**ABSTRACT**

Cryptographic hash functions for calculating the message digest of a message has been in practical use as an effective measure to maintain message integrity since a few decades. This message digest is unique, irreversible and avoids all types of collisions for any given input string. The message digest calculated from this algorithm is propagated in the communication medium along with the original message from the sender side and on the receiver side integrity of the message can be verified by recalculating the message digest of the received message and comparing the two digest values. In this paper we have designed and developed a new algorithm for calculating the message digest of any message and implemented it using a high level programming language. An experimental analysis and comparison with the existing MD5 hashing algorithm, which is predominantly being used as a cryptographic hashing tool, shows this algorithm to provide more randomness and greater strength from intrusion attacks. In this algorithm the plaintext message string is converted into binary string and fragmented into blocks of 128 bits after being padded with user defined padding bits. Then using a pseudo random number generator a key is generated for each block and operated with the respective block by a bitwise operator. This process is iterated for the whole message and finally a fixed length message digest is obtained. In the past few years, there have been significant research advances in the analysis of hash functions and it was shown that none of the hash algorithm is secure enough for critical purposes whether it is MD5 or SHA-1.Nowadays scientists have found weaknesses in a number of hash functions, including MD5, SHA and RIPEMD so the purpose of this paper is combination of some function to reinforce these functions and also increasing hash code length up to 512 that makes stronger algorithm against collision attest.

**CONTENTS**

**SL.NO. TITLE. PAGE NO**

**1. Introduction 3**

**2. Security Algorithm 4**

**3. Authentication Algorithm 7**

**4. Hash function 9**

**5. MD5 Algorithm 14**

**6. Salts 20**

**7. Conclusion 20**

**INTRODUCTION**

A security algorithm is a mathematical procedure used to encrypt data or verify a data. It is used in almost all fields or computing, networking and communication. A main feature used in security algorithms is cryptography.

Cryptography or cryptology is the practice and study of techniques for secure communication in the presence of third parties called adversaries. More generally, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages; various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation are central to modern cryptography. Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, electrical engineering, communication science, and physics. Applications of cryptography include electronic commerce, chip-based payment cards, digital currencies, computer passwords, and military communications.

Until modern times, cryptography referred almost exclusively to encryption, which is the process of converting ordinary information (called plaintext) into unintelligible text (called cipher text). Decryption is the reverse, in other words, moving from the unintelligible ciphertext back to plaintext. A cipher (or cypher) is a pair of algorithms that create the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a "key". The key is a secret (ideally known only to the communicants), usually a short string of characters, which is needed to decrypt the ciphertext. Formally, a "cryptosystem" is the ordered list of elements of finite possible plaintexts, finite possible cyphertexts, finite possible keys, and the encryption and decryption algorithms which correspond to each key. Keys are important both formally and in actual practice, as ciphers without variable keys can be trivially broken with only the knowledge of the cipher used and are therefore useless (or even counter-productive) for most purposes.

Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in practice by any adversary. It is theoretically possible to break such a system, but it is infeasible to do so by any known practical means. These schemes are therefore termed computationally secure; theoretical advances, e.g., improvements in integer factorization algorithms, and faster computing technology require these solutions to be continually adapted. There exist information-theoretically secure schemes that probably cannot be broken even with unlimited computing power—an example is the one-time pad—but these schemes are more difficult to implement than the best theoretically breakable but computationally secure mechanisms.

**Chapter 1**

**SECURITY ALGORITHMS**

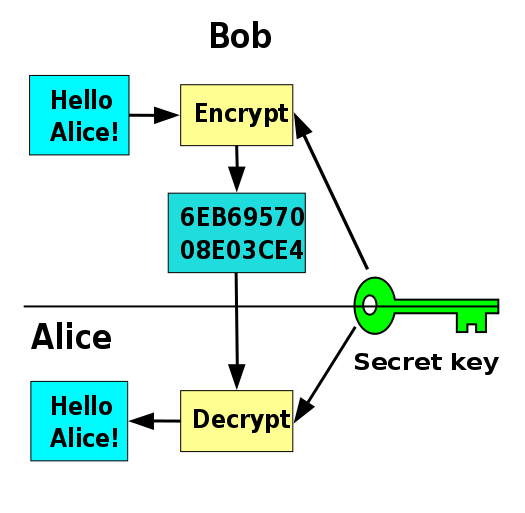
Security is a concerning feature in the modern computing field. Security of the data transmitted is a very important factor as the computing field is growing day by day. Before the modern era, cryptography focused on message confidentiality (i.e., encryption)—conversion of messages from a comprehensible form into an incomprehensible one and back again at the other end, rendering it unreadable by interceptors or eavesdroppers without secret knowledge (namely the key needed for decryption of that message). Encryption attempted to ensure secrecy in communications, such as those of spies, military leaders, and diplomats. In recent decades, the field has expanded beyond confidentiality concerns to include techniques for message integrity checking, sender/receiver identity authentication, digital signatures, interactive proofs and secure computation, among others.

he modern field of cryptography can be divided into several areas of study.

1. **Symmetric-key cryptography**

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key (or, less commonly, in which their keys are different, but related in an easily computable way). This was the only kind of encryption publicly known until June 1976.

Symmetric key ciphers are implemented as either block ciphers or stream ciphers. A block cipher enciphers input in blocks of plaintext as opposed to individual characters, the input form used by a stream cipher. The below figure (Fig 1.1) shows an illustration of a typical symmetric-key cryptography.



