Michael Lee

Assignment 2

3.1) Observation task – one-way hash and randomness

1. Created text file
2. Hash of file H1 = 3f03cd845ee8f5405d64055c08be02ca1c386b153e8f10ef19fdca85ffe406eb1c038fa4b9220ad414cc2cb88a1ef0e81b6c91339321d415808d2e0d0b7dd02d
3. 01010100 T -> 01010101 U
4. New hash of file H2 = a8ec5654b7a4b899465ff45a6b26eef75da9331756c39e91525ca22ffc782389b550250e12bda5d46d482dd5aab44ba2cb7dafe33864617a7aea8d1954b9a9a1
5. As you can see from above, the hashes are completely different using python counting the 1’s of H1 XOR H2 there are 247 different bits
6. After flipping the four bits: the difference between H1 and H3 is: Sha256 – 138 bits, Sha512 – 269 bits.

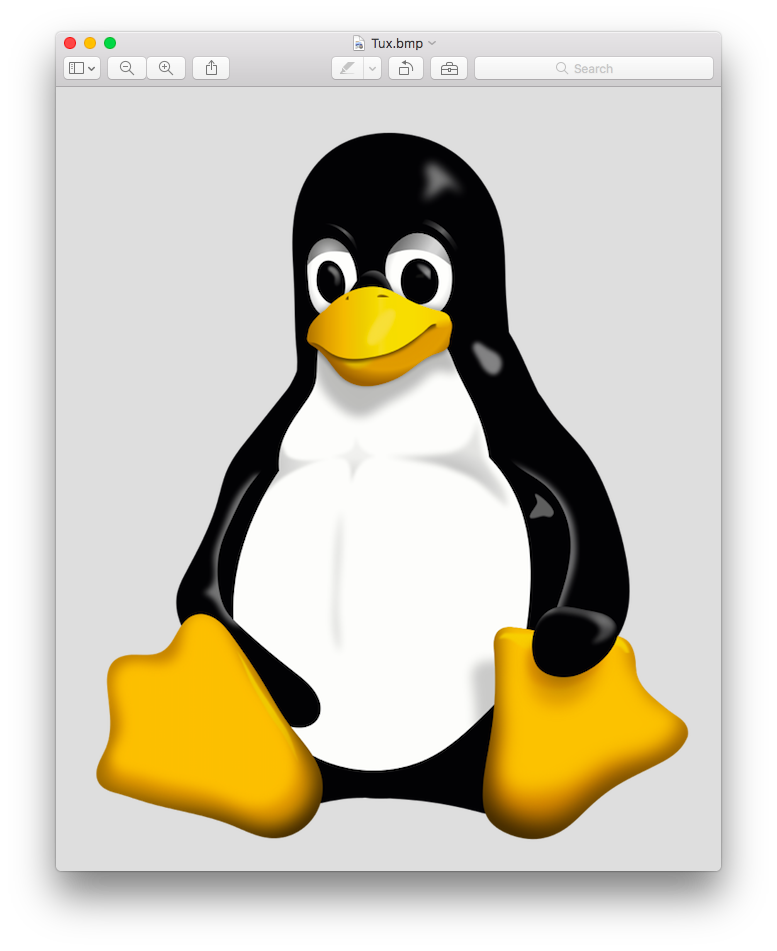
3.2) Observation task – Keyed hash and HMAC

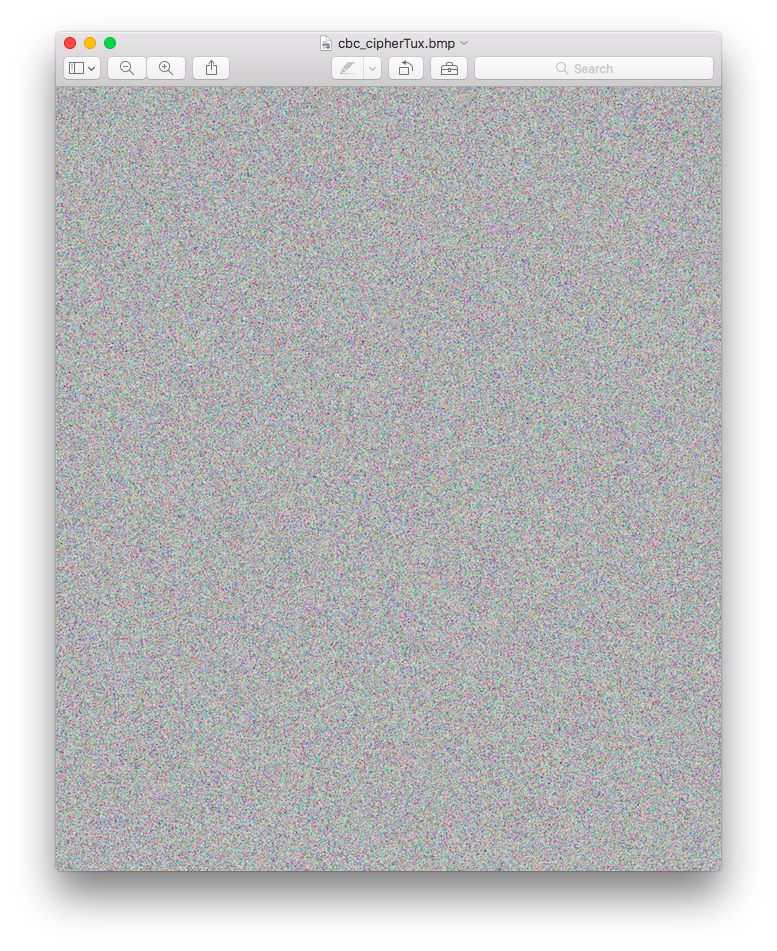
|  |  |  |  |
| --- | --- | --- | --- |
| Key length | | HMAC-SHA256 | HMAC-SHA512 |
| 128 | 0eb2f9a4f18df6da9651f76ce297f2ee73053894716cdeecb93293c03d68adad | | a3ceb3b8d41d3f52a15b3fa47973e14307566da8bf99d03819b842759c6615cb7f40a4f76680d6a7d26e6c7d0ba589ef90f55602cf5dcfd0bb79939d9149a9d6 |
| 160 | e0cf26b8d9a25ac6190783613fb9a6be446aeaf58fd765ee6c0b75c63b9df99f | | 28ec762dc6e13221b05f0585c7f2ce56bdea12f314d0825fafed94f9048b2c9dee91e011cd7288c25f760423136b8efcbdea344f6077b23b8a470c9906ed7c62 |
| 256 | 19bf321edcf1071da7519ccd1b559eb4dd1d31ad9e582792dd9de6cb922c094 | | 0d8cd404df18be92682d72b8fc9c20a8d64d5d90e2a10ac51dd95ed3f727584bed7eb4f179a37090e79e1f8785568337d9d19ae2043fc838b532811d325944e0 |

