cyclic groups are commutative — defined to be the identity element

Definition. A group $\mathbb{G}$ is cyclic if there exists a generator g such that $\mathbb{G} = \{g^0, g^1, \ldots, g^{|\mathbb{G}|-1}\}$.

Definition. For an element $g \in \mathbb{G}$, we write $\langle g \rangle = \{g^0, g^1, \ldots, g^{|\mathbb{G}|-1}\}$ to denote the set generated by g (which need not be the entire set. The cardinality of $\langle g \rangle$ is the order of g (i.e., the size of the "subgroup" generated by g)

↳ means that $g^{\text{ord}(g)} = 1$

Example. Consider $\mathbb{Z}_7^* = \{1,2,3,4,5,6\}$. In this case,

$\langle 2 \rangle = \{1, 2, 4\}$ [2 is not a generator of $\mathbb{Z}_7^*$] $\text{ord}(2) = 3$

$\langle 3 \rangle = \{1, 3, 2, 6, 4, 5\}$ [3 is a generator of $\mathbb{Z}_7^*$] $\text{ord}(3) = 6$

Lagrange's Theorem. For a group $\mathbb{G}$, and any element $g \in \mathbb{G}$, $\text{ord}(g) \mid |\mathbb{G}|$ (the order of g is a divisor of $|\mathbb{G}|$).

↳ For $\mathbb{Z}_p^*$, this means that $\text{ord}(g) \mid p-1$ for all $g \in \mathbb{G}$

Corollary (Fermat's Theorem): For all $x \in \mathbb{Z}_p^*$, $x^{p-1} = 1 \pmod p$

Proof. $|\mathbb{Z}_p^*| = |\{1, 2, \ldots, p-1\}| = p-1$

By Lagrange's Theorem, $\text{ord}(x) \mid p-1$ so we can write $p-1 = k \cdot \text{ord}(x)$ (for integer k) and so $x^{p-1} = (x^{\text{ord}(x)})^k = 1^k = 1 \pmod p$

Implication: Suppose $x \in \mathbb{Z}_p^*$ and we want to compute $x^y \in \mathbb{Z}_p^*$ for some large integer $y \gg p$

↳ We can compute this as

$$x^y = x^{y \pmod{p-1}} \pmod p$$

since $x^{p-1} = 1 \pmod p$

↳ Specifically, the exponents operate modulo the order of the group

↳ Equivalently: group $\langle g \rangle$ generated by g is isomorphic to the group $(\mathbb{Z}_q, +)$ where $q = \text{ord}(g)$

$$\langle g \rangle \cong (\mathbb{Z}_q, +)$$
$$g^x \mapsto x$$

Notation: g^x denotes $\overbrace{g \cdot g \cdots g}^{x \text{ times}}$

g^{-x} denotes $(g^x)^{-1}$ [inverse of group element g^x]

$g^{x^{-1}}$ denotes $g^{(x^{-1})}$ where x^{-1} computed mod $\text{ord}(g)$ — need to make sure this inverse exists!

Computing on group elements: In cryptography, the groups we typically work with will be large (e.g., 2^{256} or 2^{1024})

- Size of group element (# bits): $\sim \log |\mathbb{G}|$ bits (256 bits / 2048 bits)
- Group operations in $\mathbb{Z}_p^*$: $\log p$ bits per group element

addition of mod p elements: $O(\log p)$

multiplication of mod p values: naively $O(\log^2 p)$

Karatsuba $O(\log^{1.71} p)$

Schönhage-Strassen (GMP library): $O(\log p \log\log p \log\log\log p)$

best algorithm $O(\log p \log\log p)$ [2019]

↳ not yet practical ($> 2^{4096}$ bits to be faster...)

exponentiation: using repeated squaring: $g, g^2, g^4, g^8, \ldots, g^{\lceil \log p \rceil}$, can implement using $O(\log p)$ multiplications [$O(\log^3 p)$ with naive multiplication]

↳ time/space trade-offs with more precomputed values

division (inversion): typically $O(\log^2 p)$ using Euclidean algorithm (can be improved)

Computational problems: in the following, let $\mathbb{G}$ be a finite cyclic group generated by g with order q

- **Discrete log problem:** sample $x \xleftarrow{R} \mathbb{Z}_q$

  given $h = g^x$, compute x

- **Computational Diffie-Hellman (CDH):** sample $x, y \xleftarrow{R} \mathbb{Z}_q$

  given g^x, g^y, compute g^{xy}

- **Decisional Diffie-Hellman (DDH):** sample $x, y, r \xleftarrow{R} \mathbb{Z}_q$

  distinguish between (g, g^x, g^y, g^{xy}) vs. (g, g^x, g^y, g^r)

