Lecture 19: 2006.03.21

Lecturer: David Wagner *Scribe: Adrian Mettler*

19.1 Proof of Security for El Gamal

Recall, El Gamal is defined for a cyclic group $G = \langle g \rangle$, whose elements compose the domain $\mathbb{Z}_q$. It has a private key $x \xleftarrow{\$} \mathbb{Z}_q$, with corresponding public key $h = g^x$.

Encryption of a message $m \in \mathbb{Z}_q$ is given by $E(m) = (g^r, h^r \cdot m)$ for a per-message random value $r \xleftarrow{\$} \mathbb{Z}_q$.

The discrete Diffie-Hellman (DDH) assumption is that the tuple (g^x, g^y, g^{xy}) is computationally indistinguishable from (g^x, g^y, g^z) for $x, y, z \xleftarrow{\$} \mathbb{Z}_q$.

Theorem 19.1 *If the Discrete Diffie-Hellman problem is hard (i.e. if the DDH assumption holds), El Gamal is IND-CPA secure.*

Proof: Assume, by contradiction, that we have an adversary that breaks El Gamal, i.e. that it has significant advantage by a real-or-random definition,

$$AdvA = \Pr[A^E_{pk}(pk) = 1] - \Pr[A^{E_{pk} \circ \$}(pk) = 1]$$

Since El Gamal is a public key encryption scheme, if it is secure against a single query it is secure against q queries, so we only need to show that it is (t, q, ϵ) secure for $q = 1$; we can thus assume that the adversary A makes exactly one query.

Given such an adversary A that runs in time t and has advantage δ, we can construct an adversary B for DDH that runs in time $t + O(1)$ and has avantage δ. Algorithm $B(a, b, c)$ is as follows:

1. Run $A^{E_B}(a)$, where B's version of the encryption oracle E_B answers its one query m with $(b, c \cdot m)$.
2. Output the same result as A does.

In the case where B is called on a triple of the form (g^x, g^r, g^{rx}), what A sees is identical to interacting with a "real" encryption oracle, $B(g^x, g^r, g^{rx}) = A^{E_{pk}}(pk)$. In the case where B is called on a tuple of the form (g^x, g^r, g^z), A sees the values $a = g^x$ and $(b, c \cdot m) = (g^r, g^z \cdot m)$. Since g^z is selected uniformly at random, $g^z \cdot m$ is also a uniform random value and is thus completely indistinguishable from $g^r x \cdot \$(m)$. As $(g^r, g^z \cdot m)$ is the same distribution as $(g^r, g^r x \cdot \$(m)$, This makes B a perfect simulator of a random oracle in this case, $B(g^x, g^r, g^z) = A^{E_{pk} \circ \$}(pk)$.

This construction thus turns an adversary that breaks El-Gamal into one that breaks DDH with the same advantage, adding constant time complexity. ■

19.2 Efficient Asymmetric Encryption

The Golwasser-Micali construction allows us to transmit a single bit in a IND-CPA-secure manner using a trapdoor one-way function, but we'd like a more efficient way to utilize trapdoor one-way functions for encryption. In the case of the RSA trapdoor one way function, for example, we'd like only those who can compute r from r^3 to be able to decrypt the message. Actually, we'd like IND-CPA security, so only a recipient with the private key could distinguish any encrypted message from another. Note that we can't just apply an arbitrary symmetric key scheme like AES, i.e. $E(m) = (r^3, AES_r(m))$, since leaking partial knowledge about the key, in this case its cube, could render AES inscure.

Instead, schemes of the form $E(m) = (r^3, H(r) \oplus m)$ has been proposed, where H is a "secure hash function", along the lines of SHA-1. Note that such a scheme isn't secure for just any hash function, as a correct guess for m allows the adversary to see $H(r)$. If H were invertible, the aversary could then compute r^3 and see if its guess for m was correct. Our first requirement for H is thus that it is one-way. But this isn't sufficient: r^3 itself is a one-way function, but using that as the hash would making recovering m trivial. In order to convince ourselves that this encryption scheme is secure, we want to have some notion of a generic secure hash function, which has been chosen to avoid any bad interactions with the other primitives we are using.

