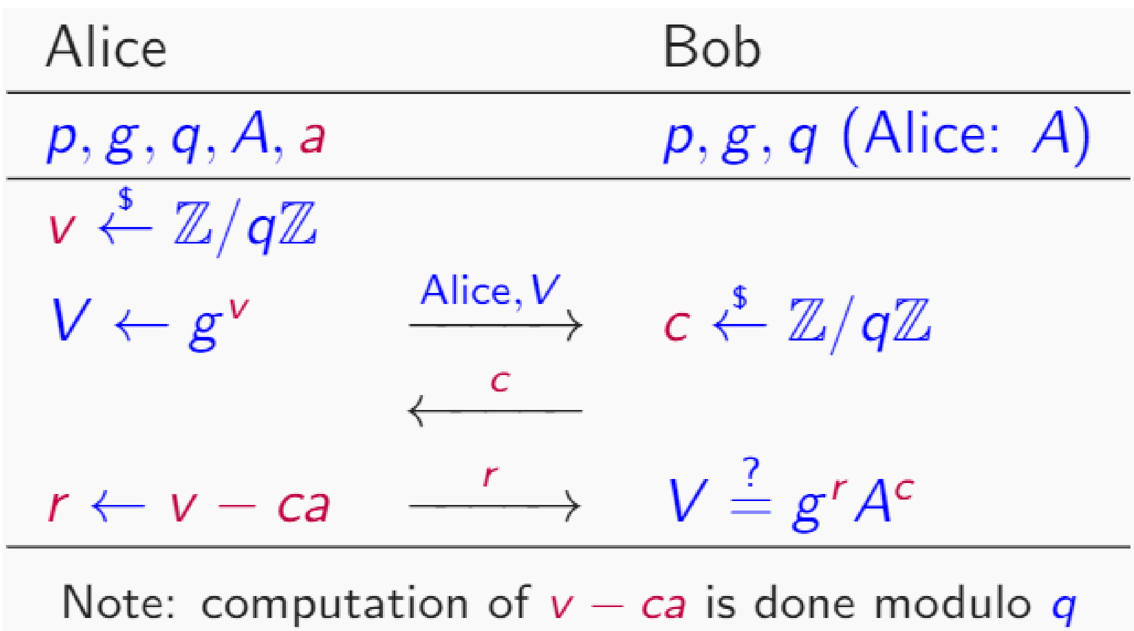
**Definition Simulator**

An algorithm that generates valid transcripts

**Definition zero-knowledge**

A protocol is zero-knowledge if there exists an efficient simulator that, given only public information, generates valid transcript that cannot be distinguished from transcript of valid protocol runs.

**Schnorr Authentication Protocol**




Eve can still cheat by guessing c but success probability is 1/q. v **shall be chosen randomly and freshly for every protocol run and never leak.**
Security of Schnorr protocol
Completeness: Schnorr has absolute and unconditional completeness
Soundness: Schnorr is sound on the condition that DL is hard
Zero-knowledge: Schnorr is (honest-verifier) zero knowledge (honest-verifier means that challenges should be generated randomly in the protocol).

CEG and Schnorr authentication are interactive protocols: 3 messages (commit, challenge, response) and they are examples of so-called ∑-protocols.

**Fiat-Shamir transform:**

- Ideas:
    - The output of a random oracle (RO) is unpredictable
    - A cryptographic hash function should behave like a RO
- Prover generates the challenge c as a hash of the commitment V
    - Verifier checks if c is indeed the hash of V
    - Also includes her public key (p,g,A) in hash input
    - This makes the pair V,c only valid for this particular prover
- So $c \leftarrow h(p;g;A;V)$
    - X;y;z injective encoding of sequence x,y,z in a string
    - For schnorr, has output shall be converted to element of $\mathbb{Z}/q\mathbb{Z}$
- Security
    - As RO is unpredictable, prover can't predict c when choosing V
    - Cheating requires, for given p,g and A, finding (V,r) that satisfies $V = g^r A^{h(p;g;A;V)}$
    - If h behaves like RO and DL is hard, this is hard
- The transcript (Alice, V, c, r) now proves knowledge of a private key

**Schnorr Signatures**

| Alice | | Bob |
|---|---|---|
| $p, g, q, A, a$ | | $p, g, q$ (Alice: $A$) |
| $v \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | |
| $V \leftarrow g^v$ | | |
| $c \leftarrow \mathrm{h}(p; g; A; V; m)$ | | |
| $r \leftarrow v - ca$ | $\xrightarrow{m,(r,V)}$ | $c \leftarrow \mathrm{h}(p; g; A; V; m)$ |
| | | $V \stackrel{?}{=} g^r A^c$ |

# Lecture 11 - ECC 1

**ECDH: Elliptic Curve Diffie-Hellman**

$\langle g \rangle \subset \mathcal{E}$ with elliptic curve group $\mathcal{E}$ and ord(G) chosen to offer safety margi against best DL attacks in 1999 and still today.

