Lecture 8

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. Hash functions, HMAC, RO.
Today. Theoretical constructions of Private-key primitives.

Tips about Random oracle model

Why are people still so reliant on RO despite of a lot of controversy? Efficiency is a big advantage, and proving security is often easier (or otherwise impossible). **There are several extremely useful properties of RO**..

1. If x has not been queried to $\mathcal{O}$, then the value of $\mathcal{O}(x)$ is uniform.

The next two are specifically relevant to security proofs by *reduction*. Suppose there is an attacker A breaking a scheme constructed in the RO model, then when we use A in a reduction to solve some hard problem or breaking some underlying assumption, we are responsible to answer A's RO queries. This gives us room for something "magic".

2. (Extractability) If $\mathcal{A}$ queries x to $\mathcal{O}$, the reduction can **see this query** and learn x.
3. (Programmability) The reduction can **set** the value of $\mathcal{O}(x)$ to a value of its choice, as long as this value is correctly di stributed, i.e., uniform.

FS NOTE: Draw reduction diagram for illustration

We will see concrete examples taking advantage of these properties. Note that there is no counterpart to extractability or programmability once we instantiate the RO with any concrete function (H is determined, and the evaluation of H will be private to the adversary).

[Today: 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 1 (Informal)**.** *One-way functions are equivalent to the existence of all private-key cryptogrraphy.*

This lecture is one of my favorite topics to cover. But it might be quite abstract and a bit advanced. You should focus on appreciating the high-level ideas and the intelligence elegance.

1 One-way functions

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

Definition 2. 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 $\mathcal{A}$,
$$\Pr[f(x') = f(x) : x' \leftarrow \mathcal{A}(1^n, f(x)), x \leftarrow \{0,1\}^n] \leq \text{negl}(n).$$

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 3. 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 $f(x)$.
- 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.)

1.1 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)? 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.

[What information is exactly hiding by f that keeps us from inverting it?]

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.

Definition 4. A function f is a hard-core predicate of f if it can be computed in poly-time, and for every PPT $\mathcal{A}$,

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

A few remarks.

- $\mathsf{hc}(\cdot)$ 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 $\mathsf{hc}(x)$. **FS NOTE**: Does this mean there is no *universal* hard-core predicate?

The first surprising theorem in our trip, which we will not prove, says that indeed there exists some one-way function for which we can construct a hard-core predicate.

Theorem 5 (Goldreich-Levin Theorem)**.** *Assume one-way functions (permutations) exist, then there exists one-way function (permutation) g and a hard-core predicate hc of g.*

2 PRGs from OWFs

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

Theorem 6 (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 $\ell(n) = n+1$.*

In fact one-way functions are also sufficient to construct a pseudorandom generator, as shown in the famous paper by HILL [1].

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*.

2.1 Computational indistinguishability

We have talked about "indistinguishability" quite a bit so far without a formal treatment. For example, we say the ciphertexts of m_0 and m_1 are hard to distinguish in a computationally secret encrytion, a pseudorandom string (i.e. output from a PRG) is hard to distinguish from a truly random string. Note that "hard" only holds against computationally bounded (i.e. PPT) adversaries. It's time to bring up the formal notion of *computational indist.*. To do so (in an asymptotic approach), we need to talk about *probability ensembles*.

We will only be concerned with probabliity ensembles indexed by natural numbers. For instance consider $\mathcal{X} := \{X_n : n \in \mathbb{N}\}$, where X_n denotes a distribution for each n. $\mathcal{X}$ needs to be *efficiently samplable*[2].

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

[2] There is a PPT algorithm S such that $S(1^n)$ and X_n are identically distributed.