**Fig 1.1**: Symmetric-key cryptography, where a single key is used for encryption and decryption.

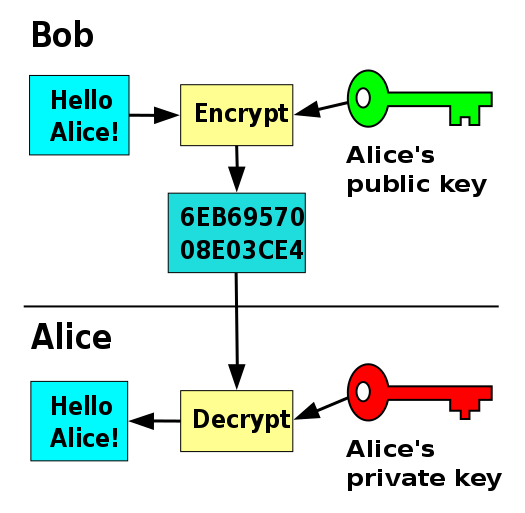
The Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are block cipher designs that have been designated cryptography standards by the US government (though DES's designation was finally withdrawn after the AES was adopted).Despite its deprecation as an official standard, DES (especially its still-approved and much more secure triple-DES variant) remains quite popular; it is used across a wide range of applications, from ATM encryption to e-mail privacy and secure remote access. Many other block ciphers have been designed and released, with considerable variation in quality. Many, even some designed by capable practitioners, have been thoroughly broken, such as FEAL.

Stream ciphers, in contrast to the 'block' type, create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character-by-character, somewhat like the one-time pad. In a stream cipher, the output stream is created based on a hidden internal state that changes as the cipher operates. That internal state is initially set up using the secret key material. RC4 is a widely used stream cipher.

Message authentication codes (MACs) are much like cryptographic hash functions, except that a secret key can be used to authenticate the hash value upon receipt; this additional complication blocks an attack scheme against bare digest algorithms, and so has been thought worth the effort.

1. **Public-key cryptography**

Symmetric-key cryptosystems use the same key for encryption and decryption of a message, although a message or group of messages can have a different key than others. A significant disadvantage of symmetric ciphers is the key management necessary to use them securely. Each distinct pair of communicating parties must, ideally, share a different key, and perhaps for each ciphertext exchanged as well. The number of keys required increases as the square of the number of network members, which very quickly requires complex key management schemes to keep them all consistent and secret. The difficulty of securely establishing a secret key between two communicating parties, when a secure channel does not already exist between them, also presents a chicken-and-egg problem which is a considerable practical obstacle for cryptography users in the real world. The below figure (Fig 1.2) shows a typical Public-key Cryptography technique.



**Fig 1.2**: Public-key cryptography, where different keys are used for encryption and decryption.

In public-key cryptosystems, the public key may be freely distributed, while its paired private key must remain secret. In a public-key encryption system, the public key is used for encryption, while the private or secret key is used for decryption.

Public-key cryptography can also be used for implementing digital signature schemes. A digital signature is reminiscent of an ordinary signature; they both have the characteristic of being easy for a user to produce, but difficult for anyone else to forge. Digital signatures can also be permanently tied to the content of the message being signed; they cannot then be 'moved' from one document to another, for any attempt will be detectable. In digital signature schemes, there are two algorithms: one for signing, in which a secret key is used to process the message (or a hash of the message, or both), and one for verification, in which the matching public key is used with the message to check the validity of the signature. RSA and DSA are two of the most popular digital signature schemes. Digital signatures are central to the operation of public key infrastructures and many network security schemes (e.g., SSL/TLS, many VPNs, etc.).

Public-key algorithms are most often based on the computational complexity of "hard" problems, often from number theory. For example, the hardness of RSA is related to the integer factorization problem, while Diffie–Hellman and DSA are related to the discrete logarithm problem. More recently, elliptic curve cryptography has developed, a system in which security is based on number theoretic problems involving elliptic curves. Because of the difficulty of the underlying problems, most public-key algorithms involve operations such as modular multiplication and exponentiation, which are much more computationally expensive than the techniques used in most block ciphers, especially with typical key sizes. As a result, public-key cryptosystems are commonly hybrid cryptosystems, in which a fast high-quality symmetric-key encryption algorithm is used for the message itself, while the relevant symmetric key is sent with the message, but encrypted using a public-key algorithm. Similarly, hybrid signature schemes are often used, in which a cryptographic hash function is computed, and only the resulting hash is digitally signed.

**Chapter 2**

**AUTHENTICATION ALGORITHMS**

Authentication is the process of verifying the identity of the sender. Authentication algorithms use a shared key to verify the authenticity of the IPsec devices. Most routing protocols include three different types of authentication schemes: Null authentication, clear text password, and cryptographic authentication. Null authentication is equivalent to having no authentication scheme at all. In a clear text scheme, also known as a "simple password" scheme, the password is exchanged completely unprotected, and anyone with physical access to the network can learn the password and compromise the integrity of the routing domain. The simple password scheme protects against accidental establishment of routing sessions in a given domain, but beyond that it offers no additional protection. In a cryptographic authentication scheme, routers share a secret key that is used to generate a message authentication code for each of the protocol packets. Today, routing protocols that implement message authentication codes often use a Keyed-MD5 [RFC1321] digest.

**Different types of authentication algorithms**

1. **Message encryption algorithms**

Encryption, by itself, can protect the confidentiality of messages, but other techniques are still needed to protect the integrity and authenticity of a message; for example, verification of a message authentication code (MAC) or a digital signature. Standards for cryptographic software and hardware to perform encryption are widely available, but successfully using encryption to ensure security may be a challenging problem. A single error in system design or execution can allow successful attacks. Sometimes an adversary can obtain unencrypted information without directly undoing the encryption. See, e.g., traffic analysis, TEMPEST, or Trojan horse.

Digital signature and encryption must be applied to the ciphertext when it is created (typically on the same device used to compose the message) to avoid tampering; otherwise any node between the sender and the encryption agent could potentially tamper with it. Encrypting at the time of creation is only secure if the encryption device itself has not been tampered with.

