CS 6238 Secure Computing Systems — Project 4
Robert Bobkoskie — Secure Shared Store (3S)

INTRODUCTION

The goal of the project is to gain hands-on experience in implementing secure distributed services. A Secure Shared Store (3S) service will be developed that allows for the storage and retrieval of documents created by multiple users who access the documents at their local machines. In the implementation, the system should consist of one or more 3S client nodes and a single server that stores the documents. Users should be able to login to the 3S server through any client by providing their private key. Session tokens would be generated upon successful authentication of the users. They can then check-in, checkout and delete documents as allowed by access control policies defined by the owner of the document. To implement such a distributed system, certificates are utilized to secure the communication between clients and the server, and to authenticate sources of requests. To achieve secure communications, a Certificate Authority (CA) is created that generates certificates for users, client nodes and the server such that All nodes trust the CA.

I. Architectural Design Details

This was a complex deliverable with many avenues to explore, innovate and create novel solutions to meet the projects requirements. This section will provide a succinct synopsis of the architecture, Additional information on the architecture can be obtained from the video presentation.

a. Assumptions

The following assumptions were made when implementing the requirements for this project:

- In order for a user to checkout a document, it must be checked in to the server.
- For grant, there is an entry that defines time duration T (in seconds) for the grant table, e.g., both checkin and checkout will be dependent on the same time element. I considered creating two entries for grant time; one for checkin and another for checkout.
- The owner of a document is **noted** during checkin and written to a checkin table as persistent data.
- The owner of a document is **verified** during checkout and the checkin table is queried for validation.
- Single user login from the same client is supported. The same user can login from another client.
- User documents are dictionaries: {“`File Name`”: “`File Contents`”}:

```
File Name: [client1-user11.doc], File Contents: [The rain in spain falls mainly on
the plain|<>|A second line in the file]
```

b. Database

To account for persistent data, a database with three tables was used. The table schema is shown below:

```
db_tables = {'login_table':['login_table','id,st,uid,client,status'],

'checkin_table':['checkin_table','did,st,uid,client,sf,fileP,fileN,iv,padding'],

'grant_table':['grant_table','did','tid','access_type','access_time','CurrentT']}
```

c. Login

If a client is authenticated and successfully logs in to the server, the server will pass a session token to the client in the http response message. This Session Token (ST) will be used to authorize the user for subsequent transactions with the server. This is accomplished by concatenating the ST with client data sent to the server, e.g., concatenating the ST with the Document ID (DID).

On the server side, a login table is created and user data, including UID, ST are written. When a checkin transaction is initiated by the client, the server will query the login table for the clients ST, which has been split from the clients DID.

d. Checkin Checkout with Grant

To support requirements for access control policy, the rules for document checkin and checkout with granted access rights are as follows:

- **Checkout**: If a document (DID) that is checked into the server and the user is the owner of the document, then the document can be checked out, else
  - If the user has been granted access to checkout, and
  - If the time granted for the access control is <= current time on the server, and
  - If the access control type is set to checkout [2,3], then
  - The document can be checked out.
- **Checkin**: Similar to checkout, except for the access control type needs to be [1,3]. The snippet below shows the checkin and grant tables. In this scenario, the owner of DID is user1, which will have checkin/checkout privileges to DID. In addition, user2, is granted [2], checkout access control. If the current time is less than the granted time, user2 can checkout DID.

  The tables below show checkin and grant data for an active user session:

```
==== CHECKIN DATA ====
['did', 'st', 'uid', 'client', 'sf', 'fileP', 'fileN', 'iv', 'padding']
 --- (u'client1-user11.doc',
u'LjUmiKr9WVXB3Oog2KxZikFaDYfYME6qELCLZilVtz7UUXmnohLg8dG6ntcZf0NscvgB7XHagpzHNcMICb68R
AP+iLNwSVZw2PXPq98TnZryPOs7R4HyHtS1ydUA93N2AFL++EG03OTag1KdJQ5TUspvPV3e7TbQgIkvfTAP8GLF
N4Gix/FY9ztiQA3bSl79kJWQfO/5ZBjnD3IJNYPfo+CNkUtellgTZaLvDUf24CYEvUznw+kSfwmmoVH3iOGNTeM
Pkf5mblkSKNzzwj9/z6mch1k6cZ3n8tPzEFMPp9/WkxS94BYsZrRWaxgyJWKRv/cBCwkrQtbKOGS9Xg3L2A==',
u'user1_client1', u'client1', u'1',
u'/home/cs6238/Desktop/Project4/server/application/documents/C_client1-user11.doc',
u'C_client1-user11.doc', u'init_vector', u'8')
127.0.0.1 - - [23/Apr/2020 13:05:57] "POST /grant HTTP/1.0" 200 -

==== GRANT DATA ====
['did', 'tid', 'accessType', 'accessTime', 'currentTime']
 --- (u'client1-user11.doc', u'user2_client1', u'2', u'2020-04-23 16:06:16.468481',
u'2020-04-23 16:04:16.468463')
```

The following capture a checkout scenario with granted access control:

```
# From user1 session:
Your Selection: 2
(' ---', 200, u'Document Successfully Checked Out: Confidentiality Verified')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

# From user2 session:
Your Selection: 2
(' ---', 702, u'USER NOT OWNER NOR GRANTED ACCESS TO DOC: Access denied to check out')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

# From the grant table: “user1” has been granted access:
==== GRANT DATA ====
['did', 'tid', 'accessType', 'accessTime', 'currentTime']
 --- (u'client1-user11.doc', u'user1_client1', u'3', u'2020-04-26 11:29:12.234063',
u'2020-04-26 11:27:12.234007')
```

