Theorem. Let $F : K \times X \to X$ be a secure PRF. Let $T_{E CBC}$ be the encrypted CBC MAC formed by $F$. Then, for all $MAC$ adversaries $A$, there exists a PRF adversary $B$ where

$$MAC_{Adv}[A, T_{E CBC}] \leq 2 \cdot PRF_{Adv}[B, F] + \frac{O^2(x+1)^2}{1x}$$

Proof. See Boneh-Shoup, Chapter 6.

Implication: Block size of PRF is important.
- 3DES: $|X| = 2^{64}$; need to update key after $< 2^{52}$ signing queries
- AES: $|X| = 2^{128}$; can use key to sign many more messages ($\sim 2^{64}$ messages)

A parallelizable MAC (PMAC) — general idea:

Can use similar ideas as CMAC (randomized prefix-free encoding) to support messages that is not constant multiple of block size.

Parallel structure of PMAC makes it easily updatable (assuming $F$ is a PRP)
- Suppose we change block $i$ from $m[i]$ to $m'[i]$:
  $\implies$ can make local updates without full recomputation

In terms of performance:
- On sequential machine, PMAC comparable to ECBC, NMAC, CMAC
- On parallel machine, PMAC much better

Summary: Many techniques to build a large-domain PRF from a small-domain one (domain extension for PRF)
- Each method (ECBC, CMAC, PMAC) gives a MAC on variable-length messages
- Many of these designs (or their variants) are standardized
So far, we have focused on constructing a large-domain PRF from a small-domain PRF in order to construct a MAC on long messages.

- Alternative approach: "compress" the message itself (e.g., hash the message) and MAC the compressed representation

Still require unforgeability: two messages should not hash to the same value [otherwise trivial attack: if \( H(m_1) = H(m_2) \), then \( \text{MAC on } m_1 \) is also MAC on \( m_2 \)]

- Counter-intuitive: if hash value, a shorter than messages, collisions always exist - so we can only require that they are hard to find.

**Definition.** A hash function \( H : M \rightarrow T \) is collision-resistant if for efficient adversaries \( A \),

\[
\text{CRFAdv}[A, H] = \Pr[ (m, m') \leftarrow A : H(m) = H(m') ] = \text{negl}.
\]

As stated, definition is problematic: if \( |M| > |T| \), then there always exists a collision \( m^*, m' \) so consider the adversary that has \( m^*, m' \) hard-coded and outputs \( m^*, m' \)

- That same adversary always exists (even if we may not be able to write it down explicitly)

- Formally, we model the hash function as being parameterized by an additional parameter (e.g., a "system parameter" or a "key") so adversary cannot output a hard-coded collision

- In practice, we have a concrete function (e.g., SHA-256) that does not include security or system parameter

- Believed to be hard to find a collision even though there are infinitely many (SHA-256 can take inputs of arbitrary length)

**MAC from CRF: Suppose we have the following**

- A MAC \((\text{Sign}, \text{Verify})\) with key space \( K \), message space \( M_0 \) and tag space \( T \)

\[
\begin{align*}
M_0 & = \{ 0, 1 \}^{256} \\
M_1 & = \{ 0, 1 \}^9
\end{align*}
\]

- A collision-resistant hash function \( H : M_1 \rightarrow M_0 \)

Define \( S'(k, m) = S(k, H(m)) \) and \( V'(k, m, t) = V(k, H(m), t) \)

**Theorem.** Suppose \( \text{Trunc} = (\text{Sign}, \text{Verify}) \) is a secure MAC and \( H \) is a CRF. Then, \( \text{Trunc} \) is a secure MAC. Specifically, for every efficient adversary \( A \), there exist efficient adversaries \( B_0 \) and \( B_1 \) such that

\[
\text{MACAdv}[A, \text{Trunc}] \leq \text{MACAdv}[B_0, \text{Trunc}] + \text{CRFAdv}[B_1, H]
\]
Proof Idea. Suppose A manages to produce a valid forgery \( t \) on a message \( m \). Then, it must be the case that
- \( t \) is a valid MAC on \( H(m) \) under Timac
- If A queries the signing oracle on \( m' \neq m \) where \( H(m') = H(m) \), then A breaks collision-resistance of \( H \)
- If A never queries signing oracle on \( m' \) where \( H(m') = H(m) \), then it has never seen a MAC on \( H(m) \) under Timac. Thus, A breaks security of Timac.

[See Boneh-Shoup for formal argument - very similar to above: just introduce event for collision occurring vs. not occurring]

Constructing above is simple and elegant, but not used in practice.
- **Disadvantage 1:** Implementation requires both a secure MAC and a secure CRHF: more complex, need multiple software/hardware implementations
- **Disadvantage 2:** CRHF is a key-less object and collision finding is an offline attack (does not need to query verification oracle)
  
Adversary with substantial preprocessing power can compromise collision resistance (especially if hash size is small)

**Birthday attack on CRHFs.** Suppose we have a hash function \( H : \{0,1\}^n \rightarrow \{0,1\}^l \). How might we find a collision in \( H \) (without knowing anything more about \( H \))

**Approach 1:** Compute \( H(1), H(2), ..., H(2^l - 1) \)

\( \leftarrow \) By Pigeonhole Principle, there must be at least one collision — runs in time \( O(2^l) \)

**Approach 2:** Sample \( m, e \in \{0,1\}^n \) and compute \( H(mi) \). Repeat until collision is found.

How many samples needed to find a collision?

**Theorem (Birthday Paradox).** Take any set \( S \) where \( |S| = n \). Suppose \( r, ..., r \in \{0,1\}^n \). Then,

\[
Pr[\exists i \neq j : r_i = r_j] \geq 1 - e^{-\frac{n^2}{2n}}
\]

**Proof.**

\[
Pr[\exists i \neq j : r_i = r_j] = 1 - Pr[r_i \neq r_j \forall i \neq j] = 1 - Pr[r_1 \neq r_2 \land r_3 \neq r_4 \land ... \land r_{n-1} \neq r_n]
\]

\[
= 1 - \left( \frac{n-1}{n} \right) \left( \frac{n-2}{n} \right) \ldots \left( \frac{n-2^{n-1}}{n} \right)
\]

\[
\geq 1 - \prod_{i=1}^{n-1} \left( 1 - \frac{i}{n} \right)\quad \text{automatically holds for } x \leq -1
\]

\[
\geq 1 - \prod_{i=1}^{n-1} e^{-\frac{i}{n}}\quad \text{since } 1 + x \leq e^x \text{ for all } x \in \mathbb{R}\quad \left[ e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots \right]
\]

\[
= 1 - e^{\sum_{i=1}^{n-1} \frac{-i}{n}} = 1 - e^{-\frac{n(n-1)}{2n}} = 1 - e^{-\frac{n^2}{2n}}
\]

When \( l \geq 1.2 \sqrt{n} \), \( Pr[\text{collision}] = Pr[\exists i \neq j : r_i = r_j] > \frac{1}{2}. \) [For birthdays, \( l \geq 0.5 \sqrt{n} \approx 25 \)]

\( \leftarrow \) Birthdays not uniformly distributed, but this only increases collision probability.

[Try proving this]

For hash functions with range \( \{0,1\}^l \), we can use a birthday attack to find collisions in time \( \sqrt{2^l} = 2^{l/2} \) can even do it with constant space!

\( \leftarrow \) For 128-bit security (e.g., \( 2^{128} \)), we need the output to be 256-bits (hence SHA-256)

\( \leftarrow \) Quantum collision-finding can be done in \( \sqrt{2^{128}} \) (cube root attack), though requires more space

[via Floyd's cycle finding algorithm]