# Maya: A Novel Block Encryption Function

Mahadevan Gomathisankaran and Ruby B. Lee

Electrical Engineering, Princeton University, Princeton, NJ 08544.
{mgomathi,rblee}@princeton.edu

**Abstract.** We propose a novel methodology to design Block Cipher functions. This methodology is illustrated with the design of a specific block cipher function **Maya**[1]. Our design philosophy is to derive the S-Boxes themselves from the secret key. This makes breaking any round function equivalent to guessing all the key-bits. Advantages of our design include much larger key sizes in relation to the block size, an order of magnitude improvement in the hardware implementation efficiency together with the necessary resistance to cryptanalysis.

## 1 Introduction

Block cipher functions form the fundamental building blocks of encryption systems. Block ciphers can be used in various modes of operation for confidentiality, message authentication, one-way function and hash functions. In their report titled "*Ongoing Research Areas in Symmetric Cryptography*" [1] the authors state that new block ciphers that may offer specific advantages over AES [2] need to be studied and designed. Our motivation for the design of **Maya** is to increase the *security* of the cipher function with respect to the number of gates required to implement it. The *security* of block cipher is directly proportional to the *key* size. Thus by reducing the percentage of gates in the implementation that are independent of the *key* we can increase the factor *security*/*gate*.

Conventional block ciphers [3,2,4] derive their security from the embedded secret *key*. One of the inputs, key, in each round is secret whereas the round functions themselves are public. The secret, however, is combined with the state in a limited way, as an XOR, during each round. We propose a simple yet novel approach wherein the round functions themselves become the secret, while the function schema is a publicly published algorithm.

[1] **Maya** in Sanskrit means *illusion*.
This work was supported in part by NSF Cybertrust CNS-0430487 and by a research gift from Intel.

The intuition is to use reconfigurable gates as round functions and define their configurations as the secret (or key). Hence the complexity of such a cryptographic function is derived from the fact that almost all of the round processing is driven by the secret (truth tables). In a traditional block cipher, the secret is combined with the state with an XOR as one of the steps in the round. This XOR step is susceptible to linear modeling of the secret and input/output relationship. When the secret is used as a Boolean gate truth table, it is inherently non-linear.

The main advantages of our design approach are:

1. Key size much greater than the block length can be easily accommodated. This allows for higher security guarantees without having to increase the latency of cipher operation.
2. Smaller block lengths are well-suited for processor level instruction encryption when the encryption system is placed in-line into a processor pipeline [5,6].
3. Area and time efficient hardware implementation reduces the processing overhead drastically for security.
4. Simplicity of the design.

Reconfigurable S-boxes have been proposed and used in GOST [7] and TREYFER [8] encryption algorithms. GOST 28147-89 is a Soviet and Russian government standard symmetric key block cipher. GOST defines the S-Boxes to be secret but does not use the *key* bits to choose the S-Boxes. The GOST philosophy is to pre-determine S-Boxes between communicating parties. TREYFER is a block cipher/MAC designed in 1997 by Gideon Yuval. TREYFER has single 8x8 S-Box which is defined by the secret key. Due to the simplicity of its key schedule, using the same 8 key bytes in each round, Treyfer was one of the first ciphers shown to be susceptible to a slide attack [9]. This cryptanalysis is independent of the number of rounds and the choice of S-box.

## 2 Perliminaries

### 2.1 Definitions

**Definition 1 (Bijective Function)** *A function* $f : X \mapsto Y$ *is* **bijective** *if for every* $y \in Y$ *there is exactly one* $x \in X$ *such that* $f(x) = y$.

### 2.2 Notations

- Let $\mathcal{S}^N$ be the set of all $N \times N$ functions ($|\mathcal{S}^{N\times N}| = 2^{N2^N}$).
- Let $\mathcal{S}_\pi^N$ be the set of all $N \times N$ *bijective* functions ($|\mathcal{S}_\pi^N| = 2^N!$).
- Let $I_N$ be the set of all $N$ bit numbers.
- Let $x \in I_N$ then $x^{\text{B}(i)}$ represents the $i^{th}$ bit[2], where $0 \leq i \leq N-1$.
- Let $x \in I_N$ then $x^{\text{NIB}_n(i)}$ represents the $i^{th}$ nibble of $n$ bits, *i.e.*

$$x^{\text{NIB}_n(i)} = \left(x^{\text{B}(in)}, x^{\text{B}(in+1)}, \ldots, x^{\text{B}(in+n-1)}\right)$$

  where $0 \leq i \leq \frac{N}{n} - 1$.
- Let $f \circ g$ denote the function $f(g(x))$.
- Let $f^2$ denote the function $f(f(x))$.
- Let $\oplus$ represent bitwise XOR operation.

## 3 Design Rationale

We propose that the design the design of block cipher should achieve the following properties: *indexability*, *symmetry*, *similarity* and *irreversibility*.

- Indexability requires that the key size of the cipher function be comparable (polynomial) to the input/output (*block*) size ($N$).
- Symmetry requires that every output bit is influenced by every input bit uniformly.
- Similarity requires that any differential input causes output differential effectively similar to the universal set. In other words the output bits should switch with a probability very close (computationally indistinguishable) to $\frac{1}{2}$.
- Irreversibility requires that given reasonable number of input output pairs no efficient algorithm can fully or partially infer the key bits.

### 3.1 Achieving Indexability

Let $N$ be the input/output block size (in *bits*) of the cipher function under design. There are $2^N!$ unique $N \times N$ bijective functions. We have to choose a subset from this universe using the $K$ key bits. The requirement for $K$ is to be size Poly($N$). The easiest way to achieve this would be to design the $N \times N$ function using $m \times m$ bijective functions, where $m = O(\log N)$.

[2] *Wlog* from the left

Let $N$ be divisible by $m$ and let $q = \frac{N}{m}$. We can realize an $N \times N$ function with $q$, $m \times m$ functions. The set $\mathcal{S}_\pi^m$ is a subset of $\mathcal{S}^m$ and $m2^m$ bits are required to represent a function in the set $\mathcal{S}^m$ uniquely. Thus all the $q$ $m \times m$ bijective functions can be represented by $N2^m$ bits. Thus the size of the key is $N2^m$. By choosing $m$ appropriately we can keep the key size at Poly($N$).

