1 Pseudorandom functions

To explain what a pseudorandom function is, we should first say a few words about (truly) random functions. To be specific let's talk about functions that map n bits into one bit, that is functions of the type $\{0,1\}^n \to \{0,1\}$. For now we will think of n as a security parameter, so "polynomial in n" is feasible but "exponential in n" is infeasible. Such a function can be uniquely specified by listing all its 2^n values $F(\mathtt{00\cdots0}), F(\mathtt{00\cdots1}), \ldots, F(\mathtt{11\cdots1})$. Since there are two choices for each value $F(x)$ there are 2^{2^n} possible functions. A random function R is a random variable whose distribution is uniform over this set of 2^{2^n} possible functions.

A random function R can be chosen by sampling the 2^n values $R(\mathtt{00\cdots0}), R(\mathtt{00\cdots1}), \ldots, R(\mathtt{11\cdots1})$ as uniform and independent random bits. This gives a natural implementation of R by a randomized program: The program first samples and stores 2^n random bits. Then on input x it outputs the x-th value in its memory. In the circuit model, an implementation of a random function $\{0,1\}^2 \to \{0,1\}$ might look like this: On input x_1x_2 and randomness $r_1r_2r_3r_4$, output

$$(x_1 \text{ AND } x_2 \text{ AND } r_1) \text{ OR } (x_1 \text{ AND } \overline{x_2} \text{ AND } r_2) \text{ OR } (\overline{x_1} \text{ AND } x_2 \text{ AND } r_3) \text{ OR } (\overline{x_1} \text{ AND } \overline{x_2} \text{ AND } r_4).$$

This type of implementation scales exponentially in n. Intuitively this should be unavoidable because any implementation of R must store 2^n independent random values $r_1, \ldots r_{2^n}$. In the circuit model this can be made precise: Any randomized circuit that implements a random function must have size exponential in n.[1] In fact, any circuit of substantially smaller size must compute a function that is statistically very far from a truly random function.

A pseudorandom function is a randomized function P that is "indistinguishable from random" but is computable by small randomized circuit. What does it mean for a function to be indistinguishable from random? Since a function is completely described by listing its values, we would like to say that the lists

$$(P(\mathtt{00\cdots0}), P(\mathtt{00\cdots1}), \ldots, P(\mathtt{11\cdots1})) \quad \text{and} \quad (R(\mathtt{00\cdots0}), R(\mathtt{00\cdots1}), \ldots, R(\mathtt{11\cdots1}))$$

are computationally indistinguishable. These lists are exponentially long, so merely reading the lists in order would take the distinguisher-circuit an infeasible amount of time.

To make sense of this idea we would like the distinguisher the to *query* the values of the function at any input x of its choice in any order that it likes, for example

$x := F(\mathtt{00000}), y := F(\mathtt{11111})$.
if $x == y$
    output $F(\mathtt{01110})$ XOR $F(x\overline{x}x\overline{x}x)$
else
    output $F(\mathtt{10101})$ AND $F(\overline{y}yyy\overline{y})$.

Distinguishers can be adaptive: queries can depend on answers to previous queries.

Circuits that can make queries are called oracle circuits. An oracle is a black box that allows the circuit to evaluate some function without knowing how that function is implemented. Formally, an

[1] We are talking about *stateless* implementations: Once the randomness is fixed the function is determined.

oracle circuit C^F has, in addition to its usual AND, OR, NOT gates, an F-gate that is meant to compute the function F. The F-gates are called *queries* and are counted towards the size of the circuit. Since the circuit graph is acyclic, the F-gates can always be evaluated in sequential order, and an oracle circuit can be viewed as the following game between an ordinary interactive circuit and an implementation of F:

Compute query x_1.
Receive value $F(x_1)$.
Compute query x_2 (which may depend on $F(x_1)$).
Receive value $F(x_2)$.
$\vdots$
Compute query x_q (which may depend on $F(x_1)$ up to $F(x_{q-1})$).
Receive value $F(x_q)$.
Compute output.

The *view* C^F consists of all the information that C learns from this interaction. This is the sequence of values $(F(x_1), F(x_2), \ldots, F(x_q))$. In general, if the circuit C takes additional inputs or randomness, these should also be included in its view.

When F is a randomized function, the view C^F is a random variable that takes different values depending on the randomness of F. Once the randomness of F is fixed the values of F at all inputs is determined and so is the view.

