The basic **threat model** is a concise description of the powers of the adversary and the operations they're permitted to perform, as well as their goals. For example, if Alice is trying to send a message to Bob, we might describe the threat model through a middleman, Mallory, who can intercept and send new messages, manipulating them arbitrarily, and trying to have Bob accept an altered message.

We look at:

1. Confidentiality
2. Integrity
3. Availability

If Alice just sends the message X, it clearly won't be sufficient. Instead, she'll need to send $(X, f(X))$ (i.e., X and some additional information). In this model, we call $f(X)$, the "Message Authentication Code" (MAC).

Say Mallory sends (a, b). Bob accepts the message $i.f.f.f(a) = b$. What does this say about f?

- f must be **deterministic**.
- f must be easily computable (by Alice and Bob).
- f must *not* be computable by Mallory.
- $\implies f$ depends on knowledge that only Alice and Bob have.

What if we decide that f is a secret function? This is actually *not* a good idea. It's very difficult to figure out the likelihood that Mallory won't be able to guess what Alice and Bob are thinking–difficult to quantify the likelihood of attack here.

A better approach: rely on a **secret key** (preference for secret key but public function over private function is *Kerckhoff's Principle*).

- Say k is a 256-bit random value known only to Alice and Bob.
- Define $f(k, X)$ to be a function of the secret key k and the message X.

At this point:

1. We can quantify the probability that Mallory guesses the key correctly (in this case, $\frac{1}{2^{256}}$).
2. If we lose the key for some reason, we can just generate a fresh key (much easier than generating a new function, as we're now just generating a single variable to refresh our security protocol).
3. We can communicate with many different individuals by swapping out the key.

Defining "Secure"

We call this the "Secure MAC Game":

1. Mallory sends us x_0.
2. We send back $f(x_i)$.
3. Mallory sends us x_i, continuing a polynomial number of times.
4. Mallory guesses by sending across $(y, f(y))$, where $y \notin \{x_0, x_1, ...\}$.
5. Mallory wins if $f(y)$ is correct.

Definition (Secure MAC)**.** *f is a **secure MAC** if and only if every "efficient" (poly-time) strategy for Mallory wins with "negligible" (goes to zero as a negative exponential in the key-size) probability.*

Example: try a random function that takes an arbitrary-size input, produces 256-bit output, and defined on a random truth table.

Theorem 0.1. *A random function is a secure MAC.*

Proof. Learning any row of the truth table provides no information on any other row. Guessing a distinct row is always $\frac{1}{2^{256}}$ probability. There's no strategy that does better than guessing. □

As this function is way too large and expensive to represent and implement, we want a function that appears random, but actually isn't (i.e., a *pseudorandom* function).

Pseudorandom Functions

Definition (Pseudorandom Function)**.** *A **pseudorandom function** is a public function $f(k, x)$ where k is a secret 256-bit key.*

*f is a **secure PRF** if and only if every "efficient" strategy for Mallory wins with prob $\leq \frac{1}{2} + \epsilon$, where ϵ is "negligible".*

The goal is that Mallory cannot tell the difference between a truly random function and our pseudorandom function: play the "Secure MAC Game", but use a random function with probability $\frac{1}{2}$ and a pseudorandom function with probability $\frac{1}{2}$. Mallory should not be able to guess whether we're using our pseudorandom function or a truly random function with probability $> \frac{1}{2}$.

Caveats:

- Mallory can win by trying all values of k; but that's not "efficient".
- Mallory can get non-zero advantage over guessing by trying a poly-size subset of k values; but this advantage is "negligible" because the pool of k values is exponential.

Theorem 0.2. *If f is a secure PRF, then $f(k, x)$ with a random k is a secure MAC.*

Proof. If Mallory could win the "Secure MAC Game", then she could simply play it to win the "Secure PRF Game". □

Do PRFs Exist?

Maybe. We don't have a theoretical reason why a PRF definitely does or doesn't exist; all we know is that there are *some* functions that appear to be PRFs and have not been proven otherwise (although there are functions that once appeared to be PRFs and *were* proven otherwise). Most theorists would say "Probably".

In practice, we use a "candidate" PRF. The standard for acceptance is based on how long we've failed to prove it as a non-PRF.

HMAC-SHA256

A common candidate PRF, defined as:

$$HMAC - SHA256(k, X) = SHA256(k \oplus z_1 || SHA256(k \oplus z_2 || X))$$

where $z_1 = 0x3636...$ and $z_2 = 0x5c5c....$

Cryptographic Hash Functions

Definition. *A* ***cryptographic hash function*** *is a function from arbitrary-sized input to fixed-size output that is "hard to reverse."*

Example: SHA256. Break input into fixed-size (512-bit) blocks: $b_0, b_1, ..., b_{k-1}$. Take some 256-bit constant c (looked up in standards document), pass c and b_0 into a function h that produces a 256-bit output; apply h again with the output and b_1, etc.

SHA256 is *not* a secure MAC on its own. However, some properties that it does have:

- Collision resistance: you can't find $x \neq y$ such that $H(x) = H(y)$.
- Second pre-image resistance: given x, you can't find y such that $H(x) = H(y)$.
- If x is chosen randomly from a high-entropy distribution, then given $H(x)$, you can't find x. (You *want* to say that given $H(x)$, you can't find x; but that isn't quite true in general.)

Aside: Security Models

Two ways to discuss security:

- On the level of a story (i.e., using Alice and Bob).
- On the level of mathematics.

Even theoretical security researchers use stories quite often. Why? Easy to follow and remember. However:

- What we describe as "Alice" and "Bob" might actually be computers.
- What we describe as "Alice" is usually a person *and* a computer. Forgetting this leads us to rule out certain kinds of attacks (e.g., phishing).