Let $\hat{f}$ represent the $N \times N$ function realized by the $q$, $m \times m$ bijective functions. Let $\Pi_i$ represent the $i^{th}$ $m \times m$ bijective function where $0 \leq i \leq q-1$. There are $q$ functions and each of these functions are chosen *uniformly* and *independently* from the set $S_\pi^m$. Let $x, y \in I_N$ be the input and output of the function $\hat{f}$ respectively. Then we define the function $\hat{f}$ as

$$y^{\text{NIB}_m(i)} = \Pi_i(x^{\text{NIB}_m(i)}), \text{ where } 0 \leq i \leq q-1.$$

$\hat{f}$ has achieved the first design goal. We will enhance the design in steps to achieve other design goals.

### 3.2 Achieving Symmetry

Function $\hat{f}$ does not satisfy the symmetry requirement. The input is split into nibbles of $m$ bits and each of these nibbles are operated upon independently. Hence the output bits are not uniformly influenced by every input bit. To achieve symmetry we use the property of $\Pi$ blocks.

Each of the $\Pi$ functions takes an $m$ bit input and produces an $m$ bit output. The $\Pi$ functions are uniformly distributed over the set $\mathcal{S}_\pi^m$, thus causing the $m$ bit outputs to exhibit the symmetry property with respect to their input bits, *i.e.*, every output bit of $\Pi$ is a uniform function of every one of the $m$ input bits. Hence, we need to design a MIX function that mixes the nibbles of output bits from $\hat{f}$ uniformly. The exact specification of MIX function is dependent on the parameters $N$ and $m$. For any given $N$ and $m$ a MIX function can be specified in such a way that the output bits are uniformly dependent on every input bit within $\log_m(N)$ rounds. This also is the minimum number of stages required to achieve this level of mixing. We define

$$\hat{g} := \text{MIX} \circ \hat{f}$$
$$\hat{h} := \hat{g}^{\log_m(N)}.$$

Realization of MIX function involves distribution of the bits across various nibbles in a such a way as to create an $m$-ary tree rooted at each nibble at

**Table 1.** Example MIX Functions

(a) $m = 4$, $N = 32$

| Inp NIB | → {Out NIB} |
|---|---|
| 0 | → {0, 2, 4, 6} |
| 1 | → {1, 3, 5, 7} |
| 2 | → {1, 3, 5, 7} |
| 3 | → {2, 4, 6, 0} |
| 4 | → {2, 4, 6, 0} |
| 5 | → {3, 5, 7, 1} |
| 6 | → {3, 5, 7, 1} |
| 7 | → {4, 6, 0, 2} |

(b) $m = 4$, $N = 64$

| Inp NIB | → {Out NIB} |
|---|---|
| 0 | → {00, 04, 08, 12} |
| 1 | → {01, 05, 09, 13} |
| 2 | → {02, 06, 10, 14} |
| 3 | → {03, 07, 11, 15} |
| 4 | → {05, 09, 13, 01} |
| 5 | → {06, 10, 14, 02} |
| 6 | → {07, 11, 15, 03} |
| 7 | → {08, 12, 00, 04} |
| 8 | → {10, 14, 02, 06} |
| 9 | → {11, 15, 03, 07} |
| 10 | → {12, 00, 04, 08} |
| 11 | → {13, 01, 05, 09} |
| 12 | → {15, 03, 07, 11} |
| 13 | → {00, 04, 08, 12} |
| 14 | → {01, 05, 09, 13} |
| 15 | → {02, 06, 10, 14} |

the first round. Tables 1(a) and 1(b) show an example MIX function for two different parameter sets. MIX function is not unique for a given parameter set, hence the functions shown in this table are merely example functions. The table should be interpreted as follows. The left column consists of the input nibble number. Each nibble has $m$ bits (in the example table $m = 4$). These input bits are distributed to the nibbles given by the indexes in each row. For example, the 4 output bits of input nibble 0 are distributed to the output nibbles 0,4,8 and 12 respectively (Bit 0 to Nibble 0 - any of its 4 input bits, Bit 1 to Nibble 4, Bit 2 to Nibble 8, and Bit 3 to Nibble 12, counting the bits in Big-endian fashion) in Table 1(b). Since the bits are not transformed MIX is a reversible function and hence the composite functions $\hat{g}$ and $\hat{h}$ are also reversible functions.

### 3.3 Achieving Similarity

Now that the function $\hat{h}$ has achieved symmetry, we need to target similarity. In order to achieve similarity the differential property of $\hat{f}$ and $\hat{h}$ needs to be studied. In the universal set $\mathcal{S}_\pi^N$ any input differential causes every bit in the output to switch with a probability of $\frac{2^{N-1}}{2^N-1}$. The cipher function design should achieve *effective similarity* to this switching probability property.

*Bias propagation in $\hat{f}$:* Function $\hat{f}$ operates on the input nibbles independently. For any differential input in a nibble $i$ the output bits of nibble $i$ switch with probability $\frac{2^{m-1}}{2^m-1}$. Let us consider the case when all the input bits of $\hat{f}$ switch independently and uniformly. Let $\delta_I$ represent the switching probability with which any input bit of $\hat{f}$ switches, *i.e.*,

$$\delta_I = \Pr[x^{\mathrm{B}(i)} \neq \bar{x}^{\mathrm{B}(i)}] \, \forall \, 0 \leq i \leq N-1$$

where $x$ and $\bar{x}$ are the differential inputs. Let $\delta_O$ represent the switching probability of output bits of $\hat{f}$. Then,

$$\delta_O = \left( (1 - (\delta_I)^m) \cdot (\frac{2^{m-1}}{2^m - 1}) \right) \tag{1}$$

Let $\delta_I = \frac{2^{N-1}}{2^N-1} - \epsilon_I$ and $\delta_O = \frac{2^{N-1}}{2^N-1} - \epsilon_O$. Let $C_0 = \frac{2^{N-1}}{2^N-1}$ and $C_1 = \frac{2^{m-1}}{2^m-1}$. Given that, $0 < \epsilon_I < C_0$ and $C_0 < C_1$. From Equation 1,

$$\begin{aligned} \epsilon_O &= C_0 - ((1 - (C_0 - \epsilon_I)^m) \, C_1) \\ \epsilon_O - \epsilon_I &= (C_0 - \epsilon_I) - ((1 - (C_0 - \epsilon_I)^m) \, C_1) \\ &= (C_0 - C_1) - (\epsilon_I) - C_1 (C_0 - \epsilon_I)^m \\ &< 0 \end{aligned} \tag{2}$$

Thus the output bias of $\hat{f}$ is always less than the input bias as long as the condition $0 < \epsilon_I < C_0$ is true. We can exploit the bias propagation property of $\hat{f}$ in achieving the *effective* similarity.

