This report compares the effectiveness of the four techniques in preventing Chosen Ciphertext Attacks (CCA), and we will then choose one of them to be implemented. As defined by Menezes et al. (1996), CCA is an attack model in which the adversary obtains access to the decryption equipment (but not the decryption key) and which allows them to deduce the plaintext from (different) ciphertext, without having access to such equipment. To understand the four techniques, we need to have a basic understanding the following terms:
**Hashing** is a mathematical function that converts an input of arbitrary length into an encrypted output of a fixed length.
**Encryption** refers to the process in which an algorithm encodes data from plaintext into ciphertext.
**Message Authentication Code (MAC)** is a symmetric key cryptographic function that verifies the integrity of data.

Listed below are a brief summary of each CCA techniques along with their pros and cons:

**Hash-then-Encrypt:** The hash-then-encrypt method creates a unique 'fingerprint' or hash of the original message and is attached with the original message. The method will then encrypt the message and hash with a private key to unlock it. Hashing is a crucial component for verifying data integrity; if someone alters the message in transit, the recalculated hash upon receipt won't match the original, signalling the breach. However, this method involves two computational processes instead of just one, and if used frequently, the overhead can be significant, especially on systems with limited resources.

**MAC-then-Encrypt:** This technique provides an enhanced protection for the MAC value since attackers only have access to the encrypted value, which creates difficulty to forge the actual value. Furthermore, the technique follows the Horton Principle, stating ‘Authenticate what you mean, not what you said’ (Ferguson et al. 2010:104). This technique prevents the issue of decrypting the ciphertext using an incorrect key which arises from the encrypt-first approach. Incorrect key results in decryption errors meaning the actual plaintext not being received. Krawczyk (2001:312) claimed that this method is not secure by presenting a successful breach to the encryption function that provides secrecy for encrypted plaintext against chosen plaintext attacks. Moreover, this technique causes inefficiency since decryption is performed before authenticating the message’s legitimacy, thus causing additional computational overhead.

**Encrypt-and-MAC:** In this method, the process of encryption and message authentication are being done and sent separately. The plaintext is passed to both the encryption and MAC algorithm to get a corresponding ciphertext and tag respectively, resulting in a ciphertext-tag pair. Because the encryption and authentication processes can happen in parallel, it increases the performance of the cryptography, though Ferguson et al. (2010) argues that security is more important than performance. However, he also states that this method cannot guarantee the protection of privacy. This is because since the processes are carried out separately, while enciphering the same message twice, the adversary can see the MAC tag of the message being identical because it is not encrypted. Thus, leaking private information about the underlying message and compromising the privacy of the secure channel (Ferguson et al. 2010).

**Encrypt-then-MAC:** This is a two-step method of data transmission that first involves encrypting the data, generating a ciphertext, then applying an MAC to the ciphertext. The ciphertext and MAC would both have to be transferred from the sender to the receiver in order to verify the authenticity of the data received. Before arriving at the recipient, the message is decrypted. This method is preferred over Authenticate-then-MAC as suggested by Krawczyk (2001) since it is more resistant against CCA-schemes. Furthermore, it is also chosen over the Encrypt-and-MAC method as the MAC is applied over the ciphertext as opposed to the plaintext, providing another layer of security. Lastly, in comparison to Hash-then-MAC, the Encrypt-then-MAC method boasts superior confidentiality of data, as the prior encrypts the hash value of the data, whereas the latter encrypts the data itself. Despite the flexibility and security of this method, flaws such as higher computational expenses, as well as longer processing times are evident due to the complexity of the method’s approach. Regardless of this, the flexibility and security of this method allows it to be our method of choice.

Based on the above discussion, we conclude that **Encrypt-then-MAC** is the best approach in preventing CCA.

## Chosen Technique: Encrypt-then-MAC

Encryption techniques can be divided into two generic categories: symmetric and asymmetric. Even though symmetric encryption is faster and more efficient for handling large volumes of data, we chose to implement **asymmetric encryption** since it is more secure than the other (Mathur V 2022).

