More fun with PRGs, and stream ciphers

Noah Stephens-Davidowitz

June 7, 2023

1 Getting more bits out of your PRG

Remember that, in order to be interesting, a PRG must have a *stretch*. I.e., it must output $m(n) > n$ pseudorandom bits, given only n bits as input. In the previous lecture, we did not say much about what $m(n)$ was. We were happy as long as it was larger than n (and, of course, bounded by some polynomial in n, since we need an efficient algorithm to compute the m-bit output given an n-bit input). Of course, $m(n) = n + 1$ pseudorandom bits are much less useful than, say, $m(n) = 2n$ pseudorandom bits or $m(n) = n^{100}$ pseudorandom bits. So, maybe we really should care what $m(n)$ is?

And, notice that in some sense it's *way* more impressive to have a PRG with very large stretch. In particular, if a PRG outputs just $m(n) = n + 1$ bits, then its image might consist of half of all $m(n)$-bit strings—if all 2^n inputs map to a distinct output, then the resulting strings will be half of the 2^{n+1} strings of this length. It might therefore seem quite plausible that the output of this PRG would look random. In particular, a random string could land in the image of the PRG with probability $1/2$. But, if a PRG outputs just $m(n) = 2n$ bits, then its image can be at most a 2^{-n} fraction of the possible $2n$-bit strings. So, a random string will *almost never* happen to land in the image of the PRG. It seems like it should be much harder to have a distribution with such small support be indistinguishable from random. And, the situation is of course even more extreme if, say, $m(n) = n^{100}$! Right?

Today, we will see that the stretch $m(n)$ actually does not really matter in the following sense: given a PRG G with minimal stretch $m(n) = n + 1$, we can construct a new PRG G' with stretch $m'(n)$ for *any* polynomial $m'(n) > n$. So, if you believe in the existence of PRGs with stretch $m(n) = n + 1$, then you must also believe in PRGs with stretch $m(n) = n^{100}$! We will actually see multiple ways to prove this.

In fact, we will do something even better. We will see how to convert n bits into a continuous *stream* of pseudorandom bits. That is, we will show a procedure that generates pseudorandom bits "on demand." We will then notice that this is *already* enough to build a semantically secure encryption scheme *provided that the encryption algorithm is allowed to be stateful*!

Our proofs will all be hybrid arguments. So, this is also a great opportunity to just do a bunch of hybrid arguments. (In general, hybrid arguments are useful when you want to apply the same security property many times. E.g., if you want to argue that $(G(x_1), G(x_2), \ldots, G(x_\ell))$ is pseudorandom if G is a PRG, as we do below, then it is convenient to use a hybrid argument to make explicit the intuitive idea that "if $(G(x_1), \ldots, G(x_\ell))$ is not pseudorandom, then there must be some i such that $G(x_i)$ is not pseudorandom.")

2 Stretching the stretch via concatenation

Let's start out with a very simple construction that does not do very well (but will actually be useful in the next lecture). Here, we are assuming for simplicity that we can divide the input of G' neatly into ℓ disjoint strings of equal length.

Theorem 2.1. *If G is a PRG, then for any $\ell \leq \text{poly}(n)$, the function G' defined by $G'(x_1, \ldots, x_\ell) = (G(x_1), G(x_2), \ldots, G(x_\ell))$ is a PRG.*

Proof. It is immediate that G' is efficiently computable and expanding. So, we only need to prove that its output is pseudorandom.

We do this via a hybrid argument. Specifically, suppose that there exists a PPT adversary $\mathcal{A}$ with non-negligible advantage

$$\varepsilon(\ell n) := \Pr_{x_1,\ldots,x_\ell \sim \{0,1\}^n}[\mathcal{A}(1^{n\ell}, G'(x_1, \ldots, x_\ell)) = 1] - \Pr_{U_1,\ldots,U_\ell \sim \{0,1\}^m}[\mathcal{A}(1^{n\ell}, U_1, \ldots, U_\ell) = 1]$$

in the PRG game against G'. Define $Y^{(i)}$ to be the random variable given by $(G(x_1), \ldots, G(x_i), U_{i+1}, \ldots, U_\ell)$, where $x_1, \ldots, x_i \sim \{0,1\}^n$ and $U_{i+1}, \ldots, U_\ell \sim \{0,1\}^m$.

