6.875J/18.425J Cryptography and Cryptanalysis Feb 27, 2017

Lecture 5: Pseudorandom Generators

Instructor: Vinod Vaikuntanathan *Scribes: Srinivasan Raghuraman*

Acknowledgements: These lecture notes are based on scribes by Mikhail Rudoy, Yinzhan Xu, Quntao Zhuang, Bingfei Cao, Bristy Sikder and Sagnik Saha from 2016 (lecturer: Shafi Goldwasser).

As we have seen, randomness is crucial towards the construction of cryptographic primitives. What are the sources of randomness? Hardware is one of them – transistor noise, heat, etc. Stock market fluctuations are fairly unpredictable as well. However, one of these are true sources of randomness. We realize that true randomness is an expensive commodity. Since we need a lot of randomness, we would like to be able to take a few truly random bits and stretch them into a large stream of bits that are sufficiently/seemingly random. This brings us to the topic of this lecture – pseudorandom generators (PRGs). We will first wrap up the proof of the Goldreich-Levin Theorem and move on to PRGs.

1 Finishing Goldreich-Levin

Let us recap and complete the proof of the Goldreich-Levin Theorem, which attempts to construct a hard core predicate for every one way function. We review the definition of a hard core predicate. The intuition behind hard core predicates is that given $y = f(x)$, a hard core predicate $b(x)$ is hard to calculate, in other words, information obtained from $f(x)$ is not helpful to calculate $b(x)$. Formally, $b : \{0,1\}^n \to \{0,1\}$ is a *hard-core predicate* for $f : \{0,1\}^n \to \{0,1\}^n$ if $\forall$ PPT algorithms $\mathcal{A}$,

$$\Pr[x \xleftarrow{\$} \{0,1\}^n : \mathcal{A}(f(x)) = b(x)] = \frac{1}{2} + \text{negl}(n) \tag{1}$$

Let us recall the Goldreich-Levin Theorem. Let f be a length-preserving one way function. Define

$$g(x,r) = f(x)||r$$

where $|x| = |r|$. Let

$$b(x,r) = \langle x, r\rangle = \sum_i x_i r_i \bmod 2$$

The theorem states that b is a hard-core predicate for g.

The proof is by contradiction. Assume there exists a PPT algorithm PRED such that

$$\Pr[x, r \xleftarrow{\$} \{0,1\}^n : \mathsf{PRED}(g(x,r)) = b(x,r)] \geq \frac{1}{2} + \epsilon$$

where $|x| = |r| = n$ and ϵ is a non-negligible function of n. We show that we can then construct a PPT algorithm $\mathcal{B}$ which inverts f and hence contradicts the fact that it is one way.

We began by showing that a non-negligible fraction $(\frac{\epsilon}{2})$ of the possible xs are "good" xs for which

$$\Pr[r \xleftarrow{\$} \{0,1\}^n : \mathsf{PRED}(g(x,r)) = b(x,r)] \geq \frac{1}{2} + \frac{\epsilon}{2}$$

Our goal is to design an algorithm which succeeds in inverting $g(x,r)$ with non-negligible probability under the assumption that x was good. Since a non-negligible fraction of xs are good, running the same algorithm on a completely random x will still result in a non-negligible probability of success.

The main strategy for this is as follows. Define e_i to be the string of length n such that only the i-th bit is 1 and the other bits are all 0. By definition, $\langle x, e_i\rangle = x_i$ is the ith bit of x. Since PRED retrieves inner products, we can use it to determine bits of x by asking it to compute the inner products of x and e_i for each $1 \leq i \leq n$. Note that e_i is by no means random.

If $\epsilon = 1$, PRED would in fact work correctly on all r and hence this strategy works. If $\epsilon\langle 1$, we have to deal with the issue that e_i is not random. We exploit linearity to randomize e_i. Because of the distributive property of the dot product,

$$\langle x, r \rangle = \langle x, z \rangle \oplus \langle x, r \oplus z \rangle$$

for any $z \in \{0,1\}^n$. We can hence pick r_i uniformly from $\{0,1\}^n$, and let

$$x'_i = \mathsf{PRED}(g(x, r_i)) \oplus \mathsf{PRED}(g(x, r_i \oplus e_i))$$

to compute the ith bit of x.