Ideally, we'd define some notion of correctness for a hash function the same way as we have for other cryptographic primitives; if we did this, we could build up schemes based on the guarantees provided by the definition of a secure hash function. There is no clear way how to do this however; intuitively, we'd like our hash function to be difficult to distinguish from random until sampled, but a notion such as

$$AdvA = \Pr[A^{SHA-1} = 1] - \Pr[R \leftarrow \mathcal{F}unc(\{0,1\} \rightarrow \{0,1\}^{160}); A^R = 1]$$

can't work because hash functions are deterministic. Consider the attack

$$A^f() = \text{if } f(\texttt{"Hello World"}) = 0x33ab... \text{ then 1 else 0}$$

19.3 The Random Oracle Model

Certainly, if our scheme would be broken for any choice of hash function, we can't blame weakneses of the scheme on hash function. We can define a "universal attack" as one that defeats a scheme regardless of the hash function that is chosen.
Try 1: A schme that uses a hash function is secure if it is provably secure against universal attacks. This isn't a good criterion however, as it could be that only a small number of hash functions are good enough for the scheme to work. One could conceive of a scheme that, for example, was insecure for any hash whose first bit was close to being evenly distributed between 0 and 1.

Instead, we can consider probabalistic behavior of the scheme over the space of possible hash functions.

We can define A as a semi-universal attack against scheme S if $\Pr[H \leftarrow \mathcal{F}unc(\{0,1\} \rightarrow \{0,1\}^{160}); A \text{ breaks } S[H]]$ is significant.
Try 2: A scheme is considered secure if it is provably secure against semi-universal attacks. The problem with this definition is that is unrealistically hard on the attacker. In essense, it says that the attacker doesn't know what hash we are using, i.e. that we have a secret hash the attacker can't evaluate. Any real scheme would use a known hash function, which the attacker could evaluate as much as it liked, limited only by compuational power.

Try 3: A scheme is secure if $\Pr[H \leftarrow \mathcal{F}unc(\{0,1\} \rightarrow \{0,1\}^{160}); A^H \text{breaks} S[H]]$. Note that here the adversary is given the hash function as an oracle. This is known as the Random Oracle Model, and is often used for evaluating the security of encryption schemes using hash functions.

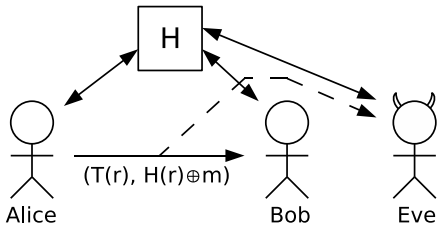

Figure 19.1: The Random Oracle Model and Hash-based asymmetric encryption scheme.

The sender, receiver, and malicious parties all have access to the random oracle H, which can be considered a random function from all strings to hashes of some fixed length.

Specifically, a scheme that uses a concrete hash function such as SHA-1 is analyzed with the hash function replaced with a random oracle. Once the scheme is proved secure in the random oracle, we can make a somewhat handwavey statement about how if SHA-1 behaves the way we'd like it to, it's a lot like a random function and thus the scheme will continue to be secure with SHA-1. This could be a dangerous move, as there's no guarantee that SHA-1 hashes are adequately random. Essentially, this reasoning boils down to the assumption that any adversary won't use any deep knowledge of the structure of SHA-1 and will treat it like a black box random function. In particular, we assume that previous queries don't help the adversary invert the hash function unless the adversary already queried the pre-image.

19.4 An IND-CPA Asymmetric Encryption Scheme in the Random Oracle Model

Real-or-random IND-CPA security in the random oracle model can be defined by

$$AdvA = |\Pr[A^{H,E^H_{pk}}(pk) = 1] - \Pr[A^{H,E^H_{pk}\circ\$}(pk) = 1]|$$

Left-or right IND-CPA is

$$AdvA = |\Pr[A^{H,E^H_{pk}\circ\S_0}(pk) = 1] - \Pr[A^{H,E^H_{pk}\circ\S_1}(pk) = 1]|$$

Note that it is inappropriate in a random-oracle-model security theorem to parameterize against the number of queries to the random oracle. This is because evaluating any real hash function is purely computational and is not visible to any of the honest parties. Evaluation of the hash function will thus only be reflected in the t factor of the security determined for a cryptosystem.

