CS 5435 – Computer Security

Homework 3

Name: Bar Kadosh, Ben Kadosh *Fall 2019*

For this assignment, exercises in sections 1.1, 1.3, 1.4, 2.1, and 2.2 required code deliverables, which have been submitted along with this report. These scripts contain comments that explain our code and approach. This report captures the responses to short answer exercises 1.5, 2.3, 2.5, 3.1, and 3.2.

Exercise 1.5

1. Why does using GCM prevent mauling and padding oracle attacks?

   In order to understand why GCM prevents mauling and padding oracle attacks, it is important to first understand how GCM works and how it is different from CBC.

   Like CBC, GCM works by taking some plain text message and splitting it into blocks. In this case, the blocks are 128 bits, 16 bytes, in size. As noted on the GCM Wikipedia page, "GCM combines the well-known counter mode of encryption with the new Galois mode of authentication. The key feature is that the Galois field multiplication used for authentication can be easily computed in parallel. This option permits higher throughput than authentication algorithms, like CBC, which use chaining modes."

   There are several notable differences between the two modes of operation, the first of which is mentioned in the quote above. GCM does not use chaining during the encryption of blocks, whereas CBC is reliant on prior blocks to encrypt subsequent blocks. As stated above, this can have a noticeable impact on performance. In GCM, the plaintext is XORed with an encrypted value of a nonce (IV) concatenated with a counter of the respective block number. In addition to the IV and cipher-text that can be found in CBC, GCM also has an additional feature - an authentication tag. In the same Wikipedia article previously cited, it states the following: "The cipher-text blocks are treated as coefficients of a polynomial which is then evaluated at a key-dependent point H, using finite field arithmetic. The result is then encrypted, producing an authentication tag that can be used to verify the integrity of the data. The encrypted text then contains the IV, cipher-text, and authentication tag." This authentication tag is a unique feature that adds integrity to the message as well as encryption. The authentication tag is created by taking each block and passing it through a GHASH function.

   Now that we have covered the basic differences, we can discuss why GCM prevents these attacks. Because GCM has an authentication tag, we can verify that the cipher-text was not altered in any way to prevent a maul attack. In addition and as noted above, GCM does not rely on chaining to encrypt the plaintext. For this reason, altering previous blocks cannot provide any information on a current block, so the padding oracle attack would become ineffective. Furthermore, since we now have an authentication tag, we can verify that the cipher-text was not altered in any way to prevent a padding oracle attack.

2. For each attack above, explain whether enabling HTTPS for the entire payment site (as opposed to just the login page) prevents the attack if no other countermeasure is applied.

Enabling HTTPS for the entire payment site would prevent the maul and padding oracle attacks if no other countermeasures are applied. As noted in the articles cited below, HTTPS is a secure communication protocol that uses TLS to encrypt HTTP. Because HTTPS uses TLS for encryption, we can determine that enabling HTTPS will prevent the maul and padding oracle attacks. As is noted in “The Transport Layer Security (TLS) Protocol Version 1.3”, a memo by the IETF regarding TLS version 1.3 (the most updated version), “a TLS-compliant application MUST implement the TLS_AES_128_GCM_SHA256 [GCM] cipher suite and SHOULD implement the TLS_AES_256_GCM_SHA384 [GCM] and TLS_CHACHA20_POLY1305_SHA256 [RFC7539] cipher suites.” From this quote, we can conclude that GCM is embedded as an encryption and authetication mechanism within TLS, and therefore within HTTPS. As we proved in 1.4, we can prevent the maul and padding oracle attacks by using GCM, and by the same reasoning, using HTTPS, which uses TLS (which uses GCM), should prevent the same attacks as well.

Links:
https://https.cio.gov/
https://en.wikipedia.org/wiki/HTTPS
https://tools.ietf.org/id/draft-ietf-tls-tls13-23.html

Exercise 2.3

This attack relies on the attacker having knowledge of the SipHash key. Assuming SipHash is a PRF, does sampling a fresh random SipHash key for the hash table every time the web server is started prevent this attack? Why or why not?

