Lecture 4

JANUARY 31

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!)

Last time. Pseudorandom generators and stream ciphers. Block ciphers intro.
Today. Block ciphers

Discussion on PRG.

- Clarification on PRG: a deterministic algorithm, and it is public (known to any adversary). It takes a random seed as input, and presumably it will introduce extra bits deterministically (it has to be expanding) and scramble them with the random seed in some sophisticated way. The output should look random (we've given two equivalent formulations of "random-looking"). Basically any efficient distinguisher taking either an output from PRG or a truly random string can not tell a difference.

- **Getting more and more**: parallel repetition ($tn \to t(n+1)$ for $\ell(n) = n+1$) and sequential repetition ($n \to n+t$). Details in a later lecture.

1 Pseudorandom functions & Block ciphers

We continue our discussion on *pseudorandom functions* (PRFs), which are meant to be "random-looking" functions. Roughly, it means that it behaves the same as a truly random function, at least as far as an efficient observer is concerned. We discussed two ways of thinking about a truly random function: 1) uniformly sample from the set of all possible functions (mapping between n-bit strings); and 2) sample "on-the-fly".

To make things concrete, let's consider length-preserving *keyed* functions

$$F : \{0,1\}^{\ell_{\text{key}}(n)} \times \{0,1\}^{\ell_{\text{in}}(n)} \to \{0,1\}^{\ell_{\text{out}}(n)} .$$

- Security parameter n.

- $\ell_{\text{key}}(n), \ell_{\text{in}}(n), \ell_{\text{out}}(n)$: the key length, input length and output length.

- **length-preserving**: assuming $\ell_{\text{key}}(n) = \ell_{\text{in}}(n) = \ell_{\text{out}}(n) = n$.

- **efficient**: for each k, there is an deterministic algorithm that on input x computes $F(k,x)$ in polynomial-time.

- Notation. For each k, $F_k := F(k,\cdot)$ determines a function $\{0,1\}^n \to \{0,1\}^n$

How do we formalize "random-looking" for a function? How about requiring no efficient distinguisher can tell apart F_k for a random k from a truly random function $f \leftarrow \mathscr{F}$? But how do we specify the function to a distinguisher D? It's just too long ($\Omega(n2^n)$) to write down an arbitrary function in $\mathscr{F}$, and a poly-time D can only see a tiny piece of it.

Instead we consider giving a distinguisher *oracle* access of function f. $D^{f(\cdot)}$ means a oracle algorithm, where D can make multiple queries x_i and get responses $y_i := f(x_i)$ during its execution.

Definition 1. Let F be an efficient, length-preserving, keyed function, F is a pseudorandom function if for any PPT distinguisher D,

$$\left|\Pr[D^{F_k(\cdot)}(1^n) = 1 : k \leftarrow \{0,1\}^n] - \Pr[D^{f(\cdot)}(1^n) = 1 : f \leftarrow \mathscr{F}]\right| \le \text{negl}(n)\,,$$

Discussion on the definition.

- usually use $\mathscr{O}(\cdot)$ to denote an unspecified oracle.
- k is unknown to distinguisher, otherwise trivial to distinguish. PPT D can make poly-many queries to $\mathscr{O}(\cdot)$ at most.
- PRFs against unbounded adversaries *impossible*.
- An insecure example: $F_k(x) = k \oplus x$. More about existence and constructions soon.

Pesudorandom permutations. Block ciphers are a special case, they are "random-looking" *permutations*, which we formalize in the notion of a *pseudorandom permutation*. Let $\Pi := \{\pi \in S_{\{0,1\}^n}\}$ be the set of permutations on n-bit strings. We can define efficient, length-preserving, keyed *permutations* $F_k(\cdot)$ similarly, and we write $F_k^{-1}(\cdot)$ as its inverse permutation[1]. We define PRP as one that is indistinguishable from a truly random permutation (of which you should be able to figure out the meaning).

Definition 2. Let F be an efficient, length-preserving, keyed *permutation*, F is a pseudorandom permutation if for any PPT distinguisher D,

$$\left|\Pr[D^{F_k(\cdot)}(1^n) = 1 : k \leftarrow \{0,1\}^n] - \Pr[D^{\pi(\cdot)}(1^n) = 1 : \pi \leftarrow \Pi]\right| \le \text{negl}(n)\,,$$

Note: We identify a block cipher and a PRP. Why call it a block cipher? It works on data blocks of a fixed length, say 128-bits.

We said earlier block ciphers are special case of PRF. But is this indeed the case? Is a PRP necessarily a PRF? (WHY NON-TRIVIAL?)

The answer is yes as long as the domain of PRP is sufficiently large (super-polynomial). The following is true:

Proposition 3. *For any D making at most q queries*

$$\left|\Pr[D^{f(\cdot)}(1^n) = 1 : f \leftarrow \mathscr{F}] - \Pr[D^{\pi(\cdot)}(1^n) = 1 : \pi \leftarrow \Pi]\right| \le O(q^2/2^n)\,.$$

[1] Here we require that F_k and F_k^{-1} both can be computed efficiently.

Intuitively, this holds because as long as you don't see a *collision*, i.e. $x \neq x'$ with $f(x) = f(x')$, a RF and RP behave identically. We will defer further discussion when we study hash functions. We immediately get the following corollary

Corollary 4 (KL-Prop.3.17)**.** *If F is a PRP and $\ell_{\text{in}}(n) \geq n$, then F is also a PRF.*
Immediate applications of Block ciphers. Let $F_k(\cdot)$ be a PRP.

- PRG from PRF (PRP).

  $$G(s) := F_s(0), F_s(1), \ldots, F_s(k)\,.$$

  This also implies that we can implement a stream cipher from block cipher (The resulting cipher is sometimes called *deterministic counter mode*). How about the converse? PRF from PRG? Future lecture, stay tuned!

- A computationally secret encryption: (G, E, D)

  - $G : k \leftarrow \{0,1\}^n$.
  - $E : c := F_k(m)$.
  - $D : m := F_k^{-1}(c)$.

1.1 Block-cipher modes of operation

As we will see, practical block ciphers work on a small data block (e.g. 128 bits). How to encipher long messages of multiple blocks? This is called modes of operation of block ciphers.

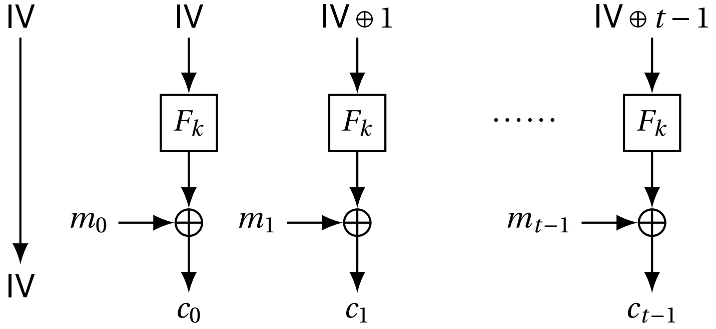
Assume PRP/PRF $F_k(\cdot) : \mathcal{M} \to \mathcal{C}$ with key space $\mathcal{K}$ (e.g. $\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^n, n = 128$), how to encrypt messages in X^t? Notation: for $m \in X^t$, usually divide it into blocks $m[i] \in X, i = 1, \ldots, t$.
Electronic Code Book (ECB) mode.. What's the simplest idea you may think of?

FS NOTE: Draw ECB diagram

Unfortunately, ECB is not secure. Consider $m = m[1] \| m[1]$ and $m' = m[1] \| m[2]$ with $m[1] \neq m[2]$. It is trivial to distinguish the ciphertexts corresponding to m and m'.
Deterministic counter mode (CTR). Recall PRG from PRF.

$$E_k(m) := (F_k(1) \oplus m[1], \ldots, F_k(t) \oplus m[t])\,.$$
$$D_k(c) := (F_k(1) \oplus c[1], \ldots, F_k(t) \oplus c[t])\,.$$

Note that Counter mode works with PRF as well, since decryption does not need F_k^{-1}.
Randomized Counter (RCTR).



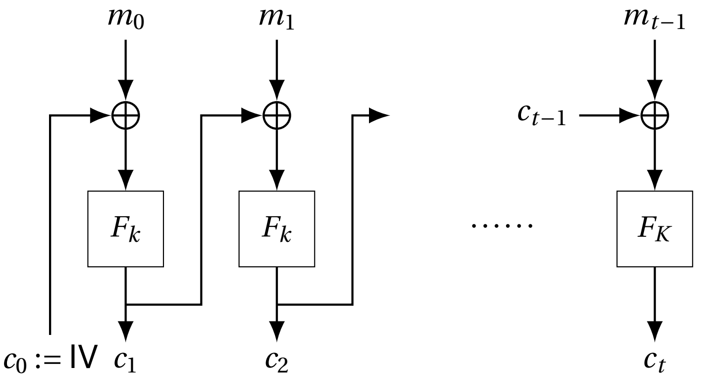
- $E_k(m)$: pick $\mathsf{IV} \leftarrow \mathcal{M}$, for $j = 1, \dots t$,
$$c[j] := F_k(\mathsf{IV} + j) \bigoplus m[j] \quad \text{(addition mod } N = 2^n\text{)}.$$
$c := (\mathsf{IV}, c[1], \dots, c[t])$.
- $D_k(c)$: $m[j] := F_k(\mathsf{IV} + j) \bigoplus c[j]$.

IV is called *initial value* or *counter*.

Distinction from CTR. We pick a random *counter* as starting point, instead of a fixed sequence of inputs to derive key stream. TCTR actually achieves a stronger security notion we will see soon (IND-CPA).

APPLICATION. A variant of AES-RCTR is used in IPsec protocol, specified in RFC 3686[2].

Cipher Block Chaining (CBC).



- $E_k(m)$ pick random $\mathsf{IV} \leftarrow \mathcal{M}$, denote $c[0] := \mathsf{IV}$. For $j = 1, \dots t$,
$$c[j] := F_k(c[j-1] \oplus m[j]).$$
$c := (c[0] = \mathsf{IV}, c[1], \dots, c[t])$.
- $D_k(c)$: for $j = 1, \dots, t$, $m[j] := F_k^{-1}(c[j]) \bigoplus c[j-1]$.

Note: achieves CPA as well. Drawbacks: we do need inverse permutation; and inherently *sequential*. Used in TLS1.0 (obsolete). In comparison, RCTR is advantageous.

[2] https://www.rfc-editor.org/rfc/rfc3686.txt. The main change is part of IV is randomly chosen and then fixed for all encryptions under a key. Our description assumes independent IV for each message

1.2 Constructions of Block ciphers

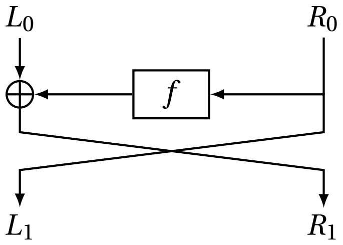
- Theoretical: PRG ⇒ PRF ⇒ PRP (⇒ PRG), i.e., they are all equivalent.
- Practical: below

Case study: DES. The Data Encryption Standard (DES) was developed at IBM in response to a solicitation for proposals from the National Bureau of Standards (NIST National institutes of standards now). Published in 1975, and adopted for "unclassified" applications in 1977.

It consists of 16 rounds of a simple round function.

FS NOTE: Draw DES diagram

- key length: 56 bits. *Key scheduling* derives $k_i, i = 1, \dots, 16$ each of 48-bit long.
- block size: 64 bits.
- Each round: a Feistel permutation, which constructs a permutation out of a function on a smaller domain.



- Round function: 32-bit input first gets expanded to 48-bits and XORed with round key k_i. The outcome goes into a *Substitution-Permutation Network* (SPN) inspired the Shannon's *Confusion-Diffusion* paradigm. Usually called the *DES mangler function*.

  FS NOTE: Draw round function with k_i, S-boxes, ...

Case study: AES.

cipher name	key-size (bits)	block size (bits)	number of rounds
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

Table 1: AES family

FS NOTE: Draw diagram