Midterm II Review Session

CS 171

March 15, 2024

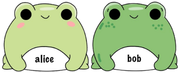


Table of Contents

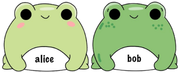
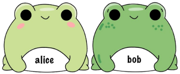


Table of Contents

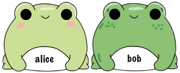


MAC: The Concept

So far in the class, we've precisely defined confidentiality for end-to-end encrypted messaging with *symmetric-key encryption*.

But how can we guarantee the **integrity** of a ciphertext?

A Message Authentication Codes (MAC) is a keyed checksum, which is sent along with the message. It takes in a fixed-length secret key and an arbitrary-length message, and outputs a fixed-length checksum. A secure MAC has the property that any change to the message will render the checksum invalid.



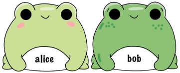
MAC: Definition

A MAC scheme consists of 3 PPT algorithms (Gen, MAC, Verify):

- $Gen(1^n)$: Outputs a key k.
- $MAC_k(m)$: Outputs a tag t.
- $Verify_k(m, t)$: Outputs $0/1$.

These satisfy 2 properties:

1. **Correctness:** $\forall n, k \leftarrow Gen(1^n), \forall m \in \{0,1\}^*$, we have that $Verify_k(m, MAC_k(m)) = 1$.
2. **Security:** $Verify_k(m, t)$ outputs 1 if and only if $MAC_k(m) = t$.

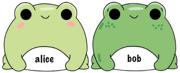


MAC: Security Game

The adversary's goal is to **forge** a MAC. The adversary wins only if they output a valid tag on a message that was never previously queried.

The game is between a challenger C and the adversary $\mathcal{A}$.
$\text{MACForge}_{\mathcal{A},\Pi}(1^n)$:

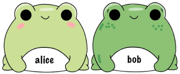
1. C samples $k \leftarrow Gen(1^n)$.
2. $\mathcal{A}$ makes MAC queries to the challenger. Let M be the list of queries $\mathcal{A}$ makes.
3. Finally, $\mathcal{A}$ outputs (m^*, t^*).
4. C outputs 1 if $Verify(m^*, t^*) = 1 \wedge m* \notin M$ and 0 otherwise.



MAC: Security Definition

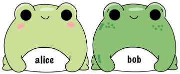
$\Pi = (Gen, MAC, Verify)$ is existentially unforgeable under the adaptive chosen attack if $\forall$ PPT $\mathcal{A}$ it holds that:

$$\Pr[\text{MACForge}_{\mathcal{A},\Pi} = 1] \leq \text{negl}(n)$$



MAC: Tips

1. If you are asked to construct a new MAC' and prove its security:
   - Use the system from the proof workshop where your secure underlying building block is the MAC.
   - Assume there is an adversary $\mathcal{A}$ that breaks MAC'.
   - Construct an external adversary $\mathcal{B}$ that simulates the MACForge game for $\mathcal{A}$ and uses this to break MAC. Contradiction!
   - **Hint:** $\mathcal{B}$ can *tinker* with the what it gets from $\mathcal{A}$ and what it forwards from its oracle to $\mathcal{A}$.
2. There can be interesting **variations** of unforgeability such as strong unforgeability from Discussion 6, Q2: Adversary can win even if they output a valid tag on a message that was previously queried.
3. You can be asked to **compare** the security properties of the MAC security definition with a new primitive.
   - E.g. define a primitive x that is not a MAC.

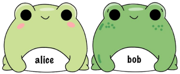


MAC: Practice Problem (Part (a))

Spring 2021 MT2 Q2

Consider a "CCA-style" extension to the definition of secure message authentication codes, where the adversary is provided with both a *MAC* and a *Verify* oracle. Our starting point will be the "standard" notion of MAC security, called "existential unforgeability under adaptive chosen-message attacks," and we will consider a variant of this definition that allows for Verify oracle queries.

(a) Provide a formal definition of CCA-secure MACs. That is, describe an experiment called $\mathsf{CCA-Mac-Forge}_{\mathcal{A},\Pi}(n)$, and provide a security requirement stating that no adversary can win your game except with negligible probability.



