

Problem 1

```
• paulh@paulhan ~/w/e/l/l/part1 (master)> ./aes
Plain text: 6bc1bee22e409f96e93d7e117393172aee2d8a571e3ac9c9eb76fac45af8e51
Encrypted data: 601ec313775789a5b7a7f54bbf3d228f443e3ca4d62b59aca84e990cacaf5c5
Cipher text: 601ec313775789a5b7a7f54bbf3d228f443e3ca4d62b59aca84e990cacaf5c5
Decrypted data: 6bc1bee22e409f96e93d7e117393172aee2d8a571e3ac9c9eb76fac45af8e51
○ paulh@paulhan ~/w/e/l/l/part1 (master)> █
```

AES-CTR-256 does not provide integrity authentication on its own. This means that AES-CTR-256 encryption alone cannot verify that the plaintext message has not been tampered with. To provide integrity to the message, a separate authentication scheme has to be implemented. For example, an HMAC-SHA-256 hash can be generated using the message itself and sent in tandem with the ciphertext. The HMAC-SHA-256 hash can be verified and once verified, the message can be decrypted. Another example of integrity checking could be the use of GMACs or Galois MACs that provide an authentication tag that is sent in tandem with the cipher text. This MAC is generated using the same key used to encrypt the counters in this case.

Problem 2

```
• paulh@paulhan ~/w/e/l/l/part2 (master)> ./rsa
encrypted="_000e
              7000M
decrypted=Hello !
○ paulh@paulhan ~/w/e/l/l/part2 (master)> █
```

As with AES-CTR-256, RSA does not provide integrity authentication as well. To provide integrity, a random symmetric key scheme can be used to provide integrity on the message. For example, a SHA-256 hash can be calculated on the message and sent in parallel with the ciphertext. When the receiver receives both hashes, the message can be decrypted and a SHA-256 can be calculated on the message. If the calculated hash matches what was received, the message integrity has been verified and the message can be used safely.