# CBEAM: Efficient Authenticated Encryption from Feebly One-Way $\phi$ Functions

Markku-Juhani O. Saarinen

Kudelski Security, Switzerland
mjos@cblnk.com

**Abstract.** We show how efficient and secure cryptographic mixing functions can be constructed from low-degree rotation-invariant $\phi$ functions rather than conventional S-Boxes. These novel functions have surprising properties; many exhibit inherent feeble (Boolean circuit) one-wayness and offer speed/area tradeoffs unobtainable with traditional constructs. Recent theoretical results indicate that even if the inverse is not explicitly computed in an implementation, its degree plays a fundamental role to the security of the iterated composition. To illustrate these properties, we present CBEAM, a Cryptographic Sponge Permutation based on a single $5 \times 1$-bit Boolean function. This simple nonlinear function is used to construct a 16-bit rotation-invariant $\phi$ function of Degree 4 (but with a very complex Degree 11 inverse), which in turn is expanded into an efficient 256-bit mixing function. In addition to flexible tradeoffs in hardware we show that efficient implementation strategies exist for software platforms ranging from low-end microcontrollers to the very latest x86-64 AVX2 instruction set. A rotational bit-sliced software implementation offers not only comparable speeds to AES but also increased security against cache side channel attacks. Our construction supports Sponge-based Authenticated Encryption, Hashing, and PRF/PRNG modes and is highly useful as a compact "all-in-one" primitive for pervasive security.

**Keywords:** CBEAM, Authenticated Encryption, Cryptographic Sponge Functions, Trapdoor $\phi$ functions, Lightweight Cryptography.

## 1 Introduction

The only nonlinear component of the SHA-3 algorithm KECCAK [1, 2] is not a traditional S-Box but a rotation-invariant $\phi$ function [3]. It has been widely observed [4] that this $5 \times 5$ - bit function, $\chi$, has a lower algebraic degree and circuit complexity than its inverse $\chi^{-1}$ (See Figure 1). This is a desirable quality in a Sponge-based cryptoprimitive as computation of inverse is not required in normal operation. Boura and Canteaut have showed that complex inverse makes the resulting iteration strong even if it is not explicitly computed [5]. We have discovered new functions of $\phi$ type which exhibit much more radical asymmetry than the $\chi$ function of KECCAK.

Sponge-based constructions offer perhaps the best route to shared-resource (combined encryption and MAC state) authenticated encryption via the Duplex construction [6–10]. This motivates our investigation of higher-degree $\phi$ functions as we believe that they are better suited for Sponge constructions than traditional block cipher design methodologies that require efficient computation in both directions.

**Our contributions and structure of this paper.** We first give some basic observations on $\phi$ functions and their cryptanalysis in Section 2. Inspired by our discovery of a unique, particularly strong 5-input $\phi$ function, we propose a cryptographic permutation named CBEAM which can be used for hashing, authenticated encryption, and other purposes. Section 3 gives a formal definition of the CBEAM Sponge Permutation $\pi$, followed by analysis in Section 4.

This "Cryptographic Swiss Army Knife" Sponge primitive uses a fast $16 \times 16$ -bit $\phi$ function of Degree 4, with 13 terms in its ANF polynomial for each output bit. Its asymmetry is evident as its inverse has degree 11 and 13465 terms for output each bit – see Section 4.3 and Appendix D.

Based on extensive experimentation we conjecture that these functions exhibit inherent feeble one-wayness as defined by Hiltgen for circuit complexity. This indicates high algebraic resistance for our construct [5] and shows that $\phi$ functions are in some ways superior to conventional designs based on S-Box lookups.

In Section 5 we describe implementations of CBEAM for x86-64 AVX2 instruction set and for the 16-bit MSP 430 ultra-low power microcontroller. CBEAM is as fast as fastest AES implementations (without dedicated AES hardware) on both of these platforms, but has significantly smaller implementation footprint on both. Significant area-speed trade-offs are possible in hardware, as demonstrated by our two reference VHDL implementations.

Our conclusions in Section 6 are followed by test vectors and cryptanalytic tables in Appendices.

## 2 Rotation-Invariant $\phi$ Functions

Introduced in Daemen's 1995 PhD Thesis [3], $\phi$ functions are rotation-invariant $n$-bit invertible (bijective) functions. We use a slightly different notation from Daemen who used $\phi$ to denote non-invertible as well as invertible rotation-invariant functions.

**Definition 1.** *Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a function from $n$-bit vectors to $n$-bit vectors. $f$ is a $\phi$ function if it is bijective (uniquely invertible) and rotation-invariant: $f(x) = y \Rightarrow f(x \lll r) = y \lll r$ for all $r$.*

**Lemma 1.** *Any $n \times n$-bit $\phi$ function $f$ is unambiguously characterized by an $n \times 1$ - bit function $f_{(1)}$ that satisfies $f_{(1)}(x) = f(x) \wedge 1$.*

*Proof.* Directly from rotation invariance. Constant 1 has Hamming weight 1. □

Each output bit of the function may be dependent only on some subset of $n$ input bits. This subset is not arbitrary; we found that neighboring input bits are more likely to yield invertible functions. In the present work $\phi_5$ is a specific $5 \times 1$ - bit function and $\phi_{16}$ is a $16 \times 16$ - bit function defined by it as per Lemma 1. We note that each output bit of the inverse function $f^-1$ may be dependent on all input bits even though this is not the case for $f$ (See Figure 1.)

### 2.1 Invertibility

It is easy to see that there are invertible $n \times n$ - bit $\phi$ functions for any $n > 1$ by considering $f(x) = cx \pmod{2^n - 1}$, where $\gcd(c, 2^n - 1) = 1$. Rotation invariance: $2^n \equiv 1 \pmod{2^n - 1}$ and $f(2^k x) = 2^k cx \pmod{2^n - 1}$. For invertibility $f^{-1}(x) = c^{-1}x \pmod{2^n - 1}$.

The inverse function $f^{-1}$ can also be characterized by an $n \times 1$ - bit function $f^{-1}_{(1)}$ (Lemma 1) since the inverse of any $\phi$ function is clearly also a $\phi$ function. It may also be the case that $f = f^{-1}$. Hummingbird-2$\nu$ is an example of a cipher that utilizes two 16-bit $\phi$ functions which are in fact involutions [11]. The SIMON family of block ciphers from NSA is an example of a cipher that utilizes a *non-surjective* rotation-invariant function $f$ as part of a Feistel construction [12].

It is nontrivial to characterize which one-bit $f_{(1)}$ functions generate invertible $f$ functions apart from simple properties such as bit balance: $\sum_{x=0}^{2^n-1} f_{(1)}(x) = 2^{n-1}$. Good $\phi$ functions appear to be rather hard to find – we resorted to optimized exhaustive tabulation methods to find our implementation-friendy and "feebly asymmetric" $\phi_5$.

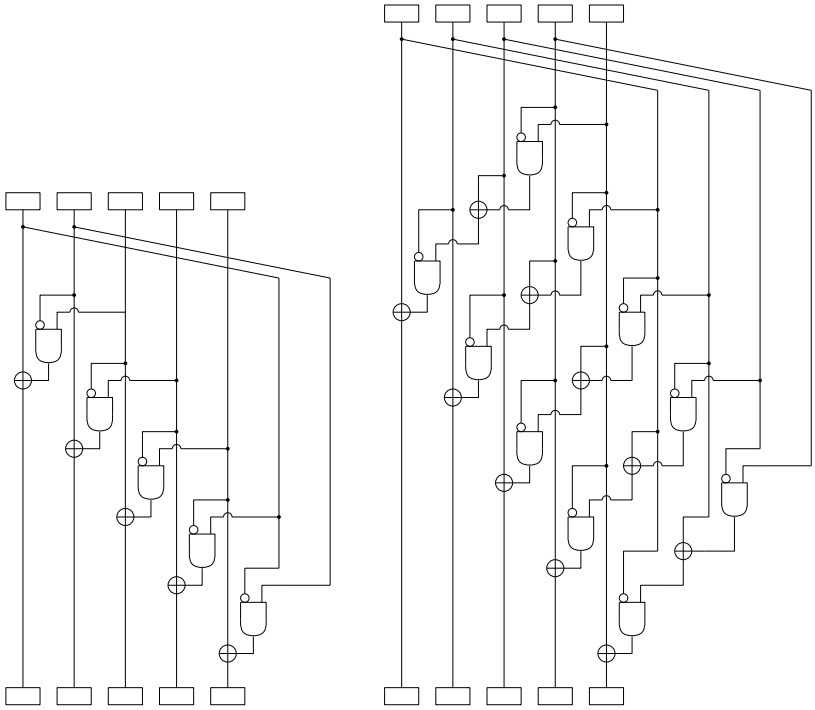

**Fig. 1.** On left, a circuit implementing KECCAK's $5 \times 5$ - bit $\chi$ component, which happens to be a rotation-invariant $\phi$ function of degree 2. On right, a circuit implementing its inverse permutation, $\chi^{-1}$, which has Degree 3 with each output bit dependent on all input bits. Such asymmetric Boolean and circuit complexity is characteristic of $\phi$ functions.