*Switching probability of $\hat{h}$ :* Function $\hat{h}$ causes any differential input to pass through $\log_m(N)$ levels of $\Pi$ functions before reaching the output. Thus, the minimum probability with which any output bit of $\hat{h}$ switches is $\left(\frac{2^{m-1}}{2^m-1}\right)^{\log_m(N)}$. Also, every output bit of $\hat{f}$ switches independently with respect to every other output bit. Thus the output of $\hat{h}$ can be passed through $\hat{f}$ multiple times to reduce the bias as close to zero as desired and possible. Since MIX function does not transform the bits the switching probability is not modified by the MIX function. Thus, we define the function $\hat{p}$ as,

$$\begin{aligned} \hat{p} &:= \hat{h} \circ \hat{g}^k \\ &:= \hat{g}^{k+\log_m(N)} \end{aligned}$$

The parameter $k$ is dependent on $N$ and $m$. We choose function $\hat{g}$ instead of $\hat{f}$ as they both have same switching probability characteristics and the resulting function $\hat{p}$ is easier to implement.

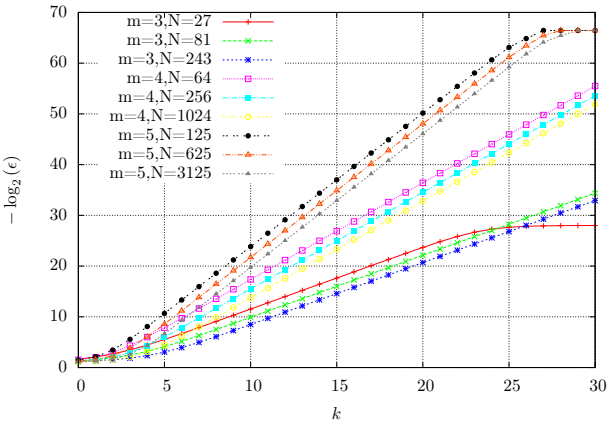


**Fig. 1.** Bias propagation in function $\hat{p}$

The switching probability bias decreases exponentially with respect to $k$. The bias propogation of $\hat{p}$ for various $m$ and $N$ combinations is plotted in Figure 1. Few observations from the graph are:

- The bias decreases *exponentially* with the number of rounds $k$.
- The rate of this decrease is directly proportional to $m$ and is almost independent of $N$.

Note that the *bias* we have estimated is averaged over all the possible bijective functions and all the possible input differential pairs which have their input difference in only one of the nibbles. Let $\epsilon_{b^k}$ represent the average bias per bit observed at the output of $\hat{p}$. Let $\epsilon_{W^k}$ be the overall bias of all the $N$ bits. Since every bit switches independently,

$$\epsilon_{W^k} = \sum_{i=1}^{N} \binom{N}{i} \frac{1}{2^{N-i}} \epsilon_{b^k}^i \tag{3}$$

Let us assume that $\epsilon_{b^k}^i < \frac{1}{N}$ then $N \cdot \epsilon_{b^k}^i < 1$. Also note that $\binom{N}{i} \leq \frac{N^i}{2^{i-1}}$ $\forall\ 1 \leq i \leq N$. Thus, putting these two inequalities in Equation 3 we get the upper bound as,

$$\begin{aligned} \epsilon_{W^k} &< \frac{N \cdot \epsilon_{b^k}}{2^{N-1}} \sum_{i=1}^{N} 1 \\ &< \frac{N^2 \cdot \epsilon_{b^k}}{2^{N-1}} \end{aligned} \tag{4}$$

Thus keeping the bias $\epsilon_{b^k}$ in the range of $\frac{1}{2 \cdot N^3}$ is sufficient to prevent any polynomial time/space experiment to differentiate $\hat{p}$ from the universal set. For $N = 64$ the bias needed is $2^{-19}$ and this is achieved with $k = 12$ rounds. We used TestU01 [10] to verify this heuristically.

**Heuristic Testing:** We implemented the case $N = 64, m = 4, k = 12$ in software and tested the function $\hat{p}$ as a PRF in counter mode. We chose the gates $\Pi_i$ at random from the set $S_\pi^m$. The initial value is a $N$ bit number chosen at random. The IV is then incremented and the output is used as a sequence of *random* bits. TestU01 [10] has different battery of tests. We tested the PRF with *Small Crush*, *Crush*, *Alpha Bit* (for $2^{32}$ bits), *Rabbit* (for $2^{32}$ bits), and NIST FIPS 140-2 tests. We tested for $p$-values lying in the range $[10^{-4}, 1 - 10^{-4}]$. Our PRF passed all the statistical tests in multiple runs.

### 3.4 Achieving Irreversibility

This is the hardest part of the design. We cannot achieve this design goal through theoretical analysis due to lack of adequate frameworks. Hence we target this design goal *empirically* by making the design foolproof against all the known cryptanalytic attacks. Cryptanalysis is done to predict the secret key. Based on the information cryptanalysis uses it can be classified as *known plaintext*, *chosen plaintext* and *related key* attacks. The encryption algorithm should be designed to resist all these forms of cryptanalysis. The conventional wisdom is to design the encryption algorithm in multiple rounds and make every round unique with a unique round key. We will also follow the same design approach.

Slide Attack [9] is the most successful attack on any round based cryptographic function. Slide attack is generic attack designed by Biryukov-Wagner [11,12]. It can be applied in both known plaintext or chosen plaintext scenarios. It exploits the similarity in round functions. Function $\hat{p}$ has two weaknesses which the slide attack can exploit. The first weakness is that all the round functions are the same. The second weakness is that the input bits are operated only on $m$-bit *nibbles* in every round. Thus the function can be attacked by guessing a small fraction of key bits and finding a *slid-pair*.

In order to make the design resistant to slide attacks we need to fix these two weaknesses. Every round function can be made unique by XORing the

output of each round with a *round-key*. The second weakness can be overcome by making these *round-keys* a function of all the *key* bits. Thus the *round-key* generation algorithm should exhibit the following properties:

- Every *round-key* bit should be uniformly dependent on every *key* bit.
- Any *small* difference in key bits should propagate to every *round-key* bit with equal probability.

