Length extension on Steroids - Pseudorandom functions.

Reading:

In the last lecture we saw the notion of *pseudorandom generators*, and introduced the **PRG conjecture** that there exists a pseudorandom generator mapping n bits to $n+1$ bits. We have seen the *length extension* theorem that we given such a pseudorandom generator, we can create a generator mapping n bits to m bits for an arbitrarily large polynomial $m(n)$. But can we extend it even further? Say, to 2^n bits? Does this question even make sense? And why would we want to do that? This is the topic of this lecture.

At a first look, the notion of extending the output length of a pseudorandom generator to 2^n bits seems nonsensical. After all we want our generator to be *efficient* and just writing down the output will take exponential time. However, there is a way around this conundrum. While we can't efficiently write down the full output, we can require that it would be possible, given an index $i \in \{1,\ldots,2^n\}$, to compute the i^{th} bit of the output in polynomial time. That is, we require that the function $i \mapsto G(S)_i$ is efficiently computable and (by security of the pseudorandom generator) indistinguishable from a function that maps each index i to an independent random bit in $\{0,1\}$. This is the notion of a *pseudorandom function generator* which is a bit subtle to define and construct, but turns out to have great many applications in cryptography.

Definition (Pseudorandom Function Generator): An efficiently computable function F taking two inputs $s \in \{0,1\}^n$ and $i \in \{1,\ldots,2^n\}$ and outputting a single bit $F(s,i)$ is a *pseudorandom function (PRF) generator* if for every polynomial time adversary A outputting a single bit and polynomial $p(n)$, if n is large enough then:

$$\left|\mathbb{E}_{s\in\{0,1\}^n}[A^{F(s,\cdot)}(1^n)] - \mathbb{E}_{H\leftarrow_R [2^n]\to\{0,1\}}[A^{H}(1^n)]\right| < 1/p(n) \ .$$

Some notes on notation are in order. The input 1^n is simply a string of n ones, and it is a typical cryptography convention to assume that such an input is always given to the adversary. This is simply because by "polynomial time" we really mean polynomial in n (which is our key size or security parameter). The notation $A^{F(s,\cdot)}$ means that A has *black box* (also known as *oracle*) access to the function that maps i to $F(s,i)$. That is, A can choose an index i, query the box and get $F(s,i)$, then choose a new index i', query the box to get $F(s,i')$, etc.. etc.. for a polynomial number of queries. The notation $H \leftarrow_R \{0,1\}^n \to \{0,1\}$ means that H is simply a completely random function that maps every index i

to an independent and random different bit. That means that the notation A^H in the equation above means that A has access to a completely random black box that returns a random bit for any new query made. Finally one last note: below we will identify the set $[2^n] = \{1, \ldots, 2^n\}$ with the set $\{0,1\}^n$ (there is a one to one mapping between those sets using the binary represnetation), and so we will treat i interchangebly as a number in $[2^n]$ or a string in $\{0,1\}^n$.

Informally, if F is a pseudorandom function generator, then if we choose a random string s, and consider the function f_s defined by $f_s(i) = F(s,i)$ then no efficient algorithm can distinguish between black box access to $f_s(\cdot)$ and black box access to a completely random function. Thus instead of talking about a pseudorandom function generator we will sometimes refer to a *pseudorandom function collection* $\{f_s\}$ where by that we mean that the map $F(s,i) = f_s(i)$ is a pseudorandom function generator.

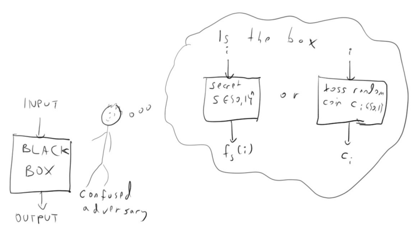

Figure 1: In a pseudorandom function, an adversay cannot tell whether they are given a black box that computes the function $i \mapsto F(s,i)$ for some secret s that was chosen at random and fixed, or whether the black box computes a completely random function that tosses a fresh random coin whenever it's given a new input i

We will later see the following theorem:

Theorem (PRF Theorem, Goldwasser, Goldreich and Micali): Assuming the PRG conjecture, there exists a secure pseudorandom function generator.

But before we see the proof of the PRF Theorem, let us see why pseudorandom functions could be useful.

One time passwords (e.g., Google Authenticator, RSA ID, etc..)