**Rings**

Take a set and two operations (e.g. addition and multiplication)

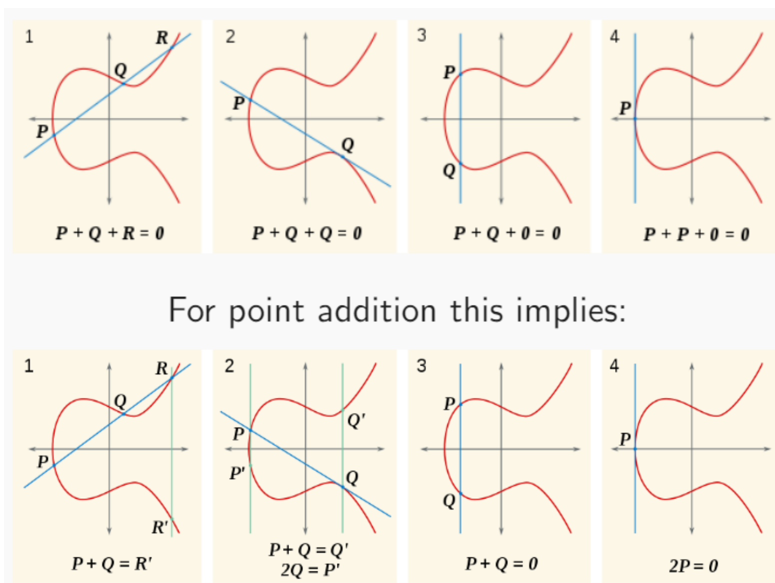
- $(\mathbb{Z}, +)$ is a group
- $(\mathbb{Z}, *)$ satisfies:
    - Closed

        - Associate
        - Has neutral element:1
- Additional property: * is distributive with respect to +
        - a(b+c) = ab + ac
- This is called a **Ring**

**Prime fields**
Consider (ℤ/pℤ, +, *) with p a prime
- (ℤ/pℤ, +) is a group
- (ℤ/pℤ \ {0}, *) is a group
- * is distributive with respect to +

This is called a **finite field** denoted as $F_p$ (or as GF(p))
Properties of GF(p):
- Additive group has order p
- Multiplicative group has order p - 1
- There is exactly one finite field per prime

**Elliptic curve groups**
Most widespread for curves of GP(p): set of points (x,y) that satisfy:
$y^2 = x^3 + ax + b$
For some fixed values p, a, b (these are domain parameters)
All elliptic curves over GP(p) can be represented this way (the Weierstrass equation)




**Ɛ is an abelian group:**
- Closure: a straight line intersecting the curve in 2 points will intersect it in a 3rd point
    - If a third-degree equation has 2 roots, it has one more
- Associativity holds
- Identity: the point at infinity O
- Inverse: if P = (x,y) then -P = (x, -y)

| **Definition of the group law** |
|---|
| Let P, Q, R ∈ Ɛ: P + Q + R = O iff they are on a straight line O is at infinity in the direction of the y-axis |

Computing R = P + Q in Ɛ with P = $(x_p, y_p)$, Q = $(x_q, y_q)$, R = $(x_r, y_r)$

$x_p \neq x_q$ slope of line $P$-$Q$

$$\lambda = \frac{y_p - y_q}{x_p - x_q}$$

$P = Q$, slope of tangent

$$\lambda = \frac{3{x_p}^2 + a}{2y_p}$$

Point on the line satisfy $y = y_p + \lambda(x - x_p)$
Substituting y in Weierstrass: $(y_p + \lambda(x - x_p))^2 = x^3 + ax + b$
Coefficient of $x^2$ in this equation is $-\lambda^2$, so $x_p + x_q + x_r = \lambda^2$, or

$x_r = \lambda^2 = x_p - x_q$

$y_r = \lambda (x_p - x_r) - y_p$

All additions, subtractions and multiplications are modulo p. Division is multiplication by inverse, requiring ext. Euclidean or exponentiation.

