How to Proof

CS 171

February 29, 2024

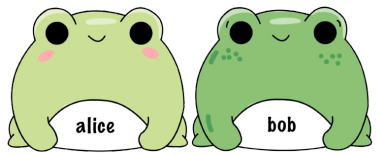


Table of Contents

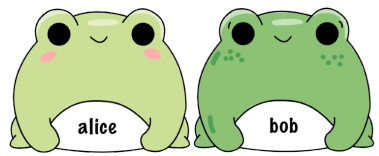
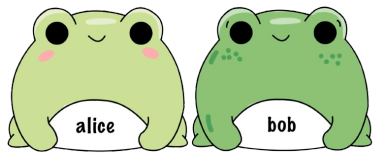


Table of Contents



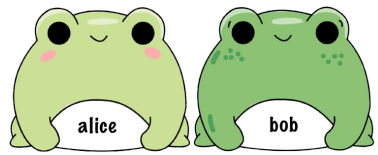
So, you want to learn how to write a security proof...

First of all...

This is a challenging thing to do... and the hardest skill to pick up from this class. So, it's okay if it takes some time to get the hang of!

There is no plug-and-chug solution to writing a correct security proof. And often, there are many correct proof strategies.

The point of this workshop is to share some tips, walk through one security proof that uses these tips, and answer your questions.



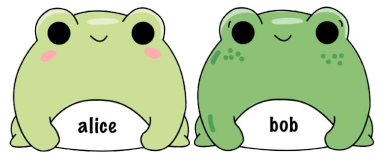
Tip #1: Identify Your Secure Building Blocks

Ask yourself...

Does this scheme x that I am trying to prove secure contain an underlying building block y that I **know** to be secure?

Examples: This scheme uses an underlying...

- pseudorandom function $F_k(\cdot)$!
- message authentication code $MAC_k(\cdot)$!
- CPA-secure encryption scheme (Gen, Enc, Dec)!

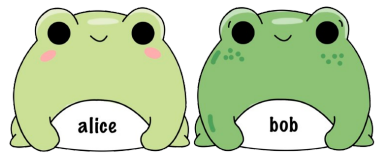


Tip #2: Create an "Outside" Adversary $\mathcal{B}$

So you found an underlying secure building block y... We **know** it's secure, so we should take advantage of this fact.

Assume that there is an adversary $\mathcal{A}$ that can break x. Your goal should be to construct an adversary $\mathcal{B}$ that can run $\mathcal{A}$ internally and **use it** to break y.

But this is a contradiction, because we assumed y was secure. So x is secure.

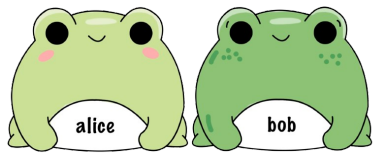


Tip #3: $\mathcal{B}$ Can Simulate $\mathcal{A}$'s Security Game

So we know we need to make an adversary $\mathcal{B}$ to break our already secure scheme y using $\mathcal{A}$, which breaks x. How? How can we actually use $\mathcal{A}$ to help $\mathcal{B}$ do what it needs to do?

$\mathcal{B}$ can **simulate the security game** for $\mathcal{A}$. This means it can effectively **serve as an intermediary** between $\mathcal{A}$ and its own oracle:

- $\mathcal{B}$ can pretend to **be** $\mathcal{A}$'s oracle.
- $\mathcal{B}$ can tinker with $\mathcal{A}$'s queries before they reach $\mathcal{B}$'s oracle.
- $\mathcal{B}$ can tinker with $\mathcal{B}$'s oracle responses to those queries before they reach $\mathcal{A}$.
- $\mathcal{B}$ can try to learn from $\mathcal{A}$'s final output.

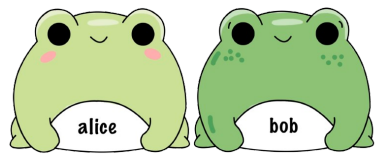


Tip #4: Get Specific with $\mathcal{B}$'s Tinkering

$\mathcal{B}$ will have quite a bit of power now that it can (1) simulate what $\mathcal{A}$ sees and (2) see what $\mathcal{A}$ outputs.

So once you have the intuition about why the scheme seems secure, use that knowledge **specific** changes to the queries that $\mathcal{B}$ will make as the "middleman" between $\mathcal{A}$ and its own oracle.

This goes hand in hand with making **specific** changes to the output of $\mathcal{A}$'s security game to get the output that $\mathcal{B}$ will need to win its own security game.



Tip #5: Quantify the Adversary's Advantage

Say you have successfully set up your security game. Now you need to show your adversary wins the game with **non-negligible probability**.

"Winning" depends on the problem, but it usually comes down to the adversary being able to detect a difference in distributions between a "uniformly random" scenario and the scheme itself. Examples:

- Your adversary will be able to output something that will allow it to win 100% of the time (probability 1).
- The random scenario happens with $\frac{1}{2}$ probability, but your adversary wins with probability $p > \frac{1}{2} + \text{non-negl}(n)$.