# From server logs: The current login is “TEST USER”, e.g., not the owner of the document (CO_uid == “TEST_USER”), note the TID is “user1” == document owner, thus user1 **can checkout** the document. For the second log, the current login is “user2”, which has not been granted access, thus **will not** be able to checkout the document:

```
CURRENT_USER [user1] CO_uid [TEST_USER]
 === CURRENT_USER [user1]   TID [user1]
127.0.0.1 - - [25/Apr/2020 18:06:42] "POST /checkout HTTP/1.0" 200 –
CURRENT_USER [user2] CO_uid [TEST_USER]
 === CURRENT_USER [user2]   TID [user1]
127.0.0.1 - - [25/Apr/2020 18:06:21] "POST /checkout HTTP/1.0" 200 -
```

e. Delete

When a client initiates a delete transaction, all documents with a matching DID will be deleted from the clients checkin/checkout directories, as well as the servers documents directory. The captures below show a delete scenario. More details on the code can be found in the Implementation section under Delete.

```
# Baseline, documents have been checked into the server, and checked out by the client:

cs6238@CS6238:~/Desktop/Project4/client1/documents/checkin$ date
Sat Apr 25 16:18:03 EDT 2020
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkin$ ls -l
total 8
-rw-rw-r-- 1 cs6238 cs6238   70 Apr 25 16:12 client1-user11.doc
drwxrwxr-x 2 cs6238 cs6238 4096 Apr 24 17:46 DOCS
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkin$ ls -l ../checkout/
total 8
-rw-rw-r-- 1 cs6238 cs6238 70 Apr 25 16:15 client1-user11.doc
-rw-rw-r-- 1 cs6238 cs6238 70 Apr 25 16:15 client1-user11.doc.tmp
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkin$ ls -ltr ../../../server/application/documents/
total 8
-rw-rw-r-- 1 cs6238 cs6238 403 Apr 25 16:15 C_client1-user11.doc.pkl
-rw-rw-r-- 1 cs6238 cs6238 373 Apr 25 16:15 I_client1-user11.doc.pkl

(' ---', 200, u'Login Successful')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

Your Selection: 1
Enter 1 for Confidentiality, 2 for Integrity: 1
(' ---', 200, u'Checkin with Confidentiality Successful')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

Your Selection: 1
Enter 1 for Confidentiality, 2 for Integrity: 2
(' ---', 200, u'Checkin with Integrity Successful')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

Your Selection: 2
(' ---', 700, u'USER NOT OWNER OF DOC and GRANT TABLE NOT CREATED')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

Your Selection: 3
(' ---', 200, u'Successfully granted access')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

Your Selection: 2
(' ---', 200, u'Document Successfully Checked Out: Confidentiality Verified')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout
```

```
# User chooses Delete

Your Selection: 4
(' ---', 200, u'client1-user11.doc successfully deleted')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

# All instances of the file, as well as the keys (which are in the meta_data if the DID. .pkl files):
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkin$ date
Sat Apr 25 16:24:31 EDT 2020
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkin$ ls -l
total 4
drwxrwxr-x 2 cs6238 cs6238 4096 Apr 24 17:46 DOCS
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkin$ ls -l ../checkout/
total 0
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkin$ ls -ltr ../../../server/application/documents/
total 0
```

f. Logout

Upon log out, the users row will be removed from the login table in the DB. In addition, if the user modified a document that was checked out from the server, the document will be sent to the server with the integrity option. A scenario is shown below, additional details can be found in the Implementation section under Logout. For this capture, server-side logs are sprinkled into the client-side transaction for ease of view:

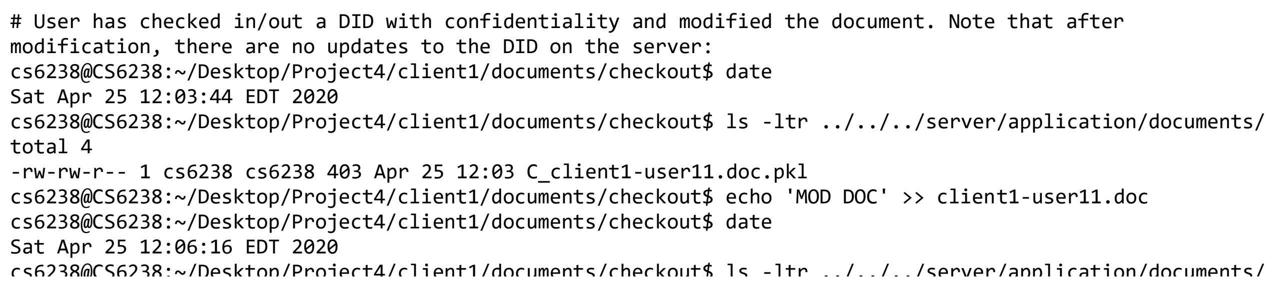
```
# Baseline, user has not checked in or checked out DIDs , so no files in servers documents dir:
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkout$ date
Sat Apr 25 12:03:16 EDT 2020
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkout$ ls -ltr ../../../server/application/documents/
total 0

cs6238@CS6238:~/Desktop/Project4/client1$ python client.py
(' ---', 200, u'Login Successful')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

Your Selection: 3
(' ---', 200, u'Successfully granted access')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

Your Selection: 1
Enter 1 for Confidentiality, 2 for Integrity: 1
(' ---', 200, u'Checkin with Confidentiality Successful')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

Your Selection: 2
(' ---', 200, u'Document Successfully Checked Out: Confidentiality Verified')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout
```



```
# User has checked in/out a DID with confidentiality and modified the document. Note that after
modification, there are no updates to the DID on the server:
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkout$ date
Sat Apr 25 12:03:44 EDT 2020
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkout$ ls -ltr ../../../server/application/documents/
total 4
-rw-rw-r-- 1 cs6238 cs6238 403 Apr 25 12:03 C_client1-user11.doc.pkl
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkout$ echo 'MOD DOC' >> client1-user11.doc
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkout$ date
Sat Apr 25 12:06:16 EDT 2020
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkout$ ls -ltr ../../../server/application/documents/
```

```
# The access time has expired, if this user is not the owner and had not been granted access for
checkin, the checkin of a modified DI on logout would fail. This was addressed by calling a grant
from the clinet (see Logout in the Implemetation section for details).

Your Selection: 2
(' ---', 702, u'ACCESS TIME HAS EXPIRED: Access denied to check out')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

Your Selection: 5
(' ---', 200, u'Successfully granted access')
(' ---', 200, u'Checkin with Integrity Successful')
(' ---', 200, u'client1 as user1_client1 successfully logged out')

# The modified document was checked into the server with integrity:
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkout$ date
Sat Apr 25 12:06:33 EDT 2020
cs6238@CS6238:~/Desktop/Project4/client1/documents/checkout$ ls -ltr
../../../server/application/documents/
total 8
-rw-rw-r-- 1 cs6238 cs6238 403 Apr 25 12:03 C_client1-user11.doc.pkl
-rw-rw-r-- 1 cs6238 cs6238 384 Apr 25 12:06 I_client1-user11.doc.pkl
```

II. Implementation Details

This section will provide details on how the requirements for the project were achieved. There will be code snippets as well as some scenario presentations. Additional details can be found in the video presentation.

a. Requirements

This section will touch on how requirements were fulfilled through discussion of code snippets.

