Fast Pseudo-Hadamard Transforms

Tom St Denis

Student. Advocate. Friend.
tomstdenis@iahu.ca

Abstract. We prove that the fast pseudo-Hadamard transform (FPHT) over a finite field has a bounded branch number. We shall demonstrate that the transform has an efficient implementation for various platforms compared to an equal dimension MDS code. We prove that when using a CS-Cipher[3] like construction the weight of any $2R$ trail is bounded for the case of an 8×8 transform. We show that the FPHT can also be combined with MDS codes to produce efficient transforms with half of the branch of a comparable sized MDS code. We present the FPHT-HASH one-way hash function which is constructed using a 32×32 FPHT which produces a 256-bit digest and processes the input at 24 cycles per byte with ISO C source code on an AMD Athlon XP processor.

Keywords. Pseudo-Hadamard Transform, Branch Analysis, One-Way Hash Function.

1 Introduction

An emerging block cipher and one-way hash function design construct is the Maximum Distance Separable (MDS) code. The goal of the MDS code is to promote a high branch through the linear components of the design to ensure a correspondingly low differential and linear "prop-ratio" [5]. Algorithms such as AES [1] and Whirlpool [2] are two prominent designs that employ this design construction. An MDS code of dimension $N \times N$ requires $O(N^2)$ time to complete which means while they offer excellent branch they do not scale well in terms of efficiency.

We present a proof that the branch of the fast pseudo-Hadamard transform (FPHT) is bounded by taking the approach of Daemen[5]. We present methods of implementing the FPHT that are faster than known methods of implementing MDS codes. We shall also demonstrate that MDS and FPHT codes can be combined to produce fast transforms with branch numbers much higher than the comparable dimension unmodified FPHT. We shall conclude with a concrete design based on the FPHT construction that achieves a very low cycle per byte processed.

The paper has been divided into the following sections. Section two discusses the relevant background theory and establishes our main theorem. Section three discusses the FPHT-HASH one-way hash function and finally section four concludes the paper.

2 Theory

2.1 Fast Pseudo-Hadamard Transform

At the foundation of these new results is the fast pseudo-Hadamard transform (FPHT) which was an idea originally employed in the SAFER [6] and SAFER+ [8] series of block cipher. It would be later changed into a fast four-point pseudo-Hadamard transform for the SAFER++ [7] block cipher in an attempt to raise the efficiency and strength of the underlying design. Stern and Vaudenay adopted their work on multi-permutations to the CS-Cipher [3] which also works on a similar principle of the FPHT.

In this paper we generalize the FPHT and prove that it has a bounded branch number when implemented over a finite field. We then show that the FPHT has several efficient means of implementation which make the design construct very flexible.

The FPHT can be characterized by a recursive linear transform defined by the relationship

$$H_n = \begin{bmatrix} 2 \cdot H_{n-1} & H_{n-1} \\ H_{n-1} & H_{n-1} \end{bmatrix} \text{ for } n \geq 1$$
$$H_0 = 1 \tag{1}$$

It is provably non-singular since the two vectors $\langle 2, 1 \rangle$ and $\langle 1, 1 \rangle$ are linearly independent. Note that within this paper decimal matrix coefficients may be considered their equivalent polynomial over $GF(2)[x]$ without loss of generality.

2.2 Branch Numbers

Originally defined in [5] the branch of a transform is the lowest sum of active inputs and outputs with respect to linear and differential cryptanalysis. The branch number only applies to linear transforms in any trivial sense. Let $||.||$ represent the number of non-zero coordinates of a given vector. Then the branch β of a linear transform $y = f(x) : x \in F^m \rightarrow y \in F^k$ is defined as minimum of $||x|| + ||y||$ for all non-trivial $x \in F^m$.

