Winter 2018 CS 485/585 Introduction to Cryptography

Lecture 10

Portland State University *Feb. 8, 2018*

Lecturer: Fang Song

Draft note. Version: February 11, 2018. Email fang.song@pdx.edu for comments and corrections.

Agenda

- (Last time) HMAC, AE, ENC-then-MAC
- CCA/AE formal definitions
- Theoretical constructions based on one-way functions

Authenticated encryption: formal definitions

What do we mean exactly by authenticated encryption? We give a formal definition integrating a strong notion of secrecy (against chosen-ciphertext-attacks) and a notion of unforgeability.

Defining CCA-secure encryption

AE requires a stronger notion for secrecy than CPA. We consider the attacking model called *chosen-ciphertext-attacks* (CCA).

A chosen-cipher-attack has access to, in addtion to the encryption oracle, the decryption oracle when playing the indistinguishability game. To avoid trivializing the definition, the obvious constraint is to refuse answering decryption query on the challenge ciphertext that the adversary needs to guess which of the two messages it encrypted. Otherwise, we put no more restrictions on how the adversary interrogates the two oracles.

Read [KL: Sect. 3.7.1] for the formal description of the CCA indistinguishability game. We denote the output of the game $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1$ iff. $\mathcal{A}$ wins the game, i.e., it guesses correctly the random bit chosen by the challenger.

Definition 1 ([KL: Def. 3.33])**.** A private-key encryption scheme Π is CCA-secure, if for all PPT adversaries $\mathcal{A}$,

$$\Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathrm{negl}(n)\,.$$

Do we have any CCA-secure schemes?

- PRF-OTP is not CCA-secure. [1]
- How about Randomized counter-mode of a block cipher?

[1] Let $m_0 = 0^n$ and $m_1 = 1^n$. Receive $c = (r, s = F_k(r) \oplus m_b)$, flip first bit of s to obtain c'. Query DEC oracle on c': if 10^{n-1} then answer 0; if 01^{n-1}, answer 1.

Is CCA realistic? Full decryption oracle seems unlikely. However, even a *partial* decryption oracle could be disastrous. A very effective attack called *padding oracle attack* breaks CBC-encryption, by exploiting an oracle that just tells if a ciphertext is well-formed.

Enc-then-MAC is CCA-secure. We claimed earlier that Enc-then-MAC with a CPA-secure encryption always gives an authenticated encryption. In particular, the resulting scheme is a CCA-secure encryption. The intuition behind it is that authenticating the ciphertext produced by the CPA-scheme essentially makes a decryption oracle useless. Consider a ciphertext that the adversary asks the decryption oracle:

i. it is the ciphertext returned from a previous encryption query. Nothing new can be learned.

ii. it is a new ciphertext. This helps the adversary for sure, however, this implies that $\mathcal{A}$ has forged a pair of ciphertext and tag. Breaking MAC!!!

Defining authenticated encryption

CCA security will be the secrecy requirement for AE. We also need some authentication condition, which is captured by unforgeability.

Definition 2. A private-key encryption scheme Π is unforgeable if for all PPT adversaries $\mathcal{A}$,

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

1. CH generates key $k \leftarrow G(1^n)$.
2. Adversary $\mathcal{A}$ is given 1^n and oracle access to $E_k(\cdot)$ (i.e., $\mathcal{A}$ can make queries m_i and obtain $c_i \leftarrow E_k(m_i)$). Let $\mathcal{L} := \{m_i\}$ be the set of all queries that $\mathcal{A}$ asked.
3. $\mathcal{A}$ outputs c in the end. Let $m \leftarrow D_k(c)$. The output of the game $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1$ iff. (1) $m \neq \perp$ and (2) $m \notin \mathcal{L}$; in this case we say that $\mathcal{A}$ wins.

Figure 1: The unforgeable encryption game $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)$

Definition 3. A private-key encryption scheme is an authenticated encryption scheme if it is CCA-secure and unforgeable.

How about MAC with verification oracles? [KL: Exercise 4.2,4.3]
Inspired by chosen-ciphertext-attacks, you may wonder, what if a MAC-attacker has in addition a verification oracle at hand?

- Simple observation: a MAC with canonical verification remains secure, since verification oracle could have implemented by an adversary by querying the signing oracle.
- Less obvious: there exists a secure MAC that becomes broken once a verification oracle is available. Can you think of one?

