Ewan Jones SID:1094396

Secret Key Cryptography

Stream Ciphers

1c.

Eve would not be able to recover a complete or partial key based off one encrypted message between Alice and Bob. It would also be impossible to get any meaningful insight into the message or its encryption key as any form of frequency analysis is impossible with such a limited dataset. In other words, without other encrypted messages between Bob and Alice within the same day it would be impossible for Eve to decipher or determine the key used by Alice and Bob.

1d.

If Eve has two encrypted messages between Bob and Alice, she could recover a partial key. Eve could do this by observing the first 22 characters of the message as they would be the exact same. The only part of both messages that would be different is the gossip itself while the protocol implemented by Alices app will never change until the private keys roll over. This means Alice could know who sent and received the message but not much else.

Block Ciphers

2a.

The code for this question is provided as aes-encrypt.py and aes-decrypt.py. Running either program with no command line arguments will provide a usage screen explaining the use of each argument while providing useful examples.

2b.

Message1 (alicexoxox:bobko0oo0l:ewan):

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
B3 95 C4 7C C4 00 C9 1C 11 2C 5A 00 72 3C C9 C5
FE 75 84 D2 20 D5 E1 B3 FF C2 54 7D 5F 02 DA 9F
```

Message 2(alicexoxox:bobko0oo0l:jones):

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
B3 95 C4 7C C4 00 C9 1C 11 2C 5A 00 72 3C C9 C5
80 BA 8E 00 E6 69 C9 6B 2B 4B 05 D8 78 E6 53 0A
```

After looking at the first 20 bytes of the hex dumps for the messages alicexoxox:bobko0oo0l:ewan and alicexoxox:bobko0oo0l: jones I can definitely see some similarities. The first 16 bytes or first block of each message contain the exact same ciphertext. The reason this is observed is due to the encryption mode used. ECB does not feature an initialization

vector like other block modes such as CBC this means given the same input the exact same ciphertext will be generated.

2c.

If Alice used a counter IV instead of a random one with my implementation of CBC a similar problem like what was observed with ECB can occur. This implementation with a counter IV, however, does provide confidentiality as the encryption key is private and used to protect the information hidden by the ciphertext. Only authorized parties would ideally know the encryption key which keeps the data confidential.

2d.

2c does not provide integrity. An attacker is still able to change the ciphertext over a wire or on disk. If this does occur the original private key may fail to correctly decrypt the ciphertext.

2e.

Set mode equal to GCM (mode=gcm) to use this feature in aes-encrypt or aes-decrypt to use this feature. GCM has the ability to verify if the ciphertext has been changed via a unique tag. This tag is generated during encryption and is then used after decryption to check if it has been altered in any way.

Public Key Cryptography

RSA

3b.

```
● ➜  a1 git:(main) ✗ echo "Ewan" | openssl pkeyutl -encrypt -pubin -inkey F24.crt -certin -out message
● ➜  a1 git:(main) ✗ openssl pkeyutl -decrypt -inkey F24.key -in message
  Ewan
○ ➜  a1 git:(main) ✗
```

The public key F24.crt is used to asymmetrically encrypt my name while the private key F24.key can be used to decrypt it.

Integrity and Authentication

4b.

```
● ➜  a1 git:(main) ✗ openssl dgst -sha256 -binary -out A1.hash A1.pdf
● ➜  a1 git:(main) ✗ openssl pkeyutl -sign -in A1.hash -inkey F24.key -out A1.sig
● ➜  a1 git:(main) ✗ openssl pkeyutl -verify -in A1.hash -sigfile A1.sig -certin -inkey F24.crt
  Signature Verified Successfully
○ ➜  a1 git:(main) ✗
```

The first command out of the three generates a unique sha256 hash for the file A1.pdf. This hash is then used in conjunction with the private key F24.key to generate a signature for the PDF file. Finally, the public key is used to verify the file along with its signature and hash.