- **Stateful Server**: The server can be restarted during a client session without affecting the client's state. This is achieved with persistent data storage using an SQLite database, as well as through utilization of the users session token (on successful login) for all subsequent transactions. The snippet below shows how this is accomplished for the logout function. This approach was also implemented for checkin, checkout and grant.

```
# session_token is passed to the server in the body of the http post request from the client. The
# session_token was obtained during a successful login by the client:

session_token = str(data['session_token'])
uid = SQL.get_current_user(session_token)
client = uid.split("_")[1]
id = uid.split("_")[0]

# This function in the servers DB_FUNCTIONS class will look up the session_token in the SQL login
# table and, if found, will return the uid:

class DB_FUNCTIONS():
       def get_current_user(self, session_token):
              UID = 'test'
              ID = 'test'

              ################################
              # Query DB Tables
              # login_table':['login_table','id,st,uid,client,status']
              ################################
              table_name = db_tables['login_table'][0]
              conn, error = self.create_connection(os.path.join(path_to_project, 'PROJECT4.sql'))
              SQL_CMD = """ SELECT * FROM {} WHERE st='{}' """
              SQL_CMD = SQL_CMD.format(table_name, session_token)
              # SQL.DEBUG_query_all_rows(conn, table_name)
              user_session, error = self.query_table(conn, SQL_CMD)
              if error:
                     print error
              if user_session:
                     UID = user_session[0]
                     ST = user_session[1]
                     ID = user_session[2]

              if (conn):
                     conn.close()
              return UID
```

- **Mutual Authentication** Is achieved through the exchange of client/server certs during login.
- **Confidentiality and Integrity**: The server supports both options.
    - Confidentiality: A pkl file is used to store document metadata:
    - `ckeckin_metadata_C[str(docID)] = [encrypted_aes_key, encrypted_file, padding, init_vector].`

- Integrity: Also uses a pkl file, although with different metadata to support verification of signed documents. Note that the key used for decrypting the file is stored encrypted: `ckeckin_metadata_C[str(docID)] = [file, signed_doc]`

```
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256
from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import AES
from Crypto.Cipher import PKCS1_OAEP

# Code for 'Confidentiality':
if security_flag == '1':
       SF_RESPONSE = 'Confidentiality'
       file_name = 'C_'+file_name
       encrypted_file, padding, aes_key, init_vector = HF.encrypt_file(file)
       encrypted_aes_key = HF.encrypt_key(server_certs, aes_key)
       ckeckin_metadata_C[str(docID)] = [encrypted_aes_key, encrypted_file, padding, init
       encrypted_file_pkl = os.path.join(server_rel_path, 'documents', file_name+'.pkl')
       with open(encrypted_file_pkl, 'wb+') as f:
              pickle.dump(ckeckin_metadata_C, f, pickle.HIGHEST_PROTOCOL)

# Code for 'Integrity':
if security_flag == '2':
       SF_RESPONSE = 'Integrity'
       file_name = 'I_'+file_name
       signed_doc = HF.encrypt_key(server_certs, file)
       ckeckin_metadata_C[str(docID)] = [file, signed_doc]
       encrypted_file_pkl = os.path.join(server_rel_path, 'documents', file_name+'.pkl')
       with open(encrypted_file_pkl, 'wb+') as f:
              pickle.dump(ckeckin_metadata_C, f, pickle.HIGHEST_PROTOCOL)
```

Output from checkin of a document with confidentiality displaying pkl metadata as well as [en/de]crypted files:

```
=== PKL metadata ===
{'client1-user11.doc':
['\x8aL\xff\xf6\xdf\xf7G?\xb9\x01tfW\xdc\x89\x83k\x1b\xac"\xaa(\xb5\xe0\xbf\xb86\
xff\xeb\xd9\x01h\xec\x01Q\xcb\x8c0Zb\x90\xee\xdd\xb0\x0c\x0cM1\xeb\xf4\x8c\x13"T\
xf2?\x8f\xa4\xbe@\xd9\xb3\xf30]\xb6jb\xd4J\x8a\x8c\xd8F\x87\xb2\xc0h\x7f\xa3{8W
\'\xe6\xc0\xac1^\xb5\x05\xf8\'z\x94eV\xba\xbd2\xc6\x1fL[\xfe\x85\x18\x96\xd8\x046
\r\x8ct<\x9f^6\x8bj\xc3\x10\xef\xc9T\x87m2\xdb]\xaa\xd2\xb3>\xbc\xffr\x80\xc4\x8d
b_\x90T\xafs\x1c\n\xa98\xba\xe1\xcdA\xeaC\xa9\xbf\x02I^\xb8\xee\xdd\xaf.\x1d.\xea
$|\xd5*}\xc0P\xed\xff\xe8sK\x8b\x02o\x00\x1d\xc5\xd9\xca\x8f\xf7\xdb\xc8\xbd\xb0\
xa2C\xbd\x1e\xb2\x91\xae\xc0\xc1b\xa5\xc5\x83\x8f\r\x99\xd6\x1eC0\x16\x93\x9c\xc0
\x0b\xed\xaf,)\x93\x91\x06\xcd]SX\xcd{U\xfb(\xea@:&\x1c\xf4\xbf\xc7\xec\r\xa4\xe6
z\x0eb\xa3\x86d\xc2',
'\xc7\xe81H\xc2\t\x16\xe7\xfa8\xd8\x81\x87\xb3\x01\xfc\xcc\xac\xe1g\x14\tl\x84\x1
9\x80\x0b\xc9\x85\x7ft\x11\xfc\xa9\xe9\xafT\x8d\x8d\xd7\x8b%\x81u\x02Y\xf1\xd40\x
ad\n\x19\x95\x12q\xc5]P\xc4\x9f\xbc\xed\x9bj\x9f4\xfc/@s^\x83\x9a\x85h\xbd\xcd\xf
b: ', 8, '\xd9)\xc1\xb3#\xe6\xf6\x16U\x96\\,\x9c\xd2\xcf\xf1']}

=== Confidentiality Decrypted Doc ===
The rain in spain falls mainly on the plain|<>|A second line in the file

=== Confidentiality Encrypted Doc ===
Çè1HÂ  çú8Ø³üÌ¬ág   l
      ×%uYñÔ0-
qÅ]PÄ¼í4ü/@s^        É
h½Íû:
127.0.0.1 - - [26/Apr/2020 15:42:51] "POST /checkin HTTP/1.0" 200 -
```

