CSCI-GA.3210-001
MATH-GA.2170-001

Introduction to Cryptography Feb 15, 2012

Lecture 4

Lecturer: Yevgeniy Dodis *Spring 2012*

This lecture will study the notion of *hardcore bit* for a given OWF f. Intuitively, such a hardcore bit $h(x)$ is easy to compute from x, but almost impossible to even *guess* well from $f(x)$. We will see specific examples of hardcore bits for modular exponentiation, RSA and Rabin's squaring function. Next we will show a groundbreaking result of Goldreich-Levin, that (more or less) shows a general hardcore bit for *any* OWF. We will then consider two natural applications of hardcore bits to the problem of encryption. Firstly, we will show an inuitively good public-key encryption for one bit, and then a "plausible" secret-key encryption which encrypts $k+1$ bits with k-bit key (thus beating the Shannon information-theoretic bound). We will then try to extend the hardcore bit construction to extracting many "pseudorandom bits", by analogy to the S/Key system. We will notice that our many-bit construction seems to satisfy a very special notion of security, which we call "next-bit unpredictability". We then will make a formal definition of a *pseudorandom generator*, which seems to be a more relevant primitive for our encryption applications. We stop by asking the question of whether our "next-bit secure" construction is indeed a pseudorandom generator.

1 Hardcore Bits

Last lecture we addressed some of the criticism over straightforward usages of OWF's, OWP's and TDP's. Specifically, our main criticism was the fact that a OWF $f(x)$ could reveal a lot of partial information about x (remember generic example $f(x_1, x_2) = (f'(x_1), x_2)$ that reveals half of its input bits, or more realistic one of exponentiation $f(x) = g^x \bmod p$ that reveals the $LSB(x) = y^{(p-1)/2}$).

The obvious solution seemed to try to *completely* hide all information about x, given $f(x)$. However, this leads to a vicious circle, since it is really equivalent to the problem of secure encryotion that we started from. Instead, we explore the idea of *completely* hiding not *all*, but only a *specific and carefully chosen partial information about* x, when given $f(x)$. The first preliminary step towards this goal is to determine how to completely hide exactly just *one* bit of information about the plaintext x. This leads us to the following definition:

Definition 1 [Hardcore bit] A function $h : \{0,1\}^* \to \{0,1\}$ is called a hardcore bit for a function f if

- $h(x)$ is polynomial time computable (from x):

$$(\exists \text{ poly-time } H)(\forall x)[H(x) = h(x)]$$

- No PPT algorithm that can predict $h(x)$ given $f(x)$ better than flipping a coin:

$$(\forall\ \mathsf{PPT}\ A)\ \Pr[A(f(x)) = h(x) \mid x \leftarrow^r \{0,1\}^k] \leq \frac{1}{2} + \mathsf{negl}(k)$$

$\diamondsuit$

Remark 1 *Notice, we compare the success of A to a $\frac{1}{2} + \mathsf{negl}(k)$ rather than $\mathsf{negl}(k)$, as we did for OWF's. The reason is that the output of h is now only one bit, so A can always guess if with probability $\frac{1}{2}$ by flipping a coin. Of course, we could have said that h is difficult to compute by saying that A succeeds with probability at most $1-\varepsilon$, where ε is non-negigible. However, we really want to say much more (and that is why we started with $h(x)$ being just* 1 *bit for now): not only is h hard to* compute*, it is even hard to* predict*.*

Remark 2 *We can also naturally define hardcore bits for* collection *of* OWF*s, where h can depend on the public key PK of f.*

Thus, a hardcore bit $h(x)$ pinpoints an aspect of x that is truly hidden given $f(x)$. Namely, the knowledge of $f(x)$ does not allow us to predict $h(x)$ any better than *without it* (i.e., by flipping a coin), so $h(x)$ looks random given $f(x)$. It is to be noted that we need not require that $h(x)$ be a bit that is selected from the string x itself, but, in general, may depend on x in more complex (but efficiently computable) ways. This is not inconsistent with the idea that $h(x)$ is supposed to represent some general information about x. We might want to attempt the construction in two ways:

1. Taking as hypothesis that a *concrete* function is OWF, exhibit a hardcore bit for that function. (This is a useful, but not very general construction.)

2. Taking as hypothesis that an *arbitrary* function is OWF, exhibit a hardcore bit for that function. (This is the strongest construction we can hope for.)

