Automatic Search for Differential Trails in ARX Ciphers

A. Biryukov V. Velichkov

Laboratory of Algorithmics, Cryptology and Security (LACS)
University of Luxembourg

RSA Conference Cryptographers' Track – 2014
February 24-28, San Francisco, USA

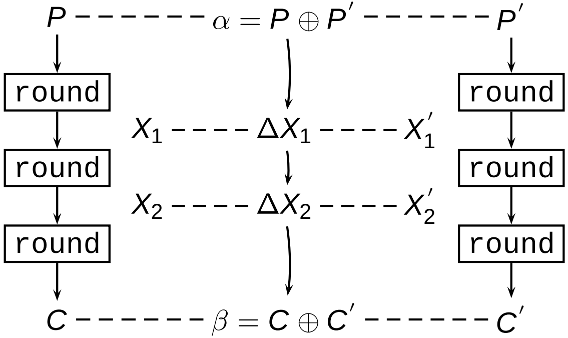
UNIVERSITÉ DU LUXEMBOURG

Outline

1. Motivation
2. Matsui's Algorithm
3. Application to ARX
4. Results

UNIVERSITÉ DU LUXEMBOURG

Differential Cryptanalysis (DC) [Biham,Shamir,1991]



Differentials, Trails and Probabilities

- Differential for r rounds:

$$(\alpha, \beta) \ .$$

- Differential trail (characteristic) for r rounds:

$$(\alpha = \Delta X_0,\ \Delta X_1\ \ldots\ \Delta X_{r-1},\ \Delta X_r = \beta) \ .$$

- Differential Probability (DP) of a single round:

$$\mathrm{DP}(\alpha \xrightarrow{F_K} \beta) = \frac{\#\{X, K : F_K(X) \oplus F_K(X \oplus \alpha) = \beta\}}{\#\{X, K\}} \ .$$

- DP of a trail $^{(*)}$:

$$\mathrm{DP}(\Delta X_0, \Delta X_1, \ldots, \Delta X_r) = \prod_{i=1}^{r} \mathrm{DP}(\Delta X_{i-1} \xrightarrow{F_{K_i}} \Delta X_i) \ .$$

$^{(*)}$ Under certain assumptions: Markov cipher, independent round keys, etc.

Difference Distribution Table (DDT)

α, β	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	2	.	.	.	2	.	2	4	.	4	2	.	.
2	.	.	.	2	.	6	2	2	.	2	.	.	.	.	2	.
3	.	.	2	.	2	.	.	.	.	4	2	.	2	.	.	4
4	.	.	.	2	.	.	6	.	.	2	.	4	2	.	.	.
5	.	4	.	.	.	2	2	.	.	.	4	.	2	.	.	2
6	.	.	.	4	.	4	.	.	.	.	.	.	2	2	2	2
7	.	.	2	2	2	.	2	.	.	2	2	.	.	.	.	4
8	.	.	.	.	.	.	2	2	.	.	.	4	.	4	2	2
9	.	2	.	.	2	.	.	4	2	.	2	2	2	.	.	.
A	.	2	2	.	.	.	.	.	6	.	.	2	.	.	4	.
B	.	.	8	.	.	2	.	2	.	.	.	.	.	2	.	2
C	.	2	.	.	2	2	2	.	.	.	.	2	.	6	.	.
D	.	4	.	.	.	.	.	4	2	.	2	.	2	.	2	.
E	.	.	2	4	2	.	.	.	6	.	.	.	.	.	2	.
F	.	2	.	.	6	.	.	.	.	4	.	2	.	.	2	.

UNIVERSITÉ DU LUXEMBOURG

Searching for the Best Trail

Matsui's branch-and-bound algorithm:

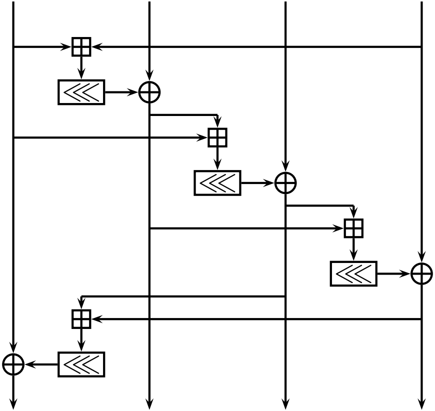
> Mitsuru Matsui, *On Correlation Between the Order of S-boxes and the Strength of DES*, EUROCRYPT'94.

