1 Properties of Pseudo Random Number Generator

The purpose of this lecture is to examine the properties of PRNGs. We examine how the security of a system relying on a PRNG cannot be greater than the resistance of the seed to an exhaustive search, which in turn is dependent on the size of the seed. Our goal is to efficiently stretch a seed derived from a random process, to a more usable length while retaining computational indistinguishably. Initial seed lengths range depending on the security required but are generally larger than 80 bits.

**Theorem 1.** If \( \exists \) a PRNG with Expansion Factor \( l(k) = k+1 \) then \( \forall \bar{l}(k) > k \), there exists a PRNG with expansion factor \( \bar{l}(k) \):

\[
G : \{0, 1\}^k \to \{0, 1\}^{k+1}
\]
\[
\Rightarrow G' : \{0, 1\}^k \to \{0, 1\}^{k+2}
\]
\[
\Rightarrow G'' : \{0, 1\}^k \to \{0, 1\}^{k+k}
\]
\[
\{0, 1\}^{2k}
\]

Theorem 1 shows how a seed can be continually stretched by the expansion factor of the PRNG, simply by taking the output of one iteration of the PRNG as the input to the next iteration.
We denote that the difference in probabilities is negligible using the following notion:

\[ U_{n+1} \approx G(U_n) \]

such that:

\[ U_k \approx G_k(G(U_n)) \]

For an algorithm applying \( G \) \( k \) times to be efficient it must grow as a polynomial of \( n \). For the output to be computed efficiently, the following must hold:

\[ \nu \times k < \frac{1}{n^c} \]

2 Cascading PRNG’s

One of the most important properties of a PRNG is the ability to stretch a random seed. Theorem 1 adds to that property by allowing the use of the same PRNG to stretch the output even more. However this can be a recipe for disaster if the cascading process does not meet certain security measures, such that the original seed should be large enough.

One way to cascade PRNG'S is to take a random seed \( k \geq 2^{80} \) and feed it to a PRNG that doubles the length of its input. Then we can split the output into two strings of length \( k \) and repeat the process.

Another example of cascading PRNG’S is the so called online PRNG. By using a PRNG that takes as an input seeds of length \( n \) and produces a \( n+1 \) bit length output, we can recursively feed the \( n \) bits of the output to the PRNG and keep the first bit. By concatenating the saved bit at every step with the bits that were saved from the previous steps we can double the length of the seed in a total of \( n \) steps.

Figure 2: cascading prng parallel
There are several ways we can cascade PRNG’s but we have to make sure that the properties of the PRNG are not violated. For example we cannot use a small length seed and stretch it until it is big enough, because an adversary can find the original seed by an exhaustive search, thus we have to make sure that the original seed is computationally impossible to be found through an exhaustive search. Another important property of the PRNG’S is that the original seed must be chosen in a random way, thus using the same seed in two different PRNG’S and then concatenate the output is not a good practice because the seed that is used as an input to the second PRNG is not random. As we can see, all the properties that apply to a single PRNG also apply to a scheme that uses cascading PRNG’S, thus we have to check every property to conclude that a cascading PRNG scheme is secure.

\[ \forall G \text{PRNG} : \]
\[ \exists_{bad} : G \rightarrow \exists(G_{bad}) \]
\[ \exists_{bad2} : G \rightarrow \exists_{bad2}(G) \]

\[ \forall G \text{ PRNG}, \exists_{bad}(G) \text{ is not a PRNG} \]
\[ \exists G \text{ PRNG} \exists_{bad2}(G) \text{ is not a PRNG} \]

3 PRNG from Quiz

\( G' \) is not a PRNG. To prove though that \( G' \) is not a secure PRNG we must first prove that \( G'' \) is secure.
Figure 4: $G''$

$G''$ is efficient.
$G''$ is deterministic.
$G''$ stretches its input.
To prove: $U_{2n+1} \approx G''(U_{2n})$

Remark 1. Chopping a uniform random bistring results in a uniform random bitstring.

Thus $G'' \equiv G(U_n) \approx U_{2n+1}$

**Assumption 1.** Suppose $\exists$ distinguisher $D''$ who can tell $U_{2n+1}$ from $G''(U_{2n})$

Then:

$$| Pr[D''[w] = 1 | w \leftarrow U_{2n+1}] - Pr[D'[w] = 1 | x \leftarrow U_{2n}, w \in G''(x)] | = \varepsilon$$

But are original assumption is that $G$ is ok so:

$$| Pr[D[w] = 1 | w \leftarrow U_{2n+1}] - Pr[D[w] = 1 | y \leftarrow U_n, w \leftarrow G(y)] | = | Pr[D''[w] = 1 | w \leftarrow U_{2n+1}] - Pr[D''[w] = 1 | x \leftarrow U_{2n}, y \leftarrow chopping(x), w \in G(y)] |$$

That means that if $G_{\{0,1\}^n \rightarrow \{0,1\}^{2n+1}}$ is a PRNG then $\rightarrow G''_{\{0,1\}^n \rightarrow \{0,1\}^{2n+1}}$ is a PRNG.

Remark 2. If $\hat{G}_{\{0,1\}^{n/2} \rightarrow \{0,1\}^{2n+1}}$ is a PRNG then $\rightarrow \hat{G}''_{\{0,1\}^n \rightarrow \{0,1\}^{2n+1}}$ is a PRNG that throws away the second half and takes the first half of the random string

Now that we proved that $G''$ is a secure PRNG we can use it as a counter example to prove that $G'$ is not secure.
Assumption 2. Suppose that $G'$ uses $G''$ to strech the input string such that: $s' = \frac{n}{2}$ is a uniformly random input then we pad $s'$ with 0's at the begining so we get a length of $n$ and then send it to $G''$.

![Diagram](image)

Figure 5: $G'$

It is clear that $G'$ will produce the same output under these conditions since $G''$ will always throw away the second part of the input and keep only the zero padded string. In terms of probability we get:

$$|Pr[D'(w) = 1|w \leftarrow U_{2n+1}] - Pr[D'(w) = 1|s' \leftarrow U_{n/2}, w \leftarrow G'(s')]| =$$

$$|Pr[D'(w) = 1|w \in U_{2n+1}] - Pr[D'(w) = 1|w \leftarrow \check{G}(0....0)]| =$$

$$|Pr[w = \check{G}(0....0)|w \leftarrow U_{2n+1}] - Pr[w = G(0....0)|w \in G(0....0)]| =$$

$$\frac{1}{2n+1} - 1 \approx 1.$$

So $G'$ is not a secure PRNG.

Summary: $G''$ is a PRNG because it takes a uniformly random string as input and streches it. We know that the first half of $s$ is random because any substring of a random string is random. Thus the output is a valid PRNG output.

References