Output from checkin of a document with integrity displaying pkl metadata as well as [un]signed files:

```
=== PKL metadata ===
{'client1-user11.doc': ['The rain in spain falls mainly on the plain|<>|A second
line in the file',
"Vw\xe9o\xeaA\xe54\x89\xf0\xfbF\xa5\xe4\xbf\xe61F0\x95\x9at\x95a\xa4\x89%!2L4i\xb1\x
d5\xe0s\xa0\x88\x0c\x86\xb2\x0e\xb0c\x93\xa0\xcf\xc9\x11=\xef@\xf7l6\\\xe0\x99~4\xf7
\xc1\xd8\xc7\xd4\x04l\xc4Q
\xfa\x16Q\x862\xf3:\x1d\x02v\r\xf8Y\xac\xcc?\x94;\x9a\xdb\xecB\xdf\x84r\xca\xd47\xa8
\xd1\xdc\xfcqSU8\x00\xf7/\x13\x94\xa4\xcb\x87\xf7Ms\xe4\xe0\xe7\x85\x00\xeb\x1ffu\x0
f\x04\x8c\x00g\xc5V\x93\x072\xc7\xf9\x1a\xee#\xb0\xafUi\\\xde\xf0K\xf4~\xac\xde\xf7B
'\x10/\xa7\xf9nM\xc0|\x97\xc7\x8d\xa5\x12\x96\xe4\x07\x8d\x93\xac\xb4fu\x0fc\xa3k\xe
5\x1bwp\x99\xd0\xbc\xc34{\xe8Yn\xf3\x1fW\xa7\x94\r\x8c\x8c\xc6H\x01K\x81\x8b1\x19\x9
1zC\xd5\xfa\xda\x80@I\x81\x80\xcf\xe3L\x81tG9\xf6\x80\xfc\x1b\xe3\xf4\xdae\xa9\x18.f
\xe7v\x97\xa7\xa67/\x86\xf0M\xc5\xe4\xc2R\xcc"]}

=== Unsigned Doc ===
The rain in spain falls mainly on the plain|<>|A second line in the file

=== Signed Doc ===
VwéoêAå4ðûF¥ä¿æ1F0ta¤%!2L4i±Õàs
ÆHK1zCÕúÚ@IÏãLtG9öüôÚe©.fçv§¦7/ðMÅäÂRÌfuc£kåpÐ¼Ã4{èYnóW§
127.0.0.1 - - [26/Apr/2020 15:43:35] "POST /checkin HTTP/1.0" 200 -
ëfugÅV2Çùî#°¯Ui\ÞðKô~¬Þ÷B'/§ùnMÀ|Ç
```

b. Database

SQLite was utilized to account for the persistent data requirement for this project. The following code was utilized to connect and query a table in the DB. A similar approach was used to write and delete data:

```
################################
# Create DB and Tables
################################
conn, error = SQL.create_connection(os.path.join(path_to_project, 'PROJECT4.sql'))
               error = SQL.create_table(conn, login_table)

################################
# Query DB Tables
#       param db_tables: A dict that is keyed by table names containing a list of the ['table
#       name', 'table schema']
################################
# status = checkin(id, docs=['client1-user11.doc', 'abd123.doc'], db_tables)
table_name = db_tables['login_table'][0]
conn, error = SQL.create_connection(os.path.join(path_to_project, 'PROJECT4.sql'))
               user_session, error = SQL.query_table(conn, id, table_name)
if (conn):
       conn.close()
if user_session[0] == id:
       response = {
       'status': 403,
       'message': 'Login Failed '+client+' as '+uid+' already logged into server'

       }
```

c. Login

The login table entry with user data. The session token (st) for a user is stored in this table, and referenced for all transactions following authenticated login. In this capture, note the DID is meta data comprised of the document name and st:

```
==== DID ====
client1-user11.doc
bBDqo1IVgP7weBPbzRuQDujQuC3hHSZfesAf5kmeLUMFWuJcj8E8KFFSSd5G+USFKZ17QWmPZKrYG4/S+MsNz1FG/05+WvaEdt
AaZaX5zgtbQn35L8SDPM0z9EaUgN/dwcLqIh7M0YIoE9XRm2V2bP1jFL42JuInzoiYLUHTmy47GFfNVAkNTQs1h0Hb/UneFAmq
NnuXGTRSt/sDbouny6052kHVOEbSG8KyemFAc8LK3kfAZBXKkj6ebnbw9G6ifVyQCG4P7J4oDKtVT+ZNQ892ZxFQC7rSm3H2RC
NesBNS//VBDPv3NYJbP+JHGlvzSaYY/xMvuxAf43A7jJShVA==

==== LOGIN TABLE ====
['id', 'st', 'uid', 'client', 'status']
 --- (u'user1_client1',
u'bBDqo1IVgP7weBPbzRuQDujQuC3hHSZfesAf5kmeLUMFWuJcj8E8KFFSSd5G+USFKZ17QWmPZKrYG4/S+MsNz1FG/05+WvaE
dtAaZaX5zgtbQn35L8SDPM0z9EaUgN/dwcLqIh7M0YIoE9XRm2V2bP1jFL42JuInzoiYLUHTmy47GFfNVAkNTQs1h0Hb/UneFA
mqNnuXGTRSt/sDbouny6052kHVOEbSG8KyemFAc8LK3kfAZBXKkj6ebnbw9G6ifVyQCG4P7J4oDKtVT+ZNQ892ZxFQC7rSm3H2
RCNesBNS//VBDPv3NYJbP+JHGlvzSaYY/xMvuxAf43A7jJShVA==', u'user1', u'client1', u'ACTIVE')
```

Single user login is accomplished by storing the ID of the user as the primary key in the login table. When a used with the same ID tries to login, the request will be rejected by the DB and thus the server. For this project, the ID is a concatenation of userX_clientY, thus userX can login from multiple clients:

```
 login_table = """CREATE TABLE IF NOT EXISTS login_table (
                                    id TEXT PRIMARY KEY,
                                    st TEXT NOT NULL,
                                    uid TEXT NOT NULL,
                                    client TEXT NOT NULL,
                                    status TEXT NOT NULL);"""
```

