TAMPERE UNIVERSITY – DEPARTMENT OF COMPUTING SCIENCES

Security Protocols: Helping Alice and Bob to Share Secrets (COMP.SEC.220)

Tutorial 7: Introduction to Functional Encryption

Antonis Michalas antonios.michalas@tuni.fi
Mindaugas Budzys mindaugas.budzys@tuni.fi
Hossein Abdinasibfar hossein.abdinasibfar@tuni.fi

August 19, 2024

SUBMISSION DEADLINE: 28.10.2024 AT 23:00

Tutorial Description

The purpose of this week's tutorial is to give students a basic understanding of how Functional Encryption (FE) schemes work. By successfully completing the exercises outlined in the tutorial, students will gain a better understanding of how to install and make use of FE libraries, and how to perform secure operations with the aforementioned schemes. By successfully completing the exercises outlined in the tutorial, students will be better equipped to tackle coursework II.

Alice has recently learned about a new encryption technique, named Functional Encryption (FE). With this technique she can securely store her encrypted data and allow a third party to perform certain operations on the encrypted data without knowledge of the original content. In this scenario, Alice can use FE to generate a functional encryption key and share it with Bob (untrusted party). This key allows Bob to perform certain functions, such as inner-product, on the encrypted data and get a decrypted output of the result. With this Bob can find out the result of the function and does not require access to the original data.

As was mentioned in Lecture 7c, a standard FE scheme contains 4 PPT algorithms: $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. FE schemes generate three or more distinct keys: master public key (mpk), master secret key (msk), and a number of functional encryption keys (sk_f). Both the mpk and msk are generated by passing a security parameter λ to a key generation (aka Setup) function:

$$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$$

These keys can be generated through various techniques, but for this example we cover the ElGamal (DDH) instantiation technique. An ElGamal scheme holds 5 parameters:

- X – the private key (not shared in the scheme);
- Y – the public key;
- G – the generator;

- P – the modulus (a large prime);
- Q – the order of G.

algorithm 1 shows how to generate these parameters.

Algorithm 1: Key Generation for ElGamal

Input: λ
Output: $\mathsf{param}_{\mathsf{EIG}}$
$P \leftarrow rand()$;
$\frac{P-1}{2} \rightarrow Q$;
$G \leftarrow (rand())^2 \ (mod\ P)$;
$X \leftarrow rand() \ (mod\ P)$;
$G^X \rightarrow Y \ (mod\ P)$;

These parameters are used to configure a DDH scheme that holds 2 additional parameters:

- L – the bit length of the modulus (the scheme operates in the $\mathbb{Z}_p$ group);
- $Bound$ – The value by which coordinates of input vectors are bounded.

This allows us to generate the master public and secret key pairs (algorithm 2).

Algorithm 2: Key Generation for DDH

Input: $\mathsf{param}_{\mathsf{EIG}}, L, Bound$
Output: msk/mpk
for i in range$(0, L)$: **do**
    $X \leftarrow rand() \ (mod\ P)$;
    $\mathsf{msk}[i] \leftarrow X$;
    $G^X \rightarrow \mathsf{mpk}[i] \ (mod\ P)$;
end

These keys work in the same sense as a traditional PKE key pairs, that is, mpk is used to encrypt data into a ciphertext while msk is used to decrypt the ciphertext. The person who has access to the msk can decrypt **everything** encrypted with the mpk. On the other hand, sk_f are generated with a separate algorithm, which derives a secret key for a specified function f using msk:

$$\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$$

A generated sk_f calculates the function f on encrypted data and outputs the decrypted result of $y = f(x)$. This key for inner-product calculations is derived as such (y is an input vector that is bounded by Bound):

$$\mathsf{sk}_f = \mathsf{msk} \cdot y \ (mod\ Q)$$

In Figure 1 we provide a simple example of how an FE scheme would be used between Alice (A) and Bob (B). **Note**, this protocol is not considered secure and is used only for presentation purposes! In this example, Alice generates the key pairs mpk/msk and then encrypts her data x (assume $x = [1,2,3]$, a vector of three integers) using mpk to get c. Bob will then request that he wants to know the sum of Alice's data ($f = \mathsf{Sum}(\cdot)$). After receiving the request Alice would generate a functional encryption key sk_f using msk and the requested function f. Alice would then send sk_f and c to Bob. Bob would use the received values to decrypt c with sk_f and receive the final value ($y = \mathsf{Sum}(x) = 6$).

Let's now test out a few FE schemes and help Alice decide, which one meets her set goals. **Note**, you may use any implementation language for this task and can make use of any FE library available as long as it achieves all the tasks listed below. Examples of possible libraries are GoFE (Go language library) or CiFEr (C language library). When writing the report, mention and link the library you used.

