# CS 5158/6058 Data Security and Privacy, Fall 2019
# Project 3: Searchable Encryption

Instructor: Dr. Boyang Wang

**Due Date:** 12/4/2019 (Wednesday), 11:59pm.
**Format:** Please submit a zip file of your code in Blackboard.
**Total Points:** 15 points
**Note:** This is a **group project** (the same group as your final project).

## 1 Project Description

In this project, you will need to implement a searchable encryption scheme by using inverted index, pseudo random function (PRF), and AES. More specifically,

- For the key generation function, given a parameter $\lambda$, your program should be able to output two secret keys $(sk_1, sk_2)$, one key $sk_1$ for PRF and one key $sk_2$ for AES, and write your secret keys to two different files.
- For the encryption function, given a set of files $\{f_1, ..., f_n\}$ and a set of keywords $\{w_1, ..., w_m\}$, your program should be able to output an encrypted inverted index $\Gamma$ and a set of encrypted files $\{c_1, ..., c_n\}$, where each file is encrypted as
$$c_i \leftarrow \mathsf{AES.Enc}(sk_2, f_i)$$
In addition, your program should be able to write this encrypted inverted index to a file.
- For the token generation function, given a keyword $w$ and a secret key $sk_1$, you program should be able to output a search token $tk$, and writes this token to a file.
- For the search function, given a token $tk$ and an encrypted inverted index $\Gamma$, you program should be able to output the file identifiers associated to this token, and decrypt the corresponding encrypted files based on those identifiers.

Note: For each group member, please submit a copy of your project in Blackboard. This will assist the instructor and the TA to record your grade easily.

## 2 Basic Requirements

**Programming Language:** You can use **either C/C++, Python or Java**. If you choose to use C/C++, CMake is recommended (but not required). You can choose any IDE you like, the code you submit should be able to compile and run in Linux or Windows.

**Crypto Libraries:** You can leverage third-party libraries, such as `openssl` (C/C++), `BouncyCastle` (Java), etc., to implement AES and SHA256. You do not need to build the functions of AES or SHA256 by yourself.

**Program Directory:** Please name your project folder as `se_m123456`, where `se` is the name of this project and `m123456` is your UCID. The recommended directories of your program should be organized as follows:

```
./se_m123456/src
./se_m123456/build
./se_m123456/data
./se_m123456/report.pdf
```

Normally, folder `src` should include all the source files and your own header files, e.g., `.cpp` and `.h` files. All the object files and executable files, e.g., `.o` files, should be under folder `build`. Folder `data` has all the given files and data, and also includes all the files and results generated by the program. In `report.pdf` file, you should describe which OS you use, show which language and version you use, and illustrate how to compile, run and use your code. In addition, you also need to include screenshots for the outputs of each function in your report (please see details in the next section).

## 3 Project Details

Assume we have a set of .txt files under a same folder, and each file contains multiple words. In this project, we assume the words in each file are separated by space, and each word will be counted as a keyword. For example, a file $f_1$ includes 3 words

```
../data/files/f1.txt:  bengals  steelers  packers
```

where those words are separated by space. All the 3 words are counted as keywords. In other words, we say file $f_1$ has keywords `bengals`, `steelers` and `packers`. In this project, we assume words in those files are all English words, no special characters, symbols, or numbers.

**You program should be generic**, i.e., it should be able to handle any number of files (under a same folder) and any number of words in each file. For ease of project description, let us assume we have 6 files $\{f_1, f_2, f_3, f_4, f_5, f_6\}$ and 4 keywords $\{\texttt{bengals}, \texttt{steelers}, \texttt{packers}, \texttt{patriots}\}$ in this example. Specifically, in each file, we have

```
../data/files/f1.txt:  bengals  steelers  packers
../data/files/f2.txt:  packers  patriots
../data/files/f3.txt:  packers
../data/files/f4.txt:  steelers  bengals
../data/files/f5.txt:  steelers  packers
../data/files/f6.txt:  bengals
```

And according to this set of files and keywords, its inverted index in plaintext is described as below

```
bengals  f1.txt  f4.txt  f6.txt
packers  f1.txt  f2.txt  f3.txt  f5.txt
steelers  f1.txt  f4.txt  f5.txt
patriots  f2.txt
```

In this project, assume we use file names as file identifiers.