The branch of the FPHT can be placed into three distinct categories. Let β_n represent the branch of a $n \times n$ FPHT matrix where n is a power of two.

$$\begin{aligned} &\beta_1 = 2 \text{ (by convention)} \\ &\beta_2 = 3 \\ &\beta_n = 2 \cdot \beta_{n/4}, n \geq 4 \end{aligned} \tag{2}$$

More specifically the exact branch of FPHTs for $n > 4$ can be expressed as

$$\begin{aligned} &\beta_n = 2^{k+1} && \text{if } n = 2^{2k} \\ &\beta_n = 2^{k+1} + 2^k && \text{if } n = 2^{2k+1} \end{aligned} \tag{3}$$

The proof of the previous equation shall come in three parts. First we shall prove β_2 is correct followed by β_4 and finally general β_n.

Theorem 1. *β_2 is equal to three when H_1 is defined over a field such that it is non-singular.*

Proof. If $||x|| = 2$ then the proof is concluded as $||y||$ must be at least one. If $||x|| = 1$ then the input is either of the form $\langle 0, p\rangle$ or $\langle p, 0\rangle$ in which case the output is of the form $\langle p, p\rangle$ or $\langle 2p, p\rangle$ respectively. Since $p \neq 0$ and therefore $2p \neq 0$ the output weight must be two. □

Theorem 2. *β_n for $n = 2^k$ and $k \geq 2$ is equal to $2 \cdot \beta_{n/4}$ when H_k is defined over a finite field such that H_k is non-singular and the "2" coefficient generates a multiplicative group of order greater than k.*

Proof. We shall prove this theorem by first proving it for the case of $k = 2$. Then we shall generalize the proof for $k + t, t \geq 1$ by assuming the proof is true for $k + t - 2$. Since eventually $k + t - 2 \in \{0, 1, 2\}$ the proof is true by induction. To prove $k = 2$ we shall delineate all four possible input weights and prove that their output weight sums to a minimum of four.

When $||x|| = 1$ the output will be the single non-zero active input coordinate multiplied by the corresponding column of the transform. Therefore, $||y|| = 4$ and gives a branch of five.

When $||x|| = 2$ there are $\binom{4}{2} = 6$ ways to select active inputs. Since the transform has symmetry about the columns and rows there is really only one unique case to consider. To prove this case we shall consider the H_2 matrix and the $\langle p, q, 0, 0\rangle$ input vector.

$$H_2 = \begin{bmatrix} 4 & 2 & 2 & 1 \\ 2 & 2 & 1 & 1 \\ 2 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{4}$$

The vector when multiplied against H_2 produces $\langle 4p + 2q, 2p + 2q, 2p + q, p + q\rangle$ and by inspection at most two of those coordinates can be zero when $p, q \neq 0$. Therefore, the minimum branch for this case is four.

When $||x|| \geq 3$ the output must allow for $||y|| \geq 1$ since the transform is non-singular. This gives a branch of four in both cases and concludes the proof for $k = 2$.

Now we shall prove the theorem for the general case of $k \geq 2$ by induction. For this proof we again consider the four possible input patterns by re-writing the FPHT in a more conventional form.

$$H = \begin{bmatrix} A & B & B & C \\ B & B & C & C \\ B & C & B & C \\ C & C & C & C \end{bmatrix} \tag{5}$$

Where $A = 4C$ and $B = 2C$ for some non-singular FPHT C in the given field. Without loss of generality we shall assume that $A \neq B$, $B \neq C$ and $A \neq C$ since the "2" coefficient generates a sufficiently large multiplicative sub-group. As a consequence for any H_t with t less than the order of the group generated by $p(x)$ the resulting matrix can be divided into four columns and rows such that it has the unique pattern shown above (eqn. 5).

Note that for this part of the proof we shall not assume that C is of any given size except that it is of the form H_t for $t \geq 1$. To make the rest of the proof easier we shall let "word" represent a subset of the input coordinates aligned with one of the four matrix terms.

When only one input word is active it is obvious that the branch is at least $4 \cdot \beta_{n/4}$ since the output is simply the product of one of the columns of the matrix and the non-zero input coordinate.

