# Cryptography and System Security
# IE Report
# on
# SHA-3, RIPEMD family of algorithms, BLAKE2/BLAKE2s/ BLAKE2b

## Prepared By:

**TE_COMP_C_11_** Sumaiya Shaikh : RIPEMD
**TE_COMP_C_12_** Abhishek Sharma : RIPEMD
**TE_COMP_C_13_** Ashutosh Sharma : RIPEMD
**TE_COMP_C_14_** Jayesh Sharma : BLAKE2/BLAKE2s/ BLAKE2b
**TE_COMP_C_15_** Sakshi Sharma : BLAKE2/BLAKE2s/ BLAKE2b
**TE_COMP_C_16_** Vinit Sharma : BLAKE2/BLAKE2s/ BLAKE2b
**TE_COMP_C_17_** Ananya Shetty : SHA-3
**TE_COMP_C_18_** Karthik Shetty : SHA-3
**TE_COMP_C_19_** Sparsha Shetty : SHA-3
**TE_COMP_C_20_** Pranali Shirsat: SHA-3

# Cryptography and System Security IE

# Topic: SHA-3 Hash Function

## Introduction:

The SHA-3 (Secure Hash Algorithm 3) is a cryptographic hash function that was developed as part of the NIST (National Institute of Standards and Technology) hash function competition in 2012. It was designed as a next-generation hash function to replace the widely-used SHA-2 hash function, which has been in use since 2001. The competition was held to address concerns about the security and longevity of hash functions, and to encourage the development of new and more secure hash functions.
The SHA-3 hash function is based on the sponge construction, which is a type of hash function construction that provides better resistance to certain types of attacks. The sponge construction allows SHA-3 to process messages of any length, with a variable output length. This means that it is suitable for a wide range of applications, from digital signatures to password storage.

## SHA-3:

The SHA-3 hash function uses a fixed-length message block of 1600 bits, and is based on a permutation function called Keccak. The Keccak permutation function is a bit-oriented design that uses a round function to transform the state of the hash function. The number of rounds used by SHA-3 varies depending on the output size of the hash function, with larger output sizes requiring more rounds. For example, a 512-bit output size requires 24 rounds, while a 256-bit output size requires 14 rounds.
To ensure that the input message is properly processed by the hash function, SHA-3 uses message padding. Message padding is the process of adding extra bits to the message to ensure that its length is a multiple of the block size. In SHA-3, the message is padded with a 1 bit followed by zero or more 0 bits, and then a 10* pattern is added to the end of the message.
The output of SHA-3 is a fixed-length hash value that is unique to the input message. The length of the output hash can be any value between 224 and 512 bits, in increments of 8 bits. This makes SHA-3 more flexible than its predecessor, SHA-2, which has a fixed output size of 224, 256, 384, or 512 bits.
The sponge construction used by SHA-3 is different from the Merkle-Damgård construction used by SHA-2, which uses a chaining mechanism to process messages of fixed length. The sponge construction provides better resistance to certain types of attacks, such as length extension attacks, because the output hash is only generated after the entire message has been absorbed. This means that an attacker cannot simply append data to the end of the message and generate a new hash without knowing the entire original message.

In terms of performance, SHA-3 is faster than SHA-2 for larger message sizes, but slower for smaller message sizes. SHA-3 can also be optimized for different platforms and applications, such as software implementations or hardware implementations. The performance of SHA-3 can be impacted by several factors, including the message size, the number of rounds used, and the implementation of the permutation function.

## Algorithm:

The SHA-3 hash function algorithm is based on the Keccak sponge construction, which consists of the following steps:

- Padding: The input message is padded with a binary 1 followed by as many 0s as necessary to ensure that the message length is a multiple of the block size (in bits). The last block is also padded with a binary 1 and a fixed-length binary representation of the original message length.
- Absorbing: The padded message is divided into blocks and absorbed by the sponge function. Each block is XORed with the current state of the sponge, and then the sponge is updated using a permutation function.
- Squeezing: The output hash is obtained by squeezing the sponge. The same permutation function is used to generate an arbitrary number of output blocks, which are concatenated to produce the final hash value.

The SHA-3 hash function uses a variable-length output, with options for output sizes of 224, 256, 384, and 512 bits. It also uses a variable number of rounds, depending on the security level required. The number of rounds is determined by the block size and is defined by the capacity c = 2n, where n is the security level (i.e., the number of bits in the output). The number of rounds is then given by 12 + 2L, where L = log2(c/25). For example, if the security level is 256 bits, then the block size is 1600 bits, c = 2^8, and L = 6. Therefore, the number of rounds is 24.

## Working:

Input: "The quick brown fox jumps over the lazy dog" Initialization: SHA3-512 Output: the resulting hash value in hexadecimal format

The SHA-3 hash function is based on the sponge construction, which is a flexible framework for designing cryptographic hash functions. The sponge construction consists of two main phases: the absorbing phase and the squeezing phase. Let's see how the SHA-3 hash function works using the input, initialization, and output parameters provided.