### 2.2 On Cryptanalysis of $\phi$ Functions

Algorithms for finding differential [13] and linear [14, 15] cryptanalytic properties of a $\phi$ function are relatively fast and straightforward to implement. Thanks to Lemma 1, when determining linear bounds we may assume that the input mask is a subset of the input bits to its $f_{(1)}$.

For differential cryptanalysis we must consider the convolution of the input differential w.r.t. a single output bit. Due to rotation we may always by convention set the bit at index 0 in the input differential.

Countermeasures must be taken against rotational cryptanalysis [16] due to inherent rotational invariance of $\phi$ functions. Algebraically these functions have surprising properties. See Section 4.3 and Appendix D for tables and conjectures related to $\phi_5$.

### 2.3 General Implementation Features

One the most useful features of $\phi$ functions is the extreme amount of implementation trade-offs allowed. Computation of a $n \times n$ - bit $\phi$ function can take anywhere from 1 (fully unrolled) to $c \times n$ cycles (serial implementation – here $c$ is some constant), depending on target hardware platform. This is illustrated in Figure 3.

On software platform, $\phi$ functions allow efficient implementation of large "S-boxes" via a Boolean sequence programming technique resembling bit-slicing [17]. Finding a good bit-slicing Boolean description for an $n \times 1$ - bit function is much easier than for a generic $n \times n$ - bit S-Box.

Such straight-line code is resistant to cache-based side-channel timing attacks such as those reported against AES implementations [18–20].

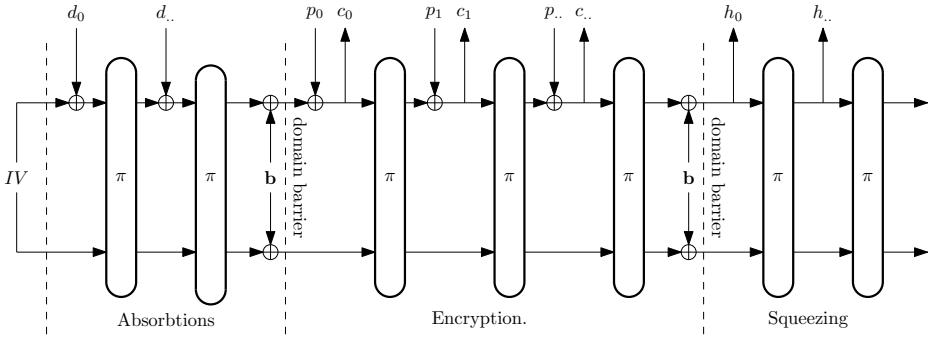


**Fig. 2.** A simplified view of a generic Sponge construction. The state is first loaded with an Initialization Vector or the final state of previous message. In CBEAM, the mixing function is $\pi = \mathsf{mx}^6$. Then Secret Key, Nonce, and Associated Authenticated Data (AAD) are absorbed and mixed - all represented by $d$ words. $\mathbf{b}$ represents some domain separating padding mechanism. The same $\pi$ function is then be used to encrypt and decrypt data and finally to extract ("squeeze") out a MAC or a hash $h$.

## 3 CBEAM and its $\pi$ Permutation

The design of CBEAM was driven by the following goals:

1. KISS: A simple design based on a single feebly one-way unkeyed permutation.
2. Fulfills all symmetric cryptographic needs of a communications security suite with a single core primitive. Usable as a Pseudorandom Function, Authenticated Encryption Algorithm, and a Collision-resistant Hash.
3. Have high performance on high-end CPUs, yet be efficiently implementable on low-end MCUs and lightweight hardware platforms such as RFID.
4. Have a high security level against attacks ($2^{128}$).

CBEAM is based on the Sponge construction (Section 3.2) with a 256-bit state size and 64-bit data rate; data transfer generally occurs in 64-bit increments.

### 3.1 Mixing Function mx

The basic building block of CBEAM is mx, which is a bijective transform on a 256-bit state variable. Six rounds of mx make up $\pi$, the fundamental permutation of CBEAM.

The mixing function mx is composed of addition of a round constant $\mathsf{rc}^r$, bit matrix transpose, linear mixing $\lambda$, and nonlinear mixing $\phi$:

$$\mathsf{mx}_r(\mathbf{s}) = (\phi \circ \lambda)(\mathbf{s} \oplus \mathsf{rc}^r)^T. \tag{1}$$

Practical software implementation notes are presented in Section 5.2 and a test trace of six rounds in Appendix A.

**Formal definition.** We index the state $\mathbf{s}$ interchangeably as a $16 \times 16$ - bit matrix $s[\,0..15\,][\,0..15\,]$, a vector of 16-bit words $s_w[\,0..15\,]$ with $s_w[\,i\,] = \sum_{j=0}^{15} 2^j s[\,i\,][\,j\,]$ or as four quadwords $s_q[\,0..3\,]$ with $s_q[\,i\,] = \sum_{j=0}^{3} 2^{16j} s_w[\,4i+j\,]$. All data is stored in little-endian format.

In the following description modulo 16 arithmetic in indexing is equivalent to logical masking with 0xF; $a \bmod 16$ is always in the range $0, 1, \cdots, 15$. To evaluate $\pi = \mathsf{mx}^6$ we compute six rounds $r = 0 \ldots 5$ of the following three steps:

**1. Round Constant $\mathsf{rc}^r$.** Let the individual round bits be $r = 4r_2 + 2r_1 + r_0$. We have $s'[\,i\,][\,j\,] = s[\,i\,][\,j\,]$ for all $0 \le i, j \le 15$ except the following:

$$\begin{aligned}
s'[\,0\,][\,0\,] &= s[\,0\,][\,0\,] \oplus (r_0 \wedge \neg r_1) & \qquad s'[\,8\,][\,2\,] &= s[\,8\,][\,2\,] \oplus (r_0 \wedge r_1)\\
s'[\,1\,][\,0\,] &= s[\,1\,][\,0\,] \oplus (r_0 \wedge r_2) & \qquad s'[\,10\,][\,2\,] &= s[\,10\,][\,2\,] \oplus r_0\\
s'[\,3\,][\,0\,] &= s[\,3\,][\,0\,] \oplus r_0 & \qquad s'[\,11\,][\,2\,] &= s[\,11\,][\,2\,] \oplus (r_0 \wedge r_2)\\
s'[\,4\,][\,1\,] &= s[\,4\,][\,1\,] \oplus r_0 & \qquad s'[\,13\,][\,3\,] &= s[\,13\,][\,3\,] \oplus r_0\\
s'[\,5\,][\,1\,] &= s[\,5\,][\,1\,] \oplus (r_0 \wedge \neg r_1) & \qquad s'[\,14\,][\,3\,] &= s[\,14\,][\,3\,] \oplus (r_0 \wedge r_1)\\
s'[\,6\,][\,1\,] &= s[\,6\,][\,1\,] \oplus (r_0 \wedge r_2) & \qquad s'[\,15\,][\,3\,] &= s[\,15\,][\,3\,] \oplus (r_0 \wedge r_2).
\end{aligned}$$

Observe that the round constants are active only on odd rounds ($r_0 = 1$).

**Table 1.** Truth table for $\phi_5$ (Equation 4.)

| $x_4$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ | $\phi_5$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |

| $x_4$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ | $\phi_5$ |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |

| $x_4$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ | $\phi_5$ |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |

| $x_4$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ | $\phi_5$ |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

**2. Linear transform $\boldsymbol{\lambda^T}$.** Let $\mathbf{s}' = \lambda(\mathbf{s}^T)$ for $0 \le i, j \le 15$:

$$\begin{aligned} s'[\,i\,][\,j\,] = \; & s[\,(j+4) \bmod 16\,][\,i\,] \oplus \\ & s[\,(j+8) \bmod 16\,][\,i\,] \oplus \\ & s[\,(j+12) \bmod 16\,][\,i\,]. \end{aligned} \tag{2}$$

We note that the $\lambda^T$ transform consists of a transpose of the matrix and a bit parity operation. The transpose and bit parity operations are individually involutions but applying their compound operation $\lambda^T$ four times results in the original matrix.

**3. Nonlinear transform $\boldsymbol{\phi}$.** We define $\mathbf{s}' = \phi(\mathbf{s})$ for $0 \le i, j \le 15$ as:

$$\begin{aligned} s'[\,i\,][\,j\,] = \phi_5\big(& s[\,i\,][\,j\,], \\ & s[\,i\,][\,(j-1) \bmod 16\,], \\ & s[\,i\,][\,(j-2) \bmod 16\,], \\ & s[\,i\,][\,(j-3) \bmod 16\,], \\ & s[\,i\,][\,(j-4) \bmod 16\,]\big), \end{aligned} \tag{3}$$

where $\phi_5$ is defined the following Algebraic Normal Form (ANF) polynomial in $\mathbb{Z}_2$:

$$\begin{aligned} \phi_5(x_0, x_1, x_2, x_3, x_4) = {} & x_0x_1x_3x_4 + x_0x_2x_3 + x_0x_1x_4 + x_1x_2x_3 + x_2x_3x_4 + \\ & x_0x_3 + x_1x_3 + x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_3 + x_4. \end{aligned} \tag{4}$$

Selection of $\phi_5$ is discussed in Section 4.1 and Table 1 gives its truth table.