## Asymmetric Encryption

Asymmetric encryption allows two users to communicate privately without prior agreement on any secret information. The receiver of the message generates a pair of keys, called the public key and the private key. Katz and Lindell (2007:317) mentions that this encryption is called asymmetric because 'the sender uses the public key to encrypt a message for the recipient, whereas the recipient uses the private key to decrypt the resulting ciphertext'. A requirement of this encryption is that it relies on the privacy of the private key rather than the public key. Hence, it is vital for the receiver to not reveal their private key to anyone. Furthermore, this encryption is also safer because the public key can be delivered via a public channel from one person to another without jeopardising security. Additionally, it allows numerous senders to privately communicate with a single receiver (Katz and Lindell 2007), which ultimately reduces the number of keys required in a large network (Menezes et al. 1996). There are numerous asymmetric encryption algorithms and after researching them, we decided to go with the **RSA algorithm**.

## RSA

Rivest-Shamir-Adleman (RSA) is a widely-used asymmetric encryption algorithm which its security relies on the difficulty in factoring the product of two prime values with the decryption key being unknown.

To generate public key $(e, n)$:

1) Compute $n$ as the product of two randomly generated large prime numbers, p and q
2) Compute the Euler totient function for $n$, $\phi(n) = (p - 1)(q - 1)$
3) Compute $e$, which is relatively prime to $\phi(n)$ using Euclid algorithm

To generate private key $d$:

1) Compute $d$ as the multiplicative inverse mod($\phi(n)$) of $e$, such that $de$ mod $(\phi(n)) = 1$ using Extended Euclidean algorithm

R. L. Rivest et al. (1978:125) recommended that $n$ be about 200-digit long such that Richard Schroeppel's factoring algorithm takes $3.8 \times 10^9$ years to perform $1.2 \times 10^{23}$ operations to get the valid $p$ and $q$ values, assuming each operation takes 1μs. However, 50-digit $n$ value took only 3.9 hours to get the two prime values which present the possibility of breaking the algorithm when the value is small enough. Having the factors of $n$ enable attackers to compute $\phi(n)$ and thus the private key $d$. Therefore, we can claim that it is computationally impractical to factor the product $n$ when public key $n$ is at least 200 digit-long.

Message $M$ is converted into a series of integer blocks for encryption using the block cipher concept in which its block sizes depend on key-length. The ciphertext is achieved using $M^e \ (mod\, n)$ and decrypting it by $C^d \ (mod\, n)$ yields the message. These methods are known as trapdoor one-way functions since it is easy to compute in one direction but difficult in the reverse without knowing the weakness of the algorithm. When the algorithm, the ciphertext, and public key are revealed to attackers, trial-and-error in decrypting ciphertext is inefficient using the encryption algorithm (Schneier 2015).

R. L. Rivest et al. proved that computing $M^e \ (mod\, n)$ and $C^d \ (mod\, n)$ are efficient since it 'requires at most $2 \cdot log_2(e)$ multiplication and $2 \cdot log_2(e)$ divisions using the exponentiation by repeated squaring and multiplication' procedure. A high-speed computer could 'encrypt a 200-digit message within a few seconds' using the procedure (Rivest et al., 1978:123). Attempting to compute the private key without computing $\phi(n)$ and factoring $n$ is ineffective since all private key values are uncommon and are differ by the least common multiple of $(p - 1)(q - 1)$, meaning brute-force search is infeasible for attackers (Rivest et al. 1978:125). Hence, RSA's security relies on management for private keys since revealing the key would break the RSA.

## Message Authentication Code

According to Alomair et al. (2014:121), MAC is 'either unconditionally or computationally secure based on their security'. Computationally secure MACs ensure message integrity against attackers with limited computational power, whereas unconditionally secure MACs are only secure when attackers have unlimited access to computing power (Alomair and Poovendran 2014). Block cipher-based, stream cipher-based, and hash function-based are the three types of computationally safe MACs. Out of the three, we choose to use **Hashing Based Message Authentication Code (HMAC)** for message authentication and integrity because it is faster than stream cipher-based MACs and is more secure than block cipher-based MACs (Buchanan, B., OBE, Prof. 2021). Below is a more thorough description of HMAC.