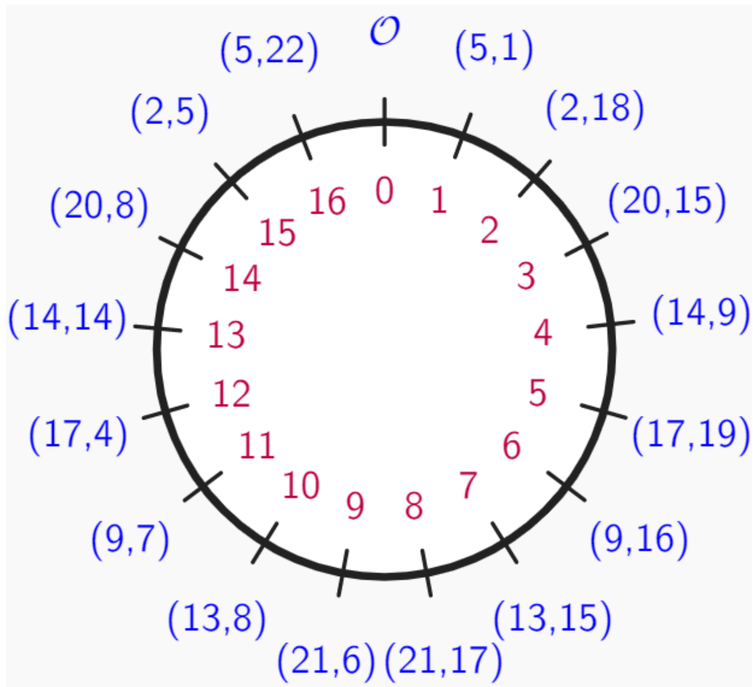
**DL in ECC:**
For G ∈ Ɛ, consider the sequence:

- i = 1 : G
- i = 2 : G + G
- i = 3 : G + G + G
- ...
- i = n : [n]G

[n]G is called the scalar multiplication of point G by scalar n. ord(G) is the smallest integer q > 0 such that [q]G = O.

| **DL problem in Ɛ** |
|---|
| Let G ∈ Ɛ have order q<br>Let $a \mapsfrom \mathbb{Z}/q\mathbb{Z}$ and $A \leftarrow [a]G$<br>Given $\langle g \rangle$ and A, determine a |

Illustration with a cyclic subgroup of:
Ɛ(GP(23)): $y^2 = x^3 - x - 4$
Is on the right.




Here $G = (5, 1) \in \mathcal{E}$ and $\mathrm{ord}(G) = 17$

For each $i \in \mathbb{Z}/17\mathbb{Z}$ we have $[i]G \in \mathcal{E}$

# Lecture 12 - ECC 2

To have n bits of security for DL it would be sufficient that:

(1) $q = ord(G) \geq 2^{2n}$ and q prime
(2) Ɛ is chosen so that it avoids some properties

Due to Langrange: ord(G) | #Ɛ, so we need Ɛ with an order that is divisible by a prime $\geq 2^{2n}$.
Expect from #Ɛ:

- For roughly half of the values $x \in GP(p)$, the expression $x^3 + ax + b$ is a square
- If so and if y is a solution, so is -y
- So #Ɛ(GP(p)) = ½ * 2 * p + 1 = p + 1

| **Theorem of Hasse** |
|---|
| For an elliptic curve over GP(p): #Ɛ = p + 1 + t with $-2\sqrt{p} \leq t \leq 2\sqrt{p}$ |

**ECC Domain parameters**

- We want Ɛ with #Ɛ = hq with q a large prime and h ≤ 10 or so
- Technique: repeat until a suitable curve is found
    - Take parameters p, a, b that would give a good curve
    - Compute #Ɛ with Schoof's algorithm
- To assure backdoor absence, choice of p, a, b should be explainable
- Curves are proposed by experts and standardization bodies

| **ECC Domain Parameters** |
|---|
| - The prime p (in general, a prime power $p^n$ including p = 2)<br>- The curve parameters a and b (may have a different shape)<br>- The generator G<br>- The order q of the generator<br>- The co-factor: h = #Ɛ/q |

**Scalar Multiplication**

Scalar multiplication is the ECC counterpart of exponentiation. Computing [a]G in a naive way takes a-1 point additions. Infeasible if a and the coordinates of G are hundreds of bits long. The ECC counterpart of square-and-multiply is double-and-add.

**Projective Space**

Remarkable: O ∈ Ɛ but no solution of the Weierstrass equation that defines a subset of the **affine plane**: {(x,y) ∈ GP(p) x GP(p)}.

