Network Security Assignment Report

106119029,106119064,106119099,106119102

Contents

HMAC

Problem Statement

Implement HMAC and verify message integrity,confidentiality and non repudiation. (Recommeneded to use your own unique hashing algorithm)

Solution

Approach

We must verify Non Repudiation, Confidentiality and Message Integrity. **HMAC** can be used to verify the **Message Integrity**. We will have to use other things in order to get **Non Repudiation** and **Confidentiality**. We are using Public Key Cryptography(in order to get Non Repudiation) and Private Key Cryptography(in order to get Confidentiality). Namely we're using **RSA** and **AES**.

Steps we followed are as follow:

1. Let the message be `MESSAGE`.
2. Both the sender and receiver have private key and public key of their own. Let the public key `PUB(sender)` and private key be `PRIV(sender)`. Similary let the public key of receiver be `PUB(receiver)` and private key of receiver be `PRIV(receiver)`.
3. Let `AESKEY` be key for AES. `AESKEY` is known only to the sender initially.
4. **Sender** encrypts the `AESKEY` with `PUB(receiver)`. Let the result be `enc1`
5. **Sender** will then encrypt `enc1` with `PRIV(sender)`. Let the result be `ENC_AES_KEY`.

```
let priv_key = rsa::encrypt_private(&rsa::encrypt_public(AES_KEY));
```

6. **Sender** will now encrypt `MESSAGE` with `AESKEY`. Let the result be `ENC_MSG`.
7. **Sender** will use hmac to generate the signature for `ENC_MSG`.
8. In our case, we're using two hashing algorithm for hmac. We're using a well known and popular cryptographic hashing algorithm `Blake3` and another one made by us which we're naming `FibMulCombineHash`.

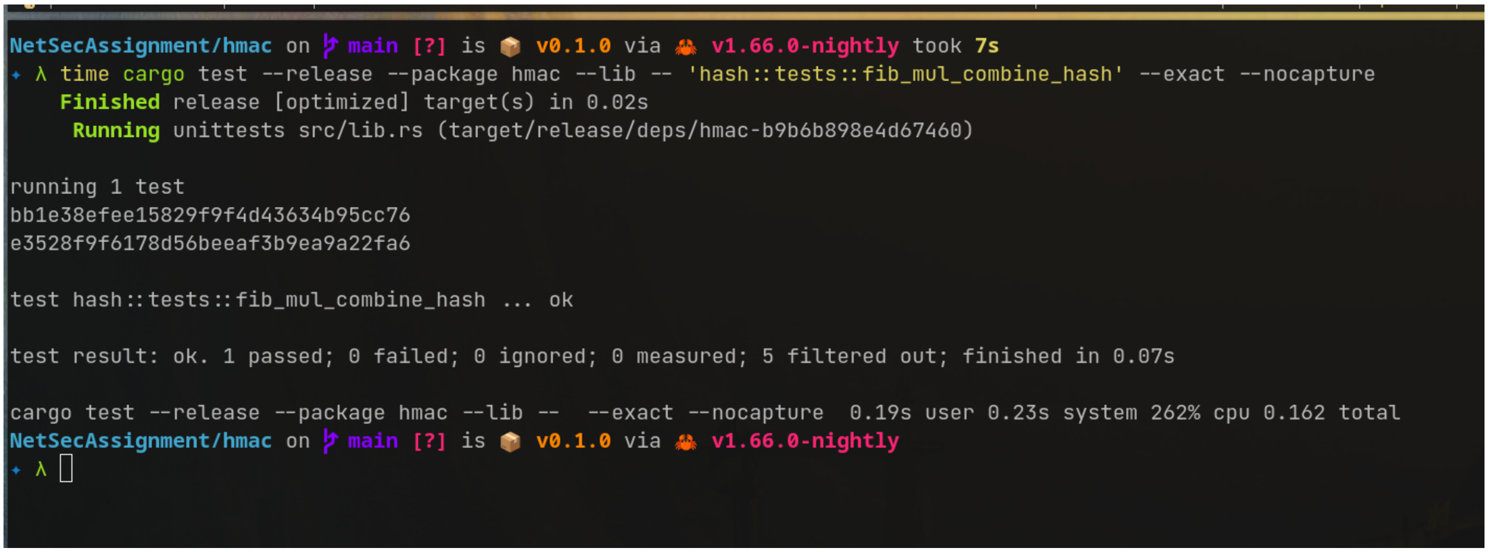
FibMulCombineHash description

- The code is available in `hmac/src/hash.rs`. There are some tests for it as well.

