Crypto for PETs – Part 1

Jorge Cuellar

WS 18-19

Notation

Key space	$\mathscr{K} = \{0,1\}^n$ where n is "small"
Shared Key	k
Public Key of A	pk_A P_A
Private Key of A	sk_A p_A
Message space	$\mathscr{M} = \{0,1\}^*$
Cipher space	$\mathscr{C}$
Key generator	$\mathscr{G} : () \rightarrow \mathscr{K}$
Encryption function	$\mathscr{E} : \{\mathscr{K} \times \mathscr{M}\} \rightarrow \mathscr{C}$
Decryption function	$\mathscr{D} : \{\mathscr{K} \times \mathscr{C}\} \rightarrow \mathscr{M}$
Random choice	$x \leftarrow \mathscr{S}$
Run algorithm A	$x \leftarrow A(i)$
	Or: $x \overset{A}{\leftarrow} i$

Notation, Comments

Key space	(1)	$\mathscr{K} = \{0,1\}^n$ where n is "small"
Message space	(2)	$\mathscr{M} = \{0,1\}^*$
Key generator	(3)	$\mathscr{G} : () \to \mathscr{K}$

1. The length of the key is considered small
   - but the number of keys is large (brute-force attacks are impossible)
2. The length of a message can be larger than the length of the key
   - usually it is larger, but – in some cases – it is not
3. $\mathscr{G}$ is a randomized algorithm that takes no input
   - You may imagine () as a set that only contains one element
     - whose name is irrelevant
     - You may also write $() = \{\bullet\}$

Notation, Comments

Random choice	(4)	$x \leftarrow \mathscr{S}$
Run algorithm A	(5)	$x \leftarrow A(i)$ or $x \stackrel{A}{\leftarrow} i$

1. $x \leftarrow \mathscr{S}$ means:
   - let x be uniformly randomly choose out of the set $\mathscr{S}$
2. $x \leftarrow A(i)$ or $x \stackrel{A}{\leftarrow} i$ means:
   - let x be the output of the possibly non-deterministic but
     - *efficient algorithm A* running on input i

Crypto Literature: Books

The following are links (you can click on them)

- Jonathan Katz and Yehuda Lindell. An Introduction to Modern Cryptography
- Oded Goldreich. Foundations of Cryptography.

Crypto Literature: Lecture notes

The following are links (you can click on them)

- Haitner-Applebaum
- Ran Canetti
  - Foundation of Cryptography (The 2008 course) and
  - On Chernoff and Chebyshev bounds.
- Salil Vadhan Introduction to Cryptography.
- Luca Trevisan Cryptography.
- Yehuda lindell Foundations of Cryptography.
- Ryan O'Donnell Probability and Computating

PETS Literature

See the web pages of following people:

- George Danezis, Univ College London
- Mark D. Ryan, Birmingham
- Claudia Diaz, KU Leuven
- Seda Gurses, Princeton
- Frank Kargl, Ulm
- Alessandro Acquisti, CMU
- Carmela Troncoso, EPFL
- Frank Piessens, KU Leuven
- Nicola Zannone, Eindhoven
- Simone Fischer Huebner, Karlstad

PETS Literature

See the pages of following Seminars/Workshops

- IEEE Security & Privacy
- Annual Privacy Forum
- IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)
- ACM Conference on Data and Application Security and Privacy
- Annual ACM workshop on Privacy in the Electronic Society
- CPDP (Computers, Privacy and Data Protection)

PETS Literature

See the following Projects

- PRIPARE (EU)
- Harvard University Privacy Tools Project (https://privacytools.seas.harvard.edu)
- https://privacyflag.eu/
- https://abc4trust.eu/
- PRIME Project FP6-IST. Privacy and Identity Management for Europe
- PrimeLife - Privacy and Identity Management in Europe for Life (primelife.ercim.eu)
- The Free Haven Project (https://freehaven.net/)

The flavor of security: PRG

To encrypt m with a one-time-pad $e := x \oplus m$

A random string x of length $|m|$, the size of m, is required

- $|x| = |m|$ could be relatively large, say $n := |x| = 10^6$ bits

This has two problems:

1. The key x is very long: how to distibute securely the key?
2. Finding random numbers may be difficult
   - obtaining $\ell = 100$ random bits is much easier than $n = 10^6$ bits

Pseudo-Random Generators (PRG)

. . . are deterministic algorithms that

- given ℓ random bits, say $\ell = 100$
- construct $n = 10^6 \gg \ell = 100$ bits that
  - "you can't distinguish from random"

The flavor of security: PRG

Compare a truly random and a pseudo-random string

$$x \in \{0,1\}^n \leftarrow \{0,1\}^n$$
$$x \in \{0,1\}^n \stackrel{\Psi}{\leftarrow} (k \leftarrow \{0,1\}^\ell)$$

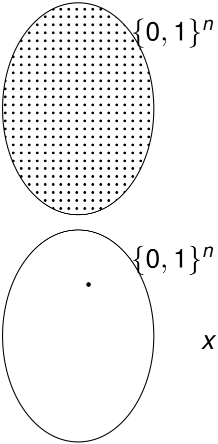
We have two distributions over $\{0,1\}^n$:

1. choose uniformly a random string in $\{0,1\}^n$
   - $\mathscr{D}_1 = \text{uniform}(\{0,1\}^n)$
2. In the second case: first choose uniformly a "seed" (or "key") in $\{0,1\}^\ell$
   - then map that key to an element of $\{0,1\}^n$,
     - via a deterministic efficient algorithm $\Psi : \{0,1\}^\ell \to \{0,1\}^n$
   - $\mathscr{D}_2 = \Psi(\text{uniform}(\{0,1\}^\ell))$

Those two distributions are very different, yet:

- the PRG Ψ is secure $\Leftrightarrow \mathscr{D}_1 \approx \mathscr{D}_2$
  - that is, the distributions are "computationally indistinguishable"

$\mathscr{D}_1 = \mathscr{D}\{x \mid x \leftarrow \{0,1\}^n\} \approx \mathscr{D}_2 = \mathscr{D}\{\Psi(k) \mid k \leftarrow \{0,1\}^\ell\}$



$x \leftarrow \{0,1\}^n$

$x \stackrel{\Psi}{\leftarrow} k \leftarrow \{0,1\}^\ell$

"From a helicopter", they are clearly distinguishable, but - samples from them are not

$\mathcal{D}_1 = \mathcal{D}\{x \mid x \leftarrow \{0,1\}^n\} \approx \mathcal{D}_2 = \mathcal{D}\{\Psi(k) \mid k \leftarrow \{0,1\}^\ell\}$

Note that the two distributions are very different

- in the first one, all points have the same positive probability
- in the second one,
  - only a very small fraction of points ($\{0,1\}^\ell \lll \{0,1\}^n$
    - has positive probability
  - an overwhelming proportion of points have probability zero

Nevertheless, given 2 samples, one from each

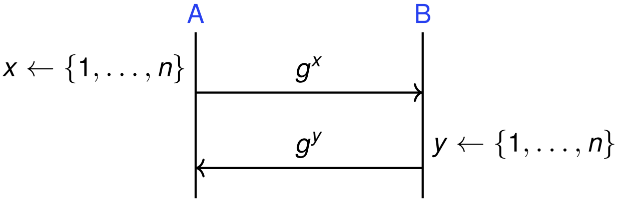
- no polynomial algorithm can distinguish which sample is which

Note:

1. the number of points in both is huge: $2^\ell, 2^n$, where $n = p(\ell)$, for some polynomial
   - $2^\ell, 2^n \geq p(n)$, for any polynomial
   - $\ell \ll n$
2. the points in the second distribution
   - show no structure

The flavor of security: DH

- The single most important building block in cryptography
  - Constructing a secure channel from an insecure channel



Both can calculate $k = (g^x)^y = g^{(x \cdot y)} = g^{(y \cdot x)} = (g^y)^x$

Figure: Diffie-Hellman Key Agreement

Diffie-Hellman (DH)

- As presented, DH has one problem
  - This is an unauthenticated DH
  - Neither A nor B is assured "who is sitting on the other side"
- A man-in-the-middle is possible
  - ! Exercise!
- A simple way of securing it, is by
  - signing at least one of the shares (g^x), (g^y)
  - Say, B does not only send (g^x) to A
    - she also sends its signature,
    - so it must come from B

DH is secure against a passive attacker

If an attacker only sees a DH exchange

- (without playing Man-in-the-Middle)
- then he does not learn the key; more precisely:
  - he cannot distinguish the key from any strange random number

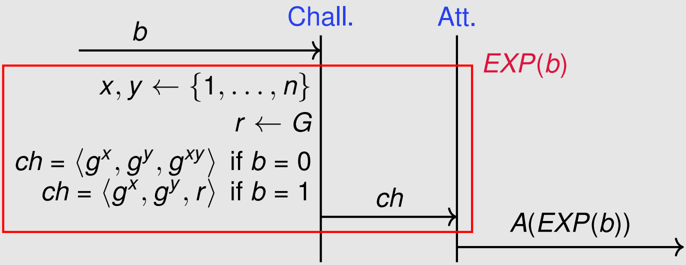
If the attacker has to choose between

- the real key that the parties A and B have agreed upon
  - and a random number of the same size
  - he will have prob $\approx \frac{1}{2}$ of guessing correctly

This is formalized as a game (next slide)

The flavor of security: DDH as a Game

Consider the game between a "challenger" and an "adversary" (or "attacker")



The adversary is able to win the game with prob. significantly $> \frac{1}{2}$

- iff he is able to distinguish the distributions
  - DH-triples: $\mathscr{D}_1 = \{\langle g^x, g^y, g^{xy} \rangle \,|\, x, y \leftarrow \{1, \ldots, n\}\}$
  - Random triples: $\mathscr{D}_2 = \{\langle g^x, g^y, r \rangle \,|\, x, y \leftarrow, r \leftarrow G\}$

Hard problems: Decisional Diffie-Hellman Problem

What does it mean that DDH is hard?

Given any arbitrary PPT (pol, poly-time) algorithm A

- and G a group with generator g as above

Choose (Note: the choices are random $\Rightarrow$ independent of A)

- $x \leftarrow \{1 \ldots |G|\}$
- $y \leftarrow \{1 \ldots |G|\}$
- $r \leftarrow G$
- $b \leftarrow \{0, 1\}$

Construct the triple (called "challenge"):

$$ch = \begin{cases} \langle g^x, g^y, g^{xy} \rangle & \text{if } b = 0 \\ \langle g^x, g^y, r \rangle & \text{if } b = 1 \end{cases}$$

Hard problems: Decisional Diffie-Hellman Problem

What does it mean that DDH is hard? (Cont)

- Let us say that "A wins" if $A(ch) = b$
  - thus the algoritm A guessed correctly the bit b
    - (Note that A can be deterministic or not)

A has always a probability $\frac{1}{2}$ of winning

- (Do not look at ch, simply trow a coin)
- But A could have a bit of advantage ε

$$P[A \text{ wins } | x, y, r, b \text{ chosen as above}] = \frac{1}{2} + \varepsilon$$

Note that ε may depend on the algorithm A

- but also on ℓ – the "size of the input" of the algorithm
  - = the size (length) of the challenge

"Winning" vs. "distinguishing"

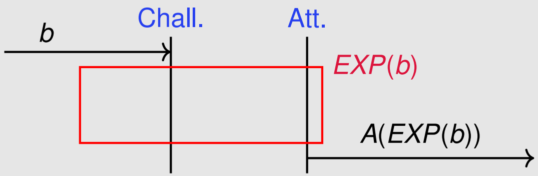
Instead of considering if an algorithm can win

- it results easier to ask if an algorithm can distinguish the two cases $b = 0$, $b = 1$

The definition is (up to a multiplicative constant on ε) equivalent:

- if an algorithm can win, it distinguishes
- if an algorithm distinguishes, either it or its negation wins

$Adv(A, EXP(0), EXP(1))$



$$Adv(A, EXP(0), EXP(1)) = \left| P[A(EXP(1) = 1)] - P[A(EXP(0) = 1)] \right|$$

The flavor of security: Hard Problems

The following problems are hard

1. DDH
2. Distinguishing a Pseudorandom from a random number
3. Factoring numbers which are the product of two large primes
4. Finding the logarithm of elements in a finite ("complicated") group

The flavor of security: large and small ns

The chance of winning the "6 in 49" Jackpot is

- 6 correct: 1 in $13,983,816 < 2^{24}$
- With only one ticket, the probability is really low

Winning the lottery by brute force

- With tens of millions of tickets, the probability of winning is high

What we want is to be secure against brute force

- ...from an attacker that can make
  - tens of millions of tries per second to hack some system
  - and he has lots of time to perform the attack

Hacking by brute force

- The number of seconds since the Big Bang is
  - about $4.32 \times 10^{17} < 2^{59}$
- Thus, assume an attacker makes
  - ten millions of tries per second 10^7
  - over a time comparable to the age of the universe
  - $\Rightarrow$ he makes in total $\approx 2^{80}$ tries
- What we want is that still such attackers have a
  - low probability of hacking the system, say 1 in 1 million $\approx 2^{20}$
- Thus we want systems in which you need roughly $\approx 2^{100}$ tries to crack it

2^{100} is a "large number"

Info The flavor of security: EC over $\mathbb{R}$

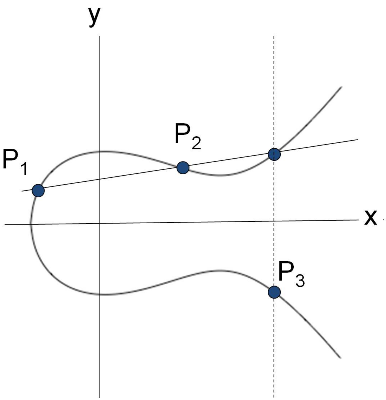


Figure: EC over $\mathbb{R}$. The "product" of two points in the EC is defined geometrically

Info Elliptic Curves over a finite filed

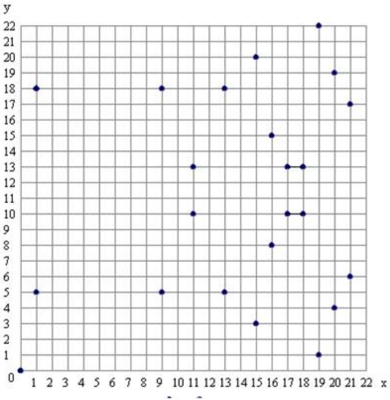


Figure: EC over a finite filed

Digests (Fingerprints or Indexes)

A digest (or a fingerprint) of a message (or file or bit sequence)

is an efficient deterministic algorithm $h : \{0,1\}^* \rightarrow \{0,1\}^n$

- maps data of arbitrary size, say a message or file, etc
  - to data of fixed size
- an calculates a not too short "checksum" or "fingerprint"

Digests (Fingerprints or Indexes)

