# Spectral Hash

Çetin Kaya Koç

koc@cs.ucsb.edu
http://cs.ucsb.edu/~koc/shash

College of Creative Studies &
Department of Computer Science
University of California Santa Barbara

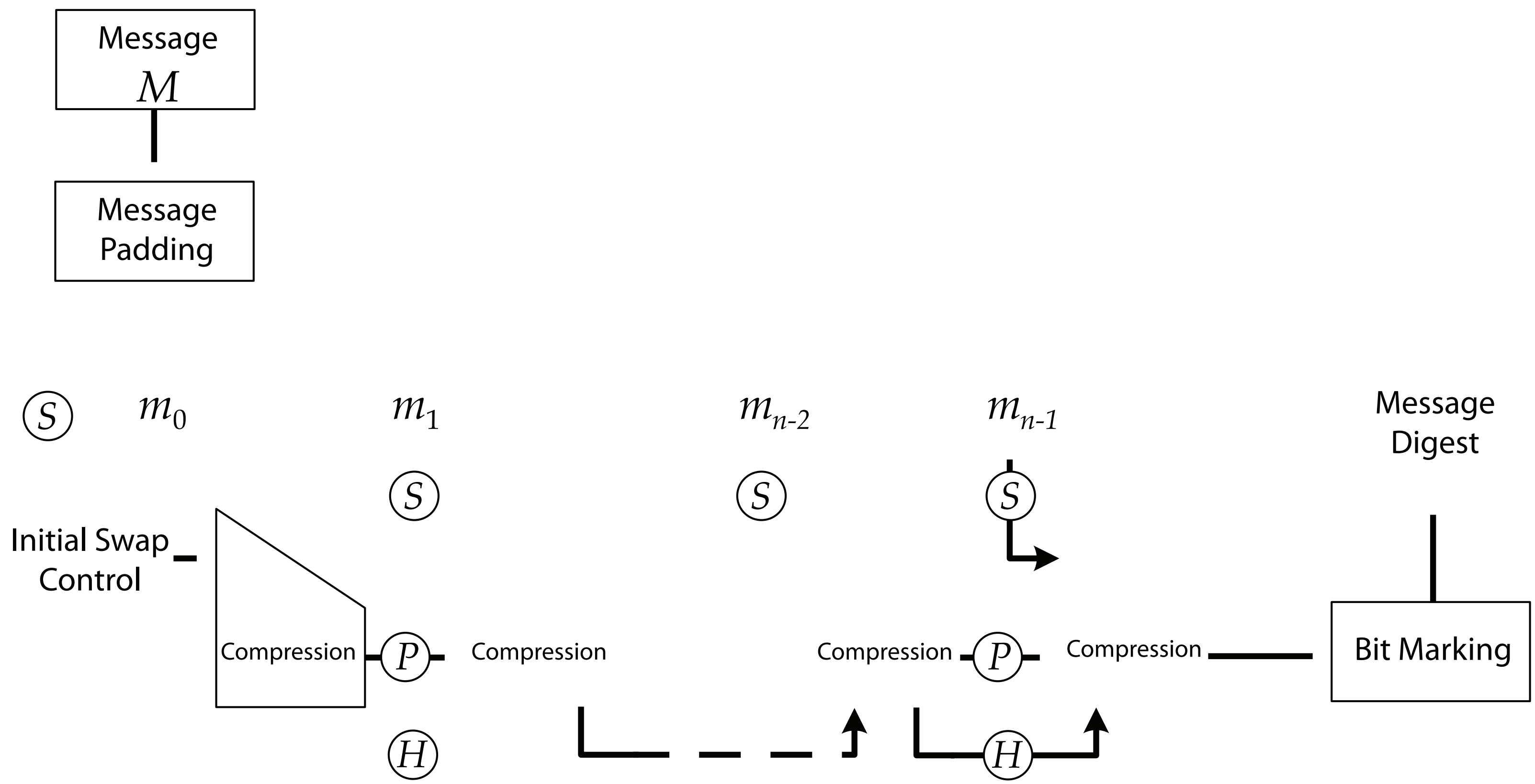
**Team:**
Gökay Saldamlı, Cevahir Demirkıran, Megan Maguire,
Carl Minden, Jacob Topper, Alex Troesch, Cody Walker

