CBEAM: Efficient Authenticated Encryption from Feebly One-Way ϕ Functions

Author: Markku-Juhani O. Saarinen

Presented by: Jean-Philippe Aumasson

CT-RSA '14, San Francisco, USA
26 February 2014

Sponge Functions

Are based on some keyless cryptographic permutation π.

Proposed and proved by G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche for:

Sponge Functions

Are based on some keyless cryptographic permutation π.

Proposed and proved by G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche for:

- Collision resistant hash algorithms [eCRYPT Hash Workshop 2007], like Keccak [SHA3 Winner 2011].
- Pseudorandom extractors (PRFs and PRNGs) [CHES 2010].
- **Authenticated Encryption** (AE,AEAD) [SAC 2011].
- Keyed Message Authentication Codes (MACs) [SKEW 2011].
- Tree hashing with Sakura [IACR ePrint 2013].

Sponge Functions

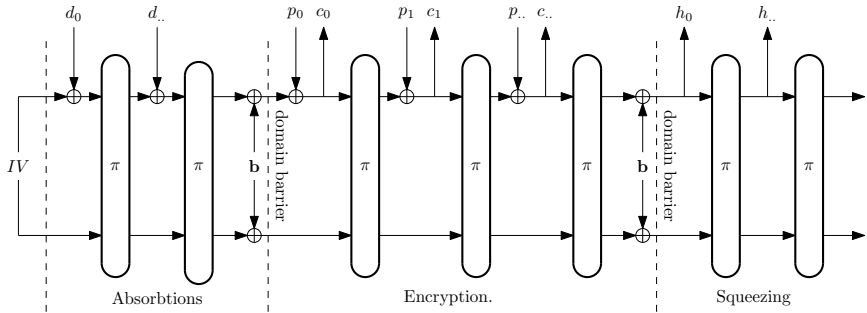
Are based on some keyless cryptographic permutation π.

Proposed and proved by G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche for:

- Collision resistant hash algorithms [eCRYPT Hash Workshop 2007], like Keccak [SHA3 Winner 2011].
- Pseudorandom extractors (PRFs and PRNGs) [CHES 2010].
- **Authenticated Encryption** (AE,AEAD) [SAC 2011].
- Keyed Message Authentication Codes (MACs) [SKEW 2011].
- Tree hashing with Sakura [IACR ePrint 2013].

.. and **BLINKER two-party protocols** [Next talk: Saarinen CT-RSA 2014].

Sponge-based Authenticated Encryption



- First the key, nonce, sequence numbers, and associated data (all represented by d_i) are absorbed in state.
- Then plaintext p_i is used to produce ciphertext c_i (or vice versa).
- Finally a MAC or hash h_i is squeezed from the state.

About π in Sponge Constructions

- π is computed only in one direction...
- Its inverse π^{-1} is **not** required for decryption or any other purpose.

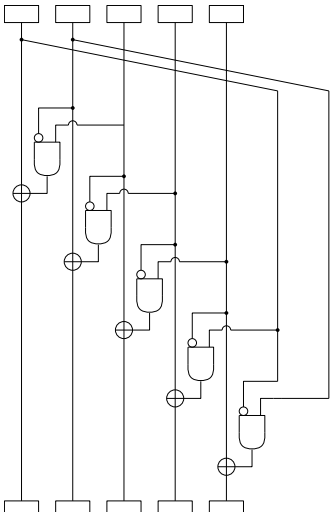
About π in Sponge Constructions

- π is computed only in one direction...
- Its inverse π^{-1} is **not** required for decryption or any other purpose.
- The state is of $b = r + c$ bits, where
  - r is the **rate** or "block size", and determines **speed**
  - c is the **capacity** and determines an upper bound for **security**
- For CBEAM, we have a $b = 256$-bit permutation with $r = 64$ **and** $c = 192$.

About π in Sponge Constructions

- π is computed only in one direction...
- Its inverse π^{-1} is **not** required for decryption or any other purpose.
- The state is of $b = r + c$ bits, where
  - r is the **rate** or "block size", and determines **speed**
  - c is the **capacity** and determines an upper bound for **security**
- For CBEAM, we have a $b = 256$-bit permutation with $r = 64$ **and** $c = 192$.
- We target Triple-DES security assuming at most 2^{40} invocations of π (8 TiB).