The property that "defines" digests is:

if x and x' are messages (or files, or bit strings)

- chosen "totally independently", the one from the other
  - example: choose two files at random from a file disk
  - example: take two sentences at random in a novel
- then $digest(x) = digest(x') \Rightarrow x = x'$
  - with a high probability

Note that "totally independently" is not well defined

But it is ok if you can construct messages with the same digest

Digests (Fingerprints or Indexes)

Can be used as an index

- If x and x' have the same digest
- then "it is safe" to assume that x and x' are the same

Digests are used

- to construct "index tables" (also called "hash tables"),
  - where the index is the digest
    - to accelerate table or database lookup or
    - to detect duplicated records or files, etc

Digests (Fingerprints or Indexes)

- To find duplicates in a set of files:
  - calculate the digests of all files
    - but if the files are small, you do not need a digest
  - create a table: $\{(\text{index}_1, \text{location}_1), (\text{index}_2, \text{location}_2), \ldots\}$
  - sort the table
    - If two indexes are the same, then the files must be identical
- And: this gives us a very efficient way
  - of remember things we have seen
  - and recognizing them again,
- This is useful because the digest is small,
  - while the files or values we want to remember are big
    - if not, there was no problem to start with

Cryptographic Hashes

Digests vs Hashes

What we call *digest* is sometimes called *hash*

- but we reserve the word hash for *Cryptographic Hash Functions*
  - which have further properties

Cryptographic Hashes

Properties of Hashes

- preimage resistance
- second-preimage resistance
- collision resistance
- hiding (puzzle friendly)
- "uniform"

Preimage resistance as a game

Consider a challenger and an adversary, as before

- and a hash function: $h : \{0,1\}^* \to \{0,1\}^n$

The challenger chooses

- randomly $y \in \{0,1\}^n$
- and presents it to the adversary

The adversary tries to find *any string* x with $h(x) = y$

The probability of finding x should be negligible

- Note that it may be easy to find a preimage
  - for some particular values of y
- but "for almost all" y's it should be difficult

Second Preimage resistance as game

A technical problem

We can't say: the challenger chooses

- some random bit string in, say $\{0,1\}^*$
- this is an enumerable set,
  - there is no standard notion of "uniform distribution" in $\{0,1\}^*$

Thus the challenger chooses a random string

- in a finite subset of $\{0,1\}^*$
- but the random string should not be too small

Let $a, b \in \mathbb{N}$ with $n \leq a \leq b$

- the challenger chooses at random some bit string in
  - $\{0,1\}^{[a,b]} := \{x \in \{0,1\}^* \mid a \leq |x| \leq b\}$
    - = the set of bit strings of length $\geq a$ and $\leq b$

Second Preimage resistance as a game

The challenger chooses

- some random bit string
  - $x \in \{0,1\}^{[n,2n]}$
- and presents to the adversary
  - x, $h(x)$ (or only x, th adversary can calculate the hash)

The adversary tries to find

any second string $x' \neq x$ with $h(x') = h(x)$

The probability of finding x' should be negligible

Second-Preimage Resistance

"Almost all"

For some choices of $h(x)$

- it may be easy to find a preimage

or for some choices of x

- it may be easy to find a second preimage of $h(x)$

Collision resistance implies second-preimage resistance

- but does not guarantee preimage resistance

Cryptographic Hash Functions

- A hash function takes as input any string
  - of any size
- It produces a fixed size output
  - BitCoin for instance uses 256 bits
- The hash is efficiently computable:
  - in a polynomial (normally: linear) amount of time (on the length of the input), it calculates the output
- Thus, it is an efficient algorithm:

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

Properties of Cryptographic Hash Functions

- First property: Collision-resistance:
  - nobody normal (read: polynomial algorithm) can find two different values x and x' with the same hash
- In other words:
  - it is unfeasible to find $x \neq x'$, such that $h(x) = h(x')$
- BUT: Many collisions do exist
  - Infinite number (or a very large number) of possible inputs
  - But only 2^n possible outputs
- Just nobody "normal" can find collisions
  - ...we will see what that means

Cryptographic Hash Functions: Collisions

Collisions can not be found

- by "regular people" using "regular computers"
  - Note: this is the notion of "efficient attacker"
  - Here this means: in a sequential (normal) computer
  - you will need around $2^{n/2}$ steps to find a collision
    - if the hash is secure

A method that works, for sure, is:

- pick $2^n + 1$ distinct values, compute the hashes of them,
  - check if there are any two outputs are equal
- Since we have more inputs than possible output values
  - some pair of them must collide

Cryptographic Hash Functions: Collisions

- Birthday paradox: with 2^{130} inputs
  - there is already a 99.8% chance that there are collisions
- But this is a large number
  - for all practical purposes
    - We do not know – in practise – how to find a collision
- We only know – in principle – how to find a collision
  - but this method takes too long to matter
- (In theory, theory and practise are the same, but not in practise)

Cryptography works because of "hard problems"

If you know the secret and private keys

and everyone know public keys

- the algorithms for encryption, decryption, signing, etc
  - are polynomial on n, the length of the keys

If you do not know them

you may still, in principle, crack the system

- but those algorithms should not be better than "brute-force"
  - which takes exponentially long on the size of the keys

Thus, we are interested in numbers

- n that are "small", but
- whose exponentials 2^n are "large"

Are Cryptogr. Hash Functions Collision-free?

There is no collision free hash function

Because the domain is larger than the codomain

- For some hash functions
  - Many people have tried hard to find collisions
    - without success
- For some hash functions
  - collisions were eventually found
    - Example: MD5
    - It was then deprecated and phased out of practical use

Some "large" numbers

- $2^{140} = 10^{42}$ The number of instructions calculated
  - Assuming 10^{13} computers
    - more than 1000 computers per person
  - each one calculating 10^{12} instructions per second
    - much more than what we have today
  - since the beginning of the universe: 10^{17} sec
- $2^{265} = 10^{80}$ The estimated
  - number of atoms in the observable Universe
- $2^{389} = 10^{120}$ a.k.a. the "Shannon number":
  - An estimated lower bound on the game-tree complexity of chess

Algebra

- Euclid's algorithm
- The notion of group
- Generator
- $\mathbb{Z}_p^*$ and $\mathbb{Z}_{pq}^*$

Groups

- A group $(G, \circ)$ is a set G
  - with an associative operation $\circ$ on G
  - which has an identity (unit element) and inverses
- That is:
  - $\circ : G \times G \to G$, with:
    - $\forall h_1, h_2, h_3 \in G, (h_1 \circ h_2) \circ h_3 = h_1 \circ (h_2 \circ h_3)$
    - $\exists_e \forall h \in G, e \circ h = h \circ e = h$
    - $\forall h \in G, \exists h^{-1}$ such that $h \circ h^{-1} = e$
- We are interested only in commutative groups that is
  - $\forall h_1, h_2 \in G, h_1 \circ h_2 = h_2 \circ h_1$

Info Cyclic Groups

Starting with any element g in any group G

- consider the set of all powers of $g \in G$

This is a subgroup of G:

- it is denoted $\langle g \rangle$ and called the *subgroup generated by g*
- Note that this group $\langle g \rangle$ is always commutative
  - even if G is not

Info Order of an element

If $\langle g \rangle$ is finite

- its size is called
  - *the order of* g, and also
  - *the order of the subgroup* $\langle g \rangle$

Thus

- $\text{ord}(g) = \text{ord}(\langle g \rangle) = |\langle g \rangle| = \min\{i \mid g^i = e\}$

