CS 4501-6501 Topics in Cryptography Wednesday, March 4

Lecture 15

Lecturer: Mohammad Mahmoody *Scribe: Kevin Clark*

1 Functional Encryption review

In session 15 we generalized our discussion of functional encryption, covered how to define security for it, extended it to cover functions that take multiple inputs and ended with an inquiry into obfuscation. To review, let us start with a concise, formal definition of a functional encryption scheme that is notationally convenient for our discussion of security. (all standard public key scheme algorithms begin with the prefix PK)

Let the set X be the plaintext space. A Functional Encryption scheme consists of the following four algorithms for all $x \in X$.

1. FE.Master_Key_Gen $\rightarrow (pk, mk)$

This is the start of a functional encryption scheme, it is a randomized process by which a key pair is generated, one public key (pk) to be released, and one master key (mk) to be kept secret by the entity running the scheme.

2. FE.Keygen$(mk,\ f(.)) \rightarrow sk_{f(.)}$

This is the key generation step. Here $f(.)$ specifies the function we wish to use the system for, and $sk_{f(.)}$ is the secret key generated for that function.

3. FE.Enc$_1(pk,\ x) \rightarrow c$

This is the encryption step whereby a value (x) is encrypted using the public key established during the setup, yielding the ciphertext c.

4. FE.Dec$(sk_{f(.)},\ c) \rightarrow$ **y**

This is the heart of the functionality we desire, it is where we use $sk_{f(.)}$ to compute f(x) from c; The equality $y =$ f(x) always holds.

2 Security Game

Defining security for a functional encryption is a non-trivial task. Intuitively, we would like for an attacker to learn nothing about how to compute $f_n(x)$ given access to the ciphertext FE.Enc(pk, x) $\rightarrow c$ and any number of secret keys from functions $f_i(.)$, such that $i \neq n$. This means that using the scheme in any way does not help an attacker learn anything about a new function he/she has not used the scheme on. To formalize this we define the following security game.

We define security against an adaptive adversary that can ask for any number of secret keys $sk_{f_i(.)}$ of the attacker's choice from the challenger. Once the attacker obtains all the secret keys he wishes, he provides the challenger with two inputs, x_1, and x_2. The challenger then provides the adversary with either FE.Enc(pk, x_1) or FE.Enc(pk, x_2) chosen at random. If the attacker has a secret key $sk_{f_i(.)}$ for some i where $f_i(x_1) \neq f_i(x_2)$ then he can answer the challenge by simply running $f_i(x_1)$ and FE.Dec($sk_{f_i(.)}$, c). If they are equal then the challenger provided him with FE.Enc(pk, x_1), if not then he gave him FE.Enc(pk, x_2).

As such, we must impose the following restriction upon the adversaries choice of x_1 and x_2:

(1): $f_j(x_1) = f_j(x_2)$ for all j where the attacker has $sk_{f_j(.)}$

With this restriction in place, the game can be defined as follows:

- Setup: Challenger runs FE.Master_Key_Gen $\rightarrow (pk, mk)$ and gives the Adversary pk

- Query: Adversary submits queries $f_i(.)$ for $i = 1, 2, ...$ and is given FE.Keygen(mk, $f_i(.)$) $\rightarrow sk_{f_i(.)}$

- Challenge: Adversary submits two inputs, x_1, x_2, that satisfy (1) and is given c = FE.nc(pk, x_b), where b is chosen at random from the set 1, 2.

- Guess: Adversary may issue further queries or ask the challenger for

FE.Dec($sk_{f_i(.)}$, c). Adversary eventually guesses whether c = FE.Enc(pk, x_1) or c = FE.enc(pk, x_2).

The adversary wins if his chance of guessing correctly is less than or equal to 1/2 + a negligible amount, $neg(n)$

3 Constructing Fucntional Encryption

Having established this definition, our desire naturally extends to looking at how we can construct function encryption schemes for an *arbitrary* set of functions.
With a public key encryption scheme in hand, it is useful to examine how we could construct such a scheme for a single function.

F := $\{f_1()\}$

1st try: Let $sk_{f_1()} = mk$

This obviously opens the security game to a trivial attack. The adversary simply queries for the secret key for $f_1()$ and is given the master key mk. When given the challenge c, he simply runs PK.Dec(mk, c), which would yield either x_1 or x_2 and allow him to win the game with probability 1.

As a new approach, let us change how we encrypt the data. Instead of encrypting x, let us encrypt f(x). By doing this, the attack in the above scenario no longer works because providing the Adversary with the decryption key no longer allows him to learn the identity of x, only $f(x)$

Let F = $\{f_1(.), f_2(.)....f_n(.)\}$, an arbitrary set of functions

We modify the Master Key Generation step and Encryption steps to produce a public-private key pair for each function and encrypt any input x into a cyphertext containing the encryption of x according to all of the keys.

FE.Master_Key_Gen $\rightarrow \{\{pk_{f_1(.)}, pk_{f_2(.)}....pk_{f_n(.)}\}, \{mk_{f_1(.)}, mk_{f_2(.)}....mk_{f_n(.)}\}\}$

FE.Keygen$(pk_{f_i(.)}, f_i(.)) \rightarrow sk_{f_i(.)}$, where $sk_{f_i(.)} = pk_{f_i(.)}$

FE.Enc$_2$(x) $\rightarrow$ {FE.Enc$_1$($pk_{f_1(.)}$, $f_1(x)$), FE.Enc$_1$($pk_{f_2(.)}$, $f_2(x)$)....FE.Enc$_1$($pk_{f_n(.)}$, $f_n(x)$)}

The ciphertext now contains, in vector form, an encrypted version of $f_i(x)$ for every i.

3.1 Proof of security

Let us examine the security game we defined within this encryption scheme given a set, Z, of n functions

- Setup: Challenger runs Master_Key_Gen $\rightarrow$ $\{\{pk_{f_1(.)}, pk_{f_2(.)}....pk_{f_n(.)}\}$, $\{mk_{f_1(.)}, mk_{f_2(.)}....mk_{f_n(.)}\}$ and gives the challenger the set $\{pk_{f_1(.)}, pk_{f_2(.)}....pk_{f_n(.)}\}$

- Query: Let us say that the adversary exhaustively queries the challenger for the entire set of secret keys, thereby obtaining the set $\{sk_{f_1(.)}, sk_{f_2(.)}....sk_{f_n(.)}\}$, this is, of course, unnecessary as $pk_{f_i(.)} = sk_{f_i(.)}$ but is included for notational consistency. At this point the adversary has learned all possible information since he has the entire set of secret keys.

- Challenge: the adversary submits two inputs, x_1 and x_2 such that $x_1 \neq x_2$ and $f_i(x_1) = f_i(x_2)$. Because of the way we changed the encryption scheme, the adversary returns, in vector form, the ciphertext

c = {FE.Enc$_1$($pk_{f_1(.)}$, $f_1(x_b)$), FE.Enc$_1$($pk_{f_2(.)}$, $f_2(x_b)$)....FE.Enc$_1$($pk_{f_n(.)}$, $f_n(x_b)$)}.

Because the adversary has the entire set of decrption keys he can decrypt this vector into

{$f_1(x_b)$, $f_2(x_b)....f_n(x_b)$}

if b = 1, then this set is {$f_1(x_1)$, $f_2(x_1)....f_n(x_1)$}
if b = 2, then this set is {$f_1(x_2)$, $f_2(x_2)....f_n(x_2)$}

Let us assume that the adversary can win the game. This means that he can tell the difference between these two sets, but because $f_i(x_1) = f_i(x_2)$ for all i, these two sets are indentical, and therefore indistinguishable. This is a contradiction, and therefore the adversary cannot win the game.

4 Multi-Input Functional Encryption

We now extend our definition of Functional Encryption to cover function classes that contain functions with multiple inputs. Informally, we want to allow a user who obtains a secret key $sk_f(.)$ and the ciphertexts $c_1 = \text{Enc}(x_1)$, $c_2 = \text{Enc}(x_2)$... $c_n = \text{Enc}(x_n)$, to compute the function f(x_1, x_2... x_n) without learning anything else about x_1, x_2... x_n.

There are several key differences from functional encryption for unary functions:

1. Firstly, the set of functions, F, we wish to define the scheme for now contains functions that take an arbitrary number of inputs, $f : x^n \rightarrow x$

2. It follows that our decryption algorithm must now be Dec(c_1, c_2, $sk_{f_i(.)}$) $\rightarrow f(x_1, x_2...x_n)$, where Dec($mk$, c_1) = x_1, Dec(mk, c_2) = x_2... Dec(mk, c_n) = x_n

4.1 extension to obfuscation

When try to build functional encryption for 2 inputs, we can construct an interesting reduction that allows us to perform obfuscation. Obfuscation is the process by which a Challenger can give an adversary the ability to run a program without learning anything about it.

Suppose we want to obfuscate a circuit C:

We first generate public and private key sets for a 2 input Functional encryption scheme. Now consider the function

(2) $f(.,.)$ where $f(C, x) = C(x)$

The objuscation of C would be then would be the tuple {Enc(pk, C)= α, $sk_{f(.,.)}$,}. Since C is encrypted we do not know anything about it or how it works

We could then Enc(pk, x) $\rightarrow c$, and fun our 2 input functional decryption algorithm Dec(Enc(pk, C), c) $\rightarrow f(C, x) = C(x)$.

.