LECTURE - 3/20

Problem set 4 (Hash functions) has some ambiguity in a definition. This ambiguity is specifically for preimage resistance, whether we take $x \in \text{image}(H^s)$ or $x \in \text{range}(H^s)$.

Additionally, no new homework going out today. Expect it Wednesday to Friday.

Remember we were talking about Key Exchange at the end of class last time. The goal today is to give a security definition for that and to talk about the computation assumption that key exchange is based on.

0.1 Discrete Log

Definition 0.1. The **Discrete Log** says that given h, g, we want x such that $g^x = h$.

This is different from a normal log as now g, h are group elements. Formally, we have that:

- G is a group generating algorithim
- $G(1^n) = \mathbb{G}$, $q = |\mathbb{G}|$, and g which generates[1] $\mathbb{G}$.

We might pick $\mathbb{G}$ such that it's always the same, but just "scales in size" in some way.

The discrete log experiment is:

1. $\mathbb{G}, q, g \leftarrow G(1^n)$
2. $h \leftarrow \mathbb{G}$.
3. $\mathcal{A}$ gets $\mathbb{G}, q, g, h$, and outputs x.
4. $\mathcal{A}$ succeeds if and only if $g^x = h$ in $\mathbb{G}$.

We talk about this because we think it's hard for some groups, and think we can build Cryptography on this assumption. This is also a natural question to ask in Math, and it's a nice sanity check that many Mathematicians also can't do this fast.

The problem is it's pretty annoying to write proofs based on this. With our security definitions, we usually output a bit and are correct half the time. But, here we have that $\mathcal{A}$ is correct negligibly if q is exponential in n.

Instead, we have the Decisional Diffie-Helman assumption. This is meant to convert discrete log into a "yes/no" question, which makes writing proofs much more easily. For this, we do the following:

- $\mathbb{G}, g, q \leftarrow G(1^n)$
- $b \leftarrow \{0, 1\}$

[1] As this has a single generator, we have that $\mathbb{G}$ has a cyclic group. We also usually think of q as being n bits.

– If $b = 0$, $\mathcal{A}$ gets $(\mathbb{G}, g, q, g^x, g^y, x^{xy})$ where $x, y \leftarrow \{0, 1, \ldots, q-1\}$.

– If $b = 1$, $\mathcal{A}$ gets $(\mathbb{G}, g, q, g^x, g^y, g^z)$ where $x, y, z \leftarrow \{0, 1, \ldots, q-1\}$.

So, when $b = 1$ you get 3 random group elements with no relationship, and when $b = 0$ you get 2 random group elements, and one "related" group element. Then, the assumption is that:

$$|\mathbb{P}[\mathcal{A}(\cdot) = 1 \mid b = 0] - \mathbb{P}[\mathcal{A}(\cdot) = 1 \mid b = 1]| \leq \text{negl}(n) \tag{1}$$

This isn't the same question as discrete log, but somehow it's very similar. If we could break discrete log, we could definitely break this (compute x, y, check if $z = xy$). This doesn't quite go the other way, meaning it's not equivalent to the computational version. But, breaking discrete logs does break this.

This is routinely referred to as DDH. So, we have that:

$$\text{Discrete log is hard for } \mathbb{G} \implies \text{DDH is hard for } \mathbb{G} \tag{2}$$

It sounds like we don't have the reverse implication for this.

We have the following now:

A		B
$\mathbb{G}, g, q \leftarrow G(1^n)$		
$x \leftarrow \mathbb{Z}_q$		
$h_A = g^x$	$\xrightarrow{\mathbb{G},g,q,h_A}$	
		$y \leftarrow \mathbb{Z}_q$
	$\xleftarrow[h_B]{}$	$h_B = g^y$
$k = (h_B)^x = g^{xy}$		$k = (h_A)^y = g^{xy}$

We want this to be such that an observer can't distinguish the key from random, meaning that they have no information about the key.

The key exchange experiment $KE^{eav}_{\mathcal{A},\Pi}(n)$ is as follows:

– A, B execute Π

- *trans* = transcript is the set of all messages sent back and forth by the two parties
- Pick $b \leftarrow \{0, 1\}$.
  - If $b = 0, \hat{k} = k$ chosen by Π.
  - If $b = 1$, $\hat{k}$ is equal to a random group element.
- The transcript and $\hat{k}$ are sent to $\mathcal{A}$, who returns b'.
- $\mathcal{A}$ succeeds if $b = b'$.

This only guards against eavesdroppers. If adversaries could modify messages, then key exchange becomes impossible. They could easily just cause everything to fail, or impersonate Bob, etc. This doesn't say we can't do public key encryption, just that we can't identify identities from scratch.

Theorem 0.2. *Diffie Hellman key exchange is secure.*

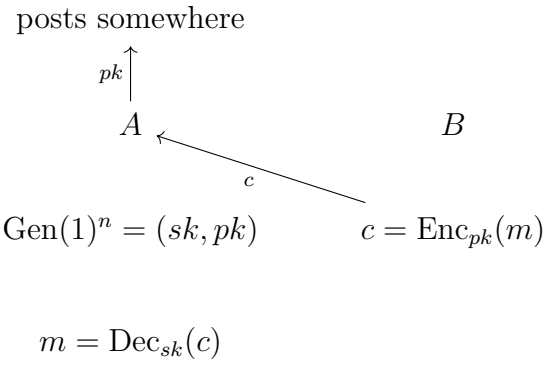
Proof. We'll show that if someone breaks key exchange they break the discrete logarithim.

In the key exchange experiment, the adversary gets $(\mathbb{G}, g, q, h_A, h_B, \hat{k})$. We have that these are really:

$$(\mathbb{G}, g, q, g^x, g^y, g^{xy} \text{ or } g^z) \tag{3}$$

So, if $\mathcal{A}$ can break key exchange, then $\mathcal{A}'$ for DDH gets $(\mathbb{G}, g, q, g^x, g^y, g^a)$ where $a \in \{z, xy\}$. So, $\mathcal{A}'$ runs $\mathcal{A}$ on this data. If $\mathcal{A}$ succeeds more than half the time, we can use it to distinguish between $a = z$ and $a = xy$.

Now, it's possible that $\mathcal{A}$ is very successful when $a = z$, and less successful when $a = xy$, but this ends up working out, and the formal probability computations are in the textbook. □



Here, the adversary can see the public key, and thus the adversary knows everything that Bob knows. If you know the function $\text{Enc}_{pk}(\cdot)$, we should be able to invert it. In principle we can, but it may be very hard to invert.

So, we need a family of functions indexed by a public that are hard to invert, unless you know have the secret key (the function is called a trapdoor function). Furthermore, we want these to generate rather quickly. We'll also need to generate the secret key on the way to getting the public key.

So, key generation algorithims will be much harder now. This doesn't seem like something that's generically possible, but it really is.

There were quickly two public key encryption systems given. One is based on DDH (a key exchange protocol), called El Gamal. The other one is RSA (uses the RSA assumption - in pop culture this says that "it's hard to factor large prime numbers", in practice it's "RSA is secure").