Theorem 19.2 *$E(m) = (T(r), H(r) \oplus m)$ is IND-CPA if T is a trapdoor one-way permutation and H is a random oracle.*

The proof will use the left-right definition of IND-CPA, and the scenario depicted in Figure 19.2.

Ideas behind the proof:

1. We need only consider one query to the encryption oracle, $q = 1$, since this is a public key encryption scheme. Since the adversary makes only one query, it must be of the following form:
   (a) Query H some number of times

(b) Query E once

(c) Query H some number of times

(d) Output "left" or "right".

2. In step (b), the value $H(r)$ used in the encryption is uniformly random, provided the adversary has not already queried r, therefore the value $H(r) \oplus m_i$ that the adversary sees is also uniform random and can't give the adversary any information.

3. As long as adversary doesn't query r in step (c), $H(r) \oplus m_i$ remains indistinguishable from random for the adversary.

Proof: Let the adversary A make one query to its E oracle. If A breaks the encryption scheme with some probabality ϵ, the following trapdoor-one-way-permutation adversary B will break T with at least the same probability. In other words, we will show that one cannot break the encryption without breaking the trapdoor.

Let r be a uniform random value in the domain of T, and let $y = T_{pk}(r)$. Then, define B as follows (also see Figure 19.3):

1. Run $A^{H,e}$ such that $e(m_0, m_1) = (y, \$(m_i))$ (the value to invert and a uniform random ciphertext)

2. Let a_i be the set of queries A made to oracle H. If $\exists i.T_{pk}(a_i) = y$, then output a_i.

3. Else fail.

If A can break the encryption sheme with advantage ϵ, B can break the trapdoor with the same advantage. Define the event *Lucky* as $\{\exists i.a_i = r\}$. In the event that *Lucky* is true, B, which has only been given access to y, has inverted the trapdoor and recovered r without using any additional information (it hasn't queried a "real" encryption oracle). This gives a procedure for inverting T with probability at least $\Pr[Lucky]$.

Note that if *Lucky* is false, the behavior of B's "fake" encryptor e is identical to that of a real encryptor making use of a random oracle. This is because the values $H(r)$, $H(r) \oplus m_0$, and $H(r) \oplus m_1$ have identically uniform distributions. Since the fake encryptor behaves identically to a real encryptor in this case, and the fake encryptor is independent of the inputs (m_0, m_1), adversary A can have no advantage in the case that *Lucky* is false. In particular, there is always an equal number of possible oracles H such that the correct answer is "Left" as there are such that the correct answer is "Right", and as long as A doesn't query r it can't tell which class of oracle it is dealing with. There is a bijective mapping between every hash function H and its opposite H' given by

$$H'(x) = \text{ if } x = r \text{ then } H(x) \oplus m_0 \oplus m_1 \text{ else } H(x)$$

This means that $AdvA = \epsilon \leq \Pr[Lucky] \leq AdvB$. ■

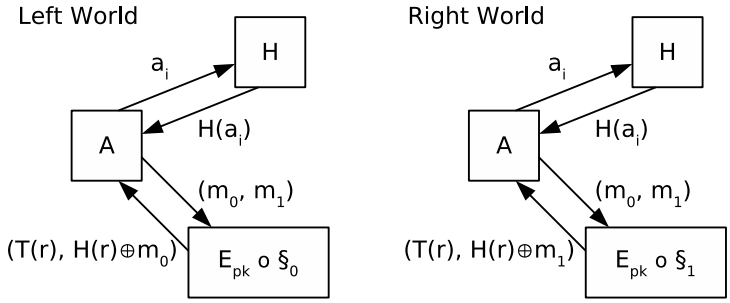

Figure 19.2: Left and right worlds for hash-based asymmetric encryption scheme.

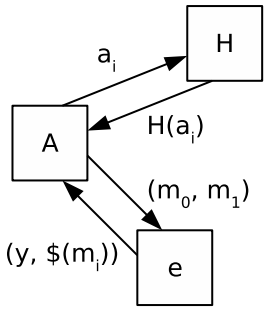

Figure 19.3: Trapdoor one-way permutation adversary B