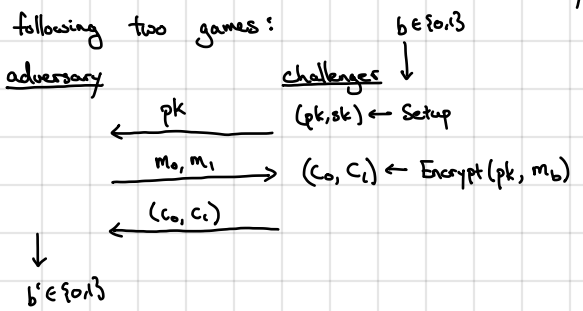
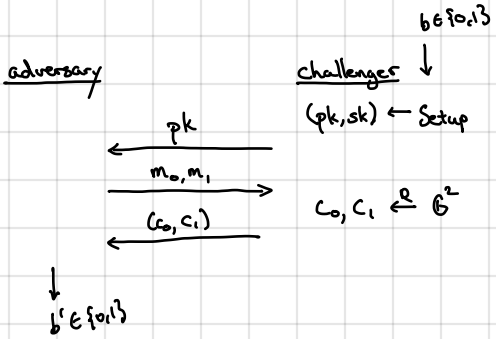
Security: If DDH holds in $\mathbb{G}$, then ElGamal is semantically secure.

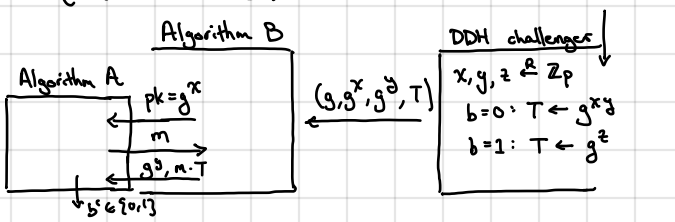
Proof. Consider following two games:





Claim: these two games are indistinguishable under DDH

Proof. Suppose there exists efficient A that can distinguish $(c_0, c_1) \leftarrow \text{Encrypt}(pk, m)$ from $(c_0, c_1) \xleftarrow{R} \mathbb{G}^2$. We use A to break DDH:



adversary's advantage in guessing b is 0 here since (c_0, c_1) is independent of (m_0, m_1)!

Observe: x is uniform over $\mathbb{Z}_p$ so g^x is a properly-generated public key (for ElGamal)

if $T = g^{xy}$, then $(g^y, T \cdot m) = (g^y, g^{xy} \cdot m)$ which is the output of Encrypt(pk, m) with randomness y — this is exactly the distribution where A sees Encrypt(pk, m)

if $T = g^z$, then $(g^y, g^z \cdot m)$ is uniform over $\mathbb{G}^2$ (since y, z are sampled independently of each other and of m) — this is exactly the distribution where A sees $(c_0, c_1) \xleftarrow{R} \mathbb{G}^2$

distinguishing advantage of B = distinguishing advantage of A

Equivalent view: Under DDH, g^{xy} looks uniform even given g, g^x, g^y, so an ElGamal ciphertext looks indistinguishable (to an efficient adversary) from a OTP encryption

What if we want to encrypt longer messages? [or messages that is not a group element]

- Hybrid encryption (key encapsulation [KEM]):
  - Use PKE scheme to encrypt a secret key
  - Encrypt payload using secret key + authenticated encryption

called key encapsulation

PKE.Encrypt(pk, k)	"header"	[slow]
AE.Encrypt(k, m)	"payload"	[fast]

secret-key operations much much faster than public-key operations!

- How to derive key from group element?

  Same as in key-exchange: hash the group element to a bit-string (symmetric key)

  e.g., Hash-ElGamal: Encrypt(pk, m): $y \xleftarrow{R} \mathbb{Z}_p$

  $c = (g^y, m \oplus H(g, h, g^y, h^y))$

  ↑ as before, can also rely on CDH + ideal hash function (random oracle)

  ↑ $H: \mathbb{G}^4 \to \{0,1\}^n$

Vanilla ElGamal described above is not CCA-secure!

