TAMPERE UNIVERSITY – DEPARTMENT OF COMPUTING SCIENCES

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

# Tutorial 8: Introduction to Symmetric Searchable Encryption

Antonis Michalas
antonios.michalas@tuni.fi

Mindaugas Budzys
mindaugas.budzys@tuni.fi

Hossein Abdinasibfar
hossein.abdinasibfar@tuni.fi

August 19, 2024

**SUBMISSION DEADLINE:** 04.11.2024 23:00

**Tutorial Description**

The purpose of this week's tutorial is to give students a basic understanding of how a Symmetric Searchable Encryption (SSE) scheme works. By successfully completing the exercises outlined in the tutorial, students will gain a better understanding of the SSE scheme introduced in Lectures 7 and 8.

Alice wishes to securely store her files with a Cloud Service Provider (CSP) without revealing the contents of the file to the CSP. Furthermore, Alice wants Bob to be able to search for specific keywords that may be contained in the files stored on the CSP without also revealing any information to the CSP. To achieve this, Alice decides to use a Symmetric Searchable Encryption (SSE) scheme. In an SSE scheme, a user encrypts their data locally with a key not known to a CSP and sends the encrypted data to a CSP. The user or another user with the key is able to search over the encrypted data without revealing the data contents or the searched keyword tot he CSP (untrusted party).

Lets help Alice out by implementing a simple SSE program that meets her set goals. **Note** that this implementation is *not* considered secure. However, completing this lab will be a valuable first step towards successfully completing the second coursework of the course!

## EXERCISE 1 – CREATING A DICTIONARY

An SSE scheme is heavily dependent on a dictionary that maps encrypted keywords to their respective encrypted files as well as the keys used for the encryption. To create this table, you are required to build a database using a database program of your choice. In the formulae, || denotes concatenation.

1. Create a database with two (2) tables, namely, **sse_csp_keywords** and **sse_keywords**;
2. **sse_csp_keywords** should have three (3) rows; *csp_keywords_id*, *csp_keywords_address*, and *csp_keyvalue*;
3. **sse_keywords** should have four (4) rows; *sse_keywords_id*, *sse_keyword*, *sse_keyword_numfiles*, and *sse_keyword_numsearch*;

4. Take a screenshot of both tables and add it to your submission;

5. Write your program to access the .txt files in a specified folder. For each .txt file in that folder, your program should extract all the words in the file and then encrypt the file with AES (you used AES in tutorial 2) using a generated key $K_{SKE}$;

6. For each extracted word, do the following:
   - Create a SHA-256 hash of the word as the **keyword**. Store as the *sse_keywords* value;
   - Initialize the *sse_keyword_numfiles* and *sse_keyword_numsearch* values. The *sse_keyword_numfiles* value should be a counter of the number of files containing the particular keyword;
   - Compute a **keywordkey** $K_w$;
     $K_w = \mathsf{SHA256}(\mathsf{keyword}||\mathsf{numsearch})$
   - Compute the *csp_keywords_address* and *csp_keyvalue* values;
     *csp_keywords_address* = $\mathsf{SHA256}(K_w||\mathsf{numfiles})$
     *csp_keyvalue* = $\mathsf{Enc}_{K_{SKE}}(\mathsf{filename}||\mathsf{numfiles})$
   - Insert the generated values into their respective fields in the tables you created.

7. Delete the original .txt files ensuring that your folder contains only the encrypted files from task 5. Take a screenshot on a successful run of your program;

8. Measure the time taken by the program to create the dictionary and encrypt 25 .txt files of varied sizes. What was the total combined size of your test files?

9. Note that *csp_keywords_id* and *sse_keywords_id* are simply incremental values that identify entries in both tables.

## EXERCISE 2 – SEARCHING IN THE DARK

Alice has successfully encrypted and stored her files with the CSP without leaking any information (hopefully). Now Bob wants to search over the encrypted data for a specific word. Lets help him to do this. Implement a search function in your program to accomplish this task.

1. The program should ask Bob to enter a word he wishes to search for;

2. Create a SHA-256 hash of the word to be searched for. This becomes the **keyword** value;

3. Using the generated keyword value, retrieve the respective *numfiles* and *numsearch* from the **sse_keywords** table;

4. Compute the **keywordkey** $K_w$;
   $K_w = \mathsf{SHA256}(\mathsf{keyword}||\mathsf{numsearch})$

5. Compute a value for the *csp_keywords_address* field of the **sse_csp_keywords** table;
   *csp_keywords_address* = $\mathsf{SHA256}(K_w||\mathsf{numfiles})$

6. Using the generated *csp_keywords_address* value, retrieve the *csp_keyvalue* values from the **sse_csp_keywords** table;

7. Retrieve the filename from the returned *csp_keyvalue* values by decrypting with $K_{SKE}$;
   $\mathsf{filename}||\mathsf{numfile} = \mathsf{Dec}_{K_{SKE}}(\textit{csp\_keyvalue})$

8. Decrypt the file.

9. Measure the time taken to search for a keyword.