University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 14

1 Private-Key Encryption, Revisited

To recap the end of last lecture, we gave a definition of *security in the sense of indistinguishability* (or an *indistinguishable* encryption scheme).

Definition 1 *An encryption scheme* $\Pi = (\mathcal{E}, \mathcal{D})$ *is* secure in the sense of left-or-right indistinguishability *if, for all* PPT *adversaries* A *the following is negligible:*

$$\left|\Pr[sk \leftarrow \{0,1\}^k; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,sk}(\cdot,\cdot)}(1^k) = b] - 1/2\right|.$$

(We no longer restrict the number of times the adversary may query LR*; of course, since* A *is a* PPT *algorithm it can query* LR *at most polynomially-many times.)*

We also noted that this definition implies security when multiple messages are encrypted, and implies security against chosen plaintext attacks. *You should be able to prove both of these statements more formally.*

It will be instructive to see examples of schemes that are *not* secure under this definition (to get a feel for the definition...). For example, take the one-time pad encryption scheme. Can you show that this scheme does not satisfy the above definition by showing an explicit adversary A for which

$$\Pr[sk \leftarrow \{0,1\}^k; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,sk}(\cdot,\cdot)}(1^k) = b]$$

is always 1, and the adversary only makes two queries to LR?

You should also be able to show that *no deterministic (stateless) encryption scheme can satisfy the above definition* (why?). So our next challenge will be to consider randomized encryption schemes. It also seems that PRGs are not quite enough to obtain an encryption scheme satisfying this definition; in the next lecture we will introduce some more powerful machinery that will enable a solution.

2 Randomized Encryption Schemes

Until now, all examples of encryption schemes that we have seen ("classical" ciphers, one-time pad, PRG construction) have been deterministic. Yet we saw above that no deterministic (stateless) encryption scheme can be secure in the sense of indistinguishability. Let's briefly introduce the notion of randomized encryption schemes. Here, the encryption algorithm might choose random bits when encrypting a message. In particular (if these random bits are to be useful), there will now be *many* ciphertexts corresponding to any single message, and encryption is no longer one-to-one. Yet the decryption algorithm must still somehow recover the original message all the time. Perhaps this sounds more difficult than it really is; we give a small example here.

Consider the following encryption scheme: the shared key is $s = s_1 \cdot s_2$, where $|s_1| = |s_2| = k$. To encrypt message m, choose a random bit r and then output $C = (r, m \oplus s_r)$. To decrypt ciphertext $C' = (r', c')$, simply compute $m' = c' \oplus s_{r'}$. Note that decryption always succeeds.

A word on notation: in describing an experiment (inside a probability expression) I have been careful to distinguish between deterministic events and probabilistic ones. So, encrypting a message m using a deterministic encryption scheme would be written $C = \mathcal{E}_s(m)$ while encryption this message using a probabilistic encryption scheme would be written $C \leftarrow \mathcal{E}_s(m)$.

What might the advantage of this encryption scheme be over the one-time pad? (After all, we share $2k$ bits but only send k-bit messages in the above scheme.) Well, this scheme "kind of" gives security when two messages are sent. In particular, if r is *different* for two messages that get encrypted (which occurs with probability 1/2) then the scheme achieves "perfect secrecy" for these two messages, since they are both encrypted with different "one-time pads". Of course, the scheme as a whole is *not* perfectly secret, since with probability 1/2 the bs are the same in which case information about the two messages is leaked. But since we are, in general, interested in computational secrecy only (where we require that the adversary not be able to distinguish between encryptions of two different messages with high probability) this approach might be plausible.