In order that $x'_i = x_i$, we need that both queries to PRED are answered either correctly or incorrectly. It is sufficient that both answers are correct and we analyze that case. We have

$$\begin{aligned}
\Pr[r_i \xleftarrow{\$} \{0,1\}^n : x'_i \neq x_i] &\leq \Pr[r_i \xleftarrow{\$} \{0,1\}^n : \mathsf{PRED}(g(x, r_i)) \neq \langle x, r_i \rangle \vee \mathsf{PRED}(g(x, r_i \oplus e_i)) \neq \langle x, r_i \oplus e_i \rangle] \\
&\leq 2\left(\frac{1}{2} - \frac{\epsilon}{2}\right) \\
&= 1 - \epsilon
\end{aligned}$$

Thus, the probability that $x'_i = x_i$ is ϵ. If $\epsilon > \frac{1}{2} + \frac{1}{p(n)}$ for some polynomial p, we can amplify this bias we have obtained in order to extract x_i with higher confidence. The idea is to repeat the procedure many $r'_i s$ for each i and take the majority for each i. We denote these as $r_{i,j}$ where $1 \leq i \leq n$ and $1 \leq j \leq m$, and let

$$x'_{i,j} = \mathsf{PRED}(g(x, r_{i,j})) \oplus \mathsf{PRED}(g(x, r_{i,j} \oplus e_i))$$

The question remains, what is m? We obtain this from the Chernoff bound. Let $E_{i,j}$ be the indicator variable for the event that $x'_{i,j} = x_i$ and let E_i be the indicator variable for the event that $\text{maj}\left(\{x_{i,j}\}_{j=1,\ldots m}\right) = x_i$. We have

$$\Pr[r_{i,j} \xleftarrow{\$} \{0,1\}^n : E_{i,j} = 1] \geq \epsilon$$

Applying the Chernoff bound,

$$\begin{aligned}
\Pr[r_{i,1}, \ldots, r_{i,m} \xleftarrow{\$} \{0,1\}^n : E_i = 0] &= \Pr\left[r_{i,1}, \ldots, r_{i,m} \xleftarrow{\$} \{0,1\}^n : \sum_{j=1}^{m} E_{i,j} < \frac{m}{2}\right] \\
&\leq \mathcal{O}\left(e^{-m\left(\epsilon - \frac{1}{2}\right)^2}\right)
\end{aligned}$$

Setting it to be less than δ, we get that our algorithm succeeds with probability $\geq 1 - \delta$ for a given i as long as

$$m \geq \mathcal{O}\left(\frac{1}{\left(\epsilon - \frac{1}{2}\right)^2} \log\left(\frac{1}{\delta}\right)\right)$$

In order that the algorithm succeeds for every i with non-negligible probability, applying a union bound, it is sufficient that $n\delta$ be non-negligible, which can be achieved by setting $\delta = \frac{1}{n^2}$. For this choice of δ, note that the running time of the algorithm is polynomial in n.

We now come to the final case of arbitrary $0 < \epsilon \leq 1$. Note in the previous case, when $\epsilon\rangle\frac{1}{2}$, for any j, the probability that $E_{i,j}$ succeeded was greater than $\frac{1}{2}$ and hence we could amplify this success probability by repeating for many j. This however provides no advantage if $\epsilon \leq \frac{1}{2}$, since that would mean that we would do at least as well by simply guessing x_i.

Suppose we are somehow given a certain number of $r_{i,j}, \langle x, r_{i,j} \rangle$ values, where the $r_{i,j}$s are random. Then, to compute the ith bit of x, it would be sufficient to compute $\langle x, r_{i,j} \oplus e_i \rangle$. This we can do whenever a computation of $\mathsf{PRED}(g(x, r_{i,j} \oplus e_i))$ succeeds, which occurs with probability at least $\frac{1}{2} + \frac{\epsilon}{2}$. Since this probability is strictly greater than $\frac{1}{2}$, for any given i, we can amplify the probability of success as before.

