What are the elements in $\mathbb{Z}_p$?

Bezout's identity: For all positive integers $x, y \in \mathbb{Z}$, there exists integers $a, b \in \mathbb{Z}$ such that $ax + by = \gcd(x, y)$.

Corollary: For prime $p$, $\mathbb{Z}_p = \{[1, 2, \ldots, p-1]\}$.

Proof: Take any $x \in \{1, 2, \ldots, p-1\}$. By Bezout's identity, $\gcd(x, p) = 1$ so there exists integers $a, b \in \mathbb{Z}$ where $1 = ax + bp$.

Modulo $p$, this is $ax = 1 \pmod{p}$ so $a = x^{-1} \pmod{p}$.

Coefficients $a, b$ in Bezout's identity can be efficiently computed using the extended Euclidean algorithm:

**Euclidean algorithm:** algorithm for computing $\gcd(a, b)$ for positive integers $a > b$:
- relies on fact that $\gcd(a, b) = \gcd(b, a \pmod{b})$:
  - to see this: take any $a > b$
    - we can write $a = b \cdot q + r$ where $q \equiv 1$ is the quotient and $0 \leq r < b$ is the remainder
  - $d$ divides $a$ and $b$ $\iff$ $d$ divides $b$ and $r$
  - $\gcd(a, b) = \gcd(b, r) = \gcd(b, a \pmod{b})$
- gives an explicit algorithm for computing $\gcd$: repeatedly divide:
  - $\gcd(60, 27): \quad 60 = 27(2) + 6 \quad [q = 2, \ r = 6] \quad \Rightarrow \quad \gcd(60, 27) = \gcd(17, 6)$
  - $27 = 6(4) + 3 \quad [q = 4, \ r = 3] \quad \Rightarrow \quad \gcd(27, 6) = \gcd(6, 3)$
  - $6 = 3(2) + 0 \quad [q = 2, \ r = 0] \quad \Rightarrow \quad \gcd(6, 3) = \gcd(3, 0) = 3$

"rewind" to recover coefficients in Bezout's identity:

**Extended Euclidean algorithm**

\[
\begin{aligned}
&\text{extended} \quad \begin{cases} 
60 = 27(2) + 6 \\
27 = 6(4) + 3 \\
6 = 3(2) + 0
\end{cases} 
\Rightarrow 
3 = 27 - 6 \cdot 4 \\
\end{aligned}
\]

From this, we can compute $x^{-1} \pmod{p}$ using the extended Euclidean algorithm.

Therefore needed: $O(\log p)$ — i.e., bitlength of the input [worst case inputs: Fibonacci numbers]

Implication: Euclidean algorithm can be used to compute modular inverses (faster algorithms also exist)
**Definition.** A group \( G \) is cyclic if there exists a generator \( g \) such that \( G = \{g^0, g^1, \ldots, g^{\text{ord}(g)}\} \).

**Definition.** For an element \( g \in G \), we write \( \langle g \rangle = \{g^0, g^1, \ldots, g^{\text{ord}(g)}\} \) 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 \( g \)).

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

\[
\langle 2 \rangle = \{1, 2, 4, 3, 6, 5\} \quad [2 \text{ is not a generator of } \mathbb{Z}_7^*] \quad \text{ord}(2) = 3
\]

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

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

\( \text{for } G \neq \{0\} \text{, this means } \forall g \in G \text{, } \text{ord}(g) \mid \text{ord}(G) \).

**Corollary (Fermat's Theorem).** For all \( x \in \mathbb{Z}_p^* \), \( x^p = 1 \) (mod \( p \)).

**Proof.** By Lagrange's Theorem, \( \text{ord}(x) \mid \text{ord}(\mathbb{Z}_p^*) = p-1 \) so we can write \( p-1 = k \cdot \text{ord}(x) \) and so \( x^{p-1} = (x^{\text{ord}(x)})^k = 1^k = 1 \) (mod \( p \)).

**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 \( x^y \equiv x^y \cdot (x^{p-1})^{x^y \\pmod{p}} \) (mod \( p \))

- Since \( x^{p-1} = 1 \) (mod \( p \))

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

- Equivalently: group \( \langle g \rangle \) generated by \( g \) is isomorphic to the group \( (\mathbb{Z}_p^*, \cdot) \) where \( g = \text{ord}(g) \)

\[
\langle g \rangle \cong (\mathbb{Z}_p^*, \cdot) \quad g \mapsto x
\]

**Notation:** \( g^x \) denotes \( g \cdot g \cdot \ldots \cdot g \)

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

\( g^x \) denotes \( g^{-x} \) where \( x \) 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): \( \sim \log 16b \) bits \( (256 \text{ bits} / 2048 \text{ bits}) \)

- 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: namely \( O(\log^2 p) \)

- Karatsuba: \( O(\log^3 p) \)

- 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 \( (> 2^{1400} \text{ bits}) \) to be faster...**

- Exponentiation: using repeated squaring: \( g, g^2, g^4, g^8, \ldots, g^{16b \cdot p^2} \), can implement using \( O(\log p) \) multiplications \( [O(\log^2 p) \text{ with naive multiplication}] \)

- Time/space trade-offs with more precomputed values

- Division (inversion): typically \( O(\log^2 p) \) using Euclidean algorithm (can be improved)
Computational problems: Let \( G \) be a finite cyclic group generated by \( g \) with order \( q \).

- **Discrete log problem:** sample \( x \in \mathbb{Z}_q \)
  
  given \( h = g^x \), compute \( x \)

- **Computational Diffie-Hellman (CDH):** sample \( x, y \in \mathbb{Z}_q \)
  
  given \( g^x, g^y \), compute \( g^{xy} \)

- **Decisional Diffie-Hellman (DDH):** sample \( x, y, r \in \mathbb{Z}_q \)
  
  distinguish between \( (g^x, g^y, g^r) \) vs. \( (g^x, g^y, g^s) \)

Each of these problems translates to a corresponding computational assumption:

**Definition.** Let \( G = \langle g \rangle \) be a finite cyclic group of order \( q \) (where \( q \) is a function of the security parameter \( \lambda \)).

The DDH assumption holds in \( G \) if for all efficient adversaries \( A \):

\[
\Pr[(x, y, r, s) \in \mathbb{Z}_q^4 : A(g^x, g^y, g^r, g^s) = 1] = \text{negl}(\lambda)
\]

The CDH assumption holds in \( G \) if for all efficient adversaries \( A \):

\[
\Pr[(x, y) \in \mathbb{Z}_q^2 : A(g^x, g^y) = 1] = \text{negl}(\lambda)
\]

The discrete log assumption holds in \( G \) if for all efficient adversaries \( A \):

\[
\Pr[x \in \mathbb{Z}_q : A(g^x) = x] = \text{negl}(\lambda)
\]

**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 \( 2^{63} \) time
  
  Much better than brute force — \( 2^{2048} \)

- Need to choose \( p \) carefully (e.g., avoid cases where \( p-1 \) is smooth)

  For DDH applications, we usually set \( p = 2^k + 1 \) where \( k \) 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 \( p-1 = 2q \)) — see NIST bitlength of the modulus

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

  - Best attack is generic attack and runs in time \( 2^{128} \) \([p\text{-algorithm} - \text{can discuss at end of semester}\]
  
  - Much faster than using \( \mathbb{Z}_p^* \): several standards
    - NIST P256, P384, P512
    - Dan Bernstein’s curves: Curve25519

  - 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)