Until now we have talked about the task of *encryption*, or protecting the *secrecy* of messages. But the task of *authentication*, or protecting the *integrity* of messages is no less important. For example, consider the case that you receive a software update for your PC, phone, car, pacemaker, etc.. over an open connection such as an unencrypted Wi-Fi. Then the contents of that update are not secret, but it is of crucial importance that no malicious attacker had modified the code and that it was unchanged from the message sent out by the company. Similarly, when you log into your bank, you might be much more concerned about the possiblity of someone impersonating you and cleaning out your account than you are about the secrecy of your information.

Let's start with a very simple scenario which I'll call **the login problem**. **Alice** and **Bob** share a key as before, but now Alice wants to simply prove her identity to Bob. What makes it challenging is that this time they need to tackle not the passive eavesdropping Eve but the active adversary **Mallory** who completely controls the communication channel between them and can modify (or *mall*) any message that they send out. Specifically for the identity proving case, we think of the following scenario. Each instance of such an **identification protocol** consists of some interaction between Alice and Bob that ends with Bob deciding whether to accept it as authentic or reject as an impersonation attempt. Mallory's goal is to fool Bob into accepting her as Alice.

The most basic way to try to solve the login problem is simply using a *password*. That is, if we assume that Alice and Bob can share a key, we can treat this key as some secret password p that was selected at random from $\{0,1\}^n$ (and hence can only be guessed with probability 2^{-n}). Why doesn't Alice simply send p to Bob to prove to him her identity? A moment's thought shows that this would be a very bad idea. Since Mallory is controlling the communication line, she would learn p after the first identification attempt and then could impersonate Alice in future interactions. However, we seem to have just the tool to protect the secrecy of p— *encryption*. Suppose that Alice and Bob share a secret key k and an additional secret password p. Wouldn't a simple way to solve the login problem be for Alice to send to Bob an encryption of the password p? After all, the security of the encryption should guarantee that Mallory can't learn p, right?

This would be a good time to stop reading and try to think for yourself whether using a secure encryption to encrypt p would guarantee security for the login problem. (No really, stop and think about it.)

The problem is that Mallory does not have to learn the password p in order to impersonate Alice. For example, she can simply record the message Alice c_1 sends to Bob in the first session and then *replay* it to Bob in the next session. Since the message is a valid encryption of p, then Bob would accept it from Mallory! (This is known as a *replay attack* and is a common concern one needs to protect against in crypgoraphic protocols.) One can try to put in countermeasures to replay this particular attack, but its existence demonstrates that secrecy of the password does not guarantee security of the login protocol.

How do pseudorandom functions help in the login problem? The idea is that they create what's known as a *one time password*. Alice and Bob will share an index $s \in \{0,1\}^n$ for the pseudorandom function generator $\{f_s\}$. When Alice wants to prove to Bob her identity, Bob will choose a random $i \leftarrow_R \{0,1\}^n$, and send i to Alice, and then Alice will send $f_s(i), f_s(i+1), \ldots, f_s(i+\ell-1)$ to Bob where ℓ is some parameter (you can think of $\ell = n$ for simplicity). Bob will check that indeed $y = f_S(i)$ and if so accept the session as authentic.

The formal protocol is as follows:

Protocol PRF-Login:

- Shared input: $s \in \{0,1\}^n$. Alice and Bob treat it as a seed for a pseudorandom function generator $\{f_s\}$.
- In every session Alice and Bob do the following:

1. Bob chooses a random $i \leftarrow_R [2^n]$ and sends i to Alice.
2. Alice sends $y_1, \ldots, y_\ell$ to Bob where $y_j = f_s(i+j-1)$.
3. Bob checks that for every $j \in \{1, \ldots, \ell\}$, $y_j = f_s(i+j-1)$ and if so accepts the session; otherwise he rejects it.

As we will see it's not really crucial that the input i (which is known in crypto parlance as a *nonce*) is random. What is crucial is that it never repeats itself, to foil a replay attack. For this reason in many applications Alice and Bob compute i as a function of the current time (for example, the index of the current minute based on some agreed-upon starting point), and hence we can make it into a one message protocol. Also the parameter ℓ is sometimes chosen to be deliberately short so that it will be easy for people to type the values $y_1, \ldots, y_\ell$ in.

Figure 2: The Google Authenticator app is one popular example of a one-time password scheme using pseudorandom functions. Another example is RSA's SecurID token.

