Name: Shaik Dawood Shameem Ahamed

Student ID: 7364283

**IN CREATION.PY FILE:**

A screenshot of a computer code

Description automatically generated

First, we import the libraries. Since we are using DSA and DSS we will need to import these libraries. Then we will import SHA256 as that is what we will be using to hash the message. Importing binascii so that we can hex and un-hex the digital signature when we place it in the file and remove it from the file. Next, we place the public key into the file.

A computer screen shot of a program code

Description automatically generated

The file then generates the p, q, and g parameters so that it can generate the public/private key pair. Once the key is generated it is then placed into this “scriptPubKey.txt” file. It will export the key in a binary format and then hex the key. Since the signature file that pairs with this public key only has 1 signature, we will place the OPCODE of 1 into the file. Then we will place the hex of the key into the file and finally since this only has 1 public key, we will place the OPCODE for 1 and the OPCODE of OP\_CHECKMULTISIG at the end of the file.

A computer code on a white background

Description automatically generated

Next is for the 2nd pubkey file. This will have 2 public keys and the signature script that pairs with this will have 2 signatures hence we change the OPCODE to the OPCODE of 2 and at the end of the file we have the OPCODE for the number of public keys.

A screenshot of a computer program

Description automatically generated

The same for the final file for the pubkey portion. It will have 3 public keys instead but there will only be 2 signatures that pair with the public key hence the OPCODE for the signature is the same but the OPCODE for the public key has been changed.

A computer code with text

Description automatically generated with medium confidence

Next, we will create the digital signature. First, we will get the message and hash the message. After that we will open the “scriptSig.txt” file. We will first need to sign the message with the key and then hex the signature. After that we will place the OPCODE for 0 at the beginning of the file and then place the hex of the signature into the file. For this signature it will need the first key, “key1”.

A computer screen shot of a computer code

Description automatically generated

It is the same for the 2nd scriptSig file. This uses the first and second key to sign the messages.

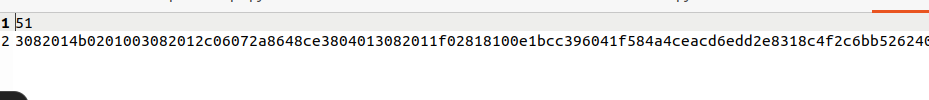
A computer screen shot of a computer code

Description automatically generated

The final portion of the “creation.py” file is the creation of the last scriptSig file where it will use the 2nd and 3rd key to sign the message.

scriptPubKey.txt:

**Beginning of the file:**



**End of the file:**

A screenshot of a computer

Description automatically generated

The OPCODE for 1 is placed at the top, then the key and on the same line at the end of the public key is the OPCODE for 1 again for the number of public keys and the OPCODE for the OP\_CHECKMULTISIG.

scriptPubKey2.txt:

**Beginning of the file:**

A screenshot of a computer

Description automatically generated

**End of the file:**

A screenshot of a computer

Description automatically generated

scriptPubKey3.txt:

**Beginning of the file:**

A screenshot of a computer

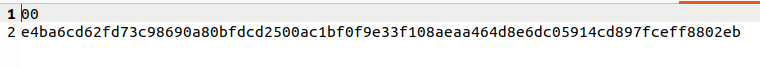
Description automatically generated

**End of the file:**

A screenshot of a computer

Description automatically generated

scriptSig.txt:



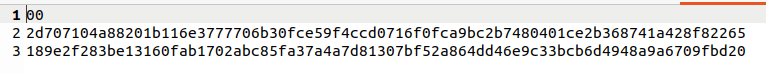
OPCODE for the 0 is placed at the top followed by the rest of the hexed signature.

scriptSig2.txt:

A screenshot of a computer

Description automatically generated

scriptSig3.txt:



**IN P2MSSCIRPT.PY FILE**

A screen shot of a computer code

Description automatically generated

The libraries are the same. Next, we have the function to retrieve the integer values of the OPCODE that are in the text files.

A close-up of a computer code

Description automatically generated

First, we will create the stack. Then we will create the hash object from the message given to check the verification. We also will create the tally to be used later to tally the number of message authentication.

A screen shot of a computer code

Description automatically generated

We will be using “scriptSig3” for this documentation. First, we will open the text file and then read the text file. We will then get the OPCODE and place it in the bottom of the stack. Then we will iterate through the “reading” list and push the digital signatures to the stack and remove the newlines. To prevent exceeding the list limit we place a break to break the loop once it reaches the end of the list.

A screen shot of a computer code

Description automatically generated

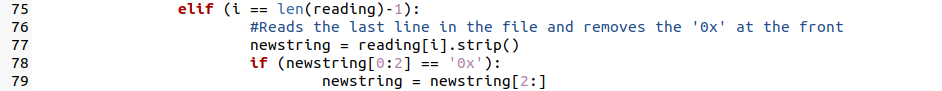
There are 3 portions to this chunk of code. The “if”, “elif” and the “else”. The code is written with the order stated but it runs the “if” portion first then the “else” portion the finally the “elif” portion last.

First the program reads the file and stores it in the reading list. Then it loops through the list. The script takes the OPCODE first and then pushes it into the stack. This is usually the OPCODE for the number of signatures. We will also save it as “tally2” so that we can use it for verification later.

A computer code with black and blue text

Description automatically generated

We have the “else” portion at the end. This reads the rest of the reading list and then pushes the keys to the stack. IT also removes the “0x” at the start and removes all the newline before pushing it into the stack.



Finally, when it reaches the end of the list, it will first remove the newline and the “0x”.

A computer code with text

Description automatically generated with medium confidence

This checks the ending of the key if it contains the OPCODE for OP\_CHECKMULTISIG. If it does not contain the OPCODE, it will exit the program. If it does, the program continues.

A computer code on a white background

Description automatically generated

As the script continues it then takes the OPCODE at the end of the script which is the number of keys in the files and then pushes the final key and then the number at the end of the stack.

A screenshot of a computer code

Description automatically generated

We will then create the list for the keys and then pop the first element of the stack which represents the number of keys. Then it will loop through the list using the number and then pop the key and place it into the list. It will reverse the list to get the proper order. A signature list is also created and the number representing the number of signatures is popped off from the stack and then using the number, the list is then iterated using the number. It will pop the signature from the stack and place it into the list and then list is then reversed.

A computer screen shot of text

Description automatically generated

Flags are created at the start to be used to break out of loops. We will be using a nested loop using the key and the signature lists. First, we will un-hex the signature and then get the key and un-hex it as well. We will then verify the message with the key and digital signature. If the message is authentic, we will add it to the tally and increment the signature list so that the current signature does not iterate through the whole list. The key is also set to ignore so that when the list is iterated, it will ignore the keys that are already checked. If the message is not authentic, it will raise an error but will continue looping to check the other keys and signature. Once the tally is equal to the number of signatures, it breaks out of the whole loop.

A screen shot of a computer code

Description automatically generated

Once the looping is done. The tally is checked and if the tally is equal to the number of signatures, it will clear the stack and push a 1 to the bottom of the stack. The stack is then checked if it contains a 1 and if it does, the script is valid.

**Example used:**

scriptSig3.txt and scriptPubKey3.txt are used for this:

A computer screen shot of a program

Description automatically generated

The scriptPubKey3.txt contains 3 keys while the scriptSig3.txt only has 2 signatures which is the 2nd and 3rd signature. Since there are only 2 signatures, and it tallies with the number of authentic messages, the script is valid