In the case of a truly random function R, the view D^R consists of the values $R(x_1), R(x_2), \ldots, R(x_q)$. These values are uniform and independent whenever the queries are distinct and identical when the queries are the same. So the view D^R can be simulated efficiently: If query x_i is new (i.e. $x_i \neq x_j$ for all $j < i$) then simulate $R(x_i)$ by an independent random bit, if not return the previous answer (the answer to x_j). A pseudorandom function should be indistinguishable from R in this sense.

Definition 1. A randomized function $P\colon \{0,1\}^n \to \{0,1\}^m$ is an (s,ε)-*pseudorandom function* if for every oracle circuit D of size at most s, the views of D^P and D^R are (s,ε)-*indistinguishable*, where R is a random function from $\{0,1\}^n \to \{0,1\}^m$.

The aim is to implement a pseudorandom function P by an efficient (randomized) circuit C. Recall that C takes two types of inputs: The input $x \in \{0,1\}^n$ and its internal randomness $K \in \{0,1\}^k$, which is called the *key* of the pseudorandom function. Different values of the key K give rise to different functions $P(x) = P_K(x) = C(K,x)$. The key $K \sim \{0,1\}^k$ is chosen at random but it is kept secret from the distinguisher: The distinguisher can ask to see values $P_K(x)$ on inputs x of its choice but it has no information about the choice of K beyond what it can deduce from these values.

2 How to construct pseudorandom functions

A pseudorandom function P_K can in particular be viewed as a pseudorandom generator: The seed is the key $K \in \{0,1\}^k$ and the output can be the list of any say $2k$ distinct evaluations $(P_K(x_1), \ldots, P_K(x_{2k}))$. By Definition 1 this is indistinguishable from a uniform random string of length $2k$.

It is remarkable that the following converse is also true: Any pseudorandom generator can be used to implement a pseudorandom function with arbitrarily long inputs. We will show this by induction on the input length n of the pseudorandom function.

We assume that the generator $G\colon \{0,1\}^k \to \{0,1\}^\ell$ is length-doubling, i.e. $\ell = 2k$. This can be achieved by the stretch-extension procedure from Lecture 2. Such a generator can be viewed as a pseudorandom function P_1 with one bit of input and k bits of output: The values $P(0)$ and $P(1)$ are the k left bits $G_0(K)$ and the k right bits $G_1(K)$ of G respectively, namely

$$P(b) = G_b(K), \quad b \in \{0,1\},$$

providing a base case for the induction.

Now suppose we have constructed a pseudorandom function P with $n-1$ bits of input (and k bits of output). Extending the input of P by one bit amounts to "stretching" the 2^{n-1} values $P(x)$ to 2^n pseudorandom values $P'(y)$, where $y \in \{0,1\}^n$. This can be done by another call to G:

$$P'(bx) = G_b(P(x)), \quad x \in \{0,1\}^{n-1}, b \in \{0,1\}. \tag{1}$$

Unwinding this inductive definition gives the following construction for any input length n.

The Goldreich-Goldwasser-Micali (GGM) pseudorandom function:

$$P(x_1 x_2 \dots x_n) = G_{x_1}(G_{x_2}(\cdots(G_{x_n}(K)\cdots)).$$

We will prove that P is a pseudorandom function by induction on n. The key is understanding the effect of transformation (1).

Lemma 2. *If P is an (s,q,ε)-pseudorandom function and G is an $(s+q(t+O(k)),\varepsilon')$-pseudorandom generator of size t then P' is an $(s-O((n+k)q^2),q,\varepsilon+q\varepsilon')$-pseudorandom function.*

The additional parameter q here refers to the number of oracle queries that the distinguisher makes: "(s,q,ε)-indistinguishable" means indistinguishable with advantage at most ε by a circuit of size at most s that makes at most q oracle queries. The number of queries can never exceed the circuit size, so q is by default upper bounded by s. This parameter plays an important role in the induction so it will be useful to track it separately from the size.

To prove that $P' = G_b \circ P$ is a pseudorandom function we need to compare the view of the distinguisher D' interacting with P' to the view of D' interacting with a truly random function $R'\colon \{0,1\}^n \to \{0,1\}^k$. To do this we'll look at the hybrid function $G_b \circ R$ where $R\colon \{0,1\}^{n-1} \to \{0,1\}^k$ is a truly random function.

