Chapter 4

SACA Based Message Authentication Schemes

This dissertation reports detailed characterization of a particular class of cellular automata and its application in the field of message/image authentication. In this background, this chapter first provides a brief survey on authentication and then introduces the proposed schemes.

The human society is currently living in cyber age. Phenomenal technological advances of this age have brought unprecedented benefits to the society. However, at the same time this has generated some unique social problems the human society has never encountered before in the history of civilization.

Secured communication in the networked society of cyber age has become a prerequisite for growth of human civilization of twenty-first century. Current demands for secured communication have focused intensive interest on ‘authentication’. There is a great demand for low cost high-speed on-line scheme for generation of message authentication code (MAC).

Electronic transfer of all types of digital files demands authentication and verification of data source, protection of copyright, and detection of intrusion. A strong trend in the development of the mechanisms for authentication of message is based on cryptographic hash functions designed for MD5 by Rivest [169]. However, the conventional MD5 based message authentication, as reported in [160, 174, 194], has its limitations.

This chapter reports a simple, high speed, low cost authentication scheme for digital messages. It employs the computing model built around single attractor CA (SACA) developed over the Galois extension field GF(2^p). The SACA generates efficient one-way hash function that has been employed for CA based authentication (CAA).

For a given digital file (viewed as a message), the SACA based one-way hash
function generates \( MAC \) and produces message digest as a string of symbols in the extension field \( GF(2^p) \) [33]. The structure of extension field and the unique behavioral model of \( SACA \) have provided strength to this model against cryptanalytic attacks.

Cryptanalysis of the proposed scheme establishes the fact that compared to other known schemes like MD5, SHA etc., the current scheme is more secure against all known attacks. Moreover, high speed execution of the proposed scheme makes it ideally suitable for real time on-line applications [135, 136]. Further, the simple, regular, modular, and cascadable structure of \( CA \) with local interconnections make the scheme ideally suitable for VLSI implementation with throughput in the range of tens of gigabytes per second.

Cellular automata based authentication (\( CAA \)) exploits the potential of \( SACA \) based unique address generator (described in the previous chapter) for generating signature referred to as message digest through the message authentication code (\( MAC \)).

The review of authentication portrays the development of methodologies from the days of random approach to the current schemes based on the framework of machine intelligence in the next section.

### 4.1 Authentication - a Survey

A number of surveys on authentication are published at regular intervals [163, 162, 238] that enables one to have a focused look into the recent trends in this field.

Authentication is the study of verifying the integrity and authenticity of information which is a prime necessity in computer and network systems. In particular two parties communicating over an insecure channel require a method by which information sent by one party can be validated as authentic (or unmodified) by the other [18].

**Protocols**

The past two decades have seen an enormous increase in the development and use of networked and distributed systems, providing increased functionality to the user and ensuring more efficient use of resources. To obtain the benefits of such systems parties will cooperate by exchanging messages over networks. The parties may be users, hosts or processes; [42] they are generally referred to as principals in authentication literature.

An authentication protocol is a sequence of message exchanges between principals that either distributes secrets to some of those principals or allows the use of some secret to be recognized [26, 194]. At the end of the protocol the principals involved may deduce certain properties about the system; for example, that only certain principals have access to particular secret information (typically cryptographic keys) or that a particular principal is operational. They may then use this information to verify claims about subsequent communication, for example, a received message encrypted
with a newly distributed key must have been created after distribution of that key and so is timely.

When we receive a message we want to be sure that it has been created recently and in good faith for a particular purpose by the principal who claims to have sent it. We must be able to detect the time when a message has been created or modified by a malicious principal or intruder with access to the network or whether a message was issued some time ago (or for a different purpose) and is currently being replayed on the network [42].

M. Burrows et. al gave informal accounts of some salient features in authentication [26]. A considerable number of authentication protocols have been specified and implemented. The area is, however, remarkably subtle and many protocols have been shown to be flawed a long time after they were published.

Two types of authentication techniques have been developed in cryptography: digital signature and message authentication codes (MACs) [124, 194] though authentication protocol does not differ in both techniques.

**Digital Signature**

The digital signature approach applies a cryptographic hash function to the message M. The hash value is then encrypted with the sender S’s private key. The result is the digital signature of the message M. The digital signature is sent to R together with the message M it authenticates.

The receiver R uses the sender’s public key to decrypt the digital signature to obtain the original hash value, which is compared with the hash value calculated from the received message M. If they are the same, then the message R received is authentic, otherwise it has been tampered [250]. The mechanism for digital signature is explained through figure 4.1. In 1978 Needham Schroeder published a message authentication protocol based on digital signature approach [141] and became the basis for many similar protocols in later years. In 1981, Denning and Sacco demonstrated that the protocol was flawed and proposed an alternative protocol based on digital signature approach [58]. This set the general trend for the field. In 1994 Martin Abadi demonstrated that the protocol of Denning and Sacco was flawed [3].

In 1995, Lowe demonstrated an attack on message authentication protocol through digital signature approach of Needham and Schroeder (seventeen years after its publication) [113]. Use of simple hash function and encryption algorithms in message authentication through digital signature approach are replaced through MAC approach which are faster and more secure [14, 13]. In the intervening years a whole host of protocols have been specified and found to be flawed [42].
Sender (S) wants to send a message (M) to receiver (R). M is passed through a hash function as input. Hash output and private key of S are invoked to an encryption function. The output produced is appended to M and is sent for transmission. R gets that appended message.

The appended part (displayed as shaded in the above figure) and public key of S are passed through a decryption function, where as M is passed through that hash function. Both outputs are compared. If they are same then R conclude that M is authenticated.

**Figure 4.1: Digital signature**

**Message Authentication Code**

MACs or, message authentication code (other terms used include integrity check Value or cryptographic checksum) have the following principle. When party A transmits a message to party B, it appends to the message a value called the authentication tag, computed by the MAC algorithm as a function of the transmitted information and the shared secret key.

At reception, B recomputes the authentication tag on the received message using the same mechanism (and key) and checks that the value he obtains equals the tag attached to the received message. Only if the values match, the information received, is considered as not altered on the way from A to B [18, 250]. The mechanism for message authentication code principle is explained through figure 4.2.

In message authentication through digital signature approach, hash function used is a simple hash function which does not require a key involvement. Along with this an encryption algorithm is also used which requires a key.
Sender or party A wants to send a message (M) to receiver or party B. A and B both shares a common secret key which is known to them only. That secret key and M are passed through a message authentication code (MAC) function as input and authentication tag (displayed as shaded in the above figure) is produced as output. This output is appended to M and is sent for transmission. B gets that appended message.

M and the secret key are passed through MAC and the output produced is compared with the appended part that is authentication tag. If they are same then R conclude that M is authenticated.

Figure 4.2: Message authentication code

Where as in message authentication through MAC approach, a MAC algorithm or function is used which is a special class of hash function. It is named as cryptographic hash function and is an one-way hash function. The output produced from MAC function is termed as message digest.

The use of cryptographic hash functions like MD5 [169] or SHA [147] for message authentication had become a standard approach for message authentication scheme based on a cryptographic hash function and are proven to be secure as long as the underlying hash function has some reasonable cryptographic strength [18].

MAC constructions based on cryptographic hash functions have been in use for a few years (see Tsudik [211] for an early description of such constructions and Touch [208] for a long list of Internet protocols that use this approach). Preneel and Oorschot [163, 164] survey existing constructions and point out to some of their properties and weaknesses [18].
Tsudik [211] has proposed methods on message authentication which are only based on one-way hash functions and use some keys to make them secure [14, 13] and this has become a standard approach in today's Internet applications and protocols.

While secret keys are an essential ingredient in a message authentication function, most cryptographic hash functions, and specifically functions like MD5 or SHA do not use keys at all and so constructing a cryptographic hash function in conjunction with a key has took place [18]. Thus the MAC function which was a one-way hash function earlier has been replaced by a keyed one-way hash function.

Some preliminary definitions regarding hash function is referred over here.

**Hash function** - A public function that maps a message of any length into a fixed-length hash value, which serves as the authenticator.

**One-way hash function** - A hash function that works in one direction: it is easy to compute a hash value from pre-image, but it is hard to generate a pre-image that hashes to a particular value.

**Keyed one-way hash function** - A keyed one-way hash function is a one-way hash function with the addition of a secret key. The theory is exactly the same as one-way hash functions, except only with the key can verify the hash value.

Bakhtiari et.al [13, 14] has formalized the salient features of a secured keyed one-way hash function (SKOWHF) as follows.

**Definition 4.1** A function $f()$ that maps a fixed length key $K$ and an arbitrary length message $M$ to a fixed length message digest $MD$ is SKOWHF, if it satisfies the following properties:

- Given $K$ and $M$, it is easy to compute $MD = f(M, K)$,
- Given $K$ and $MD$, it is hard to find $M$ with $MD = f(M, K)$,
- Given $K$, it is hard to find two values $M$ and $M' (\neq M)$ such that $f(M, K) = f(M', K)$,
- Given (possibly many) pairs of $[M, MD]$, it is hard to find the secret key $K$,
- Without knowledge of $K$, it is hard to compute $f(M, K)$ for any $M$.

Conventionally authentication of message or image is based on cryptographic hash functions designed for SHA by a research group [147] and MD5 by Rivest [169] where the key is incorporated within the message. However, the conventional MD5 based message authentication, as reported in [174], cannot withstand advanced cryptanalytic attacks. The weakness of MD5 has also been established by Bakhtiari et. al [14, 13] and by Dobbertin [59]. Preneel has criticized SHA as well as MD5 [163].
In this light the authentication scheme employing CA based SKOWHF is introduced in this thesis to ensure higher security. The special class of cellular automata (CA) referred to as single attractor CA (SACA) is used to develop the keyed one-way hash function. For a given digital file (viewed as message/image), the SACA based keyed one-way hash function generates data authentication code and produces digest as a string of symbols in the extension field GF(2^p).

According to Preneel [164] one of the major aspect of security for a SKOWHF is its message digest length which is 128 or 256 bit for MD5 and 160 bit for SHA where as the CA based SKOWHF developed by the current author can be of any bit length ranging from 128 to 1024 bit without hampering speed of the algorithm.

Details analysis and comparison of CA based SKOWHF with MD5 and SHA has been elaborately described in section 4.4. In that section the strength of the employed CA based secured keyed one-way hash function is tested through a rigorous security analysis. The cryptanalytic results prove it’s superiority over MD5. Section 4.3 describes the message authentication scheme employing CA based SKOWHF following the general introduction of message authentication scheme in next.

4.2 Message Authentication Scheme

A message authentication scheme is explained through figure 4.3. Message is sent from source A to destination B. Let source A have a message M to be sent to destination B and they share a common secret key K. A calculates message digest $C_K(M)$ from M and secret key K employing keyed one-way SACA based hash function.

Message $M$ and digest $C_K(M)$ are transmitted to B where B performs the same function on the received message to generate a new digest $C_K(M')$. The message gets authenticated if $C_K(M)$ and $C_K(M')$ match. During transmission on its way from source to destination, a hacker can change the message from M to $M'$, which will be immediately reflected through the dissimilarities between $C_K(M)$ and $C_K(M')$.

Message Digest Generating Scheme

The secured keyed one-way hash function produces message digest and so is also termed as message digest generation functions. They are of primary importance for the security protocols such as message authentication, data integrity and other different mechanisms applied in the field of E-Commerce.

The function accepts a variable-size message $M$ as input and produces a fixed-size hash code $H(M)$, referred to as message digest or fingerprint, as output. The distributed systems on net (like electronic payment system) require strong security and robustness to prevent cyber crime. Following algorithm works for message digest generation function.

**Algorithm 4.1 Message_Digest_Generator**

**Input:** $M$: message of arbitrary length and $K$: secret key.
Output: \( C_K(M) \): message digest of fixed length.

Step 1. \( M \) and \( K \) are passed through a keyed one-way hash function and a fixed length message digest \( C_K(M) \) is produced.

The message digest generation through hash function is explained through figure 4.4.

Authenticity guarantees the originator of the data and integrity guarantees that the data has not been altered in transit [49].

The desired features of message digest schemes [174] are - (i) fast digest generation, (ii) high cracking complexity - that is, hard to deliver the private key from the public key, (iii) not to have same digest for two or more random message texts (birthday attack) i.e an adversary would not be able to find two random messages \( M_1, M_2 \) - which are different, such that \( H(M_1) = H(M_2) \), where \( H \) is the hash function to generate message digest.

The limitations of the conventional MD5 [170] based message authentication are - it cannot withstand the advanced cryptanalytic attack such as birthday attacks [192], the hash function used is weak [174] and lacks high cracking complexity. The existing schemes produce the fixed length key. Key length on MD5 is of 128 bit [192]; and in RSA it is of 160 bits [160, 192]. Alterations of key lengths are very cumbersome and complex in these schemes [174].

In the above background, we propose an efficient SACA based message digest generation scheme. GF(2) SACA or, two predecessor single attractor (TPSA) cellular automata based message authentication has been reported in [33, 55]. In the TPSA CA or, SACA at GF(2) (figure 4.5), having reachable and non-reachable states, for each reachable state there exist exactly two predecessors. In the corresponding structure of SACA at GF(2^p) there are exactly \( 2^p \) number of predecessors for a reachable state. This property of SACA is exploited to enhance the cracking complexity in message authentication. The GF(2^p) CA handles symbols from the set \( \{0, 1, 2, \ldots, (2^p - 1)\} \) \( \in \) GF(2^p) rather than \( \{0, 1\} \) \( \in \) GF(2); in effect it reduces execution time of the software version of the proposed design. The execution speed also increases significantly in hardware version.

The experimentation is done on a large number of data files of different sizes. The experimental results clearly establish the fact that the proposed message digest computation method is very fast with high cracking complexity - that is, for a given digest, it is very hard to compute the private key. The proposed scheme satisfactorily qualifies the different cryptanalytic test such as the entropy test. Moreover, this method qualifies various known cryptanalysis attack like chosen-key attack, differential cryptanalysis, related key cryptanalysis etc. The SACA based design permits the easy implementation of variable key size (128, 130, .., 256, .., 512).

This chapter details the algorithm for generation of message digest. Subsequently,
Sender or source A wants to send a message (M) to receiver or destination B. A and B both shares a common secret key (K) which is known to them only. K and M are passed through a hash function which is a keyed one-way hash function as input and message digest (C_K(M)) is produced as output. This C_K(M) is appended to M and is sent for transmission to destination B. It may be the case that during transmission it is hacked and modified. In any case B gets a message say M’.

M’ and K are passed through the same keyed one-way hash function and the output message digest produced C_K(M’) is compared with the appended part, that is, C_K(M). If they are same then B conclude that M’ has certainly come from A. Otherwise B concludes that M’ has been altered during transmission that means a hacking has taken place.

Figure 4.3: Message authentication
the superiority of the message digest is illustrated through extensive experimentation [135, 136].

4.3 Cellular Automata Based Authentication (CAA) Scheme for Message

The proposed scheme employs keyed one-way hash function based authentication using GF(2^p) SACA and its dual $\overline{SACA}$. The message digest is generated through that keyed one-way hash function using GF(2^p) $SACA$, $\overline{SACA}$ pair.

4.3.1 CA Based Hash Function

Cellular automata based authentication (CAA) exploits the potential of $SACA$ based perfect hashing for generating message digest referred to as signature. The one-way hash function maps a secret key ($p$) and an arbitrary length input message data ($M$) to a fixed length (= $n$) hash output referred to as message digest ($C_p(M)$). In order to use the unique address generator $Unique\_Map, SS$ (algorithm 3.1) as a one-way hash function, we form a composite function ($H$). As already mentioned in the previous chapter, the pair ($SACA, \overline{SACA}$) represents the perfect hash function $SS$.

(Note: From henceforth, we use the term ($SACA, \overline{SACA}$) and $SS$ interchangeably).

Here we present three basic steps to this authentication scheme in a simple manner.

1. A message of any arbitrary length which will be authenticated is converted to binary file and proper padding is done with it, to make it a file of size $M$ bits.

2. These $M$ bits are grouped into $l$ blocks (tokens).

3. Each token comprises of $n$ sub-blocks each of length $p$ bits.

The basic steps of authentication are explained through a block diagram in figure 4.6.
The message is broken into tokens \((M_1, M_2, \ldots, M_k)\). The function \(\text{Unique}_\text{Map}_*\overline{S}\) is employed on each token with a different pair of \((SACA, \overline{SACA})\) constructed from the secret key at the run time. \(\text{Pick}_*\overline{S}\) function describes generation of \((SACA, \overline{SACA})\) pair. The algorithmic steps of the hash function follows.

**Algorithm 4.2 Hash-To-Generate_Message_Digest**

**Input:** Message \(M\) of length \(|M|\) bits; Private key \(\mathcal{P}\): \(n \times p\) bits: \(n\) symbols; \(p\) - extension field parameter used.

**Output:** Message Digest \(H\): \(n \times p\) bits

**Step 1:** Break Message \(M\) into \(l\) blocks \(\{M_1, M_2, \ldots, M_l\}\) each of length \(n\) symbol \([s_1, s_2, \ldots, s_n]\) where, \(s_i (i=1 \text{ to } n) \in GF(2^p)\) and set \(\overline{S}_0 = \phi\).

**Step 2:** \(\mathcal{P}_1 = \mathcal{P}\) (Private Key)

\[
\text{For}(i=1 \text{ to } l)
\]

\[
\{ \]
Message of arbitrary length $M = 12$ bits (say) 

$$1 1 1 0 1 0 0 1 0 1 0 1$$

$M (= 12)$ bits are grouped into $1 (= 2)$ blocks (tokens).

Each block or token consists of $n = 3$ sub-blocks each of length $p = 2$ bits since it is working in (Galois Field) $GF(2^p)$ where $p = 2$ as per choice.

$$\begin{array}{|c|c|}
\hline
\text{Token } M_1 & \text{Token } M_2 \\
\hline
1 1 1 0 1 0 & 1 1 1 0 1 0 \\
3 2 2 & 1 1 1 \\
\hline
\end{array}$$

**Figure 4.6:** Block diagram of basic steps of authentication

**Step 3:** $SS_i = \text{Pick}_2 SS(P_i, SS_i-1) \quad \text{/* The SACA, SACA pair is generated based on the secret key and previous SACA, SACA pair */}$

**Step 4:** $H_i = \text{Unique Map}_SS(M_i) \quad \text{/* The unique hash address is generated through the function */}$

**Step 5:** $P_{i+1} = H_i \oplus P_i \quad \text{/* The secret key for the next round depends upon the hash output of the present round */}$

**Step 6:** Output $P_{i+1}$ as the Message Digest.

In algorithm 4.2, an $(SACA, \bar{SACA})$ pair is needed at each iteration. Generation of $(SACA, \bar{SACA})$ pair can be made by two ways, static generation or dynamic
generation. Dynamic generation will be discussed in the subsequent subsections. Following subsection describes the static generation of \((SACA, SACA)\) pair.

### 4.3.2 Static Generation of SACA

In this case a static database of various \((SACA, SACA)\) pair are stored and retrieved whenever is required. As this is done through a simple look up table procedure so it is very fast.

**Example 4.1** The message digest generation is explained following the algorithm 4.2 with a simple illustration through the following steps. The \((S\overline{S})\) pairs are in \(GF(2^2)\) and each of length \(n = 3\) cells.

1. Message \(M\) is of length \(|M| = 12\) bits = 6 symbols are broken into two tokens \(\{M_1, M_2\}\).
   
   Message \(M\): [3 2 2 1 1 1], private key \(P\): [1 3 2], number of blocks \(l = 2\) and the message blocks: \(M_1 = [3 2 2], M_2 = [1 1 1]\).

2. Assign \(P\) to \(P_1\).
   
   For \(i=1\),

3. \(S\overline{S}_1 = PickS\overline{S}(P_1, \phi)\) picks a \(S\overline{S}\) from the static database and say this SACA is

   \[
   \begin{pmatrix}
   3 & 1 & 0 \\
   1 & 1 & 2 \\
   0 & 3 & 2
   \end{pmatrix}
   \]

4. \(H_1 = Unique\_MapS\overline{S}(M_1) = [2 2 1]\).

5. \(P_2 = H_1 \oplus P_1 = [1 3 0]\).

Since tokens are not exhausted, for \(i = 2\), the \(P_2 = [1 3 0]\) acts as the private key for the next token [1 1 1] and step 3 to step 5 of algorithm 4.2 are repeated. And finally \(P_3 = [3 1 3]\) is obtained as message digest. \(\square\)

The static database has the following disadvantages:

1. Overhead of maintaining the database in memory.

2. Security suffers due to availability in memory.

3. Many potential SACA pairs are not exploited to reduce memory overhead.

To overcome these disadvantages we dynamically generate the required \((SACA, SACA)\) pair during run time. The next subsection develops a unique method, based on theorem 3.7 and theorem 3.8 of SACA (from previous chapter) for generation of \(SACA\) (which is \(PickS\overline{S}(P, S\overline{S})\) - step 3 of algorithm 4.2) which does away with static database and ensures dynamic generation of SACA. This dynamic generation of \((S\overline{S})\) is also very fast. Therefore \(PickS\overline{S}\) is now modified and is discussed in next subsection along with the \((S\overline{S})\) generation time.
4.3.3 Dynamic Generation of SACA

The algorithmic steps for synthesis of an $n$ cell $GF(2^p)$ SACA and its dual are noted below with an illustrating example. There are two algorithms reported as $\text{Construct\_Initial\_SACA}$ and $\text{Pick\_S\overline{S}}$.

These algorithms are incorporated in the message digest generation package. The $\text{Construct\_Initial\_SACA}$ algorithm (step 2 of algorithm 2) synthesizes the first SACA from the private key $P$. The $\text{Pick\_S\overline{S}}$ (step 2 of algorithm 2) subsequently synthesizes the SACA from the one employed at the previous iteration.

Each of the processes is explained in algorithmic form and subsequently illustrated with an example. The synthesis algorithm is done directly depending upon the key and it is based on the theorems 3.7 and 3.8 from the previous chapter.

**Algorithm 4.3 Construct\_Initial\_SACA ($P$)**

**Input:** $P$: secret key.

**Output:** SACA ($T_0$) and SACA ($T_0, F_0$);

Step 1. Check whether all symbols of secret key $P$ is non-zero and if there exist a zero then make it non-zero symbol based upon information of neighborhood.

Step 2. Discard a random symbol from the secret key depending on all $n$ symbols of the secret key and form a $(n - 1)$ length symbol string.

Step 3. Construct the characteristic matrix $T_0$ of size $n \times n$. using the $(n - 1)$ length symbol string and making use of theorem 3.7 from previous chapter.

Step 4. Construct SACA by $T_0$ and $F_0$ where $F_0$ is the inversion vector obtained through the results of theorem 3.8 from previous chapter.

Step 5. In each iteration, the input $T$ matrix is the SACA used in the previous pass.

The next algorithm generates equivalent SACA through elementary row/column operations.

**Algorithm 4.4 Pick\_S\overline{S} ($P$)**

**Input:** $P$: secret key;

**Output:** SACA ($T$) and SACA ($T, F$);

Step 1. If ($S\overline{S} \leftarrow \phi$)
\{
  \text{Construct\_Initial\_SACA ($P$) }/* \text{The construction is initiated if there is no initial (SACA, S\overline{S}) pair */}
\}


Step 2. $T = T_0$ and $(T, F) = (T_0, F_0)$
For $(\alpha = 1$ to $k)$, where $k$ is the number of iterations

\{

Step 3. Select two rows $i$ and $j$ from $T$ and $(T, F)$ (augmented) randomly according to the secret key $P$

Step 4. Replace the $i^{th}$ row with the xor of $i^{th}$ and $j^{th}$ rows

Step 5. Replace the $j^{th}$ column with the xor of $i^{th}$ and $j^{th}$ columns

Step 6. Assign the resultant matrix in $T$ and $(T, F)$

\}

From the rules of equivalent matrix [81], this is obvious that the above algorithm will always find equivalent $SACA$, $SACA$ and the iterations are needed to enhance security.

Example 4.2 Let the secret key be $P$: [1 3 2].

1. Construct initial $SACA$ $(P)$ produces $SACA$ ($T_0$), $SACA$ ($T_0, F_0$) pair as

\[
\begin{pmatrix}
0 & 0 & 0 \\
3 & 0 & 0 \\
0 & 2 & 0
\end{pmatrix}
\] and

\[
\begin{pmatrix}
0 & 0 & 1 \\
3 & 0 & 3 \\
0 & 2 & 0
\end{pmatrix}
\]

where $F = [1 3 2]$.

2. $T_0$ is assigned to $T$ and $T_0, F_0$ is assigned to $T, F$.

3. Repeat step 3 to step 6 for $k$ iterations starting with $T$ and $T, F$. After all the iterations the final $T$ and $T, F$ are obtained as

\[
T = \begin{pmatrix}
3 & 1 & 0 \\
1 & 1 & 2 \\
0 & 3 & 2
\end{pmatrix}
\] and $(T, F)$ is constructed with $F = [1 3 1]$ as

\[
\begin{pmatrix}
3 & 1 & 0 & 1 \\
1 & 1 & 2 & 3 \\
0 & 3 & 2 & 1
\end{pmatrix}
\] $\Box$

Above example is also made clear through the block diagram shown in figure 4.7.

The generation of $SACA$ is extremely fast. The estimate of time for generating $SACA$ is presented next.
Private or secret key $P$ of length 6 bits

0 1 1 1 1 0
1 3 2

Construction of initial SACA ($T_0$) and SACA ($T_0, F_0$) pair

$T_0 = \begin{bmatrix} 0 & 0 & 0 \\ 3 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$

$T_0 \cdot F_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 3 & 0 & 0 & 3 \\ 0 & 2 & 0 & 2 \end{bmatrix}$

Construction of equivalent SACA ($T$) and SACA ($T, F$) pair

$T = \begin{bmatrix} 3 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 3 & 2 \end{bmatrix}$

$T \cdot F = \begin{bmatrix} 3 & 1 & 0 & 1 \\ 1 & 1 & 2 & 3 \\ 0 & 3 & 2 & 1 \end{bmatrix}$

Figure 4.7: Block diagram of dynamic generation of SACA
**SACA Generation Time**

table 4.1 details the generation time required to synthesize SACA for different values of \(n\) and \(p\). The experiment is performed on a Sun m/c with Solaris 5.7. From table 4.1 it is found that construction of SACA in this method is faster than the time taken to construct the SACA proposed in [55].

table 4.1 establishes the scalability of the SACA generation scheme. The growth of generation time increases but sub-linearly than that of lattice size \(n\) of SACA. Moreover, higher value of \(p\) leads to reduction of computation time. This is because rather than handling \(np \times np\) matrix with GF(2) elements, we deal with \(n \times n\) matrix with GF(2\(^p\)) elements.

<table>
<thead>
<tr>
<th>(n \times p)</th>
<th>CPU Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p=1)</td>
<td>(p=2)</td>
</tr>
<tr>
<td>32</td>
<td>0.024</td>
</tr>
<tr>
<td>64</td>
<td>0.038</td>
</tr>
<tr>
<td>128</td>
<td>0.049</td>
</tr>
<tr>
<td>256</td>
<td>0.142</td>
</tr>
</tbody>
</table>

Figure 4.8 describes CA based message authentication scheme where the (SACA, SACA) pair are generated dynamically. The next section establishes the robustness of the scheme through experimental results.

### 4.4 Performance Analysis

In this section we report the performance of CAA in terms of robustness of the scheme to withstand different types of attack and then its execution speed. We break the result of cryptanalytic attacks in following two parts - (1) Brute force attack and (2) Vulnerability test through various standard cryptanalysis.

#### 4.4.1 Brute Force Attacks

Here we discuss some brute force attacks which depends on key length used in the scheme.

**Cracking complexity**

One-way functions used in message digest generation are easy to compute, but significantly harder to reverse and this hardness is measured by cracking complexity. The CAA scheme generates a message digest out of the given message \(M\). Let, the message comprise of \(l\) number of blocks, each block having \(n\) number of subblocks,
each subblock with \( p \) bits (where \( p \) is the dimension of Galois field \( GF(2^p) \)). The message digest generation function \( C \) implemented by the algorithm 4.2 on the message \( M \) generates the digest \( S \), that is \( C(M) = S \). Let, \( C'(S) = M \), if one wants to implement the reverse function \( C' \), its complexity considering only one block of \( np \) bits is \((2^p - 1)^{n-1}\) since one needs to evaluate all possible \((2^p - 1)^{n-1}\) combinations of secret key, leading to unique SACA (theorem 3.7) to arrive at the original message block. The security level of the scheme can be enhanced by increasing the value of \( n \), \( p \), and the key size \( K \).

**Birthday attack**

In *Birthday Attack* effort is directed to find two messages which hash to the same value. The search for such pair of messages becomes difficult as the key sizes increases. If the key is of length \( m \), \( 2^{m/2} \) messages has to be hashed to find the required pair [174, 192].

Table 4.2 illustrates the number of messages needed to perform birthday attack. 2\(^{nd}\) column of table - 4.2 shows the key length of various schemes. Since the key length at our proposed scheme is largest, \( 2^{128} \) (table 4.2-last row, 3\(^{rd}\) column) number of messages has to be hashed to perform birthday attack on our scheme. The CAA scheme can easily employ variable length key of any size (128, 256, 512,..) since it employs simple, regular, modular, cascadable structure of CA and thus can prevent birthday attack in a better way.

### Table 4.2: Comparison Regarding Birthday Attack

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Key length in bits</th>
<th>Messages to hash in birthday attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>128</td>
<td>( 2^{64} )</td>
</tr>
<tr>
<td>SHA</td>
<td>160</td>
<td>( 2^{80} )</td>
</tr>
<tr>
<td>SACA at ( GF(2) )</td>
<td>128</td>
<td>( 2^{64} )</td>
</tr>
<tr>
<td>Proposed SACA at ( GF(2^p) )</td>
<td>256</td>
<td>( 2^{128} )</td>
</tr>
</tbody>
</table>

**4.4.2 Vulnerability Test through Different Cryptanalysis**

Cryptanalyst’s try to guess whether the function is such that two input data or keys, close to each other in terms of bit distance, produce the same output.

The robustness of CAA is tested in respect of the strength of the SACA based hash function employed for the scheme in the following three attacks. The attacks are much more subtle than mere brute force attack discussed above. Each of the attacks and their performance on the hash function is discussed next.
Message of arbitrary length $M = 12$ bits (say)
\[1 1 1 0 1 0 0 1 0 1 0 1\]

$M (= 12)$ bits are grouped into $l = 2$ blocks (tokens).
Each block or token consists of $n = 3$ sub-blocks each of length $p = 2$ bits since it is working in (Galois Field) $GF(2^p)$ where $p = 2$ as per choice.

**Construction of**

Token $M_2$
\[\begin{array}{c}
1 1 1 0 1 0 \\
1 1 1 1
\end{array}\]

Token $M_1$
\[\begin{array}{c}
1 1 1 0 1 0 \\
3 2 2
\end{array}\]

Unique hash address is obtained from $T$ and $F$ for $M_1$ as
\[H_1 = \begin{bmatrix}
1 0 \\
2 \\
1
\end{bmatrix}\]

**Construction of equivalent**

$T_0 = \begin{bmatrix}
0 0 0 \\
3 0 0 \\
0 2 0
\end{bmatrix}$

$T_0 \cdot F_0 = \begin{bmatrix}
0 0 0 1 \\
3 0 0 3 \\
0 2 0 2
\end{bmatrix}$

$T = \begin{bmatrix}
3 1 0 \\
1 1 2 \\
0 3 2
\end{bmatrix}$

$T \cdot F = \begin{bmatrix}
3 1 0 1 \\
1 1 2 3 \\
0 3 2 1
\end{bmatrix}$

$P_2 = H_1 \oplus P_1$
\[\begin{array}{c}
0 1 1 1 0 0 \\
1 3 0
\end{array}\]

Private or secret key $P$ of length 6 bits is assigned to $P_1$
\[\begin{array}{c}
0 1 1 1 1 0 \\
1 3 2
\end{array}\]

Construction of initial SACA ($T_0$) and SACA ($T_0, F_0$) pair

Figure 4.8: Block diagram of message digest generating hash
Avalanche Effect

**Attack**: Let $M$ be an arbitrary message while $M'$ be another data stream derived out of $M$ by flipping a randomly chosen bit of $M$. The corresponding hash digests are $C_K(M)$ and $C_K(M')$, if there are equal number of zeros and ones in the output difference of $C_K(M)$ and $C_K(M')$. That is, flipping one bit of a randomly chosen message results in a completely different data digest. So it becomes extremely difficult for hacker to get the exact message digest with the change of only one bit in the message text [1, 174].

<table>
<thead>
<tr>
<th>size of file in bytes</th>
<th>No. of ones for CAA $(C_K(M) \oplus C_K(M'))$</th>
<th>No. of ones for MD5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>key-length 128 $p = 4$</td>
<td>key-length 256 $p = 4$</td>
</tr>
<tr>
<td>3239</td>
<td>63</td>
<td>67</td>
</tr>
<tr>
<td>3239</td>
<td>65</td>
<td>67</td>
</tr>
<tr>
<td>65780</td>
<td>65</td>
<td>64</td>
</tr>
<tr>
<td>65780</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>65780</td>
<td>59</td>
<td>64</td>
</tr>
<tr>
<td>259120</td>
<td>70</td>
<td>65</td>
</tr>
<tr>
<td>259120</td>
<td>59</td>
<td>66</td>
</tr>
<tr>
<td>259120</td>
<td>61</td>
<td>64</td>
</tr>
</tbody>
</table>

**Results**: The experiment provides an insight regarding the nature of the output string $H (= C_K(M) \oplus C_K(M'))$, signifying the difference between the hash digest as noted in table 4.3. Column 2 - 5 shows the number of bits having one in the output digest $H$ for two different key length (128, 256) and two different values of $p$ (4, 8). It is seen that the number of ones in $H$ is $\approx \frac{1}{2}$ (key length). Thus, the tabulated results establish the fact that there are equal number of zeros and ones in the output difference of $C_K(M)$ and $C_K(M')$.

**Related-Key Cryptanalysis**

**Attack**: Let $M$ be an arbitrary message and $K$ be a secret key while $K'$ is another secret key derived out of $K$ by flipping a randomly chosen bit of $K$. The corresponding data digests are $C_K(M)$ and $C_{K'}(M)$. If the difference between $C_K(M)$ and $C_{K'}(M)$ ($H = C_K(M) \oplus C_{K'}(M)$) has almost same number of zeros and ones, then it can be concluded that flipping a bit of secret key $K$ results in a completely different data digest which makes a hacker’s prediction for exact message digest too difficult
[2, 99, 174].

**Results**: This is exactly what happens when CAA is used as message digest algorithm as can be seen in table 4.4. Column 2 - 5 shows the the number of bits having one in $H$ for two different key lengths (128, 256) and two different values of $p$ (4, 8). It is seen that the number of ones in $H$ is $\approx \frac{1}{2}$ (key length). In both cases, the results become better as the value of $p$ increases.

**Results of MD5**: The MD5 results reported side by side in column 6 of tables 4.3 and 4.4 shows that CAA scheme is at par if not better than MD5 in withstanding both the attacks on input message data and key respectively. We now present CAs capability to withstand a more advanced form of attack - namely differential cryptanalysis [23, 82, 217]. For sake of comparison we also report cryptanalysis results on MD5.

<table>
<thead>
<tr>
<th>input size of file in bytes</th>
<th>No. of ones for CAA $C_K(M) \oplus C_K'(M)$</th>
<th>No. of ones for MD5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>key-length 128</td>
<td>key-length 256</td>
</tr>
<tr>
<td>3239</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>3239</td>
<td>68</td>
<td>64</td>
</tr>
<tr>
<td>65780</td>
<td>57</td>
<td>64</td>
</tr>
<tr>
<td>65780</td>
<td>55</td>
<td>64</td>
</tr>
<tr>
<td>259120</td>
<td>55</td>
<td>62</td>
</tr>
<tr>
<td>259120</td>
<td>61</td>
<td>64</td>
</tr>
<tr>
<td>259120</td>
<td>57</td>
<td>65</td>
</tr>
</tbody>
</table>

**Differential Cryptanalysis**

**Attack**: Differential cryptanalysis is a chosen plain text attack. The attacker is able to select inputs and examine outputs in an attempt to derive the key. For differential cryptanalysis, the attacker will select pair of inputs, to satisfy a particular input difference, expecting that for that input difference value, a particular output difference value will occur with high probability. In an ideal situation, given a particular input message text difference each possible occurrence of an output message digest difference should be equiprobable. Consequently, the standard deviation should be zero. Thus a system is less prone to attack when the standard deviation is small.

**Results**: We perform differential cryptanalysis on our scheme with 50 different
files of different size. For each file, we take five different fixed input differences to get the output probability distributions and the average value of the standard deviations for them is calculated. We also perform the same analysis for MD5. The results of differential cryptanalytic test on different input file size is reported in table 4.5. Column VI of table 4.5 depicts the average mean standard deviation for MD5 whereas other columns show the same for CAA at different $p$ and key length. The result for the current version of CAA (column V of table 4.5) is superior to that of MD5. It is also worth noting the fact that if the percentage of the standard deviation is less than 10% a system can be considered as secured [23]. The CAA achieves standard deviation of 4%. From table 4.5 it can be observed that as key-length and $p$ increases ($p$ is the dimension of Galois field $GF(2^p)$), average standard derivation decreases.

**Entropy Test**

The entropy test measures the probability of a given message in the cipher text. In a secure cryptosystem, the probability of occurrence of any two given messages of equal length is equal [49].

The Entropy test [49] is carried out in the following way. Blocks of 16 bits from 256 bits key length are selected at random. 1000 different such blocks are sampled out. If the equiprobability of messages are maintained then entropy test will yield value equal to 16.

The last row, 3rd column of table 4.6 clearly shows the superiority of our scheme. The entropy measurement in our scheme is closest to 16 than others thus indicating a higher level of security.
The Padding Attack

When an one-way hash function which is not keyed takes part in message digest generation then a secret key is appended or, padded with the original message. For example MD5 is such a one-way hash function which is not a keyed one-way hash function. Such hash functions are prone to attacks due to this padding, detailed description of this is found in [162]. This type of attack is not possible for the proposed scheme as it employs a keyed hash function where the key is not a part of the original message.

4.4.3 Execution Time

Comparative results for MD5 and CAA are reported in respect of CPU time in table 4.7. These experimental results establish the higher speed of execution of CAA scheme based on GF(2^p) SACA. Higher value of p leads to reduction of computation time because rather than handling np x np matrix with GF(2) elements, we deal with n x n matrix with GF(2^p) elements. In software implementation, the speed is almost one and a half times less than that of MD5 at p=8. The throughput of the hardwired implementation of scheme is of the order of tens of Gigabytes/sec.

Table 4.6: Comparison Regarding Entropy

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Key length in bits</th>
<th>Entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>128</td>
<td>15.93</td>
</tr>
<tr>
<td>SHA</td>
<td>160</td>
<td>-</td>
</tr>
<tr>
<td>SACA at GF(2)</td>
<td>128</td>
<td>15.94</td>
</tr>
<tr>
<td>Proposed SACA at GF(2^p)</td>
<td>256</td>
<td>15.97</td>
</tr>
</tbody>
</table>

Table 4.7: Comparative Performance Regarding Speed in WindowsNT 4.00-1381,IBM

<table>
<thead>
<tr>
<th>Data in Bytes</th>
<th>CPU Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MD5 method</td>
</tr>
<tr>
<td>1608</td>
<td>0.055</td>
</tr>
<tr>
<td>35860</td>
<td>0.147</td>
</tr>
<tr>
<td>65780</td>
<td>0.166</td>
</tr>
<tr>
<td>142164</td>
<td>0.2053</td>
</tr>
<tr>
<td>259120</td>
<td>0.271</td>
</tr>
<tr>
<td>852984</td>
<td>0.294</td>
</tr>
</tbody>
</table>
4.5 Conclusion

This chapter describes $GF(2^p)$ CA based keyed one-way hash function and message authentication employing that function. Detailed cryptanalysis and other tests confirm the superiority of this CA-based keyed one-way hash function over MD5 and the $GF(2)$ CA based keyed one-way hash function proposed in [33, 55] both in terms of security and execution speed.

This secure message authentication along with the CA based hash function is now exploited in the field of image watermarking for image authentication in the next chapter.