Secure Computational Models: Weekly Paper 2

Siddarth Ijju

September 24th, 2018

1 Introduction

This week's lectures focused on the concept of a library and how libraries can be used to prove the security of different computational models. We explored how to use libraries to prove the security of systems such as OTP and the Secure Sharing System.

2 Libraries

2.1 Definition

We defined a library as a collection of subroutines/functions that has private/static/internal variables. We define the symbol $\mathscr{L}$ to stand for a library. From this, we also defined several other terms:

1. $A \cdot \mathscr{L}$: a program A calls a subroutine in $\mathscr{L}$
2. $A \cdot \mathscr{L} \Rightarrow z$: a program A calling a subroutine in $\mathscr{L}$ returns z

A library can be further defined using the following conditions:

1. A library provides interfaces (main method, arguments, types, return types, etc.) to its subroutines
2. The subroutines of a library can be randomized
3. External programs cannot view the internal state of a library

2.2 Interchangeability

The concept of interchangeability that we discussed in class can be summarized by the following: if two libraries have exactly equal probability distributions for their outputs, (ie. it is impossible to tell which library was used to provide a particular output) then they are interchangeable. In more formal terminology:

1. Two libraries $\mathscr{L}_1$ and $\mathscr{L}_2$ are two libraries with the same interface. They are interchangeable if any program A that outputs 0 or 1 satisfies the condition that $\Pr(A \cdot \mathscr{L}_1 \Rightarrow 1) = \Pr(A \cdot \mathscr{L}_2 \Rightarrow 1)$

In our first lecture, we used this definition to show that the a library containing the real OTP (One-Time Pad) function and another containing a randomly generated binary string were equivalent. We defined two libraries $\mathscr{L}_{\text{OTP Real}}$ and $\mathscr{L}_{\text{OTP Rand}}$.

1. $\mathscr{L}_{\text{OTP Real}}$ contained a function that took in as input a binary string of length λ (m) called Query(m). Query (m) performed the following steps:
   (a) $k \leftarrow \{0, 1\}^\lambda$
   (b) returns $k \oplus m$, where $\oplus$ symbolizes the XOR operation
2. $\mathscr{L}_{\text{OTP Rand}}$ contained the same interface as $\mathscr{L}_{\text{OTP Real}}$ except the Query function performed a different step:
   (a) returns a randomly samples binary string $c \leftarrow \{0, 1\}^\lambda$

We then showed that because the probability distributions of the outputs are equivalent, the libraries are interchangeable.

2.3 Distinguishers

We briefly discussed the concept of a distinguisher in class. A distinguisher is defined as a program whose output can indicate which of two libraries was used to produce a given output. If a distinguisher exists for any pair of libraries, it shows that the libraries are not distinguishable because there would exist a program A that could determine which library was used for a particular input. For example, we defined the function Enc(m) to take a permutation of $k$of the numbers from 1 to some $n$and then return the binary string c where for every i from 1 to n, $c[k[i]] = m[i]$. We then defined two libraries $\mathscr{L}_{\text{left}}$ and $\mathscr{L}_{\text{right.}}$

1. $\mathscr{L}_{\text{left}}$ contained a function that took in as input a two binary strings of length λ (m_l, m_r) called Query(m_l, m_r) that returns Enc(m_l).

2. $\mathscr{L}_{\text{right}}$ contains the same interface, only the function Query(m_l, m_r) returns Enc (m_r).

We then were able to find a distinguisher based upon the cardinality of m_{l} and of m_{r}. We also defined another distinguisher as the program A where A computes r = Query(0^λ, 1^λ) where λ is an arbitrary length for the binary strings. A then returned r= 0^λ. The program A is a distinguisher because the Enc function only permutes the digits of the binary string input. Therefore by giving Query two binary strings of different cardinalities, it is possible to tell which library was used since permutations of a string do not change the string's cardinality.