Claim 3. *If P is (s,q,ε)-pseudorandom then $G_b \circ P$ and $G_b \circ R$ are $(s-q(t+O(k)),q,\varepsilon)$-indistinguishable.*

Proof. A distinguisher D that tells $G_b \circ P$ from $G_b \circ R$ can be turned into a distinguisher D' that tells P from R by applying G_b to every query. More precisely, D' simulates D. Whenever D makes query $y = bx$ to its oracle, D' makes query x to its oracle applies G_b to the answer. The view D'^F is identical to the view $D^{G_b \circ F}$ for any F, so D has the same distinguishing advantage as D' on the relevant oracles. The circuit D has to do all the work that D' does plus q additional evaluations of G and $O(k)$ extra work to select the correct half G_b so its size is larger by $q(t+O(k))$. □

Claim 4. *If G is an (s',ε')-pseudorandom generator then $G_b \circ R$ is an $(s'-O((n+k)q^2),q,q\varepsilon')$-pseudorandom function.*

To gain intuition about Claim 4 take $n = 2$ and consider a distinguisher that queries its oracle F at 000, 101, and 011. When $F = G_b \circ R$, the distinguisher sees the values $G_0(R(00))$, $G_1(R(01))$ and $G_0(R(11))$. Since $R(00)$, $R(01)$, and $R(11)$ are independent random values this is like seeing (part of) the output of three independent copies of G, which we would also expect to be pseudorandom. Here is a useful composition lemma that captures this intuition.

Lemma 5. *Let $X_1, \ldots, X_q$ and $Y_1, \ldots, Y_q$ be independent random variables. If X_i and Y_i are (s, ε)-indistinguishable for every i then $(X_1, \ldots, X_q)$ and $(Y_1, \ldots, Y_q)$ are $(s, q\varepsilon)$-indistinguishable.*

Before we do the proof of Claim 4 let's consider another example. Suppose the distinguisher now queries its oracle at 000, 101, and 100. The first and third value returned by $G_b \circ R$ are no longer independent: They are evaluations of the left and right half of G on the same seed 00. It would be convenient if we could forbid the distinguisher from making sequences of queries of this type. This can be achieved by forcing the distinguisher to make the queries in pairs $0x$, $1x$. If the distinguisher wants to know $F(bx)$, we ask it to remember both $F(0x)$ and $F(1x)$ for future reference. The view of such a distinguisher will always look like the sequence

$$F(0x_1), F(1x_1), F(0x_2), F(1x_2), \ldots, F(0x_q), F(1x_q) \tag{2}$$

with $x_1, \ldots, x_q$ all distinct.

Proof. Assume D distinguishes $G_b \circ R$ from a random function R'. We convert D into a distinguisher D^* that operates as follows: Whenever D queries its oracle at bx, D^* checks if it has made this query before and if so it returns its previous answer. If not, it queries its oracle at both $0x$ and $1x$ and uses the relevant answer. D^* is larger than D by $O((n+k)q^2)$ gates, which is the amount of circuit hardware it takes to implement the memorization of previous queries and answers.[2]

The view of D^* looks like sequence (2). When $F = G_b \circ R$ this is a sequence of q independent outputs of G, while when $F = R'$ these are q independent random strings. By Lemma 5, if D^*'s distinguishing advantage is $q\varepsilon'$ then the output of G is not (s', ε')-pseudorandom. □

To prove Lemma 2 we combine the two claims, setting s' to $s + q(t + O(k))$ (this will turn out to be a convenient choice). We now apply the lemma inductively. Assume G is an $(s + q(t + O(k)), \varepsilon')$-pseudorandom generator. When $n = 1$, by Claim 4 the function P is $(s - O((1+k)q^2), q, q\varepsilon')$-pseudorandom. Applying Lemma 2 we get that when $n = 2$, P is $(s - O(1 + 2 + 2k)q^2, q, 2q\varepsilon')$-pseudorandom. Continuing this argument inductively, we conclude that for general n, P is $(s - O(n(n+k)q^2), q, nq\varepsilon')$-pseudorandom. After renaming parameters we obtain the main theorem of this lecture.

Theorem 6. *If G is an (s, ε)-pseudorandom generator then the GGM function is an $(s - qt - O((n(n+k)q^2), q, nq\varepsilon)$-pseudorandom function.*

What does this mean? To get a sense let's ignore the contribution of the terms n, k, t of "polynomial" magnitude. If we set q to a bit below $\sqrt{s}$ then we get that P is about $(s, \sqrt{s}, \sqrt{s}\varepsilon)$-pseudorandom. Since the number of queries never exceeds circuit size, P is in particular $(\sqrt{s}, \sqrt{s}\varepsilon)$-pseudorandom. If, for example, $s = 2^{k/2}$ and $\varepsilon = 2^{-k/2}$ we get a $(2^{k/4}, 2^{-k/4})$-pseudorandom function F from a $(2^{k/2}, 2^{-k/2})$-pseudorandom generator G. There is some deterioration in the parameters: To obtain the same security guarantee we have to roughly double the key length.