**The projective plane $P^2$ over a field K**

- Set of quiv. classes of triplets (X,Y,Z) (al in K) excluding (0,0,0). The equivalence relation is defined as $(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2) \Leftrightarrow \exists \lambda \in K \setminus \{0\}: (X_1, Y_1, Z_1) = (\lambda X_2, \lambda Y_2, \lambda Z_2)$

(X:Y:Z) is the equivalence class containing (X, Y, Z). Each class (X:Y:Z) corresponds to a point. If $Z \neq 0$ this is the affine point $(x,y) = (X \times Z^{-1}, Y \times Z^{-1})$. Classes (X:Y:0) are "points at infinity". Substitution of x by X/Z and y by Y/Z in the Weierstrass equation and multiplication by $Z^3$:

$$Y^2Z = X^3 + aXZ^2 + bZ^3$$

(X:Y:Z) are homogeneous coordinates. Intersection of the curve with line at infinity Z = 0 is (0:1:0).

- Neutral element: O = (0:1:0)
- Inverse: -(X:Y:Z) = (X:-Y:Z)

**Intuition: (X/R:0:Z) = (X:0:ZxR)**

**Key pair generation in Elliptic Curve Cryptography (ECC)**

Let $a \leftarrow \mathbb{Z}/q\mathbb{Z}$
$A \leftarrow [a]G$

**Elliptic Curve Diffie-Hellman (ECDH)**

| | Alice | | Bob |
|---|---|---|---|
| have in advance: | $\mathcal{E}, G, (q)$ | | $\mathcal{E}, G, (q)$ |
| | $a \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | $b \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ |
| | $A \leftarrow g^a$ | | $B \leftarrow g^b$ |
| | | $\xrightarrow{\text{Alice}, A}$ | |
| | | $\xleftarrow{\text{Bob}, B}$ | |
| | $P \leftarrow [a]B$ | | $P \leftarrow [b]A$ |

Alice and Bob arrive at the same shared secret pont P:

$$P = [a]B = [a][b]G = [ab]G = [b][a]G = [b]A$$

As shared secret one takes the x-coordinate of the shared point P. Alice and Bob derive key(s) from secret: $K \leftarrow$ ("KDF"; $x_p$)

**Elliptic Curve ElGamal**

| Alice | | Bob |
|---|---|---|
| $\mathcal{E}, G, (q), B$ | | $\mathcal{E}, G, (q), b, B(=[b]G)$ |
| $a \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | |
| $A \leftarrow [a]G$ | | |
| $C \leftarrow M + [a]B$ | $\xrightarrow{\text{Alice},(C,A)}$ | $M \leftarrow C - [b]A$ |

Cryptogram consists of two points on the curve: 4 affine coordinates. Reduce data overhead by using compressed representation:

- x-coordinate and parity of y: y mod 2
- Requires reconstruction of y-coordinate by receiver

Reconstruction: compute $x^3 + ax + b$ and take its square root.

**Elliptic Curve Schnorr authentication protocol**

| Alice | | Bob |
|---|---|---|
| $\mathcal{E}, G, q, A, a$ | | $\mathcal{E}, G, q$ (Alice: $A$) |
| $v \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ | | |
| $V \leftarrow [v]G$ | $\xrightarrow{\text{Alice}, V}$ | $c \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ |
| | $\xleftarrow{c}$ | |
| $r \leftarrow v - ca$ | $\xrightarrow{r}$ | $V \stackrel{?}{=} [r]G + [c]A$ |

Just a different cyclic group, commitment V is now much shorter and can be even shorter with compressed point representation.

**Elliptic Curve Signature Algorithm (ECDSA):** is probably the most implemented DL signature algorithm.

**EdDSA**

- Ed stands for Edwards curve
- It derives ephemeral key v from message
    - For this the private key is extended with a secret k

- This avoids weaknesses due to bad randomness but introduces other potential vulnerabilities

| Alice | | Bob |
|---|---|---|
| $\mathcal{E}, G, q, A, a, k$ | | $\mathcal{E}, G, q$ (Alice: $A$) |
| $v \leftarrow \mathrm{h}(k; m),\ V \leftarrow [v]G$ | | |
| $c \leftarrow \mathrm{h}(\mathcal{E}; G; A; V; m)$ | | |
| $r \leftarrow v + ca$ | $\xrightarrow{m,(r,V)}$ | $c \leftarrow \mathrm{h}(\mathcal{E}; G; A; V; m)$ |
| | | $[r]G \stackrel{?}{=} V + [c]A$ |

ECC is probably the most widespread public-key crypto (e.g. handshake in TLS 1.3, SSH, signatures in Bitcoin and other cryptocurrencies,..ect).