For the case of two active words we shall use the input vector $\langle p, 0, q, 0 \rangle$ and split the proof into two sections. First we consider the case of $p = q$ and finally $p \neq q$. If $p = q$ then $pA \neq qB$ since $A \neq B$ and this simply implies $pA \neq pB$. Therefore, A contributes a branch of $\beta_{n/4}$ to the output. Similarly $pB \neq qC$ since $B \neq C$ which implies $pB \neq pC$ which contributes another $\beta_{n/4}$ of branch. When $p \neq q$ the logic is similar. $pB + qB$ will contribute at most a branch of $\beta_{n/4}$ and for any co-ordinate where they collide (recall this is in a finite field so the characteristic is finite) they cannot collide in $pA + qB$ since $A \neq B$. The same logic applies to the sum $pC + qC$. It is trivial to see that this similar logic applies to the vector $\langle 0, p, 0, q \rangle$, $\langle p, q, 0, 0 \rangle$ and $\langle 0, 0, p, q \rangle$.

The next case is of the input vector $\langle p, q, r, 0 \rangle$. The first output $pA + qB + rB$ can contribute at most a branch of $\beta_{n/4}$. Collisions can be expressed as $qB \approx pA + rB$ and simplified as $qB/2 = qC \approx pB + rC$ which cannot collide in the same coordinates with $pB + qC + rB$ (the third row). The same logic applies to the second and fourth row which contribute the other $\beta_{n/4}$ branch. The logic for the triple case applies to all four different permutations of the triple.

For the last case of the input vector $\langle p, q, r, s \rangle$ the first row produces a branch of at most $\beta_{n/4}$ where collisions can be written as $sC \approx pA + qB + rB$. Like the proof for the previous case this cannot collide in the same coordinates with the second row. The same logic applies to the last two rows as a pair.

For all four cases to be true we assume the branch for the $k - 2$ case has been bounded. Since the bounds for $k \in \{0, 1, 2\}$ have been proven, the theorem is clearly true for all $k \geq 2$. □

Remark. Asymptotically an FPHT of dimension $n \times n$ has a branch of $O(\sqrt{n})$ whereas by comparison an MDS code would have a branch of exactly $n + 1$.

Corollary 3. *In an H_k transform for $k \geq 2$ the H_{k-2} sub-matrix can be replaced with any complete non-singular transform and the resulting H_k shall be non-singular as well as have twice the branch of the $2^{k-2} \times 2^{k-2}$ transform.*

Lemma 4. *The output weight of the FPHT can be placed into four distinct classifications.*

If the input weight is zero, the output weight is zero. If the input weight is one than the output weight is always 2^k for H_k. If the input weight x is less than β_k the output weight is at least $\beta_k - x$ and otherwise the output weight is at least one.

Lemma 5. *The FPHT over the ring of integers modulo 2^k (such as in [6, 8, 7]) cannot be counted in the same manner as theorem two provides.*

Since two divides the modulus the transform allows input differences p to cancel out since it is possible that $2p \equiv 0$ for $p \not\equiv 0$. As a result for example, the H_3 as used in [6] has a branch of three.

$$\langle 0,0,0,0,0,0,128,0\rangle = H_3(\langle 128,128,0,0,0,0,0,0\rangle) \pmod{256} \tag{6}$$

In the generic equation for the FPHT (eqn. 1) we cannot replace the "2" value with an odd number since the determinant will no longer be a unit in the ring. Therefore, there is no construction of the FPHT over the ring of integers modulo 2^k where the branch is at least equal to the branch of the FPHT over a field.

2.3 Implementation

In $O(n\log n)$ time. Any FPHT requires at most $O(n\log n)$ time to complete which scales nicely compared to an equal dimension MDS code which requires $O(n^2)$ time. More specifically with $O(n)$ space an FPHT requires only $O(\log n)$ time to complete.

In hardware designs the actual transform is very efficient. Since only the H_1 transform must be implemented directly (see Fig. 3) a trivial multiplication by $p(x) = x$ is all that is required. For example, the H_5 transform from our proposed HASH function FPHT-HASH only requires 1600 XOR gates and would have a delay of approximately fifteen XOR gates.