2 Hardcore Bits for Concrete OWF's

The concrete function that we consider is the exponentiation function mod p, $f(x) = y = g^x \bmod p$, where g is the generator of $\mathbb{Z}_p^*$. Recall, the least significant bit $LSB(x)$ was not hardcore for f, since it could be computed from y in polynomial time. Instead, we define the most significant bit of x, MSB, as follows:

$$MSB(x) = \begin{cases} 0, & \text{if} \quad x < \frac{p-1}{2} \\ 1, & \text{if} \quad x \geq \frac{p-1}{2} \end{cases}$$

Remark 3 *$MSB(x)$ not defined to be simply x_1 in order to make it unbiased, since the prime p is not a perfect power of* 2*.*

Theorem 1 *If $f(x) = (g^x \bmod p)$ is a* OWP*, then $MSB(x)$ is a hardcore bit for f.*

Proof Sketch: A rigorous proof for the above theorem exists. It is our usual proof by contradiction, which takes as hypothesis that MSB is *not* an hardcore bit for f, and proves that f is not OWF. The proof is constructive, and explicitly transforms any PPT that can compute $MSB(x)$ from $f(x)$ with non-negligible advantage, into a PPT that can compute the Discrete Log with non-negligible probability. This proof is, however, somewhat technical, so we settled for a simpler, but quite representative result stated below. □

Lemma 1 *If there exists a* PPT *that can* always *compute $MSB(x)$ from $f(x)$, then there is a* PPT *that can* always *invert $f(x)$ (i.e., compute the discrete log of $y = f(x)$).*

Proof: The idea of the algorithm is very simple. We know that $LSB(x) = x_k$ is easy to compute given $y = g^x \bmod p$. This way we determine x_k. Now we can transform y into $g^{[x_1 \ldots x_{k-1}0]} = (g^{[x_1 \ldots x_{k-1}]})^2 \bmod p$ (by dividing y by g if $x_k = 1$). We also know how to extract square roots modulo p. So it seems like we can compute $g^{[x_1 \ldots x_{k-1}]}$, and keep going the same way (take LSB, extract square root, etc.) until we get all the bits of x. However, there is a problem. The problem is that $g^{[x_1 \ldots x_{k-1}0]}$ has *two* square roots: $y_0 = g^{[x_1 \ldots x_{k-1}]}$ (the one we want) and $y_1 = (-g^{[x_1 \ldots x_{k-1}]}) = g^{\frac{p-1}{2}+[x_1 \ldots x_{k-1}]}$ (here we used the fact that $-1 = g^{(p-1)/2}$). So after we compute the square roots y_0 and y_1, how do we know which root is really $y_0 = g^{[x_1 \ldots x_{k-1}]}$? Well, this is exactly what the hardcore bit MSB tells us! Namely, $MSB(Dlog(y_0)) = 0$ and $MSB(Dlog(y_1)) = 1$. The complete algorithm follows.

```
$i = k$;
while ($y \geq 1$) do /* $y = f(x)$ */
begin
    output ( $x_i = LSB(Dlog(y))$ );
    /* Assertion: $x = [x_1 x_2 \ldots x_i]$ */
    if ( $x_i == 1$ ) then $y := y/g$;
    /* Assertion: $y = g^{[x_1 x_2 \ldots x_{i-1} 0]} = (g^{[x_1 x_2 \ldots x_{i-1}]})^2$ */
    Let $y_0$ and $y_1$ be square roots of $y$;
    If ( $MSB((Dlog(y_1)) == 0$ )        /* Using hypothetical algortihm */
      then $y := y_1$
      else $y := y_2$
    $i := i - 1$;
end
```

To summarize, the value of MSB plays a critical role in distinguishing which square root of y corresponds to $x/2$. This enables us to use the LSB iteratively, so that the process is continued to extract all the bits of x. □

It turns out that the other OWF's we study have natural hardcore bits as well:

1. LSB and MSB are hardcore bits for Rabin's Squaring Function.

2. All the bits of x are hardcore bits for RSA.

3 Construction of a hardcore bit for arbitrary OWF

Looking at the previous examples, we would now like to see if any OWF f has some easy and natural hardcore bits. Specifically, it would be great if at least one the following two statements was true:

1. Any OWF has some particular bit x_i in x (i depends on f) which is hardcore for f.

2. A concrete boolean function h (which is not necessarily an input bit) is a hardcore bit for *all* OWF's f.

Unfortunately, both of these hopes are false in general.

1. From arbitrary OWF f, it is possible to construct another OWF g, such that none of the bits of x are hardcore for g. (cf. Handout).

2. For any boolean function h and OWF f, if we let $g(x) = f(x) \circ h(x)$, then: (1) g is also a OWF; (2) h is not hardcore for g. Part (1) follows from the fact that an inverter A for g would imply the one for f. Indeed, given $y = f(x)$, we can ask the inverter A for g to invert both $f(x) \circ 0$ and $f(x) \circ 1$, and see if at least one of them succeeds. Part (2) is obvious. Thus, no "universal" h exists.