Info Cyclic Groups

A group G is cyclic if it has an element g s.th

- $G = \langle g \rangle$

Any finite cyclic group of order n is of the form:

- $G = \{e, \underbrace{g}, \underbrace{g \circ g}, \underbrace{g \circ g \circ g}, \ldots, \underbrace{g \circ g \circ g \circ g \circ \ldots \circ g \text{ } (n-1 \text{ times})}\}$
- $= \{e, \quad g, \quad g^2 \quad , \quad g^3 \quad , \ldots, \quad g^{n-1} \quad \}$

Notice that any two cyclic groups of the same order are isomorphic

- In particular any cyclic groups is isomorphic to some group of the form $(\mathbb{Z}_n, +_n)$ (next slide)

Info A very "simple" group

$\mathbb{Z}_n = \{0, 1, 2, 3, \ldots n-1\}$ with $+_n$ the sum modulo n as operation is a group for each $n \in \mathbb{N}$

- The size of the group is n
- This is a "simple group"
  - a group where all interesting operations are easy to evaluate - including the "discrete logarithm"
  - but it is isomorphic to cyclic groups where
    - the corresponding operations may be quite difficult

This may seem strange:

- G_1 and G_2 are isomorphic groups
  - operations in one group G_1 are simple and
  - the corresponding operations in G_2 are difficult

Info $G_1 = \langle \mathbb{Z}_n, + \rangle$ is "simple"

But $G_1 \cong G_2 = \langle g \rangle$, $g^n = 1$ may be not simple Given g, the isomorphism

- $G_1 \rightarrow G_2$ is easy to calculate (using exponentiation)
  - while the reverse isomorphism $G_2 \rightarrow G_1$ may be difficult to calculate
    - requiring the computation of a discrete logarithm

Info Examples of Groups

$\mathbb{Z}_p^*$ for some prime p

- is the set of elements
  - $\{1, 2, 3, \ldots p - 1\}$ under multiplication
- The size of the group is $p - 1$

$\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$

- $5 * 5 \equiv_7 25 \equiv_7 4$
- Inverses can be derived using Euclid's algorithm (later)
  - $3^{-1} \in \mathbb{Z}_7$ is 5 since $3 * 5 \equiv_7 15 \equiv_7 1$

$G = \{1, 2, 4\}$ is a subgroup of $\mathbb{Z}_7^*$

- But $\{1, 2, 4, 6\}$ is not:
  - $2 * 6 (\text{mod } 7) \notin G$

Elliptic Curve groups

Greatest Common Divisor (gcd); Euclid's algorithm

- Let $a, b \in \mathbb{N}$, then $\gcd(a, b)$
  - The *greatest common divisor* of a and b is:

$$\gcd(a, b) = \max\{d \in \mathbb{N} \mid (d \mid a) \text{ and } (d \mid b)\}$$

In words: it is the largest d that divides both a and b

- If $a, b \in \mathbb{Z}$, we can define:
  - $\gcd(a, b) = \gcd(|a|, |b|)$

Greatest Common Divisor (gcd); Euclid's algorithm

Note: There are 3 types of "|" in the previous slide:

- one used for set comprehension, as in $\{d \in \mathbb{N} \mid p(d)\}$
  - to denote the set of all d with the property $p(d)$
- $(d \mid a)$ to denote d divides a
- $|a|$, to denote the absolute value of a

Greatest Common Divisor (gcd); Euclid's algorithm

The residue of b modulo a, $\text{res}_a\, b$

- is the remainder (rest) of the division of b by a

If $a, b \in \mathbb{N}$ and $a \leq b$, then

- division gives two numbers $q, r \in \mathbb{N} \cup \{0\}$:
  - $b = qa + r$ with $0 \leq r < a$
  - This r is the residue of b modulo a: $r = \text{res}_a\, b$

Euclid's algorithm

Since $\gcd(a, b) = \gcd(|b|, |a|)$ and $\gcd(a, b) = \gcd(b, a)$

- We can assume that $a, b \in \mathbb{N}$ and $a \leq b$. Then:

$$\gcd(a, b) = \begin{cases} a & \text{if } \operatorname{res}_a b = 0 \\ \gcd(\operatorname{res}_a b, a) & \text{otherwise} \end{cases}$$

Euclid's algorithm

For two integers a, b not both zero, $\gcd(a, b) = ak + bl$ for some integers k, l

- Moreover, $\gcd(a, b)$ is the smallest positive integer of this form

Let $\langle a, b\rangle_{\mathbb{Z}} := \{k \cdot a + l \cdot b \mid k, l \in \mathbb{Z}\}$

$\langle a, b\rangle_{\mathbb{Z}}$ is the set of all *integer combinations* of a and b

- The given algorithm to calculate $\gcd(b, a)$
  - can also be used to calculate the $k, l \in \mathbb{Z}$
    - in the so-called "Bezout's identity": $\gcd(b, a) = k \cdot a + l \cdot b$
  - See next slide

Note

$a, b \in \langle a, b\rangle_{\mathbb{Z}}$

Calculating the coefficients of Bezout's identity

Thm

Euclid's algorithm for calculating $\gcd(a, b)$

- also provides $k, l \in \mathbb{Z}$ such that $\gcd(b, a) = k \cdot a + l \cdot b$

Each step of Euclids Algorithm transforms a pair of numbers

a_i, b_i into a new pair of numbers

- $a_{i+1} = \text{res}_{a_i}\, b_i,\ b_{i+1} = a_i$

The initial values $a_0 = a$ and $b_0 = b$ are in $\langle a, b\rangle_{\mathbb{Z}}$

- For each step, if $a_i, b_i \in \langle a, b\rangle_{\mathbb{Z}}$
  - then both $a_{i+1} = \text{res}_{a_i}\, b_i = (b_i - q \cdot a_i)$ and $b_{i+1} = a_i$ are in $\langle a, b\rangle_{\mathbb{Z}}$

By induction,

- all remainders in all steps of the algorithms are in for $\langle a, b\rangle_{\mathbb{Z}}$
  - and the coefficients can be iteratively calculated

Congruence, $\mathbb{Z}_n$

- Let $a, b \in \mathbb{Z}$ and $n \in \mathbb{N}$. We define
- $a \equiv_n b$ (also written as $a = b \pmod{n}$) by

$$a \equiv_n b \;\; :\Leftrightarrow \;\; n \mid (a - b) \;\; \Leftrightarrow \;\; \text{res}_n\, a = \text{res}_n\, b$$

$$\mathbb{Z}_n := (\mathbb{Z} \,/ \equiv_n) \; = \; \{0, 1, \ldots, n - 1\}$$

- with addition and multiplication modulo n

Inversion in $\mathbb{Z}_n$

- We are interested in $\mathbb{Z}_n$ with multiplication modulo n
  - but $(\mathbb{Z}_n, \times)$ is not a group
    - not all elements are invertible
- $x \in \mathbb{Z}_n$ is called invertible in $\mathbb{Z}_n$
  - if there is a $y \in \mathbb{Z}_n$ s.t.
  - $x \cdot y = 1$ in $\mathbb{Z}_n$
    - Such y is unique
    - is called the inverse of x
    - and is denoted by x^{-1}

Inversion in $\mathbb{Z}_n$

- Theorem:
  - $x \in \mathbb{Z}_n$ has an inverse if and only if $\gcd(x, n) = 1$
- Proof sketch:
  - $\gcd(x, n) = 1 \Leftrightarrow \exists_{a,b} a \cdot x + b \cdot n = 1 \Leftrightarrow \exists_a a \cdot x \equiv_n 1$
  - . . . in this case, x^{-1} can be calculated using Euclid's algorithm:
  - $x^{-1} = \text{res}_n\, a$, where a is a solution of
    - $a \cdot x + b \cdot n = 1$
  - This algorithm has run time $O(log^2 n)$