Background on ϕ Functions: Keccak's 5×5 - bit χ



χ is the only nonlinear component of Keccak

Usually implemented with 64× **bit-slicing** SIMD.

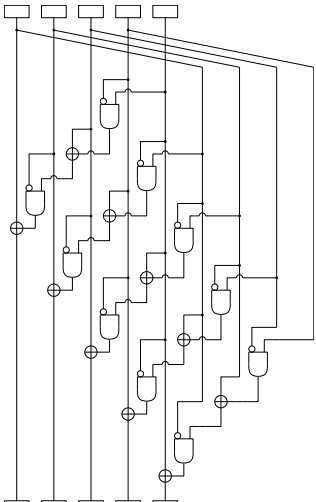
A **rotation-invariant** ϕ function:

$$\phi(x) = y \quad \Rightarrow \quad \phi(x \lll n) = y \lll n, \quad \forall\ n \in \mathbb{Z}.$$

Algebraic degree **2**.

Each output bit depends on **3** input bits.

Inverse of Keccak's 5×5 - bit χ



Inverse not required for implementing Keccak.

As an inverse of a ϕ function, χ^{-1} is also a ϕ function.

Higher circuit complexity. Algebraic degree **3**.

Each output bit depends **all** input bits.

Boura-Canteaut Inverse Algebraic Complexity Theorems

C. Boura and A. Canteaut: "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.

These theoretical results indicate that even if the inverse π^{-1} is **not** explicitly computed, an algebraically complex inverse makes the resulting iteration stronger.

We have discovered new ϕ functions with more radical computational "asymmetry" than the χ of Keccak**.**

CBEAM's "S-Box" ϕ_{16}: A 16×16 - Bit ϕ Function

First define a 5×1 - bit nonlinear function ϕ_5:

$$\phi_5(x_0, x_1, x_2, x_3, x_4) = x_0x_1x_3x_4 + x_0x_2x_3 + x_0x_1x_4 + x_1x_2x_3 + x_2x_3x_4 + x_0x_3 + x_1x_3 + x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_3 + x_4.$$

CBEAM's "S-Box" ϕ_{16}: A 16×16 - Bit ϕ Function

First define a 5×1 - bit nonlinear function ϕ_5:

$$\phi_5(x_0, x_1, x_2, x_3, x_4) = x_0x_1x_3x_4 + x_0x_2x_3 + x_0x_1x_4 + x_1x_2x_3 + x_2x_3x_4 + x_0x_3 + x_1x_3 + x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_3 + x_4.$$

This is turned into a 16×16 - bit function ϕ_{16} defined on $V[\,0 \cdots 15\,] \mapsto V'[\,0 \cdots 15\,]$ via convolution:

$$V'[\,i\,] = \phi_5\big(V[\,i\,],\ V[\,(i-1) \bmod 16\,],\ V[\,(i-2) \bmod 16\,],\ V[\,(i-3) \bmod 16\,],\ V[\,(i-4) \bmod 16\,]\big).$$

Degree is of both ϕ_5 and ϕ_{16} is clearly **4**. The Algebraic Normal Form (ANF) polynomial has **13 monomials**.

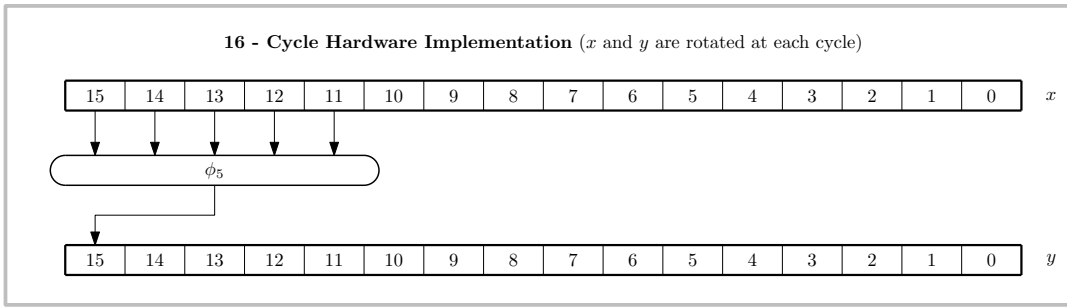
What about it's inverse ϕ_{16}^{-1} ?

