CS 6160 Cryptology Lecture 12: Constructing MACs & CCA-secure schemes

Maria Francis

October 16, 2020

A Fixed-Length (single block) MAC

- Pseudorandom Funtions are a natural tool for constructing MACs.
- i.e. PRFs is a MAC. $Verify_k(m, t) = 1$ iff $t = F_k(m)$.
- Intuition:
  - Forging a tag on a unknown/new message requires $\mathcal{A}$ to correctly guess the output of a PRF at a new point.
  - This is only negligibly greater than guessing the value of a random function which is $2^{-\ell(n)}$.
- Output length of F_k should be big enough.
  - If $Pr[MAC - forge_{\Pi,\mathcal{A}}(1^n)] = \epsilon$, then $\mathcal{A}$ can break the PRF with advantage $\mathcal{O}(\epsilon - \frac{1}{2^{\ell(n)}})$ where $\ell(n)$ is the output length/block length of F_k.
  - If $f \in Func_n$ is used then probability of forgery, $\epsilon = 2^{-\ell(n)}$.

Security Proof

- Like previous cases involving PRFs we use the Random Function model, i.e. we show replacing a PRF with a truly random one ($f \in Func_n$), to get $\overline{\Pi} = (\overline{Gen}, \overline{MAC}, \overline{Verify})$, is only a negligibly different scenario.
- We then analyze the security of $\overline{\Pi}$.
- For any message $m \notin \mathcal{Q}$, $t = f(m)$ is uniformly distributed in $\{0,1\}^n$ and thus we have,

$$Pr[MAC - forge_{\mathcal{A},\overline{\Pi}}(1^n) = 1] \leq 2^{-n}.$$

- What we need to show then is :

$$|Pr[MAC - forge_{\mathcal{A},\Pi}(1^n) = 1] \\ - Pr[MAC - forge_{\mathcal{A},\overline{\Pi}}(1^n) = 1]| \leq \mathrm{negl(n)}$$

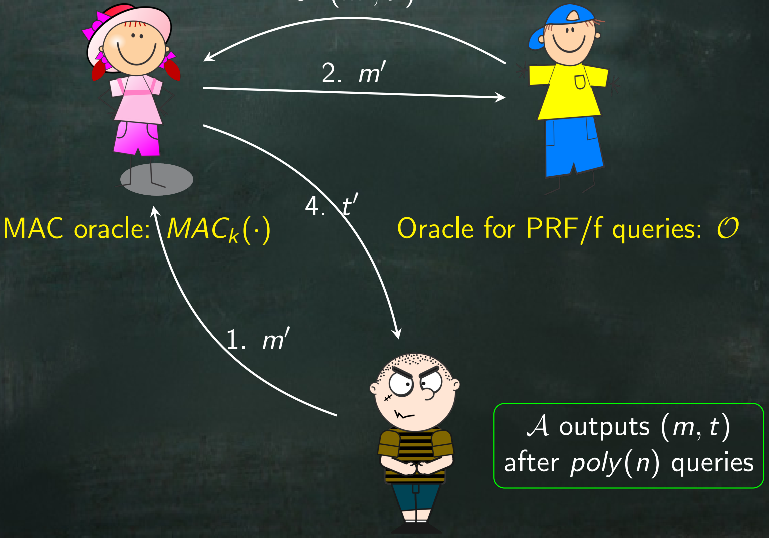
Working with f is not very different

- Note that when we have the two previous equations, we have the required final result,

$$Pr[MAC - forge_{\mathcal{A},\Pi}(1^n) = 1] \leq 2^{-n} + \mathrm{negl}(n).$$

- So now back to showing (2): we have to build a distinguisher D that distinguishes F_k and f.
- D emulates the message authentication experiment for $\mathcal{A}$ and sees if $\mathcal{A}$ succeeds in outputting a valid tag on a new message.
- If yes, D guesses that its oracle is the PRF F_k, else it guess it is $f \in Func_n$.