Why is this secure? The key to understanding schemes using pseudorandom functions is to imagine what would happen if instead of a *pseudo* random function, f_s would be an *actual* random function. In a truly random function, every one of the values $f_s(1), \ldots, f_s(2^n)$ is chosen independently and uniformly at random from $\{0,1\}$. One useful way to imagine this is using the concept of "lazy evaluation". We can think of f_S as determined by tossing 2^n different coins for the values $f(1), \ldots, f(2^n)$. Now we can think of the case where we don't actually toss the i^{th} coin until we need it. Now the crucial point is that if we have queried the function in $T \ll 2^n$ places, and now Bob chooses a random

$i \in [2^n]$ then it is extremely unlikely that any one of the set $\{i, i+1, \ldots, i+\ell-1\}$ will be one of those locations that we previously queried. Thus Mallory has *no information* on the value of the function, and would be able to predict it in all these locations with probability at most $2^{-\ell}$.

Please make sure you understand this informal reasoning, since we will now translate this into a formal theorem and proof.

Theorem: Suppose that $\{f_s\}$ is a secure pseudorandom function generator and Alice and Bob interact using it some polynomial number T of sessions (over a channel controlled by Mallory), and then Mallory interacts with Bob when Bob follows the protocols instructions and Mallory uses an arbitrary efficient computation. Then, the probability that Bob accepts the interaction after this interaction is at most $2^{-\ell} + \mu(n)$ where $\mu(\cdot)$ is some negligible function.

Proof: This proof, as so many others in this proof, uses an argument via contradiction. We assume, towards the sake of contradiction, that there exists an adversary M (for Mallory) that can break the identification scheme `PRF-Login` with probability $2^{-\ell} + \epsilon$ after T interactions and construct an attacker A that can distinguish access to $\{f_s\}$ from access to a random function using $poly(T)$ time and with bias at least $\epsilon/2$.

How do we consrtruct this adversary A? The idea is as follows. First, we prove that if we ran the protocol `PRF-Login` using an *actual random* function, then M would not be able to succeed in impresonating with probability better than $2^{-\ell} + negligible$. Therefore, if M does do better, then we can use that to distinguish f_s from a random function. The adversary A gets some black box $F(\cdot)$ and will use it while simulating internally all the parties— Alice, Bob and Mallorty (using M) in the $T+1$ interactions of the `PRF-Login` protocol. Whenever any of the parties needs to evalueate $f_s(i)$, A will forward i to its black box $F(\cdot)$ and return the value $F(i)$. It will then output 1 if an only if M succeeds in impersonation in this internal simulation. The argument above showed that if $F(\cdot)$ is a truly random function then the probability A outputs 1 is at most $2^{-\ell} + negligible$ (and so in particular less than $2^{-\ell} + \epsilon/2$ while under our assumptions, if $F(\cdot)$ is the function $i \mapsto f_s(i)$ for some fixed and random s, then this probability is at least $2^{-\ell} + \epsilon$. Thus A will distinguish between the two cases with bias at least $\epsilon/2$. We now turn to the formal proof:

Claim 1: Let `PRF-Login*` be the hypothetical variant of the protocol `PRF-Login` where Alice and Bob share a completely random function $H : [2^n] \to \{0,1\}$. Then, no matter what Mallory does, the probability she can impersonate Alice after observing T interactions is at most $2^{-\ell} + (8\ell T)/2^n$.

(If `PRF-Login*` is easier to prove secure than `PRF-Login`, you might wonder why we bother with `PRF-Login` in the first place and not simply use `PRF-Login*`. The reason is that specifying a random function H requires specifying 2^n bits, and so that would be a *huge* shared key. So `PRF-Login*` is not a protocol we can actually run but rather a hypothetical "mental experiment" that helps us in arguing about the security of `PRF-Login`.)

