s-hash: spectral hash

Abstract

We describe a new class of hash family using the discrete Fourier transform and convolution property. The method yields efficient and highly parallel architectures for hashing.

Index Terms

Hashing, SHA, DFT, number theoretical transforms.

Contents

I. Introduction

In this document we describe the spectral hashing (s-hash) algorithm. We start with mathematical background and corresponding arithmetic followed by a design rationale and the detailed description. In Section .. we turn our attention to the security considerations and present discussions on building blocks and s-hash immunity against known attacks. We conclude with the acknowledgements, the references and the list of annexes.

Patent Statement: s-hash or any of its implementations is not and will not be subject to patents.

II. Mathematical background

A. Finite Fields

Abstractly, a finite field consists of a finite set of objects together with two binary operations (addition and multiplication) that can be performed on pairs of field elements. These binary operations must satisfy certain compatibility properties. There is a finite field containing q field elements if and only if q is a power of a prime number, and in fact for each such q there is precisely one finite field denoted by $GF(q)$. When q is prime the finite field is called a prime field whereas if $q = p^k$ for a prime p and $k > 1$, the finite field $GF(p^k)$ is called an extension field. The number p is named as the characteristic of the finite field and in case of $p = 2$, the extension field is called a binary extension field.

S-hash makes use of both binary extension and prime fields. Prime fields, a topic from elementary school classes are more familiar because of being isomorphic to the well-known structure $\mathbb{Z}_p$ (i.e., the integer ring modulo prime p). On the other hand, binary extension fields may need more attention. Binary extension fields $GF(2^k)$ can be represented by the set of polynomials with polynomial addition and multiplication modulo an irreducible polynomial $f(t)$ over $GF(2)$ having degree k. The degree of the polynomial $f(t)$ is also referenced as the degree of the extension. In fact, the defining polynomial $f(t)$ characterizes the structure of the mathematical object consist of the polynomial congruent classes. If the ”irreducibility” condition on the defining polynomial $f(t)$ is dropped, we would be dealing with a structure called quotient polynomial ring in which some elements (also called zero divisors) might live without inverses. Actually a finite field is a quotient ring having no zero divisors.

Being a quotient polynomial ring, the arithmetic in these fields is the familiar modular polynomial arithmetic. Since characteristic $p = 2$, addition is performed by adding polynomials modulo two whereas multiplication involves a polynomial multiplication and a reduction with respect to the defining irreducible polynomial $f(t)$.

B. Discrete Fourier Transform (DFT)

Spectral techniques are widely accepted and used in the field of digital signal processing (see [**?**], [1], [2] and [**?**]). Most existing publications introduce the subject over the field of complex numbers. In order to present the spectral hashing properly, we need to grasp the basics of spectral theory over the finite fields (or in a more general setting over finite rings) as it is proposed by Pollard [3]. In this sense, we define the DFT as a map from a time ring to a Fourier ring after introducing these structures.

Definition 1: Let d and n be positive integers and ω be a primitive d-th root of unity in $\mathbb{Z}_n$. We define the **time ring** as a factor ring

$$\mathcal{T}_n^d = \mathbb{Z}_n[t]/ < f(t) >$$

where

$$f(t) = \Pi_{i=0}^{d-1}(t - \omega^i) \pmod n$$

Definition 2: The set $\mathcal{F}^d = \oplus_{i=0}^{d-1}\mathbb{Z}_n$ of ordered d-tuples $(X_0, X_1, \ldots, X_{d-1})$ where $X_i \in \mathbb{Z}_n$ forms a ring with component-wise addition and multiplication (also called direct sum of rings). For notation purposes, we denote these d-tuples with polynomials (i.e. $(X_1, X_2, \ldots, X_d)$ will be written as $X_0 + X_1 t + \ldots + X_{d-1}t^{d-1}$). We named the ring $\mathcal{F}^d$ as the **Fourier ring** over $\mathbb{Z}_n$; moreover the elements are called **spectral polynomials** having **spectral coefficients**.

Remark 1: Since the arithmetic is always modulo n, we add the a n subscript to our notation and denote the Fourier ring by $\mathcal{F}_n^d$.

Now we can define the DFT map.

Definition 3: Assume that $\mathcal{T}_n^d$ and $\mathcal{F}_n^d$ are time and Fourier rings over $\mathbb{Z}_n$ respectively. Let ω be a primitive d-th root of unity in $\mathbb{Z}_n$. The DFT map over $\mathbb{Z}_q$ is an invertible set map

$$\begin{array}{rccc} DFT_d^\omega : & \mathcal{T}_n^d & \rightarrow & \mathcal{F}_n^d \\ & x(t) & \mapsto & X(t) \end{array}$$

defined as follows

$$X_i = DFT_d^\omega(x(t)) := \sum_{j=0}^{d-1} x_j \omega^{ij} \bmod n \tag{1}$$

with the inverse

$$x_i = IDFT_d^\omega(X(t)) := d^{-1} \cdot \sum_{j=0}^{d-1} X_j \omega^{-ij} \bmod n \tag{2}$$

for $i = 0, 1, \ldots, d-1$. Moreover, we write

$$x(t) \quad \xleftrightarrow{DFT} \quad X(t)$$

and say $x(t)$ and $X(t)$ are transform pairs where $x(t)$ is called a **time polynomial** and sometimes $X(t)$ is named as the **spectrum** of $x(t)$.

In the literature, DFT over a finite ring spectrum (1) is also known as the *Number Theoretical Transform (NTT)*. Moreover, if q has some special form such as a Mersenne or a Fermat number, the transform named after this form; i.e., *Mersenne Number Transform (MNT)* or *Fermat Number Transform (FNT)*.

Note that, unlike the DFT over the complex numbers, the existence of DFT over finite rings is not trivial. In fact, Pollard [3] mentions that the existence of primitive root d-th of unity and the inverse of d do not guarantee the existence of a DFT over a ring. He adds that a DFT exists in ring R if and only if each quotient field R/M (where M is maximal ideal) possesses a primitive root of unity. If $R = \mathbb{Z}_q$ is taken, one gets the following corollary;

Corollary 1: There exists a d-point DFT over the ring $\mathbb{Z}_q$ that supports the circular convolution if and only if d divides $p-1$ for every prime p factor of q.

Proof: We sketch the proof given in Chapter 6 of Blahut [1]. Firstly, we cover the case where q is a prime power.

Converse is easier to prove; the DFT length d is invertible in $\mathbb{Z}_q$, if d and q are relatively prime (i.e. $dd^{-1} = 1 + kq$ for some k). Surely, any common factor of d and q must be a factor of 1, which is impossible. Moreover, any element ω having order d relatively prime to q has order that divides the Euler function $\phi(q) = (p-1)p^{m-1}$. Therefore, a d-point DFT does not exist in $\mathbb{Z}_q$ unless d divides $q-1$.

