Notes for Lecture 21

1 Key Recycling

Motivating example. Suppose we have Alice sending a message m to Bob, and an adversary that sits in potentially changes the message to m'. So we have Alice and Bob share a key k, and what Alice does is send (m, σ) and the adversary sends something of the form (m', σ').

The question we'll focus on is: can Alice use k a second time?

The simplest example of a message authentication code (MAC) is $\sigma = h(m)$ where h is a pairwise independent function (for example $am + b$ for random a, b drawn from a suitable field). This is one-time secure, since for the adversary to succeed, they need to produce two input/output pairs of the function having only seen one point.

However, the moment the adversary sees two signed messages, security is completely lost (for example, in the $am + b$ example the adversary can interpolate to recover a, b). Hence this is called a one-time MAC. Note this can be easily generalized to get a d-time MAC with a d-wise independent function.

But in all these cases, once you see $d + 1$ signed messages security is lost. An improvement is to switch to h being a PRF. But since this looks like a random function, the adversary is unable to compute any h on any other point. However, for a PRF we require computational assumptions, since a PRF cannot be secure against computationally unbounded adversaries.

More generally, a many-time MAC (i.e. no a priori bound on the number of messages signed) requires computational assumptions. The proof is similar. At a high level, each time you see a new signature, you either learn more information about the secret key, or you knew enough to forge the signature yourself.

However, it would still be nice to have schemes that don't require any computational assumptions.

1.1 The Quantum Setting

Suppose my tags $|\sigma\rangle$ can now be quantum. So Alice forwards along $m, |\sigma\rangle$, and the adversary changes this to something of the form $m', |\sigma'\rangle$ (here the messages are still classical).

Before, we were saying that every time the adversary sees a message, it gains more and more information about the secret key. Here, the adversary can still learn information about the secret key. But any information the adversary learns about σ, it necessarily perturbs σ. This is one of the fundamental notions in quantum mechanics, which is that to get information you must perturb the state. So if the adversary extracts information, it might be the case that B will reject.

Thus, the adversary has to pick between learning info about k, OR it can give B a valid σ. Then if the adversary gives B a valid σ, we should be able to re-use the key k.

So we'll let this σ be the money state from Weisner's quantum money scheme.

We'll have the key k be (k, θ). Here, k will be a one-time MAC key (we'll need more properties of this key that we'll specify later). θ will just be a string in $\{0,1\}^n$.

- $\mathsf{MAC}(k, m)$: Choose random $x \leftarrow \{0,1\}^n$. Then we construct Weisner's money state $|\$\rangle = H^\theta |x\rangle$. This notation means $(H^{\theta_1}|x_1\rangle)(H^{\theta_2}|x_2\rangle), ...$ We let H^0 be the computational basis and H^1 be the Fourier basis. So the first time the adversary sees this, it's an unforgeable quantum money state. Note that x is randomly chosen. Next we'll apply $\sigma_c = \mathsf{MAC}_c(k', (m, x))$ (here the c subscript denotes classical). So the first part is constructing a random quantum money state unrelated to the message, and the second part relates it to the message by MACing the message with the x. So we output

$$\sigma = (\sigma_c, |\$\rangle)$$

Note that we don't need to give out x, because Bob knows θ. So it can measure the money state in the basis θ to recover x. Then Bob can verify using the classical verification algorithm $\mathsf{Ver}_c(k', (m, x), \sigma_c)$ and see that this accepts.

If we were to give out x, this would not be secure.

Security Intuition. For starters, we'll point out that this might not work for a general MAC. So if we use a dumb choice of MAC, there's nothing that prevents σ_c from revealing x. If I reveal x, I'm okay for one-time. But if we use this twice, then we have the full message tag pairs for two different messages, and then all security is lost if it's a just a one-time classical MAC.

What we actually want is for the MAC to be a good "extractor." Then σ_c contains no info about k'. So k' defines a pairwise independent function $h(m, x)$ where the adversary knows m. Without the quantum money state, the adversary knows nothing about x. But even with the quantum money state, the adversary doesn't know θ so there must be information about x that the adversary doesn't know. So the input m, x has high entropy from the adversary's point of view.

Definition: h is a good extractor if for any distribution where Z has sufficient min-entropy, $(h, h(Z)) \approx_S (h, R)$ (statistically close).

In particular, this means that if I give you $h(Z)$, this gives no information at all about h (because this is true on the right-hand side, it must be true on the left-hand side). So if I didn't send the quantum money state along with the MAC, my key is information theoretically hidden. But with the money state, if you learn information, you start perturbing things.

Let's say I need to learn a quarter of the bits of x in order to break the guarantee required by the extractor. But then there will be another quarter of the bits in the wrong basis, which Bob will detect.

So Bob needs to return a bit (along a secure side-channel) that indicates if Bob accepted the previous message. If either of them reject, they both switch their keys.

1.2 What about Encryption?

We might hope for the same exact thing. In other words, if the ciphertext is the same exact state, the adversary can choose to learn something about the message and key, but then Bob will reject. But for the same reason as before, we must have some sort of authentication from Bob back to Alice.

The scheme is simple. We essentially run the signing scheme, but we don't include m in the signature. Our key is (θ, k', k'').

We choose an $x \leftarrow \{0,1\}^n$, and we let the money state be $|\$\rangle = H^\theta |x\rangle$. We compute $\sigma_c = MAC_c(k', x)$. And then we apply another extractor $p \leftarrow \mathsf{Extract}(k'', x)$ (keyed on k''). So the ciphertext is $(|\$\rangle, \sigma_c = p \oplus m)$.

To decrypt, you measure the $|\$\rangle$ in the right basis to recover x, and then you verify the MAC classically, and then if it passes you recover the message.

If the adversary learns anything about θ, it will cause Bob to accept.

Note that the adversary will not learn anything about the message m, because the adversary cannot learn all of x and we're masking the message with randomness extracted from x.

What we have is that if Bob accepts, then we can be assured that the adversary is not learning anything and we can continue encrypting messages.

1.3 Encrypting Quantum Messages with Key Recycling

We know that if m is a quantum state, a one-time pad will not be enough for security. To encrypt quantum states, choose a random key r for Enc_Q.

We'll use $\mathsf{Enc}(k, r), \mathsf{Enc}_Q(r, |\psi\rangle)$. The first scheme is the scheme we just discussed which has key recycling. My second scheme encrypts quantum states but doesn't have key recycling. But every time I encrypt, I choose a random key r, so it doesn't matter that my second scheme lacks key recycling. So I can keep using my key k just fine by the key-recycling property of the first scheme.

1.4 Authentication for Quantum Messages with Key Recycling

In this setting we don't know any hybrid schemes like the one above. We can obtain a scheme using Unitary 2-designs. These gives authentication for quantum states. The intuition for unitary 2-designs is that they're just as good as a truly random unitary. When I apply one to any state, what the adversary sees is a truly random quantum state. So if they try to learn anything at all, they are necessarily perturbing the state and this will be detected.