TAMPERE UNIVERSITY – DEPARTMENT OF COMPUTING SCIENCES

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

Tutorial 2: Cryptography I

Antonis Michalas
antonios.michalas@tuni.fi

Mindaugas Budzys
mindaugas.budzys@tuni.fi

Hossein Abdinasibfar
hossein.abdinasibfar@tuni.fi

August 30, 2024

SUBMISSION DEADLINE: 16.09.2024 AT 23:00

Tutorial Description

In this week's tutorial, you will implement some of the basic cryptographic concepts described in lecture 2. At the end of this session, you will get a hands-on experience on hashing, symmetric encryption, and asymmetric encryption algorithms.

EXERCISE 1 – SYMMETRIC ENCRYPTION

In this exercise, you are tasked with implementing a simple AES encryption and decryption program. As described in lecture 2c, symmetric encryption algorithms involve the use of a single secret key (i.e. the same key to encrypt and decrypt a message). In this exercise, you are to generate your own key and implement a simple encryption and decryption program (using AES-256). How would you exchange your generated key with a fellow student if you wish to communicate with him/her in a secure manner?

To complete this task, you are required to write a program that:

- Generates a symmetric key K for encryption and decryption.
- Requests a user to enter a text that should be encrypted. The program should encrypt the text with the generated K and output the resulting ciphertext. (An encrypted message is called the ciphertext);
- To decrypt a given ciphertext, the program should use the same K used for encryption. The program should output the resulting plaintext upon completion. (The original message in cryptography is called the plaintext);
- Implement timers to measure the execution times for both encryption and decryption. (Do you observe any difference?)
- Encrypt the text "Cryptography is the building block of Security Protocols" and send the ciphertext along with your encryption key (encoded in a readable format, i.e.***base64***) to either Mindaugas or Hossein (emails are provided above).

EXERCISE 2 – ASYMMETRIC ENCRYPTION

For this exercise, we turn our attention to asymmetric encryption (Public-Key cryptography). As described in your lecture 2d, an asymmetric encryption involves the use of two keys. A private key for decryption and a corresponding public key for encryption. These two keys are linked in a mathematical way such that knowing the public key tells you nothing about the private key, but knowing the private key enables you to decrypt a message encrypted with the corresponding public key. You are tasked with implementing the popular RSA and ECC cryptographic algorithms.

To complete this exercise, you are required to write a program that:

- Requests a user to enter a chosen name. This name should be used as an identifier for the private and public key pair. The program should generate the key pair based on the RSA scheme. (i.e. Alice_Private_Key and Alice_Public_Key).
- Generate txt file with one paragraph of text that will be encrypted. Encryption should be done with the user's public key
- Implement a decryption function that uses the user's generated private key.
- Implement timers to measure the execution times for both encryption and decryption. (Do you observe any differences as compared to exercise 2?)
- Implement the ECC variant of your program using the same steps described above. (Do you observer any differences in execution times?)

EXERCISE 3 – HASHING

In lecture 2e, cryptographic hash functions was explained to you. Hash functions provide data integrity and message authentication. The most widely used hash functions are SHA-256, SHA-512, SHA-1, SHA-3 and MD5 but for the purposes of this tutorial, we focus on the SHA-256 algorithm. In this task, you are tasked with generating the hash of file and verifying the integrity of the hashed file.

To complete this exercise:

- Generate a simple .txt file containing a paragraph of text;
- Write a simple script that takes as input the generated file and compute the SHA-256 digest for the file;
- Once the hash digest has been computed, send the file along with the SHA-256 digest to a coursemate;
- Add a function to your script earlier to verify the integrity of the file you received. (How will you accomplish this?)
- Observe what happens when you change a single letter in your initial text file.