

## CS435 Programming Assignment

Kieran Brooks

200236702

Dr. Qian Yu

### Introduction:

This documentation describes a client server application to implement my solution to the CS435 programming assignment. It comprises client and server components, a pytest testing suite, a utility to generate a text file of keys and a module of user defined classes written in python. The application uses TCP socket communication implemented using the socketserver library to manage the session. The custom classes define the logic of generating, validating and incrementing the hash keys passed between the client and server to verify each other as valid.

My approach uses a hash chain to verify the identity of the message sender by confirming the possession of a shared key by both parties and possession of the next hash in the hash chain. The hash chain is constructed recursively using a shared secret that is concatenated with each generation in the chain and hashed using SHA256. This approach achieves authentication of client and server as well as replay and masquerade protection with respect to the authentication of the correspondent identities. In this assignment the message itself is not made confidential with encryption.

### Strengths:

- This approach is strong against masquerade attacks as both parties are verified to each other for each message. The client and server can be confident that each other has the ability to produce the hash chain exactly the same.
- This approach validates bidirectionally and provides a measure of strength as it adds complexity to orchestrating a MITM attack.
- Because the shared secret is concatenated with each hash generation, a bad actor could not simply guess the hashing algorithm and attempt a selected hashtext brute force attack, as it would be useless without the shared secret and due to the one way property of the hash.
- The typical weakness of hash chain based approaches is the need for distribution of a large quantity of keys in the form of the hashes required. My approach mitigates this by allowing for generation of the chain by both parties and does not require the transmission of the entire chain. This allows for flexibility in the nature of the shared secret. It could be a file, a string, an integer, or any hashable information common to the correspondents.

### Weaknesses:

- This approach is less robust against MITM attacks. Assuming that the message could be captured without leaking the original signal through to the other correspondent, a bad actor could intercept a TCP packet, extract the unencrypted message and replace it with their own and send the packet on to the destination. They would need to continually capture each packet and resend each one in the correct order in both directions to ensure the attack could continue. The attacker would also need to intercept the response from the server, replace the acknowledgement message with the original message and send it on to the client.
- This approach is weak against eavesdropping since the message is sent in cleartext.

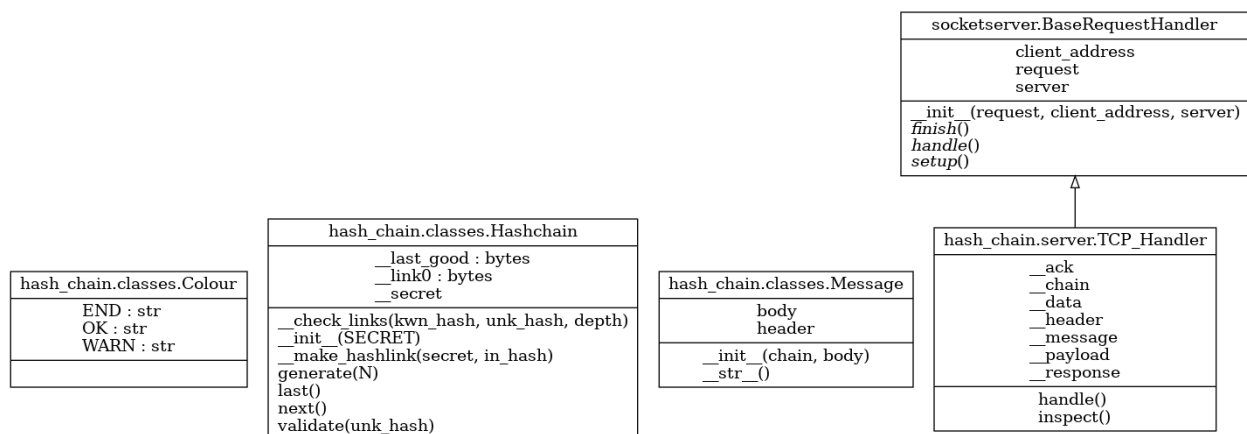
### Extendibility:

- Incorporating message hashing to add non-repudiation to the hash chain.
- Adding encryption to the message construction would be trivial as the client and server already possess a shared key.

