CS 6160 Cryptology Lecture 5: Pseudorandom Generators

Maria Francis

September 11, 2020

Generating Randomness

- As we saw from the lecture on Perfect Secrecy, generating a truly random key is key to developing a secure encryption scheme.
- Throughout this course (and most discussions in papers) it is simply assumed that we have an access to unlimited supply of independent, unbiased random bits.
- How easy is it to generate it in practice? Generate by flipping a coin? But it does not scale!
- Modern random-generators start off with a pool of high-entropy/unpredictable data (hard-disk access times, keystrokes, share prices, etc) and then use that to get nearly independent and unbiased bits of required length!

Generating Randomness in Crypto

- In crypto applications, we like to store a small random key and deterministically expand to a pseudo-random sequence.
- For cryptographic security like that of a OTP, the pseudo-random sequence generator must have the property that an adversary who has seen a portion of the generator's output y must be unable to predict unseen bits of y.
- I.e. the adversary should not be able to efficiently infer the seed from some bits of the output.
- Classical techniques which are effective for Monte Carlo simulations are typically not suitable for crypto. for example: linear feedback shift registers are insecure as one can solve the feedback pattern given a small number of output bits.

Pseudorandom Sequence

Definition (poly-time indistinguishable)

Let X_n, Y_n be prob. distributions on $\{0,1\}^n$. We say that $\{X_n\}$ is poly-time indistinguishable from $\{Y_n\}$, if

$\forall PTM\ A, \forall q(x) \in \mathbb{K}[x], \exists n_0$ s.t. $\forall n > n_0$,

$$|Pr_{t\in X_n}[A(t)=1] - Pr_{t\in Y_n}[A(t)=1]| < \frac{1}{q(n)} (\equiv \text{ negl } (n))$$

i.e. for sufficiently long strings no probabilistic TM can tell whether the string was sampled according to X_n or Y_n.

Pseudorandom Sequence

Definition (pseudorandom)

We say that X_n is pseudorandom if it is poly-time indistinguishable from the uniform probability distribution on $\{0,1\}^n$, U_n.

$\forall PTM\ A, \forall q(x) \in \mathbb{K}[x], \exists n_0$ s.t. $\forall n > n_0$,

$$|Pr_{t \in X_n}[A(t) = 1] - Pr_{t \in U_n}[A(t) = 1]| < \frac{1}{q(n)} (\equiv \text{ negl } (n))$$

In U_n, every element in $\{0,1\}^n$ has the same likelihood, $\frac{1}{2^n}$.

Pseudorandom generators (PRG)

- A PRG is an efficient, deterministic algorithm for transforming a short, uniform string, *seed* into a longer, "uniform-looking" (pseudorandom) output string.
- Input: small amount of true randomness, Output: Large amount of pseudorandomness
- Large number of true random bits – very difficult and slow.
- 1940*s* concept but cryptographical approach came in 1980s.
- For any efficient distinguisher D, D should distinguish the outputs of a PRG and a uniform string with only negligible. probability.
- It is the distribution of strings that are pseudorandom or random not a fixed string as such.

Pseudorandom generators (PRG)

Let $\ell \in F[x]$ and $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ for any $k > 0$ be a deterministic poly-time algorithm called a pseudorandom generator if the following conditions hold:

1. (Expansion:) For every k, it holds that $\ell(k) > k$.
2. (Pseudorandomness:) There exists no PPT distinguisher D that can distinguish $G(x)$ from a truly random string $\mathcal{R} \in \{0,1\}^{\ell(k)}$. I.e., if we let 1 be pseudorandom and 0 be random,

$$
\begin{aligned}
&|Pr[D(G(x)) = 1 : x \leftarrow^R \{0,1\}^k] \\
&- Pr[D(\mathcal{R}) = 0 : \mathcal{R} \leftarrow^R \{0,1\}^{\ell(k)}]| \\
&< \operatorname{negl}(k).
\end{aligned}
$$

$\ell(k)$ is the expansion factor of G

NOT a pseudorandom generator

- Let $G(x) = x \circ \oplus_{i=1}^{n} x_i$, $\ell(n) = n + 1$.
- But the output can be distinguished easily from uniform: let D on input a string w output 1 iff the final bit of w is equal to XOR of all the preceding bits of w.
- All outputs of G will return 1.
- But if w is uniform then the $Pr[D(w) = 1] = 1/2$, $\Rightarrow$ we have a constant, not negligible difference $|1/2 - 1|$ and G is not a pseudorandom generator!
- *D need not be always correct because sometimes 1 is output for the uniform string, but we are looking for whether D is a good and efficient distinguisher!*

How good is a PRG?

- Distribution of a PRG, G is far from uniform!
- Assume $\ell(n) = 2n$, i.e. G doubles the length of input.
- In a uniform distribution $\{0, 1\}^{2n}$, each of 2^{2n} possible strings with probability exactly 2^{-2n}.
- Distribution of G's output with a uniform seed of n length, number of different strings in the range of G is at most 2^n.
- So the fraction of strings of length $2n$ that are in the range of G is at most $2^n/2^{2n} = 2^{-n}$, $\Rightarrow$ vast majority of strings of length $2n$ do not occur as outputs of G.
- Given unlimited amount of time/exponential time distinguisher we can always distinguish a random string and a pseudorandom string with a non-negligible probability using a brute-force attack.

Seed and its length

- The seed is like a key, should be chosen uniformly and kept secret.
- Seed must be long enough to make sure that brute force attacks are not feasible. Set the length equal to security parameter.
- Do PRGs exist? Well we will see that if one-way functions exist, we can construct PRGs.

Pseudorandom generators from OWP

- We prove the existence of PRGs if one way permutations exist.
- We show how a hard-core predicate of a one-way permutation can be used to construct a pseudorandom generator. In fact it suffices to have a one-way function but the proof is complicated.

Theorem

Let f be a one-way permutation and let h be a hard-core predicate of f. Then, $G(x) := f(x) \circ h(x)$ is a pseudorandom generator with expansion factor $\ell = n + 1$, where n is $|x| (= |f(x)|)$.

- $h(x)$ looks random given $f(x)$ when x is random.
- f is a permutation, so $f(x)$ is uniformly distributed.
- $f(x) \circ h(x)$ is a uniform n-bit string plus one bit that looks uniform *it is pseudorandom.*

Proof

- Let $G^1(x) = f(x) \circ h(x)$. $G(x)$ is computed poly-time and if $|x| = n$, $|G(x)| = n + 1$.
- T.S.T. the distribution G^1_{n+1} is pseudorandom.
- We prove by contradiction. Note that since f is a permutation, $f(U_n)$ is uniform on $\{0, 1\}^n$.
- Assume,

$$Pr_{t \in G^1_{n+1}}[A(t) = 1] - Pr_{t \in U_{n+1}}[A(t) = 1] > \frac{1}{q(n+1)}$$

We are ignoring the absolute value.

Intuitively, it means if A answers 1 it is more likely to be from G^1_{n+1} and for 0 more likely it is from U_{n+1}.

Proof

The probability of $A(f(x) \circ b)$ returns 1 :

$$\begin{aligned} & Pr_{x \in U_n, b \in U_1}[A(f(x) \circ b) = 1] \\ & = Pr[A(f(x) \circ b) = 1 | b = h(x)](= \alpha) \cdot Pr[b = h(x)] \\ & + Pr[A(f(x) \circ b) = 1 | b = \overline{h(x)}](= \beta) \cdot Pr[b = \overline{h(x)}] \end{aligned}$$

This will be $\frac{1}{2}(\alpha + \beta)$.
But by assumption above it is more likely that $A(f(x) \circ b)$ will return 1 when $b = h(x)$. Therefore, we can get an algorithm for computing $h(x)$ from $f(x)$.

Proof Contd.