`FibMulCombineHash` is a cryptographic hashing algorithm which outputs 128 bit digest. The inspiration for this algorithm was taken from the book The Art of Computer Programming by Donald Knuth, Volume 3,Section 6.4, page 518. The algorithm is extremely fast, because it's just a multiplication followed by a shift, in order to bring the output to some [0, 2^k) domain. We don't have the shift state as we want the domain to be full [0,2^128). The hash function is known to produce a very uniform distribution of hash values, hence minimizing collisions.

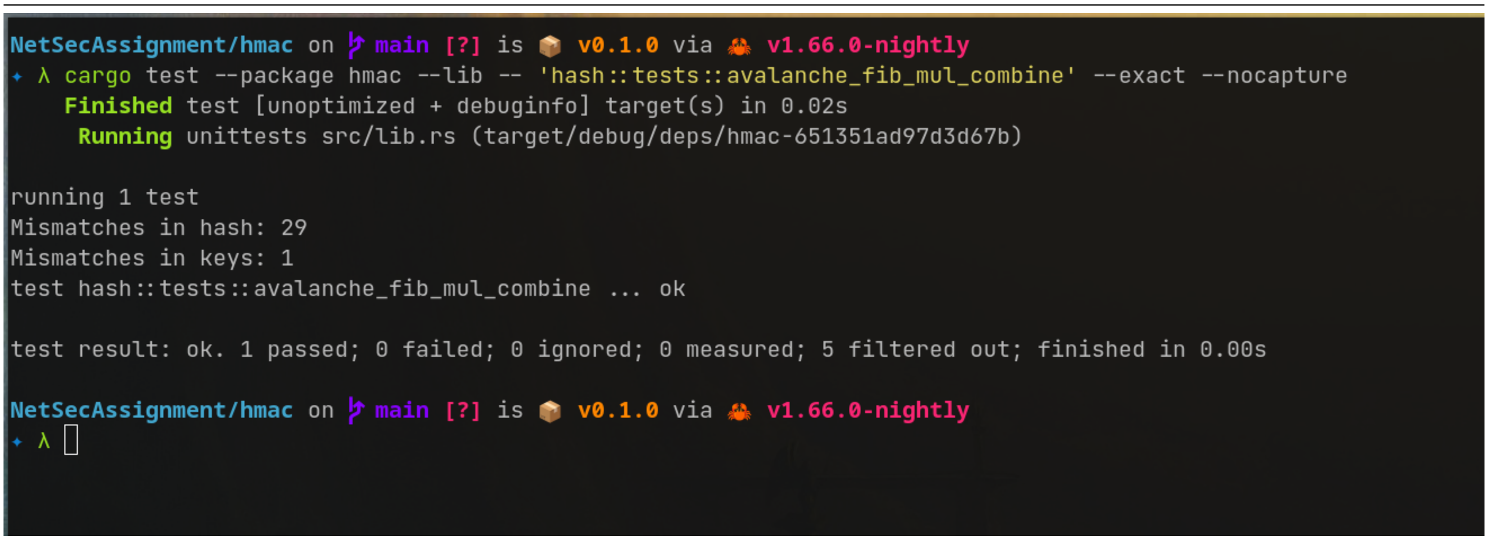
We hash each input byte with this and combine all of them parallely, which makes a very good usage of CPU cores. **In order to hash a 2 Mega Byte String, our CPU usage was well over 200% for this algorithm**. The hash combining strategy is also just a bunch of shifts and additions which will be very fast. The hash function has `Avalance Effect` as well, whcih makes it a very hash function.

