Encryption and Hashing algorithm

# Encryption

# AES (Cipher Feedback Mode)

AES in Cipher Feedback mode (CFB) mode transform a block cipher into a stream cipher on real time without incorporating padding. Because of this, it is particularly helpful for applications that manage constant data streams. The National Institute of Standards and Technology (NIST) has documented its use in securing communication protocols (National Institute of Standards and Technology [NIST], 2001).

|  |  |
| --- | --- |
| Run | AES (Cipher Feedback Mode) Key Size: 128 bits |
| 1 | 0.203 |
| 2 | 0.180 |
| 3 | 0.207 |
| 4 | 0.219 |
| 5 | 0.221 |
| Average time | 0.206 |

|  |  |
| --- | --- |
| Run | AES (Cipher Feedback Mode) Key Size: 192 bits |
| 1 | 0.227 |
| 2 | 0.221 |
| 3 | 0.217 |
| 4 | 0.203 |
| 5 | 0.206 |
| Average time | 0.2148 |

|  |  |
| --- | --- |
| Run | AES (Cipher Feedback Mode) Key Size: 256 bits |
| 1 | 0.202 |
| 2 | 0.190 |
| 3 | 0.197 |
| 4 | 0.216 |
| 5 | 0.213 |
| Average time | 0.2036 |

**\*Input text is - hello kitty is not a cat\***

Observation: The 256-bit key works better than the 192-bit key, despite the common belief that larger keys invariably result in worse performance. This suggests that longer keys can still be advantageous in some situations since hardware optimizations and implementation efficiency can occasionally overcome the expected performance drawbacks.

# AES (Counter Mode)

AES in Counter (CTR) mode convert blocks of plaintext input into encrypted ciphertext with incrementing counter, which makes parallel encryption possible. AES in counter mode is commonly used in situations where the systems require encryption completed in high speed such as disk encryption. NIST Publication covers great details on AES in counter (CTR) mode (National Institute of Standards and Technology [NIST], 2001).

|  |  |
| --- | --- |
| Run | AES (Counter Mode) Key Size: 128 bits |
| 1 | 0.199 |
| 2 | 0.196 |
| 3 | 0.199 |
| 4 | 0.193 |
| 5 | 0.192 |
| Average time | 0.1958 |

|  |  |
| --- | --- |
| Run | AES (Counter Mode) Key Size: 192 bits |
| 1 | 0.193 |
| 2 | 0.208 |
| 3 | 0.209 |
| 4 | 0.195 |
| 5 | 0.223 |
| Average time | 0.2056 |

|  |  |
| --- | --- |
| Run | AES (Counter Mode) Key Size: 256 bits |
| 1 | 0.179 |
| 2 | 0.205 |
| 3 | 0.238 |
| 4 | 0.212 |
| 5 | 0.204 |
| Average time | 0.2076 |

**\*Input text is - hello kitty is not a cat\* (21 Bytes)**

Observation: Like CFB, the CTR mode demonstrates that the 192-bit key lengthens the encryption time. Compared to the 128-bit key, the 256-bit key takes a little longer. This is consistent with the typical pattern that larger key sizes result in marginally longer processing times because of their greater complexity.

# AES (Galois Counter Mode)

AES-GCM (Galois Counter Mode) combines confidentiality along with data integrity by introducing an authentication tag. Known for speed and security, AES-GCM is often used in transport layer security (TLS). NIST nominated GCM as part of their cryptography recommendation (National Institute of Standards and Technology [NIST], 2007).

|  |  |
| --- | --- |
| Run | AES (Galois Counter Mode) Key Size: 128 bits |
| 1 | 0.191 |
| 2 | 0.237 |
| 3 | 0.246 |
| 4 | 0.201 |
| 5 | 0.203 |
| Average time | 0.2156 |

|  |  |
| --- | --- |
| Run | AES (Galois Counter Mode) Key Size: 192 bits |
| 1 | 0.168 |
| 2 | 0.304 |
| 3 | 0.272 |
| 4 | 0.328 |
| 5 | 0.289 |
| Average time | 0.2722 |

|  |  |
| --- | --- |
| Run | AES (Galois Counter Mode) Key Size: 256 bits |
| 1 | 0.267 |
| 2 | 0.300 |
| 3 | 0.262 |
| 4 | 0.258 |
| 5 | 0.319 |
| Average time | 0.2812 |