### 3.2 Hashing and Authenticated Encryption

We claim that $\pi$ can be used in all of the following proposed Sponge modes of operation. However, we suggest that unique message nonces or randomizers are always used for AE and MAC modes.

– Authenticated Encryption (AE) with SPONGEWRAP[9].

– Keyed Message Authentication Codes (MACs) [21].
– Collision resistant hashing [6].
– Tree hashing with SAKURA [22].
– Pseudorandom extractors (PRFs and PRNGs) [8].
– BLINKER two-party protocols [23].

For CBEAM described in this paper $b = 256$ and a natural choice for rate is $r = 64$, leaving a capacity of $c = 192$. This is more suitable for low-resource platforms and short messages than KECCAK with its 1600-bit state [1].

For SPONGEWRAP and other modes with frame bits it may be appropriate to have $r = 65$ or 66 in order to not break input byte boundaries. For $2^{38}$ bits of data per key we claim $2^{128}$ security based on Theorems of [10], equivalent to AES-128 and suitable for SECRET data. For $2^{46}$ data we claim $2^{112}$ security, superior to 3DES / TDEA [24].

If even faster speeds are required and unique nonces are available, one may reduce the number of rounds to $\mathsf{mx}^4$ or even $\mathsf{mx}^2$ and use the MONKEYDUPLEX construction of [25]. However, many of the security assurances will break down in this case.

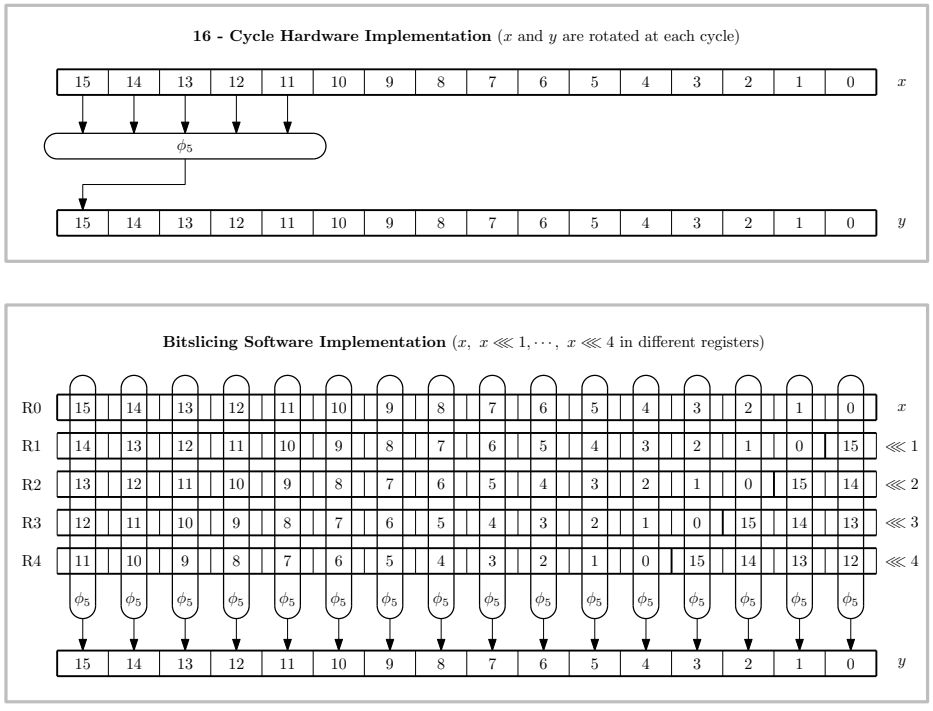


**Fig. 3.** Example of a $16 \times 16$ - bit $\phi$ function based on a $5 \times 1$ - bit Boolean function $\phi_5$. We observe 16 and 1 cycle implementations of the same function. Note that the latter example is equivalent to "bit slicing" software implementation using rotated words.

## 4 Design and Analysis

Ignoring the round constant, the mx transform may be viewed as a transpose of a matrix followed by 16 parallel, independent invocations of a 16 - bit permutation, $(\phi \circ \lambda)_{16}$. We start with the most fundamental observation:

**Theorem 1.** *The mx transform is bijective (reversible).*

*Proof.* The mx transform is bijective as all of its component functions are individually reversible. It is trivial to see that the linear transform $\lambda$ is bijective. Since convolution by a nonlinear Boolean function is generally not reversible, one may compute the $2^{16}$ - entry table of $\phi_{16}$ to verify that it is indeed bijective. □

The choice of round constants was specially crafted to deter rotational [16] and slide [26, 27] attacks.

**Theorem 2.** *Without the round constants the mx transform is shift-invariant both horizontally and vertically. Let* $\mathbf{s}' = \mathsf{mx}(\mathbf{s})$ *and* $\mathbf{t}' = \mathsf{mx}(\mathbf{t})$. *If each element* $s[\,i\,][\,j\,] = t[\,(i+4\Delta_i) \bmod 16\,][\,(j+\Delta_j) \bmod 16\,]$ *for some offsets* $\Delta_i$ *and* $\Delta_j$, *then* $s'[\,i\,][\,j\,] = t'[\,(i+\Delta_j) \bmod 16\,][\,(j+4\Delta_i) \bmod 16\,]$.

*Proof.* The theorem follows form shift-invariant properties of all component functions. Note the exchange of indices $4\Delta_i$ and $\Delta_j$ due to transpose. □

### 4.1 Selection of $\phi_5$

We analyzed all $2^{2^5} = 2^{32}$ five-input Boolean functions, searching for ones that result in invertible 16-bit $\phi$ functions with particularly good properties. Five neighboring bits are used since rotation amounts that would yield better branching (such as the set $\{0, 1, 2, 4, 8\}$) didn't result in any appropriate functions. Single left rotations are used as it is universally available (addition of number to self with carry flow-over to LSB).

There were 260 invertible functions, of which 56 were dependent on all five input bits in nonlinear fashion. Eight of these exhibited optimal differential and linear properties. However there are three independent mirror symmetries (inversion of all input and output bits and the order of input bits) and therefore $2^3 = 8$ equivalent functions. Discounting these symmetries, there is only one optimal function, $\phi_5$ (Equation 4).

Invertibility $\phi_5$ of for other word sizes besides $n = 16$ and the surprising properties of these inverse functions are analyzed in Appendix D.

### 4.2 Differential and Linear Cryptanalysis

Sponge functions can be attacked with DC [13] and LC [14, 15] even though reasonable attack models are radically different from block ciphers.

Because of $\lambda$, changing one bit of the input will spread the difference to at least three bit positions outside the first quadword which can be modified by the attacker. After four of six mx iterations, there is no easily detectable bias regardless of input difference, which we feel is an appropriate security margin. See Table 4 for an illustration of progress of differentials in the state during forward and reverse iterations.

For this analysis we view $(\phi \circ \lambda)_{16}$ row operation as a $16 \times 16$ - bit "S-Box". The highest-probability differential is $\texttt{0CCC} \rightarrow \texttt{8001}$ and its rotational equivalents. The probability of this differential is $\frac{12032}{2^{16}} \approx 0.1836$. From Table 2 in Appendix C we observe that a 1-bit input difference never yields a 1-bit output difference (branch number is greater than 2).

The best linear approximation for $(\phi \circ \lambda)_{16}$ is $\texttt{0888} \rightarrow \texttt{0001}$ and its rotational equivalents, which have a bias of $\frac{16384}{2^{16}} = \frac{1}{4}$. The other best approximations are given in Table 3. Significantly, all single bit approximations have 0 linear bias, as do 2-to-1 and 1-to-2 - bit approximations.

### 4.3 Algebraic Properties

The truth table for $\phi_5$ Boolean function is given in Table 1. From its definition in Equation 4 one easily see that the degree of $\phi_5$ is 4 (and ANF weight 13), and that is also the algebraic degree of $\phi$ state transform $\mathsf{mx}$ (see Equation 3).

The $\mathsf{mx}$ function has been designed to have a significant amount of algebraic "one-wayness" in the sense discussed by Hiltgen [28]. The following somewhat surprising observation can be verified by examining the inverse of $\phi_{16}$:

**Observation 1** *The algebraic degree of the $\phi_{16}^{-1}$ inverse function is 11. The weight (number of nonzero terms) of the ANF polynomial for each output bit of $\phi_{16}^{-1}$ is 13465.*

For a characterization of the Algebraic properties of the inverse of $\phi_n^{-1}$ for $n \neq 16$, we refer to Appendix D, where tables and conjectures are presented.

