Rationale

Asymmetric Key Generation

We used 2048 bits RSA to generate the private keys separately for Alice and Bob
Then we generate the corresponding public keys.

We are using openssl rsa with OAEP padding because it’s non-malleable, as mentioned in the writeup, so even if the Delov-Yao gets the message later on and just modifies some bits, the rsa decryption would either fail or decrypt into nonsense.

We choose des3 to encrypt the private key before outputting it because
We choose 2048 bits because according to NIST, 2048 bits is secure for encryption schemes through 2030.

Secure channel

I. Protocol narration

```
1. A->M: B, tA, c = Enc(A, kAB; K_B), s = Sign(B, tA, c; k_A)
2. M: if M forwards the exact secure channel message to B
        M->B: B, tA, c, s
3. B: if Verify((B’, tA, c’), s’; K_A) is true:
        If B’ == B && tB - tA < 2 min:
            A’, m = Dec(c’; k_B)
            If A’ == A:
                Get session_key kAB
4: A, B: derive enc and mac key from kAB using hashing
```

II. Use of cryptographic algorithm:

We used openssl to generate a 128-byte-long session key because 1) we are using aes-256 later for the symmetric encryption and MAC, so the enc key and mac key derived needs to be 256-bit long. And since we are deriving these two keys from the session key, it’s better to have the session key be longer than 256 bits. 2) The session key appended after the sender’s name needs to be encrypted by the private key generated by RSA, so our session key cannot be too long for the key size.

We used openssl rsa with OAEP padding because it’s non-malleable. So even if Mallory gets a hold of the ciphertext, she won’t be able to flip bits and forward to Bob without Bob noticing her modification.

We used openssl dgst sha256 to hash the ciphertext c and then pass it into Sign because rsa sign has a restriction on the message length that can be signed (data signed cannot be too long.) .

Key Derivation
Use of cryptographic algorithm:
We used HKDF from cryptography library because it’s a cryptographically secure hash function. We hash the given session key with different salt for enc and mac, and gets a 32 byte long key in return. The 32-byte-long (256-bit-long) keys are then used as enc and mac keys.

Part I No-cryptography

1. Protocol narration

1. A->B: establish session key with above protocol
2. A->M: plaintext message m
3. M: if M forwards m to B, M->B: m
4. B: gets the message

2. Use of cryptographic algorithms: None

Part II Enc-only

1. Protocol narration

   A: x = Enc(m ; ke, iv)
   A → M: iv,x
   M → B: iv,x
   B: m = Dec(x ; ke, iv)

2. Use of cryptographic algorithms:
   We used AES-256-CBC encryption with a random initialization vector (IV) for every message sent. AES-256 is more secure than AES-128, and CBC is used to avoid intra-message repetitions in the ciphertext. The IV is used to avoid inter-message repetitions. The length of the IV is 128 bits because it has to match with the block size.

Part III Mac-only

1. Protocol narration

   A: x = Hmac(m ; km)
   A → M: x,m
   M → B: x,m
   B: Verify x = Hmac(m ; km)

2. Use of cryptographic algorithms:
   We used HMAC with SHA-256 hashing. SHA-256 is a cryptographically secure hash function that takes arbitrary length input and hash it to a fixed size output.

Part IV Enc-then-mac

1. Protocol narration

   A: x = Enc(m ; ke, iv)
   y = Hmac(x ; km)
   A → M: iv, x, y
   M → B: iv, x, y
   B: verify y = Hmac(x ; km)
   m = Dec(x ; ke, iv)

2. Use of cryptographic algorithms:
   We used both AES-256-CBC encryption with a randomized IV and HMAC with SHA-256 hashing with the reasons described above.