Secure client-server communication

B Bharadwaj 21CSB0B07

Palacharla Sri Satya Naveen 21CSB0B69

Introduction:

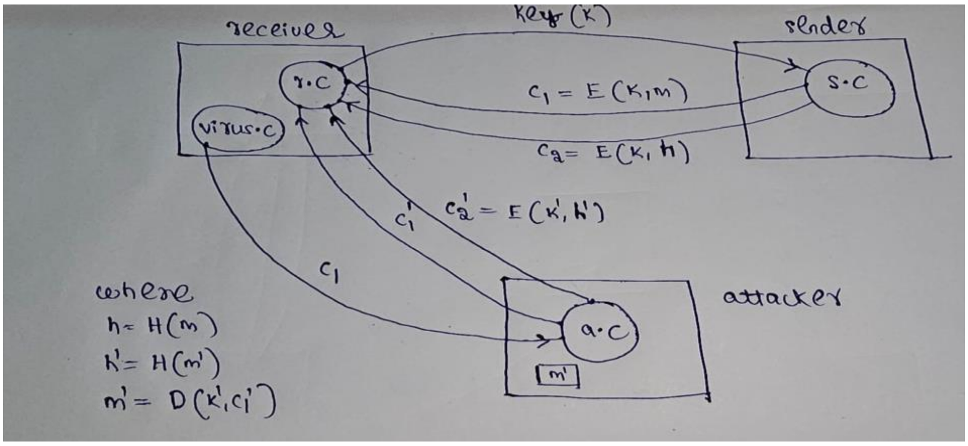
In our project, we explore a robust cryptographic communication system designed to secure message transmissions against sophisticated cyber-attacks. Initially, the communication begins with the receiver sending a secure key to the sender. Utilizing this key, the sender encrypts the message using the Advanced Encryption Standard (AES) algorithm and dispatches the ciphertext to the receiver. However, vulnerabilities arise when an attacker injects a virus into the receiver's system, enabling them to intercept and slightly modify the ciphertext before redirecting it back to the receiver. Consequently, the receiver faces the dilemma of identifying the legitimate message, as they receive two ciphertexts – one from the original sender and another from the attacker.

To counteract this security breach, our system integrates the use of encrypted hash. The sender generates an encrypted hash by encrypting a hash of the message, created using the SHA-256 algorithm, with their private key. As the attacker lacks access to the original sender's private key, they are forced to use an alternative key to forge a hash. Upon receipt, the receiver verifies both hash values against the known key of the sender. This verification process enables the receiver to accurately determine the authenticity of the message, effectively identifying the intended sender and mitigating the risk of manipulated messages. This approach not only enhances the integrity of the communication but also fortifies the overall security framework against potential cryptographic attacks.

Objectives:

- Implement a cryptographic system using AES and SHA-256 to ensure message authenticity and prevent unauthorized modifications.
- Develop robust verification mechanisms to distinguish between legitimate and tampered messages, enhancing security against cyber-attacks.

Implementation and Results analysis:



1)Receiver sending symmetric key to sender securely.

```
naveen@naveen:~/crypto_project$ g++ c1.cpp -o c1
naveen@naveen:~/crypto_project$ ./c1
key=0b2a4c78639e87ea21c2b4298aaa4afc7c4eea12f6f3446cab3ce5902e2775e6

sent successfully
```

2)Sender received symmetric key from receiver.

```
bharadwaj@Twentyone: ~/Documents/CRYPTOGRAPHY
bharadwaj@Twentyone:~/Documents/CRYPTOGRAPHY$ make
g++ keyReceiver.cpp -o kR
g++ sendMsg.cpp -o sm
bharadwaj@Twentyone:~/Documents/CRYPTOGRAPHY$ ./kR
Key is 0b2a4c78639e87ea21c2b4298aaa4afc7c4eea12f6f3446cab3ce5902e2775e6
```

3)Sender sending the ciphertext by encrypting with symmetric key.

```
bharadwaj@Twentyone:~/Documents/CRYPTOGRAPHY$ ./sm
plaintext = The sun set behind the mountains

hex string is too long, ignoring excess
ciphertext = �pSz�U\z�米�Wut�����q       ��u
                                            ���
                                               ˌ`��R2�k�x���s
48
```

4)Receiver received the ciphertext and got plaintext by decrypting ciphertext.

```
naveen@naveen:~/crypto_project$ g++ c2.cpp -o c2
naveen@naveen:~/crypto_project$ ./c2
cipher_text1=�pSz�U\z帯�Wut�����q       ��u
                                            ���
                                               ˌ`��R2�k�x���s

hex string is too long, ignoring excess
message=The sun set behind the mountains
```