Notice that $Y^{(0)}$ is uniformly random, and $Y^{(\ell)}$ is exactly equal to $G'(x_1, \ldots, x_\ell)$. It therefore follows from a hybrid argument that there exists an i such that

$$\Pr[\mathcal{A}(1^{n\ell}, Y^{(i+1)}) = 1] - \Pr[\mathcal{A}(1^{n\ell}, Y^{(i)}) = 1] \geq \varepsilon(\ell n)/\ell \ .$$

In particular as long as ℓ is polynomially bounded, this is non-negligible.

We now construct an adversary $\mathcal{A}'$ in the PRG game against G as follows. $\mathcal{A}'$ takes as input 1^n and $Y^* \in \{0,1\}^m$, which is either sampled uniformly at random *or* set to be $G(x^*)$ where $x^* \sim \{0,1\}^n$. It then samples $x_1, \ldots, x_i \sim \{0,1\}^n$ and $U_{i+2}, \ldots, U_\ell \sim \{0,1\}^m$ and sets

$$b' \leftarrow \mathcal{A}(1^{n\ell}, G(x_1), \ldots, G(x_i), Y^*, U_{i+2}, \ldots, U_\ell) \ .$$

It then outputs b'.

Clearly $\mathcal{A}'$ runs in polynomial time. To finish the proof, we observe that (1) when Y^* is sampled uniformly at random, the input to $\mathcal{A}$ is distributed identically to $Y^{(i)}$; and (2) when $Y^* = G(x^*)$ for $x^* \sim \{0,1\}^n$, the input to $\mathcal{A}$ is distributed identically to $Y^{(i+1)}$. It follows that

$$\Pr_{x^* \sim \{0,1\}^n}[\mathcal{A}'(1^{n\ell}, G(x^*)) = 1] - \Pr_{Y^* \sim \{0,1\}^m}[\mathcal{A}'(1^{n\ell}, Y^*) = 1] \geq \varepsilon(\ell n)/\ell \ ,$$

which contradicts the fact that G is a PRG (since ℓ is polynomially bounded). Therefore, our assumption that such an $\mathcal{A}$ exists is false, and G' is a PRG. □

This construction does help us go from very small stretch to slightly larger stretch. E.g., if G has stretch $m(n) = n + 1$, then G' has stretch $m'(\ell n) = \ell n + \ell$. However, this cannot yield stretch better than $m'(n') = n' + (n')^{1-\varepsilon}$ for some $\varepsilon > 0$.

3 Stretching the stretch via naive composition

Another way to increase the stretch of a PRG uses the following reasoning. Suppose we have a PRG with stretch $m(n) = n+1$. This means that we have a process for converting n truly random x bits into $n+1$ bits Y that are pseudorandom. Intuitively, Y is "just as good as uniformly random bits," so that we should be able to use Y in *any* circumstances in which we would use truly random bits. We can use Y to generate our secret keys without compromising security; we can replace the uniformly random input to our one-way functions by Y and it will be essentially just as hard to invert on this input; we can use Y in our coinflipping protocols; etc.

But, if Y is as good as random, then we should be able to use Y as the *input* to our PRG. And, if we run our PRG on input Y to compute $G(Y)$, then we will get $n+2$ bits back. Presumably if Y is as good as random, then $G(Y)$ will be as good as pseudorandom (which is as good as random)! In other words, once we have a way to "stretch random strings," we can do it twice to stretch them even more.

More generally, we can do this many times to make our strings much longer. E.g., we can run $G(G(G(G(x))))$.

In fact, we can even do this a superconstant number of times, though we have to be careful because as our input gets longer and longer, our running time increases. E.g., if the stretch of G is already say $m(n) = n^2$, then if we repeat this ℓ times, we will end up with strings of length n^{2^ℓ}, which is superpolynomial if $\ell = \ell(n)$ is superconstant. To avoid this annoyance, we will assume that we start out with a PRG with stretch $m(n) = n+1$. Then, we can do this $\ell(n) \leq \text{poly}(n)$ times in order to compute $n + \ell(n)$ pseudorandom bits, for our favorite polynomially bounded function $\ell(n)$.

Theorem 3.1. *For a PRG G with stretch $m(n) = n+1$ and any polynomially bounded function $\ell(n)$, let $G^\ell(x) := G(G(\cdots(G(x))\cdots))$, where G is applied $\ell(|x|)$ times. Then, G^ℓ is a PRG with stretch $m'(n) = n + \ell(n)$.*

Proof. It is immediate that G^ℓ is efficiently computable and expanding, so we only need to show that it is pseudorandom.

We assume the opposite, that there exists a PPT adversary $\mathcal{A}$ with non-negligible advantage

$$\varepsilon(n) := \Pr_{x\sim\{0,1\}^n}[\mathcal{A}(1^n, G^\ell(x)) = 1] - \Pr_{U\sim\{0,1\}^{m'(n)}}[\mathcal{A}(1^n, U) = 1]$$

in the PRG game against G^ℓ. And, we use this to derive a contradiction.

The proof is (surprise!) via a hybrid argument. We define

$$Y^{(i)} := G^{\ell-i}(x^{(i)})$$

where $x^{(i)} \sim \{0,1\}^{n+i}$. In other words, $Y^{(i)}$ is what you get if instead of running G a total of ℓ times in a row on a string of length n, you instead simply sample a uniformly random string of length $n+i$ and run G only $\ell - i$ times. Equivalently, $Y^{(i)}$ is obtained by "replacing the output of the first i runs of G by a uniformly random $(n+i)$-bit string."

Notice in particular that $Y^{(0)} = G^\ell(x)$ and $Y^{(\ell)}$ is uniformly random. It therefore follows from a hybrid argument that there exists an index i such that

$$\Pr[\mathcal{A}(1^n, Y^{(i-1)}) = 1] - \Pr[\mathcal{A}(1^n, Y^{(i)}) = 1] \geq \varepsilon(n)/\ell(n)\ .$$

Since $\ell(n)$ is polynomial in n, this is a non-negligible function.

We construct $\mathcal{A}'$ in the PRG game against G as follows. $\mathcal{A}'$ takes as input 1^{n+i-1} and $Y^* \in \{0,1\}^{n+i}$ (which is either uniformly random or $G(x)$ for uniformly random $x \sim \{0,1\}^{n+i-1}$). $\mathcal{A}'$ then computes $Z^* := G^{(\ell-i)}(Y^*)$ and runs $b' \leftarrow \mathcal{A}(1^n, Z^*)$ and outputs b'.

Clearly $\mathcal{A}'$ runs in polynomial time. Now, notice that if Y^* is sampled uniformly at random, then Z^* is distributed identically to $Y^{(i)}$. On the other hand, if $Y^* := G(x^*)$ for $x^* \sim \{0,1\}^{n+i-1}$, then Z^* is distributed identically to $Y^{(i-1)}$.

It follows that

$$\Pr_{x^* \sim \{0,1\}^{n+i-1}}[Y^* := G(x^*),\ \mathcal{A}'(1^{n+i-1}, Y^*) = 1] - \Pr_{Y^* \sim \{0,1\}^{n+i}}[\mathcal{A}(1^{n+i-1}, Y^*) = 1] \geq \varepsilon(n)/\ell(n)\ ,$$

which is non-negligible. This contradicts the fact that G is a PRG. So, our assumption that G^ℓ was insecure must be false, and therefore G^ℓ is a PRG. □

4 An unending "stream" of bits via a different kind of composition—and stream ciphers

The previous construction is quite useful. E.g., if Alice wants to send Bob an m-bit message securely but they only share an n-bit key k, she can use $G^{m-n}(k)$ as a one-time pad. (I typically ask my students to prove this for homework.)