The algebraic degree of $\mathsf{mx}^n$ is bound by $4^n$. We have verified that the output after six invocations actually has a degree up to 256. If state bits are observed as a function of $s_q[\ 0\ ]$, the number of terms of each degree are distributed in a way that indicates that CBEAM is not vulnerable to $d$-monomial distinguishers [29] or other traditional algebraic attacks.

**Higher-degree inverse indicates high-degree iteration.** Research by Boura and Canteaut on the algebraic degree of iterated permutations seen as multivariate polynomials shows that the degree depends on the algebraic degree of the *inverse* of the permutation which is iterated [5]. This indicates exceptional algebraic security for our proposal.

## 5 Padding and Implementation Notes

A special padding mode of operation, BLINKER [23], is proposed together with CBEAM. This multi-use padding mode allows full encryption protocols to be built from CBEAM. We note that an early version of CBEAM and BLINKER was used in the HAGRAT academic Remote Access Trojan [30], minimizing the size of the encryption component.

CBEAM is highly flexible when it comes to implementation platforms. A standard C implementation may compute four rows in parallel using 64-bit data types whereas specific implementation strategies exist that fully utilize architectures from 16-bit to 256-bit word size. Figure 4 shows how the state fits into the register sets of various CPU architectures.

In hardware implementations, an invocation of the $\mathsf{mx}^n$ transform can take anywhere between 1 and several thousand clock cycles, depending on the number of gates, peak energy and amount of surface area available. Figure 3 shows sixteen- and single clock versions of a $\phi_{16}$ - type convolution.

### 5.1 Hardware Implementations

We have designed and written VHDL for two implementations, dubbed Serial-CBEAM and Block-CBEAM. These have been found to function correctly on a Xilinx Virtex 3E FPGA board with the ISE 14.4 design flow.

**Serial Implementation.** The Serial implementation assumes external 256-bit memory for the state and operates on that state one bit at a time. The implementation sacrifices a lot of clock cycles for reduction of gates and area. The implementation requires only 16 internal register bits in addition to address/clock counters. The implementation with a 1-bit data bus requires 256 read cycles and 256 write cycles for each MX iteration, 3072 clocks in total for full $\pi = \mathsf{mx}^6$. We estimate that the implementation footprint is only about 300 GE without the 256-bit external state memory.

**Block Implementation.** This is a 1 - cycle implementation of the $\mathsf{mx}$ function (with e.g. 256 parallel $\phi_5$ circuits). Depending on target platform and area, timing constraints, it is possible to implement more than one round of $\mathsf{mx}^2$ in a single cycle. Pipelined operation using SAKURA-like [22] hopping hash trees can also be considered with this $\mathsf{mx}$ core.

### 5.2 Implementing CBEAM in Software Without Matrix Transpose

Since transposing a binary matrix is generally slow in software, one would typically want to combine two $\mathsf{mx}$ operations into a double-round with separate "vertical" and "horizontal" parts. We give some generic guidance on how to implement $\mathsf{mx}^2$ in software this way. However, one should examine the reference 16-bit, 64-bit, and 256-bit implementations for architecture-specific optimizations.

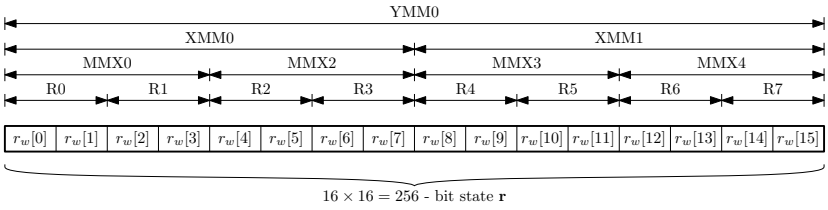


**Fig. 4.** Illustration on how to fit the 256-bit state into a single Haswell+ AVX2 YMM register, two Pentium 3+ SSE XMM registers, four Pentium+ MMX or ARM NEON registers or eight ARM general purpose registers for bit-slicing computation.

**Step 1: Vertical linear transform $\boldsymbol{\lambda}$.** This step is easiest to implement by viewing the state as 64-bit words ("quadwords") $s_q[0..3]$ with $\mathbf{s} = (\, s_q[\,0\,],\ s_q[\,1\,],\ s_q[\,2\,],\ s_q[\,3\,]\,)$.

$$\begin{aligned} t &= s_q[\,0\,] \ \oplus\ s_q[\,1\,] \ \oplus\ s_q[\,2\,] \ \oplus\ s_q[\,3\,] \\ \mathbf{s}' &= (s_q[\,0\,] \ \oplus\ t,\ s_q[\,1\,] \ \oplus\ t,\ s_q[\,2\,] \ \oplus\ t,\ s_q[\,3\,] \ \oplus\ t\,). \end{aligned} \tag{5}$$

**Step 2: Vertical nonlinear transform.** An optimized bit-slicing method for $\phi_5$ is used (See Appendix B). Note that the input words may be stored in registers and contents of registers values in shifted for each new input word. For $0 \le i \le 15$:

$$\begin{aligned} s'_w[\,i\,] = \phi_5\big(&s_w[\,i\,], s_w[\,(i-1) \bmod 16\,], s_w[\,(i-2) \bmod 16\,], \\ &s_w[\,(i-3) \bmod 16\,], s_w[\,(i-4) \bmod 16\,]\big). \end{aligned} \tag{6}$$

**Step 3: Round Constant.** As the round constants are only active at odd rounds, they are in fact always applied between vertical and horizontal rounds in this type of implementation. Written as transposed quadwords, the three nonzero round constants are:

$$\begin{aligned} \mathsf{rc}_q^1 &= \texttt{0x2000040000300009} \\ \mathsf{rc}_q^3 &= \texttt{0x6000050000100008} \\ \mathsf{rc}_q^3 &= \texttt{0xA0000C000070000B} \end{aligned} \tag{7}$$

Constants from Equation 7 are XORed over the first quadword of state at round $i$:

$$s'_q[\,0\,] = s_q[\,0\,] \oplus \mathsf{rc}_q^i. \tag{8}$$

**Step 4: Horizontal linear transform $\boldsymbol{\lambda}$.** This step is relatively slow in this type of implementation. There are many ways to do this; we note that each nibble of $t$ is equivalent to each other. This step is also parallelizable. For $0 \le i \le 15$:

$$\begin{aligned} t &= s_w[\,i\,] \ \oplus\ (s_w[\,i\,] \lll 4) \ \oplus\ (s_w[\,i\,] \lll 8) \ \oplus\ (s_w[\,i\,] \lll 12) \\ s'_w[\,i\,] &= s_w[\,i\,] \ \oplus\ t. \end{aligned} \tag{9}$$

**Step 5: Horizontal nonlinear transform.** Again a bit-slicing implementation of $\phi_5$ (Appendix B) is used, but on rotated values of each word. For $0 \le i \le 15$:

$$s'_w[\,i\,] = \phi_5\big(s_w[\,i\,], s_w[\,i\,] \lll 1, s_w[\,i\,] \lll 2, s_w[\,i\,] \lll 3, s_w[\,i\,] \lll 4\big). \tag{10}$$

### 5.3 Latest Server/Desktop/Laptop Systems: x86-64 with AVX2

The Intel Haswell (Generation 4 Core) and later x86-64 CPUs support 256-bit AVX2 (Advanced Vector Extensions 2) SIMD instructions. The AVX2 platform provides shuffle and vector shift instructions for 16-bit vector sub-units in addition to 256-bit Boolean logic for the nonlinear function $\phi_5$ (Equation 4). We can implement full 256-bit $\phi_5$ with only eight instructions (Appendix B.1). This roughly doubles the overall execution speed when compared to optimized 64-bit gcc versions.

The following speeds were measured on a MacBook Air (Q3/2013) with Intel Core i5 - 4250U CPU @1.30 GHz running Ubuntu Linux 13.04. The internal clock frequency was 1.90 GHz for all tests.

We compare to OpenSSL 1.0.1e AES implementation, which is the de facto standard AES implementation. Generic assembler optimizations were enabled but we disabled the full hardware AES for fairness.

| **Implementation** | **Troughput** | **Cycles/Byte** |
|---|---|---|
| CBEAM-GCC | 58.5 MB/s | 32.5 |
| CBEAM-AVX2 | 117.5 MB/s | 16.1 |
| OpenSSL AES-128 | 106.5 MB/s | 17.8 |
| OpenSSL AES-192 | 86.0 MB/s | 22.1 |
| OpenSSL AES-256 | 71.9 MB/s | 26.4 |

### 5.4 Sensors and Pervasive Devices: MSP430

Texas Instruments MSP430 is a well known family of low-cost and ultra-low power 16-bit SoC microcontrollers, widely used in sensor networks. CBEAM beats the more than dozen MSP430 encryption algorithm implementations reported in [31], often by an order of magnitude.

