Hamza Aziz
Haris Khan

Symmetric Key Cryptography Exploration Assignment Report

Assignment Questions:

1. For task 1, viewing the resulting ciphertexts, what do you observe? Are you able to derive any useful information about either of the encrypted images? What are the causes for what you observe?
   When CBC encryption was applied to the image none of the original image was preserved, due to the nature of the encryption on the following block where it's exclusive OR'd with the following block, creating unique cipher texts between the two blocks. When ECB encryption was applied the images were in a sense still legible where a user could probably tell which image corresponded to its respective original image. This is because due to the nature of the algorithm encrypting 16 bytes at a time relations are maintained.
2. For task 2, why is this attack possible? What would this scheme need in order to prevent such attacks?
   This attack is possible because of how CBC mode encryption works where each plain text block is XOR'd with the block that precedes it. As such the bits in each block are in sense mapped one to one where if a bit is altered in a preceding block, then the same bit will be affected in the corresponding block that follows. In order to prevent this from happening there needs to be some additional step to maintain the integrity of the data this could be done by adding a hash-based message authentication control.

3. For task 3, how do the results compare? Make sure to include the plots in your report.
   In task 3 we noticed that for AES encryption that increases/changes in block size provided negligible performance improvements relative to the throughput rate compared to the key size which significantly can impact performance with bigger key sizes enhancing performance significantly over the others. This can be seen in the graphs below. For the RSA algorithm it was interesting observing that sign operations didn't require much throughput as compared to their verify counterparts, although this was no surprise. As key size increased the throughput rate rapidly decreased.

Task 1 Details:

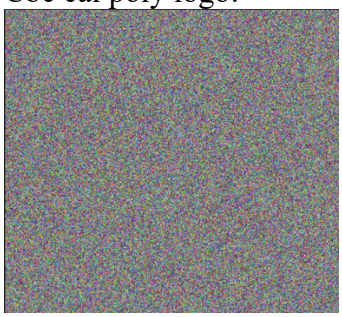
- Part one entailed implementing the ecb, cbc, and ecb encryption and decryption functions respectively. I began by importing the necessary modules from the cryptography library. I defined my block size to 16 bytes ~128 bits for AES encryption
- I made a series of utility functions including _pad_pkcs7 & _strip_pkcs7 related to applying and removing the PKCS#7 padding respectively.
- AES128 related utility functions are utilized in the ECB mode functions, they perform the AES encryption operations 16 bytes at a time when called by ECB functions.
  - Within ECB functions ecb encrypt takes plaintext data and a key, applying the padding to it following up by encrypting block by block. As you can imagine decrypt does the opposite with the result being the original plain text.
- Cbc encrypt is similar to ecb encrypt where instead it will take an additional parameter and each plain text block will be exclusive or'd with the previous block.

- Cbc decrypt will decrypt each block of cipher text and then exclusive or it with the ciphertexts block to get it's plain text.

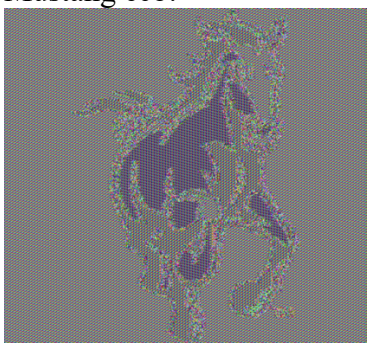
Resulting Images:

Ecb cal poly logo:

Cbc cal poly logo:



Mustang ecb:



Mustang cbc:

Task 2 Details:

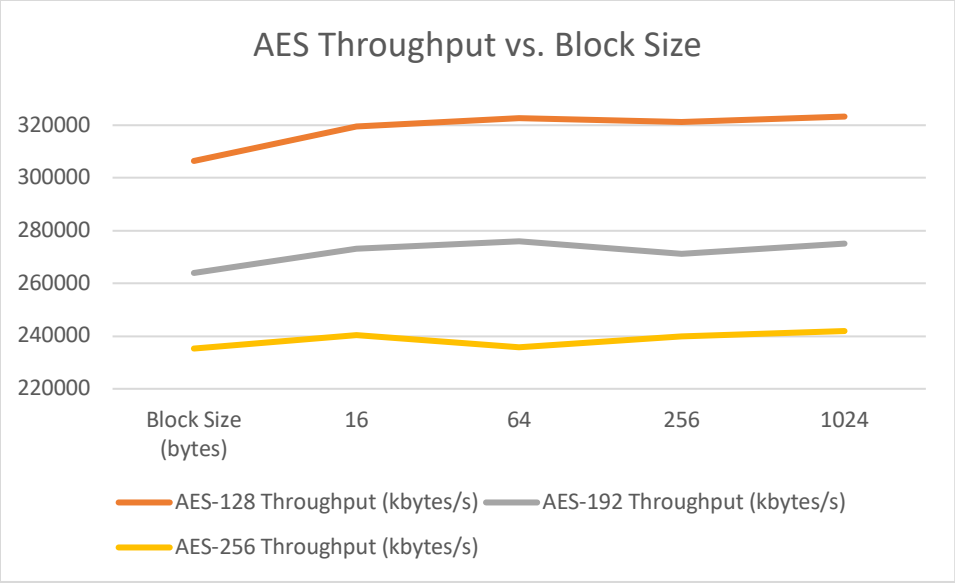
Within the second task we utilized the CBC algorithm that was written in task 1 earlier and implemented the submit and verify functions respectively. As expected the verify function will return false as expected indicating that it is in fact impossible with any input passed in by the user. In the secondary main function we did some experimenting and realized that we could flip the bits of the encrypted string for it to contain the 'admin=true' message string within it.