Despite these negative news, it turns out that we nevertheless have a very simple hardcore bit for an arbitrary OWF. This is the celebrated Goldreich Levin construction.

4 Goldreich Levin Construction

We will begin with a definition that generalizes the concept of selecting a specific bit from a binary string.

Definition 2 [Parity] If $x = x_1 x_2 \ldots x_k \in \{0,1\}^k$, and $r = r_1 r_2 \ldots r_k \in \{0,1\}^k$, then $h(x,r) = r_1x_1 \oplus r_2x_2 \ldots \oplus r_kx_k = (r_1x_1 + r_2x_2 + \ldots + r_kx_k \bmod 2)$ is called the parity of x with respect to r. $\diamondsuit$

Notice, r can be viewed as a selector for the bits of x to be included in the computation of parity. Further, the expression for the notation for the inner product, $\cdot$, can be used profitably, i.e., $h(x,r) = (r \cdot x)$ can be viewed as the inner product of binary vectors x and r modulo 2. The "basis strings" e_i with exactly one bit $r_i = 1$, give the usual specific bit selection x_i.

The Goldrecih-Levin theorem essentially says that "if f is a OWF, then most parity functions $h(x,r)$ are hardcore bits for f." To make the above statement more precise, it is convenient to introduce an auxiliary function $g_f(x,r) = f(x) \circ r$ (the concatenation of $f(x)$ and r), where $|x| = |r|$. Notice, a random input for g_f samples *both* r and x at random from $\{0,1\}^k$, so $h(x,r) = x \cdot r$ indeed computes a "random parity for (randomly selected) x". Notice also, that if f is a OWF/OWP/TDP, then so is g_f (in particular, g_f is a permutation if f is). Now, the Goldrecih-Levin theorem states that

Theorem 2 (Goldreich-Levin Bit) *f is a OWF, then $h(x,r)$ is a hardcore bit for g_f. More formally:*

$$(\forall\ \mathsf{PPT}\ A)\quad \Pr[A(f(x),r)=(x\cdot r)\mid x,r\leftarrow^r\{0,1\}^k]<\frac{1}{2}+\mathsf{negl}(k)$$

Remark 4 *A hardcore bit for g_f is as useful as the one for f, since f is really computationally equivalent to g_f (since r is part of the output, inverting g_f exactly reduces to inverting f). Thus, we will often abuse the terminology and say "since f is a OWF, let us take its hardcore bit $h(x)$". But that we really mean that in case we are not aware of some simple hardcore bit for a specific f, we can always take the Goldreich-Levin bit for g_f and use it instead. Similar parsing should be given to statement of the form "every OWF has a hardcore bit". Again, in the worst case always use the Goldrecih-Levin bit for g_f. Finally, another popular interpretation of this result is that "most parities of f are hardcore".*

A rigorous proof of the Goldreich-Levin theorem exists. For simplicity, we will assume that f (and thus g_f) are *permutations*, so that $(x\cdot r)$ is uniquely defined given $f(x)\circ r$.

The proof proceeds proof by contradiction, by taking as hypothesis that $h(x,r)$ is *not* a hardcore bit for g_f, and proves that f is not a OWF. The proof is constructive, and explicitly transforms any PPT A that can compute $h(x,r)$ with non negligible probability for most values of r, given $f(x)$ and r, into a PPT B that can compute x with non negligible probability, given $f(x)$. However and despite the simplicity of theorem statement, the full proof is extremely technical. Therefore, we will again only give a good intuition of why it works.

Before going to the general proof, in the next subsection we give two simple cases which make the proof considerably simpler. A reader not interesting in the technical proof is adviced to read only these two cases, and skip the following subsection describing the general case.

4.1 Simple Cases

First, assume that we are given a PPT A that *always* computes $(x\cdot r)$ given $f(r)\circ r$. Well, then everything is extremely simple. Given $y=f(x)$ that we need to invert, we already observed that $x\cdot e_i=x_i$ is the i-th bit of x, where e_i is a vector with 1 only at poistion i. Thus, asking $A(y,e_i)$ will give us x_i, so we can perfectly learn x bit by bit.

