Assignment 3: Secure Minecraft API - Group 26

Phase 2-1

In writing this report, we have weighed the effectiveness of each of the four techniques against preventing Chosen Ciphertext Attacks (CCAs). A CCA is an attack where the attacker can submit any ciphertexts of their choice, and receive the plaintext back from the receiver, additionally a CCA attacker has all the abilities of an attacker using a chosen plaintext attack. The four techniques can be broadly divided into two schemes, Encrypt Last, and Encrypt First.

In the first scheme, the recipient has no way of knowing whether the ciphertext is from the stated sender (compromise in authentication and non-repudiation) and must decrypt it first. In doing so, the recipient inadvertently creates the possibility that information about the shared secret key can be exposed to a man in the middle (MITM). This scheme is malleable since it is possible for the attacker to modify a ciphertext and thereby cause a predictable change to its plaintext, which the attacker can then use to their advantage.

In the second scheme, the same vulnerability observed in the first scheme does not apply as the recipient now can check the integrity of the ciphertext without decrypting it. However, this scheme has its own vulnerabilities depending on how the MAC algorithm is implemented. For example, in the Encrypt-and-MAC sequence, no integrity on the ciphertext is guaranteed since the MAC is taken against the plaintext.

Given the security concerns of other techniques further discussed below, we have chosen Encrypt-then-MAC to achieve confidentiality and integrity simultaneously. Furthermore, we have decided to use a HMAC design instead of a MAC. The HMAC construction in fact offers stronger security measures than what's required of a MAC. We will now look at each of the four techniques, and our choice in more detail.

Hash-then-Encrypt: E(K, (M || H(M)) In this version of the *encrypt last* scheme, there is no use of MAC. However, the message is instead concatenated to the hash of the message, and then this entire bundle is encrypted. The fundamental difference between a MAC and a hash is that the MAC uses a secret key to provide both data integrity and data origin authentication. The hash on the other hand only serves as a measure for integrity of the message, which implies even though the plaintext may look wholesome according to the hash, there is no way of authenticating its origin. Moreover, the ciphertext needs to be decrypted before its integrity can be verified, which opens up the possibility of ensuing CCAs.

MAC-then-Encrypt: E(K2, (M || MAC(K1, M)). As with the previous technique, the weakness of this technique is that we must decrypt before we can assess the integrity of the ciphertext. Especially an issue under CCA if the encryption is malleable, where both the MAC and encryption could be altered. Compared to the previous technique, the MAC does provide integrity as it, unlike a HASH, a MAC includes a secret key in its creation, meaning the receiver can confirm that the message's sender had access to the secret key.

Encrypt-and-MAC: (C=E(K2, M), T=MAC(K1, M) This technique is theoretically vulnerable to CCAs due to the need to decrypt the ciphertext before we can assess the integrity. . As the MAC is taken against the plaintext, another vulnerability is created as Information about the plaintext in the MAC may be revealed if the MAC is too weak and leaks information to the attacker.

Encrypt-then-MAC: (C=E(K2, M), T=MAC(K1, C) Finally, we now take a look at our chosen technique (Which is used in IPSec). Jean Paul Degabriele and Kenneth G. Paterson in their study on the (In)Security of IPsec in MAC-then-Encrypt Configurations have not found any attacks against encrypt-then-MAC configurations of IPsec. The difference between Encrypt-then-MAC and Encrypt-and-MAC implementation is that the MAC is done on the ciphertext. This means that a security implementation using this technique does not require the decryption of the ciphertext in order to find out whether it has been meddled with. With the help of the MAC on the ciphertext, any discrepancy upon comparison will prompt the receiver to discard the message without having to decrypting it, making it highly resistant to known ciphertext attack, timing attacks, and chosen-ciphertext attacks(CCA). This has the secondary advantage of making a DDOS attack against the server running this protocol much more difficult as the computational requirement is decreased (no decryption and compare, just compare). Additionally any security weakness in the MAC (not a problem for our HMAC but possible in other MAC implementations) function is of no issue, as any leakage from the MAC will only leak information about the cipher, not the message, which is not helpful to an attacker.

Phase 2-2

In this section we will be further discussing the efficiency and security trade off of our **Encrypt-then-Mac** approach, through looking at each part of its implementation. Namely the **Encryption**, **MAC**, and **Diffie-Hellman Key** exchange.