d. Checkin Checkout with Grant

The grant function was implemented to support an access time reset. That is, if a user U is logged in and grants access control privileges for time T to another user (TID), that TID will have privileges until T expires. If expired, U can choose to grant privileges to TID, thus resetting TIDs access control rights. The scenario and snippet are shown below:

```
# Values for TID in server.py that were used for testing grant:
TID = 'user2'  # FOR TESTING, COMMENT OUT
TID = '0'      # FOR TESTING, COMMENT OUT

# U has granted privileges to TID for checkout and TID has checked out the document:
Your Selection: 2
(' ---', 200, u'Document Successfully Checked Out: Confidentiality Verified')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

# Output from server log:
* Detected change in '/home/cs6238/Desktop/Project4/server/application/server.py', reloading
* Restarting with stat
* Debugger is active!
* Debugger PIN: 869-876-731
127.0.0.1 - - [24/Apr/2020 15:49:18] "POST /checkout HTTP/1.0" 200 -
       ==> Current User [user1]   TID [user2]
127.0.0.1 - - [24/Apr/2020 15:49:26] "POST /checkout HTTP/1.0" 200 -

# The access time for privileges has expired for TID, thus the document cannot be checked out:
Your Selection: 2
(' ---', 702, u'ACCESS TIME HAS EXPIRED: Access denied to check out')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

# U resets access control for TID with another grant:
Your Selection: 3
(' ---', 200, u'Successfully granted access')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

# Reset successful, thus document can be checked out:
Your Selection: 2
(' ---', 200, u'Document Successfully Checked Out: Confidentiality Verified')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout
```

The code snippet that accomplishes the reset. This was accomplished with an SQL conditional insert (`INSERT or REPLACE into`):

```
################################
# Insert data DB
# checkin_table':['checkin_table','did,st,uid,client,sf,fileP,fileN,iv,padding']
# grant_table: 'did','tid','access_type','access_time','CurrentT
################################
grant = (docID, tid, access_type, GrantT, CurrentT)
'''
       sql_grant_table = """INSERT INTO grant_table
                              (did,tid,accessType,accessTime,currentTime)
                              VALUES (?,?,?,?,?);"""
'''
sql_grant_table = """INSERT or REPLACE into grant_table
                      (did,tid,accessType,accessTime,currentTime)
                      VALUES (?,?,?,?,?);"""
grant_table_lastrowid, error = SQL.add_data(conn, sql_grant_table, grant)
```

This scenario is checkin with grant for user1 and user2 logged in:

```
# user1 checks in the document first, thus is the owner of the document:
Enter 1 for Confidentiality, 2 for Integrity: 1
(' ---', 200, u'Checkin with Confidentiality Successful')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout
# From the server logs:
=== CURRENT_USER [user1] CURRENT_OWNER [user1]

# user2 attmpts checkout, TID == 0, however access time is expited, thus checkin fails:
Enter 1 for Confidentiality, 2 for Integrity: 1
(' ---', 702, u'ACCESS TIME HAS EXPIRED: Access denied to check in')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

=== Current Time [2020-04-26 13:39:15.807460]    GR_accessTime [2020-04-26 13:19:26.927353]
=== Current User [user2]   TID [0]

# user1 resets grant:
Your Selection: 3
(' ---', 200, u'Successfully granted access')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

# Since grant was reset, user2 can now checkout document, server logs confirm access time is valid:
Your Selection: 1
Enter 1 for Confidentiality, 2 for Integrity: 1
(' ---', 200, u'Checkin with Confidentiality Successful')
(1) Checkin
(2) Checkout
(3) Grant
(4) Delete
(5) Logout

=== Current Time [2020-04-26 13:42:48.190113]    GR_accessTime [2020-04-26 13:44:39.678679]
=== Current User [user2]   TID [0]
```

e. Delete

The key concept of the delete transaction is to validate the users credentials before deleting data. This is accomplished by querying the login table to verify the users session token:

```
################################
# Query DB Tables
# login_table':['login_table','id,st,uid,client,status']
################################
table_name = db_tables['login_table'][0]
conn, error = SQL.create_connection(os.path.join(path_to_project, 'PROJECT4.sql'))
SQL_CMD = """ SELECT * FROM {} WHERE id='{}' """
SQL_CMD = SQL_CMD.format(table_name, UID)
user_session, error = SQL.query_table(conn, SQL_CMD)

if not error:
        sql_session_token = user_session[1]

if sql_session_token == session_token:
        for filename in os.listdir(all_files):
                file = os.path.join(all_files, filename)
                try:
                        if os.path.isfile(file) or os.path.islink(file):
                                os.unlink(file)
                                # elif os.path.isdir(file):
                                #       shutil.rmtree(file)
                                success = True
                except Exception as e:
                        print('Failed to delete %s: %s' % (file, e))
                        response = {
                                'status': 700,
                                'message': 'Delete Failed: '+docID+' not found on the server'
                        }
                        if (conn):
                                conn.close()
                        return jsonify(response)
```

f. Logout

This section shows the code behind the scenario when a user logs in, checks in a document to the server, checks that same document out and modifies it. Since the user modified the document, it will be checked back into the server with Integrity:

```
# Get the users ST for the transation from the login and compare with the ST passed to the function in an
http post method. If ST is a match, the users is authorized and can logout:

session_token = str(data['session_token'])
################################
# Query DB Tables
# login_table':['login_table','id,st,uid,client,status']
################################
table_name = db_tables['login_table'][0]
conn, error = SQL.create_connection(os.path.join(path_to_project, 'PROJECT4.sql'))
SQL_CMD = """ SELECT * FROM {} WHERE id='{}' """
SQL_CMD = SQL_CMD.format(table_name, uid)
user_session, error = SQL.query_table(conn, SQL_CMD)
sql_session_token = user_session[1]
# if sql_session_token == '123':         # FOR TESTING, COMMENT OUT
if sql_session_token == session_token:
        error = SQL.delete_data(conn, uid, 'login_table')
```

g. Helper Function Class

I created a class that contains miscellaneous (`HELPER_FUNCTIONS`) that were utilized throughout the server.py module. Just a few are displayed below. Setter and getter variables and functions were located in the `HELPER_FUNCTIONS` class, and were used as global variables to capture the UID for an active session. The `post_request` function was used by other class functions to pass data between other class functions in server.py, e.g., the logout class can call the checkin class.

```
class HELPER_FUNCTIONS():
        current_id_login = ''
        current_st_login = ''
        def set_id(self, id):
                global current_id_login
                self.current_id_login = id
        def get_id(self):
                return self.current_id_login
        def set_st(self, st):
                global current_st_login
                self.current_st_login = st
        def get_st(self):
                return self.current_st_login

        def post_request(self, server_name, action, body, node_certificate, node_key):
                request_url= 'https://{}/{}'.format(server_name,action)
                request_headers = {
                        'Content-Type': "application/json"
                        }
                response = requests.post(
                        url= request_url,
                        data=json.dumps(body),
                        headers = request_headers,
                        cert = (node_certificate, node_key),
                )

                return response
```