$\mathbb{Z}_n^*$

- $\mathbb{Z}_n^*$, the group of units modulo n
  - or the group of invertible elements in $\mathbb{Z}_n$ is thus:

$$\begin{aligned}\mathbb{Z}_n^* &:= \{x \in \mathbb{Z}_n \mid \gcd(x, n) = 1\}\\ &= \{x \in \mathbb{Z}_n \mid x, n \text{ are prime relative}\}\\ &= \{x \in \mathbb{Z}_n \mid x^{-1} \text{ exists}\}\end{aligned}$$

- Example: $\mathbb{Z}_{12}^* = \{1, 5, 7, 11\}$

Totient Function

- $\phi(n) := |\mathbb{Z}_n^*|$
  - ϕ is called the totient function
  - Note: $\phi(n)$ is the number of prime relatives to n
    - smaller than n
- Euler's theorem says that

$$a \in \mathbb{Z}_n^* \ (\Leftrightarrow \gcd(a, n) = 1) \ \Rightarrow a^{\phi(n)} \equiv_n 1$$

- Info Proof follows from Lagange Thm (later)

! $\mathbb{Z}_p^*$, $\mathbb{Z}_{pq}^*$, for p, q primes

- $\mathbb{Z}_n^*$ is the multiplicative group of
  - invertible elements in $\mathbb{Z}_n$
  - that is, the prime relative to n: $\mathbb{Z}_n^* = \{x \mid \gcd(x, n) = 1\}$
- In particular, for $n = p \cdot q$ (p, q primes):

$$\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\} = \mathbb{Z}_p \setminus \{0\}$$

$$\mathbb{Z}_{pq}^* = \mathbb{Z}_{pq} \setminus (\{0, p, 2p, 3p, \ldots\ (q-1)p\} \cup \{q, 2q, 3q, \ldots (p-1)q\})$$

$\mathbb{Z}_n^*$

- Example: $\mathbb{Z}_{15}^* =$
  - $\mathbb{Z}_{3 \cdot 5}^* = \{1, 2, \ldots, 14\} \setminus \{3, 6, 9, 12\} \setminus \{5, 10\} = \{1, 2, 4, 7, 8, 11, 13, 14\}$
- It follows that:
  - if p is prime $\phi(p) := p - 1$
  - if p, q are prime $\phi(pq) := (p - 1)(q - 1)$

Exponentiation

- To compute g^a efficiently, we use the following procedure:
- Determine $n = \log_2 a$
- Compute $g^{2i} = (g^i)^2$ for $i = 1, 2, 4, \ldots n$

$$g \to g^2 \to g^4 \to g^8 \to g^{16} \to g^{32} \ldots \to g^{2^n}$$

1. Let the binary representation of a be $a_n, a_{n-1}, \ldots a_2, a_1, a_0$
2. Now use the following to determine g^a :

$$g^a = (g^1)^{a_1} \cdot (g^2)^{a_2} \cdot \ldots \cdot (g^{2^n})^{a_n}$$

- Example: $53 = (110101)_2 = 2^0 + 2^2 + 2^4 + 2^5 = 1 + 4 + 16 + 32$
- Then: $g^{53} = g^{1+4+16+32} = g^1 \cdot g^4 \cdot g^{16} \cdot g^{32}$

Exponentiation

In other words,

- To compute g^a efficiently

$$g^a = \begin{cases} 1 & \text{if } a = 0 \\ (g^{a/2})^2 & \text{if } a \text{ is even} \\ g \cdot g^{a-1} & \text{if } a \text{ is odd} \end{cases}$$

It only takes $\leq 2 \cdot \log_2 a$ multiplications (in the group, e.g, modular multiplications)

- which is very fast

$\mathbb{Z}^*_{pq}$, for p, q primes

- For instance, the non-invertible elements in $\mathbb{Z}_{3\cdot 5}$ are
  - $\{0, 3, 6, 9, 12\} \cup \{0, 5, 10\}$ and therefore
    - $\mathbb{Z}^*_{15} = \mathbb{Z}^*_{3\cdot 5} = \{1, 2, 4, 7, 8, 11, 13, 14\}$
  - $\phi(15) = |\mathbb{Z}^*_{3\cdot 5}| = 8 = (5 - 1) \cdot (3 - 1)$

Inversion in $\mathbb{Z}_{pq}^*$, for p, q primes

- Euler's Theorem implies

$$\forall_{x \in \mathbb{Z}_n^*} x^{\phi(n)} \equiv_n 1$$

- Since $ord(x)$, the order of x in $\mathbb{Z}_n^*$, divides
  - $\phi(n)$, the order of $\mathbb{Z}_n^*$, it follows that there is a
    - $k \in \mathbb{Z}$ such that $ord(x) \cdot k = \phi(n)$
    - And then $x^{\phi(n)} = (x^{\text{ord}(x)})^k = 1^k = 1$
- Example: $7^{\phi(15)} = 7^{4 \cdot 2} = 7^8 = 5764801 = 384320 * 15 + 1 \equiv_{15} 1$
  - This theorem generalizes Fermat's Little Theorem and is the basis of the
    - RSA cryptosystem

Inversion in $\mathbb{Z}_{pq}^*$, for p, q primes

For any e, the function $(\cdot)^e : x \mapsto x^e$ is a permutation in $\mathbb{Z}_{pq}^*$

- If $e \cdot d \equiv_{\phi(pq)} 1$ then the functions
- $(\cdot)^e, (\cdot)^d : \mathbb{Z}_{pq}^* \to \mathbb{Z}_{pq}^*$:

$$(\cdot)^e : x \mapsto x^e$$

$$(\cdot)^d : x \mapsto x^d$$

- are inverse of each other
  - In other words, for all $x \in \mathbb{Z}_{pq}^*$

$$(x^e)^d = x, (x^d)^e = x$$

Inversion in $\mathbb{Z}^*_{pq}$, for p, q primes

- Since $e \in \mathbb{Z}^*_{pq}$
  - then $\gcd(e, (p-1)(q-1)) = 1$, and then
    - e has a multiplicative inverse mod$(p-1)(q-1)$
  - $d := e^{-1}$ can be found via Euclid's Algorithm
  - $ed = 1 + C(p-1)(q-1)$
    - but only if the factors p, q are known
- Let $y = x^e$, then
  - $y^d = (x^e)^d = x^{1+C(p-1)(q-1)} = x$
- Therefore $y \mapsto y^d$
  - is the inverse of $x \mapsto x^e$

$\mathbb{Z}^*_{pq}$, for p, q primes

- Recall $\mathbb{Z}^*_{15} = \mathbb{Z}^*_{3\cdot5} = \{1, 2, 4, 7, 8, 11, 13, 14\}$ and
  - $\phi(15) = |\mathbb{Z}^*_{3\cdot5}| = 8 = (5-1)\cdot(3-1)$
- The multiplication table for this group is:

1	2	4	7	8	11	13	14
2	4	8	14	1	7	11	13
4	8	1	13	2	14	7	11
7	14	13	4	11	2	1	8
8	1	2	11	4	13	14	7
11	7	14	2	13	1	8	4
13	11	7	1	14	8	4	2
14	13	11	8	7	4	2	1