D runs $\mathcal{A}$:



$\mathcal{A}$ outputs (m, t)
after $poly(n)$ queries

Distinguisher D

- When $\mathcal{A}$ outputs (m, t) at the end, D does the following:
  - Query $\mathcal{O}$ with m and gets response $\bar{t}$.
  - If $t = \bar{t}$ and $\mathcal{A}$ has never queried m before then D outputs 1 else 0.
- D runs in polynomial time.
- If D's oracle is PRF, then the view of $\mathcal{A}$ when it runs as D's subroutine is the same as in $MAC - forge_{\mathcal{A},\Pi}(1^n)$. D outputs 1 when $MAC - forge_{\mathcal{A},\Pi}(1^n) = 1$.

$$Pr[D^{F_k()}(1^n) = 1] = Pr[MAC - forge_{\mathcal{A},\Pi}(1^n) = 1].$$

Similarly for f $(D^{f()}(1^n) = 1)$ and $MAC - forge_{\mathcal{A},\overline{\Pi}}(1^n)$.

MAC for Multiple-Block Messages

- For messages longer than one block.
- Using MAC for single block we can build multiple-block but it is inefficient. We will see that.
- From a PRF, build a PRF that takes inputs that are of length greater than a single block.
- See how abstracting ideas helps! Seeing AES/DES as PRFs or stream ciphers as PRGs helps us them as building blocks for other primitives.

MAC for Multiple-Blocks

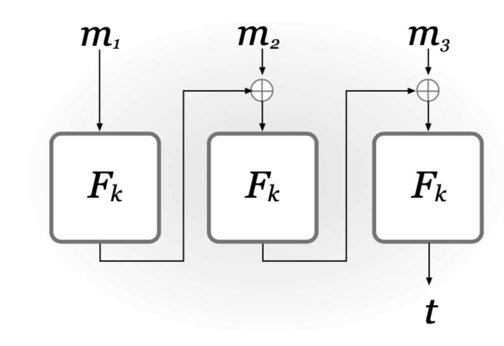
- Before we get to our simple (but inefficient) solution let us eliminate basic ideas:
- What happens when we authenticate each block separately?
  - Block reordering attack will go undetected.
- We add a sequence number i to each block. $t_i = MAC_k(i \circ m_i)\ \forall i$.
  - Truncation attack: drop blocks at the end.
- We add total length of message ℓ in bits, $t_i = MAC_k(\ell \circ i \circ m_i)\ \forall i$.
  - Mix-and-match attack: adversary combines blocks from different messages.
  - $\mathcal{A}$ obtains tags $t_1, \ldots, t_d$ and $t'_1, \ldots, t'_d$ on $m = m_1, \ldots, m_d$ and $m' = m'_1, \ldots, m'_d$ resply. $\mathcal{A}$ outputs a valid tag $t_1, t'_2, t_3, t'_4, \ldots$ on the message $m_1 m'_2, m_3, m'_4, \ldots$.
  - Use a random message identifier!

MAC for Multiple-Block Messages – Simple Solution

- That is a block $B_i = (r, \ell, i, M_i)$, r message identifier, ℓ is the total message length, i the sequence number and M_i the message.
- $MAC(m) = (r, (MAC(B_i)_{i=1,\dots,t}$.
- What are the uses of all these components?
  - r - prevents mixing of the blocks from two messages,
  - ℓ - prevents dropping, and
  - i - prevents rearranging
- Inefficient as tag length increases with message length.
- We do not consider its formal security proof.

CBC-MAC

- Widely used in practice. Secure for messages of fixed length, but not secure in general case.
  Attacks are possible by extending a previous signed message.



