Phase 2 - 2

Overview

Our CCA secure protocol will make use of encrypt then MAC to ensure the confidentiality and integrity of the message.

Our protocol will use the method of block cipher. Block cipher is a method of encryption where fixed sized blocks of data are encrypted.

The Encrypt then MAC method begins the process by encrypting the plaintext. This is done by running the plaintext through an encryption algorithm using a private key. The private key is a combination of the receiver's public key and the sender's private key. A MAC is then created on the ciphertext using the ciphertext and private key. Once the plaintext has been encrypted and the MAC has been created, the sender will append the MAC to the end of the ciphertext. This new packet of data is then sent to the receiver. While the data is being transmitted it is vulnerable to outsiders looking in and attempting to read the data or alter the contents of the package but we will return to this later.

Once the recipient has received the data, they will create a MAC on the cipher text using a private key. The private key is created by combining the sender’s public key and the receiver's private key. The private key will end up being the exact same as the one used by the sender. Once the MAC has been created, the receiver will compare it to the MAC that they received. If they are not equal, then the receiver can be certain that the data has been altered during transmission. If the two MAC are the exact same, then the receiver can be certain that the data he received is the same as the data that was originally sent to him. The receiver can then decrypt the message using his private key (which is the exact same key that the sender used to encrypt it).

Efficiency and Security advantages

Encrypt then MAC provides indistinguishability under CCA. Indistinguishability of a ciphertext is the property of privacy where an attacker can take two ciphertexts that are not distinguishable regardless of the plaintext on which the ciphertext has been created.

Our approach provides strong integrity of ciphertexts. An attacker can’t practically create a valid ciphertext other than the one the key holder generated. This is because the MAC is created on the ciphertext using the key holder's private key which means the attacker must know the private key and continually create ciphertexts until a text is created with the same MAC signature. The chances of this happening are astronomically low.

Encrypt then MAC also provides efficiency**.** The method of encrypt then MAC means that the MAC is created from the ciphertext. A packet that is received and has an inauthentic MAC can be immediately discarded without needing to be decrypted. This makes DOS attacks much harder as a system will simply discard invalid packets. Rather than needing to decrypt a packet then checking for authenticity such as an approach like MAC then encrypt.

The MAC does not provide any information about the plaintext. Given that the ciphertext appears random, the MAC will appear random too. This way the MAC doesn’t carry any information about the structure of the plaintext. This reduces the possibility of an attacker deducing the key from similarities between plain text and the MAC. This would not be true if the MAC was derived from the plaintext, i.e. MAC then Encrypt.

Reduces the attack surface which means that it reduces the points of vulnerability to an attack. For example, it denies the chance of inauthentic code that an attacker has injected into the data packet to make it into your system, as you will be able to verify its authenticity from the MAC.

