CS276 Cryptography **Spring 2004**

Lecture 7: 2.11.2004

Lecturer: David Wagner *Scribe: Henry Lin*

7.1 Definitions of Security

Recall in the last lecture, we defined security in three distinct ways. In each setting, the advantage of an attacker A against an encryption scheme E_k was defined differently.

Real-or-Random:
$Adv^{rr}\ A = Pr[A^{E_k} = 1] - Pr[A^{E_k \circ\ \$} = 1]$
Left-or-Right:
$Adv^{lr}\ A = Pr[A^{E_k \circ g_0} = 1] - Pr[A^{E_k \circ g_1} = 1]$
Find-then-Guess:

$$
\begin{aligned}
Adv^{fg}\ A = &Pr[(m_0, m_1, s) \leftarrow A^{E_k}(\text{"}find\text{"}); A^{E_k}(\text{"}guess\text{"}, E_k(m_0), s) = 1] - \\
&Pr[(m_0, m_1, s) \leftarrow A^{E_k}(\text{"}find\text{"}); A^{E_k}(\text{"}guess\text{"}, E_k(m_1), s) = 1]
\end{aligned}
$$

We say an encryption scheme E_k is (t, q, ϵ)-secure if for all algorithms A running in time t and using at most q queries, the $Adv\ A \leq \epsilon$.

7.2 Equivalence of Security Notions

Previously, we showed that real-or-random security was equivalent to left-or-right security. We will now show that left-or-right security is approximately equivalent to find-then-guess security.

Part 1 We start by showing (t, q, ϵ) left-or-right security implies $(t - O(q), q - 1, \epsilon)$ find-then-guess security.

We use a reduction to prove this by contradiction. So suppose there is a fg-attacker A with the $Adv^{fg}\ A > \epsilon$. We now construct an lr-attacker B such that $Adv^{lr}\ B > \epsilon$. Imagine we are in the lr world trying to determine if the oracle $E^k \circ g_b$ is either $E^k \circ g_0$ or $E^k \circ g_1$. Let our attacker B simulate the "find" part A, and whenever A calls E_k on message m, we will call $E_k \circ g_b$ on (m, m). Note this exactly simulates a call to oracle E_k since $(E_k \circ g_b)(m, m) = E_k(m)$. After finding (m_0, m_1, s), let us run $(E_k \circ g_b)(m_0, m_1)$, which will return either $E_k(m_0)$ or $E_k(m_1)$. Now let us run the "guess" part of the algorithm with $(E_k \circ g_b)(m_0, m_1)$ and state s to determine if we got $E_k(m_0)$ or $E_k(m_1)$. Since A can do this with probability greater than ϵ, then this means we have just produced an attack can determine if we received oracle $E_k \circ g_0$ or $E_k \circ g_1$ with probability greater than ϵ. Furthermore, our attack only uses $O(q)$ more time and one additional query.

In the other direction, we can only prove a weaker equivalence.
Part 2 We show that (t, q, ϵ) find-then-guess security implies $(t - O(q), q - 1, q\epsilon)$ left-or-right security.

When analyzing left-or-right security with q queries, our algorithm A will be given the result of q queries to the oracle $E_k \circ g_b$ and will try to distinguish:

$$\begin{array}{c} \overbrace{\begin{array}{l} E_k \circ g_0,\ E_k \circ g_0,\ E_k \circ g_0,\ E_k \circ g_0,\ E_k \circ g_0,\ \dots,\ E_k \circ g_0 \\ E_k \circ g_1,\ E_k \circ g_1,\ E_k \circ g_1,\ E_k \circ g_1,\ E_k \circ g_1,\ \dots,\ E_k \circ g_1 \end{array}}^{q \text{ calls to oracle}} \end{array}$$

This may be difficult to analyze, but we can use a hybrid argument to prove how indistinguishable these two sequences are. Our proof will consist of creating $q-1$ hybrids and showing that attackers cannot distinguish any sequential pair of hybrids very well.