Unfortunately, our assumption on A is too strong. In reality we only know that it succeeds on a slight majority of r's. In particular, maybe it always refuses to work for "basis" $r=e_i$. So let us be more reasonable and assume that

$$\Pr[A(f(x),r)=(x\cdot r)\mid x,r\leftarrow\{0,1\}^k]>\frac{3}{4}+\varepsilon \tag{1}$$

where ε is non-negligible. Here we use $3/4$ instead of $1/2$ for the reason to be clear in a second. But first we need a definition. Given a *particular* value x, we let

$$Succ(x)\overset{\text{def}}{=}\Pr[A(f(x),r)=(x\cdot r)\mid r\leftarrow\{0,1\}^k] \tag{2}$$

Namely, $Succ(x)$ is "how well" A predicts $x \cdot r$ for a particular x (averaged over r). Then, Equation (1) is equivalent to saying that the *expected value* of $Succ(x)$ is non-trivially greater than 3/4:

$$\mathbb{E}[Succ(x)] > \frac{3}{4} + \varepsilon \tag{3}$$

Let us call x "good" if $Succ(x) > \frac{3}{4} + \frac{\varepsilon}{2}$. Then a simple averaging argument show that

$$\Pr[x \text{ is good} \mid r \leftarrow \{0,1\}^k] > \frac{\varepsilon}{2} \tag{4}$$

Indeed, if Equation (4) is false, then conditioning on whether or not x is good, the largest that $\mathbb{E}[Succ(x)]$ can be is

$$\mathbb{E}[Succ(x)] \leq \frac{\varepsilon}{2} \cdot 1 + \left(1 - \frac{\varepsilon}{2}\right) \cdot \left(\frac{3}{4} + \frac{\varepsilon}{2}\right) < \frac{3}{4} + \varepsilon$$

which is contradicting Equation (3).

We now will construct an inverter B for f which will only work well for good x. But since the fraction of good x is non-negligible (at least $\varepsilon/2$ by Equation (4)), the existence of B will contradict the fact that f is a OWF. Thus, in the following we will assume that x is good when analyzing the success of B.

The idea is to notice that for any $r \in \{0,1\}^k$

$$(x \cdot r) \oplus (x \cdot (r \oplus e_i)) = \left(\sum_{j \neq i} r_j x_j + r_i x_i \bmod 2\right) \oplus \left(\sum_{j \neq i} r_j x_j + (1 - r_i) x_i \bmod 2\right) = x_i$$

Moreover both r and $(r \oplus e_i)$ are *individually random* when r is chosen at random. Hence, for any fixed index i and any good x,

$$\begin{aligned} \Pr[A(y,r) \neq (x \cdot r) \mid y = f(x),\ r \leftarrow \{0,1\}^k] &< \frac{1}{4} - \frac{\varepsilon}{2} \\ \Pr[A(y, r \oplus e_i) \neq (x \cdot (r \oplus e_i)) \mid y = f(x),\ r \leftarrow \{0,1\}^k] &< \frac{1}{4} - \frac{\varepsilon}{2} \end{aligned}$$

Thus, with probability at least $1 - 2(\frac{1}{4} - \frac{\varepsilon}{2}) = \frac{1}{2} + \varepsilon$, A will be correct in *both* cases, and hence we correctly recover x_i with probability $\frac{1}{2} + 2\varepsilon$. Thus, using A we can have a PPT procedure $B'(y,i)$ (which is part of "full" B below) which will sample a random r and return $A(y,r) \oplus A(y \oplus e_i)$, such that for *any* good x and any i,

$$\Pr[B'(y,i) = x_i \mid y = f(x)] \geq \frac{1}{2} + \varepsilon \tag{5}$$

Namely, we can predict any particular bit x_i with probability greater than 1/2. Thus, repeating $B'(y,i)$ roughly $t = O(\log k/\varepsilon^2)$ times (each time picking a brand new r; notice also that t is polynomial in k by assumption on ε) and taking the majority of the answers, we determine each x_i correctly with probability $1 - 1/k^2$.[1] Namely, by taking the majority

[1] This follows from the Chernoff's bound, stating that the probability the an average of t independent experiments is "ε-far" from its expectation is of the order $e^{-\Omega(t\varepsilon^2)}$. More formally, let Z_j be the indicator variable which is 1 if the j-test was correct, where $j = 1 \ldots t$. Then all Z_j are independent and $\mathbb{E}[Z_j] \geq \frac{1}{2} + \varepsilon$. Then, if $Z = \sum_j Z_j$, we have $\mathbb{E}[Z] \geq t(\frac{1}{2} + \varepsilon)$, and $\Pr[Z < t/2] \leq e^{\Omega(t\varepsilon^2)}$ by the Chernoff's bound.

vote we essentially "amplified" the success of B' and obtained an algortihm $B''(y, i)$ such that for *any* good x and any i,

$$\Pr[B''(y, i) = x_i \mid y = f(x)] \geq 1 - \frac{1}{k^2} \tag{6}$$

We now repeat B'' for all indices i, therefore defining $B(y)$ as running $B''(y, 1) \ldots B''(y, k)$. We get that the probability that *at least one x_i is wrong* is at most $k/k^2 = 1/k$, so B recovers the entire (good) x correctly with probability $1 - 1/k$, which is certainly non-negligible, contradicting the one-wayness of f.

