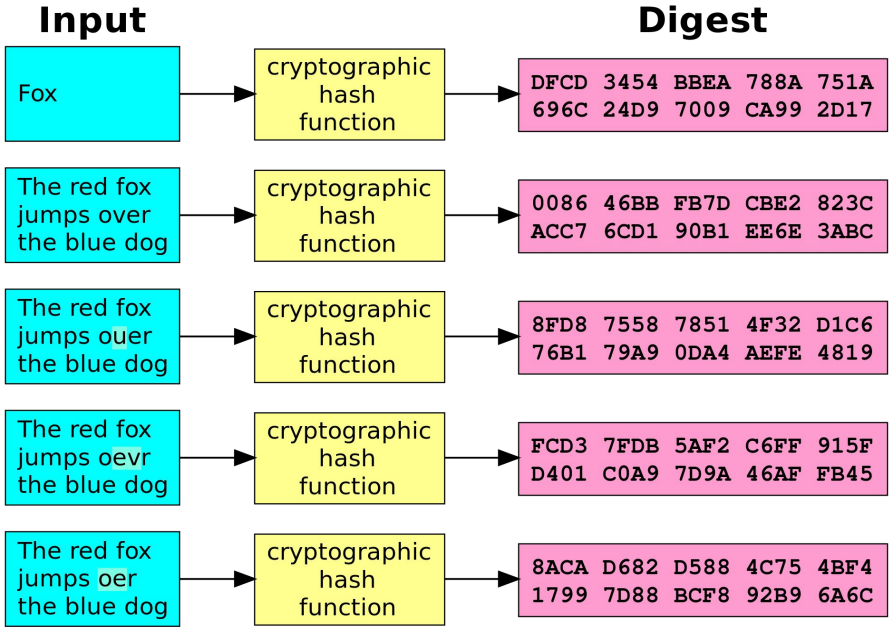
Recitation 5

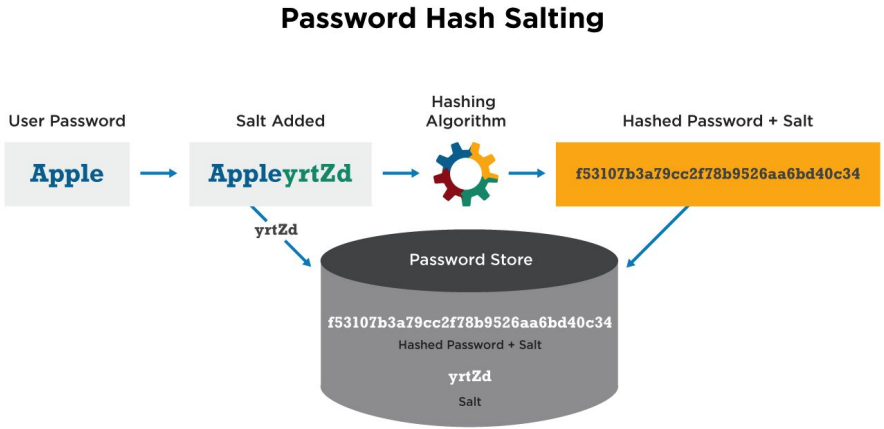
Internet Technology (Section 01)

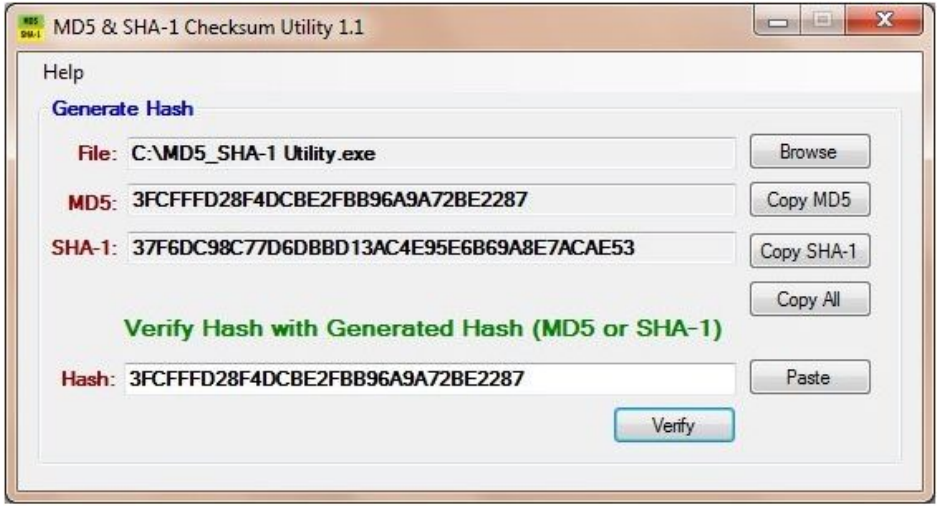
Cryptographic Hash Functions

- Map an arbitrary length binary string to a fixed binary string of *n* bits.
  - But provide some guarantees such as pre-image resistance and collision resistance



