Coursework 1 - Exercise 7

October 11, 2023

Alice wishes to store her files on a Cloud Service Provider (CSP) so that Bob could search for specific words in those files. However, Alice does not trust the CSP and hence, decides to encrypt her files locally before outsourcing them. To maximize the utility of her encrypted and outsourced files, she decides to use a Symmetric Searchable Encryption (SSE) scheme.

Alice chooses a simple SSE scheme (from Tutorial 6), which works in the following way:

- Alice initializes 2 databases: **sse_database** and **sse_csp_database**;
  - sse_keywords should have four (4) rows: *sse_keywords_id*, *sse_keyword*, *sse_keyword_numfiles*, and *sse_keyword_numsearch*;
  - sse_csp_keywords should have three (3) rows: *csp_keywords_id*, *csp_keywords_address*, and *csp_keyvalue*;
- Alice creates a SHA-256 hash of the word and stores as the *sse_keywords* value;
- Alice initializes the *sse_keyword_numfiles* and *sse_keyword_numsearch* values. The *sse_keyword_numfiles* value is a counter of the number of files containing the particular keyword;
- Computes a *keywordkey* K_w;

$$K_w = \text{SHA256}(keyword||numsearch)$$

- And computes the *csp_keywords_address* and *csp_keyvalue* values:

$$csp\_keywords\_address = \text{SHA256}(K_w||numfiles)$$

$$csp\_keyvalue = \text{Enc}_{K_{SKE}}(filename||numfiles)$$

To search for a word:

- Bob enters a word he wishes to search for;
- The word is hashed with SHA-256 and the *numfiles* and *numsearch* values are retrieved from *sse_keywords*;
- Then the values K_w and *csp_keywords_address* are computed in the same way as previously mentioned;
- The generated *csp_keywords_address* is used to retrieve *csp_keyvalue* from *sse_csp_keywords* and retrieve the filename;
- Finally, this file is decrypted and sent to Bob.

a) You are provided with an excerpt from the computed database sse_keywords:

sse_keywords_id	*sse_keyword*	*sse_keyword_numfiles*	*sse_keyword_numsearch*
8	d037e316bcc	17	4

sse_keywords_id	*sse_keyword*	*sse_keyword_numfiles*	*sse_keyword_numsearch*
9	486ea46224d	19	1
10	ba29297b6c4	7	9
11	42be22c2666	28	5
12	3be7a505483	11	0
13	35140dc7eff	1	9

* – this does not contain the full SHA-256 hash, just the first 11 charachters. For your calculations **use the full hash as computed by your chosen tool**. The answer can be provided as the first 11 charachters of the computed hashes

Alice uses a generated AES-128-ECB key **KSKE=8A5DEC68615185AD** (in hexadecimal format) computing the database entries. She decides to add a new file named "important_info.txt". The file contains this text:

```
cranberry cola
```

Using the information provided, please compute a database update which adds this new information to *sse_keywords* AND *sse_csp_keywords*.