But, what if Alice and Bob wish to continue communicating afterwards? They *cannot* use the same one-time pad twice if they want security! More generally, they cannot securely use $G^{m_1-n}(k)$ as a pad to send one message of length m_1 and *then* use $G^{m_2-n}(k)$ as a pad for a second message of length m_2 with the same key k. Their key is essentially spent after the first time they use it. So, it seems that Alice and Bob are simply out of luck.

But, actually, if Alice and Bob are just *slightly* less greedy with their pseudorandom bits (and if they are stateful; more on this soon), then they can securely send as many messages as they want! To see this, instead of generating m pseudorandom bits to send a plaintext of length m, imagine that Alice and Bob generate $n + m$ bits. They can then use the last m pseudorandom bits as a one-time pad to send their m-bit plaintext and keep the first n pseudorandom bits as a *new key*, which they can use to send the next message. By continuing in this way, they can send as many messages as they wish securely.

To make this precise, we will simplify things slightly and assume that Alice and Bob simply wish to send a single bit at a time. (Of course, if our scheme allows us to send arbitrarily many plaintexts, then one-bit plaintexts are sufficient, since we can always concatenate many bits together to get longer plaintexts.)

So, if G is a PRG with stretch $m(n) = n + 1$ and Alice and Bob share a uniformly random key $k_0 \sim \{0,1\}^n$, they can behave as follows. To send their first plaintext bit $p_1 \in \{0,1\}$, they compute $(k_1, b_1) := G(k_0)$, where k_1 represents the first n bits of the output and b_1 represents the last bit (i.e., the "extra" bit). Alice then sends the ciphertext $c_1 := b_1 \oplus p_1$. To send the next bit $p_2 \in \{0,1\}$, they compute $(k_2, b_2) := G(k_1)$ and Alice can send the ciphertext $c_2 := b_2 \oplus p_2$, etc.

In effect, Alice and Bob have converted their initial n random bits into an infinite stream of pseudorandom bits!

For a function $\ell(n)$, let's introduce the notation $G^{\otimes \ell}(k)$ to represent the bits $(b_1, \ldots, b_{\ell(|k|)}) \in \{0,1\}^{\ell(|k|)}$ generated by this process starting with input k. ($G^{\otimes \ell}$ is not standard notation, or even

particularly good notation :). I would call it G^ℓ, but I've already used that notation.) I.e., if we set $k_0 := k$, then k_i, b_i are defined by the recurrence $(k_i, b_i) := G(k_{i-1})$.

Theorem 4.1. *If G is a PRG with stretch $m(n) = n+1$, then $G^{\otimes \ell}$ is a PRG with stretch $\ell(n)$ for any ℓ with $n < \ell(n) \leq \text{poly}(n)$.*

Proof. It is immediate that $G^{\otimes \ell}$ is expanding and efficiently computable (expanding because we took $\ell(n) > n$ and efficiently computable because we took $\ell(n) \leq \text{poly}(n)$). So, we only need to prove that $G^{\otimes \ell}$ is secure.

So, we suppose not. I.e., we suppose that there exists a PPT $\mathcal{A}$ such that its advantage

$$\varepsilon(n) := \Pr_{k \sim \{0,1\}^n}[\mathcal{A}(1^n, G^{\otimes \ell}(k)) = 1] - \Pr_{U \sim \{0,1\}^{\ell(n)}}[\mathcal{A}(1^n, U) = 1]$$

is non-negligible. We will again use a hybrid argument. (See?! Lots of hybrid arguments!) Specifically, let $k_0 := k \sim \{0,1\}^n$ and define k_i, b_i as above, i.e., $(k_i, b_i) := G(k_{i-1})$. Let $Y^{(i)}$ be the output of $G^{\otimes \ell}$ when $b_1, \ldots, b_i$ and $k_1, \ldots, k_i$ are replaced by uniformly random bits (as opposed to pseudorandom bits). Equivalently, we can imagine sampling $Y^{(i)}$ by outputting i uniformly random bits $b'_1, \ldots, b'_i$ and then sampling $k'_i \sim \{0,1\}^n$ and outputting $G^{\otimes(\ell-i)}(k'_i)$.

