Notes #5

We will now continue discussing security of shared-private-key cryptosystems, or session protocols. Last time we defined integrity, or "unchangeability security". In practice, for most people most of the time, this is much more important than privacy. You may not be happy if an adversary were to learn the balance of your bank account, but you would be much less happy if he were to transfer its contents to his own account. You may wish to keep your medical records secret, but it is much more important that an adversary should not be able to change the medication that has been prescribed for you.

We will now define privacy, or "indistinguishability security". We will later define "total security" as unchangeability security plus indistinguishability security (integrity plus privacy).

We consider an adversary ADV – either probabilistic polynomial time, or polynomial size, in the security parameter n, the length of the key. For each n, the adversary will compute, or have built in, a number i indicating that ADV is trying to learn something about piece i. ADV will choose all the other pieces. That is, ADV will be interacting with A and B working on a randomly chosen key.

Definition of *privacy* or *indistinguishability security*
Let (ENC, DEC) be a shared-private-key cryptosystem (that is, session protocol). We will define privacy, or indistinguishability security in the "nonuniform adversary" setting. The adversary D will be a polynomial size family of circuits $\{D_1, D_2, \ldots\}$, where D_n is the adversary for security parameter n (that is, usually, key size n). The adversary will choose all of the pieces to be encrypted except for, say, the i-th piece. He will choose each such piece after seeing the encryption of the preceding pieces, until he gets to the i-th piece. His goal is to "learn something" about the i-th piece. We will model this by allowing the adversary to choose two possible pieces, and then give him an encryption (at random) of one of them; he will try to guess which one has been encrypted. Before making this guess, he will continue, for a polynomial amount of time, to choose pieces and see encryptions of them. When he is done doing this, he is allowed to get some information from B. That is, he is allowed to now create a string and send it to B. The adversary then sees if, and when, B outputs FAIL. Then the adversary has to guess which of the two pieces were encrypted as piece i. (The reader may ask why we don't allow the adversary to see more information about what B outputs. The reason is that the adversary may as well assume that as long as B does not output FAIL, he outputs (in order) the correct pieces that A encrypted; if this is not the case, then the more important property of unchangeability security has already been broken. Whether or not the reader approves of this definition, at least there should be no problem with our definition of "total security".)

We will define the syntax of D_n, and the experiment involving D_n, and the associated probability $q_D(n)$, all at once and somewhat more intuitively than in our definition of an adversary in the last lecture.

Firstly, D_n chooses, or has associated with it, an integer i; D_n will be trying to learn what the i-th piece of the message is. (We probably should use the more rigorous notation i_n.)
A random n-bit k is chosen, but D_n doesn't see it.

D_n chooses piece m_0 (of length $z(n)$) and sees e_0, a (possibly randomized) encryption of m_0 by *ENC* using key k;
then D_n chooses piece m_1 and sees an encryption e_1;
this continues through piece m_{i-1}.

Then D_n chooses two pieces m^0 and m^1.
A random bit b is chosen but D_n doesn't see it, and then D_n sees an encryption e_i of m^b.

Then D_n chooses piece m_{i+1} and sees an encryption e_{i+1};
then D_n chooses piece m_{i+2} and sees an encryption e_{i+2};
this continues for a polynomial amount of time.
Then D_n computes a polynomial length string α.
D_n then learns something about the result of applying *DEC* to α using key k, that is, D_n learns if, and for which piece, B would output FAIL.

Then D_n outputs b', a guess at b.
(Of course, each time D_n makes a choice, he may base his decision on *everything* he has seen so far.)

Define $q_D(n) =$ **probability**$(b' = b)$. Note that the randomness in this experiment includes the choice of k and b, as well as all the randomness, if any, used by ENC (and by the adversary if it is a probabilistic algorithm).

Definition: We say a shared-private-key encryption scheme (or session protocol) satisfies *privacy* (or is *indistinguishability secure*) if for every adversary $D = \{D_n\}$ as described above, $q_D(n) \leq \frac{1}{2} + \frac{1}{n^c}$ for each c and sufficiently large n.
End of Definition of *privacy* or *indistinguishability security*

This definition is stronger and more robust than it might at first appear. For example, it might appear that it could help the adversary if we let him choose which piece he is going to decrypt based on the encryptions he sees, rather than force him to choose i in advance. However it is possible to show that if a scheme is secure in the sense we defined above, then it is also secure in this stronger sense (Exercise!). It is also possible to show that it can't help an adversary to be able to send (supposedly) encrypted pieces to B early on, before (or while) he is choosing pieces for A to encrypt, seeing if and when FAIL occurs (Exercise!). One might ask at this point if was really necessary in the definition to allow the adversary to send stuff to B at all, observing possible failure. It turns out that this part of the definition *is* necessary, and we leave this as an exercise as well. We also leave it as an exercise to show that it would not weaken the definition if we forced the adversary to choose m_0 and m_1 so that they differed only in exactly one bit.