The following functions were used to encrypt/decrypt files and keys to meet the Integrity and confidentiality requirements of the project:

```
def encrypt_file(self, file):
        """
        # https://www.novixys.com/blog/using-aes-encryption-decryption-python-pycrypto/
        # https://stackoverflow.com/questions/14179784/python-encrypting-with-pycrypto-aes
        >>> data = '12345 67 8'
        >>> padding = 16 - (len(data) % 16)
        >>> data += chr(padding)*padding
        >>> print data
        12345 67 8
        >>> data = data[:-padding]
        >>> print data
        12345 67 8
        """

        aes_key = ''.join(chr(random.randint(0, 0xFF)) for i in range(16))
        init_vector = ''.join([chr(random.randint(0, 0xFF)) for i in range(16)])
        # aes = AES.new(aes_key, AES.MODE_CFB, iv)
        aes = AES.new(aes_key, AES.MODE_CBC, init_vector)
        padding = 16 - (len(file) % 16)
        file += chr(padding)*padding
        encrypted_file = aes.encrypt(file)

        # data = data[:-data[-1]]
        return encrypted_file, padding, aes_key, init_vector

def decrypt_file(self, file, padding, key, init_vector):
        """
        # https://www.novixys.com/blog/using-aes-encryption-decryption-python-pycrypto/
        # https://stackoverflow.com/questions/14179784/python-encrypting-with-pycrypto-aes
        """

        aes = AES.new(key, AES.MODE_CBC, init_vector)
        decrypted_file = aes.decrypt(file)

        # Remove padding
        decrypted_file = decrypted_file[:-padding]

        return decrypted_file


def encrypt_key(self, server_certs, aes_key):
        """
        https://www.dlitz.net/software/pycrypto/api/current/Crypto.PublicKey.RSA._RSAobj-class.html#decrypt
        https://stackoverflow.com/questions/30056762/rsa-encryption-and-decryption-in-python
        """

        # print type(aes_key), aes_key
        # RSA_key = RSA.importKey(open(os.path.join(server_keys, 'id_rsa.pub'), 'rb+').read())
        RSA_pub_key = RSA.importKey(open(os.path.join(server_certs, 'secure-shared-store.pub'), 'rb+').read())
        encryptor = PKCS1_OAEP.new(RSA_pub_key)
        encrypted_aes_key = encryptor.encrypt(aes_key)

        return encrypted_aes_key

def decrypt_key(self, server_certs, encrypted_aes_key):
        RSA_pri_key = RSA.importKey(open(os.path.join(server_certs, 'secure-shared-store.key'), 'rb+').read())
        decryptor = PKCS1_OAEP.new(RSA_pri_key)
        # print type(decryptor), decryptor
        decrypted_aes_key = decryptor.decrypt(encrypted_aes_key)

        return decrypted_aes_key
```

h. Implementation Issues and Work-Arounds

To achieve the requirements of this project, the following issues were encountered and needed work-arounds. Future work on this project will address these issues:

- Dealing with UTF-8 data to support the confidentiality requirements for the project was a challenge. Inserting UTF-8 and accessing UTF-8 converted data from SQLite tables was attempted. The work-around was to write the init_vector into checkin_metadata, which was stored as a pkl file in the servers documents directory:

```
# Error when attempting to INSERT UTF-8 data (init_vector) to DB:
(' --- CHECKIN: ERROR insert INSERT INTO
checkin_table\n\t\t\t\t\t\t\t\t(st,did,uid,client,sf,file,fileN,iv,padding)
\n\t\t\t\t\t\t\t\tVALUES (?,?,?,?,?,?,?,?,?);', ProgrammingError('You must not use 8-bit
bytestrings unless you use a text_factory that can interpret 8-bit bytestrings (like
text_factory = str). It is highly recommended that you instead just switch your application to
Unicode strings.',))
['st', 'did', 'uid', 'client', 'sf', 'file', 'fileN', 'iv', 'padding']
('NO ROWS', [])

# Error when attempting to ACCESS UTF-8 data (init_vector) when it was successfully INSERTED as
UTF-8 data:
OperationalError: Could not decode to UTF-8 column 'iv' with text
'H}&iuml;&iquest;&frac12;s`&iuml;&iquest;&frac12;e&iuml;&iquest;&frac12;R&iuml;&iquest;&frac12;
D&iuml;&iquest;&frac12;/&iuml;&iquest;&frac12;b'

# Attempts to enter init_vector in SQLite table. The first method enabled the INSERT to work,
however, the init_vector could not be decoded when accesses as a string. The second method
failed to INSERT, as errors were encountered with the lambda function:
    •   conn.text_factory = str
    •   conn.text_factory = lambda x: unicode(x, 'utf-8', 'ignore')

# Successful work-around. Write the init_vector into the checkin_metadata pkl file:
ckeckin_metadata_C[str(docID)] = [encrypted_aes_key, encrypted_file, padding, init_vector]
encrypted_file_pkl = os.path.join(server_rel_path, 'documents', file_name+'.pkl')
       with open(encrypted_file_pkl, 'wb+') as f:
              pickle.dump(ckeckin_metadata_C, f, pickle.HIGHEST_PROTOCOL)
```

- A unique Document ID (DID) was not implemented. I attempted to create a DID of meta data, e.g., a DID comprised of a concatenation of the Document name and the session token for the user session. However, this approach became difficult to manage, and the end-solution was to strip the session token in the server.py classes and use just the document name for the DID:

```
# Concatenated DID [document name + two tabs + session token]:
client1-user11.doc
2QkylBQMeRim0qRKCMTaBPf/nuwiyv0xwa+btpznqHSSRaQE7G1PIKvh2afMLw2acJ8ZvXQPN/pwrPk7FaavW5gw6Croyl5
5yvDFEYcOTd9SJ2MNb6h28P0K9teHFOycLUNWQMUOdJlgJbCbFx8n7wrBKTdovm9KqEbfT5AnjtNt6eOSbCMSNtFy0M8Jx1
9LwpsAUE5As+xZiGkG554WJVEDZ3tK214THv+u4LWdE7Zr4oLXTmDxk/ydgx2r5BXLEmEYsOwRtcsjeXD8qeBO4aoGhgJME
Z/YU8lVkFhSL9u31X2M/GJXAmO0lYZwrfE5MO2c2iupybfMP+rv2v+6YA==

# Checkin table with DID = document name:
==== CHECKIN DATA ====
['did', 'st', 'uid', 'client', 'sf', 'fileP', 'fileN', 'iv', 'padding']
 --- (u'client1-user11.doc',
u'2QkylBQMeRim0qRKCMTaBPf/nuwiyv0xwa+btpznqHSSRaQE7G1PIKvh2afMLw2acJ8ZvXQPN/pwrPk7FaavW5gw6Croy
l55yvDFEYcOTd9SJ2MNb6h28P0K9teHFOycLUNWQMUOdJlgJbCbFx8n7wrBKTdovm9KqEbfT5AnjtNt6eOSbCMSNtFy0M8J
x19LwpsAUE5As+xZiGkG554WJVEDZ3tK214THv+u4LWdE7Zr4oLXTmDxk/ydgx2r5BXLEmEYsOwRtcsjeXD8qeBO4aoGhgJ
MEZ/YU8lVkFhSL9u31X2M/GJXAmO0lYZwrfE5MO2c2iupybfMP+rv2v+6YA==', u'user1_client1', u'client1',
u'1', u'/home/cs6238/Desktop/Project4/server/application/documents/C_client1-user11.doc',
u'C_client1-user11.doc', u'init_vector', u'8')
```