Our implementation of $\pi$ is able to execute entirely on 12 general-purpose registers without having to resort to stack (except the top value) and therefore the running RAM requirement is equivalent to the state size, 32 bytes. The $\phi_5$ function was realized with nine logic instructions (Appendix B.2). Unfortunately the target only has 1-bit shifts and no multi-bit rotation instructions, which results in a bottleneck for "horizontal" $\lambda$.

The cipher is as fast as the very fastest AES implementations on this platform but has significantly smaller implementation footprint. The following numbers are only for cores, modes of operation not included. The IAIK [32] implementation is commercial and written in hand-optimized assembly. The Texas Instruments [33] implementation is recommended by the SoC vendor.

| **Code** | **Flash** | **Ram** | **Encryption** | **Decryption** | **Cycles/Byte** |
|---|---|---|---|---|---|
| CBEAM | 386 | 32 | 4369 | 4404 | 550.5 |
| AES-128 [32] | 2536 | ? | 5432 | 8802 | 550.1 |
| AES-128 [33] | 2423 | 80 | 6600 | 8400 | 525.0 |
| AES-256 [32] | 2830 | ? | 7552 | 12258 | 766.1 |

## 6 Conclusions

We propose the use of novel rotation-invariant $\phi$ functions in cryptographic primitives such as hashes and authenticated encryption. These functions have fascinating and attractive properties such as "feeble one-wayness"; the Boolean complexity of inversion appears to be much higher than the Boolean complexity of computing the permutation in forward direction. We have experimentally verified that the polynomial degree for the inverse of a $\phi_5$ function grows linearly as the number of input bits grows, while it remains constant in forward direction. Hence the function and its inverse are in different complexity classes (linear vs. polynomial or super-polynomial).

In a Sponge construction a large and efficient cryptographic permutation is required. The permutation needs to be computed only in one direction during normal operation. It has been shown that complexity of inversion makes collision search and other attacks more difficult. Here an asymmetric $\phi$ function is an ideal choice. This motivates us to propose a new 256-bit Sponge function, CBEAM, which can be used for cryptographic hashing, authenticated encryption, and other purposes.

In addition to the theoretical side, the main attractive feature of CBEAM is its extreme implementation flexibility; a single word encryption operation may require anywhere between 1 to thousands of cycles, depending on the area and energy requirements of the implementation. We also demonstrate that it is approximately as fast as AES on both high-end CPUs and low-end MCUs, while having a significantly smaller implementation footprint.

**Acknowledgements.** The author wishes to thank Kudelski Security, University of Haifa, and Nanyang Technological University for supporting his work. Program Committee members of CT-RSA 2014 provided invaluable suggestions for improving the quality of this paper.

## References

1. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference, version 3.0. NIST SHA3 Submission Document (January 2011)
2. NIST: NIST selects winner of secure hash algorithm (SHA-3) competition. NIST Tech Beat Newsletter (2 October 2012)
3. Daemen, J.: Cipher and Hash Function Design Strategies based on linear and differential cryptanalysis. PhD thesis, K.U. Leuven (March 1995)
4. Dinur, I., Dunkelman, O., Shamir, A.: New attacks on Keccak-224 and Keccak-256. In Canteaut, A., ed.: FSE 2012. Volume 7549 of LNCS., Springer (2012) 442–461
5. Boura, C., Canteaut, A.: On the influence of the algebraic degree of $F^{-1}$ on the algebraic degree of $G \circ F$. IEEE Transactions on Information Theory **59**(1) (January 2013)
6. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge functions. In: Ecrypt Hash Workshop 2007. (May 2007)
7. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the indifferentiability of the sponge construction. In Smart, N.P., ed.: EUROCRYPT 2008. Volume 4965 of LNCS., Springer (2008) 181–197

8. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge-based pseudo-random number generators. In Mangard, S., Standaert, F.X., eds.: CHES 2010. Volume 6225 of LNCS., Springer (2010) 33–47
9. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In Miri, A., Vaudenay, S., eds.: SAC 2011. Volume 7118 of LNCS., Springer (2011) 320–337
10. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Cryptographic sponge functions, version 0.1. http://sponge.noekeon.org/, STMicroelectronics and NXP Semiconductors (January 2011)
11. Saarinen, M.J.O.: Related-key attacks against full Hummingbird-2. In: FSE 2013: 20th International Workshop on Fast Software Encryption. 11-13 March 2013, Singapore, Singapore. (2013) To Appear.
12. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR ePrint 2013/404, http://eprint.iacr.org/2013/404 (June 2013)
13. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer (1993)
14. Matsui, M.: Linear cryptoanalysis method for DES cipher. In Helleseth, T., ed.: EUROCRYPT '93. Volume 765 of LNCS., Springer (1994) 386–397
15. Matsui, M.: The first experimental cryptanalysis of the data encryption standard. In Desmedt, Y., ed.: CRYPTO '94. Volume 839 of LNCS., Springer (1994) 1–11
16. Khovratovich, D., Nikolić, I.: Rotational cryptanalysis of ARX. In Hong, S., Iwata, T., eds.: FSE 2010. Volume 6147 of LNCS., Springer (2010) 333–346
17. Biham, E.: A fast new DES implementation in software. In Biham, E., ed.: FSE 1997. Volume 1267 of LNCS., Springer (1997) 260–272
18. Bernstein, D.J.: Cache-timing attacks on AES. Technical report, University of Chigaco (2005)
19. Acıiçmez, O., Schindler, W., Ç. K. Koç: Cache based remote timing attack on the AES. In Abe, M., ed.: CT-RSA 2007. Volume 4377 of LNCS., Springer (2007) 271–286
20. Weiß, M., Heinz, B., Stumpf, F.: A cache timing attack on AES in virtualization environments. In Keromytis, A., ed.: FC 2012. Volume 7397 of LNCS., Springer (2013) 314–328
21. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the security of the keyed sponge construction. In: SKEW 2011 Symmetric Key Encryption Workshop. (February 2011)
22. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sakura: a flexible coding for tree hashing. IACR ePrint 2013/213, http://eprint.iacr.org/2013/213 (April 2013)
23. Saarinen, M.J.O.: Beyond modes: Building a secure record protocol from a cryptographic sponge permutation. In: CT-RSA 2014: Cryptographers' Track, RSA Conference USA, 25–28 February 2014, San Francisco, USA, Springer (2014) To Appear.
24. NIST: Recommendation for the Triple Data Encryption Algorithm (TDEA) block cipher, revision 1. NIST Special Publication 800-67 (January 2012)
25. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Permutation-based encryption, authentication and authenticated encryption. In: DIAC 2012. (2012) http://keccak.noekeon.org/KeccakDIAC2012.pdf.
26. Biryukov, A., Wagner, D.: Slide attacks. In Knudsen, L.R., ed.: FSE 1999. Volume 1636 of LNCS., Springer (1999) 245–259
27. Biryukov, A., Wagner, D.: Advanced slide attacks. In Preneel, B., ed.: EUROCRYPT 2000. Volume 1807 of LNCS., Springer (2000) 589–606
28. Hiltgen, A.P.: Towards a better understanding of one-wayness: Facing linear permutations. In Nyberg, K., ed.: EUROCRYPT '98. Volume 1403 of LNCS., Springer (1998) 319–333

29. Saarinen, M.J.O.: Chosen-IV statistical attacks against eSTREAM ciphers. In: Proc. SECRYPT 2006, International Conference on Security and Cryptography, Setubal, Portugal, August 7-10, 2006. (2006)
30. Saarinen, M.J.O.: Developing a grey hat C2 and RAT for APT security training and assessment. In: GreHack 2013 Hacking Conference, 15 November 2013, Grenoble, France. (2013) To Appear.
31. Cazorla, M., Marquet, K., Minier, M.: Survey and benchmark of lightweight block ciphers for wireless sensor networks. In: SECRYPT 2013. (May 2013) `http://eprint.iacr.org/2013/295`.
32. IAIK: AES for Texas Instruments MSP430 microcontrollers. Technical report, IAIK SIC T. U. Graz `http://jce.iaik.tugraz.at/sic/Products/Crypto_Software_for_Microcontrollers`.
33. TI: AES128 - A C implementation for encryption and decryption. Technical Report SLAA397A, Texas Instruments (July 2009) `http://www.ti.com/lit/an/slaa397a/slaa397a.pdf`.

## A Trace of Execution for CBEAM

A trace (test vector) of six rounds of computation for the $\pi = \mathsf{mx}^6$ function:

$$\begin{aligned}
\mathbf{b} &= (\ \texttt{0123}, \texttt{1234}, \texttt{2345}, \texttt{3456}, \texttt{4567}, \texttt{5789}, \texttt{6789}, \texttt{789A},\\
&\qquad \texttt{89AB}, \texttt{9ABC}, \texttt{ABCD}, \texttt{BCDE}, \texttt{CDEF}, \texttt{DEF0}, \texttt{EF01}, \texttt{F012}\ )\\
\mathsf{mx}(\mathbf{b}) &= (\ \texttt{88A8}, \texttt{3333}, \texttt{BDBD}, \texttt{BFC1}, \texttt{DD5D}, \texttt{B87B}, \texttt{BF7D}, \texttt{A3B5},\\
&\qquad \texttt{88A8}, \texttt{CCCC}, \texttt{F6F6}, \texttt{FF06}, \texttt{5555}, \texttt{9999}, \texttt{EDED}, \texttt{FE0D}\ )\\
\mathsf{mx}^2(\mathbf{b}) &= (\ \texttt{6F0D}, \texttt{E713}, \texttt{4B47}, \texttt{B151}, \texttt{25BD}, \texttt{929F}, \texttt{2540}, \texttt{7780},\\
&\qquad \texttt{4985}, \texttt{511D}, \texttt{818C}, \texttt{A135}, \texttt{8426}, \texttt{9911}, \texttt{FB65}, \texttt{3991}\ )\\
\mathsf{mx}^3(\mathbf{b}) &= (\ \texttt{E50C}, \texttt{EAE4}, \texttt{07F3}, \texttt{B08A}, \texttt{6476}, \texttt{2138}, \texttt{D90D}, \texttt{F629},\\
&\qquad \texttt{3919}, \texttt{3071}, \texttt{1E59}, \texttt{1458}, \texttt{DEEC}, \texttt{15F3}, \texttt{96DF}, \texttt{1FB2}\ )\\
\mathsf{mx}^4(\mathbf{b}) &= (\ \texttt{8922}, \texttt{B751}, \texttt{6648}, \texttt{0EED}, \texttt{C285}, \texttt{89E5}, \texttt{2DFC}, \texttt{DBBF},\\
&\qquad \texttt{4310}, \texttt{77FA}, \texttt{3494}, \texttt{7F13}, \texttt{47D9}, \texttt{6DD3}, \texttt{1E59}, \texttt{E502}\ )\\
\mathsf{mx}^5(\mathbf{b}) &= (\ \texttt{2CA0}, \texttt{67B3}, \texttt{4F96}, \texttt{0A46}, \texttt{B209}, \texttt{AC7E}, \texttt{5C64}, \texttt{A125},\\
&\qquad \texttt{CF7C}, \texttt{B46F}, \texttt{EB8A}, \texttt{FAED}, \texttt{1130}, \texttt{934D}, \texttt{CC02}, \texttt{0D67}\ )\\
\mathsf{mx}^6(\mathbf{b}) = \pi(\mathbf{b}) &= (\ \texttt{5432}, \texttt{281E}, \texttt{B184}, \texttt{9481}, \texttt{AAF0}, \texttt{C9BE}, \texttt{A028}, \texttt{4C79},\\
&\qquad \texttt{4B69}, \texttt{53BF}, \texttt{53C0}, \texttt{CFE8}, \texttt{8839}, \texttt{9D2A}, \texttt{89E3}, \texttt{1300}\ )
\end{aligned}$$

## B Bit-Slicing for $\phi_5$

The ANSI C reference implementation `cbref/mx6-gcc.c` implements $\phi_5$ as a macro as follows:

```
#define CBEAM_PHI5(x0, x1, x2, x3, x4) \
    (~(x0 & ((~x3 & x4) ^ (~x2 & x3))) & \
    (x1 | (~x2 & x3))) ^ (~x2 & (~x3 & x4))
```

Here we put our trust to compiler for common subexpression elimination of `(~x3 & x4)` and `(~x2 & x3)`. One can assign these to temporary variables if necessary. We have exhaustively verified that $\phi_5$ cannot be implemented with less than eight logical instructions (ANDN `(~x & y)` is a single op).

### B.1 AVX 2

Here is a code snippet written in AVX2 C intrinsics for implementing the $\phi_5$ function with 8 logical instructions on 256-bit registers:

```
// t0 = Phi5(x0,x1,x2,x3,x4)
t0 = _mm256_andnot_si256(x3, x4);
t1 = _mm256_andnot_si256(x2, x3);
t2 = _mm256_andnot_si256(x2, t0);
t3 = _mm256_or_si256(x1, t1);
t0 = _mm256_xor_si256(t0, t1);
t1 = _mm256_and_si256(x0, t0);
t0 = _mm256_andnot_si256(t1, t3);
t0 = _mm256_xor_si256(t0, t2);
```

Please see the reference implementation file `cbref/mx6-avx2.c` for tricks on how to implement $\lambda$ and various shifts efficiently on this platform.

### B.2 MSP430

TI MSP430 has only two-operand machine instructions and hence the code is slightly longer with 9 instructions on 16-bit registers:

```
/* r14 = Phi5(r15,r14,r13,r12,r11) */
bic     r12, r11
inv     r13
and     r13, r12
and     r11, r13
xor     r12, r11
and     r11, r15
bis     r12, r14
bic     r15, r14
xor     r13, r14
```

The MSP430 reference implementation `cbeam430/mx430.s` can compute $\pi = \mathsf{mx}^6$ without utilizing stack (except the top value, which is basically free).

# C Auxiliary Tables

**Table 2.** Probabilities (%) of best differentials for $(\lambda \circ \phi)_{16}$ with specific input weight (rows) and output weight (columns). The best overall differential and the best differential with output weight 1 are emphasized.

| Wt | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1.07 | .513 | .635 | .562 | .385 | .330 | .140 | .137 | .064 | .021 | .003 | 0 | 0 | 0 |
| 2 | 0 | 4.30 | 2.15 | 2.54 | 2.15 | 1.37 | .592 | .443 | .284 | .256 | .116 | .098 | .037 | .027 | .006 | .006 |
| 3 | **17.2** | 5.47 | 5.47 | 3.91 | 1.78 | .922 | .787 | .476 | .330 | .195 | .177 | .119 | .079 | .052 | .027 | .003 |
| 4 | .009 | 1.46 | 3.37 | 5.15 | 1.95 | 1.32 | .903 | .439 | .305 | .375 | .232 | .159 | .101 | .064 | .049 | .021 |
| 5 | .684 | 2.93 | 6.74 | 2.49 | 2.20 | 1.76 | .885 | .635 | .446 | .363 | .266 | .192 | .140 | .085 | .064 | .021 |
| 6 | 7.03 | **18.4** | 5.47 | 3.91 | 2.34 | 1.37 | .894 | .702 | .412 | .354 | .214 | .168 | .131 | .128 | .052 | .018 |
| 7 | .928 | 2.00 | 4.17 | 2.12 | 3.09 | 1.64 | 1.14 | .671 | .470 | .299 | .247 | .223 | .165 | .101 | .040 | .024 |
| 8 | 2.93 | 3.22 | 3.22 | 4.15 | 3.12 | 1.95 | 1.20 | .732 | .522 | .360 | .220 | .256 | .140 | .070 | .049 | .034 |
| 9 | 8.20 | 4.00 | 11.1 | 4.59 | 3.52 | 1.95 | 1.28 | .885 | .525 | .366 | .253 | .171 | .134 | .101 | .067 | .024 |
| 10 | .598 | 1.39 | 1.66 | 2.73 | 1.46 | 2.27 | 1.44 | .781 | .586 | .323 | .220 | .208 | .131 | .153 | .043 | .021 |
| 11 | .964 | 2.44 | 3.27 | 2.05 | 3.96 | 2.22 | 1.27 | .879 | .403 | .232 | .266 | .192 | .165 | .092 | .037 | .027 |
| 12 | .781 | 5.57 | 2.83 | 6.74 | 2.34 | 1.86 | .696 | .439 | .290 | .296 | .198 | .171 | .128 | .058 | .040 | .031 |
| 13 | 0 | .122 | .159 | .323 | .247 | .269 | .327 | .317 | .272 | .214 | .223 | .119 | .082 | .058 | .031 | .009 |
| 14 | 0 | .018 | .073 | .150 | .177 | .250 | .269 | .424 | .235 | .275 | .140 | .104 | .061 | .052 | .024 | .015 |
| 15 | 0 | 0 | .003 | .006 | .079 | .064 | .122 | .058 | .076 | .064 | .055 | .037 | .043 | .034 | .021 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | .006 | .006 | .049 | .024 | .055 | .031 | .058 | .015 | .021 | 0 | .012 |

**Table 3.** Absolute biases (%) of best linear approximations for $(\lambda \circ \phi)_{16}$ with specific input mask weight (rows) and output weight (columns).