**Key Bits Representation:** So far in our construction we have used $q$ reversible $m \times m$ gates as our key. Each of these $q$ gates can be represented uniquely by its truth table with $m2^m$ bits. The truth table contains $2^m$ rows of $m$ bits each, in other words, $m$ columns and $2^m$ rows. Each column can be considered as a truth table of an $m$-bit boolean gate. Each of these columns are balanced, *i.e.*, there are equal number of $0's$ and $1's$. Let the representation of the gate be such that these $2^m$ rows of $m$ bits are written sequentially from $0^{th}$ row to the $(2^m - 1)^{th}$ row. Let us represent the truth table of $\Pi$ with $tt^{\Pi}$ and its $i^{th}$ row by $tt_i^{\Pi}$.

$$tt^{\Pi} := (\Pi(0), \Pi(1), \ldots, \Pi(2^m - 1))$$
$$tt_i^{\Pi} := \Pi(i)$$

Then for any two gates $\Pi$ and $\Pi'$ such that $\Pi \neq \Pi'$ at least two truth table rows must differ just to maintain balance, *i.e.*, $\exists\ i, j$ s.t. $i \neq j$, $tt_i^{\Pi} \neq tt_i^{\Pi'}$ and $tt_j^{\Pi} \neq tt_j^{\Pi'}$. Let $K$ be the key bits of size $N2^m$ bits. Let $k_i = K^{\text{NIB}_N(i)}$ and $\Pi_i$ be the $i^{th}$ bijective function represented by the key $K$. In other words,

$$K \equiv (\Pi_0, \Pi_2, \ldots, \Pi_{q-1}) \equiv (tt^{\Pi_0}, tt^{\Pi_1}, \ldots, tt^{\Pi_{q-1}}) \equiv (k_0, k_1, \ldots, k_{2^m-1})$$

Thus the same key can be represented in many equivalent representations. We refer to a gate $\Pi$ through its truth table as $tt^{\Pi}$.

**Round Key Generation:** The number of rounds needed to achieve symmetry was $r = \log_m N + k$. Thus we need to generate $r$ round keys of $N$ bits each. The first property that every round key should satisfy is that it be uniform function of all the key bits. This can be achieved with function $\hat{p}$. In order to achieve the second property we need to modify the construction a little bit. We want to propagate any difference in any one of the gates to all the round keys.

Consider the function $\Phi : \mathbf{K} \mapsto \lambda$ where $K$ is the key as defined earlier and $\lambda = (\lambda_0, \lambda_1, \ldots, \lambda_{r-1})$ is defined as follows:

$$z_{0,j} = \hat{g}(k_{j \,\%\, 2^m}) \forall 0 \leq j < r$$
$$z_{i,j} = \hat{g}(z_{i-1,j} \oplus k_t) \forall 1 \leq i < 2r, 0 \leq j < r, t = (i+j) \,\%\, 2^m$$
$$\lambda_j = z_{2r-1,j} \forall 0 \leq j < r-1$$

Function $\Phi$ takes key $K$ as its input to generate $r$, $N$-bit values.

*Difference Propagation:* Consider two related keys $K$ and $K'$ such that they differ in only one gate $\Pi$, *i.e.*, only one of the bijective functions $\Pi$ is chosen at random while the rest of the functions are the same. This implies that there is at least one $k_i$ such that $Pr[k_i = k_i'] = \frac{1}{2^N}$. From Section 3.3 we see that with $r = \log_m(N) + k$ rounds, any difference in the input propagates to every output bit with a bias less than $\frac{1}{N2^N}$. In function $\Phi$ we exploit this property. For every round key $\lambda_i$, the differential input gets fed in at least $r$ rounds before the output stage. Thus for every pair of related keys which differ in at least one bijective function the function $\Phi$ propagates the difference to every bit of every $\lambda_i$ with equal probability.

We finally define the cipher function $\mathcal{R}$ as having $r = \log_m(N) + k$ rounds with each round function $\hat{u}_i$ as,

$$\hat{u}_i(x) := \hat{g}(x) \oplus \lambda_i$$
$$\mathcal{R} := \hat{u}_{r-1} \circ \hat{u}_{r-2} \circ \cdots \circ \hat{u}_1 \circ \hat{u}_0$$

### 3.5 Strengthening Key Space

The **Maya** key consists of $q$ bijective functions chosen independently at random from the set $S_\pi^m$. Any $m \times m$ bijective function $\Pi$ contains $m$ boolean functions $f_1^\Pi, f_2^\Pi, \ldots, f_m^\Pi$. These $m$ boolen functions satisfy two properties, namely, every one of the $m$ boolean functions is *balanced* (equal number of zeros and ones are present in the truth table) and every one of the $m$ boolean functions is orthogonal to every other boolean function. Both these properties together make the $m$ boolean functions consitute a bijective function.

**Definition 2 (Affine Function)** *A Boolean function $f_k(x_1, x_2, ..., x_m)$ of $m$ variables is called affine if it takes the form of a polynomial $f_k(x) = a_1x_1 \oplus a_2x_2 \oplus \cdots \oplus a_nx_n \oplus c$, where $a_j, c \in GF(2)$ and $k = c + \sum_{i=1}^{n} a_i 2^i$.*

In particular, if $c = 0$ then $f$ is called a linear function. Any linear Boolean function can be generated from Walsh-Hadamard matrix $\mathbf{H}_n$ and affine functions ($c = 1$) can be generated from $-\mathbf{H}_n$. Each row of the Walsh-Hadamard matrix $\mathbf{H}_n$ is a linear function. Each row in $\mathbf{H}_n$, except for the first row which represents the constant function, has equal number of $1's$ and $-1's$.

Affine functions are *weak* from cryptographic perspective. To strengthen our key space we need to choose only those bijective functions which do not constitute any affine functions. First we will count how much of reduction in key space occurs due to this constraint. Let $NL_\pi^m$ represent the set of all bijective functions which do not constitute any *affine* function.