**\*Input text is - hello kitty is not a cat\* (21 Bytes)**

Observation: With larger key sizes, GCM exhibits an increasingly significant rise in time, especially when comparing 128-bit and 192-bit keys. It also suggests that, in contrast to CFB and CTR, GCM has higher computational expenses and larger key sizes even though it delivers integrity and confidentiality.

# ChaCha20-Poly1305

Combining ChaCha20 stream cipher with Poly1305 message authentication code achieve both confidentiality and authentication. Proven to be efficient in resource-limited environments such as handheld devices and yet demonstrates resilience against timing attacks, ChaCha20-Poly1305 is an encryption algorithm that’s popular among modern security protocols (Internet Engineering Task Force [IETF], 2018).

|  |  |
| --- | --- |
| Run | ChaCha20-Poly1305 Key Size: 256 bits |
| 1 | 0.002 |
| 2 | 0.000 |
| 3 | 0.000 |
| 4 | 0.001 |
| 5 | 0.000 |
| Average time | 0.0006 |

**\*Input text is - hello kitty is not a cat\* (21 Bytes)**

Observation: Compared to all AES modes, ChaCha20-Poly1305 is substantially faster, which is particularly beneficial in circumstances with limited resources. Regardless of key size, it performs consistently.

# Fernet

The unique element about fernet lies between combining AES in CBC mode with HMAC for integrity check while integrating high-level API to provide symmetric encryption. Fernet allows encryption and authentication to be completed within one step (Python Cryptography Developers, n.d.).

|  |  |
| --- | --- |
| Run | Fernet  Key Size: 256 bits |
| 1 | 0.375 |
| 2 | 0.390 |
| 3 | 0.275 |
| 4 | 0.375 |
| 5 | 0.274 |
| Average time | 0.3378 |

**\*Input text is - hello kitty is not a cat\* (21 Bytes)**

Observation: In terms of processing time, Fernet is slower than ChaCha20-Poly1305 and, in general, slower than AES. This longer delay may be attributed to a combination of HMAC and encryption.

# Hashing

# SHA-256

A member of SHA-2 family of cryptographic hash algorithms which provide resiliency against collision attacks and hence frequently utilized in generating digital signature and certificates. SHA-256 generates 256-bit hash value from a plaintext input (National Institute of Standards and Technology [NIST], 2012).

|  |  |
| --- | --- |
| Run | SHA-256 |
| 1 | 0.000 |
| 2 | 0.000 |
| 3 | 0.000 |
| 4 | 0.000 |
| 5 | 0.000 |
| 6 | 0.000 |
| 7 | 0.000 |
| 8 | 0.000 |
| 9 | 0.000 |
| 10 | 0.000 |
| 11 | 0.000 |
| 12 | 0.000 |
| 13 | 0.000 |
| 14 | 0.000 |
| 15 | 0.000 |
| 16 | 0.000 |
| 17 | 0.000 |
| 18 | 0.000 |
| 19 | 0.000 |
| 20 | 0.000 |
| 21 | 0.000 |
| 22 | 0.000 |
| 23 | 0.000 |
| 24 | 0.000 |
| 25 | 0.000 |
| 26 | 0.000 |
| 27 | 0.000 |
| 28 | 0.000 |
| 29 | 0.000 |
| 30 | 0.000 |
| 31 | 0.000 |
| 32 | 0.000 |
| 33 | 0.000 |
| 34 | 0.000 |
| 35 | 0.000 |
| 36 | 0.000 |
| 37 | 0.000 |
| 38 | 0.000 |
| 39 | 0.000 |
| 40 | 0.000 |
| 41 | 0.000 |
| 42 | 0.000 |
| 43 | 0.000 |
| 44 | 0.000 |
| 45 | 0.000 |
| 46 | 0.006 |
| 47 | 0.000 |
| 48 | 0.000 |
| 49 | 0.000 |
| 50 | 0.000 |
| Average time | 0.00012 |

**\*Input text is - hello kitty is not a cat\* (21 Bytes)**

Observation: Applications needing fast hash computations can benefit from SHA-256's exceptional speed and efficiency in hash generation.