- Find the best trails for up to 16 rounds of DES.
- Best $=$ maximum probability.
- Lower bound on the prob. of the best differential.
- Indication of the strength against DC; first step in a DC attack.
- **Problem:** not applicable to ciphers without S-boxes such as ARX.

UNIVERSITÉ DU LUXEMBOURG

Ciphers Based on Addition, Rotation, XOR (ARX)

In ARX ADD and XOR provide non-linearity similarly to an S-box

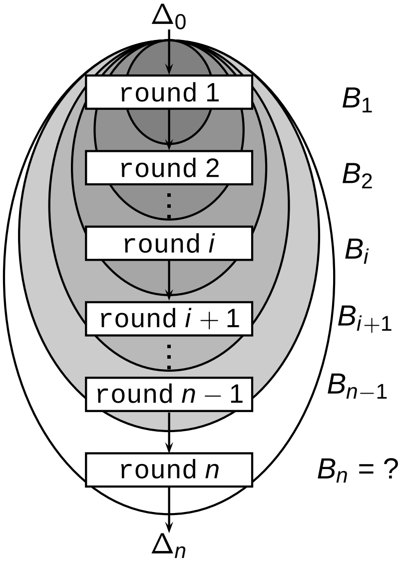


Examples: FEAL, TEA/XTEA, Salsa20, Threefish, etc.

UNIVERSITÉ DU LUXEMBOURG

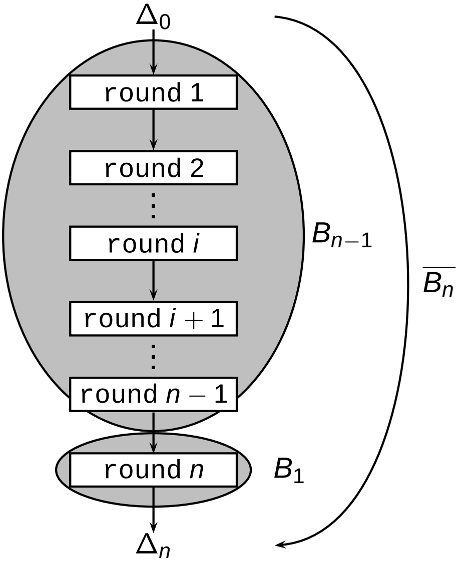
Outline

1. Motivation
2. Matsui's Algorithm
3. Application to ARX
4. Results



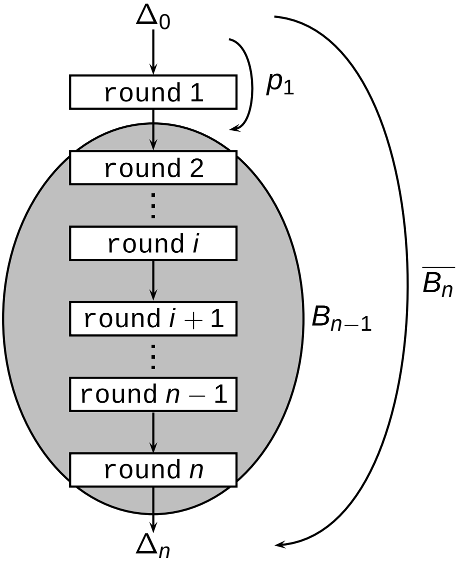
Input: best p for $n-1$ rounds:
$B_1, B_2, \ldots, B_{n-1}$

Output: best p for n rounds:
B_n

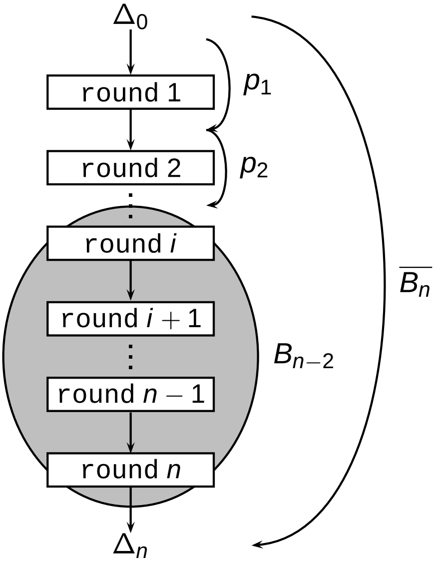


init bound:

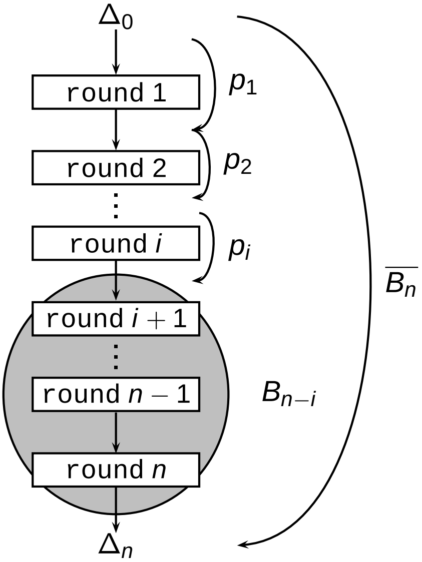
$\overline{B_n} \leftarrow B_{n-1}B_1$



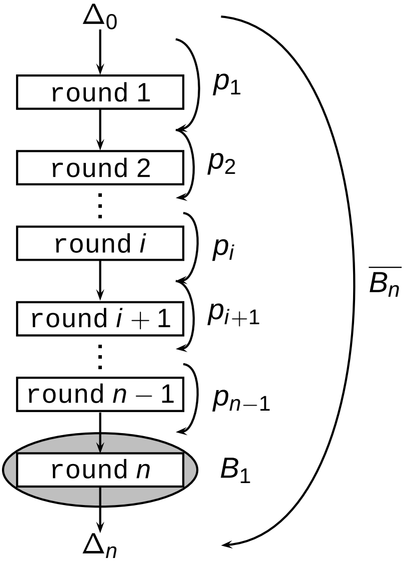
for all Δ_0 :
  if $p_1 B_{n-1} \geq \overline{B_n}$:
    call round 2



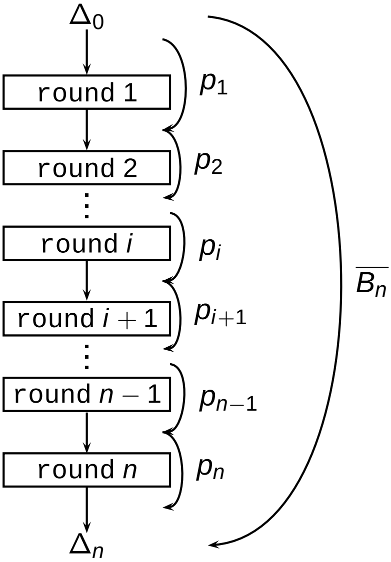
if $p_1 p_2 B_{n-2} \geq \overline{B_n}$:
call round 3



if $p_1 p_2 \ldots p_i B_{n-i} \geq \overline{B_n}$:
call round $i+1$



if $p_1 p_2 \ldots p_{n-1} B_1 \geq \overline{B_n}$:
call round n

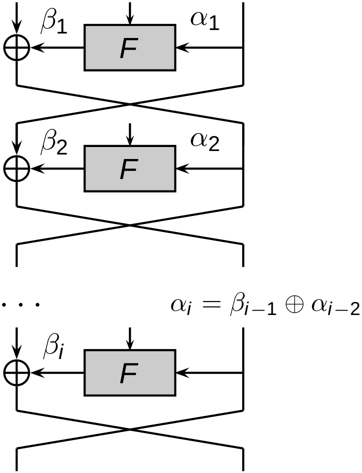


if $p = p_1 p_2 \ldots p_{n-1} p_n \geq \overline{B_n}$:

update bound:

$\overline{B_n} \leftarrow p$

Application to DES



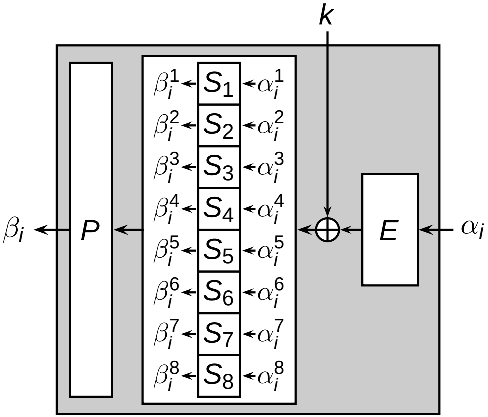
round 1
for all α_1 :
  $\beta_1 : p_1 = \max_\beta p(\alpha_1 \to \beta)$
  if $p_1 B_{n-1} \geq \overline{B_n}$:
    call round 2

round 2
for all α_2, β_2 :
  $p_2 = p(\alpha_2 \to \beta_2)$
  if $p_1 p_2 B_{n-2} \geq \overline{B_n}$
    call round 3