On the other hand, let p be a odd prime ($p = 2$ is trivial) then the non-units in $\mathbb{Z}_q$ forms a cyclic group having order $\phi(q) = (p-1)p^{m-1}$. Let π be the generator of this group and $\omega = \pi^{bp^{m-1}}$ for any b dividing $p-1$. Since non-units in $\mathbb{Z}_q$ is cyclic, ω exists, all remains is to show that the inverse DFT exist;

$$\begin{aligned} d^{-1} \cdot \sum_{j=0}^{d-1} X_j \omega^{-ij} \bmod q &= d^{-1} \cdot \sum_{j=0}^{d-1} \omega^{-ij} \sum_{j'=0}^{d-1} x'_j \omega^{-ij'} \bmod q \\ &= d^{-1} \cdot \sum_{j'=0}^{d-1} x'_j \sum_{j=0}^{d-1} \omega^{-i(j'-j)}. \end{aligned}$$

The sum on i is equal to d if $j' = j$, while if j' is not equal to j, then the geometric series summation becomes $(1 - \omega^{-(j'-j)d})/(1 - \omega^{-(j'-j)})$, which is zero since $j' - j \not\equiv 0 \pmod q$. Therefore,

$$d^{-1} \cdot \sum_{j=0}^{d-1} \omega^{-ij} \bmod q = d^{-1} \cdot \sum_{j=0}^{d-1} x_i (d\delta j j') \bmod q = x_i$$

as desired.

Now, let $q = p_1^{m_1} p_2^{m_2} \ldots p_r^{m_r}$. The use of Chinese remainder theorem guarantees the existence of a d-point DFT in $\mathbb{Z}_q$ if and only if d-point DFT exists in each factor ring, which equivalent to say d divides $p_i - 1$ for all $i = 1, 2, \ldots, r$. ■

An integer ring having $q = 2^v \pm 1$ elements for some positive integer v is the most suitable structure for computations since the modular arithmetic operations for such rings are simplified. Moreover, if the principal root of unity is chosen as a power of 2, spectral coefficients are computed by additions and circular shifts. The rings having $q = 2^v - 1$ and $q = 2^v - 1$ elements are called the *Mersenne rings* and *Fermat rings* respectively. In Table I, we tabulate the DFT parameters of some Fermat and Mersenne rings suitable for spectral hashing.

ring $\mathbb{Z}_q$	prime factors	(ω, NNT length)	
$2^3 - 1$	7	$(2,3)$	$(-2,6)$
$2^4 + 1$	17	$(2,8)$	$(4,4)$
$2^5 - 1$	31	$(2,5)$	$(-2,10)$
$2^7 - 1$	127	$(2,7)$	$(-2,14)$
$2^8 + 1$	257	$(2,16)$	$(4,8)$
$2^{13} - 1$	8191	$(2,13)$	$(-2,26)$
$2^{16} + 1$	65537	$(4,16)$	$(2,32)$

TABLE I
PARAMETERS OF NNT FOR $2^3 - 1 \leqslant q \leqslant 2^{16} + 1$

The Mersenne and Fermat rings are not the only suitable rings for efficient arithmetic. Assume that n and m are positive integers, and m (not necessarily a prime) is a small divisor of n. The integer rings having n/m elements, $Z_{n/m}$, are also quite useful. In the literature, a transform defined over $\mathbb{Z}_{n/m}$ is called a *pseudo number transform (PNT)*. PNTs tailor the rings in a way that larger length transforms are possible. For instance, Corollary 1 states that in the fairly large ring $\mathbb{Z}_{2^{31}+1}$, one can only define a 3-point DFT because $2^{31} + 1 = 3 \cdot 715827883$. However; we can still enjoy the $2^{31} + 1$ arithmetic of a transform of length 62 with $\omega = -2$ if the PNT in the ring $\mathbb{Z}_{715827883}$ is used.

In general, longer length DFTs are utmost importance in many applications. Although, NNTs are extremely fast transforms, they are considered too short for most of the digital signal processing applications. Therefore, they are rarely used in practice.

On the other hand, it is possible to attain transforms having larger lengths over the tabulated rings. One way of doing this is employing principal roots of unity which are not powers of two. For instance; in $\mathbb{Z}_{2^8+1}$, $\omega = 3$ gives a transform of lenght 256. However; every single multiplication with roots of unity needs a full 8-bit multiplication and not tolerable for our purposes. Perhaps the most elegant way of extending the transform lengths is using the multi-dimensional transforms.

Multidimensional Fourier transforms arise naturally from problems that are multidimensional. They are also used in computing one dimensional Fourier transforms. In the following section we introduce the three dimensional transform.

C. Multi-dimensional DFTs

Multi-dimensional transforms can be defined over any field of interest as 1-dimensional transform construction. A more detailed presentation of the subject can be found in books [].

Since we concentrate on 3-dimensioanl DFT transform over $GF(17)$, we explicitly give the transforms as follows

$$\boldsymbol{X_i} = DFT_d^{\omega}(x(t)) := \sum_{j=0}^{d-1} x_j \omega^{ij} \bmod n \tag{3}$$

with the inverse

$$x_i = IDFT_d^{\omega}(X(t)) := d^{-1} \cdot \sum_{j=0}^{d-1} X_j \omega^{-ij} \bmod n \tag{4}$$

for $i = 0, 1, \dots, d-1$. Moreover, we write

Surely, a direct computation of the Equation (3) is the last thing one would like to calculate. Over three decades, many fast algorithms proposed for multi-dimensional transforms. Leaving the discussions on the best algorithm for our purposes for future, we select the small-radix Cooley-Tukey [] for our computations.

III. S-hash Arithmetic

A. The field $GF(17)$

The field $GF(17)$ is the structure where all the FFT computations takes place. Fortunately, only additions and simple shifts have to be computed for FFT calculations. As given in the previous section

are subject to and the operations

First of all we remark that the field $GF(17)$ is small enough to employ look-up tables for its arithmetic. Even two or more operations can be performed in parallel by merging the look-up tables. Since There exists a vast amount of optimizations

First of all we remark that carrying arithmetic in Mersenne rings is equivalent to doing one's complement operations. Arithmetic in Fermat rings are slightly complicated than one's complement arithmetic, when certain encoding techniques are performed.

Since modulus $17 = 2^4 + 1$ the modular reduction corresponds to subtracting the most significant r bits to/from the least significant r bits. This procedure needs a correction in case of this subtraction exceeds the modulus which is carried by a final modular reduction.

The binary representation of all the elements of $GF(17)$ need 5 bits. The additional bit is required only for the number 2^4. In order to alter this redundancy, a modified binary system — diminished-1 — can be employed. where the number x is represented by $x' = x - 1$ and the value 0 is not used or handled separately [**?**]. However, for a field of size $GF(17)$ handling the value 0 might be cumbersome. Moreover, s-hash heavily needs the normal representation of numbers for p-prism permutations, a diminished-1 representation may suffer from the conversion from and to the normal number representation. For those and further discussions on Fermat arithmetic we refer the reader to [**?**] and [**?**].

As a last remark we

Let $x, y \in \mathbb{Z}_{2^v+1}$ with diminished-1 representation x' and y' respectively. Observe that

$$(x+y)' = (x+y) - 1 = (x'+1) + (y'+1) - 1 = x' + y' + 1$$

Hence by using Lemma 1, addition in the Fermat ring can be formulated as

$$x' + y' + 1 \bmod (2^v + 1) = \begin{cases} x' + y' \bmod 2^v & \text{if } x' + y' + 1 \geqslant 2^v \\ x' + y' + 1 & \text{otherwise} \end{cases}$$

which leads us the following important identity

$$x' + y' + 1 \bmod 2^v + 1 = x' + y' + \overline{c}_{out} \bmod 2^v \tag{5}$$

Note that Equation (7) is also valid for normal representations of numbers. Therefore it is possible to do arithmetic by using normal representations (see [**?**]).

Remark 2: For those algorithms heavily use a normal number representation of numbers, a diminished-1 representation suffer from the conversion from and to the normal number representation. In our case we generally do not care about a presentation of a number unless it is the final result.

Likewise Mersenne arithmetic, if the overflows of the regular addition are fed to the adder after an inversion, a Fermat arithmetic is achieved. Observe that the above methodology function well for CPAs and CSAs, moreover these adders are named as modular CPAs and CSAs after above arrangements.

Notation 1: Through out this document, for a gate-unit analysis we assume each two input monotonic gate (e.g., AND, NAND) counts as one gate (area or delay), an XOR as two gates (area or delay), and a full adder has an area of seven gates and a delay of four gates.

If Sklansky parallel-prefix adders are considered for a CPA implementation, the area and time complexity of CPA becomes same for both Fermat and Mersenne rings which is very similar to the standard integer propagate addition (see Table II)

ring	area	delay
Mersenne or Fermat (M/F)	$\frac{3}{2}n\log n + 7n$	$2\log n + 5$

TABLE II
THE COST OF CPA (SKLANSKY PARALLEL-PREFIX ADDER).

B. The Fast Fourier Transform

A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. FFTs are of great importance to a wide variety of applications, from digital signal processing and solving partial differential equations to algorithms for quick multiplication of large integers.

We refer the reader to [1] and [**?**] for excellent presentations of the subject.

C. The field $GF(2^4)$

The field $GF(2^4)$ admits a very rich choice of design and implementation flexibility. It is small enough to employ look up tables suitable for software; to write algebraic equations for simple functions; has type I ONB with which Massey-Omura circuits may be used and nice pseudo transforms may defined.

In practice, for polynomial multiplication, schoolbook or Karatsuba method is used, where the reduction is performed mostly by standard or Montgomery reduction techniques. If $f(t)$ is chosen as a special polynomial such as a trinomial or a pentanomial reduction enjoys some effective optimizations.

$f(t)$ is not an irreducible, then this set forms a binary polynomial ring where some elements (also called zero divisors) live without inverses.

When k is not a prime, the field $GF(2^k)$ is called a *composite field*.

Although composite fields enjoy the simplifications of some desirable features, their usage in practice (particularly in ECC) has some security concerns. In fact, ANSI X9.63 [**?**] explicitly exclude the use of elliptic curves over composite fields. Therefore, we do not discuss the optimizations making use of these special cases and assume k is arbitrary.

The arithmetic in binary extension fields is simply modular polynomial arithmetic; addition is performed by adding polynomials modulo two whereas multiplication is quite expensive in terms of time and area. In a standard setting, multiplication involves a polynomial multiplication and a reduction with respect to the defining irreducible polynomial $f(t)$. In practice, for polynomial multiplication, schoolbook or Karatsuba method is used, where the reduction is performed mostly by standard or Montgomery reduction techniques. If $f(t)$ is chosen as a special polynomial such as a trinomial or a pentanomial reduction enjoys some effective optimizations.

1) Addition: The addition in field $GF(2^4)$ enjoys the modulo 2 arithmetic. In other words,

Example 1: Let $x = t^3 + t + 1$ and $y = t^2 + 1$ be elements of $GF(2^4)$.

$$x + y = (t^3 + t + 1) + (t^2 + 1) = t^3 + t^2 + t$$

In binary notation, one gets $(1011)_2 + (0101)_2 = (1110)_2$ by simply xoring respective bits of x and y.

2) Multiplication: Multiplication can be performed in many different ways, starting from the trivial table look up method, here we present standard, type I ONB and embedded multiplication.

3) Inversion: Likewise multiplication inversion may be implemented in different fashions: table look up method, here we present standard, type I ONB and embedded multiplication.

4) Affine transformation: In order to eliminate the fixed points of the inverse mapping, an affine transform is applied to each entry of the state prism. The affine transform can be formulated as follows:

$$x_{(i,j,k)} := \alpha(m_{(i,j,k)})^{-1} + \gamma$$

where

$$\alpha = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

and

$$\gamma_1 = [0, 1, 0, 1]$$

The constant has been chosen in such a way that that the S-box has no fixed points (S-box(a) = a) and no opposite fixed points' (S-box(a) = a).

The ByteSub Transformation is a non-linear byte substitution, operating on each of the State bytes independently. The substitution table (or S-box) is invertible and is constructed by the composition of two transformations:

Likewise multiplication inversion may be implemented in different fashions: table look up method, here we present standard, type I ONB and embedded multiplication.

D. SRC (shift row & column)

SRC step performs circular shifts both on the rows and columns. Assume that the row and column indexes of the array start from zero, SRC is achieved as follows

- each row of the array shifted to the left as much as its row index
- each column of the array shifted to the upwards as much as its column index

E. Component-wise operations

These operations are simple component-wise operations of arrays. For instance; cINV is simply achieved by inverting the entries of the matrix. Let us formally introduce these freshman dream matrix operations.

cINV (component-wise inversion): Let $A = (a_{ij})$ be an $n \times n$ matrix over the ring $\mathbb{Z}_q$. We define

$$cINV(A) = B = (b_{ij})$$

where

$$b_{ij} \equiv a_{ij}^{-1} \bmod q \quad \text{for } i, j = 0, 1, \ldots, n$$

cMULT (component-wise multiplication): Let $A = (a_{ij})$ and $B = (b_{ij})$ be $n \times n$ matricies over the ring $\mathbb{Z}_q$.

$$cMULT(A, B) = C = (c_{ij})$$

is defined as follows

$$c_{ij} \equiv a_{ij} b_{ij} \bmod q \quad \text{for } i, j = 0, 1, \ldots, n$$

IV. Specral Hashing Algorithm

Before going into the details of the s-hash, we note that spectral hashing adapts the classical Merkle-Damgard scheme for hash generation as originally illustrated by the authors in Figure 1.

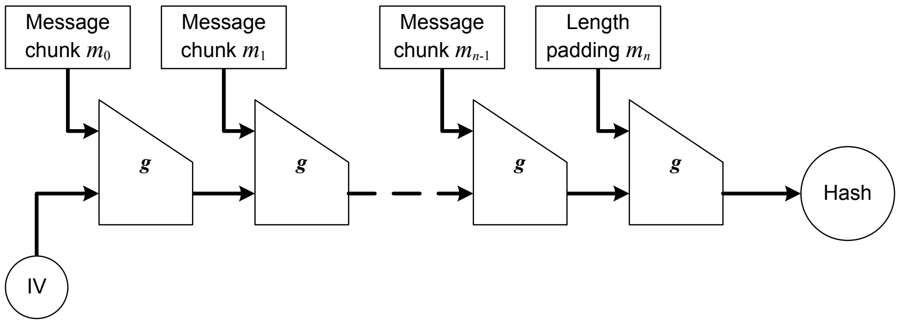

Fig. 1. Classical Merkle-Damgard hash scheme

If one puts the Figure 1 into a formal presentation, the s-hash algorithm is described as follows:

Algorithm 1: Spectral Hashing Algorithm

Input: padded message m (i.e. $m = m_0 m_1 \dots m_{n-1}$ where bit lenght $|m_i| = 512$ for all $i = 0, 1, \dots, n-1$)

Output: h, s-hash of the message m.

```
1: $h_0 := 0$
2: **for** $i = 0$ **to** $n-1$
3:     $h_{i+1} := g(m_i, h_i)$
4: **end for**
5: **return** $h_{i+1}$
```

Clearly, Algorithm 1 is far from being descriptive as its does not reveal the details of the compression function g. Starting from the padding scheme, the details of s-hash are carefully described in the following sections.

A. Initialization

Assume there is a message consists an arbitrary binary string of length l, $l \leqslant 2^{64}$. Since s-hashing operates on 512-bit blocks of message, the initialization includes a padding scheme resulting an extended message having length which is a multiple of 512. The following steps summarizes the padding process:

i. append the bit '1' to the message
ii. append k bits of '0', where k is a non-negative integer such that $l + k + 1 \equiv 448 \bmod 512$.
iii. append the 64-bit big-endian binary representation of l, returning the padded message string m.

S-Hash process the message in successive 512-bit chunks. The padded message string $m = m_0 \dots m_{n-1}$ consist of n chunks where each of m_i for $i = 0, 1, 2, \dots, n-1$ has 512-bit length.

We start with processing chunk m_0; firstly, we break the chunk into 128 words of 4-bits and then map it into an $4 \times 4 \times 8$ array or prism. Formally, let $m_0 = s_0, s_1, \dots, s_{127}$ where s_I is a 4-bit binary number for all I in the index set $\{0, 1, \dots, 127\}$. We reindex the binary string m_0 as follows:

$$S_{(i,j,k)} = s_I$$

where $I = 32i + 8j + k$, for $i, j = 0, 1, 2, 3$ and $k = 0, 1, \dots, 7$. In fact, this re-indexing corresponds to fill an $4 \times 4 \times 8$ prism as shown in Figure 2. We call this array as the state prism (s-prism). In every iteration of Merkle-Damgard scheme this state prism is filled with message chunks and processed throughout s-hash algorithm steps.

Similar to s-prism, we employ another $4 \times 4 \times 8$ array holding a permutation table that is used for setting up the non-linear system of equations of the compression function. This permutation array is called as the permutation prism (p-prism) which is initially configured as

$$P_{(i,j,k)} = I$$

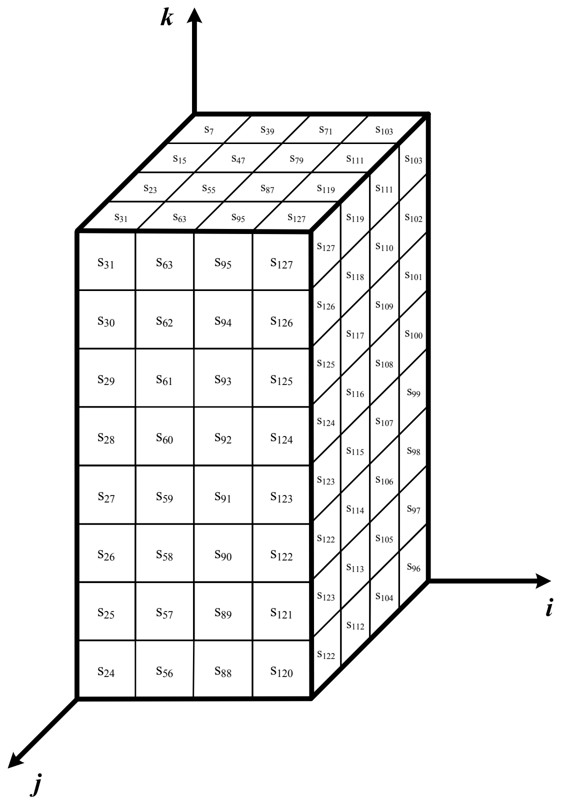

Fig. 2. State Prism

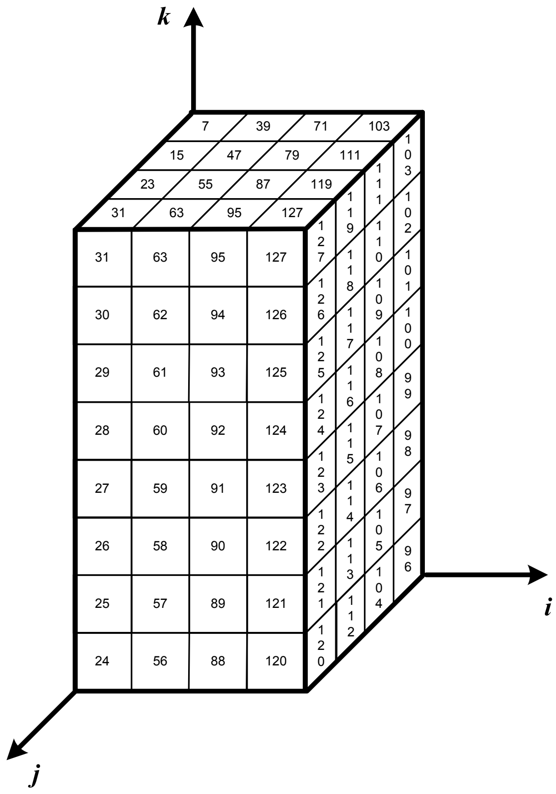

Fig. 3. Permutation Prims

for $I = 0, 1, \ldots, 127$, $i, j = 0, 1, 2, 3$ and $k = 0, 1, \ldots, 7$ (see Figure 3).

During the initialization, after filing up the s-prism with the words of m_0, we modify the p-prism by some simple entry swappings. Since s-prism initially holds 4-bit entries, we apply the following swappings to the p-prism:

$$P_{(i,j,k)} = P_{(Sh_{(i,j,k)}, Sl_{(i,j,k)}, k)}$$

where $Sl_{(i,j,k)} = S_{(i,j,k)} \bmod 4$ and $Sh_{(i,j,k)} = S_{(i,j,k)}$ div 4 for all $i, j = 0, 1, 2, 3$ and $k = 0, 1, \ldots, 7$. In other words, $Sl_{(i,j,k)}$ and $Sh_{(i,j,k)}$ represents the least and most significant two bits of the s-prism entry $S_{(i,j,k)}$ respectively.

In the rest of the algorithm we modify both of the state and permutation prism. In certain stages we combine these two parallel altered prisms.

B. *Compression function*

The compression function g of s-hash consists of three stages:

- Affine transformation
- Discrete Fourier transformation
- Non-linear system transformation

Observe that even the conventions of all three stages mainly describes the acts on the s-prism; the transformations on the p-prism are performed using the outcomes of the s-prism through these stages.

Therefore, before going into the details, we sketch how the compression function works. In Figure 4, the column on the left illustrates the consecutive transformations applied to s-prism. On the other hand, the modifications on the p-prism is controlled by the intermediate entries of the s-prism derived from the outputs of these stages. In total p-prism goes through 5 different type of swappings and some rotations shown in the figure.

We start with describing the affine transform made up of a non-linear iterative contraction vector transformation over the field $GF(2^4)$.

Affine transformation: The following affine transform is applied to each entry of the s-prism

$$s_{(i,j,k)} := \alpha(s_{(i,j,k)})^{-1} + \gamma \tag{6}$$

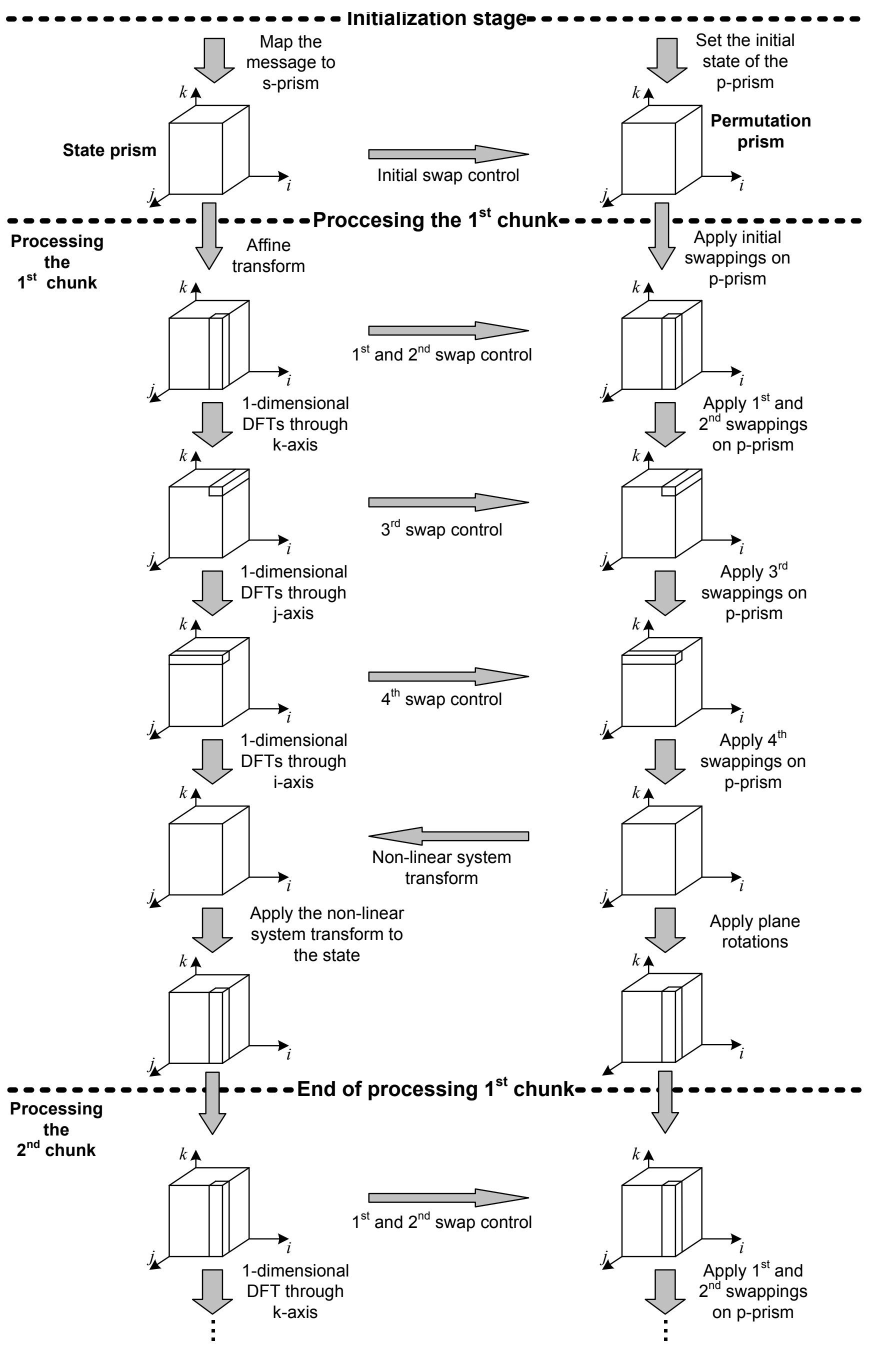
Initialization stage
Map the message to s-prism
Set the initial state of the p-prism
State prism
Permutation prism
Initial swap control
Proccesing the 1st chunk
Processing the 1st chunk
Affine transform
Apply initial swappings on p-prism
1st and 2nd swap control
1-dimensional DFTs through k-axis
Apply 1st and 2nd swappings on p-prism
3rd swap control
1-dimensional DFTs through j-axis
Apply 3rd swappings on p-prism
4th swap control
1-dimensional DFTs through i-axis
Apply 4th swappings on p-prism
Non-linear system transform
Apply the non-linear system transform to the state
Apply plane rotations
End of processing 1st chunk
Processing the 2nd chunk
1st and 2nd swap control
1-dimensional DFT through k-axis
Apply 1st and 2nd swappings on p-prism
k
j
i

Fig. 4. Swap control planes (sc-planes) on the s-prism

where

$$\alpha = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \text{ and } \gamma = [0, 1, 1, 1].$$

One can clearly see that the foremost component of the affine transform (8) is the inverse map which has the best known non-linearity [4] in the field $GF(2^4)$. Favorably, AES substitution box uses the inverse map in $GF(2^8)$ as its main non-linear component against differential and linear attacks. Therefore, the inverse map on $GF(2^4)$ is taken as a nice behaving function to build immunity for known successful attacks. Moreover, in order to eliminate the weaknesses related to the fixed points, in s-hash design, the inverse map is coped with a constant scaling and a linear iteration.

DFT map: We simply apply the 3-dimensinal DFT to the s-prism. Since some intermediate DFT calculations are needed for p-prism modifications, s-hash may not allow to practice all kinds of fast DFT methods. In particular, p-prism swappings use the intermediate values of the standard row-column method successively applied through k, j and i axis as seen in Figure 5.