# Argon2 Hash

Argon2 is a password hashing method that received awards for its strong security and ability endure brute-force attacks. Because it uses a lot of memory, it is considerably harder for attackers to break passwords using specialist gear. Because of this algorithm's great degree of adaptability, users may modify its parameters to suit their security requirements (Biryukov et al., 2016).

|  |  |
| --- | --- |
| Run | Argon2 Hash |
| 1 | 0.248 |
| 2 | 0.141 |
| 3 | 0.144 |
| 4 | 0.186 |
| 5 | 0.149 |
| 6 | 0.199 |
| 7 | 0.154 |
| 8 | 0.143 |
| 9 | 0.138 |
| 10 | 0.264 |
| 11 | 0.278 |
| 12 | 0.150 |
| 13 | 0.142 |
| 14 | 0.132 |
| 15 | 0.165 |
| 16 | 0.188 |
| 17 | 0.133 |
| 18 | 0.157 |
| 19 | 0.151 |
| 20 | 0.247 |
| 21 | 0.280 |
| 22 | 0.153 |
| 23 | 0.166 |
| 24 | 0.224 |
| 25 | 0.146 |
| 26 | 0.240 |
| 27 | 0.138 |
| 28 | 0.138 |
| 29 | 0.145 |
| 30 | 0.187 |
| 31 | 0.161 |
| 32 | 0.132 |
| 33 | 0.146 |
| 34 | 0.269 |
| 35 | 0.147 |
| 36 | 0.257 |
| 37 | 0.209 |
| 38 | 0.202 |
| 39 | 0.168 |
| 40 | 0.182 |
| 41 | 0.171 |
| 42 | 0.324 |
| 43 | 0.184 |
| 44 | 0.177 |
| 45 | 0.215 |
| 46 | 0.209 |
| 47 | 0.219 |
| 48 | 0.297 |
| 49 | 0.387 |
| 50 | 0.238 |
| Average time | 0.1924 |

A graph of a graph

Description automatically generated

**\*Input text is - hello kitty is not a cat\* (21 Bytes)**

Observation: Because to its memory-hard design, which makes it more resistant to brute-force attacks, Argon2 is slower than SHA-256. This draws emphasis to the trade-off between security and speed.

# Scrypt Hash

The password hashing function Scrypt is designed to be memory-hard and computationally complex, which helps protect against attacks that use specialized technology, such as ASICs. Because of its ability to resist extensive brute-force attacks, it is commonly used in cryptocurrencies and secure password storage (Percival, 2009).

|  |  |
| --- | --- |
| Run | Scrypt Hash |
| 1 | 56.944 |
| 2 | 60.944 |
| 3 | 61.156 |
| 4 | 129.573 |
| 5 | 133.409 |
| 6 | 52.808 |
| 7 | 58.202 |
| 8 | 53.744 |
| 9 | 63.796 |
| 10 | 49.760 |
| 11 | 52.340 |
| 12 | 50.420 |
| 13 | 51.963 |
| 14 | 51.647 |
| 15 | 50.492 |
| 16 | 62.645 |
| 17 | 63.157 |
| 18 | 61.851 |
| 19 | 59.751 |
| 20 | 52.280 |
| 21 | 51.702 |
| 22 | 51.984 |
| 23 | 57.967 |
| 24 | 54.080 |
| 25 | 50.849 |
| 26 | 62.249 |
| 27 | 60.986 |
| 28 | 60.735 |
| 29 | 50.509 |
| 30 | 51.017 |
| 31 | 63.164 |
| 32 | 63.,203 |
| 33 | 50.146 |
| 34 | 60.720 |
| 35 | 51.252 |
| 36 | 62.435 |
| 37 | 101.230 |
| 38 | 77.561 |
| 39 | 80.485 |
| 40 | 78.080 |
| 41 | 76.401 |
| 42 | 103.061 |
| 43 | 108.794 |
| 44 | 93.443 |
| 45 | 73.156 |
| 46 | 77.282 |
| 47 | 78.462 |
| 48 | 78.295 |
| 49 | 74.825 |
| 50 | 96.034 |
| Average time | 67.62828571 |