Symmetric Encryption and Key Exchange:
We have decided to implement symmetric encryption for two main reasons. Firstly, symmetric encryption is more secure than asymmetric. Secondly it is faster than asymmetric encryption due to the fact that asymmetric encryption must do complex calculations for decryption. The only real issue with symmetric encryption is that we need to share the key. We have overcome this limitation by deriving our symmetric key via Diffie-Hellman (DH) key exchange. Courtesy of the design of a reliable TLS 1.2 handshake, we have the client send the DH parameters (ClientHello) and the server sends its public number (ServerHello), after which the client sends its public value as well (ClientKeyExchange) in our implementation. DH with a key of length 1024 and below is susceptible to an attack known as a logjam, where a prime number is broken beforehand and then used in attack. For this security reason we will be using the 2048 DH. While this does increase the time taken to derive the shared key we feel that since we are using AES for our symmetric encryption we want the DH to be as strong as possible to reap the benefits of the AES's security. We will be using a SHA-256 to extract the optimal length of the shared key to create the final symmetric key for actual encryption. Another weakness of DH is that due to the lack of a MAC there is no way to verify the originator of the messages, thus giving DH a weakness to MITM type attacks. We could certainly defend against such attacks through the use of a public certificate backed by a Certificate Authority. We assume that this process of authentication is initialised beforehand for our implementation in Phase 3. Finally as we are using our own implementation of DH it is possible that we introduce unknown errors which weaken the overall security of the DH algorithm.

Choice of Implantation Technique
In our design we have decided on using the default configuration of the Fernet from the cryptology module in python for encryption. We will be then decrypting the encrypted data using the javax crypto module at the server side. We decided to use Fernet after looking through its implementation and researching each component, which we will analyse further below. An overarching reason for this choice is that an important security concern is always human error. When it comes to implementing a technique involving encryption, a small error can cause a big security issue. Therefore it is best to use the modules that are provided when possible, as they have been validated by others. There is also efficiency in implementing the module code rather than writing it again ourselves.

Encrypting the message with AES-128:
As we are implementing the Fernet system we will be using the AES encryption algorithm and symmetric key and padding with PKCS7. The advantages of using AES is that it is extremely secure and used as standard in the US federal government encryption. We will be using the AES-128 bit key, while this key is technically not as secure to a brute force attack as AES-256, the AES-128 algorithm is still totally secure, to modern computers and even secure to modern quantum computers. Some experts even advise against using AES-256 due to theoretical weakness to related key attacks (Biryukov, Khovratovich, Nikoli´c, Distinguisher and Related-Key Attack on the Full AES-256). This weakness, the AES-128's being more than secure enough for modern standards, coupled with the fact that AES-256 is slower (more complex algorithm takes the computer longer to calculate), making our decision to go with AES-128 the obvious choice. We also use an IV (Initialisation Vector, also known as a Nonce) in the implementation of AES. The IV is a random value which adds randomness into the encryption and makes sure that the same message will produce a different cipher text each time it is created. Adding an IV adds a lot of security at very little cost so is a must in all implementations of AES.

Implementing the HMAC with SHA-256.
We will be using the SHA-256 algorithm to HASH our cipher text. The SHA-256 algorithm uses a Merkle-Damgard construction. One of the main reasons we are going with the SHA-256 is that it is the default implementation in the Fernet cryptography function, as we said above making changes to this implementation can potentially weaken security by introducing human error. Although the SHA-256 was created earlier than SHA-3 they are similar only in name, as SHA-256 and SHA-3 use totally different hashing algorithms. SHA-256 is still considered safe to brute force attacks and as a newer version of the SHA-2 algorithm series it is also resistant to length extension attack. SHA-2 functions overall have better performance(speed) than SHA-3 functions and so the SHA-2 implementation is the one we will be using. Again as our minecraft server is not mission critical the SHA-2 algorithm will be more than enough security for our HMAC design. The HMAC is then appended to the cipher text, a time stamp (which helps prevent replay attacks) and the IV to create the final package that we will send to our Spigot server.

In this report we have looked at our choice of technique Encrypt-then-Mac, by comparing it to other possible implementations, we also broke down the technique into its components and looked out how these affect its security and efficiency of the technique.

References

Squeamish Ossifrage. (2019). What is the difference between SHA-3 and SHA-256 [online forum]. Crypto stackexchange: (https://crypto.stackexchange.com/questions/68307/what-is-the-difference-between-sha-3-and-sha-256)

Colin Percival.(2009). Encrypt-then-MAC. Daemonic Dispatches. (https://www.daemonology.net/blog/2009-06-24-encrypt-then-mac.html)

Colin Percival.(2009). Cryptographic Right Answers. Daemonic Dispatches(https://www.daemonology.net/blog/2009-06-11-cryptographic-right-answers.html)

Kelalaka.(2020). Should we MAC-then-encrypt or encrypt-then-MAC. [online forum] . Crypto stackexchange:(https://crypto.stackexchange.com/questions/202/should-we-mac-then-encrypt-or-encrypt-then-mac)

CodeInChaos.(2015). Why can't I use the same key for encryption and MAC.[online forum]. Crypto stackexchange: (https://security.stackexchange.com/questions/37880/why-cant-i-use-the-same-key-for-encryption-and-mac)

Himanshu Tyagi. (2020). Symmetric Vs Asymmetric Encryption – which is better?. Code it Bro. (https://www.codeitbro.com/symmetric-vs-asymmetric-encryption/#symmetric-vs-asymmetric-encryption)

Atoponce .(2019). Sha-3 in HMAC [online forum]. Reddit: (https://www.reddit.com/r/crypto/comments/c6r3ql/sha3_in_hmac/