3 Proving Interchangeability

The next topic we covered was the method in which two libraries could be proved to be interchangeable. The method we learned and utilized the most was the method of hybridization. This method consisted of making incremental changes to a library until we showed that the library could be interchanged with another equivalent library. The first example we used this method was on the following libraries:

1. $\mathscr{L}_1$ is defined as containing a function Q($m\epsilon\{0,1\}^\lambda$) which performs the following steps
   (a) $x \leftarrow \{0,1\}^\lambda$
   (b) $y = x \oplus m$
   (c) return (x, y)

2. $\mathscr{L}_2$ is defined as containing a function Q($m\epsilon\{0,1\}^\lambda$) which performs the following steps
   (a) $y \leftarrow \{0,1\}^\lambda$
   (b) $x = y \oplus m$
   (c) return (x, y)

We then used the method of hybridization to incrementally change $\mathscr{L}_1$ until is was equivalent, and thus interchangeable with $\mathscr{L}_2$. We did this by first separating steps (a) and (b) into another function called Enc. We then removed the XOR operator on $x$in the Enc function because we had previously proved that $x \oplus m$ has an equivalent probability distribution as simply x. We then proceeded to switch the variables x and y and then retrace our steps to arrive at the conclusion that $\mathscr{L}_1$is interchangeable with $\mathscr{L}_2$ or symbolically $\mathscr{L}_1 \equiv \mathscr{L}_2$.

4 Secret Sharing Systems

A secret sharing scheme follows the general concept of splitting a secret message m into $n$shares/parts. The first type of secret sharing system is called the t-out-of-n system

4.1 t-out-of-n Secret Sharing System

The general premise of a t-out-of-n SSS (Secret Sharing System), or TNSSS, is that for any number of shares less than t, the message m cannot be deciphered. However, if any t shares are chosen, the message m can be deciphered, with $t$being less than n. A t-out-of-n SSS (Secret Sharing System) can be defined with the following conditions

1. t and n are both constants
2. a Share function (which creates shares)
   (a) an input of m
   (b) an output of $n$shares, $s_1, s_2, s_3, \ldots s_n$
3. a Reconstruct function (which reconstructs shares)
   (a) input is any $t$or more shares
   (b) output is m

4.2 n-out-of-n Secret Sharing System

The second type of SSS (known as an NNSSS) is when exactly $n$shares are required to decode a message m. This could be equivalated with a TNSSS where $t$is replaced by n. For an NNSSS, the Share and Reconstruct functions can be defined more explicitly given what we have discussed so far. We defined

1. Share(m), where m is a binary string representing the message, as a function that performs the following steps
   (a) s = an array of n λ-bit strings
   (b) for i= 1 to $n-1$: $s_i \leftarrow \{0,1\}^\lambda$
   (c) $s_n = s_1 \oplus s_2 \oplus s_3 \oplus \ldots \oplus s_{n-1} \oplus m$
   (d) return $s = (s_1, s_2, \ldots, s_n)$
2. Reconstruct$(s_1, s_2, \ldots, s_n)$
   (a) return $\oplus$ of all inputs because that would output m since any binary string $\oplus$ itself is equivalent to 0^λ where λ is the length of the binary string.

4.3 TNSSS Security

We then began to discuss how we might go about proving that a TNSSS system is actually secure. We started by letting $\sum$ represent an arbitrary SSS. The system would will be secure if we can prove that $\mathscr{L}^{\sum}_{\text{left}} \equiv \mathscr{L}^{\sum}_{\text{right}}$.

1. $\mathscr{L}^{\sum}_{\text{left}}$contains an interface that consists of a function Query with 3 inputs-$m_l, m_r, \mathcal{U}$, where $\mathcal{U}$ is the set of user ids. Query then performs the following operations with the inputs
   (a) if $\mid \mathcal{U} \mid \geq t$, where t is the number of shares, then return error
   (b) else
      i. $s \leftarrow \sum.Share(m_l)$
      ii. return $\{s_i, \mid i \epsilon \mathcal{U}\}$
2. $\mathscr{L}^{\sum}_{\text{right}}$contains the same interface as $\mathscr{L}^{\sum}_{\text{left}}$except m_r is used in step (b) of the Query function instead.