### Archive Contents:

server.py	- Initiates the server side
client.py	- Initiates the client side
classes.py	- A user defined module of python classes
generate.py	- A utility to generate a text file of hashes for demonstration purposes
test_classes.py	- A definition file of test functions executed using the pytest utility
requirements.txt	- A list of python library requirements
hash_chain.txt	- A txt file of 500 hashes in a demonstration hash chain

### Documentation:



### **TCP\_Handler class**

TCP\_Handler is a custom class derived from the `socketserver.BaseRequestHandler` class in the `socketserver` library. It defines two methods: `handle` and `inspect`. The `handle` method is called when a client request is received and it reads the incoming data from the socket, unpacks it into a `Message` object, and calls the `inspect` method to display information about the incoming message. The `inspect` method checks if the header of the message is valid by calling the `validate` method of a `Hashchain` object and prints out the result along with the message and the IP address of the client. Finally, it sends an acknowledgement message back to the client. The Handler maintains a decoy hashchain to acknowledge messages that fail authentication.

### **Message class**

Message is a simple class that defines a message with two members: `header` and `body`. The `header` member is hash generated by a `Hashchain` object, while the `body` member is a string passed in as a parameter during object instantiation.

### **Hashchain class**

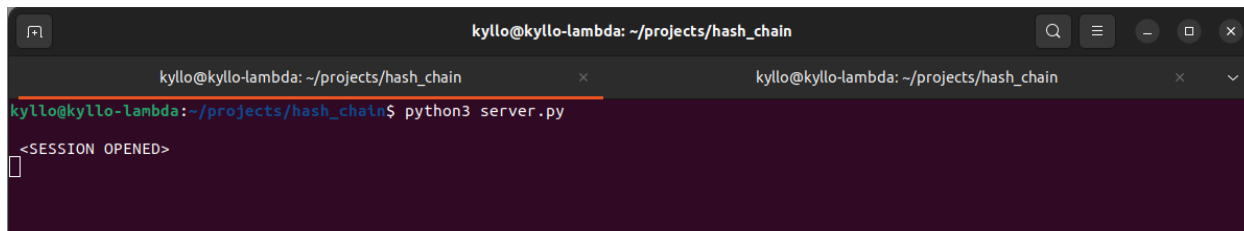
Hashchain is a class that defines a hash chain object. It is initialized with a shared secret and a length `N`, which determines the number of hashes in the chain. It contains several methods for working with the hash chain: `validate`, `next`, `last`, `chain`, and `__generate`. The `validate` method checks if the provided hash is next in the chain, and returns `True` or `False` accordingly. The `next` method returns the next hash in the chain and removes it from the list. The `last` method returns the last validated hash, or an empty byte string if no hash has been validated yet. The `chain` method returns a copy of the hash chain in its current state. The `__generate` method is a private method that generates the hash chain. The `__make_link` method is another private method that constructs an additional link from a secret and the last link.

### **Colour class**

Colour is a simple class that defines ANSI color codes to be used in console output. It contains three members: `OK`, `WARN`, and `END`, which represent the codes for green, red, and default console text color, respectively. These codes are used in the `inspect` method of the `TCP_Handler` class to highlight the verification status of incoming messages.

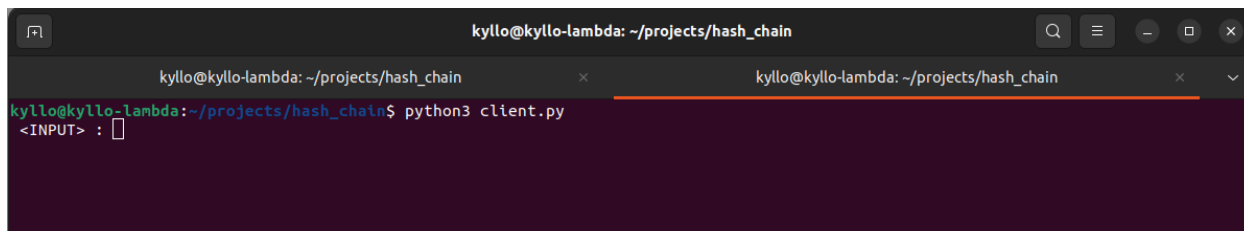
## User Guide:

The server side of the application is started by executing server.py at the command line.