In embedded software the transform can be implemented with the standard H_1 transform using only $\log_2(n) \cdot \frac{n}{2}$ table lookups and $\log_2(n) \cdot n$ XOR operations. On the Intel 8051 8-bit processor using eight bit polynomials over $GF(2)[x]$ such a transform would require exactly $4n \cdot \log_2(n)$ cycles to complete. Using such a construction on a hypothetical eight round substitution permutation network with a sixteen byte block size would imply that at least 2048 cycles, approximately 2.04 milliseconds[1] per block would be used by the linear transformation. Assuming a simple XOR key schedule along with a table driven non-linear substitution along with an overhead of 32 cycles per round yields a theoretical cipher requiring approximately 2700 cycles per block yielding a throughput of $47,410$ bits per second.

As a quick remark such a hypothetical design would only require a non-linear transform with a differential and linear profile maximums of $\frac{8}{256}$ and $\frac{32}{256}$ to make both differential and linear cryptanalysis provably impractical (requiring 2^{121} and 2^{146} texts respectively) over only six of the eight rounds.

[1] On the standard 12MHz 8051 CPU.

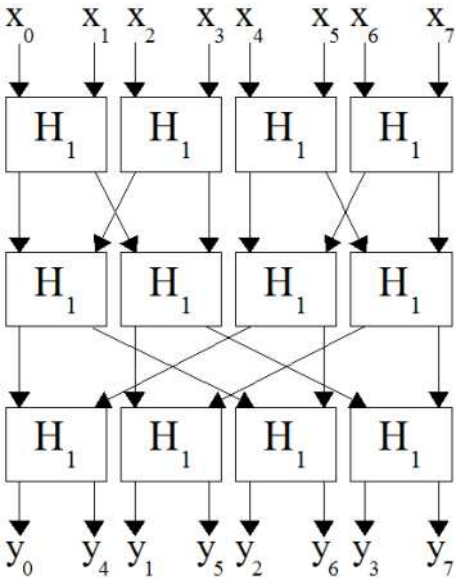

Fig. 1. H_3 as a three layer network.

Table Driven Another popular method of implementing linear transforms is with a table driven algorithm. Let ζ_i represent the i'th column of the transform. The transform $y = H_k(x)$ where $x, y \in F^{2^k}$ can be computed with the following equation.

$$y = \sum_{i=0}^{2^k-1} x_i \cdot \zeta_i \tag{7}$$

This is accomplished by pre-computing 2^k tables with $\#F$ elements each.

$$T[i][j] = j \cdot \zeta_i, 0 \leq i < 2^k, 0 \leq j < \#F \tag{8}$$

Now equation 7 can be re-written as

$$y = \sum_{i=0}^{2^k-1} T[i][x_i] \tag{9}$$

This technique is traditionally used with MDS codes with an optimization applied for circulant codes. In the case of FPHT transforms there are more optimizations available. For example, H_k for $k \geq 2$ can be implemented with only eight multiplications by H_{k-2} transforms. A comparable MDS code (of dimension $2^k \times 2^k$) even if circulant would require at least twelve multiplications by MDS codes of dimension $2^{k-2} \times 2^{k-2}$.

The optimizations can go further even if the table dimensions are fixed. For example, H_5 with H_2 tables requires 20 unique multiplications compared to the 36 that would be required for an MDS, and so on.

Remark. The series of fast pseudo-Hadamard transforms over $GF(2)[x]/v(x)$ have a distinct advantage over implementations that use the ring of integers (as in the case of [6, 8, 7]). In table driven implementations the former group allows a vector addition to be applied with a single XOR operation. The latter case cannot be implemented with a single addition as carries may propagate across coordinates of the summand.

Recently several processors have been given Single Instruction Multiple Data (SIMD) style instructions such as those of x86 series with MMX. "paddb" of the MMX instruction set, for instance, can add eight octets in parallel. It could be used to compute the SAFER linear transform with eight lookups and seven "paddb" instructions.

However, most processors do not have SIMD instructions and as a result the quickest way to implement a SAFER style FPHT is with the $O(N\log N)$ approach. The benefit of the FPHT over $GF(2)[x]/v(x)$ is clearly both the flexibility of implementation as well as the bounded branch number.

2.4 Modified Transforms

Lemma 6. *Any non-singular complete H_{k-2} will do.*

For a given H_k for $k \geq 3$ any non-singular or specifically complete bijective $2^{k-2} \times 2^{k-2}$ transform will do in the place of H_{k-2}. By "complete" it is meant that each output coordinate is a non-null function of every input coordinate. This observation includes the set of non-linear $(2^{k-2}, r)$-multipermutations [10] which would produce a transform with a branch of $2r$.

For example, consider the case of [3] where a $(2, 3)$-multipermutation is used in the place of the H_1. According to theorem 2 all that is required is that the the unique 4×4 (eqn. 5) pattern exist. As a result the branch of the transform is the same as a normal H_3 FPHT. However, as noted by Vaudenay this approach has much less linear structure. On the other hand using non-linear transforms in the place of H_{k-2} transforms incurs an efficiency penalty on most modern processors as table driven algorithms cannot be used.

Theorem 7. *The FPHT H_3 with a non-linear H_1 transform has a $2R$ trail weight of at least twelve.*

Proof. For this proof we shall make use of the graph Fig. 2 and lemma 4. In this construction we shall replace the H_1 transform with a $(2, 3)$-multipermutation where the two coordinates are first sent through a non-linear bijection and then through a non-singular 2×2 linear transform (such as H_1).

For the purposes of this theorem we shall assume that a "round-key" is added to the coordinates before every layer of the transform (similar to the round function of [3]) to create a Markov chain.

Let x represent the number of active input coordinates and similarly y the number of active output coordinates of the first round. For the second round we shall let z represent the number of active output coordinates when y input coordinates are active. In both cases of y and z we shall assume the minimum possible values.

If there are x active inputs this means that in the first layer there must be at least $\lceil x/2 \rceil$ active H_1 transforms. The H_1 has a branch of three which means over the first two layers there must be at least $3\lceil x/2 \rceil$ active non-linear functions. The same logic applies to the last and first layer of the two adjacent rounds as well as to the last two layers of the second round with y and z respectively. Over the two rounds the minimal trail weight can be expressed as

$$\rho = 3 \cdot (\lceil x/2 \rceil + \lceil y/2 \rceil + \lceil z/2 \rceil) \tag{10}$$

The lowest value of ρ occurs for $\langle x, y, z \rangle = \langle 2, 4, 2 \rangle$ in this case. □

Remark. The CS-Cipher employs a slight modification in that they permute the order of the output coordinates. As a result the same trail pattern that causes a weight of twelve through the modified FPHT does not work in CS-Cipher.

Remark. The logic of the previous proof was applied to the H_4 with a similarly modified H_1. The minimal two round weight was observed over all possible input weights as being 24. This could applied to the CS-Cipher to extend the block size to 128 bits.

Using the same non-linear function and comparable key schedule the modified version of the CS-Cipher would only require four rounds before any differential or linear trail would provably fail. Therefore, it seems that the CS-Cipher is a scalable design in embedded software and hardware.

Lemma 8. *Combining a smaller MDS transform with a larger FPHT results in a modified FPHT with higher branch.*

An MDS code is a $(n, n+1)$-multipermutation and would be optimal for the construction in terms of the branch. The transform H_k for $k \geq 3$ with a $2^{k-2} \times 2^{k-2}$ MDS code in the place of the H_{k-2} would yield an overall branch of $2^{k-1}+2$ by a simple extension of theorem 2. This modification yields a significant improvement over the branch of an unmodified FPHT.

Dimension	Branch of MDS	Branch of Modified FPHT	Branch of FPHT
16	17	10	8
32	33	18	12
64	65	34	16

Fig. 2. Comparison of modified and unmodified FPHT.

Consider H_4 where the H_2 sub-matrices were replaced with 4×4 MDS transforms. The MDS codes have a branch of five which means that the modified H_4 would have a branch of ten instead of eight. If the coordinates of the input and output vectors were octets then on a 32-bit platform a total of three quadruples of tables are required for the unique square 4×4 sub-matrices of the H_4. This approach would require 12KB of memory, 32 table lookups and 37 XORs. Compared to an equal dimension MDS code which would require 32KB of memory, 64 table lookups and 60 XORs, the FPHT is nearly twice as fast and requires less than half of the memory.