Find on the course webpage a diagram summarizing private-key cryptography `http://fangsong.info/teaching/w18_4585_icrypto/w18_cs4585_privksum.pdf`.

	fixed length	variable length
Insecure ENC		ECB mode
Perfect secrecy	OTP	
Comp. Secrecy	PRG-OTP*	Counter mode
CPA	PRF-OTP	CBC mode
		Randomized Counter mode
MAC (prefix-free)		Basic-CBC
		Cascade
MAC (fully-secure)	PRF-MAC	Encrypted CBC
		Encrypted Cascade (NMAC)
		HMAC: NMAC using hash
Authenticated ENC	Enc-then-MAC	Enc-then-MAC

Table 1: Private-key primitives: other than OTP, PRG-OTP, HMAC, the rest are all build from Block ciphers (PRF/PRP). *: practical stream ciphers are (dynamic) variable-length.

Theoretical constructions of Priv-key crypto

How to construct private-key cryptography based on complexity-theoretical foundations?

We showcase some beautiful theory in the foundations of (modern) private-key cryptography. We will see how to start from a mathematical abstraction called *one-way functions*, which looks quite weak and "bland", to build step by step the entire private-key cryptography. Namely

Theorem 4 (Informal)**.** *One-way functions are equivalent to the existence of all private-key cryptogrraphy.*

This lecture is one of my favorite. But it might be quite abstract and a bit advanced. For the most constructions, you just need to appreciate the high-level ideas and the intelligent elegance. The idea of computational indistiguishability and the technique of hybrid argument are nonetheless worth digesting and internalizing.

One-way functions

One way functions are no-doubt *a* indispensable (theoretical) foundation of modern cryptography.

Definition 5. A function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if the two conditions hold:

1. Easy to compute: There is a poly-time algorithm M_f computing f, i.e., $\forall x, M_f(x) = f(x)$.
2. Hard to invert (image of **random** input): for any PPT $\mathcal{A}$,

$$\Pr[f(x') = f(x) : x' \leftarrow \mathcal{A}(1^n, f(x)), x \leftarrow \{0,1\}^n] \leq \mathrm{negl}(n)\,.$$

If in addition f is a permutation (length-preserving and one-to-one), then we call it a one-way permutation.

You can interpret the expression in 2nd condition as an *inverting* game, and one-way means that no efficient adversary can succeed in that game except with negligible probability.

Remark 1. The definition is easy to understand intuitively, but there are many possible common misunderstandings:

- The hard-to-invert property is an *average-case* statement. It is not merely saying that there exists some y that is difficult to invert in the *worst-case*. Rather, just by picking a uniformly random x, it is already hard to find a preimage of $f(x)$.
- Similarly, a non-one-way function, i.e. a function that is NOT one-way, is not necessarily easy to invert all the time. For example, a function that is easy to invert on even inputs, but hard to invert on odd inputs is not one-way.
- Spending enough (exponential) time, inverting is always possible. So this only makes sense in the *computational* regime.

We will see more examples later. Here we give informally a candidate $f(p,q) := p \cdot q$, where (p,q) are (encodings) of prime numbers of certain length. Inverting f is the *facterization* problem, which is believed hard (at least as classical computers are concerned. It can be solved efficiently on a *quantum* computer though.)

Hard-core predicate

Suppose f is a OWF, does it mean given $y := f(x)$ on random x, it is difficult to learn any information about x (or some other preimage)? [2]

This motivates the notion of a *hard-core predicate*, $\mathsf{hc} : \{0,1\}^* \to \{0,1\}$. Basically $\mathsf{hc}(x)$ captures the one-bit information about x that is the "hard" to figure out from $f(x)$, and hence makes inverting f hard.

[2] Not necessarily. In fact, for any OWF f, we can construct

$$g(x_1, x_2) := (x_1, f(x_2))\,.$$

g reveals half of its input in plain, nonetheless, you will prove in homework that g is still a OWF.

Definition 6. A function hc is a hard-core predicate of f if it can be computed in poly-time, and for every PPT $\mathcal{A}$,

$$\Pr[b' = b : b' = \mathcal{A}(1^n, f(x)), b = \mathsf{hc}(x), x \leftarrow \{0,1\}^n] \leq 1/2 + \mathrm{negl}(n)\,.$$

Remarks about a hard-core predicate

- $\mathsf{hc}(\cdot)$ is not a stand-alone object, it is relative to a function f. While the definition applies to any function, we will be primarily concerned with hard-core predicates of a one-way function f.
- (a trivial example) if $f(x_1 \dots x_n) = x_1 \dots x_{n-1}$. Then $\mathsf{hc}(x) := x_n$ is trivially a hard-core predicate of f.
- (a simple idea fails) Is $\mathsf{hc}(x) := \bigoplus_{i=1}^{n} x_i$ a good one? $g(x) := (f(x) \oplus x_i)$ is a OWF, but it trivially reveals $\mathsf{hc}(x)$.
- for $f(p,q) = p \cdot q$, the least significant bit of $x = (p,q)$ is a hard-core bit if factoring is indeed hard.