4.2 General Case* (Technical, can be skipped)

Still, our assumption about the success of A with probability $\frac{3}{4} + \varepsilon$ is too much. We can only assume $\frac{1}{2} + \varepsilon$. It turns out the proof follows the same structure as the simplistic proof above, except the algorithm B'' will be defined more carefully.

Specifically, recall our assumption now is that

$$\Pr[A(f(x), r) = (x \cdot r) \mid x, r \leftarrow \{0, 1\}^k] > \frac{1}{2} + \varepsilon \tag{7}$$

where ε is non-negligible. As earlier, Equation (7) is equivalent to saying that the *expected value* of $Succ(x)$ (defined as in the previous section) is non-trivially greater than 1/2:

$$\mathbb{E}[Succ(x)] > \frac{1}{2} + \varepsilon \tag{8}$$

Similarly to the previous section, we call x "good" if $Succ(x) > \frac{1}{2} + \frac{\varepsilon}{2}$. Then the same averaging argument as earlier implies that

$$\Pr[x \text{ is good} \mid r \leftarrow \{0, 1\}^k] > \frac{\varepsilon}{2} \tag{9}$$

Thus, as in the previous case, it suffices to show how to invert good x, except the definition of "good" is considerably weaker than before: $Succ(x) > \frac{1}{2} + \frac{\varepsilon}{2}$ instead of a much more generous $Succ(x) > \frac{3}{4} + \frac{\varepsilon}{2}$.

As earlier, though, the way we construct our inverter B is by constructing a "bit predictor" $B''(y, i)$ which satisfies Equation (6), except "good" x is defined differently: for *any* good x and any i,

$$\Pr[B''(y, i) = x_i \mid y = f(x)] \geq 1 - \frac{1}{k^2} \tag{10}$$

Then B is defined as before to be $B''(y, 1) \ldots B''(y, k)$. Hence, we "only" need to define a new, more sophisticated B'' satisfying Equation (10).

The definition and the analysis of B'' form the heart of the Goldreich-Levin proof. The idea is the following. Recall, the algortihm B'' from the previous section was defined as follows: for some parameter t, B'' chose t random values $r_1 \ldots r_t \in \{0, 1\}^k$, and then output the majority of values of $A(y, r_j) \oplus A(y, r_j \oplus e_i)$, where j randges from 1 to t. The problem is that we can no longer argue that $\Pr_r[A(y, r) \oplus A(y, r \oplus e_i) = x_i] \geq \frac{1}{2} + \varepsilon$. A *wrong* argument to get a weaker, but still sufficient bound would be to say each value is correct with probability at least $\frac{1}{2}+\frac{\varepsilon}{2}$, so we succeed if either both are right or wrong, which happens with probability at least

$$\left(\frac{1}{2}+\frac{\varepsilon}{2}\right)^2+\left(\frac{1}{2}-\frac{\varepsilon}{2}\right)^2 \geq \frac{1}{2}+\frac{\varepsilon^2}{2}$$

The problem, of course, is that the success of A on *correlated* values r and $r \oplus e_i$ is not independent, and so we cannot just multiply these probabilities.

Instead, we have to built a more sophisticated predictor B''. As before B'' will choose t random values $r_1, \dots, r_t \in \{0,1\}^k$ (where we will determine t shortly). Now, however, B'' will also *guess* the correct value for $(x \cdot r_i)$. Specifically, B'' will choose t random bits $b_1, \dots, b_t \in \{0,1\}$, and will only work correctly, as explained below, if $x \cdot r_j = b_j$ for *all* $j = 1 \dots t$. This immediately loses a factor 2^{-t} in the success probability of B'', so we cannot make t too large. Technically, this also means that we will satisfy Equation (5) only conditioned on the event E stating that all the t guesses of B'' are correct: for *any* good x and any i,

$$\Pr[B''(y,i) = x_i \mid y = f(x), E \text{ is true}] \geq 1 - \frac{1}{k^2} \tag{11}$$

Luckily, this still suffices to prove our result if 2^{-t} is non-negligible (which it will be), since then

$$\Pr[B(y) = x \mid y = f(x)] \geq \Pr(E) \cdot \Pr[B(y) = x \mid y = f(x), E \text{ is true}] \geq 2^{-t} \cdot \left(1 - \frac{1}{k}\right)$$