More than 200% CPU usage for 2MB string



Cod

Avalanche Effect



9. We will send **HMAC** value calculated using both of these hash functions to the receiver.
10. Sender will then generate a json file called `sender.json` which follows the following struct.

```
pub struct SenderStruct {
    // stores ENC_AES_KEY
    pub rsa_enc_aes_key: Vec<u8>,
    // stores  ENC_MSG
    pub aes_encrypted_message: Vec<u8>,
```

```rust
    //stores Hmac of ENC_MSG with Blake3
    pub hmac_blake3: Vec<u8>,
    // Stores Hmac of ENC_MSG with FibMulCombineHash
    pub hmac_custom_hash: Vec<u8>,
}
```

11. **Receiver** will read `sender.json` and get the fields from it.
12. **Receiver** will then verify the `HMAC` for both the hash functions. This proves **Message Integrity**.
13. **Receiver** will then go on and decrypt the `ENC_AES_KEY` using `PRIV(receiver)` and `PUB(sender)`. It will be `rust let aes_priv_key = rsa::decrypt_private(&rsa::decrypt_public(&sender_params.rsa_enc_aes_key));`
14. This RSA decryption proves **Non Repudiation**,since private key of the sender was involved in the AESKEY encryption.
15. Now the encrypted message `ENC_MSG` can be decrypted using the `AESKEY`. This proves **Confidentiality**

Output

Sender Output

```
NetSecAssignment/hmac on main [?] is v0.1.0 via v1.66.0-nightly
λ cargo run --bin sender NetSecAssignment
    Finished dev [unoptimized + debuginfo] target(s) in 0.02s
     Running `target/debug/sender NetSecAssignment`
We have the message to send from the sender
MESSAGE: NetSecAssignment
512
1) We'll encrypt the AES KEY using RSA(public key of receiver) and then encrypt(RSA again with private key of sender) to get non repudiation
2) We'll encrypt the message using AES to get confidentiality
3) We'll use hmac to show message integrity. HMAC, we're using two variants
     i)  One variant uses Blake3 algorithm for hashing
     ii) 2nd variant uses custom hashing algorithm which outputs 128 bit digest.
--------------------------------------------------------------------
----------------------------OUTPUT----------------------------------
--------------------------------------------------------------------

ENCRYPTED AES_KEY: [54, 192, 105, 34, 212, 37, 246, 29, 52, 96, 112, 215, 25, 0, 216, 197, 156, 218, 55, 39, 16, 235, 149, 83, 195, 160, 224, 72, 114, 67, 102, 207, 195, 229, 22, 37, 124, 49,
 60, 187, 237, 72, 219, 29, 21, 200, 47, 75, 91, 15, 233, 124, 24, 15, 203, 100, 163, 104, 192, 237, 65, 194, 105, 90, 12, 113, 232, 147, 124, 126, 205, 226, 198, 210, 249, 2, 83, 253, 72, 17