Padding (cryptography). (2021, October 14). In Wikipedia. (https://en.wikipedia.org/wiki/Padding_(cryptography))

Length extension attack. (2021, October 26). In wikipedia: (https://en.wikipedia.org/wiki/Length_extension_attack)

Block size (cryptography). (2018, June 14). In wikipedia: (https://en.wikipedia.org/wiki/Block_size_(cryptography))

Tmaher. (2014). Fernet Spec. Github (https://github.com/fernet/spec/blob/master/Spec.md)

William J Buchanan. (2021), *Fernet*, Asecuritysite. (https://asecuritysite.com/encryption/fernet)

Malen Abrakhimov. (2020). How does Fernet encryption work?. FindAnyAnswer. (https://findanyanswer.com/how-does-fernet-encryption-work)

User2213.(2016). Best protection against a "chose-ciphertext" attack?[online forum]. Crypto stackexchange: (https://security.stackexchange.com/questions/9781/best-protection-against-a-chosen-ciphertext-attack)

Ubiq Security. (February 15, 2021).Encryption: All of your questions answered. Ubiq (https://www.ubiqsecurity.com/blog/128bit-or-256bit-encryption-which-to-use/)

Mihir Bellare, Tadayoshi Kohno, Chanathip Namprempre. (2004). Breaking and Provably Repairing the SSH

Authenticated Encryption Scheme: A Case Study of the Encode-then-Encrypt-and-MAC Paradigm. (https://homes.cs.washington.edu/~yoshi/papers/SSH/ssh.pdf)

Hugo Krawczyk. The Order of Encryption and Authentication for Protecting Communications (Or: How Secure is SSL?). (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.5488&rep=rep1&type=pdf)

Graham Sutherland. (2013). Why "Mac then encrypt" is a bad choice. CODEINSECURITY. (https://codeinsecurity.wordpress.com/2013/04/05/quick-crypto-lesson-why-mac-then-encrypt-is-bad/)

Cherry Simancas.(2020). What is AES IV?. ASKINGLOT.(
https://askinglot.com/what-is-aes-iv#:~:text=An%20initialization%20vector%20%28IV%29%20is%20an%20arbitrary%20number,in%20the%20ciphertext.%20Click%20to%20see%20full%20answer.)

techTargetContributor. (2011).what is an initialization vector. WhatIs.com.
(https://whatis.techtarget.com/definition/initialization-vector-IV)

Rahil Arora. (2010). What is the Difference between a Hash and MAC? [online forum]. Stack overflow.
(https://stackoverflow.com/questions/2836100/what-is-the-difference-between-a-hash-and-mac-message-authentication-code)

Jean paul Degabriele, Kenneth G. Paterson. On the (In)Security of IPsec in MAC-then-Encrypt

Configurations. (https://www.isg.rhul.ac.uk/~kp/CCSIPsecfinal.pdf)

Matteo. (2012). HMAC vs MAC functions [online forum]. Crypto stackexchange.
(https://crypto.stackexchange.com/questions/2936/hmac-vs-mac-functions)

Bruce Scheneier. (2009). Another New AES Attack. Scheneier on Security.
(https://www.schneier.com/blog/archives/2009/07/another_new_aes.html)

Alex Biryukov, Dmitry Khovratovich, Ivica Nikolic. (2009). Distinguisher and Related-Key Attack on the Full AES-256 (Extended Version). University of Luxembourg.
(https://eprint.iacr.org/eprint-bin/getfile.pl?entry=2009/241&version=20090810:075245&file=241.pdf)

Ced. (2019). What is the difference between SHA-3 and SHA-256 [online forum]. Crypto stack exchange.
(https://crypto.stackexchange.com/questions/68307/what-is-the-difference-between-sha-3-and-sha-256)

Steve Lander. Advantages & Disadvantages of Symmetric Key Encryption Advantages & Disadvantages of Symmetric Key Encryption (itstillworks.com)

Mihir Bellare, New Proofs for NMAC and HMAC: Security Without Collision-Resistance
LNCS 4117 - New Proofs for <TEX>{\sf NMAC}</TEX> and <TEX>{\sf HMAC}</TEX>: Security Without Collision-Resistance (springer.com)

Intrigano, cryptography - Security Against Chosen Ciphertext Attacks
https://www.youtube.com/watch?v=YqfAZbiZUEY

Forward secrecy. (2021, October 31). In Wikipedia. (https://en.wikipedia.org/wiki/Padding_(cryptography))
Forward secrecy - Wikipedia

Vicente Revuelto, Krzysztof Socha (2016), Weaknesses in Diffie-Hellman Key Exchange Protocol
https://cert.europa.eu/static/WhitePapers/CERT-EU-SWP_16-002_Weaknesses%20in%20Diffie-Hellman%20Key%20v1_0.pdf

Sayalibagwe. Explain Diffie Hellman key exchange algorithm with example. Ques10.
https://www.ques10.com/p/7533/explain-diffie-hellman-key-exchange-algorithm-wi-1/

Diffie–Hellman key exchange. (2021, November 2). In Wikipedia.
https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange