CSC 2426: Fundamentals of Cryptography **Fall 2023**

Lecture 5: Pseudorandom Functions, Secret-key Encryption

Instructor: Akshayaram Srinivasan *Scribe: Srisht Fateh Singh*

Date: 16 October, 2023

In the last lecture, we saw that:

- A PRG extends a string (seed) to a computationally indistinguishable random string.
- If there exists a PRG with a stretch of $1 - bit$, we can construct a PRG with a polynomially large stretch.

In this lecture, we will construct a single-bit stretch PRG using *one-way permutation* functions.

5.1 Single-bit Stretch PRG using One-way Permutation

One-way permutation functions are bijective one-way functions. Let $\{f_n : \{0,1\}^n \to \{0,1\}^n\}_{n\in\mathbb{N}}$ be a one-way permutation. Define $g_n(x,r) = f_n(x)||r$ and let $h_n(x,r) = \langle x, r\rangle$ be its hardcore predicate as defined in Goldreich-Levin Theorem. Define $G_n(x,r) = f_n(x)||r||h_n(x,r)$.

Since $f_n(x)$ is a permutation from $\{0,1\}^n \to \{0,1\}^n$, this implies $f_n(x)$ forms a uniform distribution for uniformly sampled $x \leftarrow \{0,1\}^n$. Therefore, $f_n(x)||r$ is a uniformly distributed bit-string of size $2n$. Now, we will prove that $G_n(x,r)$ is a PRG.

Proof: If $G_n(x,r)$ is not a PRG, $\exists$ n.u.PPT $\mathcal{A}$ such that

$$\left| \Pr_{x,r\leftarrow\{0,1\}^n}[\mathcal{A}(1^{2n}, G_n(x,r)) = 1] - \Pr_{y\leftarrow\{0,1\}^{2n+1}}[\mathcal{A}(1^{2n}, y) = 1] \right| = \mu(n)$$

for some non-negligible function $\mu(n)$.

The first $2n - bits$ of $G_n(x,r)$ are completely random. Therefore, the $\Pr[\mathcal{A}(y) = 1]$ can be conditioned on the last bit as follows:

$$\Pr[\mathcal{A}(y) = 1] = \Pr[y_{2n+1} = h_n(x,r)] \Pr[\mathcal{A}(f_n(x)||r||h_n(x,r)) = 1] + \Pr[y_{2n+1} = \overline{h_n}(x,r)] \Pr[\mathcal{A}(f_n(x)||r||\overline{h_n}(x,r)) = 1]$$

$$= 0.5 \Pr[\mathcal{A}(f_n(x)||r||h_n(x,r)) = 1] + 0.5 \Pr[\mathcal{A}(f_n(x)||r||\overline{h_n}(x,r)) = 1]$$

Here, $\Pr[y_{2n+1} = h_n(x,r)] = \Pr[y_{2n+1} = \overline{h_n}(x,r)] = 0.5$ since we are assigning a deterministic value to a uniform random bit. Therefore, the difference between distinguisher probabilities is

$$0.5 \left| \Pr[\mathcal{A}(f_n(x)||r||h_n(x,r)) = 1] - \Pr[\mathcal{A}(f_n(x)||r||\overline{h_n}(x,r)) = 1] \right| = \mu(n)$$

Let's assume w.l.o.g. $\Pr[\mathcal{A}(f_n(x)||r||h_n(x,r)) = 1] - \Pr[\mathcal{A}(f_n(x)||r||\overline{h_n}(x,r)) = 1] > 0$. Intuitively, $\mathcal{A}$ succeeds slightly higher when the last bit is h_n compared to $\overline{h_n}$. Therefore, we define the predictor

$\mathcal{B}(f_n(x)||r) = h_n(x,r)$ as follows: pick $b \leftarrow \{0,1\}$ and evaluate $\mathcal{A}(f_n(x)||r||b)$. If the value is 1, output b, else output $\overline{b}$. Therefore

$$\Pr[\mathcal{B}(f_n(x)||r) = h_n(x,r)] = \Pr[b = h_n]\Pr[\mathcal{A}(f_n(x)||r||h_n(x,r)) = 1] + \Pr[b = \overline{h}_n]\Pr[\mathcal{A}(f_n(x)||r||\overline{h}_n(x,r)) = 0]$$

$$= 0.5\Pr[\mathcal{A}(f_n(x)||r||h_n(x,r)) = 1] + 0.5(1 - \Pr[\mathcal{A}(f_n(x)||r||\overline{h}_n(x,r)) = 1])$$

$$= 0.5 + \mu(n)$$

which is contradicts the unpredictability of h_n.