6, 40, 19, 38, 6, 192, 204, 68, 135, 186, 60, 15, 168, 152, 245, 237, 136, 92, 1, 242, 59, 211, 182, 167, 1, 96, 11, 247, 255, 39, 6, 253, 255, 248, 197, 198, 215, 237, 240, 68, 216, 125, 252
, 208, 76, 86, 110, 94, 185, 70, 130, 204, 190, 83, 96, 66, 127, 207, 10, 201, 194, 39, 91, 70, 100, 93, 141, 172, 161, 107, 86, 246, 77, 133, 199, 221, 27, 26, 84, 228, 120, 236, 176, 151, 4
4, 174, 136, 4, 107, 130, 44, 6, 0, 90, 31, 63, 41, 252, 40, 144, 28, 60, 253, 14, 175, 247, 227, 144, 95, 72, 203, 20, 92, 13, 108, 50, 193, 59, 235, 156, 96, 67, 75, 214, 71, 194, 175, 46,
125, 12, 233, 128, 14, 139, 179, 51, 226, 247, 212, 151, 117, 172, 237, 17, 245, 254, 227, 128, 198, 32, 96, 241, 23, 166, 142, 34, 100, 144, 219, 48, 202, 246, 2, 51, 215, 180, 99, 118, 227,
 199, 181, 186, 250, 43, 174, 10, 166, 105, 172, 24, 131, 240, 200, 69, 94, 147, 9, 95, 205, 184, 242, 204, 184, 192, 116, 20, 181, 247, 80, 121, 137, 189, 185, 163, 137, 254, 40, 30, 240, 20
8, 108, 209, 74, 79, 40, 83, 3, 221, 201, 49, 43, 25, 111, 160, 169, 113, 193, 156, 178, 200, 124, 244, 80, 146, 206, 15, 151, 49, 3, 19, 93, 108, 109, 167, 121, 32, 206, 184, 103, 47, 244, 1
84, 26, 190, 98, 191, 111, 57, 22, 208, 115, 167, 104, 161, 66, 239, 252, 36, 61, 225, 210, 153, 89, 61, 194, 148, 88, 135, 32, 27, 55, 82, 19, 28, 185, 188, 163, 165, 168, 181, 127, 86, 143,
 132, 177, 137, 244, 146, 11, 105, 156, 24, 150, 65, 89, 83, 225, 135, 210, 142, 198, 48, 238, 239, 236, 137, 171, 205, 175, 172, 238, 15, 194, 58, 18, 16, 102, 21, 119, 60, 224, 215, 205, 38
, 151, 189, 126, 190, 210, 31, 24, 123, 229, 125, 199, 70, 20, 177, 239, 131, 235, 131, 156, 179, 46, 187, 118, 185, 114, 164, 5, 172, 91, 177, 177, 114, 168, 41, 123, 152, 42, 102, 59, 100,
69, 47, 100, 220, 120, 179, 181, 189, 14, 106, 165, 65, 53, 28, 163, 94, 144, 227, 227, 163, 29, 255, 124, 168, 17, 214, 115, 181, 16, 49, 235, 69, 5, 91, 189, 223, 158, 217, 96, 115, 137, 33
, 255, 69, 236, 207, 20, 162, 240, 235, 39, 161, 119, 42, 43, 221, 125, 35, 203]

ENCRYPTED MESSAGE WITH AES: [234, 34, 251, 11, 71, 139, 185, 78, 110, 182, 251, 255, 154, 110, 64, 183, 184, 90, 235, 217, 230, 236, 157, 200, 20, 87, 101, 255, 63, 235, 178, 40]

HMAC on ENCRYPTED MESSAGE(Blake3): [91, 233, 11, 224, 219, 203, 140, 22, 20, 222, 60, 54, 75, 114, 50, 88, 34, 36, 107, 157, 94, 66, 4, 253, 161, 187, 232, 174, 189, 151, 202, 176]

HMAC on ENCRYPTED MESSAGE(Custom Hash): [242, 240, 237, 52, 210, 106, 252, 70, 201, 225, 117, 84, 200, 77, 186, 162]

NetSecAssignment/hmac on main [?] is v0.1.0 via v1.66.0-nightly
λ
```