Hash Function Use Cases



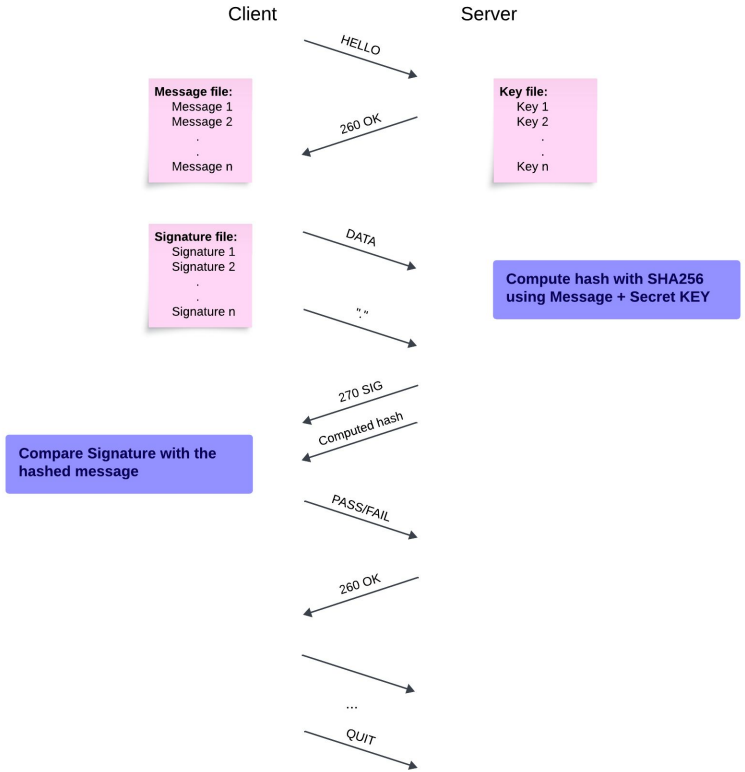


SHA-256

- Part of the Secure Hash Algorithm-2 family of hash functions
  - Always produces a 256-bit digest

```
import hashlib
>>> m = hashlib.sha256()
>>> m.update(b"Nobody inspects")
>>> m.update(b" the spammish repetition")
>>> m.digest()
b'\x03\x1e\xdd}Ae\x15\x93\xc5\xfe\\\x00o\xa5u
+7\xfd\xdf\xf7\xbcN\x84:\xa6\xaf\x0c\x95\x0fK\x
94\x06'
>>> m.hexdigest()
'031edd7d41651593c5fe5c006fa5752b37fddff7
bc4e843aa6af0c950f4b9406'
```

Project 2


Client
Server
HELLO
260 OK
Message file:
Message 1
Message 2
.
.
Message n
Key file:
Key 1
Key 2
.
.
Key n
Signature file:
Signature 1
Signature 2
.
.
Signature n
DATA
"."
Compute hash with SHA256 using Message + Secret KEY
270 SIG
Computed hash
Compare Signature with the hashed message
PASS/FAIL
260 OK
...
QUIT

Project 2 - Hashing

```
import hashlib

message = "Picture a network as a library, so switches and routers are the diligent librarians, sort
ing and directing books (data packets) to the right shelves (devices). And just like a well-organize
d library, a well-managed network is a haven for students."
key = "0e32e7adfb5e1b3bc416b1846ccfe934"

sha256_hash = hashlib.sha256()
sha256_hash.update(message.encode('ascii'))
sha256_hash.update(key.encode('ascii'))

print("SHA256 Hash (ASCII):", sha256_hash.hexdigest())
```

Toy Example: A Remote Text Editor

- Need to support some ASCII commands
  - "**OPEN**", "**CLOSE**", "**INSERT**",
  - On close we send client contents of the edited file
- Specify how to differentiate between commands, data, and end-of-line. Need a format
  - "**:<COMMAND> <DATA>\n**"
- Examples
  - "**:OPEN file.txt\n**"
  - "**:INSERT This is a line of text\n**"
  - "**:CLOSE\n**"

TCP Remote Text Editor (Server)

```
import socket
HOST, PORT="127.0.0.1", 1234
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1)
sock.bind((HOST,PORT))
sock.listen()
buffer = ""
alive = False
CMD_OPEN = ":OPEN "
CMD_INSERT = ":INSERT "
CMD_CLOSE = ":CLOSE "
file = None
while True:
    #try to accept connection if ones doesn't exist
    if not alive:
        (conn, address) = sock.accept()
        alive = True
    else:
        #read data byte by byte
        data = conn.recv(1)
        if not data:
            alive = False
            continue
        data = chr(data[0])
```

```
        #read data until newline character
        if data != '\n':
            buffer+=data
            Continue
        #state machine
        if(buffer[:len(CMD_OPEN)] == CMD_OPEN):
            filename = buffer[len(CMD_OPEN)+1:]
            file = open(filename, "a+")
            buffer = ""
        if(buffer[:len(CMD_INSERT)] == CMD_INSERT):
            text = buffer[len(CMD_INSERT)+1:]
            file.write(text + "\n")
            buffer = ""
        if(buffer[:len(CMD_CLOSE)] == CMD_CLOSE):
            file.seek(0)
            contents = file.read()
            file.close()
            buffer = ""
            conn.sendall(contents.encode())
```

TCP Remote Text Editor (Client)

```
import socket
HOST="127.0.0.1"
PORT=1234
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST,PORT))
sock.sendall(b":OPEN test file.txt\n")
sock.sendall(b":INSERT A line of text\n")
sock.sendall(b":INSERT Another line of text\n")
sock.sendall(b":CLOSE\n")
data = sock.recv(1024)
print(data.decode())
sock.close()
```

- We don't actually need this client!
  - Our protocol is ASCII based so Telnet will suffice (need to use "`\n`" instead of "`\r\n`" for end-of-line)
- Example:

```
$ telnet 127.0.0.1 1234
  > :OPEN test.txt
  > :INSERT some text
  > :CLOSE
  > some text
```