The problem now reduces to computing a large number of $r_{i,j}, \langle x, r_{i,j} \rangle$ values for random $r_{i,j}$, which are correct. We first note that the $r_{i,j}$s need not be random, rather, is it sufficient that they are pairwise independent. In this case, we use Chebyshev's inequality. Let $E'_{i,j}$ be the indicator variable for the event that $x''_{i,j} = x_i$ where

$$x''_{i,j} = \langle x, r_{i,j} \rangle \oplus \mathsf{PRED}(g(x, r_{i,j} \oplus e_i))$$

and let E'_i be the indicator variable for the event that $\text{maj}\left(\{x''_{i,j}\}_{j=1,\dots m'}\right) = x_i$. We have

$$\Pr[r_{i,j} \leftarrow \{0,1\}^n : E'_{i,j} = 1] \geq \frac{1}{2} + \frac{\epsilon}{2}$$

where the $r_{i,j}$ are pairwise independent. Applying Chebyshev's inequality,

$$\begin{aligned}\Pr[r_{i,1}, \dots, r_{i,m'} \leftarrow \{0,1\}^n : E'_i = 0] &= \Pr\left[r_{i,1}, \dots, r_{i,m'} \leftarrow \{0,1\}^n : \sum_{j=1}^{m'} E'_{i,j} < \frac{m'}{2}\right] \\ &\leq \mathcal{O}\left(\frac{1}{\epsilon^2 m'}\right)\end{aligned}$$

Setting it to be less than δ, we get that our algorithm succeeds with probability $\geq 1 - \delta$ for a given i as long as

$$m' \geq \mathcal{O}\left(\frac{1}{\epsilon^2 \delta}\right) \tag{2}$$

In order that the algorithm succeeds for every i with non-negligible probability, applying a union bound, it is sufficient that $n\delta$ be non-negligible, which can be achieved by setting $\delta = \frac{1}{n^2}$. For this choice of δ, note that the running time of the algorithm is polynomial in n.

Now, we need to devise a method to compute $r_{i,j}, \langle x, r_{i,j} \rangle$ for $1 \leq i \leq n$, $1 \leq j \leq m'$, where $r_{i,j}$ and $r_{i,j'}$ are pairwise independent for every i and $j \neq j'$. Can we guess them? No, there are too many of them and we will be correct with probability at most $2^{-m'}$ which is negligible in n. We proceed as follows. We choose $\log(m'+1) = \mathcal{O}(\log n)$ random n-bit strings $s_1, \dots, s_{\log(m'+1)}$ independently and uniformly at random. Then for every non-empty subset S of $\{1, ..., \log(m'+1)\}$, define

$$r_S = \bigoplus_{i \in S} s_i \tag{3}$$

Clearly, each r_S is uniformly distributed over the set of n-bit strings. Furthermore, for $S \neq S'$, we have that

$$r_S \oplus r_{S'} = r_T$$

is also uniformly distributed, where $T = S \triangle S' \neq \emptyset$. We conclude that the r_Ss are pairwise independent. Since there are $2^{\log(m'+1)} - 1 = m'$ non-empty subsets of $\{s_1, ..., s_{\log(m'+1)}\}$, we have produced m' different r_Ss; in other words, we have produced m' pairwise independent rs which can be used as $r_{i,j}$s for some i.

We now wish to compute $\langle x, r_S \rangle$ for each r_S. But

$$\langle x, r_S \rangle = \left\langle x, \bigoplus_{i \in S} s_i \right\rangle = \bigoplus_{i \in S} \langle x, s_i \rangle \tag{4}$$

Hence, in order to compute $\langle x, r_S \rangle$ for every S, it would be sufficient to compute $\langle x, s_i \rangle$ for every i. Let $v_i = \langle x, s_i \rangle$. We see that if the correct values of $\{v_i\}$ are known, then we can compute $\langle x, r_S \rangle$ correctly for every r_S. But v_is can take on only $2^{\log(m'+1)} = \text{poly}(n)$ different assignments of values. So, we can simply run the whole algorithm from above once for every possible assignment of $\{v_i\}$, yielding $(m'+1)$ guesses for

x. One of the runs will use the correct assignment of values to $\{v_i\}$, and in that run, there is a non-negligible probability that we recover x. Hence, there is a non-negligible chance that at least one of the $(m'+1)$ guesses is correct. Since for any guess x', we can easily check whether $f(x') = f(x)$, if any of the guesses is correct, we can identify it and output it. Therefore, the algorithm we have designed inverts f with non-negligible probability. The complete algorithm is presented below.