The first surprising theorem in our trip, which we will not prove, asserts that if one-way function exists, then there must also exist a one-way function that has a hard-core predicate.

Theorem 7 ([KL: Thm.7.5] Goldreich-Levin Theorem)**.** *Assuming one-way functions (permutations) exist, then there exists (explicit constructions of) a one-way function (permutation) g and a hard-core predicate hc of g.*

PRGs from OWFs

Because of the *unpredictability* feature of a hard-core predicate, we can naturally construct

Theorem 8 ([KL: Thm. 7.19])**.** *Let f be a one-way permutation with hard-core bit hc. $G(s) := f(s) \| \mathsf{hc}(s)$ is a PRG with expansion factor $\ell(n) = n + 1$.*

How could this construction fail if f is not necessarily a permutation?

In fact one-way functions are also sufficient to construct a pseudorandom generator, as shown in the famous paper by HILL [3].

1-bit surplus does not sound useful enough. We can increase it by composing the basic PRG in certain ways. We will illustrate an important notion *computational indistinguishable*, and a basic proof technique *hybrid argument*.

[3] A Pseudorandom Generator from any One-way Function. Read More: http://epubs.siam.org/doi/abs/10.1137/S0097539793244708.

PRG Composition: parallel

Given $G : \{0,1\}^n \to \{0,1\}^{n+1}$, construct $G' : \{0,1\}^{tn} \to \{0,1\}^{t\cdot(n+1)}$

$$PG(s_1, \dots, s_t) := G(s_1) \| \dots \| G(s_t)\,.$$

Theorem 9. *PG is a PRG.*

Parallel composition can be run efficiently on parallel processors/machines, but it does not increase the expansion factor[4].

[4] $\ell'(n)/|s'| = 1 + \frac{1}{n} = \ell(n)/|s|$

PRG Composition: sequential (Blum-Micali)

Blum-Micali proposed another strategy SG which is essentially sequential composition.

$$\begin{aligned} G_1(s) &= G(s) \\ G_2(s) &= G([G_1(s)]_{1,\ldots,n}) \| [G_1(s)]_{n+1} \\ &\cdots \\ SG(s) &:= G_t(s)\,. \end{aligned}$$

Draw Blum-Micali diagram

Let's check its expansion factor: $\ell'(n)/|s'| = 1+\frac{t}{n} > \ell(n)/|s| = 1+\frac{1}{n}$. If we take $t = \text{poly}(n)$ then we have

Theorem 10 ([KL: Thm. 7.20])**.** *If there exists a PRG G with expansion factor $n+1$, then for any polynomial* poly*, there is a PRG SG with expansion factor* poly(n)*.*

Pseudorandom functions and permutations

PRFs from PRGs

Goldreich-Goldwasser-Micali construction [KL: Construction 7.21]:

Denote $G(s) = G_0(s)\|G_1(s)$ where $G_b(s) \in \{0,1\}^n$.

$$F_k(x_1x_2\ldots x_n) := G_{x_n}(\ldots G_{x_2}(G_{x_1}(k))\ldots)\,.$$

Imagine a binary tree with depth n. Associate each node v an n-bit string $str(v)$, where at the root $str(root) = k$ we put the random key. For each node v, compute $G(str(v)) = z_1\|z_2, z_i \in \{0,1\}^n$, and then assign left child $str(L_v) = z_1$ and right child $str(R_v) = z_2$.

Each leaf can be paired up with an input-string x. From left to right $l_{0\ldots0}, l_{0\ldots1}, \ldots, l_{1\ldots1}$. Then we define $F_k(x) = str(l_x)$. Note that we don't have to construct the tree entirely in order to compute F_k. Instead, to compute $F_k(x)$, just follow the path from root to the leaf l_x and invoke G at each level. This only takes n executions of G and is efficient.

Draw GGM tree. For starters: build a function that takes 1-bit inputs, and then 2-bit inputs, ...

PRPs from PRFs

The Feistel network we saw earlier (DES) transforms a function to a permutation. Indeed Luby-Rackoff proved that

Theorem 11 ([KL: Thm. 7.23])**.** *If F is a PRF, then $\Sigma^{(3)}$ is a PRP.*

Four and more rounds make a *strong* PRP.