[8]:
```
import binascii
from Crypto.Cipher import AES
import pandas as pd
from hashlib import sha256

k_ske = binascii.unhexlify("8A5DEC68615185AD")
k_ske = bytes(16 - len(k_ske)) + k_ske  # add zeros in front to make the key 16␣
 ↪bytes long

def enc(plaintext, key):
    if len(key) != 16:
        raise ValueError(f"Incorrect AES key length ({len(key)} bytes). The key␣
 ↪must be 16 bytes (128 bits) long.")
    plaintext = plaintext.ljust((len(plaintext) // 16 + 1) * 16, b'\0')  # pad␣
 ↪the plaintext with zeros
    return AES.new(key, AES.MODE_ECB).encrypt(plaintext)

data = {
    'sse_keywords_id': [8, 9, 10, 11, 12, 13],
    'sse_keyword': ['d037e316bcc', '486ea46224d', 'ba29297b6c4', '42be22c2666',␣
 ↪'3be7a505483', '35140dc7eff'],
    'sse_keyword_numfiles': [17, 19, 7, 28, 11, 1],
    'sse_keyword_numsearch': [4, 1, 9, 5, 0, 9]
}

sse_keywords = pd.DataFrame(data)
```

```
data = {
    'csp_keywords_id': [],
    'csp_keywords_address': [],
    'csp_keyvalue': []
}

sse_csp_keywords = pd.DataFrame(data)

df = pd.DataFrame(data)


filename, content = "important_info.txt", "cranberry cola"
for word in content.split():
    # computing the hash value of the word
    hash_value = sha256(word.encode()).hexdigest()
    if hash_value[:11] in sse_keywords['sse_keyword'].values:
        sse_keywords.loc[sse_keywords['sse_keyword'] == hash_value[:11],␣
 ↪'sse_keyword_numfiles'] += 1
        print(f"Found {word} in {filename}!")
        n_files = sse_keywords.loc[sse_keywords['sse_keyword'] == hash_value[:
 ↪11], 'sse_keyword_numfiles'].values[0]
        n_search = sse_keywords.loc[sse_keywords['sse_keyword'] == hash_value[:
 ↪11], 'sse_keyword_numsearch'].values[0]

        kw = sha256((word + str(n_files)).encode()).hexdigest()
        csp_keywords_address = sha256((kw + str(n_search)).encode()).hexdigest()
        csp_keyvalue = enc((filename + str(n_files)).encode(), k_ske)
        # turning to hex
        csp_keyvalue = binascii.hexlify(csp_keyvalue).decode()
        # add to dataframe
        sse_csp_keywords = pd.concat([sse_csp_keywords, pd.
 ↪DataFrame({'csp_keywords_id': [len(sse_csp_keywords) + 1],␣
 ↪'csp_keywords_address': [csp_keywords_address[:11]], 'csp_keyvalue':␣
 ↪[csp_keyvalue[:11]]})], ignore_index=True)

# print the dataframes as tables
print("SSE Keywords")
print(sse_keywords)
print("\nSSE CSP Keywords")
print(sse_csp_keywords)
```

```
Found cranberry in important_info.txt!
Found cola in important_info.txt!
SSE Keywords
   sse_keywords_id  sse_keyword  sse_keyword_numfiles  sse_keyword_numsearch
0                8  d037e316bcc                    18                      4
1                9  486ea46224d                    19                      1
```

```
2                10  ba29297b6c4                    7                    9
3                11  42be22c2666                   29                    5
4                12  3be7a505483                   11                    0
5                13  35140dc7eff                    1                    9

SSE CSP Keywords
   csp_keywords_id csp_keywords_address csp_keyvalue
0              1.0          cab731e6943  8f748f489b3
1              2.0          139c1c894a0  8f748f489b3
```

In order for this to work, we added zeros to the key since AES-128-ECB requires 16-Byte keys and the one facilitated had just 8 Bytes

So we can see that the following updates have to be made:

- SSE Keywords:

sse_keywords_id	sse_keyword	sse_keyword_numfiles	sse_keyword_numsearch
8	d037e316bcc	18	4
11	42be22c2666	28	5

- SSE CSP Keywords

csp_keywords_id	csp_keywords_address	csp_keyvalue
1	cab731e6943	8f748f489b3
2	139c1c894a0	8f748f489b3

The SSE CSP Keywords new csp_keywords_id updates will be with an autoincrement addition, the 1,2 where representative

---

b) In your own words, describe the notions of (1) Forward Privacy and (2) Backward Privacy.

The terms Forward and Backward privacy make reference to temporal security, this is: protecting data both in the past and future

1. **Forward Privacy**: An attacker should not be able to access queries in the database in the future. So if it granted access at some point, this should not mean being able to acquire further information in about the searches in the future.
2. **Bacward Privacy**: Similarly, in the case an attacker were to access the database at some point, information from queries from the past should not be revealed to him or the possibility of being able to acess it

---

c) Is this scheme secure? Why/Why not?

If the scheme is not secure, then how could it be improved to make it secure?

Note: A simple Yes/No answer will not be graded. You must provide justification for your answer!

No, given that the number of searches is stored plainly, there is a lack of Backward Privacy