Algorithm GL-Invert$(y = f(x))$

```
1. Compute $m'$ as in Equation 2
2. Choose $s_1, \ldots, s_{\log(m'+1)} \xleftarrow{\$} \{0,1\}^n$ independently and uniformly at random.
3. For each of the $2^{\log(m'+1)}$ boolean assignments to $v_i = \langle x, s_i \rangle$
   (a) For every non-empty subset $S$ of $\{1, ..., \log(m'+1)\}$, compute $r_S$ and $\langle x, r_S \rangle$ as in Equations 3 and 4.
   (b) For every $1 \le i \le n$
       i. For every non-empty subset $S$ of $\{1, ..., \log(m'+1)\}$, compute $x''_{i,S} = \langle x, r_S \rangle \oplus \mathsf{PRED}(g(x, r_S \oplus e_i))$.
       ii. Compute $x'_i = \text{maj}\left(\{x''_{i,S}\}_{\phi \subset S \subseteq \{1,\ldots,\log(m'+1)\}}\right)$.
   (c) Let $x' = x'_n \ldots x'_1$. Check if $f(x') = y$. If so, return $x'$. Else, continue.
```

Remark. It is possible to view Goldreich-Levin as a list-decoding algorithm. The inner products of x with all n-bit strings can be thought of as a highly redundant codeword for x (this is known as the "Hadamard" code). The predictor PRED simply provides access to a noisy codeword and the reduction we designed uses access to the noisy codeword in order to do local or list decoding of the codeword.

2 Pseudorandom Generators

As discussed before, we want to design algorithms, called PRGs, that take a few truly random bits and stretch them into a large stream of bits that are sufficiently/seemingly random. Here are a few ways in which one could define a PRG.

Definition 1. *A polynomial time deterministic algorithm G, where $G : \{0,1\}^n \to \{0,1\}^m$ is a* cryptographically strong pseudorandom number generator*, if*

1. $m > n$
2. *For any PPT algorithm D,*

$$\left|\Pr[x \xleftarrow{\$} \mathcal{U}_n : D(G(x)) = 1] - \Pr[y \xleftarrow{\$} \mathcal{U}_m : D(y) = 1]\right| = \text{negl}(n)$$

The next definition asks for something seemingly weaker, which we call **next-bit tests**. It says that given any prefix of the output of a PRG, the next bit is hard to predict.

Definition 2. *A polynomial time deterministic algorithm G, where $G : \{0,1\}^n \to \{0,1\}^m$ is an* unpredictable pseudorandom number generator*, if for all $1 \le i \le m$ and all PPT algorithms $\mathcal{A}$,*

$$\Pr[y \xleftarrow{\$} G(\mathcal{U}_n) : \mathcal{A}(y_1, \ldots, y_{i-1}) = y_i] = \frac{1}{2} + \text{negl}(n)$$

The last definition one could consider would be that no PPT algorithm can compress the output of a PRG. One could then ask which of these is the right definition. Fortunately, it turns out that all these definitions are equivalent.

2.1 Equivalence of definitions

It turns out that although next-bit tests are much more constrained in their mode of operation than distinguishing the entire output of the PRG from a random string, both the tests have the same power when it comes to recognizing random bit sequences.

Theorem 3. *A pseudo-random number generator G is cryptographically secure if and only if it is unpredictable.*

Proof. $\Longrightarrow$: Suppose that G is not unpredictable. Therefore there exists a predictor $\mathcal{A}$ and $i \leq m$ such that

$$\mathcal{P}[y \xleftarrow{\$} G(\mathcal{U}_n) : \mathcal{A}(y_1, y_2, \ldots, y_{i-1}) = y_i] > \frac{1}{2} + \frac{1}{Q(n)}$$

where $\mathcal{U}_n$ denotes the uniform distribution over n-bit strings and Q is a polynomial. Then consider the distinguisher D which on input y invokes $\mathcal{A}(y_1, y_2, \ldots, y_{i-1})$ and outputs 1 if and only if the result matches y_i. If y is a random bit sequence, then this probability is exactly half (since y_i would then be 1 or 0 with equal probability.). However, if y was generated by G, then by our assumption, $\mathcal{A}$ guesses the ith bit successfully with probability at least $\frac{1}{2} + \frac{1}{Q(n)}$, and hence D outputs 1 with at least that probability. Since the probabilities corresponding to the 2 different input distributions differ by a noticeable fraction, G is not cryptographically secure.