[HILL99] showed that it is possible to construct a PRG from any one-way function.

5.2 Pseudorandom Functions

Let $\mathbb{F}_n = \{f : \{0,1\}^n \to \{0,1\}^n\}$ represent all functions from $\{0,1\}^n \to \{0,1\}^n$. One can note that there exists a total of $(2^n)^{2^n}$ functions. If we want to represent all of these functions, we require a total of $log_2((2^n)^{2^n})$ bits or $n \cdot 2^n$ bits which are exponential in n.

Instead, we shall try to construct a function that seems indistinguishable to a computationally bounded attacker. We use 2 algorithms for this:

$$\mathsf{Setup}(1^n) \to \text{Key } k \text{ (efficient and probabilistic)}$$

$$\mathsf{Eval}(k, x \in \{0,1\}^n) = y \in \{0,1\}^n \text{ (Deterministic)}$$

such that for all n.u.PPT $\mathcal{A}$ and $f \leftarrow \mathbb{F}_n$

$$\left| \Pr_{k \leftarrow \mathsf{Setup}(1^n)}[\mathcal{A}^{\mathsf{Eval}(k,\cdot)} = 1] - \Pr[\mathcal{A}^{f(\cdot)} = 1] \right| \leq \mathsf{negl}(n)$$

The above inequality implies that if an attacker is given oracle access to $\mathsf{Eval}(k,\cdot)$ and $f(\cdot)$ to query on a polynomial number of inputs, it cannot distinguish our constructed random function from a randomly sampled function.

The following theorem gives a way to construct a P.R.F from a P.R.G.

Theorem (Goldreich-Goldwasser-Micali): Given a length-doubling P.R.G $G_n : \{0,1\}^n \to \{0,1\}^{2n}$

$$\mathsf{Setup}(1^n) : k \leftarrow \{0,1\}^n$$

$$\mathsf{Eval}(k, x \in \{0,1\}^n) \text{ is constructed as follows:}$$

Let $x = [x_1, x_2 \ldots x_n]$ be the bit representation of a query of an attacker. Then

$i_0 \leftarrow k$
for r from 0 to $n-1$ **do**
  $o_{r,left}||o_{r,right} \leftarrow G_n(i_r)$
  if $x_r == 0$ **then**
    $i_{r+1} \leftarrow o_{r,left}$
  else
    $i_{r+1} \leftarrow o_{r,right}$
  end if
end for

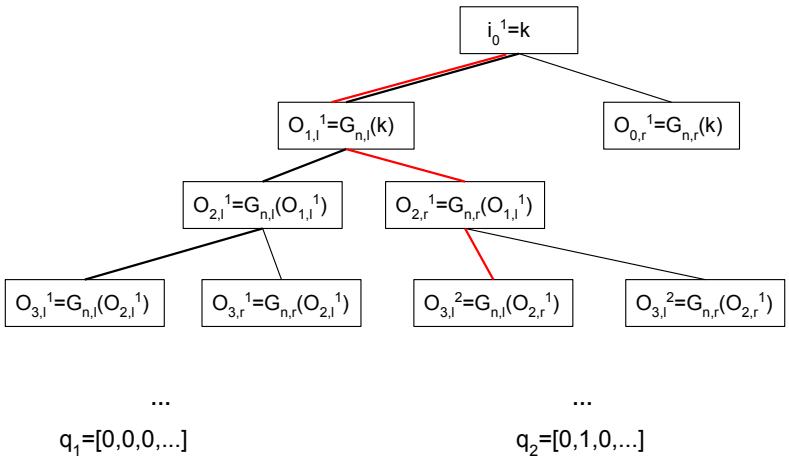

Figure 5.1: An example consisting of two queries.

After n steps, i_n is the output of our pseudorandom function.