A graph of a graph

Description automatically generated

**\*Input text is - hello kitty is not a cat\* (21 Bytes)**

Observation: Scrypt, which is intended to be computationally costly in order to deter brute-force attacks, is noticeably slower. Its robustness against specific hardware attacks is proven by its significant processing time.

# Bcrypt Hash

The password hashing method Bcrypt was created to be slow and impervious to brute-force attacks. It guarantees that cracking a password hash becomes computationally costly by using salting and a customizable effort factor. Since its creation, Bcrypt has been the industry standard for safe password storing (Neill, 2011).

|  |  |
| --- | --- |
| Run | Bcrypt Hash |
| 1 | 0.238 |
| 2 | 0.203 |
| 3 | 0.241 |
| 4 | 0.504 |
| 5 | 0.779 |
| 6 | 0.193 |
| 7 | 0.241 |
| 8 | 0.191 |
| 9 | 0.275 |
| 10 | 0.186 |
| 11 | 0.198 |
| 12 | 0.192 |
| 13 | 0.262 |
| 14 | 0.195 |
| 15 | 0.202 |
| 16 | 0.243 |
| 17 | 0.240 |
| 18 | 0.242 |
| 19 | 0.248 |
| 20 | 0.193 |
| 21 | 0.190 |
| 22 | 0.193 |
| 23 | 0.186 |
| 24 | 0.193 |
| 25 | 0.200 |
| 26 | 0.249 |
| 27 | 0.224 |
| 28 | 0.244 |
| 29 | 0.190 |
| 30 | 0.190 |
| 31 | 0.251 |
| 32 | 0.257 |
| 33 | 0.190 |
| 34 | 0.237 |
| 35 | 0.198 |
| 36 | 0.245 |
| 37 | 0.284 |
| 38 | 0.284 |
| 39 | 0.424 |
| 40 | 0.291 |
| 41 | 0.319 |
| 42 | 0.278 |
| 43 | 0.439 |
| 44 | 0.423 |
| 45 | 0.293 |
| 46 | 0.272 |
| 47 | 0.339 |
| 48 | 0.278 |
| 49 | 0.277 |
| 50 | 0.274 |
| Average time | 0.26356 |

**A graph of a graph

Description automatically generated**

**\*Input text is - hello kitty is not a cat\* (21 Bytes)**

Observation: Bcrypt, which is significantly quicker than Scrypt but slower than Argon2 and SHA-256, also offers an acceptable compromise between security and speed.

# Comparison of average time spent by each hashing algorithm

A graph with numbers and a line

Description automatically generated

Bcrypt Hash (0.26356 ms): Bcrypt offers a considerable degree of protection through computational complexity, however it is slower than Argon2. Although it performs well for general use, Argon2 is more efficient.

Scrypt Hash (67.63 ms): Because of its memory-intensive architecture, Scrypt takes the longest by a considerable amount. It is very secure yet computationally expensive because it is designed to withstand hardware brute-force attacks.

Argon2 Hash (0.1924 ms): Argon2 offers strong security yet is considerably quicker than Scrypt. Nowadays, it's considered one of the most efficient and secure memory-hard functions, achieving a balance between security and speed.  
  
SHA-256 (0.00012 ms): SHA-256 is a straightforward cryptographic hash function that lacks intrinsic memory hardness, resulting in it much faster than the others. It is less appropriate for password hashing because of its speed, but that comes at the expense of weaker resilience against brute-force attacks.  
  
