**Rationale**

RSA:

RSA with OAEP padding was used to encrypt MAC and AES keys so that Alice and Bob can securely agree on symmetric keys. RSA with OAEP can resist padding oracle attacks and have no known major attacks.

SHA256withRSA was used to sign the initial transmission that establishes our MAC key. Even though the scheme is vulnerable against padding oracle attacks, we are assuming that we have been actively monitoring requests for things that are signed by our RSA key before it reaches our code.

Our RSA keys are of length 2048 bits. This length is recommended by NIST and is predicted to be safe until 2030.

Symmetric:

AES-128 was used for symmetrically encrypting our message because there are no currently known practical attacks. The key size that we used is 128 bits which is the minimum size recommended by NIST.

HmacSHA256 keyed-hashing algorithm was used to generate MAC. This method allows us to hash messages with a key, which makes the hash function even more reliable. The key size that we used is 256 bits, which according to our research online is secure.

For the configuration that uses both MAC and Encryption, we are using “Encrypt then MAC” method because it is currently the most secure of the options available. We are also able to detect whether MAC fails or not without wasting energy to decrypt.

**Protocol Narration:**

No encryption:

A -> B: Bob, typeOfMsg, session#, tA, m

B: print m

Encryption Only:

A -> B: Bob, typeOfMsg, session#, tA, Enc(K\_AB; K\_PublicB), Enc(m; K\_AB)

B: Dec(Enc(K\_AB; K\_PublicB); K\_PrivateB) => K\_AB

Dec(Enc(m; K\_AB); K\_AB) => print m

Mac-key-agreement Protocol with Digital Signature (for ‘Mac Only’ and ‘Encrypt-then-Mac’):

A: s = B, typeOfMsg, session#, tA, Enc(K\_MAC, K\_PublicB)

t = Sign(s; K\_PrivateA)

A->B: s,t

B: if Ver(s; t; K\_PublicA) then

Dec(Enc(K\_MAC, K\_PublicB); K\_PrivateB) => K\_MAC

else abort

Mac Only:

A: s = Bob, typeOfMsg, session#, tA, m

t = Mac(s; K\_MAC)

A->B: s,t

B: t’ = MAC(s; K\_MAC)

if (t = t’), then print m

else abort

Encrypt-then-Mac:

A: s = Bob, typeOfMsg, session#, tA, Enc(K\_AB; K\_PublicB), Enc(m; K\_AB)

t = Mac(s; K\_MAC)

A -> B: s, t

B: t’ = MAC(s; K\_MAC)

if (t = t’), then

Dec(Enc(K\_AB; K\_PublicB); K\_PrivateB) => K\_AB

Dec(Enc(m; K\_AB); K\_AB) => m

print m

else abort

Bob: The receiver of the message

typeOfMsg: Indicates whether the message is to establish MAC keys or to convey a message

session#: The count associated with the message; each subsequent message increment the count by 1

tA: Timestamp sampled from Alice's local clock

K\_AB: Session key used to symmetrically encrypt message

K\_PublicA: Public RSA key of Alice

K\_PrivateA: Private RSA key of Bob

K\_PublicB: Public RSA key of Alice

K\_PrivateB: Private RSA key of Bob

K\_MAC: MAC key used to ensure integrity of message

m: Message from Alice to Bob

**Specification:**

If MAC is utilized, then Alice will initiate the MAC-key-exchange protocol with Bob without prompting for user input. This is to simplify the process for the user.

We are assuming that the user will not input uncommon character/character sequences that cannot be printed or encoded to Base64.

We made the choice to also include the session# in the transmission between Alice and Bob to better guard against replay attacks.

We have differentiated the possible messages into 2 types of messages: “MacExchange” and “NewMessage”. This was done in order Bob to better process the data.

**External Libraries:**

Bouncy Castle was used to implement RSA with OAEP padding.

**Known Problems:**

Because we are signing our keys using the SHA256RSA scheme that is vulnerable against padding oracle attacks, we are assuming that we are actively monitoring requests for things that are signed by our RSA key before it reaches our code.

In our program, we assume that the Mac-key-exchange between Alice and Bob will not be interrupted by Mallory. This is due to the limitation that Bob cannot tell Alice that the MAC key has been compromised. Thus, we are making the assuption that Alice will successfully share the MAC key with Bob.

If Mallory discrupts the synchronization of the counter between Alice and Bob by deleting a message, Bob will able to detect this interuption, but the counter will stay offset for the remaining messages. We considered this issue to not be problematic because if something similar happens in the real world, it is most likely that Alice and Bob will address the issue or start a new connection.

If Mallory replays the Mac-key-exchange message, Bob will just conclude that the MAC is off, and the conversation will continue. Since Bob would have had successfully shared a MAC key with Alice, we assume that Bob would not care about replays of the Mac-key-exchange.