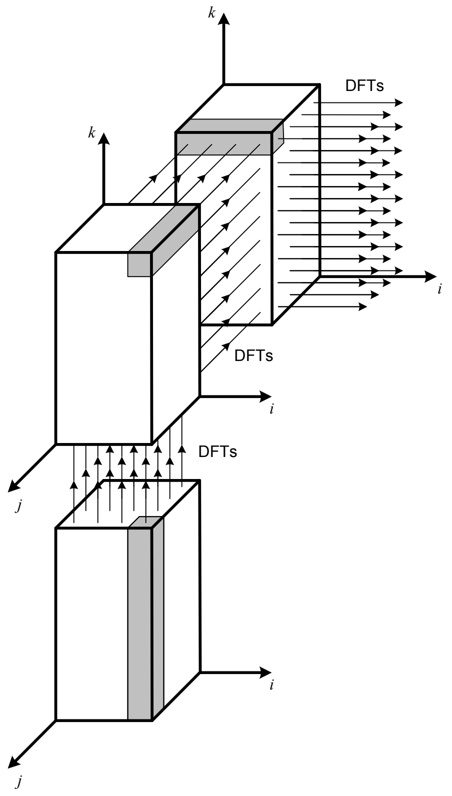

Fig. 5. Three dimensional DFT can be realized by row-column method: applying 1-dimensional DFTs to the state prism through k, j and i axis successively

As described in the previous sections, DFT is defined over the prime field $GF(17)$, permitting a transform having length 8 and 4 for the principle root of unities $\omega = 2$ and $\omega = 4$ respectively. Moreover, observe that in the first iteration of the row-column method (i.e. DFT through k-axis) one has to compute 16 different 1-dimensional 8-point DFTs. On the other hand, through i and j axis, we need to calculate 32 different 4-point DFTs for each axis.

Unlike DFT revision of s-prism, p-prism modifications are a little bit more complicated. We perform data (s-prism) depended swappings on the p-prism similar to those done during the initialization stage.

The first modifications after the initial refinement on the p-prism is performed using swap control plane (sc-plane) of the s-prism. The 1st sc-plane is a 4×4 array extracted by xoring the two planes (matrices) perpendicular to the k-axis, particularly, the two having k co-ordinates equal to 0 modulo 4 (see Figure 6). If explicitly written, the following gives the swappings for all $i, j = 0, 1, 2, 3$:

$$\begin{aligned}
&\text{if } ((S_{(i,j,0)}[0] \oplus S_{(i,j,4)}[0]) = 0) \quad \text{then,} \quad \text{swap } (P_{(i,j,0)}, P_{(i,j,7)}),\\
&\text{if } ((S_{(i,j,0)}[1] \oplus S_{(i,j,4)}[1]) = 0) \quad \text{then,} \quad \text{swap } (P_{(i,j,1)}, P_{(i,j,6)}),\\
&\text{if } ((S_{(i,j,0)}[2] \oplus S_{(i,j,4)}[2]) = 0) \quad \text{then,} \quad \text{swap } (P_{(i,j,2)}, P_{(i,j,5)}),\\
&\text{if } ((S_{(i,j,0)}[3] \oplus S_{(i,j,4)}[3]) = 0) \quad \text{then,} \quad \text{swap } (P_{(i,j,3)}, P_{(i,j,4)}).
\end{aligned}$$

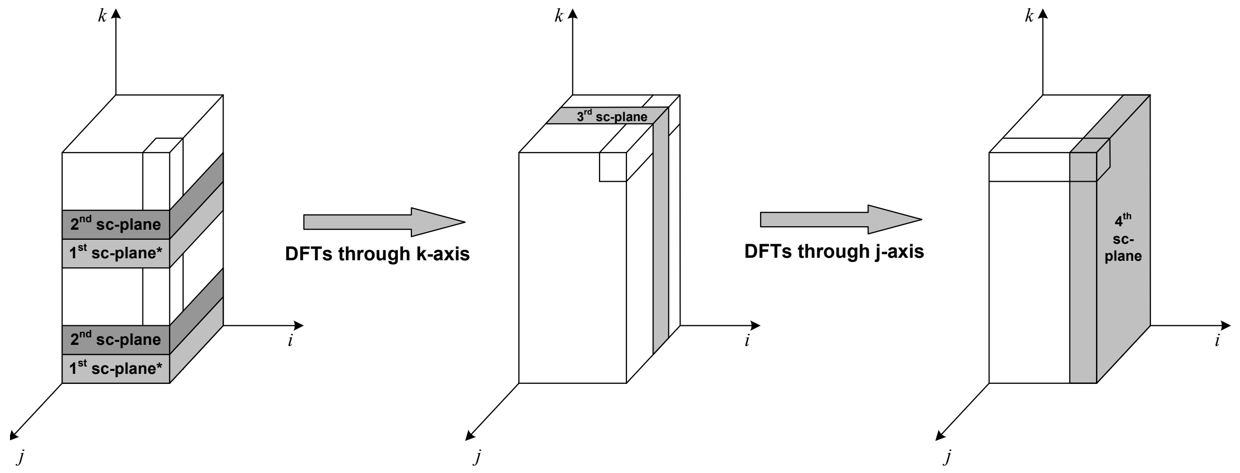

Fig. 6. Swap control planes (sc-planes) on the s-prism. (*) 1st sc-plane is computed by XORing these two)

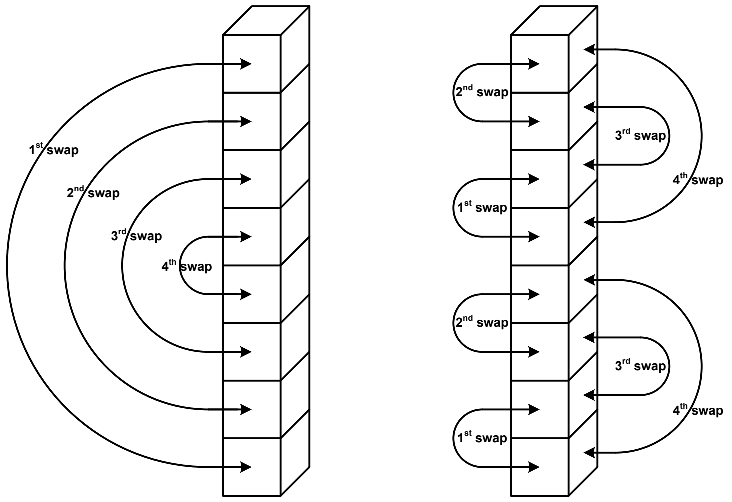

Fig. 7. 1st and 2nd swappings on the vectors of p-plane

On the other hand, the 2nd sc-plane consists of two 4×4 arrays controlling the upper and lower halves of the p-prism. Once again these two planes perpendicular to the k-axis with $k \equiv 1 \bmod 4$ as seen in Figure 6. The lower half is controlled by the lower array of the 2nd sc-plane as follows:

$$\begin{array}{lll} \text{if } (S_{(i,j,1)}[0]=0) & \text{then,} & \text{swap } (P_{(i,j,0)}, P_{(i,j,1)}), \\ \text{if } (S_{(i,j,1)}[1]=0) & \text{then,} & \text{swap } (P_{(i,j,2)}, P_{(i,j,3)}), \\ \text{if } (S_{(i,j,1)}[2]=0) & \text{then,} & \text{swap } (P_{(i,j,1)}, P_{(i,j,2)}), \\ \text{if } (S_{(i,j,1)}[3]=0) & \text{then,} & \text{swap } (P_{(i,j,0)}, P_{(i,j,3)}), \end{array}$$

where the upper half is controlled by the upper plane of the s-prism:

$$\begin{array}{lll} \text{if } (S_{(i,j,5)}[0]=0) & \text{then,} & \text{swap } (P_{(i,j,4)}, P_{(i,j,5)}), \\ \text{if } (S_{(i,j,5)}[1]=0) & \text{then,} & \text{swap } (P_{(i,j,6)}, P_{(i,j,7)}), \\ \text{if } (S_{(i,j,5)}[2]=0) & \text{then,} & \text{swap } (P_{(i,j,5)}, P_{(i,j,6)}), \\ \text{if } (S_{(i,j,5)}[3]=0) & \text{then,} & \text{swap } (P_{(i,j,4)}, P_{(i,j,7)}), \end{array}$$

Notice that 1st and 2nd sc-planes controls the swappings of 16 vectors on the p-prism. Each of these vectors has 8 entries and are parallel to the k-axis.