Proof: We need to prove that the final generated output i_n is pseudorandom. One approach is to build a binary tree with 2^n inputs. At each level r, switch i_r's (which are 2^r in count) from pseudorandom to random and proceed to the next level until we cover all leaf nodes. However, this approach does not work as we end up using the hybrid argument for all leaf nodes which are 2^n and thus not polynomial in n.

Instead, we use the fact that the attacker makes a polynomial number of queries and build a random looking tree on-the-fly. We use the following algorithm: Let $q_1, q_2, \ldots, q_n$ be the queries of the attacker.

- For q_1, switch $o_{r,left}$ and $o_{r,right}$ $\forall r \in [0, n-1]$ to random while traversing along the path from root to the leaf labeled q_1. Specifically, we switch the two children of the root to random. This change is indistinguishable to attacker from the security of PRG. This creates two sub-tress that are rooted at the left and the right child of the root. We continue to answer the rest of queries as in the real algorithm. In the next sub-hybrid, we choose either the left sub-tree or the right sub-tree depending on the first bit of q_1 and repeat the above step by replacing both its children with random strings and creating two more sub-tress. At each step, the we create one more sub-tree and at the end, we have a forest of n trees. In each step, we continue to answer the rest of queries as in the real algorithm.

- For q_j $j \in [2, n]$, we repeat the process we did for the first query. However, if at any level r, the value of o_r was computed during previous queries, we reuse them and do not create new sub-tress. Only when we find a new child that has not yet been switched to random, do we break into sub-trees. This ensures that our pseudorandom function gives the same output if queried with the same input multiple times.

The above reduction invokes the security of PRG $\mathsf{poly}(n)$ times. Hence, to the attacker, our function seems indistinguishable from a randomly sampled function.

Figure 5.1 shows an example with two queries of the form $q_1 = [0, 0, 0 \ldots], q_2 = [0, 1, 0, \ldots]$ which share the same first bit. Since, $o_{1,l}$ and $o_{2,r}$ were computed during the first query (shown in bold black path), these values are reused for q_2 (shown in bold red path) as well.

5.3 Secret-key Encryption

Suppose Alice and Bob know a secret k beforehand. Now, they want to communicate (send message m) over a public channel such that nobody else can eavesdrop on their conversation. Formally, we want to create an encryption scheme *Enc* and a corresponding decryption scheme *Dec* such that:

$$\mathsf{KeyGen}(1^n) \to \text{Key } K$$

$$\mathsf{Enc}(k, m) \to \text{Ciphertext } c$$

$$\mathsf{Dec}(k, c) \to m$$

For all messages m, and $k \leftarrow \mathsf{KeyGen}(1^n)$, SKE should always conform to the following correctness constraints:

$$c \leftarrow \mathsf{Enc}(k, m)$$

$$\mathsf{Dec}(k, c) \to m$$

SKE is known as single-message CPA secure if for $k \leftarrow \mathsf{KeyGen}(1^n)$, $c \leftarrow \mathsf{Enc}(k, m)$, and $\forall$ messages m_0, m_1:

$$\{k \leftarrow \mathsf{KeyGen}(1^n) : \mathsf{Enc}(k, m_0)\} =_c \{k \leftarrow \mathsf{KeyGen}(1^n) : \mathsf{Enc}(k, m_1)\}$$

SKE is known as multi-message CPA secure if for all q pair of messages $(m_{0,1}, m_{1,1}), \ldots, (m_{0,q}, m_{1,q})$

$$\{k \leftarrow \mathsf{KeyGen}(1^n) : \mathsf{Enc}(k, m_{0,1}), \ldots, \mathsf{Enc}(k, m_{0,q})\} =_c \{k \leftarrow \mathsf{KeyGen}(1^n) : \mathsf{Enc}(k, m_{1,1}), \ldots, \mathsf{Enc}(k, m_{1,q})\}$$

Note that multi-message CPA security requires Enc to be randomized. If Enc is a deterministic function, then an attacker with the knowledge of $\mathsf{Enc}(k, m_0)$ and $\mathsf{Enc}(k, m_1)$ (which can be obtained by settings $m_{0,1} = m_{1,1} = m_0$ and $m_{0,2} = m_{1,2} = m_1$) can then trivially distinguish $(\mathsf{Enc}(k, m_0), \mathsf{Enc}(k, m_1))$ thus breaking the multi-message CPA security.

References

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.