Lecture 6

FEBRUARY 7

Instructor: Fang Song

Disclaimer. Draft note. No guarantee on completeness nor soundness. Read with caution, and shoot me an email at fsong@pdx.edu for corrections/comments (they are always welcome!)

Logistics. Quiz 1 solutions posted on D2L.
Last time. CPA, MAC intro.
Today. MAC from PRFs, domain extension.

1 Data Integrity

Secrecy vs. integrity. Secrecy is not the only concern. Sometimes we care less about whether the message is heard by someone else, but rather, we want to make sure the message was not modified intentially by an adversary. Two examples

- Software update. You update your computer or phone applications from e.g. Apple Store from time to time. Here secrecy is irrelevant, but how do you know the patch program is indeed coming from Apple and the copy you downloaded hasn't been modified ?(In practice, this is achieved by public-key authentication, i.e., digital signatures.)
- Bank transfer. Suppose the Bank receives a money transfer request from user A to user M of amount X. Is this a valid request? Is X the right amount? (what if M is greedy and malicious?)

Formally speaking, there are two problems we are facing *identity* authentication (identification) and *message* authentication. But in our setting identification is usually implied by message authentication assuming certain secret information (i.e. key) for achieving message authentication is only known to honest users and safely protected. Therefore we will focus on message authentication hereafter.

Encryption does not immediately provide integrity.

- OTP or stream cipher. Suppose we encrypt one-bit by OTP as our vote (for P? 1 →C and 0 → T). Then it's easy to just flip the ciphertext which will completely change the winner.

[How about CPA?]

- (R)CTR: $c[j] := F_k(\mathsf{IV} + j) \oplus m[j]$ suffers from the bit-flipping attack for the same reason.
- CBC: $c[0] = \mathsf{IV}, c[1] = F_k(\mathsf{IV} \oplus m[1])$. Therefore if I just flip the ℓth bit of IV, then the decrypted message $\hat{m}[1]$ will have the ℓth bit flipped too.

Note that in these examples secrecy is not compromised since the adversary didn't gain any useful information about the underlying plaintext. Nonetheless, he is able to change what honest user gets to see after decryption.

1.1 Message authentication codes (MAC)

Instead the right tool cryptographers designed for protecting integrity is called *message authenticaton code* (MAC).

FS NOTE: Draw MAC diagram

Definition 1 (KL-Def. 4.1)**.** A message authentication code (MAC) consists of three poly-time algorithms (G, T, V):

- (Key-Gen, randomized) G: takes 1^n as input and generates k with $|k| \geq n$.
- (Tag-Gen, possibly randomized) S: takes k and message $\{0,1\}^*$, generates $t \leftarrow S_k(m)$.
- (Verification, deterministic) V: takes key k, a message $m \in \{0,1\}^*$ and a tag t. Output $b := V_k(m,t)$ where $b = 1$ indicates valid, and $b = 0$ indicates invalid.

Canonical Verification for deterministic MAC. Suppose T is deterministic. $V_k(m,t)$ often goes as follows: compute $\tilde{t} \leftarrow S_k(m)$ and compare $\tilde{t} \overset{?}{=} t$.

1.2 Defining secure MAC

What do we want from a MAC?

- Security goal: intuitively we want that $\mathscr{A}$, without knowing the secret key, should not be able to produce a valid pair (m,t) such that $V_k(m,t) = 1$. We call it a forgery.
- Attack model: we assume the communication channel between honest users is public (such as the Internet), so $\mathscr{A}$ could intercept many valid (m_i, t_i) pairs. It's also feasible for $\mathscr{A}$ to control over the messages sometimes. Consider the bank transfer example. Suppose user M is an ebay seller, and carries out a few legit transactions with user A. The content of the messages, i.e. the prices, is under user M's control. Therefore, to be safe, we allow an adversary to request MAC tags on *any* messages of its choice.
- Security goal **refined**: we consider it a "break" of MAC if $\mathscr{A}$ could come up with a forgery (m,t) and it never asked a tag of m before.