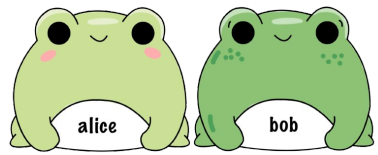
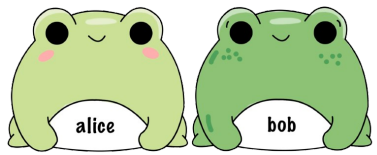


Table of Contents

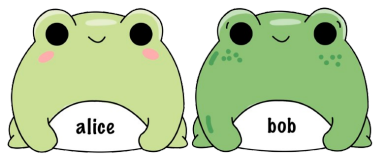


Discussion 6, Q2: Difference Between Regular & Strong Security for MACs

Construct a message authentication code $\text{MAC}' := (\text{Gen}', \text{Mac}', \text{Verify}')$ that is secure but not strongly secure. In your construction, you may start with a secure MAC, $\text{MAC} := (\text{Gen}, \text{Mac}, \text{Verify})$:

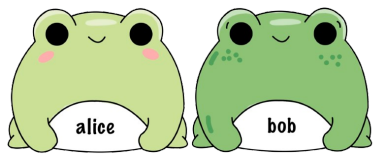
- $\text{Gen}'(1^n)$: Run $\text{Gen}(1^n)$.
- $\text{Mac}'(k, m)$:
  1. Compute $t = \text{Mac}(k, m)$.
  2. Sample $b \leftarrow \{0, 1\}$.
  3. Output $t' := t||b$.
- $\text{Verify}'(k, m, t)$: Let $t_{\text{truncated}}$ be t with the final bit removed. Run $\text{Verify}(k, m, t_{\text{truncated}})$, and output the result.

We will prove that MAC' is a secure MAC.



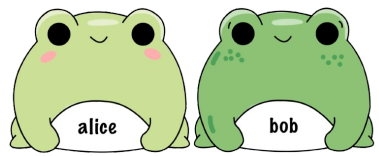
Discussion 6, Q2: Use the Tips to Get Started!

- Tip #1: What is your secure building block?
- Tip #2: What will your "outside" adversary be?



Discussion 6, Q2: Use the Tips to Get Started!

- Tip #1: What is your secure building block? **MAC**.
- Tip #2: What will your "outside" adversary be? **An adversary $\mathcal{A}$ that can break the MAC.**



Discussion 6, Q2

Let's draw it.

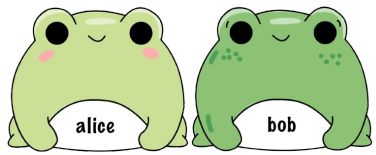
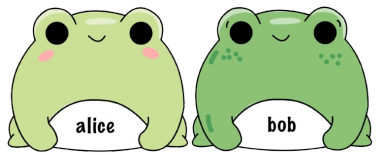


Table of Contents

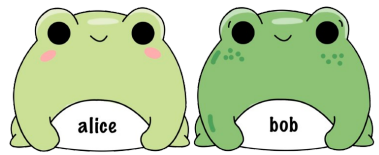


Midterm 1, Q4: CPA-Secure Encryption

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure encryption scheme. Below, we will construct another encryption scheme and prove that it is also CPA-secure. In the encryption scheme below, let the message m belong to $\{0,1\}^n$.

- $\mathsf{Gen}_1(1^n)$: Sample the key as follows: $k \leftarrow \mathsf{Gen}(1^n)$.
- $\mathsf{Enc}_1(k, m)$: Sample $r \leftarrow \{0,1\}^n$ uniformly at random. Then compute $c_0 := \mathsf{Enc}(k, r)$ and $c_1 := r \oplus m$. Output the ciphertext $c = (c_0, c_1)$.
- $\mathsf{Dec}_1(k, (c_0, c_1))$: $\mathsf{Dec}_1(k, (c_0, c_1))$: Compute $r' := \mathsf{Dec}(k, c_0)$ and then compute $m' := r' \oplus c_1$. Output m'.

Prove that $(\mathsf{Gen}_1, \mathsf{Enc}_1, \mathsf{Dec}_1)$ satisfies CPA security.



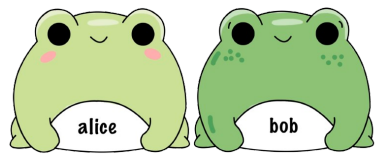
Midterm 1, Q4: An "Informal" Solution

This is the style of solution that works to get partial credit, but does not constitute a correct proof.

Sometimes, intuition checks out but there are nits that make the scheme insecure. You can also have bugs in your proof, but it is a significantly more surefire than an intuitive explanation. But let's start with one...

The encryption scheme Enc *is secure. Also, r is like a uniformly sampled one time pad, so it also doesn't ruin the security. So* Enc_1 *is secure.*

This is on the right track... but it's not a proof and is not rigorous.



Midterm 1, Q4

Let's draw it.