Abstract: A hash (also called a hash code, digest, or message digest) can be thought of as the digital fingerprint of a piece of data. You can easily generate a fixed length hash for any text string using a one-way mathematical process. It is next to impossible to (efficiently) recover the original text from a hash alone. It is also vastly unlikely that any different text string will give you an identical hash – a 'hash collision'. These properties make hashes ideally suited for storing your application's passwords. Although an attacker may compromise a part of your system and reveal your list of password hashes, they can't determine from the hashes alone what the real passwords are. A cryptographic hash function is a deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string, the (cryptographic) hash value, such that an accidental or intentional change to the data will change the hash value. The data to be encoded is often called the "message," and the hash value is sometimes called the message digest or simply digest. Cryptographic hash functions have many information security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption.

Keywords: Message Authentication Codes (Mac's), Checksums, Cryptographic.

I. INTRODUCTION

Cryptographic algorithms can be divided into three several classes, which are public key algorithms, symmetric key algorithms, and hash functions. While the first two are used to encrypt and decrypt data, the hash functions are one-way functions that do not allow the processed data to be retrieved. This algorithm is iterative, one-way hash functions that can produce a message to produce a condensed representation called a message digest. These algorithms enable the determination of a message's integrity; any change to the message will, with a very high probability, result in a different message digests. This property is useful in message authentication codes. A security protocol (cryptographic protocol or encryption protocol) is an abstract or concrete protocol that performs a security-related function and applies cryptographic methods, often as sequences of cryptographic primitives. A protocol describes how the algorithms should be used. A sufficiently detailed protocol includes details about data structures and representations, at which point it can be used to implement multiple, interoperable versions of a program. Cryptographic protocols are widely used for secure application-level data transport. For example, Transport Layer Security (TLS) is a cryptographic protocol that is used to secure web (HTTP) connections. It has an entity authentication mechanism, based on the X.509 system; a key setup phase, where a symmetric encryption key is formed by employing public-key cryptography; and an application-level data transport function. These three aspects have important interconnections. Standard TLS does not have non-repudiation support. There are other types of cryptographic protocols as well, and even the term itself has various readings; Cryptographic application protocols often use one or more underlying key agreement methods, which are also sometimes they referred to as "cryptographic protocols". For instance, TLS employs what is known as the Diffie-Hellman key exchange, which although it is only a part of TLS per se, Diffie-Hellman may be seen as a complete cryptographic protocol in itself for other applications. Cryptographic protocols can sometimes be verified formally on an abstract level. When it is done, there is a necessity to formalize the environment in which the protocol operates in order to identify threats. This is frequently done through the Dolev-Yao model.

A wide variety of cryptographic protocols go beyond the traditional goals of data confidentiality, integrity, and authentication to also secure a variety of other desired characteristics of computer-mediated collaboration. Blind signatures can be used for digital cash and digital credentials to prove that a person holds an attribute or right without
revealing that person's identity or the identities of parties that person transacted with. Secure digital time-stamping can be used to prove that data (even if confidential) existed at a certain time. Secure multiparty computation can be used to compute answers (such as determining the highest bid in an auction) based on confidential data (such as private bids), so that when the protocol is complete the participants know only their own input and the answer. End-to-end auditable voting systems provide sets of desirable privacy and auditability properties for conducting e-voting. Undeniable signatures include interactive protocols that allow the signer to prove a forgery and limit who can verify the signature. Deniable encryption augments standard encryption by making it impossible for an attacker to mathematically prove the existence of a plaintext message. Digital mixes create hard-to-trace communications.

II. SECURE HASH ALGORITHM

This Standard specifies secure hash algorithms - SHA-1 for computing a condensed representation of electronic data (message), i.e message digest. When a message of any length < 2^64 bits (for SHA-1) is input to an algorithm, the result is an output called a message digest. The message digests range is the length of 160 bits. The hash algorithm specified in this standard are called secure because, for a given algorithm, it is computationally infeasible 1) to find a message that corresponds to a given message digest, or 2) to find two different messages that produce the same message digest. Any change to a message will, with a very high probability, result in a different message digest. The SHA-1 produces a single output 160-bit message digest (the output hash value) from an input message. The input message is composed of multiple blocks. The input block, of 512 bits, is split into 80 x 32-bit words, denoted Wt as, one 32-bit word for each computational round of the SHA-1 algorithm, as depicted in Block diagram. Each round comprises additions and logical operations, such as bitwise logical operations (ft) and bitwise rotations to the left (ROTL^i). The calculation of ft depends on the round being executed, as well as the value of the constant Kt. The SHA-1 80 rounds are divided into four groups of 20 rounds, each with different values for and the applied logical functions (ft). The initial values of the A to E variables in the beginning of each data block calculation correspond to the value of the current 160-bit hash value, DM0 to DM1. After the 80 rounds have been computed, the A to E 32-bit values are added to the current DM.

A. Bit Strings and Integers

The following terminology related to bit strings and integers will be used.

- A hex digit is an element of the set {0, 1, ..., 9, a, b, ..., f}. A hex digit is the representation of a 4-bit string. For example, the hex digit "7" represents the 4-bit string “0111", and the hex digit “a" represents the 4-bit string “1010".

- A word is a w-bit string that may be represented as a sequence of hex digits. To convert a word to hex digits, each 4-bit string is converted to its hex digit equivalent, as described in (1) above. For example, the 32-bit string 1010 0001 0000 0111 1110 1010 0011 can be expressed as “a103f623", and the 64-bit string 1010 0001 0000 0111 1110 1010 00110011 0010 1110 1111 0011 0000 0101 can be expressed as “a103f6232f301". Throughout this specification, the “big-endian" convention is used when expressing both 32- and 64-bit words, so that within each word, the most significant bit is stored in the left-most bit position.

- An integer may be represented as a word or pair of words. A word representation of the message length, l, in bits, is required for the padding technique. An integer between 0 and 232-1 inclusive may be represented as a 32-bit word. The least significant four bits of the integer are represented by the right-most hex digit of the word representation. For example, the integer 291=28 + 25 + 21+20=256+32+2+1 is represented by the hex word 00000123. The same holds true for an integer between 0 and 264-1 inclusive, which may be represented as a 64-bit word.

B. Operations on Words

The following operations are applied to w-bit words in secure hash algorithm. SHA-1 operate on 32-bit words (w=32).

1. Bitwise logical word operations: U, Ũ, A, and Ø.

2. Addition modulo 2.

   The operation x + y is defined as follows. The words x and y represent integers X and Y, where 0 ≤ X < 2w and 0 ≤ Y < 2w. For positive integers U and V, let U mod V be the remainder upon dividing U by V. Compute Z=(X + Y) mod 2w. Then 0 ≤ Z < 2w. Convert the integer Z to a word, z, and define z=x + y.

3. The right shift operation SHR n(x), where x is a w-bit word and n is an integer with 0 ≤ n < w, is defined by

   SHR n(x)=x >> n (1)

4. The rotate right (circular right shift) operation ROTR n(x), where x is a w-bit word and n is an integer with 0 ≤ n < w, is defined by

Fig 1: SHA-1 Round calculation.
Implementation Of SHA Algorithm For Secure Realization Of Digital Signature Application

After a message has been padded, it must be parsed into N m-bit blocks before the hash computation can begin.

III. IMPLEMENTATION OF HASH

Suppose that the length of the message, M, is 1 bits. Append the bit “1” to the end of the message, followed by k zero bits, where k is the smallest, non-negative solution to the equation $l + 1 + k \cdot 448 \text{mod} 512$. Then append the 64-bit block that is equal to the number 1 expressed using a binary representation. For example, the (8-bit ASCII) message “abc” has length $8'3 = 24$, so the message is padded with a one bit, then $448 - (24 + 1) = 423$ zero bits, and then the message length, to become the 512-bit padded message. For SHA-1 the padded message is parsed into N 512-bit blocks, M(1), M(2), …, M(N). Since the 512 bits of the input block may be expressed as sixteen 32-bit words.

A. Setting the Initial Hash Value (H(0))

Before hash computation begins for each of the secure hash algorithms, the initial hash value, H(0), must be set. The size and number of words in H(0) depends on the message digest size.

1. SHA-1 Example: For SHA-1, the initial hash value, H(0), shall consist of the following five 32-bit words, in hex: 67452301, efcdab89, 98badcfe, 10325476, c3d2e1f0

Padding the message: The message, M, shall be padded before hash computation begins. The purpose of this padding is to ensure that the padded message is a multiple of 512.

Parsing the padding message: After a message has been padded, it must be parsed into N m-bit blocks before the hash computation can begin.

Setting the initial hash value (h(0)): Before hash computation begins for each of the secure hash algorithms, the initial hash value, H(0), must be set. The size and number of words in H(0) depends on the message digest size. For SHA-1, the initial hash value, H(0), shall consist of the following five 32-bit words, in hex:

- $H(0) = 67452301$
- $H1(0) = efcdab89$
- $H2(0) = 98badcfe$
- $H3(0) = 10325476$
- $H4(0) = c3d2e1f0$

D. Preprocessing

Preprocessing shall take place before hash computation begins. This preprocessing consists of three steps: padding the message (M), parsing the padded message into message blocks, and setting the initial hash value, $H(0)$.

1. Padding the Message

The message, M, shall be padded before hash computation begins. The purpose of this padding is to ensure that the padded message is a multiple of 512.

2. Parsing the Padded Message

Before hash computation begins for each of the secure hash algorithms, the initial hash value, $H(0)$, must be set. The size and number of words in $H(0)$ depends on the message digest size.
For \( i = 1 \) to \( N \):

1. Prepare the message schedule, \( \{ W_t \} \):

\[
\begin{align*}
\{ W_t \} & = \\
& \begin{cases} \\
M_t^0 & \text{for } 0 \leq t \leq 15 \\
ROT_3(W_{t-5} \oplus W_{t-10} \oplus W_{t-15}) & \text{for } 16 \leq t \leq 79
\end{cases}
\end{align*}
\]

2. Initialize the five working variables, \( a, b, c, d, \) and \( e \), with the \((i-1)\)st hash value:

\[
\begin{align*}
a &= H_t^{(i-1)} \\
b &= H_t^{(i-1)} \\
c &= H_t^{(i-2)} \\
d &= H_t^{(i-3)} \\
e &= H_t^{(i-4)}
\end{align*}
\]

3. For \( t = 0 \) to \( 79 \):

\[
\begin{align*}
T &= ROTL^3(a) + f_t(b,c,d) + e + K_t + W_t \\
e &= d \\
d &= c \\
c &= ROTL^3(b) \\
b &= a \\
a &= T
\end{align*}
\]

4. Compute the \( i \)th intermediate hash value \( H(i) \):

\[
\begin{align*}
H_t^{(i)} &= a + H_t^{(i+1)} \\
H_t^{(i+1)} &= b + H_t^{(i+3)} \\
H_t^{(i+2)} &= c + H_t^{(i+5)} \\
H_t^{(i+3)} &= d + H_t^{(i+7)} \\
H_t^{(i+4)} &= e + H_t^{(i+9)}
\end{align*}
\]

After repeating steps one through four a total of \( N \) times (i.e., after processing \( M(N) \)), the resulting 160-bit message digest of the message, \( M \), is

\[
H(N) \parallel H(N+1) \parallel \ldots \parallel H(N+15).
\]

B. SHA-1 Algorithm

Where \( << \) denotes circular shift to the left by \( s \) bits and \( \oplus \) is a logical xor operation. Let \( K_t \) be a constant value for step \( t \). The values of \( K \) are set as follows:

\[
K_t = \begin{cases} \\
"5a82799�" & 0 \leq t \leq 19 \\
"6ed9eba1" & 20 \leq t \leq 39 \\
"8f1bcde" & 40 \leq t \leq 59 \\
"ca62e1db" & 50 \leq t \leq 79
\end{cases}
\]

A function \( F(X,Y,Z) \) depending on the step \( t \) is defined as follows.

\[
F(X,Y,Z) = \begin{cases} \\
(X \land Y) \oplus (\neg X \land Z) & 0 \leq t \leq 19 \\
X \oplus Y \oplus Z & 20 \leq t \leq 39 \\
(X \land Y) \oplus (X \land Z) \oplus (Y \land Z) & 40 \leq t \leq 59 \\
X \oplus Y \oplus Z & 60 \leq t \leq 79
\end{cases}
\]

1. SHA-1 example

Suppose that the length of the message, \( M \), is 1 bits. Append the bit “1” to the end of the message, followed by \( k \) zero bits, where \( k \) is the smallest, non-negative solution to the equation \( 1 + 1 + k = 448 \mod 512 \). Then append the 64-bit block that is equal to the number \( 1 \) expressed using a binary representation. For example, the (8-bit ASCII) message “abc” has length 8x3=24. So the message is padded with a one bit, then 448–(24 + 1)=423 zero bits, and then the message length, to become the 512-bit padded message (see fig 2).

![Fig 2: Length of the padded message should now be a multiple of 512 bits.](image)

2. Parsing the Padded Message

After a message has been padded, it must be parsed into \( N \) \( m \)-bit blocks before the hash computation can begin. For SHA-1, the padded message is parsed into \( N \) 512-bit blocks, \( M_1, M_2, \ldots, M_N \). Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block \( I \) are denoted \( M^{(i)}_0 \); the next 32 bits are \( M^{(i)}_1 \), and so on up to \( M^{(i)}_{15} \). Setting the Initial Hash Value (\( H^{(0)} \)). Before hash computation begins for each of the secure hash algorithms, the initial hash value, \( H(0) \), must be set. The size and number of words in \( H(0) \) depends on the message digest size. SHA-1: For SHA-1, the initial hash value, \( H(0) \), shall consist of the following five 32-bit words, in hex:
Implementation Of SHA Algorithm For Secure Realization Of Digital Signature Application

\[
\begin{align*}
H_0^{(0)} & = 67452301 \\
H_1^{(0)} & = efcdab89 \\
H_2^{(0)} & = 98badcfe \\
H_3^{(0)} & = 10325476 \\
H_4^{(0)} & = c3d2e1f0.
\end{align*}
\]

These words were obtained by taking the first thirty-two bits of the fractional parts of the square roots of the first eight prime numbers. The final digest message for the given input message “abc”. The resulting 160-bit message digest is a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d.

3. General Description

The OL_SHA core is a fully compliant hardware implementation of the SHA-1 algorithm, suitable for a variety of applications. The SHA-1 algorithm is based on principles similar to those used by Professor Ronald L. Rivest of MIT when designing the MD4 message digest algorithm, and is closely modeled after that algorithm. It operates on message blocks of 512 bits for which a 160-bit (5 x 32-bit words) digest is produced. Corresponding 32-bit words of the digest from consecutive message blocks are added to each other to form the digest of the whole message. The block diagram of the core is shown in Figure 3. SHA-1 may be used to hash a message, M, having a length of \( \ell \) bits, where \( 0 \leq \ell < 2^{64} \). The algorithm uses 1) a message schedule of eighty 32-bit words, 2) five working variables of 32 bits each, and 3) a hash value of five 32-bit words. The final result of SHA-1 is a 160-bit message digest. The words of the message schedule are labeled \( W_0, W_1, \ldots, W_{79} \). The five working variables are labeled \( a, b, c, d, \) and \( e \). The words of the hash value are labeled \( H_0^{(i)}, H_1^{(i)}, \ldots, H_4^{(i)} \), which will hold the initial hash value, \( H(0) \), replaced by each successive intermediate hash value (after each message block is processed), \( H(i) \), and ending with the final hash value, \( H(N) \). SHA-1 also uses a single temporary word, \( T \).

IV. RESULTS

The following figures 3, 4 and 5 shows the results of the RTL schematic, technology schematic and waveforms.

Fig 3: Block Diagram for the SHA-1 processor.

Fig 4: RTL Schematic.

Fig 5: Technology Schematic.
V. CONCLUSION

SHA is famous message compress standard used in computer cryptography. The improved version SHA-1 algorithm has been analyzed in this paper, and implemented by VHDL. Using the SHA-1 module, a long message can be generated a short and safe message abstract in a very short time i.e message digest. This algorithm used in many authentication applications. It provides more security compared other cryptography algorithms.

VI. REFERENCES