3 FPHT-HASH

We have used the FPHT H_5 to produce a fast one-way hash function. The FPHT-HASH function compresses blocks of 512-bits and produces a digest 256-bits long. The input message is terminated with a one bit followed by enough zero bits to make the message length congruent to 448 modulo 512. The 64-bit representation of the length of the original message in bits is then appended. Every block of 512 bits is then sent through the compression function. All values are loaded and stored in little endian fashion.

3.1 Compression Function

The following pseudo-code describes the compression function. It accepts a message block $T_{0..15}$ and internal state $S_{0..7}$ and updates the state before completion. The internal state is initially set to $S_i = 286331153 \cdot i$. Both S and T are arrays of 32-bit words.

1. for $x = 0$ to 7 do $L_x \leftarrow S_x$
2. for $x = 0$ to 15 do $W_x \leftarrow T_x$
3. for $x = 16$ to 64 do $W_x \leftarrow (W_{x-16} \oplus W_{x-14} \oplus W_{x-8} \oplus W_{x-3} \oplus x)^{<<11}$
4. for $x = 0$ to 5 do
   (a) for $y = 0$ to 7 do $L_y \leftarrow L_y \oplus W_{16+8x+y}$
   (b) $L \leftarrow \theta(\gamma(L))$
5. for $x = 0$ to 7 do $S_x \leftarrow S_x + L_x$

In this design $\gamma(x)$ which is the AES substitution box is applied to all 32 coordinates of the input simultaneously. $\theta(x)$ is H_5 over the field $GF(2)[x]/(x^8 + x^4 + x^3 + x^2 + 1)$. The $^{<<11}$ notation indicates a left cyclic rotation by eleven bits.

3.2 Implementation

FPHT-HASH has been implemented on the AMD Athlon XP processor with ISO C source code[2]. The H_5 has been implemented using tables for the four unique

[2] URL to be given

4×4 transforms[3] requiring a total of 16KB of memory. The Fig. 3 summarizes the speed of various hash functions (all written in portable C).

From the table it is evident that FPHT-HASH compares fairly well. The only design faster than itself is MD5 which produces a message digest half the size. In hardware the FPHT-HASH function scales nicely since the $O(n\log n)$ based approach can be used reducing the time complexity to $O(\log n)$ and the space complexity to $O(n)$.

Hash Function	Cycles per byte	Message Digest Size (bits)	Relative Rate
MD5	9	128	0.38x
SHA-1	16	160	0.66x
RIPEMD-160	26	160	1.08x
TIGER/192	27	192	1.13x
FPHT-HASH	**24**	**256**	**1x**
SHA-256	34	256	1.41x
SHA-512	74	512	3.08x

Fig. 3. Comparison of Various One-Way Hash Functions.

3.3 Analysis

Non-Linear Transform γ The AES substitution box was chosen because it has a very low differential and linear profile. It was also readily available to prototype this design. The function has several fast hardware implementations which should render this design equally as efficient in software as hardware.

Linear Transform θ The θ transform is the H_5 transform. It has several fast software implementations as well as a trivial implementation in hardware. The transform is complete in that every output coordinate is a function of all input coordinates. This promotes a very high level of diffusion. The transform also has a branch of 12 which allows the minimal trail weight to be bounded over two compositions of the transform.

The polynomial $x^8 + x^4 + x^3 + x^2 + 1$ was chosen because it is irreducible over $GF(2)[x]$ and the polynomial $p(x) = x$ is a primitive generator in the multiplicative sub-group.

[3] The unique tables are for $H_2, 2 \cdot H_2, 4 \cdot H_2$ and $8 \cdot H_2$.

Key Schedule The key schedule (step three) was designed to use all of the input in the first round and also make it hard to control the input in a meaningful manner. It is a simple LFSR generator with a rotation to further ensure higher diffusion of the key material. The purpose of the key schedule is to make differential trails less likely to succeed. Since the initial input state is fixed the attacker could create a differential trail of probability one through rounds where the input key were used verbatim. However, since the remaining round keys are a function of the input it is harder to ensure the probability one differential (it either exists or it does not).