- Padding: The first step in the SHA-3 hash function is to pad the input message. In this case, the input is "The quick brown fox jumps over the lazy dog", which is a 43-byte message or 344 bits. We need to add padding to make the message length a multiple of the block size, which is 576 bits for SHA3-512. We start by appending a binary 1 to the message, followed by as many 0s as necessary to reach the block size. In this case, we need to add 576 - 344 -

1 = 231 bits of padding, so we add 231 0s to the message. The resulting padded message is:

```
01010100 01101000 01100101 00100000 01110001 01110101 01101001 01100011 01101011
00100000 01100010 01110010 01101111 01110111 01101110 00100000 01100110 01101111
01111000 00100000 01101010 01110101 01101101 01110000 01110011 00100000 01101111
01110110 01100101 01110010 00100000 01110100 01101000 01100101 00100000 01101100
01100001 01111010 01111001 00100000 01100100 01101111 01100111 00101100 00000001
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
...
 00000000 0000000 00000000 00000000 00000000 00000000 00000000 00000000 00001000
```

- Absorbing: The next step is to absorb the padded message into the sponge. The sponge consists of a state array of size (b+c), where b is the block size and c is the capacity. In this case, b = 576 bits and c = 1024 bits, so the state array has a size of 1600 bits (25 words of 64 bits each). The sponge also uses a permutation function f(), which operates on the state array and updates it in a nonlinear and reversible way. The absorbing phase consists of dividing the padded message into blocks of size b, XORing each block with the state array, and then applying the permutation function f() to update the state array. This process is repeated until the entire message has been absorbed. In this case, there is only one block of size 576 bits, which is XORed with the initial state array of all 0s. The permutation function is applied 24 times to update the state array.
- Squeezing: The final step is to squeeze the sponge to obtain the output hash. The squeezing phase consists of generating blocks of output by applying the permutation function f() to the state array and XORing the resulting block with the output buffer. Each output block has a size of b, which is 576 bits for SHA3-512. This process is repeated until the desired output length is reached.

To better understand how the SHA-3 hash function works, let's consider an example using the following parameters:
Input: "Hello world" Initialization: The state array is initialized with a fixed value depending on the chosen hash function, in this case, SHA3-256. The state array is then XORed with the input message block.
Message block processing: The input message is split into blocks of 1088 bits (or 136 bytes). If the last block is less than 1088 bits, it is padded with bits such that the resulting block is exactly 1088 bits. The message block is then XORed with the state array, and the permutation function f() is applied to the resulting state array. This process is repeated for each message block until the entire message has been processed.
Output: The output hash is obtained by squeezing the sponge. The output buffer is initialized with a fixed value, and the permutation function () is applied to the state array. The resulting block is XORed with the output buffer to produce the output hash. This process is repeated until the desired output length is reached.

For the input "Hello world" and SHA3-256,
the output hash would be: 5d41402abc4b2a76b9719d911017c592945 ecd1.

## Applications:

1. Password storage: SHA-3 can be used to store passwords securely. When a user creates a password, it is first hashed using SHA-3 and the resulting hash is stored in the database. When the user enters their password, it is hashed again using SHA-3 and compared to the stored hash to verify the password.
2. Digital signatures: SHA-3 can be used in digital signature schemes to provide message authentication. When a message is signed using a private key, the hash of the message is computed using SHA-3 and the resulting hash is signed using the private key. The recipient can then verify the signature by computing the hash of the message using SHA-3 and verifying the signature using the public key.
3. Blockchain: SHA-3 is used in many blockchain implementations as the hashing algorithm to ensure the integrity of the data stored in the blocks. Each block in the blockchain contains a hash of the previous block, ensuring that any tampering with the data in one block will be detected.
4. Data integrity: SHA-3 can be used to ensure the integrity of data being transmitted over a network or stored in a database. A hash of the data is computed using SHA-3 and sent along with the data. The recipient can then verify the integrity of the data by computing the hash of the data using SHA-3 and comparing it to the hash sent with the data.
5. Cryptographic key generation: SHA-3 can also be used in key generation algorithms to produce random keys for encryption and decryption. Since SHA-3 produces a secure and random hash, it can be used to generate cryptographic keys that are difficult to guess.
6. Secure messaging: SHA-3 can be used in secure messaging protocols such as Secure Sockets Layer (SSL) and Transport Layer Security (TLS) to ensure the integrity of the messages being sent. Each message is hashed using SHA-3 before being sent, and the recipient verifies the hash to ensure that the message has not been tampered with.
7. Digital forensics: SHA-3 can be used in digital forensics to detect the presence of specific files or data on a storage device. By computing the SHA-3 hash of a file or data, investigators can compare it against a database of known hashes to determine whether the file or data has been previously identified as relevant to the investigation.
8. Software updates: SHA-3 can be used to ensure the authenticity and integrity of software updates. By hashing the software update using SHA-3 and distributing the resulting hash along with the update, users can verify that the update has not been tampered with or modified during distribution.
9. Intrusion detection: SHA-3 can be used in intrusion detection systems to detect unauthorized changes to system files or configuration files. By computing the SHA-3

hash of critical system files and periodically checking them against the stored hash, an intrusion detection system can detect changes that may indicate a security breach.

10. Password less authentication: SHA-3 can be used in password less authentication systems, where a user's identity is verified by computing a hash of a unique identifier, such as a biometric characteristic or a hardware token. By comparing the resulting hash to a stored value, the system can authenticate the user without requiring a password.

## Conclusion:

In conclusion, the SHA-3 hash function is a next-generation cryptographic hash function designed to be more secure and flexible than its predecessor, SHA-2. It uses the sponge construction to process messages of any length, with a variable output length, and is based on the Keccak permutation function. The sponge construction provides better resistance to certain types of attacks, such as length extension attacks, and SHA-3 can be optimized for different platforms and applications. The algorithm consists of three main steps: padding, absorbing, and squeezing. The number of rounds used by SHA-3 varies depending on the output size of the hash function. SHA-3 is faster than SHA-2 for larger message sizes but slower for smaller message sizes. Its performance can be optimized for different platforms and applications. Overall, SHA-3 is a significant improvement over SHA-2 and is expected to be widely adopted in the coming years.

## References:

- https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
- https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
- https://keccak.team/files/Keccak-submission-3.pdf
- https://en.wikipedia.org/wiki/SHA-3