Let $\mathbf{Y}^f = \left[y_0, y_1, ..., y_{2^m-1}\right]^T$ be the two-valued ($\{0, 1\}$) truth vector of $f(x_1, x_2, ..., x_m)$ and $\mathbf{Z}^f = \left[z_0, z_1, \ldots, z_{2^m-1}\right]$ be the recoded two-valued truth vector of the boolean function $f(x_1, x_2, \ldots, x_m)$ such that $z_i = (-1)^{y_i}$, in other words 0 is recoded as 1 and 1 is recoded as $-1$.

**Definition 3 (Orthogonal Boolean Functions)** *The two boolean functions $f_1(x_1, x_2, \ldots, x_m)$ and $f_2(x_1, x_2, \ldots, x_m)$ are orthogonal if the dot products of their recoded truth vectors, $\mathbf{Z}^{f_1} \cdot \mathbf{Z}^{f_2}$, is* zero.

Let $L_\pi^m$ be the set of all bijective functions which constitute at least one affine function. We will calculate the size of $L_\pi^m$ by counting the choices for the functions $f_1$ to $f_m$. *Wlog* let $f_1$ be an affine function. Then there are $2^{m+1} - 2$ choices for $f_1$. $f_2$ has to be balanced and orthogonal with $f_1$. $\mathbf{Z}^{f_1}$ has $2^{m-1}$ $1's$ and $2^{m-1}$ $-1's$. We need $\mathbf{Z}^{f_1} \cdot \mathbf{Z}^{f_2}$ to be *zero*. In $\mathbf{Z}^{f_2}$ half of the $1's$ should be aligned with $1's$ of $\mathbf{Z}^{f_1}$ and the rest should be aligned with $-1's$ of $Z^{f_1}$. Thus there are $\binom{2^{m-1}}{2^{m-2}} \cdot \binom{2^{m-1}}{2^{m-2}}$ functions that can be orthogonal to $f_1$. $f_3$ has to be orthogonal to both $f_1$ and $f_2$. Extending the same logic we get the count as

$$|L_\pi^m| = \left(2^{m+1} - 2\right) \cdot \left(\binom{2^{m-1}}{2^{m-2}}\right)^2 \cdot \left(\binom{2^{m-2}}{2^{m-3}}\right)^4 \cdots \left(2\right)^{2^{m-1}} \tag{5}$$

For $m = 4$ we get an upper bound of $2^{36}$ bijective functions which constitute at least one affine function. The total space of bijective functions for $m = 4$ is approximately $2^{44}$ leading to key space reduction of $|NL_\pi^4| = 2^{44} - 2^{36} \approx 2^{43.9944}$. Thus the key space reduction is negligible.

### 3.6 Alternate View

The final design we have arrived at looks very similar to any SPN based encryption algorithm (e.g.: Rijndael) except that the substitution boxes themselves are chosen by the *key* bits. This is the major distinction between our approach and the conventional design approaches. The similarity of our design to the conventional designs buys us the strength (in fact increases the strength) of the encryption algorithms whereas the differences buy us tremendous improvements both in performance and area for hardware implementations.

## 4 Design Specification

In this section we will formally specify the cipher function **Maya**(64,4) based on the intuitive design presented in Section 3. **Maya**(64,4) is a 64-bit block cipher which uses 4-bit bijective functions. We choose to specify **Maya**(64,4) instead of a generic family of functions **Maya**($N$,$m$) as we can find tighter bounds for the specific instance. Our future work will involve generalizing the results for the family of functions **Maya**($N$,$m$).

**Construction 1 (Maya(64,4))** ***Maya(****64,4****)*** *is a triple* $(\mathbf{G}_4^{64}, \mathbf{E}_4^{64}, \mathbf{D}_4^{64})$, *where* $\mathbf{G}_4^{64}$ *is the key-generator algorithm,* $\mathbf{E}_4^{64}$ *is the encryption algorithm and* $\mathbf{D}_4^{64}$ *is the decryption algorithm.*

**Construction 2** ($\alpha : \mathbf{K} \times \mathbf{X} \mapsto \mathbf{Y}$) *Where*

$$K = (\Pi_0, \Pi_1, \ldots, \Pi_{15}),\ \Pi_i \in NL_\pi^4 \forall 0 \leq i \leq 15$$
$$X, Y \in I_{64}$$

$$Y := \alpha(X)$$
$$Y^{\text{NIB}_4(i)} := \Pi_i(X^{\text{NIB}_4(i)}),\ \textit{where}\ 0 \leq i \leq 15.$$

**Construction 3** ($\mu : \mathbf{X} \mapsto \mathbf{Y}$) *Where* $X, Y \in I_{64}$. *The* $\mu$ *function is a bit-permutation network. The input bit position to output bit position connection network is as shown in the Table 2(a) and its inverse is shown in Tabel 2(b).*

**Table 2.** Bit Permutation Function

(a) Function $\mu$

| {Inp Bit Pos} → {Out Bit Pos} |
|---|
| $\{00,01,02,03\} \rightarrow \{00,16,32,48\}$ |
| $\{04,05,06,07\} \rightarrow \{04,20,36,52\}$ |
| $\{08,09,10,11\} \rightarrow \{08,24,40,56\}$ |
| $\{12,13,14,15\} \rightarrow \{12,28,44,60\}$ |
| $\{16,17,18,19\} \rightarrow \{21,37,53,05\}$ |
| $\{20,21,22,23\} \rightarrow \{25,41,57,09\}$ |
| $\{24,25,26,27\} \rightarrow \{29,45,61,13\}$ |
| $\{28,29,30,31\} \rightarrow \{33,49,01,17\}$ |
| $\{32,33,34,35\} \rightarrow \{42,58,10,26\}$ |
| $\{36,37,38,39\} \rightarrow \{46,62,14,30\}$ |
| $\{40,41,42,43\} \rightarrow \{50,02,18,34\}$ |
| $\{44,45,46,47\} \rightarrow \{54,06,22,38\}$ |
| $\{48,49,50,51\} \rightarrow \{63,15,31,47\}$ |
| $\{52,53,54,55\} \rightarrow \{03,19,35,51\}$ |
| $\{56,57,58,59\} \rightarrow \{07,23,39,55\}$ |
| $\{60,61,62,63\} \rightarrow \{11,27,43,59\}$ |