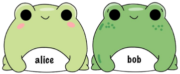
MAC: Practice Problem (Part (a) Solution)

(a) Provide a formal definition of CCA-secure MACs. That is, describe an experiment called $\mathsf{CCA-Mac-Forge}_{\mathcal{A},\Pi}(n)$, and provide a security requirement stating that no adversary can win your game except with negligible probability.

1. The challenger samples $k \leftarrow \mathsf{Gen}(1^n)$.
2. The adversary $\mathcal{A}$ is given input 1^n and oracle access to $\mathsf{Mac}_k(\cdot)$ and $\mathsf{Verify}_k(\cdot,\cdot)$. The adversary eventually outputs a pair (m,t). Let Q denote the set of all queries that $\mathcal{A}$ asked to its $\mathsf{Mac}_k(\cdot)$ oracle.
3. The output of the experiment is defined to be 1 if and only if (1) $\mathsf{Verify}_k(m,t) = 1$ and (2) $m \notin Q$.

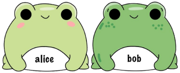
Π is a CCA-secure MAC if for all adversaries $\mathcal{A}$,

$$\Pr[\mathsf{CCA-Mac-Forge}_{\mathcal{A},\Pi}(n) = 1] = \mathsf{negl}(n).$$



MAC: Practice Problem (Part (b))

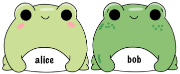
(b) Assume that Π is a standard secure *deterministic* MAC that has *canonical verification*, meaning that i) the Mac algorithm is deterministic and ii) the Verify algorithm, on input (m, t), recomputes $t' := \mathsf{Mac}_k(m)$ and accepts if $t' = t$. Prove that Π also satisfies your definition from part (a).



MAC: Practice Problem (Part (b) Solution)

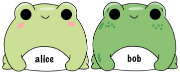
When Π is deterministic and has canonical verification, each message has only a single valid tag. Thus, if the scheme is secure, then access to a Verify oracle does not help (and so Π is secure in the sense of the definition given in part (a)). To see this, note that for any query (m, t) to the Verify oracle there are 3 possibilities:

1. m was previously queried to the Mac oracle, and response t was received. Here the adversary already knows that $\text{Verify}_k(m, t) = 1$.
2. m was previously queried to the Mac oracle, and response $t' \neq t$ was received. Since Π is deterministic, the adversary already knows $\text{Verify}_k(m, t) = 0$.
3. m was not previously queried to the Mac oracle. By security of Π, we can argue that $\text{Verify}_k(m, t) = 0$ with all but negligible probability because otherwise, m, t is a valid forgery. Let's prove it.

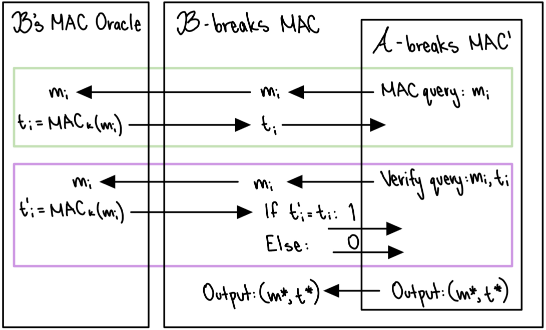


MAC: Practice Problem (Part (b) Solution Continued)

We want to show that if m was not previously queried to the Mac oracle, by security of Π, we can argue that $\text{Verify}_k(m, t) = 0$ with all but negligible probability because otherwise, m, t is a valid forgery.
Let MAC' be a CCA-secure MAC. Assume that $\text{Verify}_k(m, t) = 1$. Then there exists an adversary $\mathcal{A}$ that can query a message m to the verify oracle in the CCA-secure MAC scheme to obtain a valid MAC.
Now construct an adversary $\mathcal{B}$ that simulates the security game for $\mathcal{A}$ to win the Π security game.



MAC: Practice Problem (Part (b) Solution Continued)



We successfully simulate the game for $\mathcal{A}$ because its queries are accurately answered. So $\mathcal{A}$ can produce a message that was not previously queried such that $\text{Verify}_k(m, t) = 0$, then so can $\mathcal{B}$. Contradiction.