The point, however, is that if B'' correctly guessed all the t paritites $x \cdot r_j$ (which we assume for now), then B'' also correctly knows 2^t parities of all the linear combinations of the r_j's. Concretely, for any non-empty subset $J \subseteq \{1 \dots t\}$, we let $r_J = \oplus_{j \in J}\, r_j$ and $b_J = \oplus_{j \in J}\, b_j$. Then

$$x \cdot r_J = x \cdot (\oplus_{j \in J}\, r_j) = \oplus_{j \in J}\, (x \cdot r_j) = \oplus_{j \in J}\, b_j = b_J$$

Now we will let B'' call A on 2^t values $A(y, r_J \oplus e_i)$, for all non-empty subsets $J \subseteq \{1 \dots t\}$, and output the majority of $b_J \oplus A(y, r_J \oplus e_i)$. The rational is that whenever A is correct on $r_J \oplus e_i$, the value

$$b_J \oplus A(y, r_J \oplus e_i) = x \cdot r_J \oplus x \cdot (r_J \oplus e_i) = x \cdot e_i = x_i,$$

as needed. The tricky part is to argue that for "small enough" t, B'' is correct with probability $1 - 1/k^2$ (once again, conditioned on E being true).

We do this as follows. Define an indicator random variable Z_J to be 1 if and only if $A(y, r_J \oplus e_i) = x \cdot (r_J \oplus e_i)$; i.e., if A is correct on $r_J \oplus e_i$. Then, B'' is incorrect if and only if a majority of Z_J are incorrect, i.e. $Z \overset{\text{def}}{=} \sum_{J \neq \emptyset} Z_J < 2^{t-1}$. But let us compute the expected value of Z. First, for any non-empty J, r_J is random in $\{0,1\}^k$, and thus, $r_J \oplus e_i$ is also random. Since x is good, this means $\mathbb{E}[Z_J] \geq \frac{1}{2}+\frac{\varepsilon}{2}$, so $\mathbb{E}[Z] \geq 2^{t-1}(1+\varepsilon)$. On the other hand, the probability that B'' failed is $\Pr[Z < 2^{t-1}]$.

Ideally, we would like to use the Chernoff's bound, like we did before, but we cannot do it, since the values Z_J are not independent. Luckily, there are *pairwise independent.*

This means than for any non-empty and distinct subsets I and J, the values r_I and r_J are independent, which means that $r_I \oplus e_i$ and $r_J \oplus e_i$ are independent, which in turn means that Z_I and Z_j are independent. For such pairwise independent random variables, it turns out we can apply the so called Chebyschev's inequality which states that

Lemma 2 (Chebyschev's inequality) *If $W = \sum_{j=1}^{T} W_j$, where W_j are pairwise independent indicator variables with mean at least p, then for any $\delta > 0$,*

$$\Pr[Z < T(p - \delta)] \leq \frac{1}{\delta^2 T}$$

We now apply this lemma to our $T = 2^t - 1$ variables Z_J by writing

$$\Pr[Z < 2^{t-1}] \leq \Pr\left[Z \leq (2^t - 1)\left(\left(\frac{1}{2} + \frac{\varepsilon}{2}\right) - \frac{\varepsilon}{2}\right)\right] \leq \frac{4}{(2^t - 1)\varepsilon^2}$$

By setting $t = \log(1+4k^2/\varepsilon^2) = O(\log(k/\varepsilon))$, we get $\Pr[B''(y,i) \neq x_i \mid E \text{ is true}] \leq 1/k^2$, as needed. The only thing to observe is that $2^{-t} = O(\varepsilon^2/k^2)$ is indeed non-negligible, since ε is non-negligible.

This concludes the proof of the Goldreich-Levin's Theorem.

5 Public Key Cryptosystem for One Bit

We will now look at how to make use of what we have at hand. We will not be terribly rigorous for the time being, and will proceed with the understanding that speculative adventures are acceptable. This time we will hand-wave a little, but will return to the problematic sections in the next lecture.

It seems intuitive that, if we have a hardcore bit, we *should* be able to send *one* bit of information with complete security in the Public Key setting. And we show exactly that.

- **Scenario**

  Bob(B) wants to send a bit b to Alice(A). Eve(E) tries to get b. Alice has a public key PK and a secret key SK hidden from everybody.

- **Required Primitives**

  1. TDP f will be the public key PK and its trapdoor information t will be Alice's secret key SK.
  2. Hardcore bit h for f. If needed, can apply Goldrecih-Levin to get it.

- **Protocol**

  B selects a random $x \in \{0,1\}^k$ and sends A the ciphertext $c = \langle f(x), h(x) \oplus b \rangle$.

- **Knowledge of the Concerned Parties before Decryption**

  B: b, x, c, f, h.

  E: c, f, h.

  A: c, t, f, h.