## Lecture 13 - RSA

We define $(\mathbb{Z}/n\mathbb{Z})^* = \{m \mid m \in (\mathbb{Z}/n\mathbb{Z})^* \text{ and } \gcd(m,n) = 1\}$

| **Definition Euler's totient function** |
|---|
| Euler's totient function of an integer n, denoted $\varphi(n)$, is the number of integers smaller than n and coprime to n |

- For prime p, all integers 1 to p-1 are coprime to p: $\varphi(p) = p-1$
- If n = a*b with a and b coprime: $\varphi(a*b) = \varphi(a)\varphi(b)$
- For the power of a prime $p^k$: $\varphi(p^k) = (p-1)p^{k-1}$
- Computing $\varphi(n)$:
    - Factor n into primes and their powers
    - Apply $\varphi(p^k) = (p-1)p^{k-1}$ to each of the factors

Computing $\varphi(n)$ is as hard as factoring n

| **Euler's theorem** |
|---|
| If $\gcd(x,n) = 1$ then $x^{\varphi(n)} \equiv 1 \bmod n$ |

Euler's theorem can be used for computing inverses in $(\mathbb{Z}/n\mathbb{Z})^*$ with exponentiation:

$$x^{-1} = x^{\varphi(n)-1} \bmod n$$

**RSA**

Keys: public key $(n,e)$ and private key $(n,d)$ with

- Modulus $n = pq$ with $p$ and $q$ two large primes
- Public exponent $e$ that satisfies $\gcd(e, \varphi(n)) = 1$

- Private exponent $d$ with $ed \equiv 1 \bmod \varphi(n)$

Bob encrypts a message $m \in (\mathbb{Z}/n\mathbb{Z})^*$ for Alice

| Bob | | Alice |
|---|---|---|
| Alice's public key $(n, e)$ | | Alice's private key $(n, d)$ |
| $c \leftarrow m^e \bmod n$ | $\xrightarrow{c}$ | $m' \leftarrow c^d \bmod n$ |

Alice signs a message $m \in (\mathbb{Z}/n\mathbb{Z})^*$

| Alice | | Bob (or anyone) |
|---|---|---|
| Alice's private key $(n, d)$ | | Alice's public key $(n, e)$ |
| $s \leftarrow m^d \bmod n$ | $\xrightarrow{\text{Alice}, m, s}$ | $m \stackrel{?}{=} s^e \bmod n$ |

$x = y^d$ when $y = x^e$ because:

(1) Substitution gives $y^d = (x^e)^d = x^{ed}$
(2) Euler's theorem says $x^{\wedge}(\varphi(n)) = 1$ so $x^{ed} = x^{ed \bmod {}^{\wedge}(\varphi(n))}$
(3) By the definition of $d$ we have $ed \bmod \varphi(n) = 1$
(4) It follows $x^{ed \bmod {}^{\wedge}(\varphi(n))} = x$

Computation of $d$ from $e$ and $p,q$

- Inverse of $e$ modulo $\varphi(n) = (p-1)(q-1)$
- It only exists if $\gcd(e, p-1) = 1$ and $\gcd(e, q-1) = 1$

Security of textbook RSA:

- Encryption breaks down if Eve can find the $e^{th}$ root of c
- Signing breaks down if Eve can find the $e^{th}$ root of some chosen m
- This is called inverting RSA

Security of textbook RSA requires factoring to be hard. (Turns out textbook RSA is actually non-secure if factoring is hard)

**Chinese Remainder Theorem (CRT)**

Let n = p * q with p,q primes, then the map
$x \rightarrow (x_1, x_2)$ with $x \in \mathbb{Z}/n\mathbb{Z}$, $x_1 = x \bmod p$ and $x_2 = x \bmod q$
Defines a ring isomorphism:
$\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$

Which means: any sum or product of elements in $\mathbb{Z}/n\mathbb{Z}$ is matched by that of the corresponding elements in $\mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$

CRT is used for computing x from $(x_1, x_2)$

**Chinese Remainder Theorem RSA-specific (CRT)**

If n = p * q with p,q primes, then the system of congruence relations:
$x \equiv x_1 \pmod{p}$
$x \equiv x_2 \pmod{q}$
Has a unique solution $x \in \mathbb{Z}/n\mathbb{Z}$ for any couple of integers $(x_1,x_2)$

The mapping from x to $(x_1,x_2)$ is injective: different values x cannot give equal tuples $(x_1,x_2)$
The number of possible values for x and $(x_1,x_2)$ is both n and hence the mapping in a bijection