5)A virus program runs in receiver system and captures the ciphertext and sends to attacker.

```
naveen@naveen:~/crypto_project$ gcc a.c -o a -lpcap
naveen@naveen:~/crypto_project$ sudo ./a
Device: wlp1s0
Number of packets: 4
Filter expression: port 9501

Packet number 1:
       From: 10.42.0.1
         To: 10.42.0.79
       Protocol = 6
   Src port: 38712
   Dst port: 9501

Packet number 2:
       From: 10.42.0.79
         To: 10.42.0.1
       Protocol = 6
   Src port: 9501
   Dst port: 38712

Packet number 3:
       From: 10.42.0.1
         To: 10.42.0.79
       Protocol = 6
   Src port: 38712
   Dst port: 9501

Packet number 4:
       From: 10.42.0.79
         To: 10.42.0.1
       Protocol = 6
   Src port: 9501
   Dst port: 38712
   Payload (1448 bytes):
00000   e6 70 53 7a c6 55 5c 7a  e5 b8 9f 81 57 75 74 81   .pSz.U\z....Wut.
00016   8f 1d a4 c4 f2 71 09 ab  e5 75 0b 87 ec ee bf 0b   .....q...u......
00032   02 db af 60 c0 cf 52 32  d4 6b f4 78 85 b0 dd 73   ...`..R2.k.x...s
00048   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00064   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

6)Attacker after receiving the ciphertext modifies it slightly and sends back to receiver.

```
kannasuma@kannasuma-VirtualBox: ~/Music/BHARADWAJ ...
kannasuma@kannasuma-VirtualBox:~/Music/BHARADWAJ PROJECT$ g++ mitm.cpp -o m
kannasuma@kannasuma-VirtualBox:~/Music/BHARADWAJ PROJECT$ ./m
Message is ◆pSz◆U\z帀◆Wut◆◆◆◆◆q ◆◆u
                                  ◆◆◆
                                     ʝ`◆◆R2◆k◆x◆◆◆s
Sending message ...
◆pSz◆U\z帀◆Wut◆◆◆◆◆q    ◆◆u
                           ◆◆◆
                              ʝ`◆◆R2◆k◆x◆◆◆s
sent successfully
```

7)Now receiver gets another ciphertext and decrypts it.

```
cipher_text2=◆pSz◆U\z帀◆Wut◆◆◆◆◆q       ◆◆u
                                           ◆◆◆
                                              ʝ`◆◆R2◆k◆x◆◆◆s
hex string is too long, ignoring excess
message=◆%◆,◆*Y◆Al◆u◆nd the mountains
```

8)Now receiver has two messages and he can't distinguish between sender and attacker.

```
naveen@naveen:~/crypto_project$ g++ c2.cpp -o c2
naveen@naveen:~/crypto_project$ ./c2
cipher_text1=◆pSz◆U\z帀◆Wut◆◆◆◆◆q       ◆◆u
                                           ◆◆◆
                                              ʝ`◆◆R2◆k◆x◆◆◆s
hex string is too long, ignoring excess
message=The sun set behind the mountains
cipher_text2=◆pSz◆U\z帀◆Wut◆◆◆◆◆q       ◆◆u
                                           ◆◆◆
                                              ʝ`◆◆R2◆k◆x◆◆◆s
hex string is too long, ignoring excess
message=◆%◆,◆*Y◆Al◆u◆nd the mountains
```

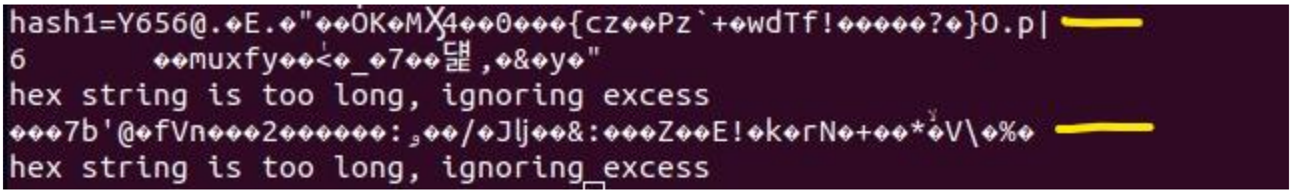
9)Now in order to prevent this attack we use the idea of encrypted hash using sha256 and symmetric key used for encryption. Since attacker don't know the key he produces the encrypted hash by encrypting hash with his own key and sends to the receiver.