Theorem: Assuming the function generator F is pseudo-random, Cryptosystem I (defined in the last lecture) satisfies privacy.

Proof Outline: Note that B never outputs fail, so we need only consider an adversary $D = \{D_n\}$ that only gets information from seeing the encryptions of pieces. Let $\{q_D(n)\}$ be as defined above, and assume that for infinitely many n, $q_D(n) > \frac{1}{2} + \frac{1}{n^c}$; fix such an n. We will construct a distinguisher for F on key length n. Let $i_n = i$ be the index of the piece that D_n is trying to guess the decryption of.

Consider an alternative experiment where instead of using F_k for random k to form the encryptions $e_j = m_j \oplus F_k(\bar{j})$, we use a randomly chosen function f, so that $e_j = m_j \oplus f(\bar{j})$. In this case, the only information the adversary sees about b is $m^b \oplus f(\bar{i})$, where none of the other encryptions

involve $f(\bar{i})$. So in this case the adversary would get no information at all about b, and so his success at guessing b would be $1/2$. [1]

We therefore propose the following adversary D'_n for breaking F. Given a (black box for a) function $f : \{0,1\}^n \to \{0,1\}^n$, D'_n simulates both A and D_n; whenever D_n wants an encryption of a j-th piece m_j, D'_n creates $e_j = m_j \oplus f(\bar{j})$. Also, D'_n simulates the person choosing the random bit b. So at the end of the simulation, D'_n knows whether or not b', the bit output by the simulated D_n, is equal to b; if it is then D'_n accepts, otherwise D'_n rejects.

The probability D'_n accepts when f is chosen to be F_k for a random k is exactly $p(n) > \frac{1}{2} + \frac{1}{n^c}$. The probability D'_n accepts when f is chosen completely randomly is exactly $1/2$. So the difference between these two probabilities is at least $\frac{1}{n^c}$. □

We now consider some other attempts to create session protocols satisfying privacy. The following system uses probabilistic encryption, but no history need be remembered for encryption or decryption.

Cryptosystem III. Let F be a (hopefully pseudo-random) function generator.

For an n-bit k and n-bit message pieces $m_0, m_1, m_2, \ldots$, define the $2n$-bit encryptions $e_i = [r_i, m_i \oplus F_k(r_i)]$, where the $\{r_i\}$ are randomly chosen n bit strings.

DEC works in the obvious way.

Theorem: If the function generator used in Cryptosystem III is pseudo-random, then the system satisfies privacy.

Proof: Exercise.

Cryptosystem IV. Let F be a (hopefully pseudo-random) *permutation* generator.

For an n-bit k and n-bit message pieces $m_0, m_1, m_2, \cdots$, define the n-bit encryptions $e_i = F_k(m_i)$.

DEC works in the obvious way, namely for each i we decrypt e_i by computing $m_i = F_k^{-1}(e_i)$.

Theorem: No matter how the permutation generator F is chosen, Cryptosystem IV does *not* satisfy privacy.

Proof: The intuition is that an adversary can tell if a message piece repeats, by seeing if the encryption repeats.

More formally, the adversary chooses $i = 1$, chooses m_0 to be anything, chooses $m^0 = m_0$ and $m^1 \neq m_0$; if $e_0 = e_1$ then he outputs 0, otherwise he outputs 1. It is easy to see that $q(n) = 1$. □

Cryptosystem V. Let F be a (hopefully pseudo-random) permutation generator.

For an n-bit k and n-bit message pieces $m_0, m_1, m_2, \ldots$, define the n-bit encryptions $e_i = F_{F_k(\bar{i})}(m_i)$.

DEC works in the obvious way, namely for each i we decrypt e_i by computing $m_i = F^{-1}_{F_k(\bar{i})}(e_i)$.

Theorem: If the permutation generator used in Cryptosystem V is pseudo-random, then the system satisfies privacy.

Proof: Exercise.