**CRT formula (RSA-specific)**

The solution $x \in \mathbb{Z}/n\mathbb{Z}$ with n = pq for
$x \equiv x_1 \pmod{p}$
$x \equiv x_2 \pmod{q}$
With p,q primes is given by
$x = u_1x_1 + u_2x_2 \bmod n$
With $u_1 = (q^{-1} \bmod p) * q$ and $u_2 = (p^{-1} \bmod q) * p$
(the constants $u_i$ can be used for any vector $(x_1, x_2)$)

$$u_1 \equiv 1 \pmod{p} \qquad u_1 \equiv 0 \pmod{q}$$
$$u_2 \equiv 0 \pmod{p} \qquad u_2 \equiv 1 \pmod{q}$$

**Garner's algorithm**

INPUT: (p,q) with p > q and $(x_1, x_2)$
OUTPUT: x
$I_q = q^{-1} \bmod p$
$t = x_1 - x_2 \bmod p$
$x = x_2 + q * (t * i_q \bmod p)$

**RSA private key exponentiation in the product ring**

Given y we must compute x that satisfies $y = x^e \bmod pq$. For $(x_1, x_2) \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$ we get $y_1 = x_1^e \bmod p$ and $y_2 = x_2^e \bmod q$. These are solved by:

- $x_1 \leftarrow y_1^{dp} \bmod p$ with dp the solution of $ed_p \equiv 1 \pmod{p-1}$
- $x_2 \leftarrow y_2^{dq} \bmod q$ with dq the solution of $ed_q \equiv 1 \pmod{q-1}$

This works for **all** values of $y_1$ and $y_2$ (including 0). Thanks to CRT, it follows that $x \leftarrow y^d \bmod n$ always works with:

- d mod (p-1) = dp
- d mod (q-1) = dq

Using CRT speeds up RSA private key exponentiation with a factor 4.

**RSA key pair generation**
Generating an RSA key pair with given modulus length |n| = l
Procedure to generate an RSA key pair:
(1) Choose e: often this is fixed to $2^{16} + 1$ by the context
(2) Randomly choose prime p with |p| = l/2 and gcd(e, p-1) =1
(3) Randomly choose prime q with |pq| = l and gcd(e, q-1) =1
(4) Compute modulus n = p * q
(5) Compute private key exponent(s) (d for no CRT and dp, dq, $i_q$ for CRT)

| **Prime counting function $\pi$(n)** |
|---|
| $\pi(n) = \#p_i$, $p_i \leq n$ where $p_i$ is a prime (e.g. $\pi(100) = 25$ |

**RSA security: advances of factoring over time**
- State of the art factoring: two important aspects
    - Reduction of computing cost: Moore's law
    - Improvements in factoring algorithms
- Factoring algorithms
    - Sophisticated algorithms involving many subtleties
    - Two phases:
        - Distributed phase: equation harvesting
        - Centralized phase: equation solving
    - Best known: general number field sieve (GNFS)

For 128 bits of security, NIST currently advises 3072-bit modulus

**Using RSA**
Enhancements for textbook RSA:
- Encryption randomized by including random r: m ← PIN; r
- For freshness: include challenge c from card: m ← PIN; r; c

Solutions for RSA encryption:
- Apply a hybrid scheme
    - Use RSA encrypting a symmetric key K
    - Encrypt (and authenticate) with symmetric cryptography
- Sending an encrypted key
    - Randomize message before encryption
    - Add redundancy and verify it after decryption
    - If NOK, return error

Hybrid encryption scheme using RSA-KEM
- The hybrid encryption scheme including RSA-KEM is proven IND-CPA secure if
    - Invertin RSA is hard

- h is indistinguishable from RO
- The symmetric cryptosystem is secure

| Bob has Alice's public key $(n, e)$ | | Alice with private key $(n, d)$ |
|---|---|---|
| $r \xleftarrow{\$} \mathbb{Z}/n\mathbb{Z}$ | | |
| $c \leftarrow r^e \bmod n$ | | |
| $K \leftarrow \mathrm{h}(\text{"KDF"}; r)$ | | |
| $CT \leftarrow \mathrm{Enc}_K(m)$ | $\xrightarrow{c, CT}$ | $r \leftarrow c^d \bmod n$ |
| | | $K \leftarrow \mathrm{h}(\text{"KDF"}; r)$ |
| | | $m \leftarrow \mathrm{Dec}_K(CT)$ |

**Problems of textbook RSA signatures**

- RSA malleability
    - Given signatures $s_1 = m_1^d$ and $s_2 = m_2^d$, Eve can sign $m_3 = m_1 * m_2 \bmod n$ by computing $s_3 = s_1 * s_2 \bmod n$.
      $M_m^d = (m_1 \times m_2)^d = m_1^d \times m_2^d = s_1 \times s_2$
    - This is forgery: signing without knowing private key
- Limitation on message length