...

round i
$\alpha_i = \beta_{i-1} \oplus \alpha_{i-2}$
for all $\beta_i : p_i = p(\alpha_i \to \beta_i)$:
  if $p_1 p_2 \ldots p_i B_{n-i} \geq \overline{B_n}$:
    call round $i+1$

Divide-and-Conquer the Input to the S-box Layer

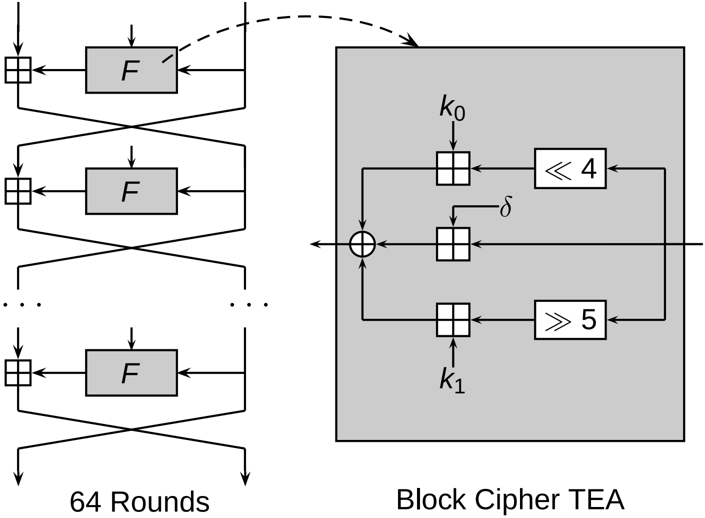


`round 2`
for $j : 1 \leq j \leq 8$:
  for all α_2^j, β_2^j :
    $p^j = p(\alpha_2^j \rightarrow \beta_2^j)$
    $p_2 = p^1 p^2 \ldots p^j$
    if $p_1 p_2 B_{n-2} \geq \overline{B_n}$
      $j = j + 1$
    if $j > 8$
      call `round 3`

Note: p is computed using the DDT of the S-boxes.

Outline

Application to ARX



Application to ARX: Problems and Solutions

Problems: ☹

1. No S-boxes $\Longrightarrow$ divide-and-conquer trick does not work.
2. Infeasible to compute full DDT for ADD or XOR.

Solutions: ☺

1. Partial difference distribution table (pDDT).
2. The country roads and highways analogy.

The Catch?

- Not guaranteed to find the (provably) best trail.

Partial Difference Distribution Table (pDDT)

Definition

A partial difference distribution table (pDDT) D for the non-linear mapping S is a DDT that contains differentials $(\alpha \xrightarrow{S} \beta)$ with probabilities larger than or equal to a fixed threshold $p_{\text{thres}} > 0$:

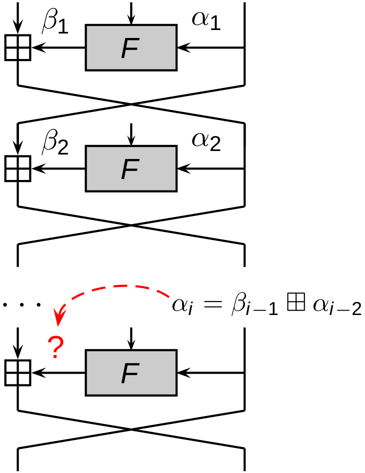
$$(\alpha \xrightarrow{S} \beta) \in D \iff p(\alpha \xrightarrow{S} \beta) \geq p_{\text{thres}} \quad .$$

Definition

A pDDT is said to be complete (resp. incomplete) if it contains all (resp. not all) differentials that have probability $\geq p_{\text{thres}}$.

UNIVERSITÉ DU LUXEMBOURG

Problem: for given α_i no transition in the pDDT



<u>round 1</u>

for all $\alpha_1, \beta_1 \in \mathrm{pDDT}$:

$p_1 = p(\alpha_1 \to \beta_1)$

if $p_1 B_{n-1} \geq \overline{B_n}$:

call round 2

<u>round 2</u>

for all $\alpha_2, \beta_2 \in \mathrm{pDDT}$:

$p_2 = p(\alpha_2 \to \beta_2)$

if $p_1 p_2 B_{n-2} \geq \overline{B_n}$

call round 3

. . .