1. **Message address code(MAC)**

In cryptography, a message authentication code (MAC), sometimes known as a tag, is a short piece of information used to authenticate a message—in other words, to confirm that the message came from the stated sender (its authenticity) and has not been changed. The MAC value protects both a message's data integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content.

Informally, a message authentication code consists of three algorithms:

* A key generation algorithm selects a key from the key space uniformly at random.
* A signing algorithm efficiently returns a tag given the key and the message.
* A verifying algorithm efficiently verifies the authenticity of the message given the key and the tag. That is, return accepted when the message and tag are not tampered with or forged, and otherwise return rejected.

For a secure unforgeable message authentication code, it should be computationally infeasible to compute a valid tag of the given message without knowledge of the key, even if for the worst case, we assume the adversary can forge the tag of any message except the given one.

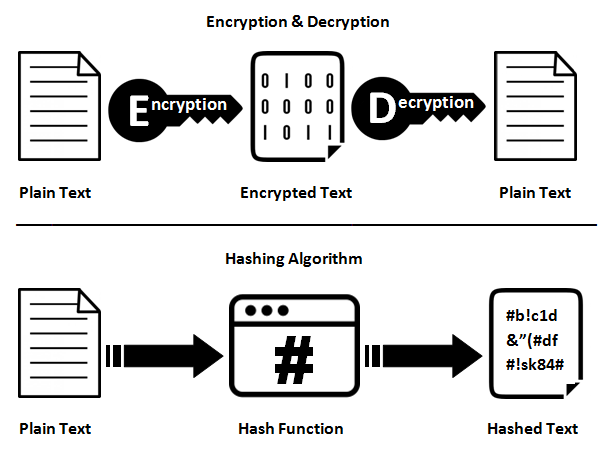
While MAC functions are similar to cryptographic hash functions, they possess different security requirements. To be considered secure, a MAC function must resist existential forgery under chosen-plaintext attacks. This means that even if an attacker has access to an oracle which possesses the secret key and generates MACs for messages of the attacker's choosing, the attacker cannot guess the MAC for other messages (which were not used to query the oracle) without performing infeasible amounts of computation.

MACs differ from digital signatures as MAC values are both generated and verified using the same secret key. This implies that the sender and receiver of a message must agree on the same key before initiating communications, as is the case with symmetric encryption. For the same reason, MACs do not provide the property of non-repudiation offered by signatures specifically in the case of a network-wide shared secret key: any user who can verify a MAC is also capable of generating MACs for other messages. In contrast, a digital signature is generated using the private key of a key pair, which is public-key cryptography. Since this private key is only accessible to its holder, a digital signature proves that a document was signed by none other than that holder. Thus, digital signatures do offer non-repudiation. However, non-repudiation can be provided by systems that securely bind key usage information to the MAC key; the same key is in the possession of two people, but one has a copy of the key that can be used for MAC generation while the other has a copy of the key in a hardware security module that only permits MAC verification. This is commonly done in the finance industry.

1. **Hash Function**

A hash function is any function that can be used to map data of arbitrary size to data of a fixed size. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes. Hash functions are often used in combination with a hash table, a common data structure used in computer software for rapid data lookup. Hash functions accelerate table or database lookup by detecting duplicated records in a large file. One such application is finding similar stretches in DNA sequences. They are also useful in cryptography.

A cryptographic hash function allows one to easily verify that some input data maps to a given hash value, but if the input data is unknown, it is deliberately difficult to reconstruct it (or any equivalent alternatives) by knowing the stored hash value. This is used for assuring integrity of transmitted data, and is the building block for HMACs, which provide message authentication. The below figure (Fig 2.1) shows the comparison between a hash function and an encryption algorithm.



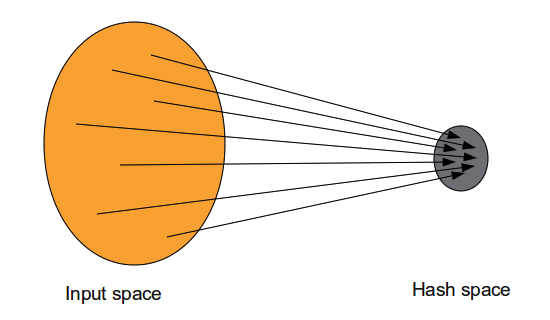
**Fig 2.1**: Comparison between a hash function and an encryption function.

Hash functions are related to (and often confused with) checksums, check digits, fingerprints, lossy compression, randomization functions, error-correcting codes, and ciphers. Although the concepts overlap to some extent, each one has its own uses and requirements and is designed and optimized differently. The HashKeeper database maintained by the American National Drug Intelligence Center, for instance, is more aptly described as a catalogue of file fingerprints than of hash values.

**Chapter 3**

**HASH FUNCTIONS**

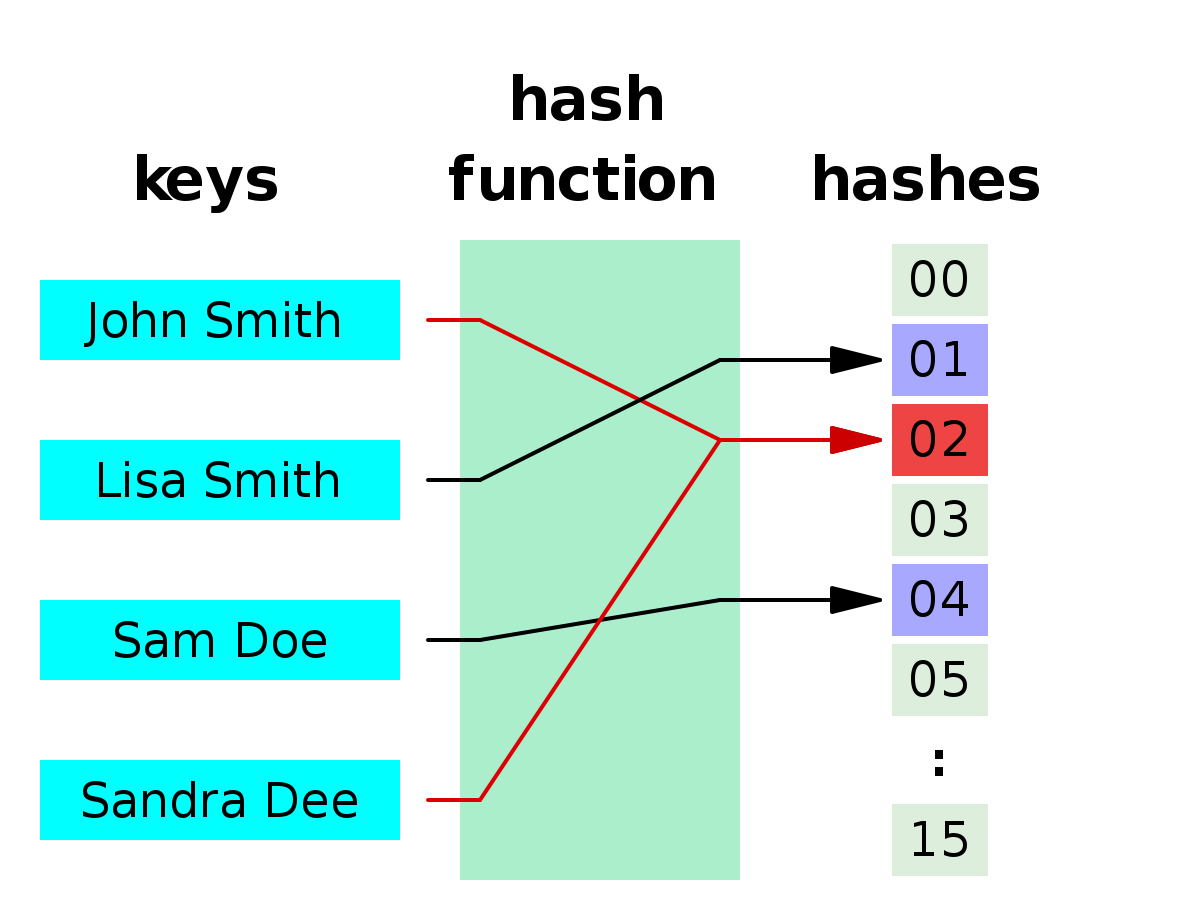
A hash function is a function which will convert a message into hash. A hash is a string or number generated from a string of text. The resulting string or number is a fixed length, and will vary widely with small variations in input. Hash functions are mainly used for comparison purposes. Hash functions are generally one way functions (Fig 3.1), i.e. a hash cannot be reversed to the plain text we gave as input in the starting to the hash function.



**Fig 3.1**: A one-way function.

A hash function is any function that can be used to map data of arbitrary size to data of a fixed size. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes. Hash functions are often used in combination with a hash table, a common data structure used in computer software for rapid data lookup.

Hash functions accelerate table or database lookup by detecting duplicated records in a large file. One such application is finding similar stretches in DNA sequences. They are also useful in cryptography. A cryptographic hash function allows one to easily verify that some input data maps to a given hash value, but if the input data is unknown, it is deliberately difficult to reconstruct it (or any equivalent alternatives) by knowing the stored hash value. This is used for assuring integrity of transmitted data, and is the building block for HMACs, which provide message authentication.



**Fig 3.2**: Hashing using a hash function.

**How passwords are saved in a database:**

Typically hash functions are used for saving of passwords in a database. The steps for creating and verifying a password is as follows:

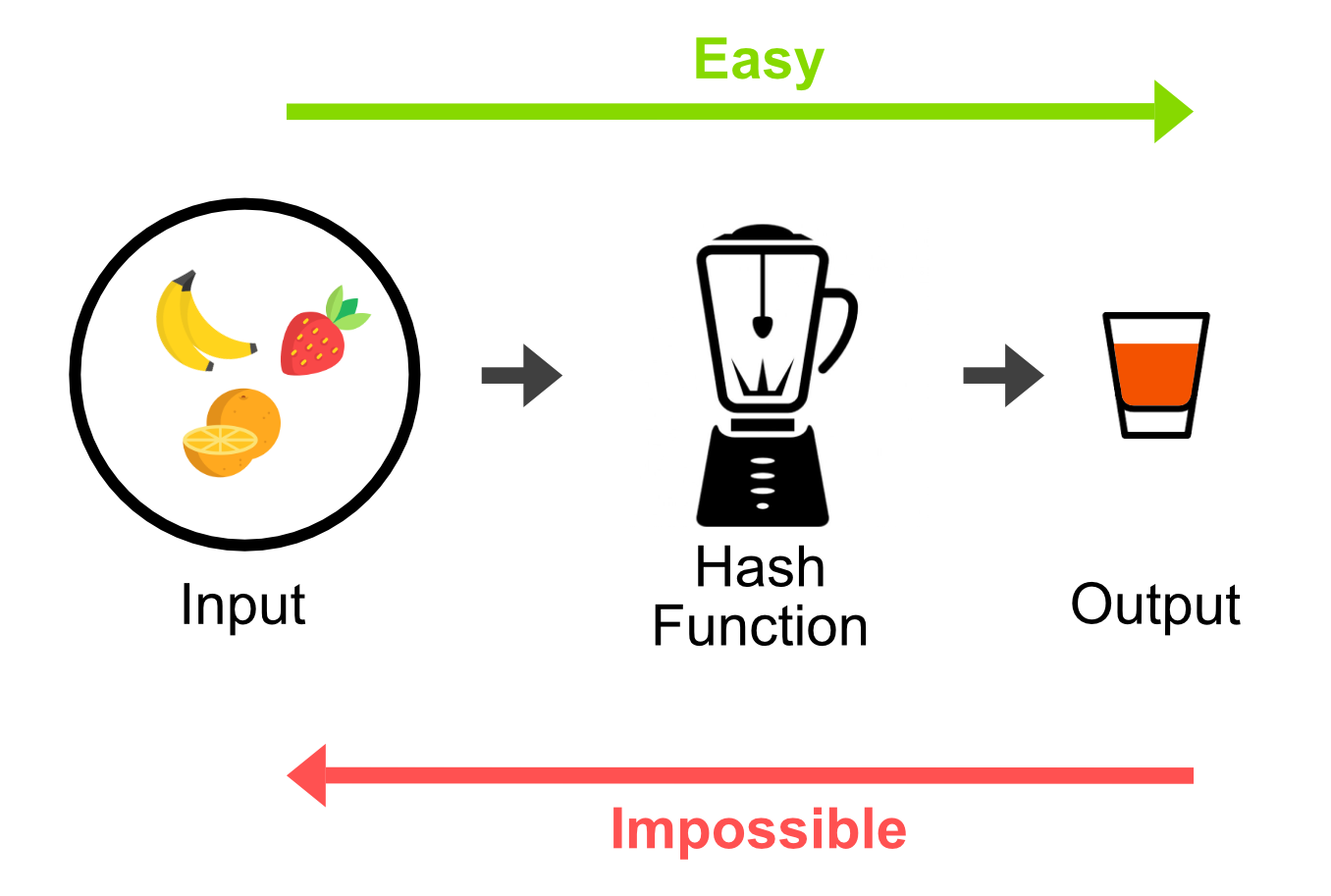
* User creates a UserID and password.
* The password is given to a hash function, which will return the hash of the password.
* The hash of the password is stored in the database of the website.
* When user tries to sign in into his/her account, the entered password is converted into hash and compared with the hash in database. If they match the user is authenticated.

Since hash functions are one way functions, even if an attacker gets access to the database of the site, they get only hashes of the user’s passwords which are of no use to them because they cannot reverse them.

**Characteristics of hash functions**

1. **Irreversible**

Hash functions are one way functions; it can only be mapped from one space to other not back. A one-way function is a function that is easy to compute on every input, but hard to invert given the image of a random input. Here, "easy" and "hard" are to be understood in the sense of computational complexity theory, specifically the theory of polynomial time problems. Not being one-to-one is not considered sufficient of a function for it to be called one-way.

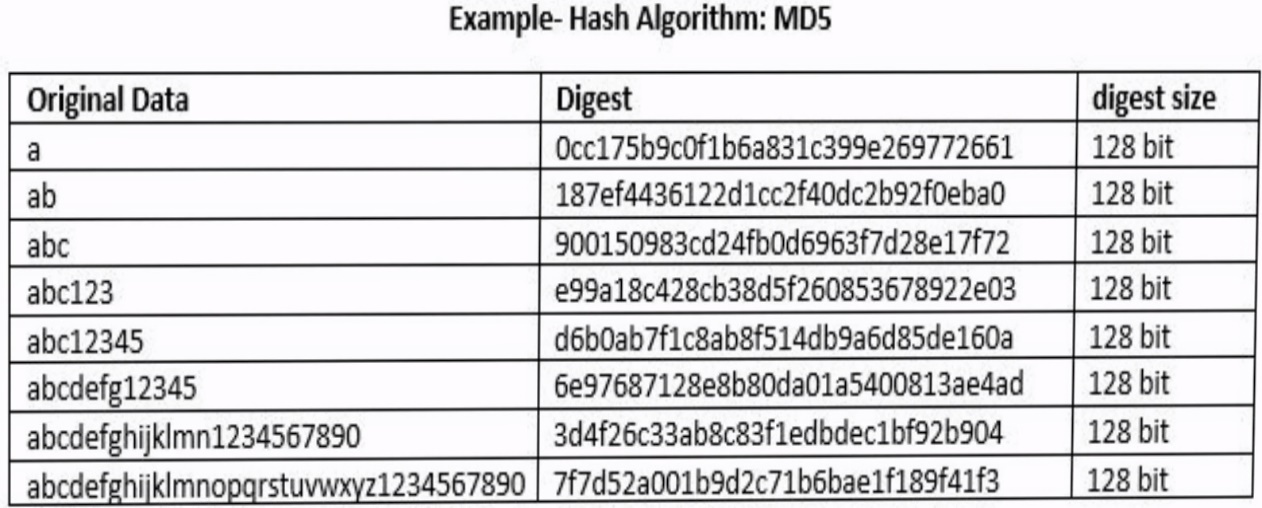


**Fig 3.3**: A real world example to one-way function.

A real world example of one-way function is shown in the figure above (Fig 3.3).

1. **Produces fixed size output**

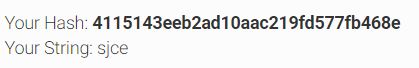
A hash function always produces a fixed sized output not being concerned about the data given. Whatever be the length of data, a hash function should be able to produce a fixed sized output. The below figure (Fig 3.4) shows different outputs for strings of different lengths using MD5 hashing algorithm. MD5 algorithm



**Fig 3.4:** Demonstration of fixed sized output for different inputs using md5 hashing.

1. **Unique**

Every hash is unique on its own. There won’t be same hash for different words (hash collision) or same word won’t be having different hashes. The below figures show the uniqueness of hashes. The first hash is generated using the string ‘sjce’ (Fig 3.5) and the second hash (Fig 3.6) is generated using the string ‘sjc e’. Both of the strings are hashed using MD5 hashing algorithm.