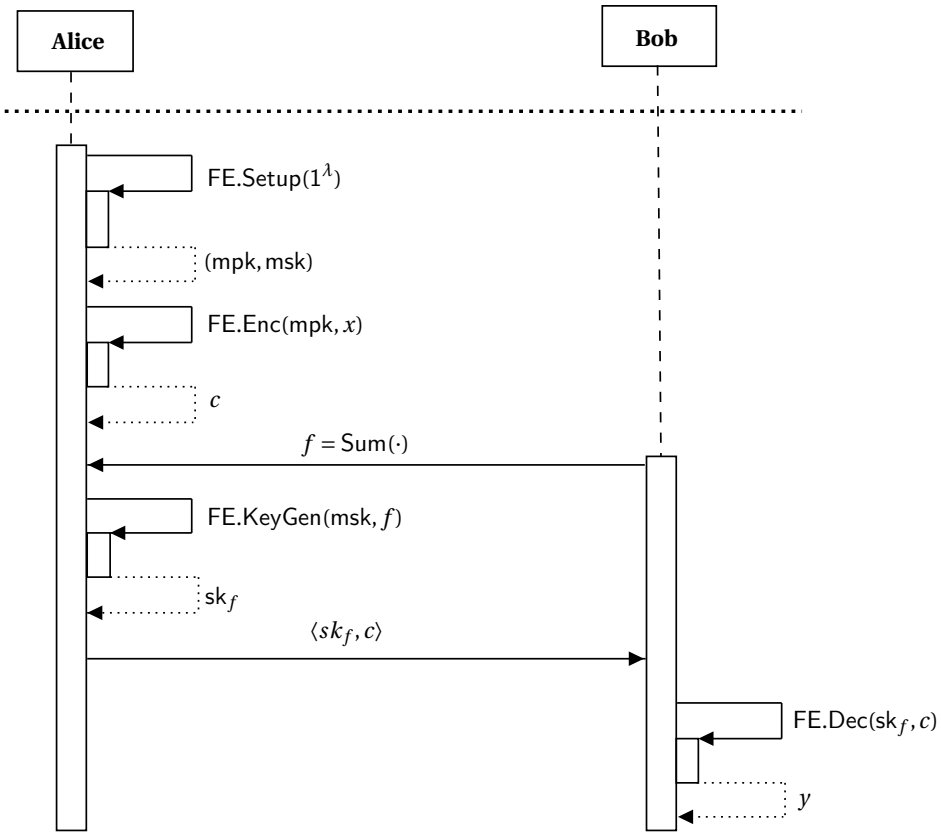

Figure 1: Simple FE protocol between Alice and Bob

Exercise 1 – Generation and Computation

For this exercise, we want to test out a **s-IND-CPA** (selective security under chosen-plaintext attacks) FE scheme (instantiate each scheme with an Elgamal/DDH scheme). We require you to implement a code and executes the following points:

- Generate the mpk/msk key pair used for encryption and decryption, respectively. Also generate a sk_f for calculating an inner-product function $(x_1 \cdot z_1 + \ldots + x_n \cdot z_n)$;
- Request a user to input two integer vectors (x and y) of the same size, x should be encrypted (algorithm 3), and y should remain in plaintext. The vector length should be larger than 2 and you may freely choose the numbers, however in your report manually calculate the inner product (in plaintext) to confirm that the code is working as intended. The program should encrypt the vector x with the generated mpk and output the resulting encrypted vector c;

Algorithm 3: Encryption Algorithm for DDH

Input: mpk, $\mathsf{param}_{\mathsf{ElG}}$, x
Output: c
while $2 < r < Q$ **do**
  $r \leftarrow rand()$;
end
$G^r \rightarrow ct_0 \ (mod\ P)$;
$c = ct_0$;
for i in range$(0, len(x))$ **do**
  $\mathsf{mpk}^r * G^{x_i} \rightarrow ct_i \ (mod\ P)$;
  $c = c||ct_i$;
end

- Decrypt the encrypted vector with sk_f (algorithm 4). How do the results differ?

Algorithm 4: Decryption Algorithm for DDH

Input: $\mathsf{sk}_f, \mathsf{param}_{\mathsf{EIG}}, c, y$
Output: res
$\prod_{i=1}^{n} ct_i^{y[i]} \rightarrow num \ (mod\ P)$;
$ct_0^{\mathsf{sk}_f} \rightarrow d \ (mod\ P)$;
$num * d^{-1} \rightarrow r \ (mod\ P)$;
$res =$ Solve DLP, when $P, Q, G, r, Bound$ are known;

- Implement timers to measure the execution times for key generation, encryption and decryption with sk_f. Measure the time it takes to encrypt different vector lengths (1, 3, 5, 10 and 15 elements). Add these results to a table in your report. How do these results compare to asymmetric encryption schemes such as RSA from the previous tutorial?

Exercise 2 – Evaluation of Different Schemes

Repeat the same experiments as previously but test three different FE schemes, which:

1. Allows multiple inputs from **two** different clients;
2. Has a security guarantee of **IND-CPA**;
3. Allows for decentralized key generation.

Briefly describe the schemes you picked. Report the results of the experiments in a table. Are there any noticeable differences? Discuss in what case, Alice should choose each FE scheme and what are the merits of each scheme.