While this level of security is quite reasonable, one weakness of the GGM construction is its efficiency: To calculate an output of P one needs to make n sequential calls to G. There are specific instantiations of G for which the implementation can be parallelized leading to better efficiency. In practice, however, pseudorandom functions built "from scratch" achieve much better efficiency/security tradeoffs: The AES construction, for example, is believed to be extremely secure.[3]

[2] In more reasonable models of computations like RAM programs D^* would be more efficient and the loss in security would be lower.

[3] AES is in fact a *pseudorandom permutation*: When the key is fixed, AES permutes the elements of $\{0,1\}^n$.

Proof of Lemma 5. To keep the notation simple let's prove it for $q = 2$. Assume that some D of size at most s distinguishes (X_1, X_2) and (Y_1, Y_2) with advantage more than 2ε. Then D must distinguish (X_1, X_2) and (X_1, Y_2) or it must distinguish (X_1, Y_2) and (Y_1, Y_2) with advantage more than ε. The two cases are completely analogous so we focus on the first one. The bits X_1 can be fixed to the value that maximizes the advantage of D, giving a distinguisher of size s between X_2 and Y_2. More formally, if

$$\Pr[D(X_1, X_2) = 1] - \Pr[D(X_1, Y_2) = 1] > \varepsilon$$

then by averaging

$$\mathrm{E}_{X_1}\left[\Pr_{X_2}[D(X_1, X_2) = 1] - \Pr_{Y_2}[D(X_1, Y_2) = 1]\right] > \varepsilon$$

so there must be some particular choice $X_1 = x_1$ for which $D(x_1, \cdot)$ distinguishes X_2 from Y_2. □

3 Chosen plaintext attacks

We can now consider the following candidate randomized encryption scheme based on a pseudo-random function $P\colon \{0,1\}^n \to \{0,1\}^m$:

$$Enc(K, M) = \begin{cases} \text{choose random } X \sim \{0,1\}^m \\ \text{output } (X, M + P_K(X)) \end{cases} \qquad Dec(K, (X, C)) = C + P_K(X).$$

Based on the intuition we developed in Lecture 2 we want to argue that (Enc, Dec) is message simulatable for two and more messages. It actually satisfies a stronger notion of security against so-called chosen plaintext attacks.

In message indistinguishability, Eve chooses some number of messages that should maximize her chance of breaking the scheme and gets to see all their encryptions. In a chosen plaintext attack, Eve can choose her messages *adaptively*. After she observes the encryption of her first message of choice, she chooses a second message whose encryption she gets to see, and so on.

While this type of attack may appear unrealistic it has precedent in the history of cryptography. Here is an excerpt from Wikipedia about the breaking of the Enigma Codes in World War II:

> Occasionally, when there was a particularly urgent need to break German naval Enigma, such as when an Arctic convoy was about to depart, mines would be laid by the RAF [Royal Air Force] in a defined position, whose grid reference in the German naval system did not contain any of the words (such as sechs or sieben) for which abbreviations or alternatives were sometimes used. The warning message about the mines and then the "all clear" message, would be transmitted both using the dockyard cipher and the U-boat Enigma network.

To define chosen plaintext attack (CPA) security we model Eve as a circuit interacting with an *encryption oracle*. The encryption oracle Enc_K holds its key K fixed but uses independent instantiations of its internal randomness X when it is called at different times. So if Eve asks to see two encryptions of the same message she is likely to observe two very different outcomes.

Definition 7. (Enc, Dec) is (s, q, ε)-universally CPA simulatable (by size t) if there exists a sampler Sim (of size t) such that for every size-s, q-query oracle circuit D, the view D^{Enc_K} and the output of Sim are (s, q, ε)-indistinguishable.[4]

[4] This is stronger than the standard definition of CPA-security that you will find in textbooks.

Theorem 8. *If P is an (s, q, ε)-pseudorandom function then (Enc, Dec) is $(s - O(mq), q, \varepsilon + \binom{q}{2} \cdot 2^{-n})$-universally CPA simulatable by size zero.*

The loss in security can be made arbitrarily small by choosing a sufficiently large value of n. For example, if $s = 2^{k/2}$ and $\varepsilon = 2^{-k/2}$ then q is at most $2^{k/2}$, so by choosing $n = 1.5k$ we get that (Enc, Dec) is $(2^{k/2}, 2 \cdot 2^{-k/2})$-secure.