**Fig 3.5:** Hash of ‘sjce’ using MD5 hashing algorithm.

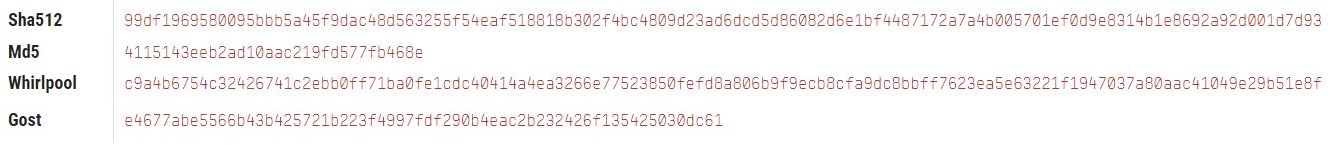


**Fig 3.6:** Hash of ‘sjc e’ using MD5 hashing algorithm.

**DIFFERENT HASHING ALGORITHMS**

There are enormous number of hashing algorithm’s available for hashing a string. Some of them are listed below. Hashing algorithms produce different sized output based on what algorithm and block size the algorithm uses. All the algorithms follow the characteristics of hashing algorithm and are used for real time hashing (Some of them were found to be raising hash collisions easily).

1. **SHA512-512bits**
2. **MD5-128bits**
3. **WHIRLPOOL-512bits**
4. **GHOST-256bits**



**Chapter 4**

**MD5 ALGORITHM**

MD5, with the full name of the Message-digest Algorithm 5, is the fifth generation on behalf of the message digest algorithm. In August 1992, Ronald L.Rivest submitted a document to the IETF (The Internet Engineering Task Force) entitled The MD5 Message-Digest Algorithm, which describes the theory of this algorithm. For the publicity and security of algorithm, it has been widely used to verify data integrity in a variety of program languages since the 1990s. MD5 was developed from MD, MD2, MD3 and MD4. It can compress any length of data into an information digest of 128bits while this segment message digest often claims to be a digital fingerprint of the data. This algorithm makes use of a series of non-linear algorithm to do the circular operation, so that crackers can not restore the original data. In cryptography, it is said that such algorithm as an irreversible algorithm, can effectively prevent data leakage caused by inverse operation. Both the theory and practice have good security, because the use of MD5 algorithm does not require the payment of any royalties, time, and cost less which make it be widely used in the general non-top-secret applications. But even the top-secret area, MD5 may well be an excellent intermediate technology. MD5 is an irreversible transformation transforming a set of data of any length into a hash value of 128-bit length and it is a consecutive processing method. Before operation, it first fills data to be processed, and adds 64-bit binary digits to the end of data representing the bit length of the original data. After filling, the bit length of data which is being processed becomes a multiple of 512. Then the data are divided into groups of 512 bits and computations are performed on each group orderly. The input of the first group operation is a 128-bit initial value; the input of the next group operation is a 128-bit output of the previous group operation. The 128-bit output of the last group operation is the MD5 hash value of the whole data. The key of MD5 algorithm is to perform 4 rounds of hash operation on the data packets of 512 bits. The processing logic is shown in Fig. 1

