A3 Secure Channels: Maxwell Rose Stefanos Stoikos; Overview

1. Rationale: Explain the use of cryptography in your system. The document should include protocol narrations to document the cryptographic protocol used in each of your four configurations. The document should also provide a justification for your choices of key lengths and cryptographic algorithms.
   - **No Encryption**: We use no cryptographic protocol
   - **Symmetric Only:** We use AES-256 encryption with a previously shared secret key. Here, we send both a message count and the encrypted message. This is vulnerable to a truncating attack/replay attack because we do not sign it. We chose a 32-byte key because it is long enough where a modern computer cannot use a brute force method in a reasonable amount of time.
   - **Mac Only:** We use a HashMac from the cryptography library to create a tag to verify the integrity of the message. We send over the count, the plain text, and a tag of the plain text. That tag is then verified using the previously shared key to ensure that the message was indeed not changed. We use a 32-byte key because it is not possible to use a brute force attack in a reasonable amount of time.
   - **Symmetric then Mac:** We use AES-256 to encrypt the plain text under the previously shared secret key. Then, we take the message count and the encrypted text and create a tag using a HashMac with the previously shared HMAC key. These are all sent to Bob who can verify the tag using the shared HMAC key and can decrypt the message using the shared AES key. We chose 32-byte keys because they are long enough where a modern computer cannot use a brute force method in a reasonable amount of time.
2. Specification. This part of the assignment is deliberately underspecified. Detail and justify your choices here.
   - We implemented our own pad/unpad functions that append the number of needed bytes to the end of a block to make it the correct block size. The cryptography package pad wasn't working but we did this because it seems to be the standard.
   - When using encryptions such as mac, symmetric, and symmetric_mac the message sent from Alice to Bob contained multiple objects (timestamps, identification, ciphertext, signature, etc.). In order to extract them in one piece on Bob's end, we sent them as one string, separating them by 2 spaces " ". This way Bob, can use the split string function specifying the 2 spaces without disrupting the included info since it is very unlikely for a ciphertext to contain two consecutive empty spaces.
   - Our handshake sends to keys, an AES key, and a mac_key. We did not want to use the same for both and it seemed to be the simplest to just send them at the same time. It then signs everything and sends that as well so Bob can ensure the integrity of the message. The messages are signed with RSA because we needed an asymmetric way to get initial keys across.

- Keys are generated with os.urandom(). This appears to be used in many cryptography schemes. We used 32-byte keys as they are large enough to defend against brute force attacks from modern computers.
- We use AES-256 CBC so we can securely handle large bodies of text. We generate a fresh iv for each message to prevent against attacks that leverage the iv staying constant. We used CBC because it secure and had good support from the cryptography package
- We use RSA to encrypt the handshake because it is a secure way to establish shared keys. RSA is only used in the handshake.

External libraries. If you use any external libraries, list them here along with what your code uses them for.

- We use the python cryptography library for our encryption/decryption and signing
- We use the python base64 library for base64 encoding and decoding
- We use the datetime package to determine when messages were sent and received by Bob; check for delays that could mean an adversary
- We also use sys, socket, and os which were provided in the starter code
- We used the time package to check to deliberately delay the transition of the message from Alice - > Mallory -> Bob, to check if Alice time stamp was sent no longer than 2 minutes after Bob received the message

3. Known problems. Detail any known problems with your rationale, specification, or implementation. You may also include anything else that you see fit.
   - We don't specify an attack taking advantage of this, but we don't encrypt the count in symmetric only so it would be easy for an attacker to change the count and do a reply attack. This is fixed by the tagging in symmetric_mac
   - We don't include a timestamp with each message, so it would be possible for an attacker to hold onto messages and send them later for whatever reason
   - We don't verify that messages are received or that the receiver is indeed Bob
   - We have no way of verifying that the sender was actually Allice, Mallory could set up a connection and share a key and act like Allice