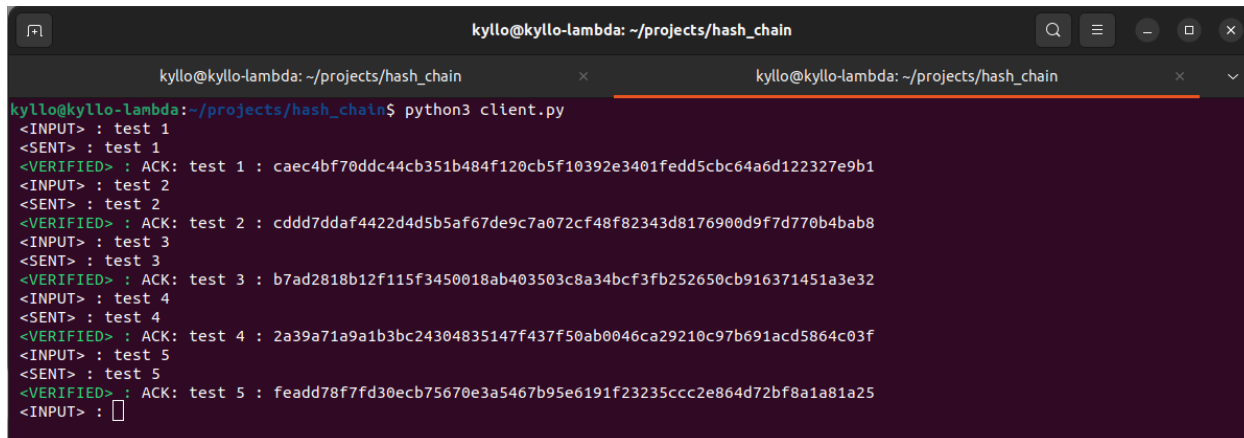
```
kyлло@kyлло-lambda: ~/projects/hash_chain
kyлло@kyлло-lambda: ~/projects/hash_chain
kyлло@kyлло-lambda:~/projects/hash_chain$ python3 server.py
<SESSION OPENED>
```

Next the client is started with client.py, and a connection prompt is presented to the user. Starting a newline will send the input text to the server.



```
kyлло@kyлло-lambda: ~/projects/hash_chain
kyлло@kyлло-lambda: ~/projects/hash_chain
kyлло@kyлло-lambda:~/projects/hash_chain$ python3 client.py
<INPUT> : 
```

The message is packaged and sent to the server, the server validates the hash and responds with an ACK message containing the original message. The validated hash is output to the console.



```
kyлло@kyлло-lambda: ~/projects/hash_chain
kyлло@kyлло-lambda:~/projects/hash_chain$ python3 client.py
<INPUT> : test 1
<SENT> : test 1
<VERIFIED> : ACK: test 1 : caec4bf70ddc44cb351b484f120cb5f10392e3401fedd5cbc64a6d122327e9b1
<INPUT> : test 2
<SENT> : test 2
<VERIFIED> : ACK: test 2 : cddd7ddaf4422d4d5b5af67de9c7a072cf48f82343d8176900d9f7d770b4bab8
<INPUT> : test 3
<SENT> : test 3
<VERIFIED> : ACK: test 3 : b7ad2818b12f115f3450018ab403503c8a34bcf3fb252650cb916371451a3e32
<INPUT> : test 4
<SENT> : test 4
<VERIFIED> : ACK: test 4 : 2a39a71a9a1b3bc24304835147f437f50ab0046ca29210c97b691acd5864c03f
<INPUT> : test 5
<SENT> : test 5
<VERIFIED> : ACK: test 5 : feadd78f7fd30ecb75670e3a5467b95e6191f23235ccc2e864d72bf8a1a81a25
<INPUT> : 
```

On the server side the connection is visible and the result of key validation is visible including the hexdigest of the hash. The server sends an acknowledgement message including another hash in the chain and the client is able to send another message.



## Test Documentation:

These are a set of unit tests for the Hashchain class. Each function tests a specific aspect of the class's functionality, and asserts that the expected behavior is exhibited by the class.

**test\_make\_message():** Tests that a Message object can be constructed with content.

**test\_make\_empty\_message():** Tests that a Message object can be constructed with empty content.

**test\_make\_hashchain():** Tests the Hashchain constructor.

**test\_next\_collision():** Tests that different secrets make different starting links in the hash chain.

**test\_next\_agreement():** Tests that the same secrets make the same hash.

**test\_generate\_N():** Tests that a hash chain of length 500 is generated.

**test\_nodupes():** Tests that there are no duplicates within a hash chain of length 500.

**test\_long\_collision():** Tests for hash collisions in the first 500000 hashes.

**test\_long\_agreement():** Tests for hash agreement at hash[500000].

**test\_detect\_badsecret():** Tests that hashes generated with different secrets will not validate.

**test\_detect\_emptyhash():** Tests that an empty hash will not validate.

**test\_detect\_badhash():** Tests that a random bad hash will not validate.

**test\_validate\_valid():** Tests that hashes will validate on another chain with the same secret.

**test\_detect\_invalid\_order():** Tests that chains generated with the same secret will not validate on each other in generated order.