- First of all, there is an inverse, which is by no means obvious.
- We tested all 2^{32} 5-input Boolean functions to find ϕ_5.
- ϕ_{16}^{-1} **has degree 11 for each output bit and 13465 monomials in its ANF**

What about it's inverse ϕ_{16}^{-1} ?

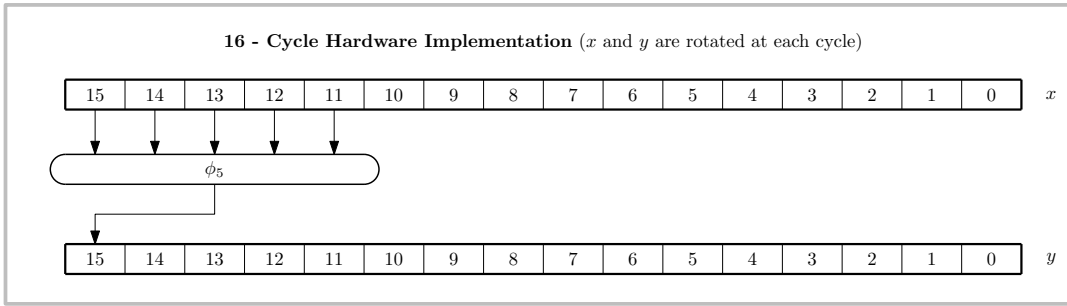
- First of all, there is an inverse, which is by no means obvious.
- We tested all 2^{32} 5-input Boolean functions to find ϕ_5.
- ϕ_{16}^{-1} **has degree 11 for each output bit and 13465 monomials in its ANF**
- Each output bit depends on all input bits (only $\frac{5}{n}$ in case of ϕ_n.)
- The degree of ϕ_n^{-1} grows **linearly** with n and the number of ANF monomials **exponentially**.

Implementation Technique 1: 16-Cycle Hardware



Two cyclic shift registers x and y and a single nonlinear ϕ_5 function.
The direction of shift does not matter.

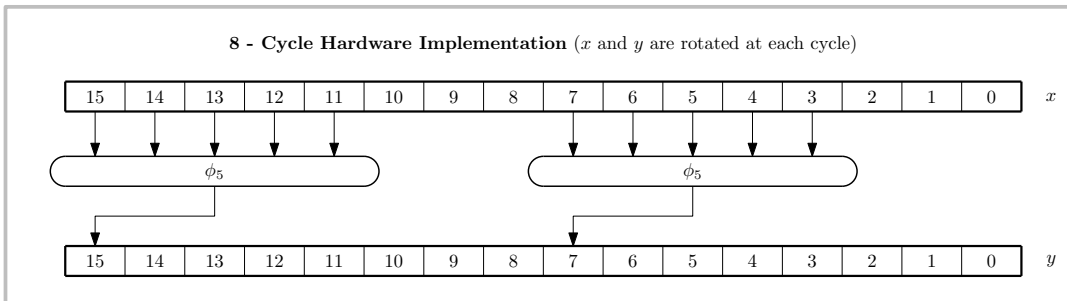
Implementation Technique 1: 16-Cycle Hardware



Two cyclic shift registers x and y and a single nonlinear ϕ_5 function.
The direction of shift does not matter.

After 16 cycles, $y = \phi_{16}(x)$. The hardware area is very small ($\approx$ 100 GE).