## HMAC

'The definition of HMAC requires a cryptographic hash function, which we denote by H, and a secret key K. We assume H to be a cryptographic hash function where data is hashed by iterating a basic compression function on blocks of data.' (Krawczyk et al. 1997:3). The HMAC formula is as follows:

$$HMAC(K, m) = H((K \oplus opad) \,||\, H((K \oplus ipad) \,||\, m))$$

Key generation involves a few crucial steps:

1) Obtain the secret key (K) and the message (m).
2) If K's length is more than the block size (64 bytes or 128 bytes depending on what function is being used), we use MD5 as an example which is 64 bytes, hash (H) K. If it's less, pad K with zeros until it equals the block size.
3) Create ipad by XORing K with a 0x36 byte block, and opad by XORing K with a 0x5C byte block.
4) Concatenate ipad and m, hash this to get the inner hash.
5) Concatenate opad and the inner hash, then hash it to get the HMAC.

Using HMAC-SHA256, by using the message *"This is a very secret message"* and the key *"mykey"*, you should get "870c12ab67556e7798d1ea92d189552d5fb7d601bca563b8b5949aaf46677874" as the result.

As long as the underlying hash function is safe, HMAC offers great integrity and authenticity and is incredibly resilient to cryptographic assaults, such as collision attacks and preimage attacks (Bellare 2006). HMAC SHA-256 is considered so secure due to the sheer amount of computational power required to 'break' it. There are approximately $2^{256}$ possible hash outputs. In other words, even if we could check one quintillion hashes every second on one quintillion computers for one quintillion years, we'd only have a 0.00000000000000000000000000001% chance of finding a duplicate hash (a collision) using the "birthday paradox" formula (Boneh, D. and Shoup, V. 2023:310). However, Krawczyk et al. (1997:3) states that 'HMAC requires careful key management as its security depends on the secrecy of the key used.' Moreover, Menezes et al. (1996:328) emphasise that 'it does not provide non-repudiation as the same key is shared between the sender and receiver.' Despite these challenges, HMACs are essential in cybersecurity for their strength and versatility, but they should be used as part of a broader security strategy rather than as a standalone solution.

## HMAC with SHA-256

Secure Hash Algorithm 256, is known for its cryptographic strength and reliability. Compared to other hashing algorithms such as MD5 and SHA-1, SHA-256 offers a higher degree of security due to its larger bit length, resulting in a more complex hash value that is less susceptible to collision attacks (National Institute of Standards and Technology 2008:8). For instance, MD5 has been declared cryptographically broken and unsuitable for further use due to its vulnerability to collision attacks (Stevens 2012:46). Similarly, SHA-1 has been officially deprecated for most cryptographic functions since it too is vulnerable to attacks (Dang 2012:7). SHA-256 remains a robust choice due to its widespread support and proven security track record.

## REFERENCES


Alomair, B. and Poovendran, R. (2014). 'Efficient Authentication for Mobile and Pervasive Computing', *IEEE Transactions on Mobile Computing*, 13(3), New Jersey: IEEE, pp.469–481, doi:10.1109/tmc.2012.252.

Alomair, B., Clark, A. and Poovendran, R. (2010) 'The power of primes: security of authentication based on a universal hash-function family', *Journal of Mathematical Cryptology*, 4(2), Berlin: De Gruyter, pp.121-148. doi:10.1515/jmc.2010.005.

Bellare, M. (2014). 'New Proofs for NMAC and HMAC: Security without Collision Resistance', *Journal of Cryptology*, 28(4), p.28. doi:10.1007/s00145-014-9185-x.