**test\_return\_last():** Tests that the last hash validated successfully is returned by Hashchain.last().

**test\_return\_last():** Tests that the last hash validated unsuccessfully is not returned by Hashchain.last().

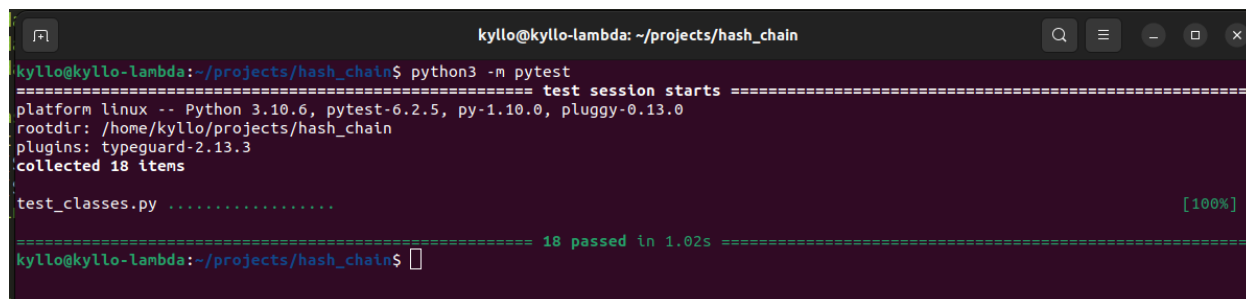
**test\_detect\_replay():** Tests that a hash just validated will not validate immediately after.

**test\_detect\_late\_replay():** Tests that a hash validated in the past will not validate later following other validations.

**test\_simulate\_chat():** Tests the full sequence of steps in a simulated chat between two parties that are communicating using a hash chain.

## Test Results:

Executing the test suite using pytest results in the following output.

A terminal window with a dark background and light green text. The window title is 'kylo@kylo-lambda: ~/projects/hash\_chain'. The command 'python3 -m pytest' has been executed. The output shows the start of a test session, platform information (linux, Python 3.10.6, pytest-6.2.5, py-1.10.0, pluggy-0.13.0), root directory, plugins (typeguard-2.13.3), and 18 collected items. A progress bar for 'test\_classes.py' is shown at 100%. The final result is '18 passed in 1.02s'.

```
kylo@kylo-lambda: ~/projects/hash_chain
kylo@kylo-lambda:~/projects/hash_chain$ python3 -m pytest
===== test session starts =====
platform linux -- Python 3.10.6, pytest-6.2.5, py-1.10.0, pluggy-0.13.0
rootdir: /home/kylo/projects/hash_chain
plugins: typeguard-2.13.3
collected 18 items

test_classes.py ..... [100%]

===== 18 passed in 1.02s =====
kylo@kylo-lambda:~/projects/hash_chain$
```

## Libraries:

“PYTEST,” pytest documentation. [Online]. Available:

<https://docs.pytest.org/en/7.1.x/contents.html>. [Accessed: 05-Mar-2023].

“Hashlib - secure hashes and message digests,” Python *documentation*. [Online]. Available:

<https://docs.python.org/3/library/hashlib.html>. [Accessed: 05-Mar-2023].

“Pickle - python object serialization,” Python *documentation*. [Online]. Available:

<https://docs.python.org/3/library/pickle.html?highlight=pickle#module-pickle>. [Accessed: 05-Mar-2023].

“Socket - low-level networking interface,” Python *documentation*. [Online]. Available:

<https://docs.python.org/3/library/socket.html?highlight=socket#module-socket>. [Accessed: 05-Mar-2023].

“Socketserver - A Framework for network servers,” Python *documentation*. [Online]. Available:

<https://docs.python.org/3/library/socketserver.html?highlight=socketserver#module-socketserver>. [Accessed: 05-Mar-2023].