**First SHA-3 Candidate Conference - Feb 25-28, 2009**

## Spectral Hash Fundamentals

- Hashing methodology: Merkle-Damgard construction
- Organization of data: 512-bit data is broken into 128 4-bit blocks, and then, it is treated as a 3-d array of dimension $4 \times 4 \times 8$ with 4-bit entries
- Mathematical structures: Finite fields GF$(17)$ and GF$(2^4)$, 3-d Discrete Fourier transform of length 4 and 8 over the finite field GF$(17)$
- Operations: 3-d DFTs, Swaps, Rubic-type rotations, Affine transformations (similar to the AES S-box), Data controlled permutations, Nonlinear transformations
- Platforms: Highly suitable for hardware (FPGA & ASIC), but, optimized software implementations are also possible

# Merkle-Damgard Construction



# Initial Steps: Data to S-Prism

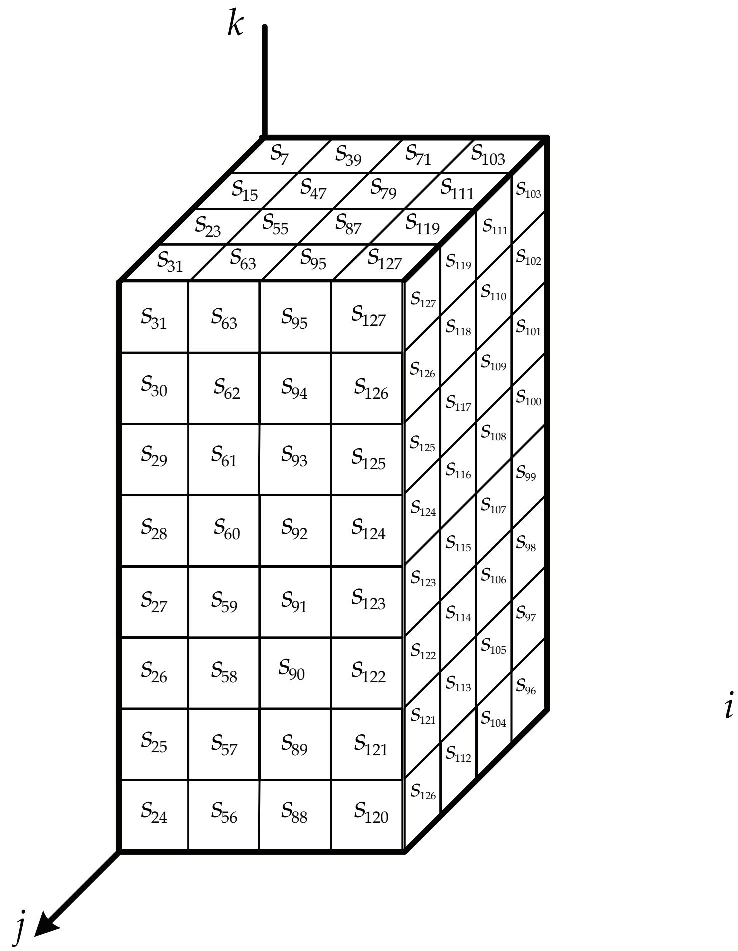
- Input is processed in 512-bit chunks $m_i$ for $i = 0, 1, \ldots$ with $|m_i| = 512$
- Take the 512-bit data chunk $m_i$ in step $i$, break it into a linear array of 128 4-bit blocks: $s_n$ for $n = 0, 1, \ldots, 127$ with $|s_n| = 4$
- Construct a $4 \times 4 \times 8$ array: s-prism, such that

$$S_{ijk} = s_{32i+8j+k}$$

Given $s_n$, we can also compute the 3-d index set as

$$\begin{aligned} i &= \lfloor n/32 \rfloor \pmod 4 \\ j &= \lfloor n/8 \rfloor \pmod 4 \\ k &= n \pmod 8 \end{aligned}$$

# S-Prism


k
S7 S39 S71 S103
S15 S47 S79 S111
S23 S55 S87 S119
S31 S63 S95 S127
S31 S63 S95 S127
S30 S62 S94 S126
S29 S61 S93 S125
S28 S60 S92 S124
S27 S59 S91 S123
S26 S58 S90 S122
S25 S57 S89 S121
S24 S56 S88 S120
S103 S111 S119 S127
S102 S110 S118 S126
S101 S109 S117 S125
S100 S108 S116 S124
S99 S107 S115 S123
S98 S106 S114 S122
S97 S105 S113 S121
S96 S104 S112 S126
i
j