Notice that $Y^{(\ell)}$ is uniformly random while $Y^{(0)}$ is distributed identically to $G^{\otimes \ell}(k)$ for $k \sim \{0,1\}^n$. So, it follows from a hybrid argument that there exists an i such that

$$\Pr[\mathcal{A}(1^n, Y^{(i)}) = 1] - \Pr[\mathcal{A}(1^n, Y^{(i+1)}) = 1] \geq \varepsilon(n)/\ell(n) \ ,$$

which is non-negligible.

Now, we construct an $\mathcal{A}'$ in the PRG game against G as follows. $\mathcal{A}'$ takes as input 1^n and $Y^* \in \{0,1\}^{n+1}$, which is either sampled uniformly at random or equal to $G(k^*)$ for $k^* \sim \{0,1\}^n$. $\mathcal{A}'$ then samples $b_1, \ldots, b_i$ uniformly at random, and sets $(b_{i+1}, \ldots, b_{\ell(n)}) := G^{\otimes(\ell-i)}(Y^*)$. Finally, it runs

$$b' \leftarrow \mathcal{A}(1^n, b_1, \ldots, b_{\ell(n)})$$

and outputs b'.

Notice that if $Y^* \sim \{0,1\}^{n+1}$ is sampled uniformly at random, then $b_1, \ldots, b_{\ell(n)}$ are distributed identically to $Y^{(i+1)}$. On the other hand, if $Y^* = G(k^*)$ for uniformly random $k^* \sim \{0,1\}^n$, then they are distributed identically to $Y^{(i)}$.

It follows that

$$\Pr_{k^* \sim \{0,1\}^n}[\mathcal{A}'(1^n, G(k^*))] - \Pr_{Y^* \sim \{0,1\}^{n+1}}[\mathcal{A}'(1^n, Y^*)] \geq \varepsilon(n)/\ell(n) \ ,$$

which is non-negligible. But, this contradicts the assumption that G is a PRG. It follows that $G^{\otimes \ell}$ is in fact a PRG, as claimed. □

4.1 Stream ciphers

The encryption scheme based on $G^{\otimes \ell}$ that we described above is an example of a *stream cipher*. Stream ciphers are encryption schemes in which Alice and Bob maintain a *state* in order to encrypt and decrypt a "stream" of bits. The state here is simply the updated key k_i. Each time that Alice encrypts, she uses the current state k_i to compute the ciphertext c_i *and* her new state k_{i+1}.

Similarly, when Bob decrypts, he uses his current state k_i to compute the plaintext m_i *and* the new state k_{i+1}.

Notice that this does not actually satisfy our definition of an encryption scheme as we saw it in the first two lectures. Our definition did not say anything about a state. But, if we modify the definition appropriately (or just expand our model of computation to include *stateful algorithms*), then this scheme actually is many-message semantically secure! So, ignoring the state for the moment, we have just seen our first semantically secure encryption scheme! (We will not actually prove this to avoid getting bogged down in the formal definition of stateful encryption. The proof is essentially identical to the proof in the homework that PRGs yield good one-time pads.) It is also extremely efficient, requiring simply a one-bit ciphertext to encrypt a single-bit plaintext. Better yet, combined with what we've learned in the previous lectures, we know that we can build such a scheme using only the assumption that one-way functions exist!

In fact, stream ciphers are still used in practice, though we will not see more of them in this class.