Implementation Technique 2: 8-Cycle Hardware

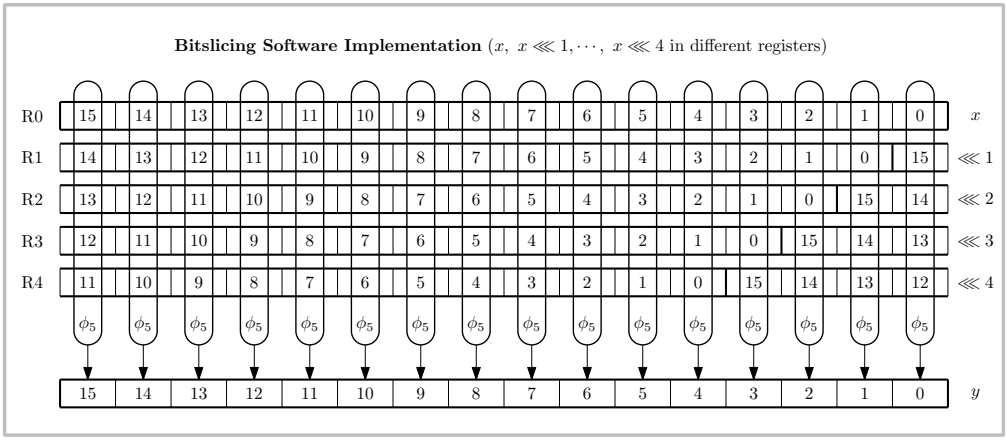


Again two cyclic shift registers x and y but two nonlinear ϕ_5 functions.

After 8 cycles, $y = \phi_{16}(x)$. The number of GE is is increased somewhat.

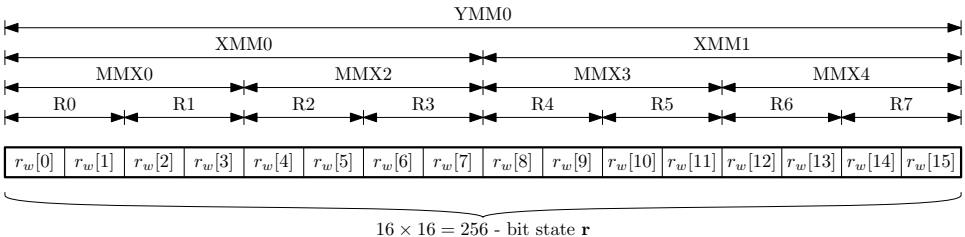
This way we can have speed/area trade-offs for $1, 2, 4, 8, 16$ cycles.

Implementation Technique 3: Rotational Bit-Slicing



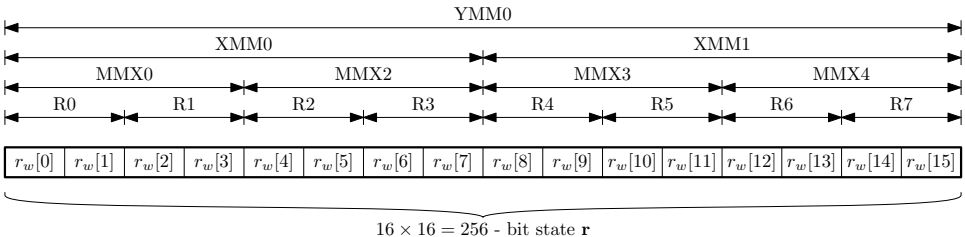
Get cyclic rotations of input word $x_i = x \lll i$ for $0 \leq i \leq 4$ into five 16-bit registers Ri: R0, R1, R2, R3, R4. Then compute $16\times$ ϕ_5 in parallel using bitwise logic.

Implementation Technique 4: Massive Parallelism



Intel Haswell AVX2 (Gen. 4 Core) arch. has **256-bit YMM** registers and instructions. Most new PCs sold this year have AVX2. Older PCs have at least SSE2, which has 128-bit XMMs.

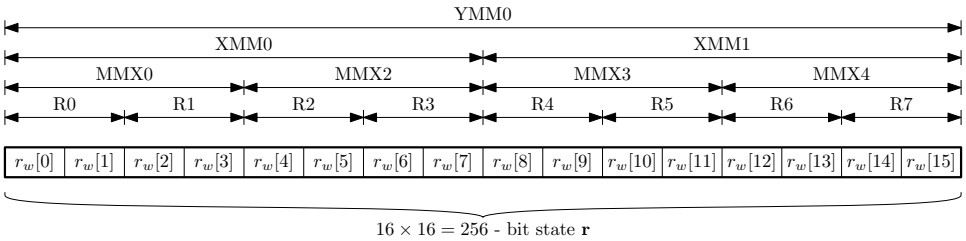
Implementation Technique 4: Massive Parallelism



Intel Haswell AVX2 (Gen. 4 Core) arch. has **256-bit YMM** registers and instructions. Most new PCs sold this year have AVX2. Older PCs have at least SSE2, which has 128-bit XMMs.