In the case of this hash function we chose not to use the input key as verbatim at all. Instead we use the the linear mixing of the input through a LFSR with the key word number added in to prevent trivial slide attacks. For example, the first round keys $W_{16..23}$ are a linear function of the input $W_{0..15}$ as well as the first five generated keys $W_{16..20}$. This construction is thought to make the design resistant to related key attacks.

Differential and Linear Cryptanalysis Essentially FPHT-HASH is a 256-bit Substitution-Permutation Network applied in the Davies-Meyers mode (scheme 19 of [9]) to create a one-way hash function. The H_5 transform has a branch of 12 which means over the four rounds of the compression function there must be at least 24 active sboxes. As a result the best theoretical differential and linear trails would have a probability and bias of $2^{-6 \cdot 24} = 2^{-144}$ and $2^{-4 \cdot 24} \cdot 2^{23} = 2^{-73}$ respectively.

It does not seem possible for a differential attack to cause a collision within the compression function faster than by the birthday paradox. We conjecture that FPHT-HASH offers 2^{128} time resistance to collision finding.

4 Conclusion

The FPHT has been analyzed with respect to its speed and security. The transform has a provably bounded branch value for any given dimension as well as a fast implementation which requires at most $O(N \log N)$ time to complete. We have also shown it is possible to join the FPHT and MDS to create a fast transform that has higher branch than the FPHT alone.

We have also shown a relatively simple one-way hash function which achieves a competitive (if not superior) processing throughput when compared to other well known published designs. We would like to thank Matthew Johnson of the University of Western Australia, Greg Rose of QUALCOMM Australia, Robert Gilmour, David Malan of Harvard University and Michael Gschwandtner of the University of Salzburg for valuable peer review.

References

[1] Joan Daemen, Vincent Rijmen, *AES Proposal: Rijndael*

[2] Paulo S.L.M. Barreto and Vincent Rijmen, *The WHIRLPOOL Hashing Function*
[3] J. Stern and S. Vaudenay. CS-Cipher. In Fifth International Workshop on Fast Software Encryption, Berlin, Germany, March 1998. Springer-Verlag.
[4] S.Vaudenay, "On the Security of the CS-Cipher", Fast Software Encryption, March 1999, Springer-Verlag, pp. 260-274
[5] J. Daemen. Cipher and hash function design: strategies based on linear and diferential cryptanalysis. PhD thesis, Katholieke Universiteit Leuven, March 1995
[6] J.L. Massey, "SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm", Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 1–17.
[7] J.L. Massey, G.H. Khachatrian and Kuregian M.K. Nomination of SAFER++ as Candidate Algorithm for NESSIE. Available at www.cryptonessie.org.
[8] J. Massey, G. Khachatrian, and M. Kuregian, "Nomination of SAFER+ as Candidate Algorithm for the Advanced Encryption Standard (AES)", NIST AES Proposal, 1998.
[9] B. Preneel, R. Govaerts and J. Vandewalle, "Hash Functions based on block ciphers: a synthetic approach.", Crypto '93, Springer-Verlag, 1993, pp. 368–378
[10] S. Vaudenay, "On the Need for Multipermutations: Cryptanalysis of MD4 ad SAFER", LIENS - 94 - 23, November 1994

Appendix A - Test Vectors

The following are test vectors for FPHT-HASH.

```
""
        d5 e9 5a a7 4d a3 9c 10 a9 ea 11 bd 22 49 a0 b3
        be 67 e3 78 58 0d 72 f0 64 be 80 c5 3f c8 13 24

"abc"
        b4 bd 4f ce cf 2f c3 c5 5d 1a 77 dd 2b 6e 6f 77
        be bc 26 1b 1a b0 8e 32 1a 36 6a 66 3e f0 ad 3c

"The New Fast Pseudo-Hadamard Transform Hash Function."
        b7 b9 be 3b 64 f7 2b ab 97 1f a7 71 3d ca d0 de
        b1 1d ea 3e 67 39 5f cd 2b d9 d5 91 c9 5a 90 77

"aaaabbbb" x 12 times
        43 27 10 2d e6 7b 49 f6 85 a4 12 e9 a9 7f 45 49
        7a 9f 3d b3 d4 22 42 bb 93 cd c7 a0 97 55 b8 0f
```