From the assumption we have,

$$\begin{aligned} Pr_{x \in U_n}[A(f(x) \circ h(x)) = 1] - Pr_{x \in U_n}[A(f(x) \circ b) = 1] \\ &= \alpha - \frac{1}{2}(\alpha + \beta) \\ &= \frac{1}{2}(\alpha - \beta) \\ &> q(n) \end{aligned}$$

Algorithm for computing $h(x)$ given $f(x)$

We construct a poly-time algo A' that on input $f(x)$ gives $h(x)$) with success probability $> \frac{1}{2}$

- A' takes $f(x)$ as input and outputs either 0 or 1.
    1. Choose $b \in \{0, 1\}$
    2. Run $A(f(x) \circ b)$
    3. If $A(f(x) \circ b) = 1$ output b, else output $\overline{b}$.
- Claim: $Pr[A'(f(x)) = h(x)] > \frac{1}{2} + \frac{1}{q(n)}$

Proof of Claim

$$
\begin{aligned}
Pr[A^{'}(f(x)) = h(x)] &= Pr[A(f(x) \circ b) = 1 | b = h(x)] Pr[b = h(x)] \\
&+ Pr[A(f(x) \circ b) = 0 | b = \overline{h(x)}] Pr[b = \overline{h(x)}] \\
&= \alpha \cdot \frac{1}{2} + (1 - \beta) \cdot \frac{1}{2} \\
&= \frac{1}{2} + \frac{1}{2}(\alpha - \beta) \\
&> \frac{1}{2} + \frac{1}{q(n)}
\end{aligned}
$$

Increasing the expansion factor

Theorem

If there exists a pseudorandom generator G with expansion factor $n+1$ then for any polynomial poly *there exists a pseudorandom generator G' with expansion factor* $\mathrm{poly}(n)$.

We will see the construction but not the proof of correctness.

PRG G' from G

- G - a PRG that expands random strings of length n to a pseudorandom string of length $n+1$.
- For a polynomial $q(n)$, T.S.T $G' : \{0,1\}^n \to \{0,1\}^{q(n)}$.

$$
\begin{aligned}
x &\to G \to f(x) \circ h(x) \\
f(x) \circ h(x) &\to G \to f(f(x)) \circ h(f(x)) \\
f^2(x) \circ h(f(x)) &\to G \to f^3(x) \circ h(f^2(x)) \\
&\vdots \\
f^{q(n)-1}(x) \circ h(f^{q(n)-1}(x)) &\to G \to f^{q(n)}(x) \circ h(f^{q(n)-1}(x))
\end{aligned}
$$

We can keep applying f since f is a permutation.

PRG G' from G

How does the output look like?

$$G'(x) = h(x) \circ h(f(x)) \circ \cdots \circ h(f^{q(n)-1}(x)).$$

Showing this is pseudorandom is available in Katz and Lindell textbook or in the Goldwasser and Bellare textbook.

Pseudorandom generators and Stream Ciphers

- The parity of any fixed subset of the output bits of a PRG is equal to 1 with probability very close to 1/2.
- Generating a stream cipher:
    1. Init: Takes as input a seed s and an optional initialization vector IV and outputs an initial state st_0.
    2. GetBits: Takes as input state information st_i and outputs a pseudorandom bit y and updated state st_{i+1}. Typically y is a block of bits, but here we take y as single bit for simplicity.
- For any expansion factor ℓ we define an algorithm G_ℓ which maps inputs of length n to outputs of length $\ell(n)$.
- It runs Init and then calls GetBits ℓ times.

Stream Ciphers

- A pseudorandom generator gives us a natural way to construct a secure encryption scheme (some call this as a stream cipher) with a key shorter than the message.
- Idea is to use pseudorandom pad instead of OTP.
- Encryption: XOR with the pseudorandom pad
- Decrypt also XOR with the same pad.
- What is shared? The seed!
- Example : RC4. (designed by Rivest), trade secret but leaked in 1994.
- Fast and simple, used in SSL and WEP, extremely insecure!