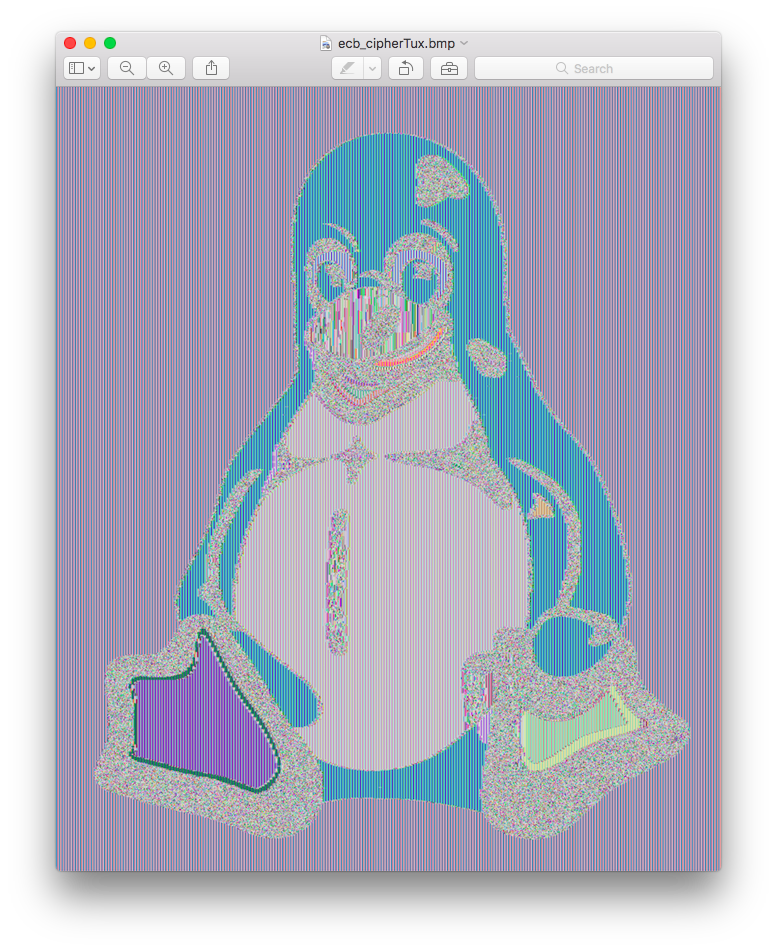
Judging from the keyed hashes that we get back in the table above, they are all distinct even though we aren’t using a fixed key size. From research I’ve found that as long as the key is above 128 bits it is infeasible to crack in our lifetimes, and so as long as we hit this point the keyed hashes will be secure, and any longer than the bit size of the hash is unnecessary (256 bits for sha256 and 512 for sha512).

3.3) Encryption using different ciphers and modes

-on next page-

Original - 

CBC - 

ECB - 

3.4) Observation Task: CBC Encryption using different IVs

1. Encrypt1.bin and Encrypt2.bin’s contents are exactly the same because we used the same key, iv, and file contents. It’s fairly obvious that this would result in the exact same output as the IV hasn’t been change so there’s no variability in the outcome.
2. Encrypt1.bin and Encrypt3.bin’s contents do not match, because we changed the IV and so it changes because the IV is XORed to alter the output.

3.5) These set of questions are based on a real world vulnerability in an open source File Storage application called OwnCloud. Please refer to this blog post for the details about the vulnerability and answer the following questions in brief(less than 5 lines):

1. Data is encrypted and stored on the Owncloud server using their own encryption module, but it is stored with exact filenames and other header info. They use malleable encryption modes, like CFB and CTR, which allow a hacker to flip cipher-text bits which get flipped in the resulting decrypted data. The hacker uses this principle to gain control of certain bits in the user’s uploaded file. Then the user then infects their own computer as Owncloud doesn’t verify what the user is downloading.
2. The encryption mode they used was AES-256 in Cipher Feedback Mode.
3. CFB has malleability which means that any bit flipped in the cipher-text gets flipped in the decrypted code. This means that the hacker can XOR the cipher-text with their own code and insert what they want into the decrypted code.
4. As mentioned in the article and what they eventually upgraded to, they could use authenticated encryption to overcome the limitation of malleability with just encryption. The author mentions AEADs, Galois/Counter-Mode, etc.
5. Code reuse is the attack method and they allow attackers to execute arbitrary code using jump statements to bypass security measure that prevent execution from certain regions of memory.