# P-Prism and H-Prism

- We also have two additional prisms of the same shape: P and H
- P-prism is initially configured as

$$P_{ijk} = 32i + 8j + k$$

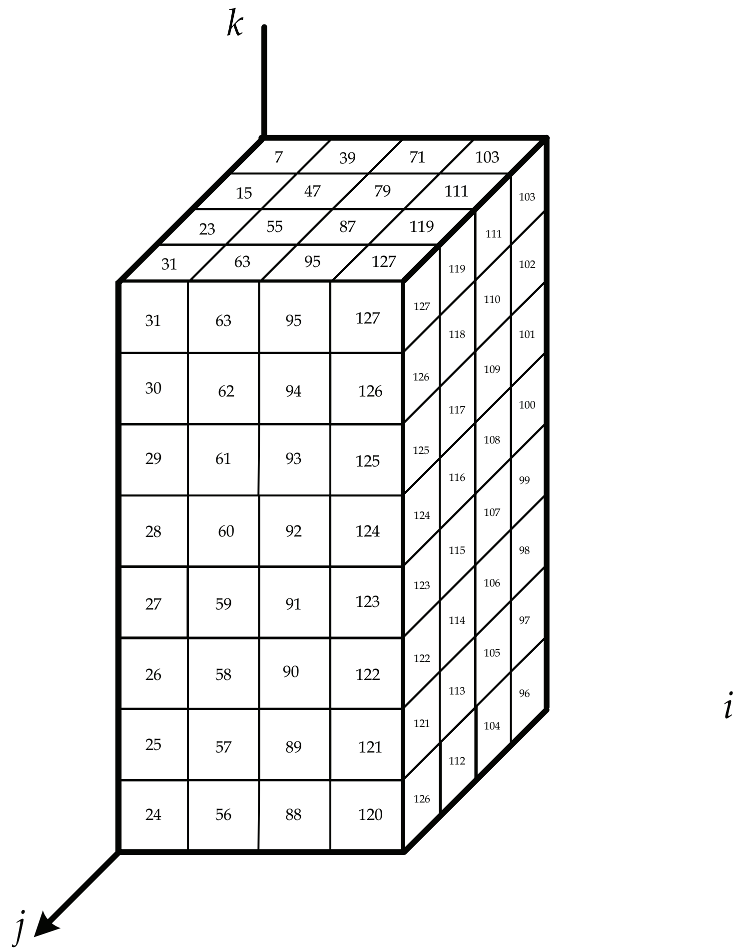
for $i, j = 0, 1, 2, 3$ and $k = 0, 1, \ldots, 7$. It holds a permutation of the 7-bit index set: $\{0, 1, 2, \ldots, 127\}$.

- H-prism holds the data from the S-prism of the previous round, therefore, like S-prism, it will contain 4-bit values. Its initial configuration is all zero:

$$H_{ijk} = 0$$

for $i, j = 0, 1, 2, 3$ and $k = 0, 1, \ldots, 7$.

# P-Prism (Initial Configuration)



# Begin Hashing: Initial Swap

- Take $m_0$ and create the initial S-prism
- Initialize P-prism
- Apply Initial Swap function to P-prism using the data in S-prism, using the following definitions:

$$\begin{aligned} SH(ijk) &= S_{ijk} \text{ div } 4 \qquad \text{(higher 2 bits of } S_{ijk}) \\ SL(ijk) &= S_{ijk} \bmod 4 \qquad \text{(lower 2 bits of } S_{ijk}) \end{aligned}$$

Initial Swap
for $k = 0, 1, \ldots, 7$
for $i = 0, 1, 2, 3$
for $j = 0, 1, 2, 3$
Swap$(P_{i\ j\ k}, P_{SH(ijk)\ SL(ijk)\ k})$

## Compression Function on $m_i$

Inputs: $m_i$, P-prism, H-prism
Outputs: P-prism, H-prism
P and H were updated in previous step with $m_{i-1}$

Take chunk $m_i$ and form S
S $=$ AffineTransform(S)
P $=$ SwapControl1(S,P)
P $=$ SwapControl2(S,P)
S $=$ DFT$_k$(S)
P $=$ SwapControl3(S,P)
S $=$ DFT$_j$(S)
P $=$ SwapControl4(S,P)
S $=$ DFT$_i$(S)
S $=$ NLST(S,P,H)
H $=$ S
P $=$ PlaneRotate(P)

## Affine Transformation on S-Prism

- For all $i, j, k$ do the following:
- Take an element of S-prism $S_{ijk}$ and compute its inverse

$$U = S_{ijk}^{-1} \in GF(2^4)$$

- Let $U$ be $U_0 + U_1 x + U_2 x^2 + U_3 x^3$
- Compute the matrix-vector product