(b) Function $\mu^{-1}$

| {Inp Bit Pos} → {Out Bit Pos} |
|---|
| $\{00,01,02,03\} \rightarrow \{00,30,41,52\}$ |
| $\{04,05,06,07\} \rightarrow \{04,19,45,56\}$ |
| $\{08,09,10,11\} \rightarrow \{08,23,34,60\}$ |
| $\{12,13,14,15\} \rightarrow \{12,27,38,49\}$ |
| $\{16,17,18,19\} \rightarrow \{01,31,42,53\}$ |
| $\{20,21,22,23\} \rightarrow \{05,16,46,57\}$ |
| $\{24,25,26,27\} \rightarrow \{09,20,35,61\}$ |
| $\{28,29,30,31\} \rightarrow \{13,24,39,50\}$ |
| $\{32,33,34,35\} \rightarrow \{02,28,43,54\}$ |
| $\{36,37,38,39\} \rightarrow \{06,17,47,58\}$ |
| $\{40,41,42,43\} \rightarrow \{10,21,32,62\}$ |
| $\{44,45,46,47\} \rightarrow \{14,25,36,51\}$ |
| $\{48,49,50,51\} \rightarrow \{03,29,40,55\}$ |
| $\{52,53,54,55\} \rightarrow \{07,18,44,59\}$ |
| $\{56,57,58,59\} \rightarrow \{11,22,33,63\}$ |
| $\{60,61,62,63\} \rightarrow \{15,26,37,48\}$ |

**Construction 4** ($\gamma : \mathbf{K} \times \kappa \mapsto \lambda$) *Where*

$$
\begin{aligned}
K &= (\Pi_0, \Pi_1, \ldots, \Pi_{15}),\ \Pi_i \in NL_\pi^4 \forall 0 \leq i \leq 15 \\
\kappa &= (\kappa_0, \kappa_1, \ldots, \kappa_{15}),\ \kappa_i \in I_{64}\ \forall 0 \leq i \leq 15 \\
\lambda &\in I_{64}
\end{aligned}
$$

$$
\begin{aligned}
z_{-1} &:= 0 \\
z_i &:= \mu(\alpha(K, z_{i-1} \oplus \kappa_t)),\ t = i\%16,\ \forall 0 \leq i \leq 31 \\
\lambda &:= z_{31}
\end{aligned}
$$

**Construction 5** ($\mathbf{G}_4^{64} : \mathbf{K} \mapsto (\mathbf{K_e}, \mathbf{K_d})$) *Where*

$$
\begin{aligned}
K &= (\Pi_0, \Pi_1, \ldots, \Pi_{15}),\ \Pi_i \in NL_\pi^4 \forall 0 \leq i \leq 15 \\
&\equiv (k_0, k_1, k_2, \ldots, k_{15}),\ k_i \in I_{64} \forall 0 \leq i \leq 15 \\
K_e &= (\Pi_0^e, \Pi_1^e, \ldots, \Pi_{15}^e, \lambda_0^e, \lambda_1^e, \ldots, \lambda_{15}^e),\ \Pi_i^e \in NL_\pi^4, \lambda_j^e \in I_{64} \forall 0 \leq i \leq 15 \\
K_d &= (\Pi_0^d, \Pi_1^d, \ldots, \Pi_{15}^d, \lambda_0^d, \lambda_1^d, \ldots, \lambda_{15}^e),\ \Pi_i^d \in NL_\pi^4, \lambda_j^d \in I_{64} \forall 0 \leq i \leq 15
\end{aligned}
$$

$$
\begin{aligned}
\Pi_i^e &:= \Pi_i \\
\Pi_i^d &:= \Pi_i^{-1} \\
\kappa^i &:= (\kappa_0^i, \kappa_1^i, \ldots, \kappa_{15}^i), \forall 0 \leq i \leq 15 \\
\kappa_k^i &:= k_t,\ t = (i+k)\%16 \\
\lambda_j^e &:= \gamma(K, \kappa^j) \\
\lambda_j^d &:= \lambda_{15-j}^e
\end{aligned}
$$

In the construction of $\mathbf{G}_4^{64}$ to generate the $\lambda$, we use the equivalent representation of the key bits as explained in Section 3.4.

**Construction 6** ($\mathbf{E}_4^{64} : \mathbf{K_e} \times \mathbf{X} \mapsto \mathbf{Y}$) *Where*

$K_e = (\Pi_0^e, \Pi_1^e, \ldots, \Pi_{15}^e, \lambda_0^e, \lambda_1^e, \ldots, \lambda_{15}^e)$, $\Pi_i^e \in NL_\pi^4$, $\lambda_j^e \in I_{64}$, $\forall 0 \leq i \leq 15$
$X, Y \in I_{64}$.

$$
\begin{aligned}
Y &:= \mathbf{E}_4^{64}(K_e, X) \\
Z_{-1} &:= X \\
K_\pi^e &:= (\Pi_0^e, \Pi_1^e, \ldots, \Pi_{15}^e) \\
Z_i &:= \mu(\alpha(K_\pi^e, Z_{i-1})) \oplus \lambda_i^e \, \forall 0 \leq i \leq 15 \\
Y &:= Z_{15}
\end{aligned}
$$

**Construction 7** ($\mathbf{D}_4^{64} : \mathbf{K_d} \times \mathbf{X} \mapsto \mathbf{Y}$) *Where*

$K_d = (\Pi_0^d, \Pi_1^d, \ldots, \Pi_{15}^d, \lambda_0^d, \lambda_1^d, \ldots, \lambda_{15}^d)$, $\Pi_i^d \in NL_\pi^4$, $\lambda_j^d \in I_{64}$, $\forall 0 \leq i \leq 15$
$X, Y \in I_{64}$.

$$
\begin{aligned}
Y &:= \mathbf{D}_4^{64}(K_d, X) \\
Z_{-1} &:= X \\
K_\pi^d &:= (\Pi_0^d, \Pi_1^d, \ldots, \Pi_{15}^d) \\
Z_i &:= \alpha(K_\pi^d, \mu^{-1}(Z_{i-1} \oplus \lambda_i^d)) \, \forall 0 \leq i \leq 15 \\
Y &:= Z_{15}
\end{aligned}
$$

## 5 Analysis

### 5.1 Bijective Function Set ($NL_\pi^4$)

The set $NL_\pi^4$ comprises of all *non-linear* functions of 4 input and 4 output bits. The degree [13] of any function $\Pi \in NL_\pi^4$ is given by

$$\begin{aligned} \deg(\Pi) &= \max_{0 \le i < 4} \deg(f_i^\Pi) \\ &\ge 2. \end{aligned} \tag{6}$$

This follows from our choice of nonlinear functions in Section 3.5. Thus the overall degree of a cipher function from Maya family is

$$\begin{aligned} \deg(\textbf{Maya}) &\ge (\deg(\Pi))^r \\ &= 64. \end{aligned} \tag{7}$$

Thus every instance of a **Maya** cipher function has high algebraic degree. Thus algebraic attacks on **Maya** have very small chance of success.