To prove Theorem 8 we pretend, as usual, that the pseudorandom function P behaves like a truly random function R, argue security for this "ideal" encryption scheme, then analyze the security loss in replacing R by P. To do this we consider the following hybrid encryption scheme in which P is replaced by a truly random function $R\colon \{0,1\}^n \to \{0,1\}^m$, whose description is the "key" of the encryption scheme:

$$Enc^R(M) = \begin{cases} \text{choose random } X \sim \{0,1\}^m \\ \text{output } (X, M + R(X)). \end{cases}$$

It should by now be clear that if P is an (s, q, ε)-pseudorandom function then the views D^{Enc_K} and D^{Enc^R} are ε-indistinguishable for any $(s - O(mq))$-size, q-query circuit D. (The PRF-distinguisher D' emulates the CPA-distinguisher D; whenever D queries its oracle at M, D' queries its oracle F at a random X and forwards $(X, M + F(X))$ as the answer.)

To finish the analysis we need to show how to simulate the view D^{Enc^R} without knowing R. Since we expect different queries to Enc^R to result in independent random ciphertexts, it is natural to consider a simulator Sim that outputs uniform random bits. This is indeed true if all the X's generated in the q different calls to Enc^R are distinct. The probability of getting a collision among the X's is at most $\binom{q}{2} \cdot 2^{-n}$: There are $\binom{q}{2}$ pairs, each of which collides with probability 2^{-n}. In summary, the random variables Enc^R and Sim are identically distributed unless an event of probability at most $\binom{q}{2} \cdot 2^{-n}$ occurs. It follows that (Enc^R, Dec^R) is $(\infty, \infty, \binom{q}{2} \cdot 2^{-n})$-CPA simulatable and the theorem follows by the usual hybrid argument.

4 Pseudorandom functions and learning

The utility of pseudorandom functions extends well beyond CPA-secure encryption. You can use one to keep the data on your hard drive encrypted yet pay only a small overhead in access. When you throw it away, all you need to do is wipe out the secret key and the data will be forever lost. In the next few lectures we will see how to use pseudorandom functions for identification, authentication, and achieving even stronger forms of secure encryption.

The significance of pseudorandom functions in fact extends far beyond cryptography. They are the ultimate examples of behaviours that cannot be learned — not because of lack of information but because of limitations in computational power. In the model of probably approximately correct (PAC) learning, a learning algorithm can observe input-output samples of the form $(x, f(x))$, where f is some efficient but unknown function. For example, f could be the "circuit of the mind" that takes as input some visual scene x and outputs 1 if x has a dog in it. The objective of the learning algorithm is, after being trained on some sequence of samples $(x_1, f(x_1)), \ldots, (x_2, f(x_q))$, to make a prediction for $f(x)$ on some previously unseen sample x.[5]

[5]These ideas were also worthy of a Turing Award (Leslie Valiant in 2010).

We can model this learning process as a game in which the following type of learner L interacts with an unknown concept f:

Training phase:
Query sample x_1. Receive example $f(x_1)$.
Query sample x_2. Receive example $f(x_2)$.
$\vdots$
Query sample x_q. Receive example $f(x_q)$.

Prediction phase:
Choose test sample x different from x_1 up to x_q.
Output a prediction y for $f(x)$.

The learner succeeds if y equals $f(x)$. This is an extremely weak notion of learning: Not only does the learner get to choose which examples to see, but she also chooses what to be tested on (as long as she is disallowed to see the answer to the exam question itself)!

Theorem 9. *Assume $P_K \colon \{0,1\}^n \to \{0,1\}$ is an $(s, q+1, \varepsilon)$-pseudorandom function. For every learner L of size at most s there is a key K such that L^{P_K} succeeds with probability at most $1/2 + \varepsilon$.*

In words, the learner cannot do any better at passing the test than guessing the answer at random!

Proof. We prove the contrapositive. Suppose L^{P_K} succeeds with probability $1/2$ for every K. Then it succeeds with this probability for a random K. Let D be a distinguisher that simulates L, making the same q training queries, but then makes one more query at x and outputs L's exam grade: 1 if $y = f(x)$ and 0 if not. The accepting probability of D^P then equals the probability that L^P succeeds, which is $1/2 + \varepsilon$. On the other hand, if R is a truly random function, the value $R(x)$ is independent of everything the learner has seen, so it is independent of his prediction y. The probability that they are equal is exactly $1/2$, so D^R outputs 1 with probability exactly $1/2$. It follows that D distinguishes P and R, so P is not pseudorandom. □

In conclusion, pseudorandom functions are examples of functions that cannot be learned from examples, provided that they exist. What if we lived in a world in which pseudorandom functions didn't exist? Then all functions would be learnable up to very high precision even from completely random training data. That would be a paradise-world for machine learners (even though they are already doing quite well in the real world).