AVX2 allows 256$\times$ of ϕ_5 to be computed with just eight instructions (can be as low as 8 cycles). We have an implementation that computes 16$\times$ ϕ_{16} in one go.

Implementation Technique 4: Massive Parallelism



Intel Haswell AVX2 (Gen. 4 Core) arch. has **256-bit YMM** registers and instructions. Most new PCs sold this year have AVX2. Older PCs have at least SSE2, which has 128-bit XMMs.

AVX2 allows 256$\times$ of ϕ_5 to be computed with just eight instructions (can be as low as 8 cycles). We have an implementation that computes 16$\times$ ϕ_{16} in one go.

Massive improvement over traditional S-Box lookups of similar size in both low-end and high-end software and hardware.

TI MSP430 (16-bit) and Intel AVX2 (256-bit)

TI MSP430 assembly for $16 \times \phi_5$
with 9 instructions on 16-bit regs:

```
// r14 = Phi5(r15, .. ,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
```

TI MSP430 (16-bit) and Intel AVX2 (256-bit)

TI MSP430 assembly for $16 \times \phi_5$ with 9 instructions on 16-bit regs:

```
// r14 = Phi5(r15, .. ,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
```

AVX2 C intrinsics for $256 \times \phi_5$ with 8 instructions on 256-bit regs:

```
// 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);
```

TI MSP430 (16-bit) and Intel AVX2 (256-bit)

TI MSP430 assembly for $16 \times \phi_5$ with 9 instructions on 16-bit regs:

```
// r14 = Phi5(r15, .. ,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
```

AVX2 C intrinsics for $256 \times \phi_5$ with 8 instructions on 256-bit regs:

```
// 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);
```

This is optimal: Eight instructions required in 3-operand architectures (x86 SIMD, ARM, PPC, MIPS) and nine in sensible 2-operand architectures (MSP430).

Putting it together: Mixing Function mx

Six rounds of mx make up $\mathrm{mx}^6 = \pi$, the core CBEAM permutation.

Putting it together: Mixing Function mx

Six rounds of mx make up $\mathsf{mx}^6 = \pi$, the core CBEAM permutation.

mx is composed of addition of a round constant rc^r, bit matrix transpose, linear mixing λ, and nonlinear 256-bit mixing ϕ:

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

Putting it together: Mixing Function mx

Six rounds of mx make up $\mathsf{mx}^6 = \pi$, the core CBEAM permutation.

mx is composed of addition of a round constant rc^r, bit matrix transpose, linear mixing λ, and nonlinear 256-bit mixing ϕ:

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

- $.^T$ Transpose of the 16×16 - bit state makes mixing efficient.
- λ Parity operation on 4-bit nibbles.
- ϕ Is just 16 independent invocations of nonlinear ϕ_{16}.

Due to transpose, mx is usually implemented as double rounds mx^2 ("vertical" and "horizontal" round) in software.

Speed on 64-bit x86

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	Throughput	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

Speed on 64-bit x86

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	Throughput	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

Here $r = 64$ and therefore 8 bytes are processed per each π invocation. Approximately same speed for CBEAM holds for encryption, decryption, hashing, etc.

Speed on 64-bit x86

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	Throughput	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

Here $r = 64$ and therefore 8 bytes are processed per each π invocation. Approximately same speed for CBEAM holds for encryption, decryption, hashing, etc.

CBEAM implementation is much more compact and is not vulnerable to cache timing attacks as it is only straight-line code.

On MSP430 16-bit Sensor Platform

CBEAM is as fast as the fastest AES implementations on this platform but has significantly smaller footprint.

On MSP430 16-bit Sensor Platform

CBEAM is as fast as the fastest AES implementations on this platform but has significantly smaller footprint.

The numbers for AES are only for cores, modes of operation not included.

On MSP430 16-bit Sensor Platform

CBEAM is as fast as the fastest AES implementations on this platform but has significantly smaller footprint.