**1. Key Generation Function:**

(a) Given a parameter $\lambda$, generate two secret keys, one secret key $sk_1$ for PRF and one secret key $sk_2$ for AES, write PRF's secret key $sk_1$ to a file "`../data/skprf.txt`" and write AES's secret key $sk_2$ to a file "`../data/skaes.txt`".

(b) For AES encryption, you should choose AES-CBC-256. For PRF, you can use either AES-ECB-256 or SHA256 to simulate a PRF. If you use AES-ECB-256 to simulate PRF, then $sk_1$ and $sk_2$ share the same parameter, i.e., $\lambda = 256$. If you use SHA256 to simulate PRF, technically, there is no key for PRF, and you could set your key $sk_1$ as null, and the corresponding key file for $sk_1$ should be empty.

(c) There are no requirements in terms of the format of your keys when you write keys to files. You could use any format you like, e.g., binary, hexadecimal, etc., as long as it is consistent with your read functions.

(d) Take a screenshot for the output of your key generation function in step (a) and include it in your report.

**2. Encryption Function:**

(a) Read all the data from the files and read your secret keys from the key files to build an encrypted inverted index. Specifically, each keyword extracted from those files should be encrypted with PRF.

(b) You can use file names as file identifiers. For example, informally speaking, if the outputs of PRF for those keywords are $\mathsf{PRF}_{sk_1}(\texttt{bengals}) = \texttt{QWER}$, $\mathsf{PRF}_{sk_1}(\texttt{packers}) = \texttt{ASDF}$, $\mathsf{PRF}_{sk_1}(\texttt{steelers}) = \texttt{ZXCV}$, $\mathsf{PRF}_{sk_1}(\texttt{patriots}) = \texttt{UIOP}$, then one example of the encrypted index you have would look like the following

```
QWER  c1.txt  c4.txt  c6.txt
ASDF  c1.txt  c2.txt  c3.txt  c5.txt
ZXCV  c1.txt  c4.txt  c5.txt
UIOP  c2.txt
```

The order of the outputs of PRF or the order of file identifiers for each encrypted keyword in your encrypted index could be different. You can use any existing data structures, e.g., hash tables, linked lists, arrays, from C/C++, Java or Python to implement your encrypted index. As long as you can build a correct encrypted index, the running time of building an encrypted index and the search performance will not affect your grade in this project. In this project, you do not need to apply advanced methods, such as adding dummy nodes or encrypting pointers, in your encrypted index.

(c) After building an encrypted inverted index, you need to write the index to a file "`../data/index.txt`". There are no requirements in terms of the format in the index file. You can use any format you like, as long as it is consistent with your read functions.

(d) In addition, you also need to encrypt all the data files, e.g., $\{f_1, f_2, f_3, f_4, f_5, f_6\}$, with AES-CBC-256, and write the ciphertexts to a set of encrypted files $\{c_1, ..., c_n\}$, where each file is encrypted as

$$c_i \leftarrow \mathsf{AES.Enc}(sk_2, f_i)$$

The name of each encrypted file $c_i$ should be `ci.txt`. For example, if a plaintext file is `f2.txt`, then its encrypted file is `c2.txt`. And all the encrypted files should be under a same folder named "`ciphertextfiles`", e.g.,

```
../data/ciphertextfiles/c1.txt
../data/ciphertextfiles/c2.txt
../data/ciphertextfiles/c3.txt
../data/ciphertextfiles/c4.txt
../data/ciphertextfiles/c5.txt
../data/ciphertextfiles/c6.txt
```

There are no requirements in terms of the format in an encrypted file. You can use any format you like, as long as it is consistent with your read functions. For the file identifiers in your encrypted index, please use the file names of those encrypted files as file identifiers (as the example described in (b)).

(e) You program should be as generic, which should be enable to build an encrypted index from a different number of files and a different number of keywords. For ease of implementation, we assume the maximum number of files is 10,000 and the maximum number of keywords is 3,000.

(f) Take a screenshot for the encrypted index you generated in step (a) and (b), and include it in your report.