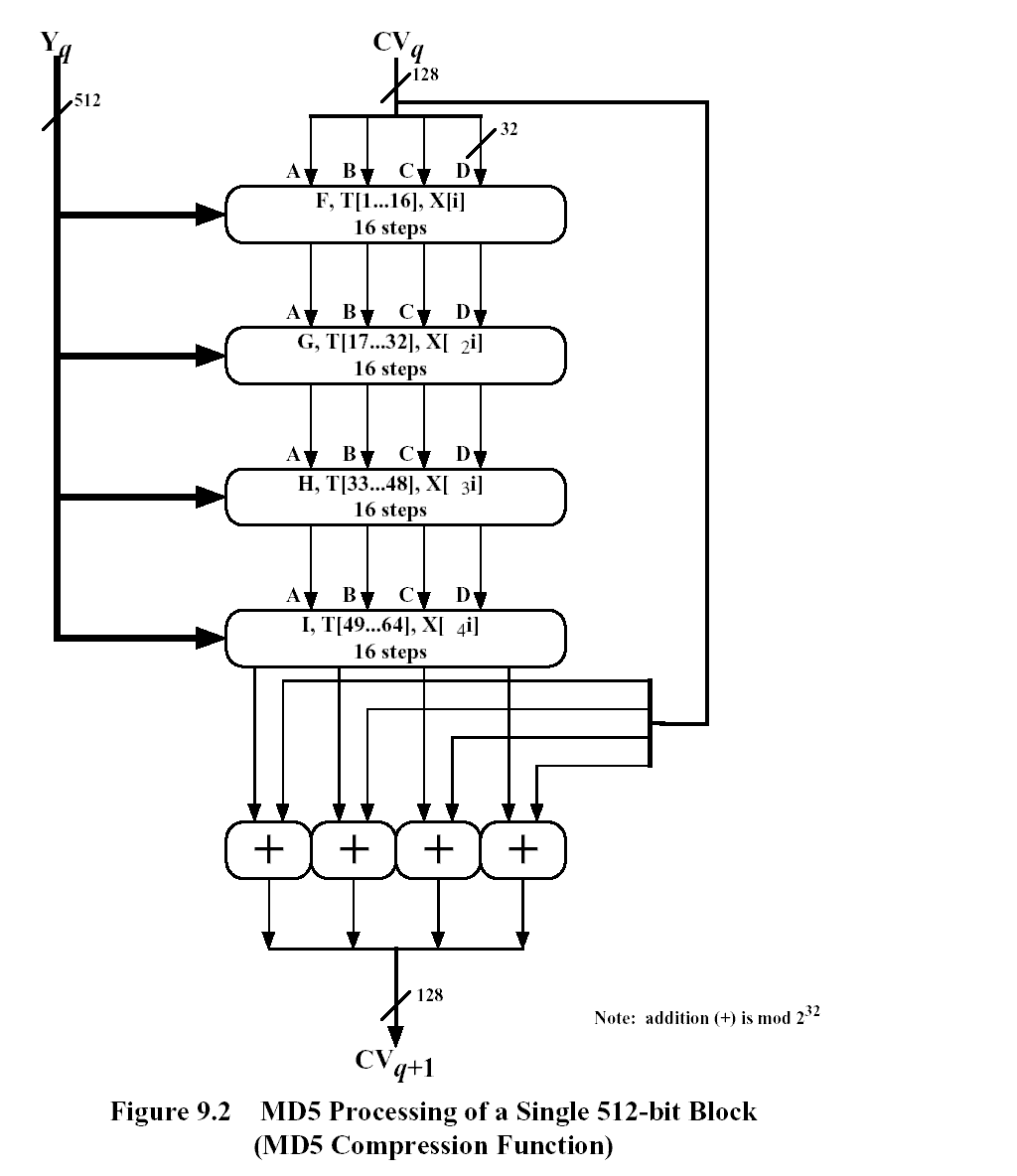


Figure 1: The processing logic

The 4-rounds of processing have similar structure but each of them has different logic function.

The function used in each round is as follows.

F(x,y,z) = (x & y) | ((~x) & z)

G(x,y,z) = (x & z) | (y & (~z))

H(x,y,z) = x ^ y ^ z

I(x,y,z) = y ^ (x | (~z)) Here, we only briefly introduce the operation structure of MD5. Detailed information on MD5 can be found in RFC1321. In high-density data computing environment, the application of MD5 can be divided into two categories. One is to perform hash operation on a large data block and get its hash value, such as the ones used in data integrality validation. In this application, the data block requiring MD5 operation is larger, perhaps even reaching one million bytes or ten million bytes. The other is to perform hash operation on many small data blocks and gets a hash value of each small data block. Such as in the application of character string detection, we should detect a character string in a very large data dictionary. in which input data must be divided into small pieces. In this application, each of the data blocks requiring MD5 operation is smaller, generally ranging from several bytes to hundreds of bytes. In order to distinguish the two categories, we call them large block hashing and large number hashing respectively.

**CODE**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <math.h>

typedef union uwb {

unsigned w;

unsigned char b[4];

} MD5union;

typedef unsigned DigestArray[4];

unsigned func0( unsigned abcd[] ){

return ( abcd[1] & abcd[2]) | (~abcd[1] & abcd[3]);} // Function F

unsigned func1( unsigned abcd[] ){

return ( abcd[3] & abcd[1]) | (~abcd[3] & abcd[2]);} //Function G

unsigned func2( unsigned abcd[] ){

return abcd[1] ^ abcd[2] ^ abcd[3];} //Function H

unsigned func3( unsigned abcd[] ){

return abcd[2] ^ (abcd[1] |~ abcd[3]);} //Function I

typedef unsigned (\*DgstFctn)(unsigned a[]);

/\*Table generated by the function below

k[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }

k[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }

k[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }

k[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }

k[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }

k[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }

k[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }

k[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }

k[32..35] := { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }

k[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }

k[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }

k[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }

k[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }

k[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }

k[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }

k[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }\*/

unsigned \*calctable( unsigned \*k) // Calculating the above values without hard coding( Seudo code from Wikipedia)

{

double s, pwr;

int i;

pwr = pow( 2, 32);

for (i=0; i<64; i++) {

s = fabs(sin(1+i));

k[i] = (unsigned)( s \* pwr );

}

return k;

}

/\*Rotate Left r by N bits

r[ 0..15] := {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22}

r[16..31] := {5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20}

r[32..47] := {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23}

r[48..63] := {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21}

\*/

unsigned rol( unsigned r, short N )

{

unsigned mask1 = (1<<N) -1;

return ((r>>(32-N)) & mask1) | ((r<<N) & ~mask1);

}

unsigned \*md5( const char \*msg, int mlen)

{

/\*Initialize Digest Array as A , B, C, D \*/

static DigestArray h0 = { 0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476 }; //Intial values

static DgstFctn ff[] = { &func0, &func1, &func2, &func3 };

static short M[] = { 1, 5, 3, 7 };

static short O[] = { 0, 1, 5, 0 };

static short rot0[] = { 7,12,17,22};

static short rot1[] = { 5, 9,14,20};

static short rot2[] = { 4,11,16,23};

static short rot3[] = { 6,10,15,21};

static short \*rots[] = {rot0, rot1, rot2, rot3 };

static unsigned kspace[64];

static unsigned \*k;

static DigestArray h;

DigestArray abcd;

DgstFctn fctn;

short m, o, g;

unsigned f;

short \*rotn;

union {

unsigned w[16];

char b[64];

}mm;

int os = 0;

int grp, grps, q, p;

unsigned char \*msg2;

if (k==NULL) k= calctable(kspace);

for (q=0; q<4; q++) h[q] = h0[q]; // initialize

{

grps = 1 + (mlen+8)/64;

msg2 = malloc( 64\*grps);

memcpy( msg2, msg, mlen);

msg2[mlen] = (unsigned char)0x80;

q = mlen + 1;

while (q < 64\*grps){ msg2[q] = 0; q++ ; }

{

MD5union u;

u.w = 8\*mlen;

q -= 8;

memcpy(msg2+q, &u.w, 4 );

}

}

for (grp=0; grp<grps; grp++)