**Full domain has (FDH) RSA signature**

| Alice with private key $(n, d)$ | | Bob with Alice's public key $(n, e)$ |
|---|---|---|
| $H \leftarrow \mathrm{h}(m)$ | | |
| $s \leftarrow H^d \bmod n$ | $\xrightarrow{\text{Alice}, m, s}$ | $H \leftarrow \mathrm{h}(m)$ |
| | | $H \stackrel{?}{=} s^e \bmod n$ |

Secure against forgery if:

- Inverting RSA is hard
- The hash function behaves like a random oracle
- WIth co-domain of h equal to $\mathbb{Z}/n\mathbb{Z}$

Can easily be realized using XOF: generate output string longer than the length of n, interpret the result as an integer and reduce modulo n.

# Lecture 14 - DL

| **Elliptic curve DL problem** |
|---|
| Determine a given G and A $\in \langle g \rangle$ with [a]G = A |

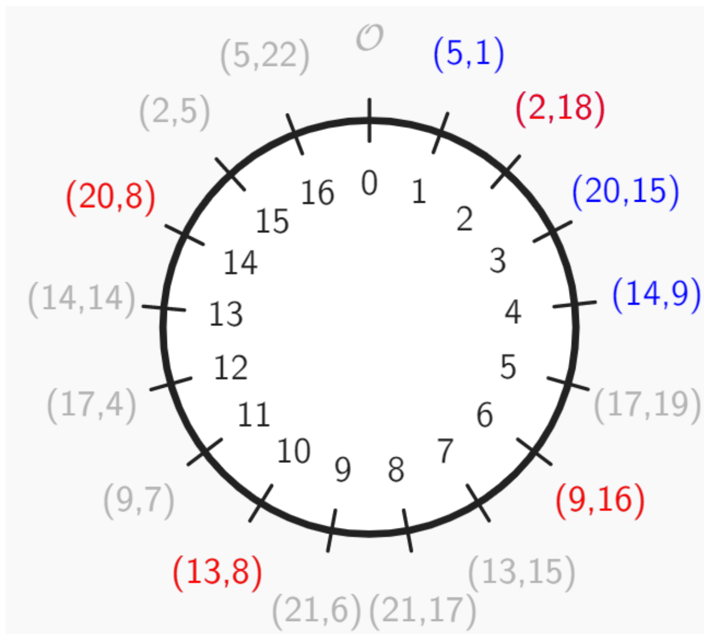
There are two types of methods: generic methods (work for any cyclic group, including EC) and specific methods (exploit properties of the group).

**Baby-step giant-step (Generic method)**

```
Input: A, G and table size m
Output: a that satisfies [a]G = A
i ← 0, X ← G, T ← {(X, 1)}
repeat
     i ← i + 1, X ← X + G, T ← T U {(X,i)}      (baby step)
until i = m
j ← 0, Y ← A
repeat
     j ← j + 1, Y ← Y - [m]G                    (giant step)
until ∃(X, i) ∈ T with X = Y
return i + mj
```

Example on the right:
Take E(GP(23)): $y^2 = x^3 - x - 4$
Say G = (5,1) and A = (20,8)

m = 4, baby steps, giant steps A - [3*4]G = [2]G ⇒ a = 14




Baby steps:
- Compute the values of [i]G for i up to m
- Store them in table T
- Work: m point additions
- Storage: m points

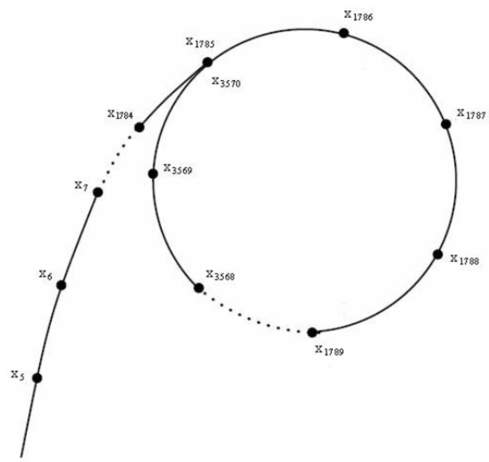
Giant steps
- Compute A, A - [m]G, A - [2m]G, etc.
- Until the point A - [jm]G is also in table T
- Expected work: ord(G)/2m point additions and table checks

Matching points satisfies [i]G = A - [jm]G so A = [i + mj]G
# point addition minimized by taking m ≈ √(ord(G))
Storage and table check cost may favor m ≪ √(ord(G))