**3. Token Generation Function:**

(a) Given a keyword $w$, read a secret key $sk_1$ from file "`../data/skprf.txt`", generate a token $tk$, print it in the terminal and write this token to a file "`../data/token.txt`". There are no requirements in

terms of the format in a token file. You can use any format you like, as long as it is consistent with your read functions.

(b) Take a screenshot for the output of your token generation function in step (a) by using keyword `packers` and include it in your report.

**4. Search Function:**

(a) Given a token file and an encrypted index file, read a token from token file "`../data/token.txt`", read an encrypted index from file "`../data/index.txt`", find files associated with this token over this encrypted index, print file identifiers of all the encrypted files associated with this token in the terminal, decrypt corresponding encrypted files with AES-CBC-256, and print the decryption of those encrypted files in the terminal. For instance, if a token is `ZXCV`, then your program should print out information like the following:

```
c1.txt  c4.txt  c5.txt

c1.txt  bengals  steelers  packers
c4.txt  steelers  bengals
c5.txt  steelers  packers
```

In addition, you also need to write the information above to a result file "`../data/result.txt`". If the search result of a token is empty, then the result file should also be empty.

(b) Take a screenshot for the output of your search function in step (a) by using keyword `packers` and include it in your report.

**5. Project Report:**

In addition to screenshots, you should also describe how you implement this searchable encryption in details. Moreover, you also need to test the running time of building an encrypted index on this 6-file example. This running time should include the time to build an encrypted index and also the time to encrypt all the original files. You also need to report the search time for keyword `packers`. This search time should include the time to find those encrypted files and also the time to decrypt those files.

In your report, you should describe details of your implementation, such as OS, programming language, crypto libraries, encryption parameters, etc. You can use tables, figures or screenshots to help you present your results in your report. Please put this report under your project folder and submit it together with your code. **Please make sure you have the names of your group members in your report.**

## 4 Evaluation

Your project will be evaluated in three aspects.

1. **Correctness of Functions (80%):** Your program should be able to correctly run all the functions described in this project. If for some reason, your code cannot be compiled but the logic of your code is correct, you will still get partial credits.
2. **Comments and Descriptions (10%):** Write comments and briefly explain each function in your code, such as inputs, outputs, etc. You may need some of the functions in other projects. Detailed comments on each function can save your time in other projects. In addition, please clearly explain how to compile and run your code in `report.pdf`.
3. **Coding Style (10%):** A good coding style is always important, especially for large projects. Please keep each function simple, try to avoid long functions, and create multiple `.h` and `.cpp` files if needed. For example, it is not a good idea to put everything in the main function.

## 5 Examples

This section provides some examples, which can help you understand the functions of your project. If your code can provide the same functionalities, you can customize the number of arguments and the order of arguments, as long as you describe it clearly in your report.

**Example 1:** The following command calls the key generation function

```
se  keygen  ../data/skprf.txt ../data/skaes.txt
```

where `se` is the name of your executable file, `keygen` is the argument for key generation function, `../data/skprf.txt` is the secret key file for PRF, `../data/skaes.txt` is the secret key file for AES. Note that, depending on where your executable file is, the paths of your input and output files might be different. Security parameter $\lambda = 256$ is default and pre-defined in your program.

**Example 2:** The following command calls the encryption function

```
se  enc  ../data/skprf.txt ../data/skaes.txt  ../data/index.txt
../data/files  ../data/ciphertextfiles
```

where `../data/skprf.txt` is the secret key file for PRF, `../data/skaes.txt` is the secret key file for AES, `../data/index.txt` is the index file, `../data/files` is the folder including all the plaintext files, and `../data/ciphertextfiles` is the folder including all the ciphertext files.

**Example 3:** The following command calls the token generation function

```
se  token  packers ../data/skprf.txt ../data/token.txt
```

where `packers` is a keyword, `../data/result.txt` is the secret key file for PRF, and `../data/token.txt` is the token file.

**Example 4:** The following command calls the search function

```
se  search ../data/index.txt ../data/token.txt ../data/ciphertextfiles
../data/skaes.txt
```

where `../data/index.txt` is the index file, `../data/token.txt` is the token file, `../data/ciphertextfiles` includes all the encrypted files, and `../data/skaes.txt` is a secret key file for AES-CBC-256.