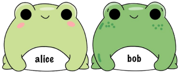
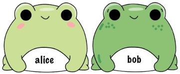


Table of Contents

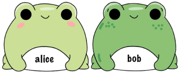


CRHF: Basic Definitions

- Syntax:

$$H^s(x) = y$$

- A **collision** in H^s is a pair (x, x') such that $x \neq x'$ but $H^s(x) = H^s(x')$.
- H is guaranteed to have collisions. We require that $|y| < |x|$ (H is **compressing**).
- If it's hard to find those collisions, then the hash function is **collision-resistant**.

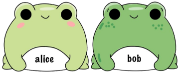


CRHF: Formal Syntax

- The hash function $\mathcal{H}$ is a pair of algorithms: $\mathcal{H} = (\mathrm{Gen}, H)$.
- Gen: outputs a random key/seed s:

$$s \leftarrow \mathrm{Gen}(1^n)$$

  The key is allowed to be public.
- H^s: This is also sometimes referred to as the hash function.
  The output length – and sometimes the input length – are fixed.
  H^s is deterministic.

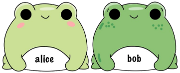


CRHF: Security Game

- **Summary:** The adversary is given s and a description of H, and they try to find a collision in H^s with non-negligible probability.
- Hash-coll$_{\mathcal{A},\mathcal{H}}(n)$:
  1. The challenger samples a key $s \leftarrow \mathrm{Gen}(1^n)$ and gives s to the adversary $\mathcal{A}$.
  2. $\mathcal{A}$ produces two inputs (x, x') to H^s.
  3. $\mathcal{A}$ wins (and the game outputs 1) if (x, x') are a collision:

$$x \neq x' \text{ and } H^s(x) = H^s(x')$$

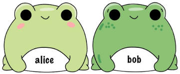
  Otherwise, $\mathcal{A}$ loses (the game outputs 0).
- Note that the adversary can compute H^s by themselves.



CRHF: Security Definition

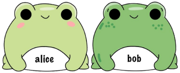
- $\mathcal{H}$ is **collision-resistant** if for any PPT adversary $\mathcal{A}$, there is a negligible function negl such that:

$$\Pr[\text{Hash-coll}_{\mathcal{A},\mathcal{H}}(n) = 1] \leq \text{negl}(n)$$



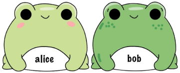
CRHF: Tips

- The adversary in the CRHF security game is given s and a description of H, so they can compute $H^s(x)$ on any input x of their choosing.



CRHF: Practice Problem

- Summary: The problem shows you how to reprogram a hash function so that a given x^* maps to a given y^*, while maintaining collision-resistance.
- Source: Midterm 2, Fall 2019, Q 5.2.b



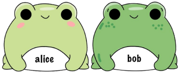
CRHF: Practice Problem

The problem:

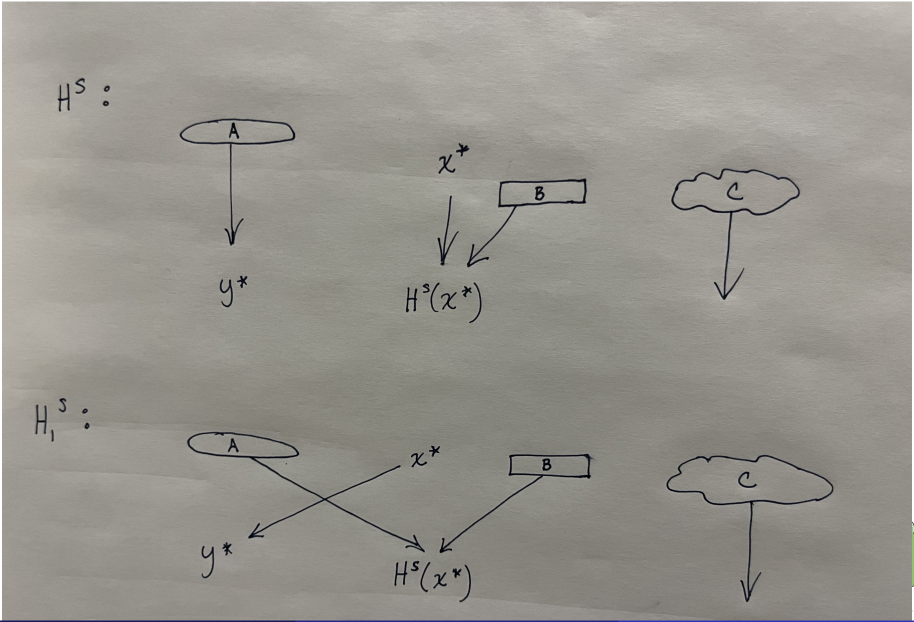
- Let $\mathcal{H} = (\text{Gen}, H)$ be a CRHF. Let x^* belong to the domain of H^s, and let y^* belong to the range of H^s.
- Next, for any $s \leftarrow \text{Gen}(1^n)$:

$$\text{let } H_1^s(x) = \begin{cases} y^* & \text{if } x = x^* \\ H^s(x^*) & \text{if } x \neq x^* \text{ and } H^s(x) = y^* \\ H^s(x) & \text{otherwise} \end{cases}$$

- Prove that (Gen, H_1) is a CRHF.



CRHF: Practice Problem



CRHF: Practice Problem Solution

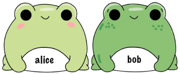
Theorem

(Gen, H_1) *is a CRHF.*

Proof:

<u>Overview:</u>

- Assume toward contradiction that (Gen, H_1) is not a CRHF. Then there exists an adversary $\mathcal{A}$ that wins the CRHF game for H_1 (by finding a collision in H_1) with non-negligible probability.
- We will use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that wins the CRHF game for H with non-negligible probability.
- This is a contradiction because (Gen, H) is a CRHF. So our initial assumption was false and (Gen, H_1) is also a CRHF.



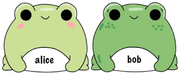
CRHF: Practice Problem Solution

Construction of $\mathcal{B}$:

1. In the CRHF game for H, the challenger samples $s \leftarrow \text{Gen}(1^n)$ and gives s to the adversary $\mathcal{B}$.
2. $\mathcal{B}$ will run $\mathcal{A}$ on input s until $\mathcal{A}$ produces two inputs (x, x').
3. $\mathcal{B}$ makes a list of collision candidates:

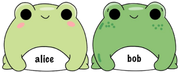
$$C := \{(x, x'), (x, x^*), (x', x^*)\}$$

   and checks whether each candidate $(x_1, x_2) \in C$ satisfies the conditions: $x_1 \neq x_2$ and $H^s(x_1) = H^s(x_2)$.
4. $\mathcal{B}$ outputs the first candidate $(x_1, x_2) \in C$ that satisfies the conditions.

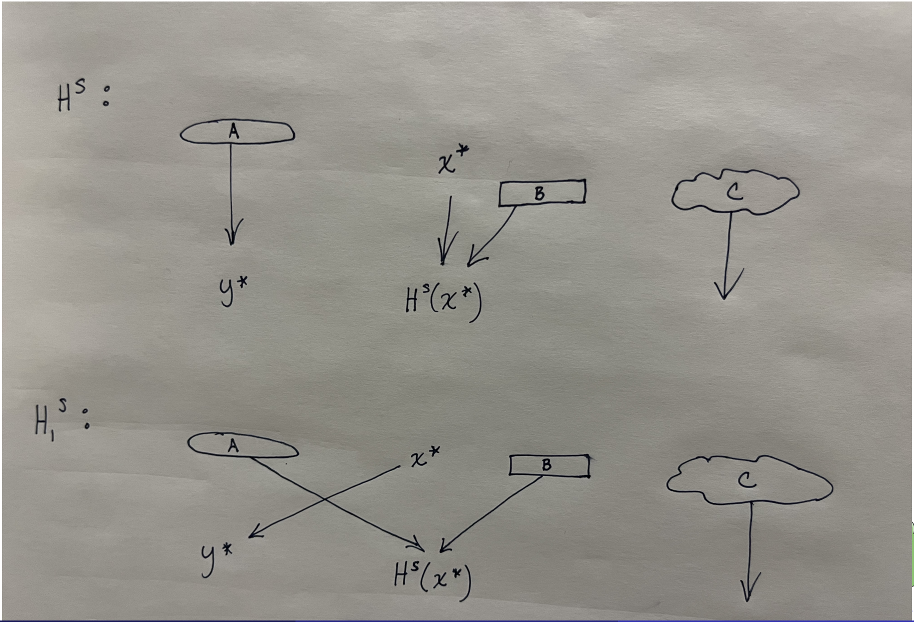


CRHF: Practice Problem Solution

- Note that with non-negligible probability (x, x') will be a collision in H_1^s:
$$x \neq x' \text{ and } H_1^s(x) = H_1^s(x')$$
- We will prove that in this case, $\mathcal{B}$ will succeed in finding a collision in H^s.