| Wt | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 6.25 | 3.71 | 4.69 | 3.12 | 2.73 | 1.56 | 1.17 | .586 | .439 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 12.5 | 7.03 | 9.38 | 6.25 | 4.88 | 3.71 | 2.93 | 3.32 | 1.86 | 1.27 | 1.27 | .684 | .391 | 0 | 0 |
| 3 | **25.0** | 12.5 | 9.38 | 6.25 | 5.08 | 4.49 | 4.59 | 3.71 | 2.73 | 1.95 | 1.66 | 1.42 | 1.37 | .830 | .684 | 0 |
| 4 | 0 | 7.03 | 9.38 | 7.81 | 7.81 | 7.03 | 3.91 | 3.81 | 3.03 | 2.88 | 4.39 | 6.15 | 3.37 | 2.73 | 1.27 | 2.15 |
| 5 | 6.25 | 12.5 | 9.38 | 8.59 | 9.38 | 5.86 | 5.08 | 3.91 | 5.27 | 6.05 | 6.84 | 3.47 | 2.64 | 2.49 | 1.95 | 1.07 |
| 6 | 18.8 | 18.8 | 15.6 | 10.9 | 7.03 | 5.47 | 5.27 | 6.45 | 6.25 | 8.01 | 4.83 | 3.32 | 2.15 | 1.95 | 1.46 | 1.17 |
| 7 | 0 | 7.81 | 10.9 | 12.5 | 7.81 | 6.64 | 7.42 | 8.40 | 8.40 | 4.59 | 3.91 | 4.15 | 4.74 | 2.78 | 3.42 | 1.27 |
| 8 | 6.25 | 15.6 | 14.1 | 9.38 | 10.2 | 8.59 | 8.59 | 9.77 | 6.45 | 4.79 | 4.83 | 3.96 | 3.76 | 3.76 | 2.25 | .977 |
| 9 | 18.8 | 18.8 | 15.6 | 10.9 | 8.59 | 10.5 | 10.9 | 6.25 | 4.69 | 3.96 | 4.39 | 3.56 | 2.88 | 2.15 | 1.95 | .879 |
| 10 | 0 | 7.03 | 7.81 | 9.38 | 9.38 | 14.1 | 8.59 | 5.08 | 4.49 | 4.54 | 3.61 | 3.96 | 3.76 | 4.98 | 2.83 | 2.25 |
| 11 | 6.25 | 7.81 | 7.81 | 10.9 | 15.6 | 7.81 | 6.25 | 4.59 | 3.61 | 3.32 | 4.20 | 4.88 | 3.08 | 2.98 | 1.86 | 1.17 |
| 12 | 6.25 | 9.38 | 14.1 | 17.2 | 9.38 | 6.25 | 3.91 | 3.32 | 3.37 | 3.32 | 3.66 | 4.20 | 2.98 | 2.59 | 1.46 | 1.56 |
| 13 | 0 | 0 | 1.95 | 2.93 | 2.93 | 2.93 | 3.71 | 2.88 | 3.27 | 4.20 | 3.52 | 2.78 | 3.27 | 3.52 | 5.08 | .586 |
| 14 | 0 | 0 | .391 | .684 | 1.90 | 2.05 | 2.44 | 2.29 | 2.93 | 2.98 | 2.59 | 2.98 | 2.69 | 2.78 | 1.46 | .977 |
| 15 | 0 | 0 | 0 | .684 | .635 | 1.22 | 1.66 | 1.76 | 1.81 | 1.76 | 2.05 | 1.86 | 2.29 | 1.17 | .684 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .488 | .537 | .684 | .977 | 1.37 | 1.32 | 2.78 | 1.17 | 9.38 |

**Table 4.** Progression of differentials in consecutive invocations of $\mathsf{mx}$. Here the zeroth bit has been flipped; $\Delta = 0^{255} \,||\, 1$. We observe that the full state is affected and there is no detectable bias after $\mathsf{mx}^4$. The $\pi$ transform has six rounds by default.

$\mathsf{mx}(x) \oplus \mathsf{mx}(x \oplus \Delta)$

| # | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 38 | 50 | 63 | 50 | 37 | 50 | 62 | 50 | 37 | 50 | 63 | 38 | 00 | 00 | 00 | **25** |
| 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 02 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 03 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 04 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 05 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 06 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 07 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 08 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 09 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 10 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 11 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 12 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 13 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 14 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 15 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

$\mathsf{mx}^2(x) \oplus \mathsf{mx}^2(x \oplus \Delta)$

| # | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 09 | 12 | 16 | 12 | 09 | 12 | 16 | 12 | 09 | 12 | 16 | 09 | 00 | 00 | 00 | 06 |
| 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 02 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 03 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 04 | 14 | 19 | 23 | 19 | 14 | 19 | 23 | 19 | 14 | 19 | 23 | 14 | 00 | 00 | 00 | 09 |
| 05 | 23 | 31 | 39 | 31 | 23 | 31 | 39 | 31 | 23 | 31 | 39 | 23 | 00 | 00 | 00 | 16 |
| 06 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 19 | 00 | 00 | 00 | 12 |
| 07 | 14 | 19 | 23 | 19 | 14 | 19 | 23 | 19 | 14 | 19 | 23 | 14 | 00 | 00 | 00 | 09 |
| 08 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 19 | 00 | 00 | 00 | 13 |
| 09 | 23 | 31 | 39 | 31 | 23 | 31 | 39 | 31 | 23 | 31 | 39 | 23 | 00 | 00 | 00 | 16 |
| 10 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 19 | 00 | 00 | 00 | 13 |
| 11 | 14 | 19 | 23 | 19 | 14 | 19 | 23 | 19 | 14 | 19 | 23 | 14 | 00 | 00 | 00 | 09 |
| 12 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 19 | 00 | 00 | 00 | 12 |
| 13 | 23 | 31 | 39 | 31 | 23 | 31 | 39 | 31 | 23 | 31 | 39 | 23 | 00 | 00 | 00 | 16 |
| 14 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 25 | 19 | 25 | 31 | 19 | 00 | 00 | 00 | 12 |
| 15 | 14 | 19 | 24 | 19 | 14 | 19 | 23 | 19 | 14 | 19 | 23 | 14 | 00 | 00 | 00 | 09 |

$\mathsf{mx}^3(x) \oplus \mathsf{mx}^3(x \oplus \Delta)$

| # | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 33 | 34 | 33 | 32 | 33 | 34 | 33 | 32 | 33 | 36 | 36 | 37 | 38 | 39 | 36 | 32 |
| 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 02 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 03 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 04 | 40 | 42 | 41 | 39 | 40 | 42 | 41 | 39 | 40 | 43 | 43 | 42 | 44 | 45 | 43 | 39 |
| 05 | 47 | 48 | 47 | 45 | 47 | 49 | 47 | 45 | 46 | 48 | 48 | 47 | 49 | 49 | 48 | 45 |
| 06 | 44 | 46 | 45 | 43 | 44 | 46 | 45 | 43 | 44 | 46 | 46 | 46 | 47 | 48 | 46 | 43 |
| 07 | 40 | 42 | 41 | 39 | 40 | 42 | 41 | 39 | 40 | 43 | 43 | 42 | 44 | 45 | 43 | 39 |
| 08 | 44 | 46 | 45 | 43 | 44 | 46 | 45 | 43 | 44 | 46 | 46 | 46 | 47 | 48 | 46 | 43 |
| 09 | 46 | 48 | 47 | 45 | 47 | 49 | 47 | 45 | 46 | 48 | 48 | 47 | 48 | 49 | 48 | 45 |
| 10 | 44 | 46 | 45 | 43 | 44 | 46 | 45 | 42 | 44 | 46 | 46 | 46 | 47 | 48 | 46 | 43 |
| 11 | 40 | 42 | 41 | 39 | 40 | 42 | 41 | 39 | 40 | 42 | 43 | 42 | 44 | 45 | 43 | 39 |
| 12 | 44 | 46 | 45 | 43 | 44 | 46 | 45 | 43 | 44 | 46 | 46 | 46 | 47 | 48 | 46 | 43 |
| 13 | 46 | 48 | 47 | 45 | 47 | 49 | 47 | 45 | 46 | 48 | 48 | 47 | 48 | 49 | 48 | 45 |
| 14 | 44 | 46 | 45 | 43 | 44 | 46 | 45 | 43 | 44 | 46 | 46 | 46 | 47 | 48 | 46 | 43 |
| 15 | 40 | 42 | 41 | 39 | 40 | 42 | 41 | 39 | 40 | 43 | 43 | 42 | 44 | 45 | 43 | 39 |

$\mathsf{mx}^4(x) \oplus \mathsf{mx}^4(x \oplus \Delta)$

| # | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 50 | 50 | 50 | 49 | 49 |
| 01 | 49 | 49 | 50 | 49 | 49 | 50 | 50 | 49 | 49 | 50 | 50 | 50 | 50 | 50 | 50 | 49 |
| 02 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| 03 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 49 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| 04 | 49 | 50 | 50 | 49 | 49 | 50 | 50 | 49 | 49 | 50 | 50 | 50 | 50 | 50 | 50 | 49 |
| 05 | 49 | 49 | 50 | 49 | 49 | 49 | 50 | 49 | 49 | 49 | 50 | 50 | 50 | 50 | 50 | 49 |
| 06 | 49 | 50 | 50 | 49 | 50 | 50 | 50 | 49 | 49 | 50 | 50 | 50 | 50 | 50 | 50 | 49 |
| 07 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 50 | 50 | 50 | 50 | 50 | 49 |
| 08 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 50 | 50 | 50 | 49 |
| 09 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 50 | 50 | 50 | 50 | 49 |
| 10 | 50 | 50 | 50 | 49 | 49 | 50 | 50 | 49 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 49 |
| 11 | 49 | 49 | 49 | 49 | 49 | 49 | 50 | 49 | 49 | 49 | 50 | 50 | 50 | 50 | 50 | 49 |
| 12 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 50 | 50 | 50 | 49 |
| 13 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 50 | 50 | 50 | 50 | 49 |
| 14 | 50 | 50 | 50 | 49 | 50 | 50 | 50 | 49 | 49 | 50 | 50 | 50 | 50 | 50 | 50 | 49 |
| 15 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 50 | 50 | 50 | 50 | 50 | 49 |

Progression of differentials in consecutive invocations of inverse function $\mathsf{mx}^{-1}$. Here again the zeroth bit is flipped. After third round there is no longer any detectable bias.