$\mathbb{Z}_{pq}^*$, for p, q primes

- Notice that on the diagonal of the multiplication table
  - we find the set of squares (or "quadratic residues")
    - which is $(\mathbb{Z}_{15}^*)^2 = \{x^2 \mid x \in \mathbb{Z}_{15}^*\} = \{1, 4\}$
- Since $4^2 = 1$ (in $\mathbb{Z}_{15}^*$),
  - then $x^4 = 1$ for all x and
    - therefore $\mathbb{Z}_{15}^*$ is not cyclic

Info $\mathbb{Z}_p^*$ is cyclic

- Remember that $\mathbb{Z}_p^*$ has $p-1$ elements
- Another theorem of Euler says
  - $\mathbb{Z}_p^*$ is cyclic, that is: there is a $g \in \mathbb{Z}_p^*$, such that

$$\langle g \rangle := \{g^i : i \in \mathbb{Z}\} = \{1, g, g^2, g^3, \ldots, g^{p-2}\} = \mathbb{Z}_p^*$$

- Example: 3 is a generator in $\mathbb{Z}_7^*$:

$$\{1, 3, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\} = \mathbb{Z}_7^*$$

- But not every element is a generator:

$$\{1, 2, 2^2, 2^3, 2^4, 2^5\} = \{1, 2, 4\}$$

Info $\mathbb{Z}_p^*$ is cyclic

- More generally,

$$\mathbb{Z}_n^* \text{ is cyclic } \Leftrightarrow n = 2, 4, p^k, 2p^k$$

- where p^k is a power of an odd prime number
- A generator of this cyclic group is called
  - a primitive root modulo n
    - or a primitive element of $\mathbb{Z}_n^*$

Computationally Hard Problems

- The setting for cryptography is always the following:
  - One entity, or a set of them,
    - know one or several secrets related to each other
    - and perhaps also to some "public information"
    - known by all, honest parties as well as attackers
- If a party knows a secret,
  - he is able to perform an operation efficiently
    - that without knowing the secret
    - would be too complex or unfeasible to perform
- The idea of "a certain operation is easy"
  - if you know a certain secret
- but it is difficult if you don't
  - is usually expressed as a
    - "Computationally Hard Problems" or as a
    - "Cryptographic Assumption"

Discrete log problem (DLog)

- The discrete logarithm is
  - just the inverse operation of exponentiation
- Example: consider the equation
  - $3^k \equiv_{17} 13$ for k
  - One solution is $k = 4$,
    - but it is not the only solution,
    - any number of the form $k = 4 + 16n$ is one:
- Since $3^{16} \equiv_{17} 1$
  - (by Fermat's little theorem) then
    - $3^{4+16n} = 3^4 * 3^{16n} = 3^4 * (3^{16})^n \equiv_{17} 3^4$
- And it is true that
- $3^k \equiv_{17} 13 \Leftrightarrow k \equiv_{16} 4$

Discrete log problem (DLog)

- In general, let G be any group, and $g, b \in G$
  - Then any $k \in \mathbb{N}$ that solves $g^k = b$
    - is a *discrete logarithm* (or simply, *logarithm*) of b
    - to the base g: $k = \log_g b$
- Depending on b and g
  - it is possible that no discrete logarithm exists
    - or that more than one discrete logarithm exists
- Let $\langle g \rangle$ be the finite cyclic subgroup of G
  - generated by g
- Then $\log_g b$ exists for all $b \in \langle g \rangle$

Discrete log problem (DLog)

- But no efficient algorithm
  - for computing general discrete logarithms $\log_b g$ is known
    - for an arbitrary group
- There exist groups for which
  - computing discrete logarithms is apparently difficult
- In the case of
  - large prime order subgroups of the group
    - $\mathbb{Z}_p^*$ there is not only no known efficient algorithm known
    - for the worst case,
    - but the average-case complexity
    - can be shown to be about as hard as the worst case

Integer factorization

To factor the product of two large primes

- of roughly the same length is believed to be difficult
- A related problem is the RSA problem

RSA problem (weaker than factorization)

Given n – a product of two large primes

- If one could factor n as $n = pq$, then one can calculate
  - $\phi(n) = (p-1)(q-1)$ and therefore given n (= pq), and
  - if $e \in \mathbb{Z}_n^*$ one could find $d \in \mathbb{Z}_n^*$ with
    - $e \cdot d \equiv_{\phi(n)} 1$

This is used in the RSA system (later):

- Exponentiation to the e-th power is the inverse of
- exponentiation to the d-th power

Quadratic Residuosity Assumption ("Hard Problem")

Let, as above $n = p \cdot q$ be a positive integer, product of 2 large primes

- A number a is called a "quadratic residue," or QR mod n,
  - if there exists x such that $x^2 = a \bmod n$
- Otherwise, a is called a "quadratic nonresidue" or QNR mod n

QR assumption

It is computationally hard to distinguish

- numbers that are QRs modulo n from those that are not
  - unless one knows the factorization of n

One-Way Function

- A one-way function is
  - easy to compute on every input
  - but hard to invert
    - given the image of a random input
    - (but perhaps not on all)
- "Easy" and "hard" are meant
  - in the sense of computational complexity
    - that is, "easy" means "polynomial time problem"
    - while "difficult" or "unfeasible" means not "easy"

One-Way Function

- The existence of such one-way functions is only a conjecture
  - their existence would prove
    - $P \neq NP$
  - solving the foremost problem of computer science

One-Way Function

- A function $f : \{0, 1\}^* \to \{0, 1\}^*$
  - is *one-way*
- if and only if f can be
  - computed by a polynomial time algorithm
- but any Probabilistic Polynomial Algorithm
  - that attempts to compute $\hat{f}$, a pseudo-inverse for f
    - succeeds with negligible probability

Info Trapdoor

- Trapdoor permutation (or trapdoor function)
  - is a *keyed* collection $\mathscr{F} = \{f_i | i \in I\}$
    - (We will call i the "forward key")
- In the following sense:
  - there are two "indexes/keys"
  - one is i, the (forward) key
    - required to compute the function
  - another one is a "secret" s_i, the backward key
    - required to compute the inverse efficiently

Info Trapdoor

- A collection $\mathscr{F} = \{f_i : X_i \to Y_i | i \in I\}$
  - of one-to-one functions such that
  - f_i is efficiently computable
  - For $y \in \mathscr{D}(f_i)$, given a secret s_i
    - is feasilbe to calculate a preimage x with $f(x) = y$
  - For $y \in \mathscr{D}(f_i)$
    - without information about the secret
    - it is unfeasilbe to calculate a preimage

Info Trapdoor

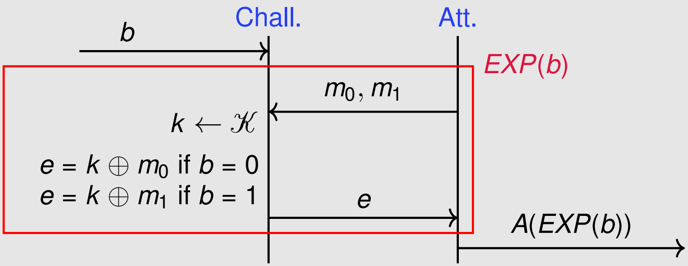
- The key (= index) for the forward direction
  - can be know to the adversary
  - and f_i may be known to him
    - not as a black box but also "as code/specification"
  - and still this will not help him
  - to invert the function
- That is, for any i, the function f_i is
  - one-way to anybody
    - whod does not know the invertion key or "trapdoor"
- Note: a slight generalization allows that for some i,
  - f_i is invertible, but his happens with a small probability