- **Decryption by A**

  x is obtained from $f(x)$ using the trapdoor t;

  $h(x)$ is computed from x;

  b is obtained from $(h(x) \oplus b)$ using $h(x)$.

- **Security from E**

  Intuitively, to learn anything about b, E must learn something about $h(x)$. But E only knows $f(x)$. Since h is hardcore, E cannot predict $h(x)$ better than flipping a coin, so b is completely hidden.

We note that this scheme is grossly inefficient. Even though it *seems* like we are using an elephant to kill an ant, look what we accoplished: we constructed the first secure public-key cryptosystem!

Having said this, we would still like to improve the efficiency of this scheme. Can we send arbitrary number of bit by using a single x above? More generally, can we extract a lot of random looking bits from a single x?

6 Public Key Cryptosystem for Arbitrary Number of Bits

We will try to generalize the system in a manner analogous to what we did with the S/Key system. Remember, there we published the value $y_0 = f^T(x)$, and kept giving the server the successive preimages of y_0. So maybe we can do the same thing here, except we will use *hardcore bits of successive preimages to make them into a good one-time pad!* Notice also that publishing $f^t(x)$ will still allow Alice to get back all the way to x since she has the trapdoor.

- **Scenario**

  Bob(B) wants to send a string $m = m_1 \dots m_n$ to Alice(A). Eve(E) tries to get "some information" about m. Alice has a public key PK and a secret key SK hidden from everybody.

- **Required Primitives**

  1. As before, TDP f will be the public key PK and its trapdoor information t will be Alice's secret key SK.
  2. Hardcore bit h for f. If needed, can apply Goldrecih-Levin to get it.

- **Protocol**

  B selects a random $x \in \{0,1\}^k$ and sends A the ciphertext $c = \langle f^n(x), G'(x) \oplus m \rangle$, where

  $$G'(x) = h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \ldots \circ h(x) \tag{12}$$

  Notice, $G'(x)$ really serves as a "computational one-time pad".

- **Knowledge of the Concerned Parties before Decryption**

  B: m, x, c, f, h.

  E: c, f, h.

  A: c, t, f, h.

- **Decryption by A**

  x is obtained from $f^n(x)$ using the trapdoor t by going "backwards" n times;

  $G'(x)$ is computed from x by using h and f in the "forward" direction n times;

  m is obtained from $(G'(x) \oplus m)$ using $G'(x)$.

- **Security from E**

  The intuition about the security is no longer that straightforward. Intuitively, if we let $p_1 = h(f^{n-1}(x)), \ldots, p_n = h(x)$ be the one-time pad bits, it seems like p_1 looks random given $f^n(x)$, so m_1 is secure for now. On the other hand, $p_2 = h(f^{n-2}(x))$ looks secure even given $f^n(x)$ and p_1 (since both can be computed from $f^{n-1}(x)$ and p_2 is secure even if $f^{n-1}(x)$ is completely known. And so on. So we get a very strange kind of "security". Even if the adversary knows $f^n(x)$ (which he does as it is part of c), and even if he somehow learns $m_1, \ldots, m_{i-1}$, which would give it $p_1, \ldots, p_{i-1}$, he still cannot predict p_i, and therefore, m_i is still secure. So we get this "next-bit security": given first $(i-1)$ bits, E cannot predict the i-th one. It is completely unclear if

  - "Next-bit security" is what we really want from a good encryption (we certainly want as least this security, but does suffice?) For example, does our system satisfy analogously defined "previous-bit security"?
  - Our scheme satisfies some more "reasonable" notion of security.

  The answers to these questions will come soon.

At this point, we turn our attention to Secret Key Cryptography based on symmetric keys. We would like to contemplate whether we could transcend Shannon's Theorem on key lengths.

7 Secret Key Cryptosystems

Recall, our main question in secret key encryption was to break the Shannon bound. Namely, we would like to encrypt a message of length n using a secret key of a much smaller size k, i.e. $k < n$ (and hopefully, $k \ll n$). We right away propose a possible solution by looking at the corresponding public-key example for encrypting many bits.

Recall, in the public key setting we used $G'(x)$ (see Equation (12)) as a "computational one-time pad" for m, where x was chosen at random by Bob. Well, now we can do the same thing, but make x the shared secret! Notice also that now we no longer need the trapdoor, so making f OWP suffices.

- **Scenario**

  Bob(B) wants to send a string $m = m_1 \dots m_n$ to Alice(A). Eve(E) tries to get "some information" about m. Alice and Bob share a random k-bit key x which is hidden from Eve.