The answer to this question is two-fold.

Assuming the attacker sends a dictionary of 1,000 strings as a parameter to the server, if we sample a fresh random SipHash key for the hash table every time the web server is started, it will essentially guarantee that the attack will be prevented. To understand why, let's look at the following fact stated in the original research paper that proposed SipHash, "SipHash: a fast short-input PRF". In the paper (link below), the authors state:

"Brute force will recover a key after on average 2^{127} evaluations of SipHash, given two input/output pairs (one being insufficient to uniquely identify the key). The optimal strategy is to work with 1-word padded messages, so that evaluating SipHash-c-d takes c + d SipRounds."

If we don't randomize the SipHash key for the hash table every time the web server is started, it is still very time and computationally intensive to brute force an attack to guess the key. If we do randomize the key, it will be nearly impossible to brute force and guess the key. Since it is so difficult to guess the key, and assuming the attacker is sending 1000 strings as input, the likelihood that a significant percentage of them will be collisions in the same bucket (if any) is extremely low given the randomness of the guess. Therefore, randomizing the key makes it difficult enough to dissuade an attacker from trying in most cases.

With that being said, while randomizing the key does prevent the attack in this situation, if the number of strings in the dictionary input can be expanded from 1,000 to 10,000,000, for example, then regardless of how often we randomize the key and how effective SipHash is, hash tables will by default eventually experience collisions if enough data is hashed. With that in mind, if an attacker sends enough data, even without knowing the key, it is possible that there will eventually be enough random strings hashed to a given bucket, generating enough collisions to cause a DoS attack to still be effective.

Link: https://131002.net/siphash/siphash.pdf

Exercise 2.5

One suggested countermeasure to denial-of-service attacks is proof of work: forcing clients to perform some computational work and including a proof in the request. Is this an effective countermeasure? Why or why not?

Yes, this is an effective countermeasure against DoS attacks. If a client wants to send a large number of requests to a server in order to flood the server(for example - with many SYN packets), making it computationally intensive to do so can help in several ways. It may dissuade the attacker and direct them elsewhere. By making it more computationally intensive to send a request, it also means the flow of traffic becomes slower, and if we slow down the traffic enough, the server will be able to process all the requests at a rate faster than the requests coming in, thus preventing a spike in requests that causes the server to crash. This does assume that the attacker will need to make numerous requests in order for the DoS attack to be effective. However, it should be noted that if, as mentioned above, the attacker can send a massive amount of data as a parameter to cause a hash DoS attack in one request, proof of work would not be an effective countermeasure. However, to compute such a massive amount of data would be very time-intensive, so the attacker might be dissuaded regardless.

Another challenge with PoW is finding the right balance of how much work needs to be done by a client when making a request, as we don't want to discourage genuine users. Lastly, PoW would not necessarily prevent DDoS attacks as each client could send one request and show proof of work. Proof of work would not be intensive for one request per client in the distributed system, but in total the request count would be large enough to still overwhelm the server. The assumption here is that having to show proof of work for repeated requests will eventually add up and discourage an attacker; however, one request with proof of work will not be so computationally expensive.

Exercise 3.1

Prove any encryption scheme which meets perfect secrecy also meets Shannon secrecy.

Given perfect secrecy we know the following: M = a random variable denoting a message sampled from a distribution D, and let C = a random variable defined as Enc(K, M), the encryption of M under a key K output by Gen. For all distributions D, all messages m, and ciphertexts c, P[M = m | C = c] = P[M = m] over the coins used to sample M and K, and any coins used in Enc.