$\mathsf{mx}^{-1}(x) \oplus \mathsf{mx}^{-1}(x \oplus \Delta)$

| # | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | **50** |
| 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 02 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 03 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 04 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 05 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 06 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 07 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 08 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 09 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 10 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 11 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 12 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 13 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 14 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |
| 15 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 50 | 50 | 50 | 50 |

$\mathsf{mx}^{-2}(x) \oplus \mathsf{mx}^{-2}(x \oplus \Delta)$

| # | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 20 | 25 | 19 | 25 | 34 | 34 | 24 | 25 | 24 | 25 | 25 | 25 | 25 | 25 | 25 | 25 |
| 01 | 24 | 21 | 22 | 20 | 20 | 20 | 20 | 21 | 20 | 21 | 21 | 20 | 21 | 21 | 21 | 21 |
| 02 | 23 | 21 | 21 | 21 | 21 | 21 | 20 | 21 | 20 | 21 | 21 | 21 | 21 | 21 | 21 | 22 |
| 03 | 39 | 39 | 39 | 39 | 40 | 40 | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 39 |
| 04 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| 05 | 49 | 50 | 50 | 50 | 49 | 49 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| 06 | 46 | 47 | 47 | 48 | 49 | 49 | 48 | 47 | 48 | 47 | 47 | 48 | 47 | 47 | 47 | 47 |
| 07 | 49 | 48 | 47 | 47 | 47 | 47 | 46 | 47 | 46 | 48 | 48 | 47 | 48 | 48 | 48 | 49 |
| 08 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| 09 | 49 | 50 | 50 | 50 | 47 | 47 | 49 | 50 | 49 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| 10 | 44 | 46 | 44 | 47 | 51 | 51 | 47 | 46 | 47 | 46 | 46 | 47 | 46 | 46 | 46 | 46 |
| 11 | 50 | 47 | 48 | 47 | 47 | 47 | 46 | 47 | 46 | 47 | 47 | 47 | 47 | 47 | 48 | 48 |
| 12 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| 13 | 49 | 49 | 49 | 49 | 47 | 47 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 50 |
| 14 | 41 | 45 | 42 | 45 | 50 | 50 | 46 | 45 | 46 | 45 | 45 | 45 | 45 | 45 | 45 | 45 |
| 15 | 47 | 38 | 40 | 36 | 38 | 38 | 35 | 37 | 34 | 38 | 37 | 36 | 39 | 38 | 39 | 40 |

## D Tables and Conjectures on Algebraic Properties of $\phi_n^{-1}$

The $\phi_5$ (Equations 4 and 10) Boolean mapping also defines reversible $n \times n$ - bit shift-invariant functions for other $n$ apart from $n = 16$. Each forward function has degree 4. The characteristics of the Algebraic Normal Form of inverse functions up to $n = 32$ are given below. Each column contains the number of monomials of given degree; the last column has the number of nonzero terms for all degrees.

*Conjecture 1.* The inverse of $\phi_n$ is defined for each $n \geq 5$ with $n \neq 0 \pmod 3$ and $\deg \phi_n^{-1} = \left\lceil \frac{2}{3} n \right\rceil$.

*Conjecture 2.* Computation of $\phi_n^{-1}$ has at least polynomial complexity (with degree $\geq 2$).

The computation of $\phi_n$ has linear complexity $O(n)$ but the complexity of $\phi_n^{-1}$ is at least $O(n^2)$ since the number of input bits grows with $n$ as per observation in Conjecture 1. Super-polynomial complexity has not been ruled out as we do not know a polynomial time algorithm for $\phi_n^{-1}$. Based on current evidence we are reluctant to believe in exponential complexity, however.

| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | Poly n.z.t. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **6** | | | | | | | | | | | | | | | | | | | | | | | *Nonsurjective.* |
| **7** | 4 | 11 | 17 | 15 | 6 | | | | | | | | | | | | | | | | | | 53 |
| **8** | 3 | 9 | 13 | 13 | 9 | 2 | | | | | | | | | | | | | | | | | 49 |
| **9** | | | | | | | | | | | | | | | | | | | | | | | *Nonsurjective.* |
| **10** | 5 | 21 | 55 | 91 | 95 | 56 | 14 | | | | | | | | | | | | | | | | 337 |
| **11** | 4 | 18 | 45 | 75 | 88 | 69 | 28 | 4 | | | | | | | | | | | | | | | 331 |
| **12** | | | | | | | | | | | | | | | | | | | | | | | *Nonsurjective.* |
| **13** | 6 | 34 | 125 | 303 | 502 | 565 | 408 | 168 | 30 | | | | | | | | | | | | | | 2141 |
| **14** | 5 | 30 | 106 | 253 | 433 | 543 | 471 | 252 | 72 | 8 | | | | | | | | | | | | | 2173 |
| **15** | | | | | | | | | | | | | | | | | | | | | | | *Nonsurjective.* |
| **16** | 7 | 50 | 236 | 753 | 1705 | 2797 | 3293 | 2686 | 1430 | 446 | 62 | | | | | | | | | | | | 13465 |
| **17** | 6 | 45 | 205 | 640 | 1456 | 2504 | 3236 | 3017 | 1912 | 766 | 172 | 16 | | | | | | | | | | | 13975 |
| **18** | | | | | | | | | | | | | | | | | | | | | | | *Nonsurjective.* |
| **19** | 8 | 69 | 397 | 1570 | 4506 | 9678 | 15684 | 19001 | 16832 | 10532 | 4402 | 1104 | 126 | | | | | | | | | | 83909 |
| **20** | 7 | 63 | 351 | 1356 | 3866 | 8472 | 14450 | 18965 | 18645 | 13266 | 6554 | 2114 | 396 | 32 | | | | | | | | | 88537 |
| **21** | | | | | | | | | | | | | | | | | | | | | | | *Nonsurjective.* |
| **22** | 9 | 91 | 617 | 2910 | 10112 | 26816 | 55170 | 88281 | 109077 | 102570 | 71834 | 36250 | 12464 | 2618 | 254 | | | | | | | | 519073 |
| **23** | 8 | 84 | 553 | 2548 | 8750 | 23352 | 49428 | 83181 | 110136 | 112723 | 87302 | 49868 | 20260 | 5510 | 892 | 64 | | | | | | | 554659 |
| **24** | | | | | | | | | | | | | | | | | | | | | | | *Nonsurjective.* |
| **25** | 10 | 116 | 905 | 4956 | 20216 | 63770 | 158824 | 315095 | 498190 | 624397 | 614364 | 467824 | 269904 | 114084 | 33356 | 6036 | 510 | | | | | | 3192557 |
| **26** | 9 | 108 | 820 | 4390 | 17654 | 55622 | 140638 | 288151 | 477827 | 636095 | 671875 | 555352 | 353222 | 168890 | 58546 | 13834 | 1980 | 128 | | | | | 3445141 |
| **27** | | | | | | | | | | | | | | | | | | | | | | | *Nonsurjective.* |
| **28** | 11 | 144 | 1270 | 7918 | 37078 | 135562 | 396082 | 936523 | 1801051 | 2816653 | 3568633 | 3638674 | 2956588 | 1887016 | 925480 | 336844 | 85766 | 13646 | 1022 | | | | 19545961 |
| **29** | 10 | 135 | 1161 | 7083 | 32664 | 118764 | 349392 | 843177 | 1676448 | 2740338 | 3661044 | 3966297 | 3452310 | 2386518 | 1289610 | 532002 | 161404 | 33822 | 4348 | 256 | | | 21256783 |
| **30** | | | | | | | | | | | | | | | | | | | | | | | *Nonsurjective.* |
| **31** | 12 | 175 | 1721 | 12033 | 63606 | 264432 | 886320 | 2431089 | 5500476 | 10297548 | 15947808 | 20378433 | 21385950 | 18304116 | 12646968 | 6947652 | 2965474 | 948556 | 214062 | 30408 | 2046 | | 119228885 |
| **32** | 11 | 165 | 1585 | 10855 | 56487 | 232938 | 781992 | 2171889 | 5029839 | 9731040 | 15696456 | 21023385 | 23257191 | 21114276 | 15602790 | 9279726 | 4369660 | 1589364 | 429714 | 81042 | 9468 | 512 | 130470385 |