{

memcpy( mm.b, msg2+os, 64);

for(q=0;q<4;q++) abcd[q] = h[q];

for (p = 0; p<4; p++) {

fctn = ff[p];

rotn = rots[p];

m = M[p]; o= O[p];

for (q=0; q<16; q++) {

g = (m\*q + o) % 16;

f = abcd[1] + rol( abcd[0]+ fctn(abcd) + k[q+16\*p] + mm.w[g], rotn[q%4]);

abcd[0] = abcd[3];

abcd[3] = abcd[2];

abcd[2] = abcd[1];

abcd[1] = f;

}

}

for (p=0; p<4; p++)

h[p] += abcd[p];

os += 64;

}

return h;

}

int main( int argc, char \*argv[] )

{

int j,k;

const char \*msg = "operating systems";

printf("Input String to be Encrypted using MD5 : %s",msg);

unsigned \*d = md5(msg, strlen(msg));

MD5union u;

printf("\n\nMD5 Encryption Successfully Completed!!\n\n");

printf("\n\nThe MD5 hash code for input string is :");

printf("0x");

for (j=0;j<4; j++) //4 Steps so the loop is iterated 4 times

{

u.w = d[j];

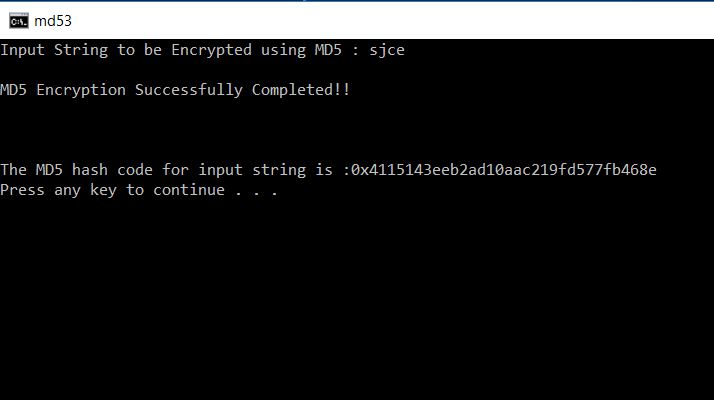
for (k=0;k<4;k++) //4 Buffers

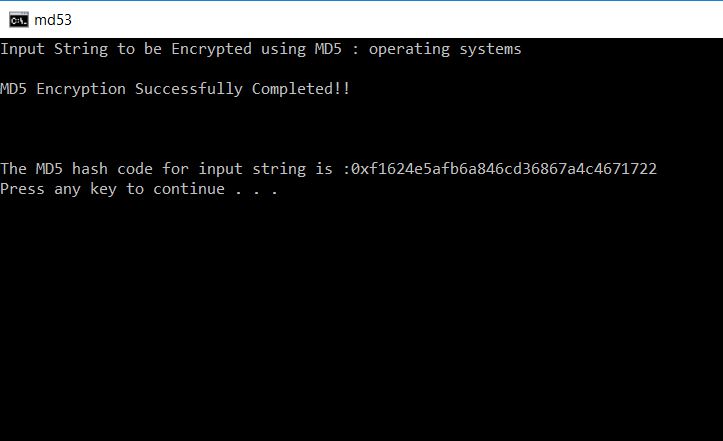
printf("%02x",u.b[k]);

}

return 0;

}

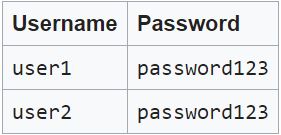
**CODE OUTPUT:**

****

**Chapter 5**

**SALTS**

In cryptography, a **salt** is random data that is used as an additional input to a one-way function that "hashes" data, a password or passphrase. Salts are used to safeguard passwords in storage.





**CONCLUSION**

This MD5 algorithm is for the 512 bit message transfer and also with the high security and Stream controlled transfer logic. This algorithm can be used in sending messages for 3G, 4G network. This can also be used for 5G network for which the work has been started. Here we are using 128 bit algorithm and using that as a basic element and create a application for 512 bit messages. The output would always is of 512bit message. In this way the secret information (e.g. passwords) can be shared with the peer. There is one more application of this algorithm is Message Authentication Code (MAC). This is an integrity check mechanism based on cryptographic hash functions using a secret key. Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties. In SCTP (Stream controlled transfer protocol), it is used by an endpoint to validate the State Cookie information that is returned from the peer in the COOKIE ECHO chunk .An Application of MD5 algorithm is implemented for the stream controlled transfer messages in the network. This would be a high security algorithm for data transfer in mobile networking with stream controlled logic. There may a vast number of applications for this algorithm in data transfer in various types of network.

**REFERENCES**

**1.**[**https://www.google.co.in/url?sa=t&source=web&rct=j&url=https://en.m.wikipedia.org/wiki/MD5&ved=2ahUKEwjF55qF79HeAhXGMI8KHVZ6BYkQFjAKegQIAhAB&usg=AOvVaw2\_kItQXNZDahJe\_pt5IhnK**](https://www.google.co.in/url?sa=t&source=web&rct=j&url=https://en.m.wikipedia.org/wiki/MD5&ved=2ahUKEwjF55qF79HeAhXGMI8KHVZ6BYkQFjAKegQIAhAB&usg=AOvVaw2_kItQXNZDahJe_pt5IhnK)

**2.https://www.google.co.in/url?sa=t&source=web&rct=j&url=https://ieeexplore.ieee.org/document/933564/&ved=2ahUKEwjF55qF79HeAhXGMI8KHVZ6BYkQFjAMegQIBxAB&usg=AOvVaw33IijnGQsPHgLPUOh05wTq**

**3.https://www.google.co.in/url?sa=t&source=web&rct=j&url=https://en.m.wikipedia.org/wiki/Salt\_(cryptography)&ved=2ahUKEwir7rS78NHeAhUmSY8KHaPeAVcQFjAAegQIABAB&usg=AOvVaw10hem\_RW1spTnLEX6mlCiz**