It will be worthwhile to investigate what sort of security guarantees we can give for this scheme. Given an adversary A, let's look at:

$$\left|\Pr[s \leftarrow \{0,1\}^{2k}; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,s}(\cdot,\cdot)}(1^k) = b] - 1/2\right|,$$

where we now restrict our adversary to making only two queries to LR. Note that there are two possibilities for the two queries to LR: either the same r is used each time or not, and each case occurs with probability 1/2. Let r_1 denote the bit used to answer the first query to LR and let r_2 denote the bit used to answer the second query. Thus:

$$\begin{aligned}
&\left|\Pr[s \leftarrow \{0,1\}^{2k}; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,s}(\cdot,\cdot)}(1^k) = b] - 1/2\right| \\
&= \Big|1/2 \cdot \Pr[s \leftarrow \{0,1\}^{2k}; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,s}(\cdot,\cdot)}(1^k) = b \mid r_1 = r_2] \\
&\quad + 1/2 \cdot \Pr[s \leftarrow \{0,1\}^{2k}; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,s}(\cdot,\cdot)}(1^k) = b \mid r_1 \neq r_2] - 1/2\Big|.
\end{aligned}$$

The important thing to note is that, when $r_1 \neq r_2$, the adversary *has no information about the value of* b (because two different one-time pads are used). Thus, $\Pr[s \leftarrow \{0,1\}^{2k}; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,s}(\cdot,\cdot)}(1^k) = b \mid r_1 \neq r_2] = 1/2$, and

$$\begin{aligned}
&\left|\Pr[s \leftarrow \{0,1\}^{2k}; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,s}(\cdot,\cdot)}(1^k) = b] - 1/2\right| \\
&= \left|1/2 \cdot \Pr[s \leftarrow \{0,1\}^{2k}; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,s}(\cdot,\cdot)}(1^k) = b \mid r_1 = r_2] - 1/4\right| \\
&\leq 1/4,
\end{aligned}$$

where the final inequality holds because probabilities are always between 0 and 1. The point is that the adversary does not guess b correctly with probability 1 (as was the case if we allow the adversary to query LR twice in the one-time pad scheme), so at least we are making some progress!

Of course, this is not negligible, but let's see how far we can extend this approach. What if, instead of sharing two keys s_1, s_2 the parties shared N keys $s_1, \ldots, s_N$ (now the total size of the shared key s is Nk bits)? Now, the probability that the same key is used in both encryptions is reduced to $1/N$. You should check that this gives (using the same derivation as above):

$$\left| \Pr[s \leftarrow \{0,1\}^{Nk}; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,s}(\cdot,\cdot)}(1^k) = b] - 1/2 \right| \leq 1/2N.$$

Thus, to make this negligible (in k) we just set $N = 2^k$. And we are done... or are we?

The first thing to notice is that setting $N = 2^k$ is not only impractical, but is not even technically allowed. The encryption and decryption algorithms need to run in polynomial time (after all, why should the sender and receiver be more powerful than the adversary?) and thus cannot even store a key of length $k \cdot 2^k$. So we have to improve this somehow. Furthermore, we seem to have gone through an awful amount of work and we still haven't achieved our goal: the scheme above may protect against an adversary making two queries to the LR oracle, but we want security against an adversary making any polynomial number of queries to LR!

To recap, what we'd like to do is to be able to access $N \stackrel{\text{def}}{=} 2^k$ different (shared) random keys, yet not have to store them all. Perhaps we can do better by sharing *pseudorandom* keys instead of *random* keys. How might this look? Well, let's imagine we have a PRG G that stretches k-bit seeds to kN-bit pseudorandom strings. Then the parties could share a seed s of length k and use G to derive keys $s_1, \ldots, s_N$ (when needed). There are two problems with this approach: (1) it will require exponential time to compute $G(s)$ (after all, $G(x)$ has exponentially-many bits!) and this is not allowed; (2) we stated earlier that our PRG constructions were guaranteed to be secure *only* when they stretch their inputs by a *polynomial* amount. But here we need to stretch by an exponential amount!

In fact, we will need a primitive which is stronger than a PRG.

3 Pseudorandom Functions and Permutations

A PRG was nice, because it allowed us to share random keys of small size and generate shared pseudorandom keys of large size. But even nicer than having shared access to a pseudorandom *string* would be shared access to a pseudorandom *function*. Let's look at what this might mean.

For simplicity, we consider functions with fixed input and output length. Consider a function $F : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$ in which the first input to F is called the *key* and the second input to F is simply called the *input*. k is the *key length*, m is the *input length* and n is the *output length* of F. F is in fact a *keyed function* in the following sense: fixing a particular key $s \in \{0,1\}^k$ defines a function $F_s : \{0,1\}^m \rightarrow \{0,1\}^n$ as follows: $F_s(x) = F(s, x)$. So you can view choosing a particular key as choosing a particular function from some large set of functions.