CRHF: Practice Problem Solution



CRHF: Practice Problem Solution

Let's assume that (x, x') are a collision in H_1^s. Then consider the following cases:

- Case 1: $(x, x') \in A$. Then

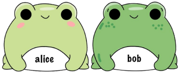
$$H^s(x) = y^* = H^s(x')$$

  so (x, x') are a collision in H^s.

- Case 2: $(x, x') \in B \cup C$. Then

$$H^s(x) = H_1^s(x) = H_1^s(x') = H^s(x')$$

  so (x, x') are a collision in H^s.



CRHF: Practice Problem Solution

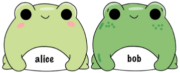
- Case 3: $x \in A, x' \in B$. Then

$$H^s(x') = H^s(x^*)$$

  so (x', x^*) are a collision in H^s.
- Case 4: $x \in B, x' \in A$. Then

$$H^s(x) = H^s(x^*)$$

  so (x, x^*) are a collision in H^s.



CRHF: Practice Problem Solution

- We also need to consider the possibility that $H^s(x^*) = y^*$. In this case, reprogramming the hash function doesn't do anything, so $H^s = H_1^s$. Then (x, x') are a collision in H^s.

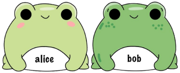
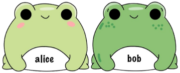


Table of Contents

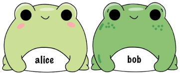


OWF: Definition

- Syntax:

$$f(x) = y$$

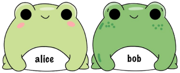
- A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is *one-way* if
- It's easy to compute, i.e., computing $f(x)$ runs in "probabilistic polynomial time.", but
- It's hard to invert, i.e., there is no "probabilistic polynomial time" algorithm that can compute $f^{-1}(y)$.
- Note: $\{0,1\}^* \rightarrow \{0,1\}^*$ means the input and output can be arbitrarily long bit strings.



OWF: Security

- How can we formally define "hard to invert"?
- OWF-Sec$_{\mathcal{A},f}(n)$:
  1. The challenger randomly samples an input $x \leftarrow \{0,1\}^n$ and gives $f(x)$ to the adversary $\mathcal{A}$ along with 1^n.
  2. $\mathcal{A}$ produces a value $x' \in \{0,1\}^n$.
  3. $\mathcal{A}$ wins (and the game outputs 1) if $f(x') = f(x)$
  4. Otherwise, $\mathcal{A}$ loses (the game outputs 0).
- The probability $\mathcal{A}$ wins the above game should be at most $\mathsf{negl}(n)$ for f to be secure.
- This can be expressed equivalently as:

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] \leq \mathsf{negl}(n).$$

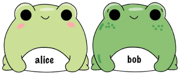


OWF: Tips

- OWF's are "almost universal" in the sense that most cryptographic primitives imply the existence of OWFs.
- If a question asks you to construct a OWF from a standard-looking primitive, you probably do it.
- The only gotcha is if the given primitive is contrived, e.g. constructing a OWF f from a PRP F as follows:

$$f(x_0 \parallel x_1) = F(x_0, x_1)$$

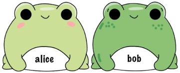
- See discussion 8 for detail on why this example fails.



OWF: Example Questions

Example questions: construct a one-way function from one of the following primitives:

- A PRG $G : \{0,1\}^{n/2} \rightarrow \{0,1\}^n$
- a CRHF (Gen, H) where $H^s : \{0,1\}^n \rightarrow \{0,1\}^{n/2}$
- a one-to-one function (permutation) $F : \{0,1\}^n \rightarrow \{0,1\}^n$ with a hard-concentrate predicate $hc(\cdot)$.



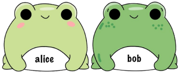
OWF: Practice Problem

- Question: construct a one-way function from a CRHF (Gen, H) such that $H^s : \{0,1\}^n \to \{0,1\}^{n/2}$.