- User login from a second client: I did not implement a check for a second user login from a second client, i.e., a user will be able to log in from multiple clients. For example user1 logs in from client1, user1 logs in from client2. In this scenario, both sessions will be active.

III. Security Analysis of the Implemented Secure Shared Store 3S

I analyzed server.py code with pylint and pyflakes. Pylint had a very verbose output, mostly finding unused variable that were declared, it reported a rating of`9.07/10`. Pyflake had a more succinct output, mostly with the same result as pylint. Future work will consider these suggestions and implement them for P4.v2.

```
cs6238@CS6238:~/Desktop/Project4/server/application$ find . -type f -name "server.py" | xargs pylint
Your code has been rated at -9.07/10 (previous run: -9.07/10, +0.00)

cs6238@CS6238:~/Desktop/Project4/server/application$ pyflakes server.py
server.py:5:1 'time' imported but unused
server.py:6:1 'subprocess' imported but unused
server.py:6:1 'sys' imported but unused
server.py:6:1 'shutil' imported but unused
server.py:7:1 'uuid' imported but unused
server.py:7:1 'certifi' imported but unused
server.py:7:1 'hashlib' imported but unused
server.py:13:1 'Crypto.Random' imported but unused
server.py:171:46 local variable 'e' is assigned to but never used
server.py:311:3 local variable 'CO_did' is assigned to but never used
server.py:312:3 local variable 'CO_st' is assigned to but never used
server.py:316:3 local variable 'path_to_file' is assigned to but never used
server.py:317:3 local variable 'CI_fn' is assigned to but never used
server.py:322:3 local variable 'CURRENT_CLIENT' is assigned to but never used
server.py:369:5 local variable 'GR_did' is assigned to but never used
server.py:447:5 local variable 'security_flag' is assigned to but never used
server.py:504:3 local variable 'ckeckin_metadata_I' is assigned to but never used
server.py:517:28 local variable 'e' is assigned to but never used
server.py:610:5 local variable 'GR_did' is assigned to but never used
server.py:921:3 local variable 'session_token' is assigned to but never used
server.py:929:4 local variable 'id' is assigned to but never used
server.py:930:4 local variable 'client' is assigned to but never used
server.py:1033:40 undefined name 'sql_checkin_table'
server.py:1059:3 local variable 'file_name' is assigned to but never used
server.py:1065:3 local variable 'CURRENT_USER' is assigned to but never used
server.py:1066:3 local variable 'CURRENT_CLIENT' is assigned to but never used
server.py:1080:28 undefined name 'uid'
server.py:1157:3 local variable 'id' is assigned to but never used
server.py:1177:4 local variable 'error' is assigned to but never used
server.py:1331:4 local variable 'ST' is assigned to but never used
server.py:1332:4 local variable 'ID' is assigned to but never used
cs6238@CS6238:~/Desktop/Project4/server/application$
```

a. Shell Security

Running bash script and setting env variables from python shell with root privileges may enable escalation of effective UID (EUID) in the shell that would allow a non-privileged user read/write access to privileged files.

```
# Running client.py as root executes this code:
# $ sudo python client.py
# ('Trigger menu display by HotKey set as:', 'alt+x')
# Press ESC to stop.

# Need to be root to execute this code:
# $ python client.py
# File "/usr/local/lib/python2.7/dist-packages/keyboard/_nixcommon.py", line 174, in ensure_root
# raise ImportError('You must be root to use this library on linux.')
# ImportError: You must be root to use this library on linux.

import keyboard
def on_triggered(id, client):# define function to be executed on hot_key press
       print(text_to_print)

hot_key = 'alt+x'
print('Trigger menu display by HotKey set as:', hot_key)
print("Press ESC to stop.")
keyboard.add_hotkey(hot_key, on_triggered)  # attach the function to hot-key
keyboard.wait('esc')
```

b. Database Security

Database contains sensitive information, e.g., user. I created the DB before starting Topic 13: Database Security . Basics, Inference Attacks & Data Privacy. For future database work, this sensitive data would be removed. Given the time constrains of this project deliverable, modifying the database tables and data were not undertaken.

However, the SQL code in this project follows best practices from python documentation[1].