Now, informally speaking, our PRGs had the property that their outputs "looked random" when evaluated on a random seed. The analogous property we would like from F is that it should "look like a random function" when a random key is chosen. But what exactly is a random function?

A random function can be viewed in the following way: if you give the function an input $x \in \{0,1\}^m$ that it has not seen before, the function picks a random $y \in \{0,1\}^n$, stores (x, y) for future reference, and outputs y. If you later ask the function an input x which it *has* seen before, it finds the appropriate pair (x, y) and returns y (so the function always reports the same answer if you ask it the same query).

We can also view all these choices as being made in advance, instead of upon receiving a query. Thus, for every possible input $x_i \in \{0,1\}^m$ the function chooses a random $y_i \in \{0,1\}^n$ and stores (x_i, y_i) in some huge table. In fact, if we order the xs lexicographically, it is enough to just store the ordered list $y_1, \ldots, y_n$.

How much space would this actually take if we were to implement this? Well, each y_i requires n bits and there are 2^m of them to store, for a total of $n \cdot 2^m$ bits of storage just to represent a random function. Clearly, it is infeasible to store a truly random function for moderate values of m, n. But we can certainly imagine such a thing existing as an *oracle*. (In fact, you could imagine implementing a random function if the answers were chosen "on-the-fly" instead of in advance. But there would be some bound on the number of different queries you could ask it, depending on the amount of storage that is available. If you want to *share* a random function, however, the answers must be decided upon in advance. And since you don't know what the queries are going to be, this requires sharing the full $n \cdot 2^m$ bits.) We denote the set of all functions from $\{0,1\}^m$ to $\{0,1\}^n$ by $\mathsf{Rand}^{m \to n}$.

We will now define a formal notion of a pseudorandom function (PRF). Recall our informal definition that a PRF "looks like" a random function; more formally, a PPT adversary cannot tell them apart with too high a probability. In what way? Well, say we give an adversary a black box and tell it that the box implements either a random function or a pseudorandom function and ask it to determine which one. Then the probability that the adversary guesses "pseudorandom" should be the same whether the function is actually pseudorandom or not. More formally, a function $F : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n$ is a (t, ϵ)-PRF if for all algorithms A running in time at most t the following holds:

$$\left| \Pr[s \leftarrow \{0,1\}^k : A^{F_s(\cdot)} = 1] - \Pr[F \leftarrow \mathsf{Rand}^{m \to n} : A^{F(\cdot)} = 1] \right| \leq \epsilon.$$

(Note that here we have k a fixed value, so the above is not a function of a parameter k as was the case in previous lectures. Similarly, instead of making A run in polynomial time we have instead fixed an upper bound t for the adversary's running time. The intention was to simplify things a little bit!) In words, the left expression consider the experiment in which a random key s is chosen, and A is given oracle access to the function F_s. In the right experiment, a completely random function F from m bits to n bits is chosen, and A is given oracle access to F. We claim that A cannot distinguish between these two cases with probability better than ϵ.

We always implicitly assume that it is "easy" to compute $F_s(x)$, given both s and x. But the above definition says, in particular, that it is "hard" to predict the value of $F_s(x)$ if s is unknown and randomly chosen (and even if x is known).

Although we did not give a formal, complexity-theoretic definition of one-way functions, we hope the reader will see how such a definition would proceed.[1] And we therefore state the following theorem with little justification and no proof.

Theorem 1 *Pseudorandom functions exist if and only if one-way functions exist.*

3.1 PRPs

We can define an analogous notion of a pseudorandom permutation (PRP). Here, we consider functions $P : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ such that, for all $s \in \{0,1\}^k$, we have that P_s is a permutation over $\{0,1\}^m$ (and call functions with this property *keyed permutations*). Also, we define Perm^m as the set of all permutations on $\{0,1\}^m$. In a manner completely analogous to the above, we say that P is a (t, ϵ)-PRP if it is a keyed permutation such that for all algorithms A running in time at most t we have:

$$\left|\Pr[s \leftarrow \{0,1\}^k : A^{P_s(\cdot)} = 1] - \Pr[P \leftarrow \mathsf{Perm}^m : A^{P(\cdot)} = 1]\right| \leq \epsilon.$$