Task 3 Details:
Data from running OpenSSL on AES key sizes (128 bits, 192 bits, & 256 bits).
Commands invoked to get data:

1. openssl speed aes-128-cbc
2. openssl speed aes-192-cbc
3. openssl speed aes-256-cbc

	A	B	C	D
1	Block Size (bytes)	AES-128 Throughput (kbytes/s)	AES-192 Throughput (kbytes/s)	AES-256 Throughput (kbytes/s)
2	16	306442.49	263955.51	235272.23
3	64	319418.87	273084.79	240297.16
4	256	322730.47	275967.81	235683.54
5	1024	321167.15	271264.89	239959.41
6	8192	323284.02	275069.2	241921.98

Plot of data representing AES throughput vs. different block sizes:



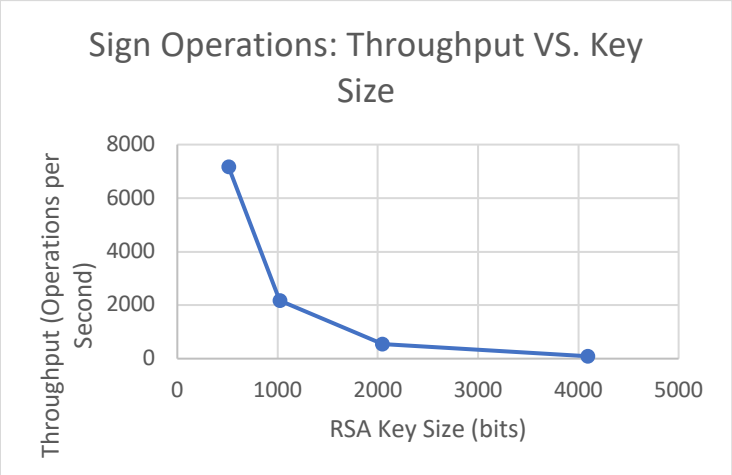
Data from open SSL speed tests on different RSA key sizes (512 bits, 1024 bits, 2048 bits, 4096 bits)
Commands invoked to get data:

1. openssl speed rsa512
2. openssl speed rsa1024
3. openssl speed rsa2048
4. openssl speed rsa4096

Sign operations:

	A	B	C
1	RSA Key Size (bits)	Time (s)	Operations/s
2	512	0.00014	7157.6
3	1024	0.000462	2163.2
4	2048	0.001862	537.1
5	4096	0.01118	89.4

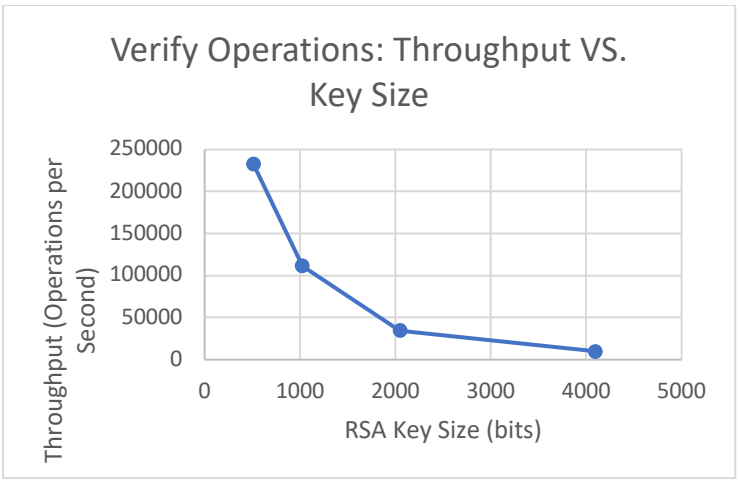
Plot:



Verify operations:

	A	B	C
1	RSA Key Size (bits)	Time (s)	Operations/s
2	512	0.000004	232458
3	1024	0.000009	111482.2
4	2048	0.000029	34513.6
5	4096	0.0001	9992.3

Plot:


Verify Operations: Throughput VS. Key Size
Throughput (Operations per Second)
250000
200000
150000
100000
50000
0
0
1000
2000
3000
4000
5000
RSA Key Size (bits)