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

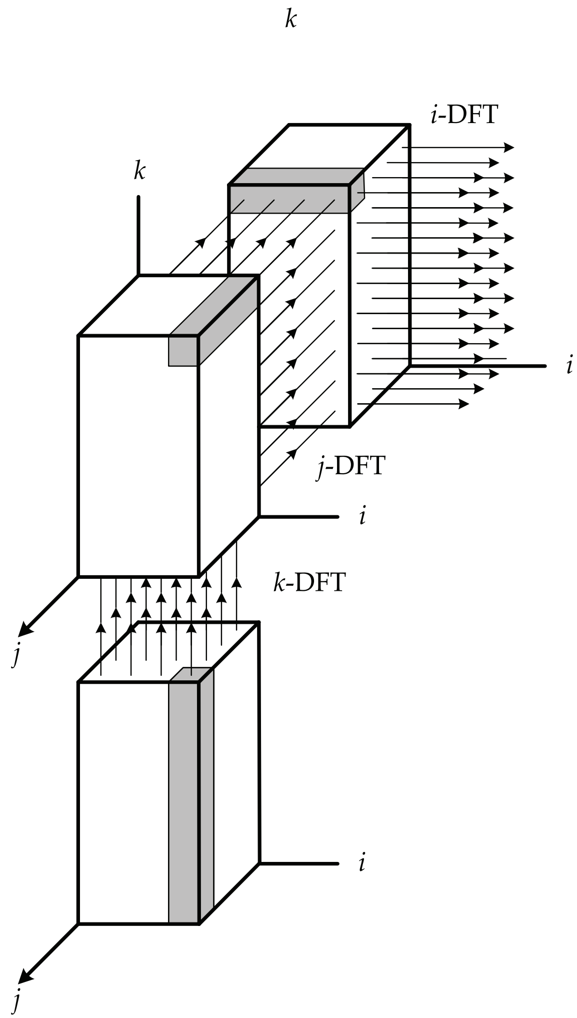
- Replace $S_{ijk}$ in the S-prism with the output $(S_3 S_2 S_1 S_0)$

## 3-d Discrete Fourier Transformations on S-Prism

- DFT over the $k$ axis: 16 simultaneous 1-dimensional 8-point DFT computations over GF$(17)$ with $w_8 = 2$
- DFT over the $j$ axis: 32 simultaneous 1-dimensional 4-point DFT computations over GF$(17)$ with $w_4 = 4$
- DFT over the $i$ axis: 32 simultaneous 1-dimensional 4-point DFT computations over GF$(17)$ with $w_4 = 4$

$$\mathbf{X}_i = \mathsf{DFT}_d(\mathbf{x}) = \sum_{j=0}^{d-1} w^{i \cdot j} \mathbf{x}_j \pmod{17}$$

# 3-d Discrete Fourier Transformations on S-Prism



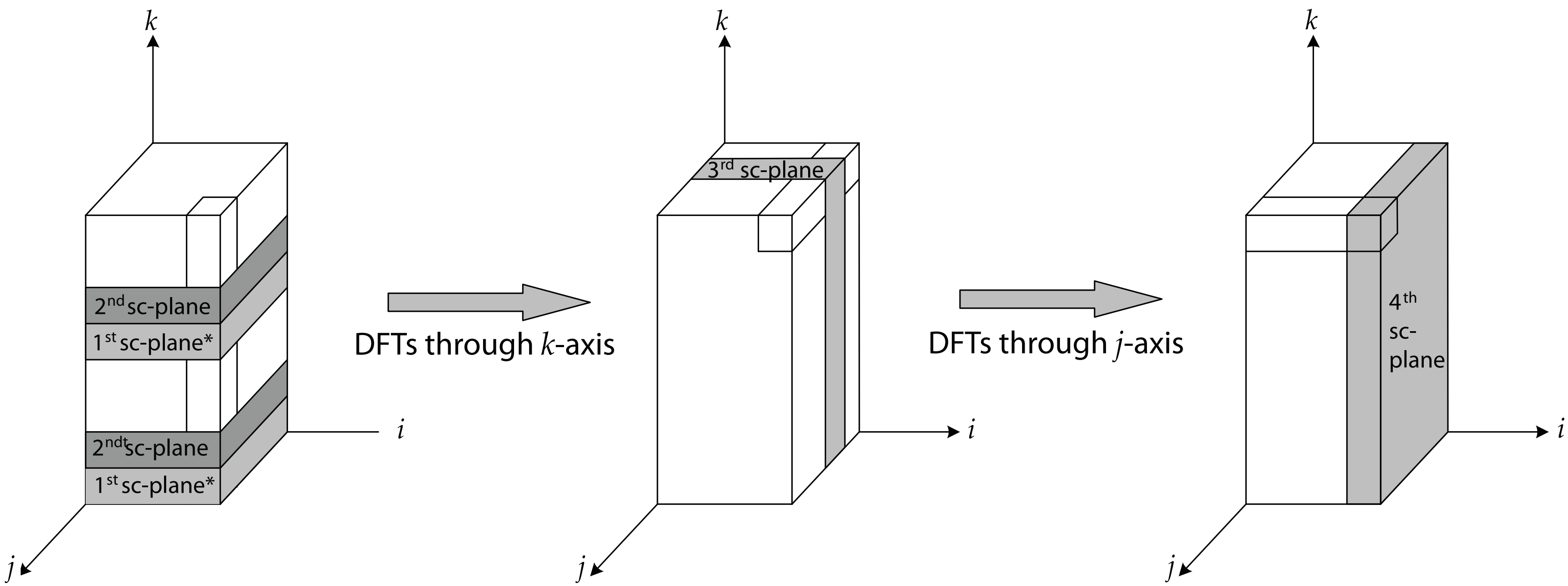
## Discrete Fourier Transformations on P-Prism