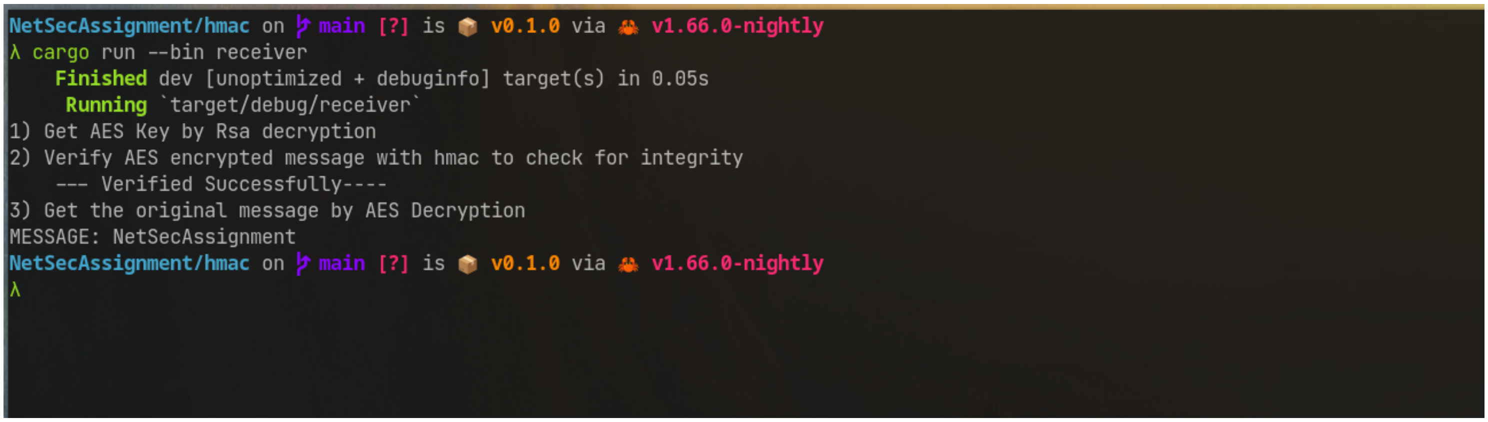
Sender Json

```
NetSecAssignment/hmac on main [?] is v0.1.0 via v1.66.0-nightly
λ bat sender.json
       File: sender.json
```


```
{"rsa_enc_aes_key":[54,192,105,34,212,37,246,29,52,96,112,215,25,0,216,197,156,218,55,39,16,235,149,83,195,160,224,72,114,67,102,207,195,229,22,37,124,49,60,187,237,72,219,29,21,200,
47,75,91,15,233,124,24,15,203,100,163,104,192,237,65,194,105,90,12,113,232,147,124,126,205,226,198,210,249,2,83,253,72,176,40,19,38,6,192,204,68,135,186,60,15,168,152,245,237,136,92,
1,242,59,211,182,167,1,96,11,247,255,39,6,253,255,248,197,198,215,237,240,68,216,125,252,208,76,86,110,94,185,70,130,204,190,83,96,66,127,207,10,201,194,39,91,70,100,93,141,172,161,1
07,86,246,77,133,199,221,27,26,84,228,120,236,176,151,44,174,136,4,107,130,44,6,0,90,31,63,41,252,40,144,28,60,253,14,175,247,227,144,95,72,203,20,92,13,108,50,193,59,235,156,96,67,7
5,214,71,194,175,46,125,12,233,128,14,139,179,51,226,247,212,151,117,172,237,17,245,254,227,128,198,32,96,241,23,166,142,34,100,144,219,48,202,246,2,51,215,180,99,118,227,199,181,186
,250,43,174,10,166,105,172,24,131,240,200,69,94,147,9,95,205,184,242,204,184,192,116,20,181,247,80,121,137,189,185,163,137,254,40,30,240,208,108,209,74,79,40,83,3,221,201,49,43,25,11
1,160,169,113,193,156,178,200,124,244,80,146,206,15,151,49,3,19,93,108,109,167,121,32,206,184,103,47,244,184,26,190,98,191,111,57,22,208,115,167,104,161,66,239,252,36,61,225,210,153,
89,61,194,148,88,135,32,27,55,82,19,28,185,188,163,165,168,181,127,86,143,132,177,137,244,146,11,105,156,24,150,65,89,83,225,135,210,142,198,48,238,239,236,137,171,205,175,172,238,15
,194,58,18,16,102,21,119,60,224,215,205,38,151,189,126,190,210,31,24,123,229,125,199,70,20,177,239,131,235,131,156,179,46,187,118,185,114,164,5,172,91,177,177,114,168,41,123,152,42,1
02,59,100,69,47,100,220,120,179,181,189,14,106,165,65,53,28,163,94,144,227,227,163,29,255,124,168,17,214,115,181,16,49,235,69,5,91,189,223,158,217,96,115,137,33,255,69,236,207,20,162
,240,235,39,161,119,42,43,221,125,35,203],"aes_encrypted_message":[234,34,251,11,71,139,185,78,110,182,251,255,154,110,64,183,184,90,235,217,230,236,157,200,20,87,101,255,63,235,178,
40],"hmac_blake3":[91,233,11,224,219,203,140,22,20,222,60,54,75,114,50,88,34,36,107,157,94,66,4,253,161,187,232,174,189,151,202,176],"hmac_custom_hash":[242,240,237,52,210,106,252,70
,201,225,117,84,200,77,186,162]}
```


```
NetSecAssignment/hmac on main [?] is v0.1.0 via v1.66.0-nightly
λ
```

Receiver Output



```
NetSecAssignment/hmac on  main [?] is 📦 v0.1.0 via 🦀 v1.66.0-nightly
λ cargo run --bin receiver
    Finished dev [unoptimized + debuginfo] target(s) in 0.05s
     Running `target/debug/receiver`
1) Get AES Key by Rsa decryption
2) Verify AES encrypted message with hmac to check for integrity
    --- Verified Successfully----
3) Get the original message by AES Decryption
MESSAGE: NetSecAssignment
NetSecAssignment/hmac on  main [?] is 📦 v0.1.0 via 🦀 v1.66.0-nightly
λ
```

Buffer Overflow

Illegal Packet

DOS

Shrew Attack