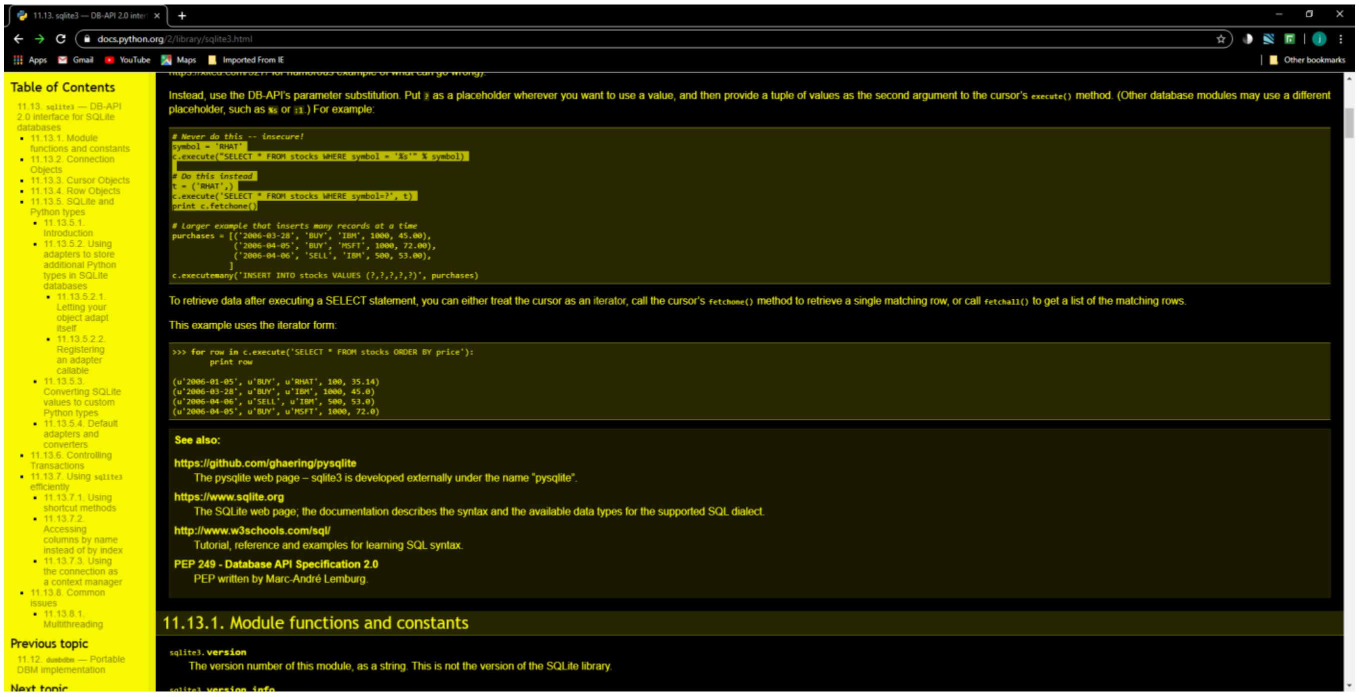

11.13. sqlite3 — DB-API 2.0 inter
docs.python.org/2/library/sqlite3.html
Apps
Gmail
YouTube
Maps
Imported From IE
Other bookmarks

Table of Contents

- 11.13. sqlite3 — DB-API 2.0 interface for SQLite databases
  - 11.13.1. Module functions and constants
  - 11.13.2. Connection Objects
  - 11.13.3. Cursor Objects
  - 11.13.4. Row Objects
  - 11.13.5. SQLite and Python types
    - 11.13.5.1. Introduction
    - 11.13.5.2. Using adapters to store additional Python types in SQLite databases
      - 11.13.5.2.1. Letting your object adapt itself
      - 11.13.5.2.2. Registering an adapter callable
    - 11.13.5.3. Converting SQLite values to custom Python types
    - 11.13.5.4. Default adapters and converters
  - 11.13.6. Controlling Transactions
  - 11.13.7. Using sqlite3 efficiently
    - 11.13.7.1. Using shortcut methods
    - 11.13.7.2. Accessing columns by name instead of by index
    - 11.13.7.3. Using the connection as a context manager
  - 11.13.8. Common issues
    - 11.13.8.1. Multithreading

Previous topic

11.12. dumbdbm — Portable DBM implementation

Next topic

Instead, use the DB-API's parameter substitution. Put ? as a placeholder wherever you want to use a value, and then provide a tuple of values as the second argument to the cursor's execute() method. (Other database modules may use a different placeholder, such as %s or :1.) For example:

```
# Never do this -- insecure!
symbol = 'RHAT'
c.execute("SELECT * FROM stocks WHERE symbol = '%s'" % symbol)

# Do this instead
t = ('RHAT',)
c.execute('SELECT * FROM stocks WHERE symbol=?', t)
print c.fetchone()

# Larger example that inserts many records at a time
purchases = [('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
             ('2006-04-05', 'BUY', 'MSFT', 1000, 72.00),
             ('2006-04-06', 'SELL', 'IBM', 500, 53.00),
            ]
c.executemany('INSERT INTO stocks VALUES (?,?,?,?,?)', purchases)
```

To retrieve data after executing a SELECT statement, you can either treat the cursor as an iterator, call the cursor's fetchone() method to retrieve a single matching row, or call fetchall() to get a list of the matching rows.

This example uses the iterator form:

```
>>> for row in c.execute('SELECT * FROM stocks ORDER BY price'):
        print row

(u'2006-01-05', u'BUY', u'RHAT', 100, 35.14)
(u'2006-03-28', u'BUY', u'IBM', 1000, 45.0)
(u'2006-04-06', u'SELL', u'IBM', 500, 53.0)
(u'2006-04-05', u'BUY', u'MSFT', 1000, 72.0)
```

See also:

https://github.com/ghaering/pysqlite
The pysqlite web page – sqlite3 is developed externally under the name "pysqlite".

https://www.sqlite.org
The SQLite web page; the documentation describes the syntax and the available data types for the supported SQL dialect.

http://www.w3schools.com/sql/
Tutorial, reference and examples for learning SQL syntax.

PEP 249 - Database API Specification 2.0
PEP written by Marc-André Lemburg.

11.13.1. Module functions and constants

sqlite3.version
The version number of this module, as a string. This is not the version of the SQLite library.

IV. Threat Modelling

To accomplish the requirement of writing a modified document that was checked out upon logout, one must consider the condition that the user that checked out and modified the file is not the owner. To account for this condition, the grant function is called from client.py, thus ensuring the user can checkin the modified document. However, since this is an ephemeral grant, the access rights should be as limited in scope as possible. My analysis is shown in the comments below:

```
if os.path.isfile(tmp_file) and os.path.isfile(user_file):
        if not filecmp.cmp(user_file, tmp_file):
                # grant(file_name, '0', 3, 120)     #Security risk, all users: long access time granted for both checkin/out
                grant(file_name, 'id_global', 1, 5) #Better security profile: Short access time with specifi access rights
                time.sleep(1)
                checkin(file_name, '2', server_file)
```

V. Appendix

This section includes miscellaneous elements associated with the project.

a. Unit Testing

White-box unit testing was performed to validate requirements by creating no-sunny-day scenarios, e.g., error conditions; thus error responses from the server. For example, the snippet below shows an approach to execute the code following the conditional evaluation. By modifying the `doc_owner` to a user that does not exist, thus will not match the `cur_login`, the server response will be set accordingly. This approach was used liberally throughout the code.

```
# doc_owner = 'FOR_TEST'        # FOR TESTING, COMMENT OUT
if doc_owner != cur_login:
        response = {
                'status': 702,
                'message': 'Access denied to grant access'
        }
```

[1] https://docs.python.org/2/library/sqlite3.html