We start with the expression for perfect secrecy:

$$P[M = m|C = c] = P[M = m] \tag{1}$$

Using the definition of conditional probability, we can represent this equation as:

$$\frac{P[M = m \cap C = c]}{P[C = c]} = P[M = m] \tag{2}$$

Multiplying both sides of the equation by the denominator on the left hand side rearranges the equation, as can be seen below. From this step we can observe that P[M = m] and P[C = c] and therefore M and C are independent random variables:

$$P[M = m \cap C = c] = P[M = m]P[C = c] \tag{3}$$

Again, we rearrange the equation to move P[M=m] to the denominator on the left hand side of the equation:

$$\frac{P[M = m \cap C = c]}{P[M = m]} = P[C = c] \tag{4}$$

We can now present the equation with the following conditional probability:

$$P[C = c|M = m] = P[C = c] \tag{5}$$

We can then substitute C with the value below on the left hand side as given by perfect secrecy:

$$P[Enc(K, M) = c|M = m] = P[C = c] \tag{6}$$

We can then drop the conditional statement on the left hand side given we already proved C and M are independent above

$$P[Enc(K, M) = c] = P[C = c] \tag{7}$$

Now we can see that since m is randomly chosen, P[Enc(K,m') = c] = P[C = c] for any given m'.

Therefore, we can substitute in two arbitrary values m_1 and m_2 such that,

$$P[Enc(K, m_1) = c] = P[C = c] \tag{8}$$

$$P[Enc(K, m_2) = c] = P[C = c] \tag{9}$$

Therefore, we can conclude that any encryption scheme which meets perfect secrecy also meets Shannon secrecy.

Exercise 3.2

Prove the converse: any encryption scheme meeting Shannon secrecy also meets perfect secrecy.

We start with the following given information:

The encryption scheme meets Shannon secrecy if for any pair of messages m1, m2 and any ciphertext c, P[Enc(K, m1) = c] = P[Enc(K, m2) = c] over the coins used to sample the key K using Gen.

First, let us establish that by definition of the above, given a message, the encryption of that given message results in some ciphertext, c, that is independent of any given message it is passed.

Now, let's generalize the above for any pair of messages m and m'. We start with the following expression:

$$P[Enc(K,m) = c] = P[Enc(K,m') = c] \tag{1}$$

We will rearrange these equations to account for the condition of the given messages from M:

$$P[Enc(K,m) = c|M = m] = P[Enc(K,m') = c|M = m'] \tag{2}$$

We can then say more generally that P[C = c] can be represented as the below where we take the right hand side of equation 2 for all possible messages m' in M (equation 3 is a generalization of equation 2 for all m'):

$$\sum_{m'} P[Enc(K,m') = c|M = m'] = P[C = c] \tag{3}$$

Given our independence assumption above, we can rearrange the probability as follows - we will also expand the left side and bring back P[C = c] further down:

$$\sum_{m'} P[Enc(K,m') = c]P[M = m'] \tag{4}$$

Because an encryption of a specific message m, results in a specific c, it can be factored out of the summation calculation. To do so we will select a specific value m and rewrite the above as follows:

$$P[Enc(K,m) = c]\sum_{m} P[M = m'] \tag{5}$$

At this step, we will notice that the probability of all possible messages m' in the space of random variable M must sum to 1. Therefore, we can simplify the expression to the following:

$$P[Enc(K, m) = c] * 1 \tag{6}$$

At this point we will begin to generalize this from a specific variable m to the random variable M. To do so, we will condition on M = m

$$P[Enc(K, M) = c|M = m] \tag{7}$$

We can then represent Enc(K, M) as C

$$P[C = c|M = m] \tag{8}$$

From here we will remember that this is equal to P[C = c] from equation 3 :

$$P[C = c|M = m] = P[C = c] \tag{9}$$

From here we can see as was proved in exercise 3.1 in equations 1 through 5 that P[C = c | M = m] = P[C = c] can be rewritten as P[M = m | C = c] = P[M = m] in reverse order from step 5 to step 1.

Thus from the above, we can prove that any encryption scheme meeting Shannon secrecy also meets perfect secrecy.