The 3rd and 4th sc-planes are obtained from the s-prism after the 1-dimensional DFT calculations through k and j axis respectively. In fact, 3rd sc-plane is chosen as $S_{(i,2,k)}$ where 4th sc-plane is $S_{(i,j,3)}$.

Although the input values for DFT belongs to the binary field $GF(2^4)$ (i.e. values represented by at least 4-bits), DFT operates over the prime field $GF(17)$ and might return 5 bit entries. We select the least significant 4-bits of the sc-plane entries as the swap control bits. Moreover, in order to balance distribution of the swapping, we involve the index in the calculations. To be more concrete, the following swapping is applied to the p-prism for all $i = 0, 1, 2, 3$ and $k = 0, 1, \ldots, 7$:

$$\begin{array}{lll} \text{if } ((S_{(i,2,k)}[0] \oplus k[0]) = 0) & \text{then,} & \text{swap } (P_{(i,0,k)}, P_{(i,1,k)}), \\ \text{if } ((S_{(i,2,k)}[1] \oplus k[1]) = 0) & \text{then,} & \text{swap } (P_{(i,2,k)}, P_{(i,3,k)}), \\ \text{if } ((S_{(i,2,k)}[2] \oplus k[2]) = 0) & \text{then,} & \text{swap } (P_{(i,1,k)}, P_{(i,2,k)}), \\ \text{if } ((S_{(i,2,k)}[3] \oplus i[0]) = 0) & \text{then,} & \text{swap } (P_{(i,0,k)}, P_{(i,3,k)}), \end{array}$$

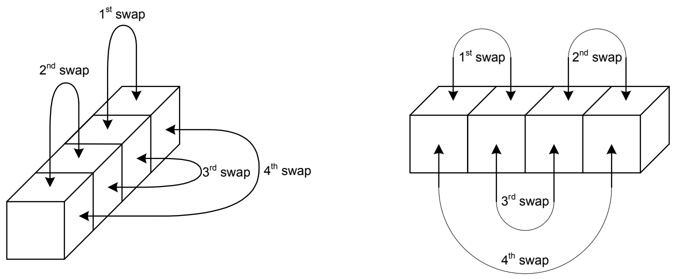

Fig. 8. 3rd and 4th swappings on the vectors of p-plane

Similarly, the following swapping is applied to the p-prism using the 4th sc-plane for all $i = 0, 1, 2, 3$ and $k = 0, 1, \ldots, 7$:

$$\begin{array}{lll} \text{if } ((S_{(3,j,k)}[0] \oplus k[0]) = 0) & \text{then,} & \text{swap}(P_{(0,j,k)}, P_{(1,j,k)}), \\ \text{if } ((S_{(3,j,k)}[1] \oplus k[1]) = 0) & \text{then,} & \text{swap}(P_{(2,j,k)}, P_{(3,j,k)}), \\ \text{if } ((S_{(3,j,k)}[2] \oplus k[2]) = 0) & \text{then,} & \text{swap}(P_{(1,j,k)}, P_{(2,j,k)}), \\ \text{if } ((S_{(3,j,k)}[3] \oplus i[0]) = 0) & \text{then,} & \text{swap}(P_{(0,j,k)}, P_{(3,j,k)}), \end{array}$$

Non-linear system transform: At this step of the compression function we collect and combine the data from s-prism and p-prim to setup a non-linear system of equations. The non-linear system transform is especially designed against pre-image attacks and related weaknesses. We further discuss security related issues in Section **??**.

Non-linear system transform applies the following map to each entry of the s-prism.

$$S_{(i,j,k)} := (S'_{(i,j,k)} + Pl_{(i,j,k)})^{-1} + (S'_{P_{(i,j,k)}} + Ph_{(i,j,k)})^{-1} + H_{(i,j,k)} \tag{7}$$

for all $i, j = 0, 1, 2, 3$ and $k = 0, 1, \ldots, 7$. Clearly, Equation () would be more clear after explicitly describe the objects $S'_{(i,j,k)}$, $Pl_{(i,j,k)}$ and $Ph_{(i,j,k)}$.

As we discussed earlier, DFT operates over the prime field $GF(17)$ and outcomes of DFT might be 5-bit entries. We assign the least significant 4-bits of the s-plane entries S'. In other words;

$$S'_{(i,j,k)} = S_{(i,j,k)} \bmod 2^4, \quad \text{for } i, j = 0, 1, 2, 3 \text{ and } k = 0, 1, \ldots, 7$$

Similary, we pick the least significant 4-bits of p-prism entries and assing them to Pl;

$$Pl_{(i,j,k)} = P_{(i,j,k)} \bmod 2^4$$

On the other hand, Ph is the concetination of the 5th bit of S and remaining 3-bits of P (recall that entries of p-prism are 7-bit numbers) where the symbol ”||” stands for concetination of bit strings.

$$Ph_{(i,j,k)} = (S_{(i,j,k)} \text{ div } 16) \mid\mid (P_{(i,j,k)} \text{ div } 2^4)$$

If the message consists of a single chunk, the hash value is deduced from the s-prism at this point. Otherwise, s-hash algorithm behaves according to Merkle-Damgard scheme. S-prism goes into the successive round as the previous round outcome, i.e. $H_{(i,j,k)}$ which initially equals to zero. On the other hand, p-prism goes through some rotations (called rubics rotations, see Figure 9) described as follows:

$$\begin{array}{lll}
\text{if } (k \equiv 0 \bmod 4) & \text{then} & P_{(i,j,k)} := P_{(i,j,k)}, \\
\text{if } (k \equiv 1 \bmod 4) & \text{then} & P_{(i,j,k)} := P_{(4-j,i,k)}, \\
\text{if } (k \equiv 2 \bmod 4) & \text{then} & P_{(i,j,k)} := P_{(j,i,k)}, \\
\text{if } (k \equiv 3 \bmod 4) & \text{then} & P_{(i,j,k)} := P_{(j,4-i,k)},
\end{array}$$