Ciphertexts are malleable: given $ct = (g^y, h^y \cdot m)$, can construct ciphertext $(g^y, h^y \cdot m \cdot g)$ which decrypts to message $m \cdot g$

↳ directly implies a CCA attack

Several approaches to get CCA security from DH assumptions:

- Cramer-Shoup (CCA-security from DDH) - based on hash-proof systems
- Fujisaki-Okamoto transformation (using an ideal hash function + CDH)
- Make stronger assumption ("interactive" CDH + use ideal hash function):
  - Setup: $x \xleftarrow{R} \mathbb{Z}_p$ $\quad pk: h$
    $h \leftarrow g^x$ $\quad sk: x$
  - Encrypt(pk, m): $y \xleftarrow{R} \mathbb{Z}_p$ $\quad k \leftarrow H(g, g^x, g^y, h^y)$ $\quad ct' \leftarrow Enc_{AE}(k, m)$
    $c \leftarrow (g^y, ct')$
  - Decrypt(sk, c): $k \leftarrow H(g, g^x, c_0, c_0^x)$
    $m \leftarrow Dec_{AE}(k, c_1)$

Essentially ElGamal where key derived from hash function

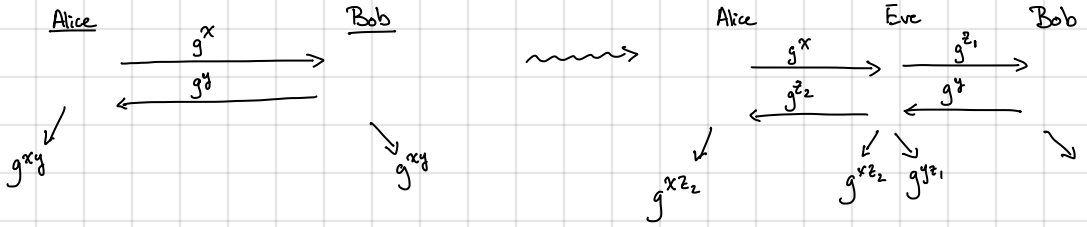
We do not know of any groups where CDH believed to be hard, but interactive CDH is easy.

↑ also called strong DH assumption

↑ "CDH is hard even given access to a DDH oracle"

↓ symmetric authenticated encryption scheme

Diffie-Hellman key-exchange is an anonymous key-exchange protocol: neither side knows who they are talking to
↳ vulnerable to a "man-in-the-middle" attack



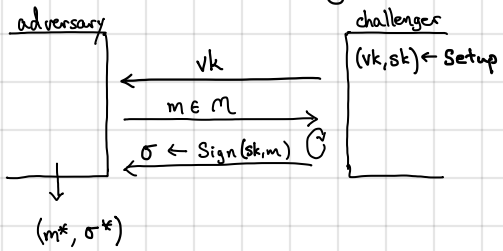
What we require: authenticated key-exchange (not anonymous) and relies on a root of trust (e.g., a certificate authority)
↳ On the web, one of the parties will authenticate themself by presenting a certificate

To build authenticated key-exchange, we require more ingredients — namely, an integrity mechanism [e.g., a way to bind a message to a sender — a "public-key MAC" or digital signature]

We will revisit when discussing the TLS protocol

Digital signature scheme: Consists of three algorithms:
- Setup $\rightarrow (vk, sk)$: Outputs a verification key vk and a signing key sk
- Sign $(sk, m) \rightarrow \sigma$: Takes the signing key sk and a message m and outputs a signature σ
- Verify $(vk, m, \sigma) \rightarrow 0/1$: Takes the verification key vk, a message m, and a signature σ, and outputs a bit $0/1$

Two requirements:
- Correctness: For all messages $m \in \mathcal{M}$, $(vk, sk) \leftarrow$ Setup, then
$$\Pr[\text{Verify}(vk, m, \text{Sign}(sk, m)) = 1] = 1.$$
[Honestly-generated signatures always verify]
- Unforgeability: Very similar to MAC security. For all efficient adversaries $\mathcal{A}$, $\text{SigAdv}[\mathcal{A}] = \Pr[W = 1] = \text{negl}(\lambda)$, where W is the output of the following experiment:



Let $m_1, \ldots, m_Q$ be the signing queries the adversary submits to the challenger. Then, $W = 1$ if and only if:
$$\text{Verify}(vk, m^*, \sigma^*) = 1 \text{ and } m^* \notin \{m_1, \ldots, m_Q\}$$
Adversary cannot produce a valid signature on a new message.

Exact analog of a MAC (slightly weaker unforgeability: require adversary to not be able to forge signature on new message)
↳ MAC security required that no forgery is possible on any message [needed for authenticated encryption]

digital signature algorithm — elliptic-curve DSA } standards (widely used on the web — e.g., TLS)

It is possible to build digital signatures from discrete log based assumptions (DSA, ECDSA)
↳ But construction not intuitive until we see zero knowledge proofs
↳ We will first construct from RSA (trapdoor permutations)