- DFT computations on P-prism are accomplished using data (S-prism) dependent swaps on P-prism
- A swap-control-plane (sc-plane) is generated using S-prism; these sc-planes are then used to update P-prism
- The 1st sc-plane is generated using these explicit formulae:

$$\begin{array}{lllll}
\text{if} \; S_{ij0}[0] \oplus S_{ij4}[0] & = & 0 & \text{then} & \text{swap}(P_{ij0}, P_{ij7}) \\
\text{if} \; S_{ij0}[1] \oplus S_{ij4}[1] & = & 0 & \text{then} & \text{swap}(P_{ij1}, P_{ij6}) \\
\text{if} \; S_{ij0}[2] \oplus S_{ij4}[2] & = & 0 & \text{then} & \text{swap}(P_{ij2}, P_{ij5}) \\
\text{if} \; S_{ij0}[3] \oplus S_{ij4}[3] & = & 0 & \text{then} & \text{swap}(P_{ij3}, P_{ij4})
\end{array}$$

- The other sc-plane definitions (2nd, 3rd, and 4th) are found in the document

# Swap Control Planes



## Nonlinear Transformations on S-prism

- For all $i, j, k$ do the following on S-prism:

$$S_{ijk} = (S'_{ijk} \oplus PL_{ijk})^{-1} \oplus \left(S'_{P_{ijk}} \oplus PH_{ijk}\right)^{-1} \oplus H_{ijk}$$

- Here, we have

$$\begin{aligned} S'_{ijk} &= S_{ijk} \pmod{16} \quad \text{for all} \;\; i, j, k \\ PL'_{ijk} &= P_{ijk} \pmod{16} \quad \text{for all} \;\; i, j, k \\ PH'_{ijk} &= (S_{ijk} \;\text{ div }\; 16) \,||\, (P_{ijk} \;\text{ div }\; 16) \quad \text{for all} \;\; i, j, k \end{aligned}$$