**Pollard's ρ method (Generic method):**
Requires a transformation f over ℤ/qℤ x ℤ/qℤ with q = ord(G) .
- Given $(c_i, d_i)$, it computes $(c_{i+1}, d_{i+1}) = f(c_i, d_i)$
- Let $P_i = [c_i]A + [d_i]G$ then
    - f shall define a mapping f' over⟨g⟩: $P_{i+1} = f'(P_i)$
    - f' shall behave like a random transformation

Algorithm:
- Pick random couple $(c_0, d_0)$
- Compute the sequence $(c_i, d_i)$ with $(c_i, d_i) = f(c_{i-1}, d_{i-1})$
- Stop if for some $i < j = P_i = P_j$
- Now $[c_i]A + [d_i]G = [c_j]A + [d_j]G$ or $[c_i - c_j]A = [d_j - d_i]G$
- So if $(c_i, d_i) \neq (c_j, d_j)$ it follows that $a = (d_j - d_i)/(c_i - c_j)$

It is unlikely this ends up in $(c_i, d_i)= (c_j, d_j)$




Assume f' behaves like a random mapping:
- Probability that $P_i$ equals one of the previous points: $(i-1)/q$
- Probability there is a collision after n iterations ≈ $n^2/2q$
- Expected value of n until the collision: $\sqrt{(\pi q)/2}$

Storing all points $P_i$:
- Requires about $\sqrt{q}$ storage comparison
- Not better than baby-step giant-step

Reducing storage with method of distinguished points
- Only store point sthat have some rare property
- E.g. x-coordinate ends in l trailing zeroes
- Reduces storage size by a factor $2^{-l}$
- Expected overshoot of $2^{l-1}$ additional iterations into the loop
- Taking $2^l$ close to $\sqrt{q}$ solves storage problem

Choosing f:
- Partitioning approach:
    - Partition$\langle g \rangle$in s classes of similar size
    - Have a different function f (and f') per class
    - Choose classes so that it is easy to find the class of a point
- Classical choice: s = 3
    - $S_0$: f(c,d) = (2c, 2d) so f'(P) = [2]P
    - $S_1$: f(c,d) = (c + 1, d) so f'(P) = P + A
    - $S_2$: f(c,d) = (c, d + 1) so f'(P) = P + G
- ECC-oriented chose: s = 20
    - Per class $S_k$ randomly choose $a_k, b_k \in \mathbb{Z}/q\mathbb{Z}$
    - $S_k$: $f(c,d) = (c + a_k, d + b_k)$ so $f'(P) = P + M_k$ with $M_k = [a_k]A + [b_k]G$

**Pohlig-Hellman (Generic method)**

Let $ord(G) = p_1p_2$ with $p_1$ and $p_2$ coprime
- We look for a that satisfies [a]G = A for given G and A
- Multiply both sides by $[p_1]$: $[p_1][a]G = [p_1]A$
    - Let $G_{p2} = [p_1]G$ and $A_{p2} = [p_1]A$
    - We have $ord(G_{p2}) = p_2$ and $A_{p2} \in \langle G_{p2} \rangle$
    - If a satisfied [a]G = A, it also satisfies $[a]G_{p2} = A_{p2}$

    - If a is a solution of $[a]G_{p2} = A_{p2}$ so is a mod $p_2$
    - So solving $[a]G_{p2} = A_{p2}$ gives $a_{p2}$ = a mod $p_2$
    - With pollard's ρ this costs roughly $\sqrt{p_2}$ computations
- Multiply both sides by $[p_2]$: $[a][p_2]G = [p_2]A$
    - Along similar lines this gives $a_{p1}$ = a mod $p_1$
    - Costs roughly $\sqrt{p_1}$ computations
- Compute a from $a_{p1}$ and $a_{p2}$ using CRT

If ord(G) is composite, pohling hellman allows to:
- Solve the discrete log problem for each of the factors of ord(G)
- Combine the results with CRT

For each prime power $p^n$ | ord(G), work factor is $\sqrt{p}$
- If n = 1, this is straightforward
- If n > 1: out of scope for course

Pohlig-Hellman DL algorithm is the reason why groups⟨G⟩for DL crypto have prime order.

**Index Calculus (Specific method)**

Works for⟨g⟩a subgroup of multiplicative groups $(\mathbb{Z}/p\mathbb{Z})^*$. Index calculus is much faster than generic attacks and scales better with increasing p. Forces us to take $p \geq 2^{3072}$ for 128 bits of security. Works even better for subgroups of the multiplicative groups in prime power fields $GP(p^2)$. Index calculus does not work on elliptic curve groups.