$$\begin{array}{rl} & \overbrace{E_k \circ g_0,\ E_k \circ g_0,\ E_k \circ g_0,\ E_k \circ g_0,\ E_k \circ g_0,\ \dots,\ E_k \circ g_0}^{q \text{ calls to oracle}} \\ (q-1) \text{ hybrids} & \left\{ \begin{array}{l} E_k \circ g_1,\ E_k \circ g_0,\ E_k \circ g_0,\ E_k \circ g_0,\ E_k \circ g_0,\ \dots,\ E_k \circ g_0 \\ E_k \circ g_1,\ E_k \circ g_1,\ E_k \circ g_0,\ E_k \circ g_0,\ E_k \circ g_0,\ \dots,\ E_k \circ g_0 \\ E_k \circ g_1,\ E_k \circ g_1,\ E_k \circ g_1,\ E_k \circ g_0,\ E_k \circ g_0,\ \dots,\ E_k \circ g_0 \\ \qquad\qquad\qquad\qquad \dots \\ \qquad\qquad\qquad\qquad \dots \\ \qquad\qquad\qquad\qquad \dots \end{array} \right. \\ & E_k \circ g_1,\ E_k \circ g_1,\ E_k \circ g_1,\ E_k \circ g_1,\ E_k \circ g_1,\ \dots,\ E_k \circ g_1 \end{array}$$

Given that E_k is (t, q, ϵ) find-then-guess secure, we will show that any two steps in the hybrid argument is ϵ indistinguishable. The triangle inequality then implies that any attacker given q queries to oracle $E_k \circ g_b$ cannot determine whether or not the oracle is $E_k \circ g_0$ or $E_k \circ g_1$ with probability greater than $q\epsilon$.

We now prove any two sequential hybrids are difficult to distinguish. Suppose our hybrids differ in only position $i+1$ where one hybrid returns $E_k \circ g_0$ and the other returns $E_k \circ g_1$. We show that if E_k is (t, q, ϵ) find-then-guess secure, then these two hybrid oracles cannot be distinguished by more than ϵ, with $t - O(q)$ time and $q-1$ queries. For contradiction, let us assume there is an attacker A that distinguishes the two hybrids with probability greater than ϵ. Using A with one more query and $O(q)$ more time we will produce a find-then-guess attack B with advantage greater than ϵ.

Define B
    B^{fg} simulates A;
    On A's i^{th} query (m_0, m_1)
        if $j \le i$ answer $E_k(m_0)$
        if $j = i+1$ answer $g(m_0, m_1)$
        if $j \ge i+2$ answer $E_k(m_1)$
    Return the same answer as A does

(In B, $g(m_0, m_1)$ is the either the value $E_k(m_0)$ or $E_k(m_1)$, which we are given in the "guess" stage of the algorithm).

Since this attack succeeds with probability ϵ, this contradiction shows these two hybrids must be $(t-O(q), q-1, \epsilon)$ indistinguishable. Then with the triangle inequality we have our final result: (t, q, ϵ) find-then-guess security implies $(t - O(q), q-1, q\epsilon)$ left-or-right security.

7.3 Weak Find-Then-Guess

There is also the notion of weak find-then-guess security, where the attacker no longer needs to find the m_0 and m_1. For E_k to be weak find-then-guess secure, the following has to hold for all A, m_0, m_1:
$Pr[A^{E_k}(E_k(m_0)) = 1] - Pr[A^{E_k}(E_k(m_1)) = 1] \le \epsilon$

Find-then-guess security implies weak find-then-guess security, since if there is an attacker A that distinguishes $E_k(m_0)$ from $E_k(m_1)$ in the weak find-then-guess setting, then we can create a find-then-guess attacker that distinguishes as well as A does. In the "find" stage, our attacker B will just return $(m_0, m_1, -)$, and in the "guess" stage, our algorithm will just run A.