The One Time Pad

- The One Time Pad is a secure cipher
  - but only if the key (= "pad") is used only once
- $\mathscr{G} : () \to \mathscr{K}$
  - $k \leftarrow \mathscr{K} = \{0,1\}^n$
- $\mathscr{M} = \mathscr{C} = \{0,1\}^n$
- $\mathscr{E}, \mathscr{D} : \{0,1\}^n \to \{0,1\}^n$
  - $\mathscr{E}(k,x) = \mathscr{D}(k,x) := x \oplus k$

OTP is perfectly secure

Consider the usual game



The adversary wins always with prob. exactly $= \frac{1}{2}$

- there are exactly two keys consitent with his observations:
  - $k_0 = m_0 \oplus e$ and $k_1 = m_1 \oplus e$
  - but both keys have the same probability

RSA problem (weaker than factorization)

Given n – a product of two large primes – and $e \in \mathbb{Z}_n^*$

find $d \in \mathbb{Z}_n^*$ with $e \cdot d \equiv_{\phi(n)} 1$

RSA Cryptosystem ("textbook version") is a triple:

1. $\mathscr{G}()$: Generates a public and a private key: $(e = P_A, d = p_A)$
   - choose integers e, d s.t. $e \cdot d \equiv_{\phi(n)} 1$
     - e and d are the public and private keys
   - Notice that you can do that if
     - you first choose random primes p, q of $\approx$ 1024 bits
     - and let $n = pq$,
2. $\mathscr{E}(P_A, \cdot) : \mathscr{M} \rightarrow \mathscr{C}$
   - $\mathscr{E}(P_A, m) = \mathscr{E}(e, m) = m^e$ in $\mathbb{Z}_n$
3. $\mathscr{D}(p_A, \cdot) : \mathscr{C} \rightarrow \mathscr{M}$
   - $\mathscr{D}(p_A, c) = \mathscr{D}(d, c) = c^d$ in $\mathbb{Z}_n$
   - it inverts $\mathscr{E}(\mathrm{P_{A,}.})$:
     - $\mathscr{D}(d, \mathscr{E}(e, m)) = (x^e)^d = x^{ed} = x^{k \cdot \phi(n)+1} = (x^{\phi(n)})^k \cdot x = x$ in $\mathbb{Z}_n$

"Textbook RSA", a simplified version of RSA

- Beware:
  - There are many attacks against "Textbook RSA"
- Let $n = pq$ be the product of two primes
  - n is a public number, known to all parties, but
  - $\phi(n) = (p-1)(q-1) = pq - p - q + 1$ is a secret number
    - only known to the CA
- Note that, given $n = pq$, the product of two primes
  - n it is very difficult to calculate
    - $\phi(n) = (p-1)(q-1) = pq - p - q + 1$
    - if the factorization of n is not known
- For any user A, the CA chooses a "public key"
  - $\text{pk}_A = e \in \mathbb{Z}^*_{pq}$, that is $\gcd(e, \phi(n)) = 1$
- and calculates the "private key" $sk_A = d$
  - with $d \cdot e \equiv_{\phi(n)} 1$
- Encryption of $m \in \mathbb{Z}^*_{pq}$ is defined by $c = \mathcal{E}(m) \equiv_n m^e$
- Decryption of $c \in \mathbb{Z}^*_{pq}$ is defined by $m = \mathcal{D}(c) \equiv_n c^d$

"Textbook RSA" Algorithms: Key generation

- The encryption key e is known to all
  - whereas the decryption key d is
    - the private key of the receiver
    - known only to him
- p and q are fairly large in size
  - say 512 or 1024 bits
- Basic operations needed:
  - A fast primality testing algorithm, to choose the primes
  - multiplication
  - gcd computation
  - modular inverse computation

D-H Algorithm

- Since the communication uses a public channel
  - $X = g^x$ and $Y = g^y$ are visible to all
- If one can efficiently compute
  - x from g and g^x or
  - y from g and g^y
    - one can also get the private key g^{xy}
- Computing z from g and g^z in $\mathbb{Z}^*_{q-1}$
  - is the discrete logarithm problem

D-H Algorithm

- Like for integer factoring
  - the currently best algorithm
  - for computing discrete logarithm
    - has subexponential but superpolynomial time complexity
- It is not known
  - if breaking the Diffie-Hellman protocol
    - is equivalent to computing discrete logarithm

From D-H to El Gamal

Let us now transform D-H into an encryption system

Instead of the first message in the D-H exchange

$$A \xrightarrow{g^a} B$$

$$A \xleftarrow{g^b} B \qquad k = g^{ab} = (g^a)^b = (g^b)^a$$

- Let us view g^a as the public key (of A) and
  - assume that B already knows it
- B wants to encrypt a message m with that public key
- instead of sending g^b
  - What he sends is

$$\mathcal{E}(g^a, m) := (g^b, (g^a)^b \oplus m)$$

Hard Problem: Decisional Diffie-Hellman (DDH)

An adversary should not be able to compute the key g^{xy} given g^x, g^y

- But one wants more:
  - Indistinguishability of the shared key from a uniformly random one

For DH, that means the following:

Given a group G and a generator g

- Consider the following game:
- Choose randomly x, y, r and present two options to the adversary:
  - (g^x, g^y, g^{xy}) – the DH triple – or
  - (g^x, g^y, r)
    - x, y not given
- DDH problem: given the 2 triples in random order, decide
  - Which of the two options is a DH-triple
  - and which has a random third coordinate

The adversary should not be able to distinguish them

- with a probability $> 0.5 + negl$

Key-Agreement: Security against passive attacker

- The property we want is that the adversary
  - can't win the following game with a probability $> 0.5 + negl$:
- The two honest parties
  - this can be generalized to any number of parties
- run the protocol
  - using some security parameter
  - n (= length of shared key to be agreed upon)
    - resulting in a transcript *trans* and a (shared) key k

Key-Agreement: Security against passive attacker

- The challenger presents the adversary
  - the transcript *trans* and
  - $k' \in \mathcal{K} = \{0,1\}^n$, chosen like this: either
    - $k' = k$, or
    - $k' \leftarrow \{0,1\}^n$
    - with prob 0.5 for each case
  - The adversary guesses which case the challenger chose

Public Key Encryption System

- PK Encryption Sys is a triple: $(\mathscr{G}, \mathscr{E}, \mathscr{D})$
  - 1. $\mathscr{G}()$: randomized alg. that outputs a key pair (P_A, p_A)
  - 2. $\mathscr{E}(P_A, m)$: randomized alg. that takes $m \in M$ and outputs $c \in C$
  - 3. $\mathscr{D}(p_A, c)$: deterministic alg. that takes a private key (p_A) and a cyphertext $c \in C$
    - and outputs a message $m \in M$ or $\perp$
- With the following consistency condition:
  - $\forall_{(P_A, p_A) \in \text{dom}(\mathscr{G})} \forall_{m \in M} \mathscr{D}(p_A, \mathscr{E}(P_A, m)) = m$

Security of Public Key Encryption Sys

- $(\mathscr{G}, \mathscr{E}, \mathscr{D})$ is semantically secure
  - under CCA (chosen ciphertext attack)
    - iff A, the Adversary, can only win the following game with a *negligible* probability