- **Required Primitives**

  1. OWP f which is known to everyone.
  2. Hardcore bit h for f. If needed, can apply Goldrecih-Levin to get it.

- **Protocol**

  B sends A the ciphertext $c = G'(x) \oplus m$, where $G'(x)$ is same as in Equation (12).

  $$G'(x) = h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \dots \circ h(x) \tag{13}$$

  As before, $G'(x)$ really serves as a "computational one-time pad".

- **Knowledge of the Concerned Parties before Decryption**

  B: m, x, c, f, h.

  E: c, f, h.

  A: c, x, f, h.

- **Decryption by A**

  $G'(x)$ is computed from secret key x by using h and f in the "forward" direction n times;

  m is obtained from $(G'(x) \oplus m)$ using $G'(x)$.

- **Security from E**

  Again, the intuition about the security is not straightforward. Similar to the public-key example, it seems like we get what we called "next-bit security": given first $(i-1)$ bits of m (or of $G'(x)$), E cannot predict the i-th bit of m (or $G'(x)$). It is completely unclear if

  - "Next-bit security" is what we really want from a good encryption (we certainly want as least this security, but does suffice?) For example, does our system satisfy analogously defined "previous-bit security"?
  - Our scheme satisfies some more "reasonable" notion of security.

  Again, the answers to these questions will come soon,

Before moving on, we also make several more observations. First, notice that we could make n very large (in particular, much larger than k). Also, we can make the following optimization. Recall that in the public key scenario we also send $f^n(x)$ to Alice so that she can recover x. Now, it seems like there is no natural way to use it, so we really computed it almost for nothing... But wait, we could use $f^n(x)$ to make our one-time pad longer! Namely, define

$$G(x) = f^n(x) \circ G'(x) = f^n(x) \circ h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \ldots \circ h(x) \tag{14}$$

Now we can use $G(x)$ as the one-time pad for messages of length $n + k$, which is *always* greater than our key size k, even if $n = 1$! Indeed, we claim that our intuitively defined "next-bit security" holds for $G(x)$ as well. Indeed, for $i < k$, $f^n(x)$ is *completely random* (since x is random), so predicting the i-th bit based on the first $(i - 1)$ bits is hopeless. While for $i > k$ our informal argument anyway assumed that Eve knows $f^n(x)$ (it was part of the encryption). We will discuss later if it really pays off in the long run to use this efficiency improvement (can you think of a reason why it might be good to keep the same x for encrypting more than one message?)

However, using either $G(x)$ or $G'(x)$ as our one-time pads still has its problems that we mentioned above. Intuitively, what we really want from a "computational one-time pad" is that it really *looks completely random to Eve.* We now formalize what it means, by defining an extremely important concept of a *pseudorandom number generator.*

8 Pseudo Random Generators

Intuitively, a *pseudorandom number generator* (PRG) stretches a short random seed $x \in \{0,1\}^k$ into a longer output $G(x)$ of length $p(k) > k$ which nevertheless "looks" like a random $p(k)$-bit strings to any computationally bounded adversary. For clear reasons, the adversary here is called a *distinguisher.*

Definition 3 [Pseudo Random Generator] A deterministic polynomial-time computable function $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ (defined for all $k > 0$) is called a *pseudorandom number generator* (PRG) if

1. $p(k) > k$ (it should be stretching).
2. There exists no PPT distinguishing algorithm D which can tell $G(x)$ apart from a truly random string $R \in \{0,1\}^{p(k)}$. To define this formally, let 1 encode "pseudorandom" and 0 encode "random". Now we say that for any PPT D

$$|\Pr(\ D(G(x)) = 1 \mid x \xleftarrow{r} \{0,1\}^k\ ) - \Pr(\ D(R) = 1 \mid R \xleftarrow{r} \{0,1\}^{p(k)}\ )| < \mathsf{negl}(k)$$

$\diamondsuit$

We observe that we require the length of x be less than the output length of $G(x)$. This is done since otherwise an identity function will be a trivial (and useless) PRG. It should not be that easy! On the other hand, requiring $p(k) > k$ makes this cryptographic primitive quite non-primitive to construct (no pun intended).

Secondly, we are not creating pseudorandomness from the thin air. We are taking a *truly random seed* x, and stretch it to "computationally random" output $G(x)$. In other words, $G(x)$ is computationally indistinguishable from a random sequence (i.e., looks random), only provided that (much shorter seed) x is random.

9 Points to Ponder

We would like to conclude with some questions which are food for thought.

1. Is our public-key encryption really good?
2. What about the secret-key encryption?
3. Are $G'(x)$ and $G(x)$ (see Equations (13) and (14)) pseudorandom generators?
4. Can we output the bits of $G'(x)$ in "forward" order?
5. Is "next-bit security" enough to imply a true PRG?