Each of these problems translates to a corresponding computational assumption:

Definition. Let $\mathbb{G} = \langle g \rangle$ be a finite cyclic group of order q (where q is a function of the security parameter λ) e.g., $q = 2^\lambda$

The DDH assumption holds in $\mathbb{G}$ if for all efficient adversaries A:

$$\left| \Pr[x, y \xleftarrow{R} \mathbb{Z}_p : A(g, g^x, g^y, g^{xy}) = 1] - \Pr[x, y, r \xleftarrow{R} \mathbb{Z}_p : A(g, g^x, g^y, g^r) = 1] \right| = \mathrm{negl}(\lambda)$$

The CDH assumption holds in $\mathbb{G}$ if for all efficient adversaries A:

$$\Pr[x, y \xleftarrow{R} \mathbb{Z}_q : A(g, g^x, g^y) = g^{xy}] = \mathrm{negl}(\lambda)$$

The discrete log assumption holds in $\mathbb{G}$ if for all efficient adversaries A:

$$\Pr[x \xleftarrow{R} \mathbb{Z}_q : A(g, g^x) = x] = \mathrm{negl}(\lambda)$$

Certainly: if DDH holds in $\mathbb{G}$ $\Rightarrow$ CDH holds in $\mathbb{G}$ $\Rightarrow$ discrete log holds in $\mathbb{G}$

there are groups where CDH believed to be hard, but DDH is easy

??

Major open problem: does this hold?
Can we find a group where discrete log is hard but CDH is easy?

Diffie-Hellman key exchange

- Let $\mathbb{G}$ be a group of prime order p (and generator g) – choice of group, generator, and order fixed by standard

Alice		**Bob**
$x \xleftarrow{R} \mathbb{Z}_p$		$y \xleftarrow{R} \mathbb{Z}_p$
	$\xrightarrow{\quad g^x \quad}$	
	$\xleftarrow{\quad g^y \quad}$	
compute $g^{xy} = (g^y)^x$		compute $g^{xy} = (g^x)^y$

shared secret: g^{xy}

But usually, we want a random bit-string as the key, not random group element

↳ Element g^{xy} has $\log p$ bits of entropy, so should be able to obtain a random bit-string with $\ell < \log p$ bits

↳ Solution is to use a "randomness extractor"

  ↳ Information-theoretic constructions based on universal hashing / pairwise-independent hashing (loses some bits of entropy)

↳ Use a "random oracle" or an "ideal hash function" [Heuristic : SHA-256 (g, g^x, g^y, g^{xy})] [binds the key to the entire transcript]
(very efficient in practice) good practice: hash all inputs

↳ Arguing security : 1. Rely on HashDH assumption $(g, g^x, g^y, H(g, g^x, g^y, g^{xy})) \overset{c}{\approx} (g, g^x, g^y, r)$
where $H: \mathbb{G}^4 \to \{0,1\}^n$ and $r \xleftarrow{R} \{0,1\}^n$

2. Model H as ideal hash function $H: \mathbb{G}^4 \to \{0,1\}^n$ (i.e., random oracle) and rely on CDH in $\mathbb{G}$ [inability to evaluate H on g^{xy} $\Rightarrow$ output is random string]

Instantiations: Discrete log in $\mathbb{Z}_p^*$ when p is 2048-bits provides approximately 128-bits of security

↳ Best attack is General Number Field Sieve (GNFS) — runs in time $2^{\tilde{O}(\sqrt[3]{\log p})}$ time

Much better than brute force — $2^{\log p}$

cube root in exponent not ideal! if we want to double security, need to increase modulus by 8x! (e.g., 16384-bit modulus for 256 bits of security)

group operations all scale linearly (or worse) in bitlength of the modulus

↳ Need to choose p carefully
(e.g., avoid cases where $p-1$ is smooth) having small prime factors

for DDH applications, we usually set $p = 2q+1$ where q is also a prime (p is a "safe prime") and work in the subgroup of order q in $\mathbb{Z}_p^*$ ($\mathbb{Z}_p^*$ has order $p-1 = 2q$)

Elliptic curve groups: only require 256-bit modulus for 128 bits of security

↳ Best attack is generic attack and runs in time $2^{\log p / 2}$ [ρ-algorithm — can discuss at end of semester]

↳ Much faster than using $\mathbb{Z}_p^*$: several standards
- NIST P256, P384, P512
- Dan Bernstein's curves: Curve 25519

can discuss more at end of semester (or in advanced crypto class)

↳ Widely used for key-exchange + signatures on the web

When describing cryptographic constructions, we will work with an abstract group (easier to work with, less details to worry about)