The numbers for AES are only for cores, modes of operation not included.

IP Core	Flash Size	RAM Size	Encrypt Cycles	Decrypt Cycles	Cycles / Byte
CBEAM	386	32	4369	4404	550.5
AES-128 [1]	2536	?	5432	8802	550.1
AES-128 [2]	2423	80	6600	8400	525.0
AES-256 [1]	2830	?	7552	12258	766.1

On MSP430 16-bit Sensor Platform

CBEAM is as fast as the fastest AES implementations on this platform but has significantly smaller footprint.

The numbers for AES are only for cores, modes of operation not included.

IP Core	Flash Size	RAM Size	Encrypt Cycles	Decrypt Cycles	Cycles / Byte
CBEAM	386	32	4369	4404	550.5
AES-128 [1]	2536	?	5432	8802	550.1
AES-128 [2]	2423	80	6600	8400	525.0
AES-256 [1]	2830	?	7552	12258	766.1

The IAIK [1] implementation is commercial and written in hand-optimized assembly.
The Texas Instruments [2] AES core is recommended by the SoC vendor themselves.

Security Theorems

For MonkeyWrap and BLINKER modes with t-bit authentication tags, N invocations of π, k-bit key, and max q queries:

$$\mathrm{Adv}^{\mathsf{priv}}_{\mathsf{enc}}(\mathcal{A}) < q2^{-k} + \frac{N(N+1)}{2^{c+1}} \tag{1}$$

$$\mathrm{Adv}^{\mathsf{auth}}_{\mathsf{enc}}(\mathcal{A}) < q2^{-k} + 2^{-t} + \frac{N(N+1)}{2^{c+1}} \tag{2}$$

against any single adversary $\mathcal{A}$ if $K \xleftarrow{\$} \{0,1\}^k$.

Security Theorems

For MonkeyWrap and BLINKER modes with t-bit authentication tags, N invocations of π, k-bit key, and max q queries:

$$\mathrm{Adv}^{\mathsf{priv}}_{\mathsf{enc}}(\mathcal{A}) < q2^{-k} + \frac{N(N+1)}{2^{c+1}} \tag{1}$$

$$\mathrm{Adv}^{\mathsf{auth}}_{\mathsf{enc}}(\mathcal{A}) < q2^{-k} + 2^{-t} + \frac{N(N+1)}{2^{c+1}} \tag{2}$$

against any single adversary $\mathcal{A}$ if $K \xleftarrow{\$} \{0,1\}^k$.

We claim security equivalent or better than Triple-DES with $t = 128$, $k \geq 128$, $q \leq 2^{32}$ and $N \leq 2^{40}$.

Security against Differential, Linear, and especially Algebraic cryptanalysis. We recommend the MonkeDuplex-like single-use nonce modes for additional security.

Conclusions

- Rotation-invariant ϕ functions are excellent alternatives to traditional SPNs, especially in sponge constructions where π^{-1} is not needed.

Conclusions

- Rotation-invariant ϕ functions are excellent alternatives to traditional SPNs, especially in sponge constructions where π^{-1} is not needed.
- Modern SIMD architectures allow **fast, parallel** computation of ϕ functions with "rotational bit-slicing". Much faster than S-Box lookups.
- **Compact straight-line** code, hence no cache timing attacks as in AES. Highly flexible implementations in high- and low-performance platforms.
- **Hardware-friendly**, can sacrifice cycles for gates. Suitable especially for lightweight applications due to small implementation footprint.

Conclusions

- Rotation-invariant ϕ functions are excellent alternatives to traditional SPNs, especially in sponge constructions where π^{-1} is not needed.
- Modern SIMD architectures allow **fast, parallel** computation of ϕ functions with "rotational bit-slicing". Much faster than S-Box lookups.
- **Compact straight-line** code, hence no cache timing attacks as in AES. Highly flexible implementations in high- and low-performance platforms.
- **Hardware-friendly**, can sacrifice cycles for gates. Suitable especially for lightweight applications due to small implementation footprint.
- **Further research**: discovery of surprising features of ϕ functions, refined quantification of security from feedble one-wayness.