Bellare, M., Kohno, T. and Namprempre, C. (2002). 'Authenticated encryption in SSH: provably fixing the SSH binary packet protocol', *Proceedings of the 9th ACM conference on Computer and communications security*, pp.1-11.

Bodewes, M. (2018). *When do I need to use CBC and HMAC?*, Stack Exchange website, accessed 30 May 2023. https://crypto.stackexchange.com/questions/61475/when-do-i-need-to-use-cbc-and-hmac

Boneh, D., and Shoup, V. (2023). 'Case Study : HMAC', *A Graduate Course in Applied Cryptography*, p.310. https://toc.cryptobook.us/

Buchanan, B., OBE, Prof. (2021). *I Know HMAC, But What's CMAC?*, Medium website, accessed 30 May 2023. https://medium.com/asecuritysite-when-bob-met-alice/i-know-hmac-but-whats-cmac-b859799af732#:~:text=Generally%20CMAC%20will%20be%20slower,likely%20run%20faster%20than%20HMAC

Dang, Q. (2012). *Recommendation for Applications Using Approved Hash Algorithms.* Maryland: National Institute of Standards and Technology, pp.7. Available at: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=911479

Ferguson, N., Schneier, B. and Kohno, T. (2010). *Cryptography Engineering*. Indianapolis, Indiana: Wiley Publishing, pp.37, 43, 102–104, 128.

fgrieu (2018). *Secure MAC and secure encryption schemes combined to form insecure Authenticated Encryption?*, Stack Exchange website, accessed 28 May 2023. https://crypto.stackexchange.com/questions/59744/secure-mac-and-secure-encryption-schemes-combined-to-form-insecure-authenticated

Fortinet (n.d.) *What is a Message Authentication Code?*, Fortinet website, accessed 29 May 2023. https://www.fortinet.com/resources/cyberglossary/message-authentication-code#:~:text=To%20encrypt%20a%20message%2C%20the,computation%20is%20the%20message's%20MAC.

Fuchs, Jonathan, Rotella, Yann, and Daemen, Joan. (2002). 'On the Security of Keyed Hashing Based on Public Permutations', *The International Association for Cryptologic Research*, 11(72), p.1.

Katz, J. and Lindell, Y. (2007). *Introduction to Modern Cryptography*. London: Chapman & Hall/CRC, p.317.

Krawczyk, H. (2001). *The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)*. Heidelberg, Berlin: Springer-Verlag, pp.310–331. doi:10.1007/3-540-44647-8_19.

Krawczyk, H., Bellare, M. and Canetti, R. (1997). *HMAC: Keyed-Hashing for Message Authentication.* RFC Editor website, p.3, doi:10.17487/RFC2104.

Mathur, V. (2022). *Symmetric vs Asymmetric Encryption*, Analytics Steps website, accessed 29 May 2023. https://www.analyticssteps.com/blogs/symmetric-vs-asymmetric-encryption

Menezes, A., van Oorschot, P. V. and Vanstone, S. (2001). *Handbook of Applied Cryptography*. Florida: CRC Press, p.41, 328.

Miya, A. (2023). *Message Authentication Code (MAC) in Cryptography*, usemynotes website, accessed 30 May 2023. https://usemynotes.com/message-authentication-code-mac/

National Institute of Standards and Technology (2008). *Recommendation for Applications Using Approved Hash Algorithms*. SP 800-107, p.8.

Rivest, R.L., Shamir, A. and Adleman, L. (1978). *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. New York: Association for Computing Machinery, 21(2), pp.120-126. doi:10.1145/359340.359342.

Schneier, B. (2015) 'Chapter 19.3 RSA', *Applied cryptography: Protocols, algorithms and source code in C 20th Anniv*. Indianapolis, Indiana: John Wiley & Sons.

Stevens, M. (2012). *Attacks on Hash Functions and Applications*. Netherlands: Universiteit Leiden. p.46.

Tutorialspoint (n.d.) *Message Authentication*, Tutorialspoint website, accessed 29 May 2023. https://www.tutorialspoint.com/cryptography/message_authentication.htm#