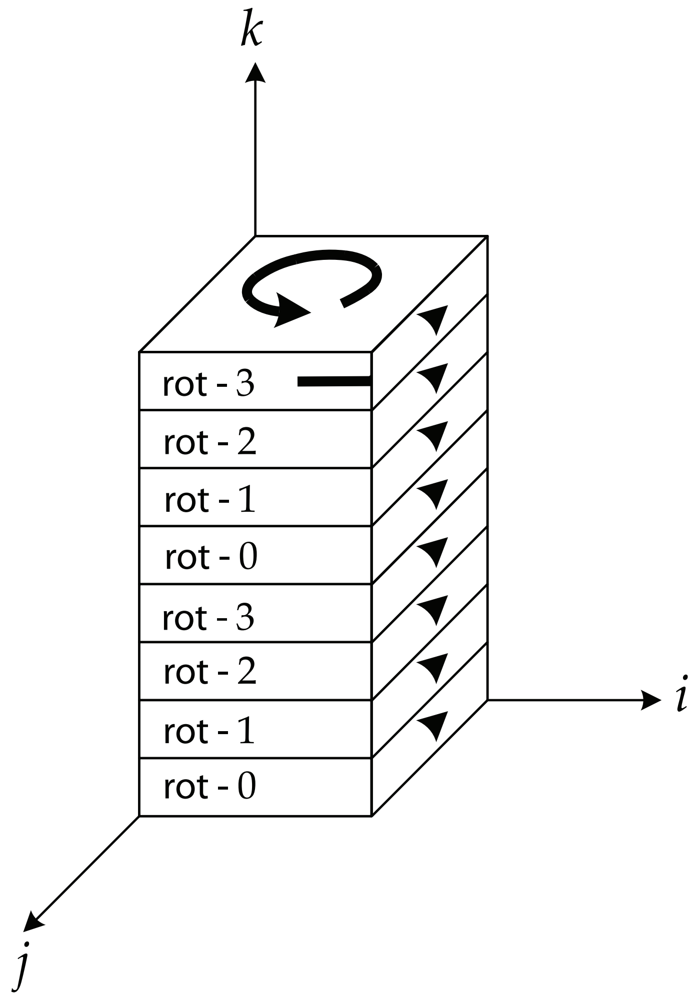
- Recall that GF$(17)$ produces 5-bit numbers, in the range $[0, 16]$. With these nonlinear transformations, they are reduced back to 4 bits.

# Rubic Rotations on P-prism

- The last step of the compression function involves plane rotations along the k-axis, defined as follows:

$$
\begin{array}{llll}
\text{if} & (k = 0 \bmod 4) & \text{then} & P_{ijk} = P_{ijk} \\
\text{if} & (k = 1 \bmod 4) & \text{then} & P_{ijk} = P_{(3-j)ik} \\
\text{if} & (k = 2 \bmod 4) & \text{then} & P_{ijk} = P_{jik} \\
\text{if} & (k = 3 \bmod 4) & \text{then} & P_{ijk} = P_{j(3-i)k}
\end{array}
$$

# Rubic Rotations on P-prism



## Hash Output via sg-Tables

- Spectral Hash algorithm can be configured to return hash values in 32 multiples of bit lengths between 128 and 512 bits, and thus, include sizes 224, 256, 384, and 512 bits.
- The hash value bits are selected from S-prism using the sg-table.
- For each hash length (128, 224, 256, 384, & 512), there are (4, 6, 8, 12, & 16) stars in the sg-table which gives the indices of P-prism and S-prism to be selected.
- For example, for 256-bit hash, the following sg-table is used.

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

## Final Hash Output

- After the selection process, the resulting S-prism looks like a swiss cheese, unselected parts are the holes.
- The final hash value is obtained from this punctured S-prism using the index mapping

$$h = H_0 \mid\mid H_1 \mid\mid \cdots \mid\mid H_{127}$$

such that

$$H_I = S_{ijk}$$

where $S_{ijk}$ is the punctured S-prism, and

$$I = 32i + 8j + k$$

for $i, j = 0, 1, 2, 3$ and $k = 0, 1, \ldots, 7$

## Security Claims

- DFT provides near perfect diffusion.
- No look-up tables – Prevents side channel attacks.
- Affine Transform prevents fixed points.
- The inversion function over a finite field has the best known non-linearity.
- Our current NIST-approved submission has an issue with the padding specification for messages whose lengths are multiples of 512 bits. There exists a trivial fix for this issue. Our website has a fixed version of the code.
- As of now, there are no current attacks (on the fixed code) that have shown any weakness in Spectral Hash.

## Performance for 512-bit Hashing

Software (Unoptimized reference implementation)

- 40 Mb/s on a Core Duo 2.16Ghz.
- 400 cycles per byte.
- 15,000 cycles of overhead.
- 256 bytes of internal state.

Hardware (FPGA)

- A maximum-area and speed implementation running on a 100-MHz Virtex-4 FPGA will produce 512-bit hashes at 51.2 gigabits per second! We will publish our code at `opencores.org`.
- Same hardware produces hashes of any length.
- We are currently working on an area-optimized ASIC implementation.

# What Makes Spectral Hash Special?

- It is the work of 5 undergraduate students (Megan Maguire, Carl Minden, Jacob Topper, Alex Troesch, Cody Walker) from the College of Creative Studies at the University of California Santa Barbara (UCSB), working with a postdoc and a professor.
- UCSB is the home of Crypto (We are natural cryptographers!)
- In software, we are not the slowest (Thanks ECOH!)
- Spectral rhymes with Special!