```
hex string is too long, ignoring excess
◆◆◆7b'@◆fVn◆◆◆2◆◆◆◆◆◆:¸◆◆/◆Jǉ◆◆&:◆◆◆Z◆◆E!◆k◆rN◆+◆◆*◆V\◆%◆
sent successfully
kannasuma@kannasuma-VirtualBox:~/Music/BHARADWAJ PROJECT$
```

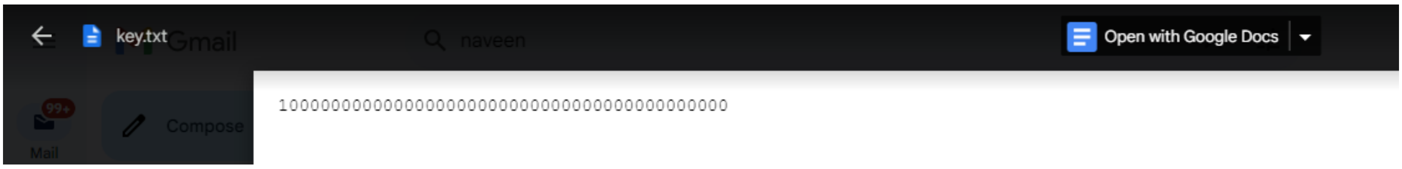
10)Now sender will send the encrypted hash to the receiver by encrypting hash with symmetric key.

```
hex string is too long, ignoring excess
hash = Y656@.◆E.◆"◆◆ÕK◆MX4◆◆0◆◆◆{cz◆◆Pz`+◆wdTf!◆◆◆◆◆?◆}O.p|
6       ◆◆muxfy◆◆<◆_◆7◆◆뎊,◆&◆y◆"
sent successfully
bharadwaj@Twentyone:~/Documents/CRYPTOGRAPHY$
```

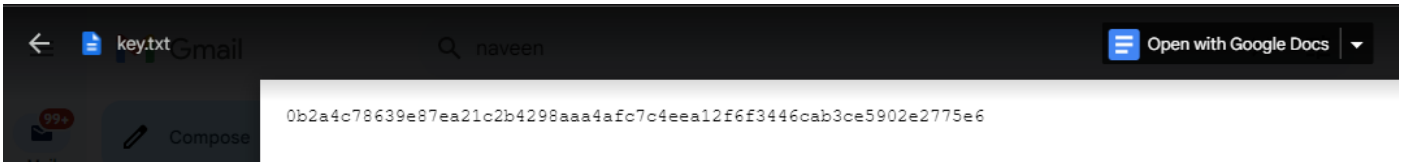
11)Receiver receives both the encrypted hash values. Now receiver will decrypt both encrypted hash values using the symmetric key.



12)Since the attacker's key is different from the symmetric key receiver finds the hash computed from plaintext sent by attacker is different from the hash found by decrypting the encrypted hash sent by attacker.

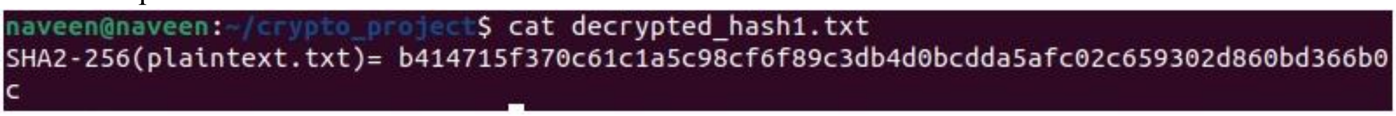


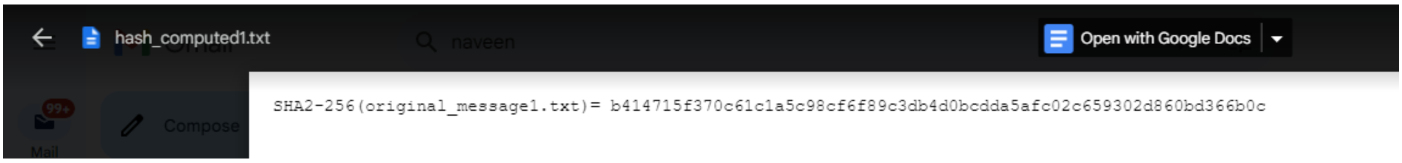
Attacker key



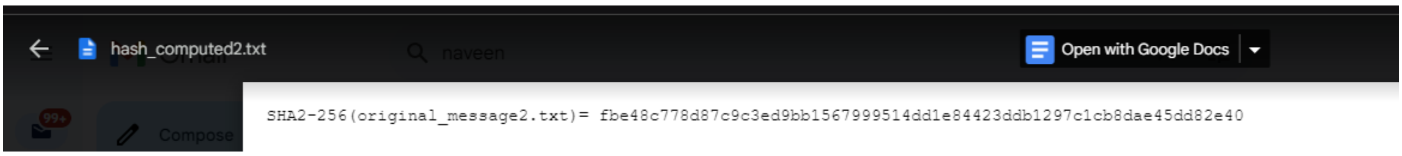
Symmetric key

13) Decrypted hash value1 of encrypted hash1 and Computed hash value from decrypted message of the received cipher text1.





14) Decrypted hash value2 of encrypted hash2 and Computed hash value from decrypted message of the received cipher text2.



Conclusion:

Since receiver finds difference in hash values from one of the sender so he concludes that this corresponding sender is attacker.

In conclusion, our project has successfully addressed the vulnerabilities in cryptographic communication systems by integrating AES encryption and SHA-256 hashed signatures. By employing these techniques, we have enhanced message authenticity and integrity while mitigating the risk of manipulation by cyber attackers. The combination of encrypted hash and key verification ensures that only legitimate messages from the intended sender are accepted, bolstering overall security against potential cryptographic attacks. This comprehensive approach underscores our commitment to developing robust solutions for secure communication in the face of evolving cyber threats.

Learning Outcomes:

- Gained insights into how Advanced Encryption Standard and Secure Hash Algorithm 256 work to ensure secure and reliable message encryption and hashing.
- Learned to implement encrypted hash to verify message authenticity and integrity, enhancing security protocols.
- Developed a deeper understanding of the role of cryptography in protecting against cyber-attacks, particularly in communication systems.

Source codes:

Code at attacker side:

```
int main()
{
    char dev = "wlp1s0";              / capture device name */
    char errbuf[PCAP_ERRBUF_SIZE];        /* error buffer */
    pcap_t handle;                    / packet capture handle */
    char filter_exp[] = "port 9501";          /* filter expression [3] */
    struct bpf_program fp;            /* compiled filter program (expression) */
    bpf_u_int32 mask;                 /* subnet mask */
    bpf_u_int32 net;                  /* ip */
    int num_packets = 4;                  /* number of packets to capture */
    pcap_if_t *devlist;
}
int flag=1;
    if (size_payload > 0)
    {
        flag=1;
        for(int i=0;i<10;i++)
        {
            if(payload[i]=='\0')
            {
                flag=0;
                break;
            }
        }
        if(flag==1)
        {
            printf("   Payload (%d bytes):\n", size_payload);
            for(int i=0;i<100;i++)
                buf[i]=payload[i];
            print_payload(payload, size_payload);
        }
```

Code at sender side:

```
//key receiveing
char key[1000];
int size=0;
size=recv(nsfd,key,sizeof(key),0);
key[size]='\0';
cout<<"Key is "<<key<<endl;
```

```
system("openssl enc -aes-128-ecb -e -in plaintext.txt -out ciphertext.txt -K $(cat key.txt)");//plaintext to ciphertext
system("openssl dgst -sha256 -hex plaintext.txt > hash_value.txt");//computing hash
system("openssl enc -aes-128-ecb -e -in hash_value.txt -out cipherhash.txt -K $(cat key.txt)");//encypting hash with same key used for msg
```

Code at receiver side:

```
//key generation code
system("openssl rand -hex 32 > key.txt");
char key[10000];
memset(key,0,sizeof(key));
int fd2=open("key.txt",O_RDONLY);
read(fd2,key,sizeof(key));
```

```
//decrypting received ciphertexts
system("openssl enc -aes-128-ecb -d -nopad -in cipher_text2.txt -out decrypted2.txt -K $(cat key.txt)");
system("openssl enc -aes-128-ecb -d -nopad -in cipher_text1.txt -out decrypted1.txt -K $(cat key.txt)");
//decrypting encrypted hash
system("openssl enc -aes-128-ecb -d -nopad -in cipher_hash2.txt -out decrypted_hash2.txt -K $(cat key.txt)");
system("openssl enc -aes-128-ecb -d -nopad -in cipher_hash1.txt -out decrypted_hash1.txt -K $(cat key.txt)");
//computing hash value from message
system("openssl dgst -sha256 -hex original_message2.txt > hash_computed2.txt");
system("openssl dgst -sha256 -hex original_message1.txt > hash_computed1.txt");
```

References:

1)Used inbuilt commands of OPENSSL.

2)Used some code of tcpdump for capturing packets.