$\Longleftarrow$: For the 'if' direction, we show that if there exists a distinguisher D that distinguishes between $\mathcal{U}_m$ and $G(\mathcal{U}_n)$ with probability greater than ϵ, then we can create a predictor that can predict some bit of the output of G with probability larger than $\frac{1}{2} + \frac{\epsilon}{m}$. If ϵ is greater than some polynomial fraction, then so is $\frac{\epsilon}{m}$, which completes the proof.

The proof strategy is what is known as a **hybrid argument**. Let us define $m+1$ hybrid distributions as follows:-

- D_0 is the uniform distribution on m bits, i.e., $\mathcal{U}_m$
- D_i is defined as the distribution obtained by choosing the first i bits from $G(\mathcal{U}_n)$ and the rest $m-i$ bits uniformly and independently at random.
- D_m is the distribution obtained by applying G on a random n-bit string, i.e., $G(\mathcal{U}_n)$.

Intuitively, these hybrid distributions gradually transition from $\mathcal{U}_m$ to $G(\mathcal{U}_n)$, and hence if A can distinguish between the two extreme distributions, it can also distinguish between at least one adjacent pair of hybrids. To formalize this notion, we define $p_i = \mathcal{P}[y \xleftarrow{\$} D_i : D(y) = 1]$. We have

$$|p_m - p_0| > \epsilon$$

This implies that

$$\sum_{i=1}^{m} |p_i - p_{i-1}| > \epsilon$$

An averaging argument tells us that there exists an i such that

$$|p_i - p_{i-1}| > \frac{\epsilon}{m}$$

Without loss of generality, assume that

$$p_i - p_{i-1} > \frac{\epsilon}{m}$$

We now use D to build a predictor $\mathcal{A}$ for the ith bit of the output of G. On input $(y_1, y_2, \ldots, y_{i-1})$, invoke D on $(y_1 y_2 \ldots y_{i-1} c r_1 r_2 \ldots r_{m-i})$ where c and all the r_j are randomly and independently chosen. Finally, output c if A outputs 1 and $\overline{c}$ otherwise. The intuition for the construction of $\mathcal{A}$ is as follows. Notice that $p_m - p_0 > \epsilon$. Hence, D tends to output 1 more often when it is fed a pseudorandom input. We would hence like to identify D's output of 1 with the world of pseudorandom strings. Similarly, since $p_i > p_{i-1}$, D outputs 1 more often when it is fed the input with the ith bit being the ith bit in the output of the PRG. Hence, if D outputs 1, we would like to output c itself as the ith bit of the output of the PRG. However, if D outputs 0, it was because the bit that it was sent was the wrong bit, and hence we flip c. We formalize this below.

Let us analyze the success probability of this predictor. For convenience, let us define p'_i to be the probability that D outputs 1 when the input is taken from D'_i, which is the same as D_i with its ith bit flipped. Since the ith bit of D_{i-1} is uniformly chosen, it's easy to see that

$$p_{i-1} = \frac{p_i + p'_i}{2}$$

We have

$$\begin{aligned}
&\mathcal{P}[s \xleftarrow{\$} G(\mathcal{U}_n) : \mathcal{A}(s_1, s_2, \ldots, s_{i-1}) = s_i] \\
&\qquad = \mathcal{P}[s \xleftarrow{\$} G(\mathcal{U}_n), c \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \mathcal{U}_{m-i} : D(s_1, s_2, \ldots, s_{i-1} \| c \| r) = 1 \wedge c = s_i] \\
&\qquad\quad + \mathcal{P}[s \xleftarrow{\$} G(\mathcal{U}_n), c \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \mathcal{U}_{m-i} : D(s_1, s_2, \ldots, s_{i-1} \| c \| r) = 0 \wedge c = \overline{s_i}] \\
&\qquad = \mathcal{P}[x \xleftarrow{\$} D_i : D(x) = 1 \wedge r_1 = s_i] + \mathcal{P}[x \xleftarrow{\$} D'_i : D(x) = 0 \wedge r_1 = \overline{s_i}] \\
&\qquad = \frac{p_i}{2} + \frac{1 - p'_i}{2} \\
&\qquad > \frac{1}{2} + \frac{\epsilon}{m}
\end{aligned}$$

□

In the next lecture, we are going to see how to construct PRGs from OWPs.

References

[1] Goldreich, Oded, and Leonid A. Levin. "A hard-core predicate for all one-way functions." Proceedings of the twenty-first annual ACM symposium on Theory of computing. ACM, 1989.