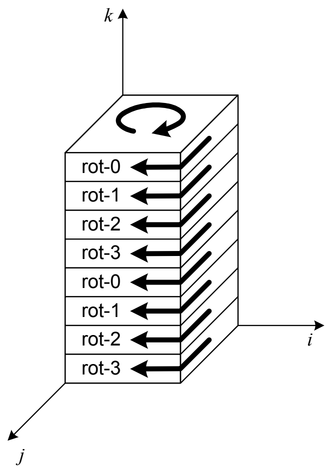

Fig. 9. Rubics rotations on p-prism

Algorithm 2: Compression function
Input: a message chunk $s_0, s_1, \ldots, s_{127}$ and h_i
Output: h_{i+1}

```
1:  $S_{(i,j,k)} = s_I$, map message chunk into s-prism
2:  Affine-trans($S_{(i,j,k)}$), apply affine transform to the s-prism
2:  swap($P_{(i,j,k)}$), apply affine transform to the s-prism
3:  Affine-trans($S_{(i,j,k)}$), apply affine transform to the s-prism
12: return $H_i$
```

Observe that At the end of the

Algorithm 3: . g function
Input: m_i, m_{i+1} and h_{i-1}
Output: s-hash of the message

```
1:  $x_i := \alpha(m_i)^{-1} + \gamma$
2:  $X_i := DFT(x_i)$
3:  $H_i := (X_i + \sigma_H(i))^{-1} + (X_{\sigma(i)} + \sigma_L(i))^{-1} + h_{i-1}$
12: return $H_i$
```

C. Hash generation

Spectral hashing algorithm can be configured to return hash values which are multiples of 32-bits in between 128 and 512. These lengths clearly includes the bit sizes 224, 256, 384 and 512.

The procedure is quite simple; applied whenever the final states of the s-prism and p-prism are reached. In other words, desired hash string is generated after s-prism goes through the non-linear system transform and p-prism is modified via the rubics rotations at the end of the final chunk's processing.

The bits of the hash value are selected from the s-prism entries determined by the s-hash generation table (sg-table). The sg-table is a co-ordinate matrix with rows showing the bit positions on $S_{(i,j,k)}$ and columns pointing the least significant two bits of $P_{(i,j,k)}$. A plotted star on the sg-table means that "assign the corresponding bit on all the $S_{(i,j,k)}$ entries to the hash value if the least significant two bits of the corresponding $P_{(i,j,k)}$ coincides with the selected column". For instance, Table III states that if the two least significant bits of $P_{(i,j,k)}$ is "00" (observe that 32 such entry exist) then the 0th bit of $S_{(i,j,k)}$ has to be assigned to the hash value.

$P_{(i,j,k)}[1:0]$		00	01	10	11
bit position on $S_{(i,j,k)}$	3				*
	2			*	
	1		*		
	0	*			

TABLE III
128-BIT HASH GENERATION.

Notice that Table III presents a generation of $32 * 4 = 128$-bit hash value. In fact, every cell (= bit) selection on the sg-table adds extra 32-bits to the assigned hash bits. Therefore by adding more stars to the sg-table, one can generate longer hash values obviously bounded by 512-bits (i.e. the s-prism itself).

For instance; 224-bit hash generation needs $224/32 = 7$ stars to place. With the following description, we fixed the star placing process.

While counting the number of stars, on every count we put a single star on the sg-table, two stars in the same hole is not allowed. We place the stars through the diagonals starting from main diagonal as seen in 128-bit sg-table (i.e. Table III). We put the stars on a diagonal from the least to the most significant cells (defined with $S_{(i,j,k)}$ bits). When the holes on one diagonal finishes we continue with filling the longest diagonal parallel to it. If two such diagonals exist we pick the one on top first.

According to the above generation process, one can have the sg-tables for 244, 256, 384 and 512-bit in Tables 2,3,4 and 5 respectively.

$P_{(i,j,k)}[1:0]$		00	01	10	11
bit position on $S_{(i,j,k)}$	3			*	*
	2		*	*	
	1	*	*		
	0	*			

TABLE IV
224-BIT HASH GENERATION.

$P_{(i,j,k)}[1:0]$		00	01	10	11
bit position on $S_{(i,j,k)}$	3			*	*
	2		*	*	
	1	*	*		
	0	*	*		

TABLE V
256-BIT HASH GENERATION.

$P_{(i,j,k)}[1:0]$		00	01	10	11
bit	3		*	*	*
position	2	*	*	*	*
on	1	*	*	*	
$S_{(i,j,k)}$	0	*	*		

TABLE VI

384-BIT HASH GENERATION.

$P_{(i,j,k)}[1:0]$		00	01	10	11
bit	3	*	*	*	*
position	2	*	*	*	*
on	1	*	*	*	*
$S_{(i,j,k)}$	0	*	*	*	*

TABLE VII

512-BIT HASH GENERATION.

After the selection the resulting s-prism is called a punctured s-prism looks like a swiss cheese. The final hash value is simply deduced from the final state of the punctured s-prism by reversing the message map indexing. Formally, the hash string

$$h := h_n = H_0 H_1 \ldots H_{127}$$

consists of the 4-bit words $H_I = S_{(i,j,k)}$, where $I = 32i + 8j + k$ for $i, j = 0, 1, 2, 3$ and $k = 0, 1, \ldots, 7$.

1	1	1
2	2	2

1	1	1
2	2	2

V. Security Considerations

Finding a differential path on spectral hash is infeasible. When one goes through its steps, it can be easily seen that the first step is bijective. In addition, almost every bit of the input to the DFT step affects the output which means even a small number of bit change creates a large amount of propagation. The DFT step precludes the search for differential paths.

The usage of the inverse function at steps 1 and 3 are for its non-linearity. Inverse function in the field $GF(2^4)$ has non-linearity property.(reference) Thus, it is hard to approximate it by linear equations and creates resistance against linear cryptanalysis.

The internal states of spectral hash are bijective transformations. Going back through step 3 to produce an ”internal hash value” from a pre-specified hash values is hard because of the existence of the data dependent permutation in step 3. Thus, finding a matching internal state is not possible to construct a collision due to the fact that constructing different internal states requires finding inverses of different permutations each of which is specified by a different message initially unknown to the adversary.

Moreover, the specification of data dependent permutation from a given message shows uniform distribution; which means each different 512-bit length message block generates a different permutation. As a result the probability of finding a pre-specified message from the hash is $1/2^n$, where n is the number of bits of the output of the Spectral Hash.

Concluding, we conjecture that Spectral Hash is resistant to known attacks and it is not possible to find a collision under the complexity bound $O(2^{(n/2)})$ required for birthday attack. It has pre-image resistance with complexity $O(2^n)$. Spectral hash also admits a random distribution which makes it a suitable candidate for an ideal cryptographic hash function.

VI. Conclusions

We proposed new techniques

References

[1] R. E. Blahut, *Fast Algorithms for Digital Signal Processing*, Addison-Wesley publishing Company, 1985.
[2] J. E. Hopcroft A. V. Aho and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley publishing Company, 1974.
[3] J. M. Pollard, “Implementation of number theoretic transform,” *Electronics Letters*, vol. 12, no. 15, pp. 378–379, July 1976.
[4] K. Nyberg, “Differentially uniform mappings for cryptography,” in *Advances in Cryptology, Proceedings Eurocrypt’93, LNCS 765, T. Helleseth, Ed., Springer-Verlag*, 1994, pp. 55–64.