Game

- Setup: $(P_A, p_A) \leftarrow \mathscr{G}()$
- CCA-Phase: A chooses any (polynomial) number of
  - ciphertexts c_i and receives $\mathscr{D}(c_i)$
- Challenge: A chooses messages m_0, m_1
  - The challenger chooses $m_? \leftarrow \{m_0, m_1\}$ (not known to A)
  - and sends $c_? = \mathscr{E}(P_A, m_?)$ to A
- Guess: A guesses if $c_?$ corresponds to m_0 or m_1
  - A wins if he chooses correctly

Info

Subgroups

- $H \subseteq G$ is a *subgroup* of G
  - written as $H \leq G$

$\Leftrightarrow$

- H is itself a group with respect to the operation of G

Info Lagrange's Theorem: $H \leq G \Rightarrow |H|$ divides $|G|$

- Proof: Let G be a group
  - H be a subgroup of G
- For each $x \in G$ consider

$$xH := \{x \circ h \mid h \in H\}$$

Claim 1: the sets xH are all of the size

Claim 2: the sets xH form a partition of G

Claims $\Rightarrow$ size of H divides size of G

Claim 1: the sets xH are all of the size

For any x, |xH| = |H|:

The function from H to xH

- $h \in H \mapsto x \circ h \in xH$

is a bijection

- it is 1-1
  - $x \circ h_1 = x \circ h_2 \Rightarrow h_1 = h_2$
    - cancelling x, i.e multyplying to the left with x^{-1}
- and onto
  - because $xH := \{xh \mid h \in H\}$

Claim 2: the sets xH form a partition of G

$x \in xH$ (since $e \in H$), it remains to show

For $x, y \in G$, $xH \neq yH \Rightarrow xH \cap yH = \emptyset$

If $xH \cap yH \neq \varnothing$ then

- there are $h_1, h_2 \in H$ such that
  - $x \circ h_1 = y \circ h_2$
  - and thus for any $h \in H$ it follows
    - $x \circ h = y \circ h_2 \circ h_1^{-1} \circ h \in yH$

Thus $xH \subseteq yH$ and

- by symmetry $xH = yH$

Info Exercise on Lagrange's Theorem

- Let G be a group
  - H be a subgroup of G
  - $x \in G$ and $xH := \{x \cdot h \mid h \in H\}$ as before
- For every $x, y \in G$ let
  - $x \sim y :\Leftrightarrow xH = yH$
  - $x \sim y \Leftrightarrow x^{-1}y \in H$
- $\sim$ is an equivalence relation and the equivalence classes are precisely the sets xH
  - Exercice: In the particular case of $G = (\mathbb{Z}, +)$ and $H = n\mathbb{Z}$ the subgroup of multiples of n
  - calculate $\sim$ and $G/\sim$

Info

Fermat's Theorem, Euler's Theorem

Defs (recall): Order, generator

Assume G is a finite group,

- $\langle g \rangle := \{g^i : i \in \mathbb{Z}\} = \{1, g, g^2, g^3, \ldots, g^{\text{order}(g)-1}\}$
- $|\langle g \rangle| = \text{order}(g) := \min_i\{g^i = 1\}$

$g \in G$ is called a *generator* of G if

- $\langle g \rangle = G$ or equivalently,
- the order of g is $|G|$

Fermat's Theorem, Euler's Theorem

Euler's Theorem

The order of any $g \in G$ divides $|G|$

- This follows directly from Lagrange's Theorem
  - since the size of the subgroup $\langle g \rangle$
    - divides the size of the group

Fermat's Theorem

For every prime p and $g \in \mathbb{N}$,

- $g^{p-1} = 1 \pmod{p}$
  - This follows directly from Euler's Theorem
  - Exercise: Fill in the details!!

Info Application: generating random primes

- Suppose we want to generate a large random prime p of length 1024 bits (i.e. $p \approx 2^{1024}$)
- Choose a random integer $p \in [2^{1024}, 2^{1025} - 1]$
- Test if $2^{p-1} = 1$ in $\mathbb{Z}_p$
  - If yes, done
  - If not, try another p
- This is a simple algorithm, but not the best

$$\Pr[p \text{ passes the test but is not prime}] < 2^{-60}$$

Info Choosing a Group

- For some cryptographic applications
  - we need prime-order groups
    - Because some problems, like dlog, are easier
    - if the order of the group has small prime factors
- To find a prime-order subgroup of some $\mathbb{Z}_p^*$, where p prime:
- First find primes p, q and a number t s.th. $p = tq + 1$
  - Take the subgroup of t^{th} powers, i.e.,
    - $G = (\mathbb{Z}_p^*)^t := \{x^t \mid x \in Z_p^*\}$
- This is a group because $x^t \cdot y^t = (x \cdot y)^t$
  - It has order $(p-1)/t = q$
  - Since q is prime, the group is cyclic
- In particular, $p = 2q + 1$
  - p is called a "safe prime" and
    - $(\mathbb{Z}_p^*)^2$ is the group of quadratic residues

Info Determining a generator: Primitive root modulo n

- Definition $\mathrm{ord}_{\mathbb{Z}_n^*}(a)$ is called the multiplicative order of
  - a modulo n

g is a *primitive root* modulo n

$$\begin{aligned} &\Leftrightarrow \mathrm{ord}_{\mathbb{Z}_n^*}(g) = \phi(n) \\ &\Leftrightarrow \mathrm{ord}_{\mathbb{Z}_n^*}(g) = |\mathbb{Z}_n^*| \\ &\Leftrightarrow \mathrm{ord}_{\mathbb{Z}_n^*}(g) = \min\{k \mid g^{k-1} = 1\} \end{aligned}$$

- has to be the smallest power of a which is congruent to 1 modulo n

Info Example:

- Consider the multiplicative group of $Z_p = \{1, 2, \ldots, p-1\}$ under multiplication
- Say for $p = 11$, we have $G = \{1, 2, \ldots, 10\}$, and not all elements are generators, e.g. 11 is not
- But 2 is a generator of Z_{11}:
  - $2^1 = 2, 2^2 = 4, 2^3 = 8, 2^4 = 16 = 5, 2^5 = 10 = -1,$
  - $2^6 = -2 = 9, 2^7 = -4 = 7, 2^8 = -8 = 3, 2^9 = 6, 2^{10} = 12 = 1$

Info Algorithm: Finding a Generator for $\mathbb{Z}_p^*$

- If we choose $p = 2q + 1$, where q is also prime (p is called a "safe prime") then $g \neq \pm 1$ is a generator of Z_p^* iff
- $g^{(p-1)/2} \equiv_p -1$
- This is easy to see: the order of $g \in \mathbb{Z}_p^*$ must divide the order of $\mathbb{Z}_p^*$, which is $(p-1) = 2 \cdot q$, but if $g^{(p-1)/2} = g^q \equiv_p -1$ and
- $g^2 \not\equiv_p 1$ (because $g \neq \pm 1$), then the order of g must be $(p-1)$
- There are $\phi(\phi(n)) = \phi(2q) = q - 1$ many primitive elements, picking a few random numbers and testing them will give a generator

Info Algorithm: Finding a Generator for $\mathbb{Z}_p^*$

More generally,

- given a prime p, along with the prime factorization
- $p - 1 = \Pi_{i=1}^{r} p_i^{k_i}$

The following non-deterministic algorithm outputs a generator for $\mathbb{Z}_p^*$

- for $i \leftarrow 1$ to r do
  - loop
    - choose $\alpha \leftarrow \mathbb{Z}_p^*$
  - until $\alpha^{(p-1)/p_i} \neq 1$
  - $\gamma_i \leftarrow \alpha^{(p-1)/p_i^{k_i}}$
- output $\gamma \leftarrow \Pi_{i=1}^{r} \gamma_i$