### 5.2 Key Size

**Maya**$(64, 4)$ has 16 $4 \times 4$ bijective functions. Each of these bijective functions is represented by its 64 bits truth-table. Thus the key size is 1024 bits. But the effective key size is $16 * 43.99 \approx 703$ bits based on our estimate of $|NL_\pi^4|$ in Section 3.5. This effective key size is still larger than the contemporary cipher key sizes.

### 5.3 Rounds

In Section 3.3 we showed the relationship between *bias* and number of rounds in **Maya**. A bit-level bias of $\frac{1}{2N^3}$ achieves overall bias less than $\frac{1}{N2^N}$. For $N = 64$ and $m = 4$ the number of rounds required to achieve this is 15 ($k = 12$, from Figure 1). One of the properties of a bijective function is that by observing output bits one can infer whether the input was colliding or not. This property makes the last round redundant for an adversary with access to the output bits. Thus we increased the number of rounds to 16 to take care of this redundancy.

### 5.4 Security

The fundamental question that needs to be answered in any crypto-system is the difficulty of extracting the secret. In **Maya** the secret or *key* consists of 16 bijective functions. Thus any cryptanalysis will have to find *full* or *partial* truth-table bits. In other words any successful attack on **Maya** should be able to answer the question $y \stackrel{?}{=} \Pi_i(x)$. While we don't have a theoretical proof yet we believe that this is an NP-Complete problem. But given that S-boxes themselves are secret it would be much harder to infer the secret in **Maya** compared to any other algorithm whose S-boxes are public.

### 5.5 Cryptanalysis

Two properties of **Maya** make it resistant to conventional cryptanalysis methods, namely,

1. Every round is dependent on all the key bits, *i.e.*, attacks which depend on guessing small number of *round key* bits fail.
2. The S-boxes themselves are chosen by the key bits thus making the analysis dependent on the complete set.

*Linear Cryptanalysis: Linear cryptanalysis* is a general form of cryptanalysis based on finding affine approximations to the cipher function. The technique [14] has been applied to attack ciphers like FEAL [15] and DES [3]. Linear cryptanalysis exploits the high probability of occurrences of linear expressions involving *plaintext*, *ciphertext*, and *sub-key* bits. This attack becomes possible on the *conventional* cipher function design as the *key* bits are primarily XOR'ed with round inputs. The linear approximations of known components (S-boxes) in the system further aid the analysis. In the case of **Maya** none of these favorable conditions are present. The completeness property of $NL^4_\pi$, *i.e.*, for every $\Pi \in NL^4_\pi$ its complement ($\bar{\Pi}$) is also present, makes the bias of every linear approximation zero.

*Differential Cryptanalysis: Differential cryptanalysis* [16] exploits the property of difference being propagated from input to the output of the cipher function. This attack again requires the properties of the known components in the system (S-boxes) in order to estimate the difference propagation probabilities. In **Maya**, the S-boxes are chosen at random from the

set $NL_\pi^4$ thus requiring any differential analysis to be done on this whole set. But the set $NL_\pi^4$ is complete with respect to differential analysis in the sense that every differential pair is equally probable. Thus differential cryptanalysis is not feasible against **Maya**. A variant of this attack is *impossible difference attack* [17] which again uses the principle of identifying differences that do not propagate from input to output.

*Boomerang Attack:* This attack [18] relies on the difference analysis of round function properties and existence of some block in the system which is independent of the input to cipher function. This can be thought of as meet-in-the middle version of differential cryptanalysis. Again **Maya** is resistant to this attack as there are no blocks (gates) in the system that are independent of either the key or the input.

*Sliding Attack:* This attack [11] exploits the weakness of the round functions. It assumes that given two pairs $P, C$ and $P', C'$ such that $P' = f(P)$ and $C' = f(C)$, the round function can be deciphered or at least a significant fraction of key bits can be extracted. These attacks once again use the property of round functions being built using some known components (S-boxes) and *key* bits being used only in XOR operations. **Maya** has unique round keys which make every round function unique thus resisting slide attack.

*Related Key Attack:* Round key generation algorithm of **Maya** mixes the key bits in such a way that small difference in the key bits gets propagated to all the round key bits.

## 6 Implementation

One of the main advantages of **Maya** is that the design is easily implementable in hardware hence should have better efficiency results compared to the conventional cipher algorithms. We used AES-128 bit implementation as the basis for our comparison as this is the most widely used cipher function.

We implemented **Maya** in VHDL and used the 128-bit AES from Opencores [19]. The designs are synthesized, placed and routed using Xilinx ISE 9.1 [20] targeting Virtex-4 xc4vlx100-12ff1148 with Block RAM turned off. We maximize the clock frequency to a value that ISE can place and

**Table 3.** FPGA Implementation Results

(a) **Maya**(64,4)

| Rounds per Stage | Period ns | Latency ns | Gate Equiv. $10^3$ | Throughput $10^9$bits/s | Energy/bit $10^{-12}$J/bit |
|---|---|---|---|---|---|
| 16 | 21.5 | 21.5 | 21.0 | 2.98 | 429.34 |
| 8 | 11.0 | 22.0 | 21.5 | 5.82 | 261.89 |
| 4 | 6.5 | 26.0 | 22.5 | 9.85 | 192.73 |
| 2 | 4.5 | 36.0 | 24.6 | 14.22 | 160.76 |
| 1 | 3 | 48.0 | 28.7 | 21.33 | 144.03 |

(b) AES-128

| Rounds per Stage | Period ns | Latency ns | Gate Equiv. $10^3$ | Throughput $10^9$bits/s | Energy/bit $10^{-12}$J/bit |
|---|---|---|---|---|---|
| 10 | 50.0 | 50.0 | 293.4 | 2.56 | 538.94 |
| 5 | 40.0 | 80.0 | 272.9 | 3.20 | 486.17 |
| 2 | 29.0 | 145.0 | 276.4 | 4.41 | 370.61 |
| 1 | 28.0 | 280.0 | 283.1 | 4.57 | 390.83 |

route without timing violation. We updated AES design to make it run in pipelined mode. In order to get the measure of total area consumed by the logic, Gate Equivalent metric is chosen. This data approximates the number of gates needed in ASIC. We also measure the power consumption of the FPGA using Xilinx XPower. The results are presented in Tables 3(a) and 3(b).