It is, of course, a bit unsatisfying that our stream cipher requires a state. And, notice that it relies on it *heavily*. Of course, for correctness, Alice and Bob must agree on k_i at every step. If at some point Bob does not receive one of Alice's messages, or they otherwise get out of sync, then the correctness of the scheme totally breaks. E.g., if Alice thinks that we are spending the ith message but Bob thinks we are sending the jth message, then Alice will compute some ciphertext $c := p \oplus b_i$, and Bob will try to decrypt this as $p' := c \oplus b_j$, which will not necessarily give the right answer, since we might have $b_i \neq b_j$.

Even worse, *security* can break if Alice is not careful! In particular, if Alice ever reuses a key k_i, then the scheme will no longer be secure! Specifically, if she reuses a key, then she will be using the same one-time pad twice! (Don't use the same *one-time* pad twice!) So, Alice must be sure to update her state at each step, and if she forgets to, then security is completely compromised. More generally, Alice must never return to somewhere earlier in the sequence (e.g., if her computer temporarily loses power). This might not sound so difficult, but it becomes quite messy over the internet and particularly so if, e.g., Alice is actually a large data center consisting of many computers linked together, or even many large data centers scattered around the globe, with many different machines communicating with Bob. (In practice, Alice really is some gigantic collection of data centers, though it's a lot more fun to describe her as a person :). Of course, I am not a practitioner and have no idea how this actually works :).)

Anyway, by declaring that we are not satisfied with stateful encryption schemes, we will find an excuse to study some more beautiful cryptography. So, we choose to be unsatisfied :).

4.2 Towards making the scheme stateless

If we want to convert our stream cipher into a stateless encryption scheme, the first step is to notice that we can make Bob (aka, the decryption algorithm) stateless if he is willing to be a bit less efficient. Specifically, suppose that Alice outputs not just the ciphertext c_i but also a counter i. So, the full ciphertext looks like (i, c_i). Then, Bob can compute k_{i-1} himself using only the original key k_0 (and no state), and he can then use k_i to decrypt the ciphertext c_i.

So, as long as Alice maintains a counter and appends it to her ciphertexts, Bob does not need to be stateful. This is really quite useful because it eliminates the need for Alice and Bob to remain synchronised in order to maintain correctness.

However, this does not solve the more important issue, which is that Alice must maintain a state for security, since security can be compromised if she reuses a key. As a first attempt towards solving this issue, let's notice that i doesn't really need to be a counter. Alice can encrypt the ith plaintext p_i under the rth key k_r for whatever r she pleases (not necessarily $r = i$), computing the ciphertext $(r, b_r \oplus p_i)$. Bob can still decrypt this. And, as long as she does not use the same r to encrypt two different plaintexts, this will still be secure.

We call a value that should only be used once like this a *nonce*. Often, when we need a nonce r, we simply sample it randomly. Intuitively, as long as we sample it from a distribution with high enough entropy, the probability that we will see the same value for r twice should be negligible. So, we can imagine that Alice samples r from some suitable distribution, computes b_r using k_0 and the PRG G, and outputs $(r, b_r \oplus p)$ as her ciphertext.

But, which distribution should we sample r from in this case? Here, we run into a problem. Notice that the pseudorandom bits $b_1, \ldots, b_r$ must be computed *sequentially*. So, the amount of time needed to compute b_r is proportional to r. Therefore, r must be polynomially bounded in the security parameter n in order to maintain efficiency. (This is also necessary for security. Remember that $G^{\otimes \ell}$ is *not* necessarily secure if ℓ is superpolynomial.) This means that the number of possible values for r must be bounded by some polynomial in the security parameter. And, this means that the probability of using the same r twice is *non*-negligible (even if we only use the encryption scheme twice)!

In some sense, this sequential issue means that the $G^{\otimes \ell}$ construction only allows us to expand a random n-bit seed into poly(n) pseudorandom bits. In the next lecture, we will see a different scheme that allows us to "expand a random n-bit seed into 2^n pseudorandom bits!" (We will do this in a way that actually makes sense :).)