To make it formal, we consider the following game. Let $\Pi = (G, T, V)$ be a Mac, $\mathscr{A}$ an adversary,

FS NOTE: Draw eu-cma diagram

1. CH generates key $k \leftarrow G(1^n)$.
2. Adversary $\mathscr{A}$ is given 1^n and oracle access to $S_k(\cdot)$ (i.e., $\mathscr{A}$ can make queries m_i and obtain $t_i \leftarrow S_k(m_i)$). Let $\mathscr{L}$ be the set of all queries that $\mathscr{A}$ asked.
3. $\mathscr{A}$ outputs (m^*, t^*) in the end. $\mathscr{A}$ succeeds iff. (1) $V_k(m^*, t^*) = 1$ and (2) $m^* \notin \mathscr{L}$. In that case define the output of the game $\mathsf{Mac\text{-}forge}_{\mathscr{A},\Pi}(n) = 1$.

Figure 1: The message authentication experiment $\mathsf{Mac\text{-}forge}_{\mathscr{A},\Pi}(n)$

Definition 2. MAC Π is existentially unforgeable under an chosen-message attack or eu-cma-secure, if for all PPT $\mathscr{A}$,

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathscr{A},\Pi}(n) = 1] \leq \mathrm{negl}(n)\,.$$

Remarks and discussion.

- Did you wonder, since secrecy is not a concern here, is a secret key really necessary? It is!
- "Existentially unforgeable" refers to the fact that the adversary cannot forge a valid tag on *any* message, i.e., there doesn't exist a message efficient $\mathscr{A}$ can forge on.
- Is the definition "too" strong? After all, we probably only care about no-forgery on "meaningful" messages. But "meaningful" is not an absolute term and is *application dependent*. Be conservative and save further worries in applications.
- **Replay attacks** The security definition itself does not prevent a simple attack in practice: copy a previous message-tag pair and resend it to an honest user at a later point. Again, consider the greedy ebay seller Mr. M, what if he keeps forwarding the money transfer request ? Two common techniques for thwarting such attacks:

  1. *Sequence number (counter)*
  2. *time-stamps* $t = S_k(\mathsf{TIME} \| m)$ and verify $V_k(\mathsf{TIME} \| m, t) = 1$, and TIME is "recent".

  Both need synchronization to some extend.

1.3 A fixed-length eu-cma-secure MAC from PRFs

Let $F_k : \{0,1\}^n \to \{0,1\}^n$ be a PRF, construct MAC scheme $\Pi = (G, S, V)$

- $G(1^n)$: $k \leftarrow \{0,1\}^n$ (random key for F_k).
- $S(m)$: $t := F_k(m)$.
- $V(m,t)$: compute $t' := F_k(m)$ and check $t' \overset{?}{=} t$.

Figure 2: A fixed-length MAC from any PRF

Theorem 3 (KL-Thm. 4.6)**.** Π *is an eu-cma MAC (for messages of length n).*
Idea. Intuitively forging in this scheme amounts to predict the output of a PRF on a new point, which should be infeasible, especially if we think of a truly random function. Specifically, let $\mathscr{A}$ be any adversary trying to produce a forgery. Let $q(n)$ be an upper bound on its number of MAC-queries. Again, consider a variant

$$[\ \tilde{\Pi} \text{ where we use a truly random function } f \leftarrow \mathscr{F}.\ ]$$

For any candidate forgery (m^*, t^*), where m^* is a new message, $y := f(m^*)$ is sampled uniformly at random (by the "sample-on-the-fly" interpretation of a truly random function). Therefore y would differ from t^* except with probability 2^{-n}. We state this in a Lemma.

Lemma 4. *For any* $\mathscr{A}$, $\Pr[\mathsf{Mac\text{-}forge}_{\mathscr{A},\tilde{\Pi}}(n)] \le 2^{-n}$.