One of **Maya**'s main advantages in FPGA implementation is that we can make use of the native 4-LUTs to store our key and perform multiplexer function. This greatly reduces the number of LUTs needed compared with AES FPGA implementation. From the results, we can see that an order of magnitude improvement with respect to speed and area. Any configuration of **Maya** requires only 7% of the gates needed for the minimal configuration of the AES. The efficiency of design is reflected in the parameter *throughput per gate*, which for **Maya** is 46 times higher than that of AES. This shows that every gate is used much more effectively in **Maya** design. Even with a higher throughput than that of AES the energy consumed by **Maya** is less than half of that of AES. This proves our contention that **Maya** design is inherently suitable for hardware implementation.

AES throughput can be increased further by increasing the number of pipeline stages [21,22,23,24,25]. The highest throughput design is presented by Zambreno et al. [25] implemented on XC2V4000 FPGA by unrolling and heavily pipelining the AES design with Distributed ROM prim-

itives. The throughput achieved by this design is 23.57Gbps using 16938 slices which is 1.39Mbps/slice. Our **Maya** design achieves a throughput of 21.33Gps using 2050 slices, *i.e.*, 10.4Mbps/slice which is 7.49 times more efficient than the best known pipelined design for AES.


## Acknowledgments

The authors wish to thank Prof. Ahkilesh Tyagi of Iowa State University for his help in the design and analysis of **Maya** algorithm. The authors wish to thank Ka-Ming Keung of Iowa State University for his help in the VHDL design and synthesis of **Maya** and AES-128 ciphers.


## 7 Conclusion

We have argued for a block cipher design based on S-boxes or gates derived from the secret, but with a published dataflow architecture. This is a novel design methodology for Block Cipher functions which results in better hardware implementation efficiency, and increased cryptanalysis resistance. Our design has used key-dependent S-boxes to the limit. We have designed a specific cipher,**Maya**, based on this methodology. We analyzed the properties of the **Maya** and argued its resistance to conventional cryptanalytic techniques. The fundamental advantages of our methodology are increased secret size in relation to block size and much more efficient hardware implementation compared to the conventional cryptographic design. As a proof of concept we show that the FPGA implementation of **Maya** is an order of magnitude efficient in every design parameter compared to 128-bit AES implementation.

## References


1. Canteaut, A., Augot, D., Biryukov, A., Braeken, A., Cid, C., Dobbertin, H., Englund, H., Gilbert, H., Granboulan, L., Handschuh, H., Hell, M., Johansson, T., Maximov, A., Parker, M., Pornin, T., Preneel, B., Robshaw, M., Ward, M.: Ongoing research areas in symmetric cryptography. In: ECRYPT Report. (Jan 2006)
2. National Institute of Standards and Technology: FIPS PUB 197: Advanced Encryption Standard (AES). (Nov 2001)
3. National Institute of Standards and Technology: FIPS PUB 46-3: Data Encryption Standard (DES). (Oct 1999)
4. Anderson, R.J., Biham, E., Knudsen, L.R.: The case for serpent. In: AES Candidate Conference. (2000) 349–354





mgomathi,rblee@princeton.edu

5. Lie, D., Thekkath, C.A., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J.C., Horowitz, M.: Architectural support for copy and tamper resistant software. In: Architectural Support for Programming Languages and Operating Systems. (2000) 168–177
6. Suh, G., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: aegis: Architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17 Int'l Conference on Supercomputing. (2003) 160–171
7. Wikipedia Article: GOST 28147-89. (1970-)
8. Wikipedia Article: TREYFER. (1997)
9. Biryukov, A.: Slide attack. In van Tilborg, H.C.A., ed.: Encyclopedia of Cryptography and Security. Springer (2005)
10. L'Ecuyer, P., Simard, R.J.: Testu01: A c library for empirical testing of random number generators. ACM Trans. Math. Softw. **33**(4) (2007)
11. Biryukov, A., Wagner, D.: Slide attacks. In Knudsen, L.R., ed.: FSE. Volume 1636 of Lecture Notes in Computer Science., Springer (1999) 245–259
12. Biryukov, A., Wagner, D.: Advanced slide attacks. In: EUROCRYPT. (2000) 589–606
13. Canteaut, A., Videau, M.: Degree of composition of highly nonlinear functions and applications to higher order differential cryptanalysis. In: EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, London, UK, Springer-Verlag (2002) 518–533
14. Matsui, M.: Linear Cryptoanalysis Method for DES Cipher. In: EUROCRYPT. (1993) 386–397
15. Miyaguchi, S.: The FEAL Cipher Family. In: CRYPTO. (1990) 627–638
16. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology **4**(1) (1991) 3–72
17. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. J. Cryptology **18**(4) (2005) 291–311
18. Wagner, D.: The boomerang attack. In: Fast Software Encryption. (1999) 156–170
19. OpenCores: AES IP Core
20. Xilinx Incorporated: Xilinx ISE 9.1
21. Feldhofer, M., Lemke, K., Oswald, E., Standaert, F.X., Wollinger, T., Wolkerstorfer, J.: State of the art in hardware architectures. Technical Report D.VAM.2, ECRYPT, European Network of Excellence in Cryptology (September 2005)
22. Chodowiec, P., Khuon, P., Gaj, K.: Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining. In: FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays, New York, NY, USA, ACM (2001) 94–102
23. Hodjat, A., Verbauwhede, I.: A 21.54 gbits/s fully pipelined aes processor on fpga. Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on (April 2004) 308–309
24. Zhang, X., Parhi, K.K.: High-speed vlsi architectures for the aes algorithm. IEEE Trans. Very Large Scale Integr. Syst. **12**(9) (2004) 957–967
25. Zambreno, J., Nguyen, D., Choudhary, A.N.: Exploring area/delay tradeoffs in an aes fpga implementation. In: FPL. (2004) 575–585