CBC-MAC for fixed-length messages

- Let F be a PRF and fix length function $\ell > 0$.
- *MAC*: On input $k \in \{0,1\}^n$ and m of length $\ell(n) \cdot n$,
  - $t_0 = 0^n$ and parse m as $m = m_1, \cdots, m_\ell$, $|m_i| = n$.
  - For $i = 1$ to ℓ: Set $t_i := F_k(t_{i-1} \oplus m_i)$.
  - t_ℓ is the tag.
- *Verify*: If m is not of length $\ell(n) \cdot n$ then output 0 else output 1 iff $t = MAC_k(m)$.

CBC-MAC

Theorem

Let ℓ be a polynomial. If F is a PRF, then the above construction is a secure MAC for messages of length $\ell(n) \cdot n$.

We do not go into the details of the proof.

Why cannot the above construction just be extended to arbitrary multiples of n? The construction is only secure when the length of the message being authenticated is fixed and agreed upon ind advance by the honest parties! – (Practice Question!)

CBC-MAC

- Unlike first case, here we can authenticate longer messages.
- CBC-MAC very similar to CBC, but there are differences:
  - CBC uses random IV for security while CBC-MAC uses no IV or rather a fixed value 0^n for security. – (Practice q)
  - CBC outputs all intermediate values not CBC-MAC. If it outputs all the $\{t_i\}$ it is no longer secure. – (Practice q)

CBC-MAC for arbitrary(poly) length messages

- We still want to produce a single block tag and the MAC should be secure if the underlying function is a PRF.
- Prepend the message m with its length $|m|$ and then do basic CBC-MAC. Appending $|m|$ to the end is not secure. – (Assignment q)
- Change the scheme so that Gen chooses two independent keys k_1 and k_2. To authenticate m:
  - $t = CBC - MAC_{k_1}(m)$
  - $\bar{t} := F_{k_2}(t)$ – the actual tag for m.
- You can authenticate without knowing message length in advance but you need two keys - not desirable.
- These variations are called CMAC and EMAC.
- MAC from a hash function instead of a PRF - HMAC!Later!

Authenticated Encryption

- Can any CPA scheme Π_E (with k_E key) and any MAC scheme Π_M (with k_M key) give us authenticated encryption? No! They need to combined in a certain way else the result can be insecure even if the underlying tools are secure!
- Three natural approaches:
  1. Encrypt-and-authenticate: Π_E and Π_M work in parallel. For a plaintext m, the ciphertext $\langle c, t\rangle$ is formed in this way:
  $$c \leftarrow Enc_{k_E}(m) \text{ and } t \leftarrow MAC_{k_M}(m).$$
  2. Authenticate-then-encrypt: For m, c is computed as:
  $$t \leftarrow MAC_{k_M}(m) \text{ and } c \leftarrow Enc_{k_E}(m \circ t)$$
  3. Encrypt-then-authenticate: For m, c is computed as:
  $$c \leftarrow Enc_{k_E}(m) \text{ and } t \leftarrow MAC_{k_M}(c)$$

What can go wrong?

1. No secrecy. $t \leftarrow MAC_{k_M}(m)$ can leak m to Eve.
2. A specific attack is possible where the *Verify* fails not just when the tag is not valid but also when there is a bad padding (See 3.7.2. In Katz and Lindell textbook)
3. This approach is sound and results in an authenticated encryption scheme as long as the MAC is a strong MAC.
4. We omit the proof.

CCA-secure Vs Authenticated Encryption

- Encryption with authentication implies CCA-secure encryption!
- Modifying ciphertexts in a CCA is linked to message integrity.
- Can there be CCA-secure SKE schemes that are not unforgeable? - Yes! (Practice q)
- But most constructions of CCA satisfy the stronger definition of authenticated encryption. I.e, why use a CCA-secure scheme that is not an authenticated encryption scheme, when any construction satisfying the latter is more efficient than constructions achieving the former?
- But conceptually different: CCA looks at $\mathcal{A}$ that can interfere and in MACs we are looking for message integrity.
- In Public Key systems the difference is more pronounced.