Definition 7. Two probability ensembles $\mathscr{X} = \{X_n : n \in \mathbb{N}\}$ and $\mathscr{Y} = \{Y_n : n \in \mathbb{N}\}$ are computationally indistinguishable, denote $\mathscr{X} \approx_c \mathscr{Y}$, if for every PPt disginguisher D

$$\mathbf{Adv}_D^{\mathscr{X},\mathscr{Y}} := \left| \Pr_{x \leftarrow X_n}[D(1^n, x) = 1] - \Pr_{y \leftarrow Y_n}[D(1^n y) = 1] \right| \leq \mathrm{negl}(n).$$

Remark 8. You may be wondering, is there a notion of *non-computational* indist.? Yes. There is *statiscal* indist., which we denote $\mathscr{X} \approx_s \mathscr{Y}$. We just remove the PPT restriction in definition 7. It is equivalently characterized to a distance mesure *statistical distance*. It is sometimes helpful (but not always correct) to think of $\mathbf{Adv}^{\mathscr{X},\mathscr{Y}}$ as a computational analogue of distance measure of two distributions. If $\mathscr{X} = \mathscr{Y}$, then they are *perfectly* indistinguishable. Do you recall any such examples?

As an example we can restate the condition of a PRG $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ as

$$\{G(U_n)\} \approx_c \{U_{\ell(n)}\},$$

where U_m denotes uniform distribution on $\{0,1\}^m$.

Lemma 9. *$\approx_c$ is "bounded-transitive". Namely if $\mathscr{X} \approx_c \mathscr{Y}$ and $\mathscr{Y} \approx_c \mathscr{Z}$, then $\mathscr{X} \approx_c \mathscr{Z}$. This holds up to any polynomially many applications.*

2.2 PRG Composition: parallel

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

$$G'(s_1, \ldots, s_t) := G(s_1) \| \ldots \| G(s_t).$$

Theorem 10. *G' is a PRG.*

The proof is not difficult, but we will illustrate a common proof technique.

Proof. We want to show $G'(U_{tn}) \approx_c U_{t(n+1)}$. Define the following *hybrid* distributions:

$$\begin{aligned} Z_0 := G'(U_{tn}) &= G(s_1) \| G(s_2) \| \ldots \| G(s_t), \quad s_i \leftarrow U_n; \\ Z_1 &:= G(s_1) \| \ldots \| G(s_{t-1}) \| U_{n+1}; \\ &\vdots \\ Z_{t-1} &:= G(s_1) \| U_{n+1} \| \ldots \| U_{n+1}; \\ Z_t := U_{t(n+1)} &= U_{n+1} \| \ldots \| U_{n+1}. \end{aligned}$$

We claim that $Z_i \approx_c Z_{i+1}$ for all $i = 0, \ldots t-1$, which will imply $Z_0 \approx_c Z_t$.

Suppose for contradiction that $Z_i \not\approx_c Z_{i+1}$. Then we construct D' to distinguish $G(U_n)$ from U_{n+1}.

□

Reflect on hybrid argument.

- Extreme hybrid distributions match the original cases of interest.
- Translate distinguishing consecutive hybrids into breaking some underlying assumption.
- The number of hybrids should be polynomial.

2.3 PRG Composition: sequential (Blum-Micali)

Parallel composition is not very efficient, in particular it does not increase the expansion factor ($\ell'(n)/|s'| = 1 + \frac{1}{n} = \ell(n)/|s|$). Blum-Micali proposed a more efficient strategy.

$$G_1(s) = G(s), G_2(s) = G([G_1(s)]_{1,\ldots,n}) \| [G_1(s)]_{n+1}, \ldots, G'(s) = G_t(s).$$

FS NOTE: Draw diagram

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

Theorem 11 (KL-Thm. 7.20)**.** *If there exists a PRG G with expansion factor $n+1$, then for any polynomial* poly*, there is a PRG G' with expansion factor* poly(n).

3 Pseudorandom functions and permutations

3.1 PRFs from PRGs

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

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

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.

FS NOTE: Draw GGM

3.2 PRPs from PRFs

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

Theorem 12 (KL-Thm. 7.23)**.** *If F is a PRF, then $\Sigma^{(3)}$ is a PRP.*

Moreover 4 and more rounds makes a *strong* PRP. **FS NOTE**: Restate when dealing with CCA However, ≤ 2 rounds are insecure.

4 Putting it together

Theorem 13 (Informal)**.** *Existence of PRG, or computationally secrete encryption, or eu-cma-MAC implies the existance of one-way functions.*

FS NOTE: Draw equivalence diagram