[1] Technically, this will be only true if n is sufficiently large that the (polynomial in n) number of encrypted pieces that D_n sees is less than 2^n.

The following cryptosystem uses cipher-block chaining. It is often asserted in the informal literature that it satisfies privacy, but according to our definition it does not.

Cryptosystem VI. Let F be a (hopefully pseudo-random) permutation generator.

For an n-bit k and n-bit message pieces $m_0, m_1, m_2, \ldots$, define the n-bit encryptions $e_0, e_1, e_2, \ldots$ as follows:
$e_{-1} = F_k(\overline{0})$;
$e_i = F_k(e_{i-1} \oplus m_i)$ for $i \geq 0$.

DEC works in the obvious way.

Theorem: No matter what permutation generator is used, Cryptosystem VI does *not* satisfy privacy.

Proof: The adversary chooses $i = 2$, chooses m_0 to be anything, sees e_0, chooses $m_1 = e_0$, sees $e_1 = F_k(\overline{0})$, chooses $m^0 = e_1$ and $m^1 \neq m^0$ and sees e_2; if $e_2 = e_1$ then he outputs 0, otherwise he outputs 1. It is easy to see that $q(n) = 1$. □

The reader may complain that the adversary just described doesn't seem very harmful, since he is just learning something about uninteresting message pieces. However it is often the case that what at first appears to be a minor insecurity becomes, with greater inspection, a more major insecurity. That is the case here. For example, an adversary can choose $m_{100} = e_{99}$, causing $e_{100} = F_k(\overline{0})$; later he can choose $m_{200} = e_{199}$, causing $e_{200} = F_k(\overline{0}) = e_{100}$. In this way, he can tell whether or not the piece sequence $m_{101}, m_{102}, \ldots, m_{199}$ equals $m_{201}, m_{202}, \ldots, m_{299}$.

It is interesting to note that if we weakened our adversary model by not allowing our adversary to see any e_j until he has chosen m_{j+1}, then (I think) Cryptosystem VI would become secure. However, without changing our definitions we can make this cryptosystem secure by slightly modifying it:

Cryptosystem VI′. Let F be a (hopefully pseudo-random) function generator. (It need not be a permutation generator.)

For an n-bit k and n-bit message pieces $m_0, m_1, m_2, \ldots$, define the n-bit encryptions $e_0, e_1, e_2, \ldots$ as follows:
$e_{-1} = \overline{0}$;
$e_i = F_k(e_{i-1}) \oplus m_i$ for $i \geq 0$.

DEC works in the obvious way.

Theorem: If the function generator used in Cryptosystem VI′ is pseudo-random, then the system satisfies privacy.

Proof: Slightly difficult exercise.

Total Security

Definition: We say a shared-private-key cryptosystem or session protocol, is *totally secure* if it satisfies both unchangeability security (integrity) and indistinguishability security (privacy).

It is not hard to show that if pseudo-random generators exist, then totally secure cryptosystems exist.

Cryptosystem VII. Let F be a (hopefully *strongly* pseudo-random) permutation generator where for an n-bit k, $F_k : \{0,1\}^{2n} \to \{0,1\}^{2n}$.

For an n-bit k and n-bit message pieces $m_0, m_1, m_2, \ldots$, define the $2n$-bit encryptions:
$e_i = F_k(\bar{i}\, m_i)$.

DEC works in the obvious way. To decrypt $e'_0, e'_1, \ldots$ using key k where each $|e'_i| = 2n$, we do the following for each i:
compute $[u, v] = F_k^{-1}(e'_i)$ where $|u| = |v| = n$; if $u = \bar{i}$ then output v, otherwise output FAIL and abort.

Theorem: If the permutation generator used in Cryptosystem VII is strongly pseudo-random, then the system is totally secure.

Proof: Exercise. (Do you see why it is necessary that that the permutation generator be *strongly* pseudo-random?)

Cryptosystem VIII. One general way to create a totally secure shared-private-key cryptosystem is to encrypt so as to satisfy privacy, and then, with an independent key, encrypt the encrypted pieces so as to satisfy integrity. For example, we can combine Cryptosystems I and II (from last week) by using two n-bit keys k_1 and k_2, and encrypting piece m_i as
$e_i = [m_i \oplus F_{k_1}(\bar{i}), F_{k_2}(\bar{i},\ m_i \oplus F_{k_1}(\bar{i}))]$.
DEC works in the obvious way.
(Note that we are assuming that F is just a function generator and that F_k takes as inputs – and is pseudo-random with respect to – inputs of length n and $2n$.) (Of course, if one only wants to use a single n-bit key, one can use a pseudo-random number generator to expand it two a $2n$-bit key.)