In conclusion, Argon2 delivers the best security and efficiency balance for hashing functions where security is crucial (such as password storing). While Bcrypt is still a good, if slower, choice, Scrypt offers robust protection at a high computational cost. Although SHA-256 is the fastest, its susceptibility to brute-force attacks renders it inappropriate for password hashing.

# Final verdict

Effect of Key Size: Increasing the key size often results in significantly longer processing speeds for encryption methods such as AES. However, the impact is not tremendous, especially in comparison to the considerably slower hashing algorithms.  
  
For hashing, Scrypt, Bcrypt, and Argon2 are renowned password hashing algorithms, each with unique performance traits and security features. Scrypt is resistant to ASIC assaults since it is slow and memory hard. Bcrypt is less memory-intensive yet offers scalable security through a cost factor. The most adaptable and contemporary choice, Argon2, strikes a compromise between memory hardness and adjustable settings, making it appropriate for high-security applications. Each meets various standards by balancing performance and security according to certain use cases.

For encryption, the popular encryption algorithms AES (GCM), AES (CTR), AES (CFB), ChaCha20-Poly1305, and Fernet each have special features. High-performance applications can benefit from the efficient encryption and integrity checks that AES in Galois/Counter Mode (GCM) offers. Though it lacks integrated integrity checks, AES in Counter Mode (CTR) is quick and enables parallel processing. Because it depends on earlier ciphertext blocks, AES in Cipher Feedback Mode (CFB) is slower yet perfect for streaming data. ChaCha20-Poly1305 offers unparalleled security and efficiency, especially on limited devices, by combining the speed of ChaCha20 encryption with Poly1305 authentication. For developers looking for ease of use, Fernet offers a more straightforward API for secure encryption with integrated key management and integrity checks. Every algorithm balance security, performance, and accessibility to meet certain objectives.

Choosing the appropriate hashing and encryption algorithms is essential for protecting sensitive data in the constantly changing field of cybersecurity. In regard to password hashing, Scrypt, Bcrypt, and Argon2 each have unique advantages. Their performance and security levels vary to suit various purposes. Similar to this, AES (GCM), AES (CTR), AES (CFB), ChaCha20-Poly1305, and Fernet offer strong encryption choices that meet various requirements according to efficiency, integrity, and simplicity of use. Developers and security experts can improve the security posture of their systems by understanding the advantages and disadvantages of these algorithms. This will ensure that data is protected against evolving threats while establishing an equilibrium between usability and performance standards.

## References

National Institute of Standards and Technology. (2001). *Recommendation for block cipher modes of operation: CFB mode* (NIST Special Publication 800-38A). <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf>

National Institute of Standards and Technology. (2001). *Recommendation for block cipher modes of operation: CTR mode* (NIST Special Publication 800-38C). <https://csrc.nist.gov/publications/detail/sp/800-38c/final>

National Institute of Standards and Technology. (2007). *Recommendation for block cipher modes of operation: GCM and GMAC* (NIST Special Publication 800-38D). <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>

Internet Engineering Task Force. (2018). *ChaCha20 and Poly1305 for IETF protocols* (RFC 8439). <https://tools.ietf.org/html/rfc8439>

Python Cryptography Developers. (n.d.). *Fernet* [Documentation]. <https://cryptography.io/en/latest/fernet/>

National Institute of Standards and Technology. (2012). *Secure hash standard* (FIPS PUB 180-4). <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>

Percival, C. (2009). Stronger key derivation via sequential memory-hard functions. <https://www.tarsnap.com/scrypt/scrypt.pdf>

Biryukov, A., Dinu, L., & Khovratovich, D. (2016). Argon2: The memory-hard function for password hashing. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 255-272). <https://doi.org/10.1145/2976749.2978428>

Neill, A. (2011). Bcrypt: A password hashing function. *Linux Journal*, 2011(202), 4. <https://www.linuxjournal.com/content/bcrypt-password-hashing-function>