We'll prove the following theorem:

Theorem

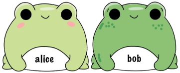
$f(s \parallel x) = s \parallel H^s(x)$ is a OWF.



Proving $f(s \parallel x) = s \parallel H^s(x)$ is a OWF

Step 1: Stating our argument.

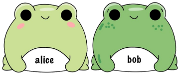
- Suppose for the sake of contradiction that f is not a OWF.
- This implies that there exists an adversary $\mathcal{A}$ that can win the OWF-Sec$_{\mathcal{A},f}(n)$ security game with nonnegl(n) probability.
- We will construct an adversary $\mathcal{B}$ from $\mathcal{A}$ that wins Hash-coll$_{\mathcal{A},H}(n)$ with nonnegl(n) probability.



Proving $f(s \parallel x) = s \parallel H^s(x)$ is a OWF

Step 2: Construction of $\mathcal{B}$:

1. $\mathcal{B}$ is given the truly random seed s from the CRHF challenger.
2. $\mathcal{B}$ samples a random $x \leftarrow \{0,1\}^n$ and runs $\mathcal{A}$ on $H^s(x)$ to obtain x'.
3. If $x = x'$, abort.
4. Otherwise, output (x, x') as a collision.

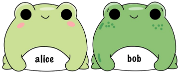


Proving $f(s \parallel x) = s \parallel H^s(x)$ is a OWF

Step 3: Analysing $\mathcal{B}$:

1. We need to lower bound the probability that we don't abort (i.e., the probability we win).
2. First, observe that the probability our random x collides with x' by chance ($H^s(x) = H^s(x')$) is upper bounded by the birthday bound, $2^{-n/2}$. Note: we have no control over the particular x' that $\mathcal{A}$ got from inverting $f(x)$, but the x that $\mathcal{B}$ sampled itself *is* uniformly random, meaning that chance $x = x'$ is still random even if $\mathcal{A}$ doesn't choose x' randomly.
3. Conditioned on the above *not* happening, the probability that $x \neq x'$ is at least $1/2$. This follows from the fact that H takes n bits to $n/2$ bits, implying $\Pr[x = x'] = \frac{1}{2^{n/2}} < \frac{1}{2}$.
4. Putting these two point together:

$$Pr[x \neq x' | H^s(x) = H^s(x')] \geq \frac{1}{2} - \frac{1}{2^{n/2}}.$$



Proving $f(s \parallel x) = s \parallel H^s(x)$ is a OWF

Step 4: Wrapping up:

1. We proved that we don't abort probability $\frac{1}{2} - \frac{1}{2^{n/2}}$.
2. In the case that $\mathcal{B}$ doesn't abort, it follows from the construction that (x, x') are a valid collision.
3. Thus,

$$\begin{aligned}
&\Pr[\text{Hash-coll}_{\mathcal{B},H}(n) = 1] \\
&= \Pr[\text{OWF-Sec}_{\mathcal{A},f}(n) = 1] \cdot Pr[x \neq x' | H^s(x) = H^s(x')] \\
&= \text{nonnegl}(n) \cdot (\frac{1}{2} - \frac{1}{2^{-n/2}}) \\
&= \text{nonnegl}'(n).
\end{aligned}$$

4. That is, given an adversary $\mathcal{A}$ that wins $\text{OWF-Sec}_{\mathcal{A},f}(n)$ with non-negligible probability, $\mathcal{B}$ wins $\text{Hash-coll}_{\mathcal{B},H}(n)$ with non-negligible probability—a contradiction $\square$.

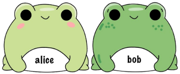
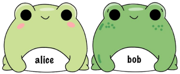


Table of Contents



Public Key Encryption: Definition

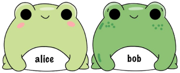
(The syntax and most properties are very similar to private/symmetric key encryption that we've seen earlier.)

A PKE scheme consists of three PPT algorithms (Gen, Enc, Dec) where

- $Gen(1^n) \to (\mathbf{sk}, \mathbf{pk})$
- $Enc(\mathbf{pk}, m) \to c$
- $Dec(\mathbf{sk}, c) \to m/\perp$

and these satisfy two properties

- **Correctness**: $Dec(sk, Enc(pk, m)) = m$.
- **Security**: EAV = CPA security / CCA security



PKE: CPA Security

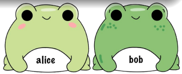
- Challenger samples $(sk, pk) \leftarrow Gen(1^n)$ and gives pk to $\mathcal{A}$.
- $\mathcal{A}$ outputs two messages m_0, m_1.
- Challenger samples a bit $b \in \{0, 1\}$ and outputs $Enc(pk, m_b)$.
- $\mathcal{A}$ outputs b' as a guess for b.

CPA-secure if for all PPT $\mathcal{A}$

$$Pr[b' = b] \leq \frac{1}{2} + negl(n)$$

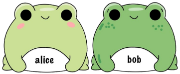
Intuition

Looking at the ciphertext should not reveal which message was encrypted.



Tips

- *pk* is given to the adversary, so no encryption oracle is needed – $\mathcal{A}$ can locally encrypt whatever it wants.
- *sk* is unknown, so decryption is not possible – CCA game for PKE gives access to a decryption oracle to $\mathcal{A}$.
- Most proof techniques are similar to that of private key encryption schemes:
  - Show that a certain scheme is not CPA/CCA secure – construct an adversary for the game that is able to figure out which message was encrypted.
  - Show that a certain scheme is secure – often relies on the security of some other primitive $\rightarrow$ *Proof by contradiction*.



PKE Example: El Gamal Encryption

PKE scheme based on DDH.

- $Gen(1^n)$: Generate cyclic group $\mathbb{G}$ of order q and a generator g.
  Sample $x \in \mathbb{Z}_q$ and $h = g^x$.
  Output $pk = (\mathbb{G}, q, g, h)$, $sk = x$
- $Enc(pk, m) \to (c_1, c_2)$: Sample $r \in \mathbb{Z}_q$.
  Output $(c_1, c_2) = (g^r, m \cdot h^r)$
- $Dec(sk, (c_1, c_2)) \to m$: **Output** $m = \frac{c_2}{c_1^x}$

Correctness

$$Dec(sk, Enc(pk, m)) = Dec(sk, (g^r, mh^r)) = \frac{mh^r}{(g^r)^x} = \frac{mh^r}{h^r} = m$$

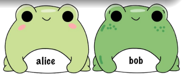
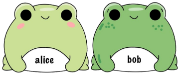


Table of Contents

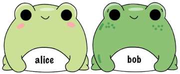


Key Exchange

Consists of three randomized algorithms (P_1, P_2, P_3):

1. Alice computes $(m_1, st) \leftarrow P_1(1^n)$ and sends m_1 to Bob.
2. Bob computes $(m_2, k) \leftarrow P_2(m_1)$. Then he sends m_2 to Alice and outputs k.
3. Alice computes $k \leftarrow P_3(st, m_2)$ and outputs k.

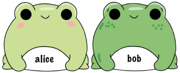
- **Correctness**: Both parties get the same key k.
- **Security**: No eavesdropper can distinguish between (m_1, m_2, k) and (m_1, m_2, r) where r is a random element.



Problem: Key exchange from CPA-Secure PKE

Question

Given a PKE scheme (Gen, Enc, Dec), construct a secure key exchange scheme (P_1, P_2, P_3).



Problem: Key exchange from CPA-Secure PKE

Question

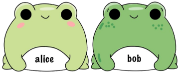
Given a PKE scheme (Gen, Enc, Dec), construct a secure key exchange scheme (P_1, P_2, P_3).

Construction

$P_1(1^n)$: Run $Gen(1^n) \to (sk, pk)$. Return $(m_1, st) = (pk, sk)$.

$P_2(m_1)$: Sample random r and run $Enc(m_1, r) \to c$.
Return $(m_2, k) = (c, r)$.

$P_3(m_2, st)$: Run $Dec(st, m_2) \to r'$ and return r.



Solution: Key exchange from CPA-Secure PKE

By contradiction: Suppose the Key exchange scheme is not secure. Then we have $\mathcal{A}$ that can distinguish (m_1, m_2, k) from (m_1, m_2, r) where r is random.

We'll construct $\mathcal{B}$ for the CPA game that distinguishes between encryptions of m_0 or m_1.