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

- <u>Discrete log problem</u>: sample $x \xleftarrow{R} \mathbb{Z}_q$
  given $h = g^x$, compute x
- <u>Computational Diffie-Hellman (CDH)</u>: sample $x, y \xleftarrow{R} \mathbb{Z}_q$
  given g^x, g^y, compute g^{xy}
- <u>Decisional Diffie-Hellman (DDH)</u>: 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:

<u>Definition</u>. 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 $\mathcal{A}$:

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

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

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

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

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

<u>Certainly</u>: 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?

<u>Instantiations</u>: 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 chose 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 <u>generic</u> 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)

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 — $x \xleftarrow{R} \mathbb{Z}_p$

Bob — $y \xleftarrow{R} \mathbb{Z}_p$

Alice $\xrightarrow{g^x}$ Bob

Alice $\xleftarrow{g^y}$ Bob

Alice: compute $g^{xy} = (g^y)^x$

Bob: 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})] (very efficient in practice)

good practice to hash all components

[binds the key to the entire transcript]

↳ 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]

Public-key encryption: Encryption scheme where encryption is public (does not require shared secrets)

- $\text{Setup}(1^\lambda) \to (pk, sk)$ [generates a public/private key-pair — also called KeyGen]
- $\text{Encrypt}(pk, m) \to c$
- $\text{Decrypt}(sk, c) \to m$

Everyone can publish a public key (in a directory)

↳ Can encrypt to anyone without exchanging keys (recipient can be offline)

Correctness: $\forall m \in \mathcal{M}: \Pr[(pk, sk) \leftarrow \text{Setup}(1^\lambda) : \text{Decrypt}(sk, \text{Encrypt}(pk, m)) = m] = 1$

Security: semantic security from secret-key setting, but adversary also gets public key

adversary

challenger — $b \in \{0,1\}$; $(pk, sk) \leftarrow \text{Setup}(1^\lambda)$

adversary $\xleftarrow{pk}$ challenger

adversary $\xrightarrow{m_0, m_1 \in \mathcal{M}}$ challenger

adversary $\xleftarrow{c_b \leftarrow \text{Encrypt}(pk, m_b)}$ challenger

adversary: $b' \in \{0,1\}$

$$\text{SSAdv}[\mathcal{A}, \Pi_{PKE}] = \left|\Pr[\mathcal{A} \text{ outputs } 1 \mid b = 0] - \Pr[\mathcal{A} \text{ outputs } 1 \mid b = 1]\right|$$