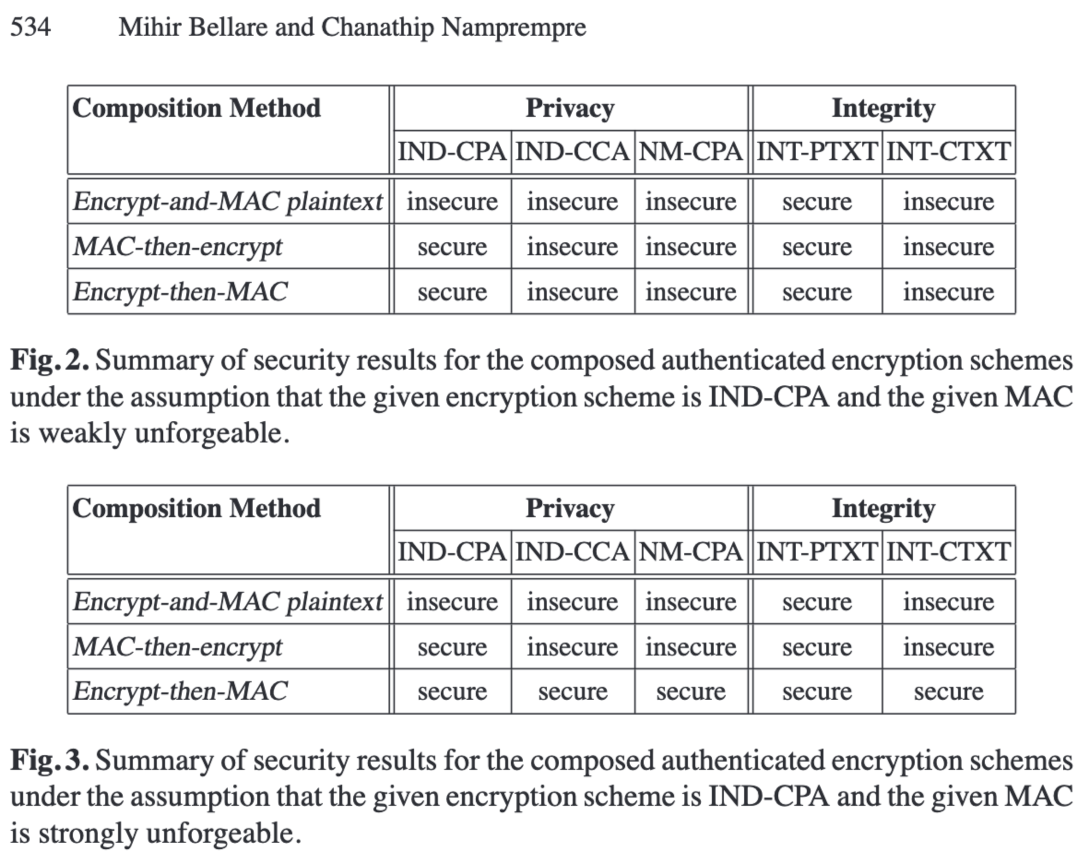
Efficiency and Security disadvantages

A potential disadvantage of encrypt then MAC is that an attacker may conduct a timing attack. A timing attack could be used as a method to find the valid MAC is where an attacker tries to brute force and send packets with different MAC's. This way if the attacker sends a valid MAC, the target system will take longer to compute as it decrypts the packet. However, MAC's range from 128-256 bits meaning that with current technology it would be unfeasible to try all possible combinations of MAC's.

Technical details and library use

For the MCPI API, we will use Python PyCryptoDome for our cryptographic functions. It supports RSA key generation, encryption, decryption and HMAC. We chose this library because upon research, there seems to be a lot of documentation and help online for the development of our protocol. For key generation we will be using RSA-2048. This means that the RSA key length is 2048 bits. This is acceptable for our application and the US National Institute of Standards and Technology has included this as a recommended key length, albeit the minimum recommended length. The keys will be stored in PEM format. For encryption, we will use PKCS#1 with OAEP padding. For ciphertext integrity, we will implement HMAC using SHA-256 and the private key. SHA-256 is a relatively modern algorithm for hashing and produces a 256 bit hash value. It currently does not have any known vulnerabilities to make it insecure. With SHA-256 it is almost impossible to reconstruct the ciphertext from the hash value and also having two ciphertexts with the same hash is extremely unlikely.

For Raspberry Juice (Java), we will use Javax Crypto and Java Security for our cryptographic functions. We will use these libraries to retrieve our generated key from PyCryptoDome and format it properly in Java so that we can decrypt it. Through Javax Crypto, we can specify our decryption algorithm and method which is RSA/ECB/OAEPPadding. With our private key and ciphertext we can use these libraries to decrypt the message. Through Javax crypto, we can also take our ciphertext and create a HMAC using SHA-256 and the private key. Again we found that these libraries are commonly used and had plenty of documentation.

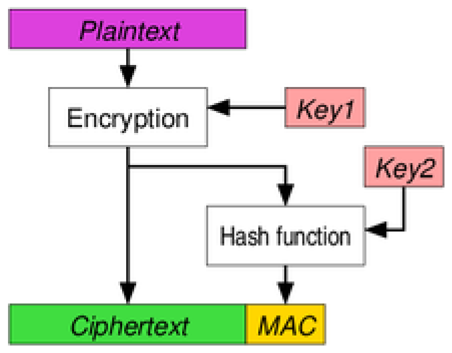


Composition Method	Privacy			Integrity	
	IND-CPA	IND-CCA	NM-CPA	INT-PTXT	INT-CTXT
Encrypt-and-MAC plaintext	insecure	insecure	insecure	secure	insecure
MAC-then-encrypt	secure	insecure	insecure	secure	insecure
Encrypt-then-MAC	secure	insecure	insecure	secure	insecure

Fig. 2. Summary of security results for the composed authenticated encryption schemes under the assumption that the given encryption scheme is IND-CPA and the given MAC is weakly unforgeable.

Composition Method	Privacy			Integrity	
	IND-CPA	IND-CCA	NM-CPA	INT-PTXT	INT-CTXT
Encrypt-and-MAC plaintext	insecure	insecure	insecure	secure	insecure
MAC-then-encrypt	secure	insecure	insecure	secure	insecure
Encrypt-then-MAC	secure	secure	secure	secure	secure

Fig. 3. Summary of security results for the composed authenticated encryption schemes under the assumption that the given encryption scheme is IND-CPA and the given MAC is strongly unforgeable.



References

- Bellare, M., & Namprempre, C. (2000, December). Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *International Conference on the*

Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg. Available at: https://link.springer.com/content/pdf/10.1007/3-540-44448-3_41.pdf (Accessed: October 26, 2022).

- Ringsmuth, E. (2014) *Encrypt-then-MAC*, *Medium*. Medium. Available at: https://medium.com/@ErikRingsmuth/encrypt-then-mac-fc5db94794a4 (Accessed: October 27, 2022).
- Stüvel, dr. S.A. (2020) *Python-RSA*, *dr. Sybren*. Available at: https://stuvel.eu/software/rsa/ (Accessed: October 27, 2022).