Then we show that switching back to a PRF, does not make the adversary's life any easier based on the security of PRF. Therefore we conclude that Theorem 3 is eu-cma.

Lemma 5. $$\left|\underbrace{\Pr[\mathsf{Mac\text{-}forge}_{\mathscr{A},\Pi}(n) = 1]}_{p_{\mathscr{A},\Pi}} - \underbrace{\Pr[\mathsf{Mac\text{-}forge}_{\mathscr{A},\tilde{\Pi}}(n) = 1]}_{p_{\mathscr{A},\tilde{\Pi}}}\right| \le \mathrm{negl}(n).$$

Proof. For any $\mathscr{A}$, we construct a distinguisher D and show that

$$\left|p_{\mathscr{A},\Pi} - p_{\mathscr{A},\tilde{\Pi}}\right| \le \left|\underbrace{\Pr[D^{F_k}(1^n) = 1]}_{p_{D,k}} - \underbrace{\Pr[D^{f}(1^n) = 1]}_{p_{D,f}}\right| \le \mathrm{negl}(n)\,.$$

```
Distinguisher $D$: given input $1^n$ and oracle access $\mathscr{O} : \{0,1\}^n \to \{0,1\}^n$:
  1. Run $\mathscr{A}(1^n)$. Whenever $\mathscr{A}$ makes a MAC-query on $m$, forward $m$ to
     $\mathscr{O}$ and return $t := \mathscr{O}(m)$.
  2. $\mathscr{A}$ outputs $(m^*, t^*)$ in the end. Let $Q = \{m_i\}$ be the list of $\mathscr{A}$'s MAC-
     queries. $D$ does the following
       a) Query $\mathscr{O}$ with $m^*$ and obtain $\hat{t} := \mathscr{O}(m^*)$.
       b) Output 1 iff. both $\hat{t} = t$ and $m^* \notin Q$ hold.
```

Observe that

- if $\mathscr{O}$ is truly random: then $\mathscr{A}$ sees exactly as in the forgery game $\mathsf{Mac\text{-}forge}_{\mathscr{A},\tilde{\Pi}}(n)$. Therefore D outputs 1 iff. $\mathscr{A}$ produces a valid forgery (i.e. succeeds in $\mathsf{Mac\text{-}forge}(n)$). We have $p_{D,f} = p_{\mathscr{A},\tilde{\Pi}}$.
- similarly, if $\mathscr{O} = F_k$ is pseudorandom, we see that $p_{D,k} = p_{\mathscr{A},\Pi}$.

Thus $|p_{\mathscr{A},\Pi} - p_{\mathscr{A},\tilde{\Pi}}| = |p_{D,k} - p_{D,f}| \le \mathrm{negl}(n)$. □

2 Domain-extension for PRFs

In practice, our block ciphers work on a data block of small length, e.g. 128-bit, how do we MAC long messages? We will discuss two general approaches:

1. Hash-and-MAC paradigm. Apply a hash function to "compress" the input string to a shorter one that fits your MAC. (NEXT LECTURE)
2. Direct approach: domain extension. (Below)

Given MAC on short inputs, construct a MAC on long inputs.

Natural ideas (that often fail).

1. block-by-block? *Reordering attack*
2. including block index? *truncation attack*
3. including message length in each block? *mix-&-match attack*. Consider
$$m = m[1]\|\ldots\|m[d], t = t_1,\ldots,t_d\,; \quad m' = m'[1]\|\ldots\|m'[d], t' = t'_1,\ldots,t'_d\,.$$
Then $\langle t_1, t'_2, t'_3, \ldots\rangle$ is a valid tag for $m[1], m'[2], m'[3], \ldots$.
4. additional random identifier. $S'(m[i]) := S(r\|\ell\|i\|m[i])$. This works, but very inefficient! We will not discuss further about it. **FS NOTE**: Read [KL: Thm 4.8] for details.

Domain extension for PRFs. We ask a slightly different question:

> Given PRF on short inputs, construct a PRF on long inputs.
> i.e., given $F: X \to Y$ how to get $F': X^{\leq \ell} \to Y'$, for $\ell \geq 1$? (assume $X = Y = Y' = \{0,1\}^n$)

If this is possible, then we will just use the PRF on longer messages to achieve message authentication on long messages. We show this is indeed possible in two steps.

- Sect. 2.1: extend any PRF to a PRF against prefix-free distinguishers.
- Sect. 2.2: make it fully-secure.

2.1 Prefix-free secure PRFs on long messages

Basic CBC. Let $m = m[1]\|\ldots\|m[d], d \leq \ell$.

$$t_i = F_k(m[i] \oplus t_{i-1}) \quad \text{and only output } t_d\,.$$

FS NOTE: Draw CBC

How it differs from CBC mode for *encryption*?

1. no IV, or rather IV $= 0^n$ is fixed rather than randomized.
2. no output of intermediate output strings. Clearly a gain in efficiency.

Both are crucial to ensure a secure (fixed-length) MAC.
Cascade construction.

$$t_1 = F_k(m[1]), t_i = F_{t_{i-1}}(m[i]) \quad \text{and only output } t_d\,.$$

FS NOTE: Draw Cascade

These two constructions are almost secure PRFs with one constraint.

Definition 6 (Prefix-free set & algorithm)**.** A set of strings $P \subseteq (\{0,1\}^n)^*$ is *prefix-free* if it does not contain the empty string (i.e. $\varepsilon \notin P$), and no string $x \in P$ is a prefix of any other string $x' \in P$. We call algorithm D with oracle access to f *prefix-free* if D only queries on a prefix-free set.

Theorem 7. *If F is a* PRF*, then* CBC_F *and* $\mathsf{CASCADE}_F$ *are* PRF *against any prefix-free PPT distinguisher D.*

In particular if we fixed the message length to be $\{0,1\}^{n\cdot\ell}$ for any ℓ, then the prefix-free constraint is trivally true because no string can be a prefix of another string of the same length. As an immediate consequence, we have

Corollary 8. *If F is a* PRF*, then* $\mathsf{CBC}_{F,\ell}$ *and* $\mathsf{CASCADE}_{F,\ell}$ *are eu-cma MACs for messages of length* $\{0,1\}^{n\cdot\ell}$ *for any fixed* $\ell \geq 1$.

2.2 Fully-secure PRFs on long messages

Prefix-free Encoding. A natural idea would be to introduce an encoding mechanism that ensures prefix-freeness. Some examples[1]

- prepending message-length. Not practical since it's not suitable for data streams.
- stop bits. $m[1]\|0, m[2]\|0, \ldots, m[d]\|\mathbf{1}$. Inefficient.
- randomized encoding. NIST standard: **CMAC**. CBC with randomized prefix-free encoding.

Encrypted PRF.

- **ECBC**.

$$\mathsf{ECBC}_{k_1,k_2}(\cdot) := F_{k_2}(\mathsf{CBC}_{F_{k_1}}(\cdot)).$$

  FS NOTE: Draw ECBC diagram

  Variants as ANSI (ANSI X9.9 and ANSI X9.19) and ISO (ISO 8731-1 and ISO/IEC 9797) standards.

- **Encrypted Cascade** a.k.a **NMAC** (Nested MAC). A variant of it (using a hash function instead of a PRF in the cascade construction) called HMAC is widely used in the Internet (rfc2104).

$$\mathsf{ECAS}_{k_1,k_2}(\cdot) := F_{k_2}(\mathsf{CASCADE}_{F_{k_1}}(\cdot)).$$

  FS NOTE: Draw NMAC diagram

Theorem 9. *ECBC and NMAC are* PRF*s.*

Formal proofs are beyond the scope of this course. Read Boneh-Shoup Chapter 7 if interested.

[Note that both are **streaming** MACs, since we do not need to know the message length ahead of time.]

[1] Read more on Boneh-Shoup Sect. 6.6.