Proof of Claim 1: Let $i_1,\ldots,i_{2T}$ be the nonces chosen by Bob and recieved by Alice in the first T iterations. That is, i_1 is the nonce chosen by Bob in the first iteration while i_2 is the nonce that Alice received in the first iteration (if Mallory doesn't modify it then $i_1 = i_2$), similarly i_3 is the nonce chosen by Bob in the second iteration while i_4 is the nonce recieved by Alice and so on and so forth. Let i be the nonce chosen in the $T+1^{st}$ iteration in which Mallory tries to impersonate Alice. We claim that the probability that there exists some $j \in \{1,\ldots,2T\}$ such that $|i - i_j| < 2\ell$ is at most $(8\ell T/2^n$. Indeed, let S be the union of all the intervals of the form $\{i_j - 2\ell + 1, \ldots, i_j + 2\ell - 1\}$ for $1 \leq j \leq 2T$. Since it's a union of $2T$ intervals each of length less than 4ℓ, S contains at most $8T\ell$ elements, but so the probability that $i \in S$ is $|S|/2^n \leq (8T\ell)/2^n$. Now, if there does *not* exists a j such that $|i - i_j| < 2\ell$ then it means in particular that all the queries to $H(\cdot)$ made by either Alice or Bob during the first T iterations are disjoint from the interval $\{i, i+1, \ldots, i+\ell-1\}$. Since $H(\cdot)$ is a completely random function, the values $H(i),\ldots,H(i+\ell-1)$ are chosen uniformly and independently from all the rest of the values of this function. Since Mallory's message y to Bob in the $T+1^{st}$ iteration depends only on what she observed in the past, the values $H(i),\ldots,H(i+\ell-1)$ are *independent* from y, and hence under this condition that there is no overlap between this interval and prior queries, the probability that they equal y is $2^{-\ell}$. QED (Claim 1).

The proof of Claim 1 is not hard, but it is somewhat subtle, and it's good to go over it again and make sure you are sure you understand it.

Now that we have Claim 1, the proof of the theorem follows as outlined above. We build an adversary A to the pseudorandom function generator from M by having A simulate "inside its belly" all the parties Alice, Bob and Mallory and output 1 if Mallory succeeds in impersonating. Since we assumed ϵ is non-negligible and T is polynomial, we can assume that $(8\ell T)/2^n < \epsilon/2$ and hence by Claim 1, if the black box is a random function then we are in the `PRF-Login*` setting and Mallory's success will be at most $2^{-\ell} + \epsilon/2$ while if the black box is $f_s(\cdot)$ then we get exactly the same setting as `PRF-Login` and hence under our assumption the success will be at least $2^{-\ell} + \epsilon$. We conclude that the difference in probability of A outputting one between the random and pseudorandom case is at least $\epsilon/2$ thus contradicting the security of the pseudorandom function generator. QED

> **A useful observation:** In the course of constructing this one-time-password scheme from a PRF, we have actually proven a general statement that is useful on its own: that we can transform standard PRF which is a collection $\{f_s\}$ of functions mapping $\{0,1\}^n$ to $\{0,1\}$, into a PRF where the functions have a longer output ℓ (see the problem set for a formal statement of this result) Thus from now on whenver we are given a PRF, we will allow ourselves to assume that it has any output size that is convenient for us.

Message Authentication Codes

One time passwords are a tool allowing you to prove your *identity* to, say, your email server. But even after you did so, how can the server trust that future communication comes from you and not from some attacker that can interfere with the communication channel between you and the server (so called "man in the middle" attack). Similarly, one time passwords may allow a software company to prove that their identity before they send you a software update, but how do you know that an attacker does not change some bits of this software update on route between their servers and your device?

This is where *message authentication codes* come into play- their role is to authenticate not merely the *identity* of the parties but also their *communication*. Once again we have **Alice** and **Bob**, and the adversary **Mallory** who can actively modify messages (in contrast to the passive eavesdropper Eve). Similar to the case o encryption, Alice has a *message* m she wants to send to Bob, but now we are not concerned with with Mallory *learning* the contents of the message, but they want to make sure that Bob gets precisely the message m sent by Alice. Actually this is too much to ask for, since Mallory can always decide to block all communication, but we can ask that either Bob gets precisely m or he detects failure and accepts no message at all. Since we are in the *private key* setting, we assume that Alice and Bob share a key k that is unknown to Mallory.

What kind of security would we want? We clearly want Mallory not to be able to cause Bob to accept a message $m' \neq m$. But, like in the encryption setting, we want more than that. We would like Alice and Bob to be able to use the same key for *many* messages. So, Mallory might observe the interactions of Alice and Bob on messages $m_1, \ldots, m_T$ before trying to cause Bob to accept a message $m'_{T+1} \neq m_{T+1}$. In fact, to make our notion of security more robust, we will even allow Mallory to *choose* the messages $m_1, \ldots, m_T$ (this is known as a *chosen message* or *chosen plaintext* attack). The resulting formal definition is below:

Definition (Message Authentication Codes (MAC)): Let (S, V) (for *sign* and *verify*) be a pair of efficiently computable algorithms where S takes as input a key k and a message m, and produces a tag $\tau \in \{0,1\}^*$, while V takes as input a key k, a message m, and a tag τ, and produces a bit $b \in \{0,1\}$. We say that (S, V) is a *Message Authentication Code (MAC)* if:

- For every key k and message m, $V_k(m, S_k(m)) = 1$.
- For every polynomial-time adversary A and polynomial $p(n)$, it is with less than $1/p(n)$ probability over the choice of $k \leftarrow_R \{0,1\}^n$ that $A^{S_k(\cdot)}(1^n) = (m', \tau')$ such that m' is *not* one of the messages A queries and $V_k(m', \tau') = 1$.[1]

[1] Clearly if the adversary outputs a pair (m, τ) that it did query from its oracle then that pair will pass verification. This suggests the possiblity of a *replay* attack whereby Mallory resends to Bob a message that Alice sent him in the past. As above, once can thwart this by insisting the every message m begins with a fresh nonce or a value derived from the current

If Alice and Bob share the key k, then to send a message m to Bob, Alice will simply send over the pair (m, τ) where $\tau = S_k(m)$. If Bob receives a message (m', τ'), then he will accept m' if and only if $V_k(m', \tau') = 1$. Now, Mallory could observe t rounds of communication of the form $(m_i, S_k(m_i))$ for messages $m_1, \ldots, m_t$ of her choice, and now her goal is to try to create a new message m' that was *not* sent by Alice, but for which she can forge a valid tag τ' that will pass verification. Our notion of security guarantees that she'll only be able to do so with negligible probability.[2]

MACs from PRFs

We now show how pseudorandom function generators yield message authentication codes. In fact, the construction is so immediate, that much of the more applied cryptographic literature does not distinguish between these two concepts, and uses the name "Message Authentication Codes" to refer to both MAC's and PRF's.

Theorem (MAC Theorem): Under the PRG Conjecture, there exists a secure MAC.

Proof: Let $F(\cdot, \cdot)$ be a secure pseudorandom function generator with $n/2$ bits output (as mentioned above, such PRF's can be constructed from one bit output PRF's). We define $S_k(m) = F(k, m)$ and $V_k(m, \tau)$ to output 1 iff $F_k(m) = \tau$. Suppose towards the sake of contradiction that there exists an adversary A that queries $S_k(\cdot)$ $poly(n)$ many times and outputs (m', τ') that she did *not* ask for and such that $F(k, m') = \tau'$. Now, if we had black box access to a completely random function $H(\cdot)$, then the value $H(m')$ would be completely random in $\{0,1\}^{n/2}$ and independent of all prior queries. Hence the probability that this value would equal τ' is at most $2^{-n/2}$. That means that such an adversary can distinguish between an oracle to $F_k(\cdot)$ and an oracle to a random function H. QED

Input length extension for MACs and PRFs

So far we required the message to be signed m to be no longer than the key k (i.e., both n bits long). However, it is not hard to see that this requirement is not really needed. If our message is longer, we can divide into blocks $m_1, \ldots, m_t$ and sign each message (i, m_i) individually. The disadvantage here is that the size of the tag (i.e., MAC output) will grow with the size of the message. However,

time.

[2] A priori you might ask if we should not also give Mallory an oracle to $V_k(\cdot)$ as well. After all, in the course of those many interactions, Mallory could also send Bob many messages (m', τ') of her choice, and observe from his behaviour whether or not these passed verification. It is a good exercise to show that adding such an oracle does not change the power of the definition, though we note that this is decidedly *not* the case in the analogous question for encryption.

even this is not really needed. Because the tag has length $n/2$ for length n messages, we can sign the *tags* $\tau_1, \ldots, \tau_t$ and only output those. The verifier can repeat this computation to verify this. We can continue this way and so get tags of $O(n)$ for arbitrarily long messages. Hence in the future, whenever we need to, we will assume that our PRFs and MACs can get inputs in $\{0,1\}^*$ — i.e., arbitrarily length strings.

We note that this issue of length extension is actually quite a thorny and important one in practice. The above approach is not the most efficient way to achieve this, and there are several more practical variants in the literature (see KL Sections ??, and Boneh-Shoup Sections ??). Also, one needs to be very careful on the exact way one chops the message into blocks and pads it to an integer multiple of the block size. Several attacks have been mounted on schemes that performed this incorrectly.