More generally, this idea works as follows. We use key k_1 with an indistinguishability secure system S_1 to encrypt message pieces $m_0, m_1, \ldots$ as $e_0, e_1, \ldots$; we then treat $e_0, e_1, \ldots$ as the message pieces in a unchangeability secure system S_2 and encrypt them with key k_2 to obtain $f_0, f_1, \ldots$. Let's call this new system S_3. We decrypt in S_3 in the obvious way: Say we are given strings $f'_0, f'_1, \ldots$ of the right size to decrypt; we decrypt each f'_i as follows (assuming we have not already aborted): we use k_2 as in S_2 to decrypt f'_i, and if this results in FAIL then S_3 outputs FAIL and we abort; if this succeeds and outputs e'_i, then we use k_1 as in S_1 to decrypt e'_i as m'_i; if this results in FAIL then S_3 outputs FAIL and aborts, otherwise S_3 outputs m'_i.

To see why this is totally secure, first imagine that we can break the indistinguishability security (privacy) of S_3 using adversary D. We will use D to break the indistinguishability security of S_1. We begin by randomly choosing a key k_2 for S_2. We then simulate D breaking S_3, using k_2 whenever necessary to simulate S_2. Here are the details.

D chooses a position i, which is the position we will use to break S_1. D then chooses m_0, which is the value we use for m_0; we are then given a value e_0 (an encryption of m_0 using the unknown k_1), which we encrypt using k_2 to f_0, which we give to D; we continue in this way for $m_1, e_1, f_1, \ldots, m_{i-1}, e_{i-1}, f_{i-1}$. D now chooses pieces m^0, m^1, which are the two pieces we choose as well. We are then given e_i, an encryption of m^b using k_1, which we encrypt using k_2 to f_i, which we give to D. We then continue as before for $m_{i+1}, e_{i+1}, f_{i+1}, \ldots, m_\ell, e_\ell, f_\ell$. We want to guess b.

Now, D computes a sequence $f'_0, f'_1, \ldots$, and he wants to know if and when this causes FAIL in S_3. Using k_2, we can compute if and when this causes FAIL in S_2, decrypting (up to that point) in S_2 to obtain $e'_0, e'_1, \ldots, e'_p$; this may or may not be followed by FAIL, but let us assume the more interesting case that it is. We then treat $e'_0, e'_1, \ldots, e'_p$ as the string to send to B in attacking S_1, and we are told if and when this causes FAIL in S_1. If FAIL occurs at some point here, we give that point to D, otherwise we give $p+1$ to D. D then outputs a guess at b which will also be our guess, and we will succeed with the same probability that D does.

Now imagine that we can break the unchangeability security (integrity) of S_3 using adversary C. We will use C to break the unchangeability security of S_2. We begin by randomly choosing a

key k_1 for S_1. We then simulate C breaking S_3, using k_1 whenever necessary to simulate S_1. This simulation chooses pieces $m_0, m_1, \ldots$ creates and creates their encryption under S_1, $e_0, e_1, \ldots$, and we will treat $e_0, e_1, \ldots$ as the input pieces we will use to break S_2. C sees encryptions in S_2 of these pieces, and then creates a string $f'_0, f'_1, \ldots$ to send to B to break S_3. C succeeds if and only if some f'_i decrypts in S_3 to something besides FAIL and m_i, but (because we have properly simulated S_1) this will happen only if some f'_i decrypts in S_2 to something besides FAIL and e_i. So we will use the same string $f'_0, f'_1, \ldots$ to try to break S_2, and we will succeed with the same probability that C does.

There are other ways of creating totally secure systems. For example we could use:
$e_i = [m_i \oplus F_{k_1}(\bar{i}), F_{k_2}(\bar{i}, m_i)]$
where B decrypts in the obvious way. We leave it as an exercise to prove that this is totally secure if F is pseudo-random.

One last remark about unchangeability security. Note that the unchangeability secure Cryptosystem II encrypted each piece of size n by a string of length $2n$, thereby sending a bit stream that is twice as long as the actual message. However, more generally, it is easy to see how to encrypt each piece of size $z(n)$ by a string of length $z(n) + n$ and obtain unchangeability security. We can therefore reduce the communication overhead, if we are willing to increase the granularity $z(n)$ of our system.