Finally, we mention that once we have a keyed permutation it is natural to talk about taking the inverse of P_s, for a particular s. Again, in this case we would hope that it be "easy" to compute $P_s^{-1}(y)$, given s and y (note that this is not necessarily the case — as we have seen already in this class, there are plenty of permutations for which the forward direction can be computed easily but the inverse cannot be). But of course, we might hope that it be "difficult" to predict $P_s^{-1}(y)$ when s is unknown. In fact, there are secure PRPs for which it is "easy" to predict the inverse of $P_s^{-1}(y)$, for a particular choice of y and even for a randomly chosen s.

So this requires another definition. We say that P is a *strong* (t, ϵ)-PRP if it is a keyed permutation such that for all algorithms A running in time at most t we have:

$$\left|\Pr[s \leftarrow \{0,1\}^k : A^{P_s(\cdot), P_s^{-1}(\cdot)} = 1] - \Pr[P \leftarrow \mathsf{Perm}^m : A^{P(\cdot), P^{-1}(\cdot)} = 1]\right| \leq \epsilon.$$

Note that (in each experiment) we now give A access to two oracles: one representing the forward evaluation of the permutation and one representing the inverse.

As mentioned earlier, not every PRP is automatically a strong PRP. We state this here as a theorem:

Theorem 2 *Assuming the existence of one-way functions, there exists a PRP P which is definitely not a strong PRP.*

Finally, we state the following theorem for reference:

Theorem 3 *Strong PRPs exist if and only if one-way functions exist.*

[1] For the very interested reader, we point out that the complexity-theoretic definition can no longer consider functions F with input length; instead, the input length must depend on the key length.

3.2 Block Ciphers

What is done in practice? In practice, people use *block ciphers.* These block ciphers are keyed functions, just as described above. Famous examples of block ciphers include DES, IDEA, triple-DES, and AES. As an example, DES is really a keyed permutation $\mathsf{DES} : \{0,1\}^{56} \times \{0,1\}^{64} \rightarrow \{0,1\}^{64}$. That is, the key length of DES is 56 bits, and once a key s is fixed, DES_s is a permutation over $\{0,1\}^k$. Furthermore, it is efficient to compute and to invert DES if you know the key.

When block ciphers are used in an application, it is important to ask what properties of the cipher are assumed. For example, does security rely on the fact that the cipher is a PRP or does it require the stronger assumption that the cipher is a strong PRP? On the one hand, it is reasonable to assume that DES or AES is a strong PRP (ciphers are designed with this in mind). On the other hand, this is a stronger assumption, so basing the security of your scheme on a weaker assumption is preferable. Finally, some "off-the-shelf" ciphers may be PRPs but not string PRPs.

Finally, we mention that the security of block ciphers (as opposed to what we have been aiming for in this class) is largely heuristic. In other words, we have no particular reason to believe that AES is a good block cipher, other than the fact that it was designed by experts and studied intensively by cryptographers for the past 2 years. On the other hand, we could (by Theorem 3) construct a strong PRP based on the hardness of factoring. The hardness of factoring is backed up by 300 years of intense scrutiny by many, many people who were experts in a variety of fields. The **huge** disadvantage of this approach is that the resulting function would be orders of magnitude slower than DES or AES, and therefore not practical for use.

4 Private-Key Encryption Using PRFs

We stress that PRFs and PRPs alone are not enough to immediately give secure encryption schemes; some care is needed. For example, consider the "natural" encryption scheme in which P is a PRP (with efficient inversion) and the sender and receiver share a key s in advance. To encrypt message m, the sender computes $C = P_s(m)$ and sends this to the receiver. To decrypt, the receiver computes $m = P_s^{-1}(C)$ (since P is a permutation, decryption always succeeds). However, while this is secure against ciphertext only attacks (when a single message is encrypted) it does *not* satisfy our notion of indistinguishability (in fact, it cannot since this encryption scheme is deterministic).

Can you develop an indistinguishable encryption scheme based on PRFs??