<u>round i</u>

$\alpha_i = \beta_{i-1} \oplus \alpha_{i-2}$

$\nexists \beta : (\alpha_i, \beta) \in \mathrm{pDDT}$

UNIVERSITÉ DU LUXEMBOURG

The Highways and Country Roads Analogy

Highways and Country Roads

Definition (Highway)

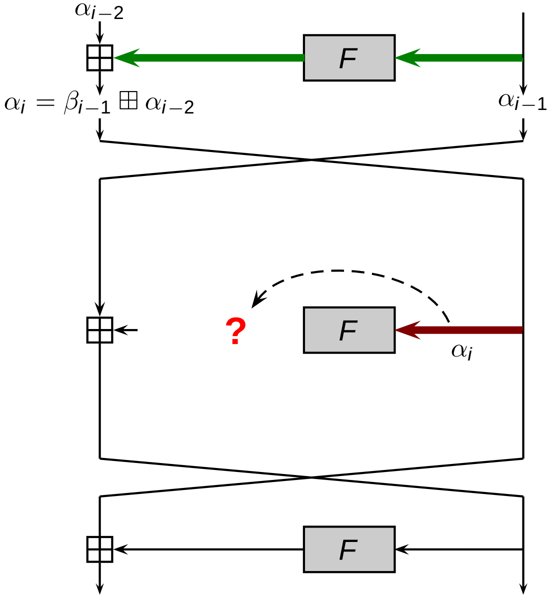
A highway is a transition $(\alpha \rightarrow \beta)$ such that $p(\alpha \rightarrow \beta) \geq p_{\text{thres}}$ for some fixed probability threshold p_{thres}.

Definition (Country road)

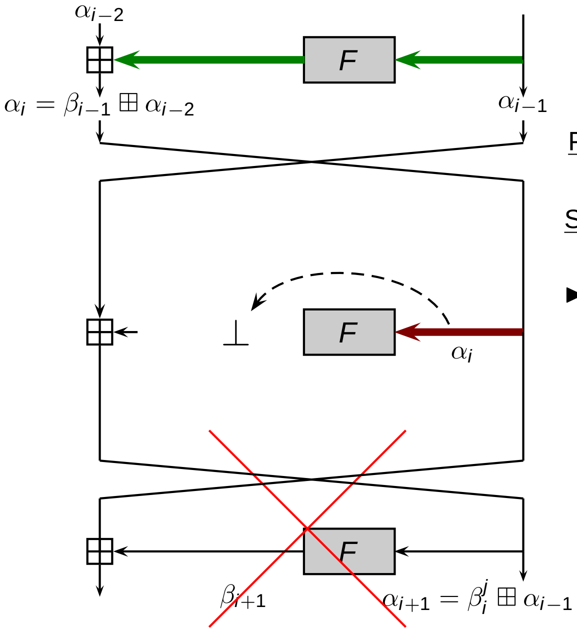
All transitions that are not highways are country roads.

Remark

All transitions in a pDDT are highways.



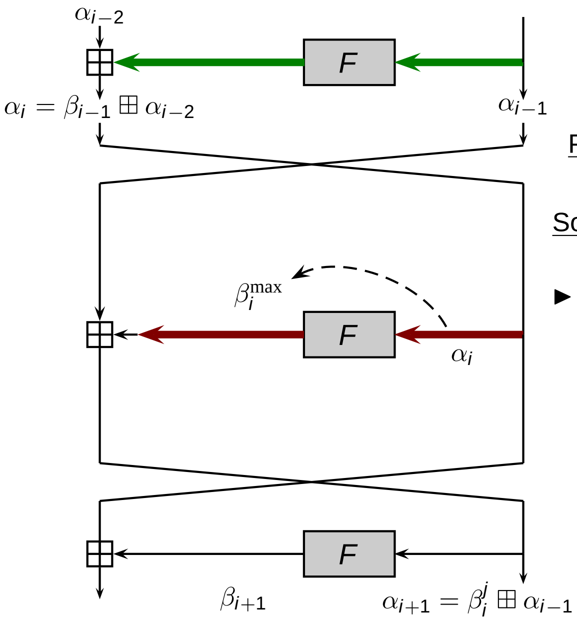
Problem: $\nexists \beta : (\alpha_i, \beta) \in D$



Problem: $\nexists \beta : (\alpha_i, \beta) \in D$

Solution 1: do nothing

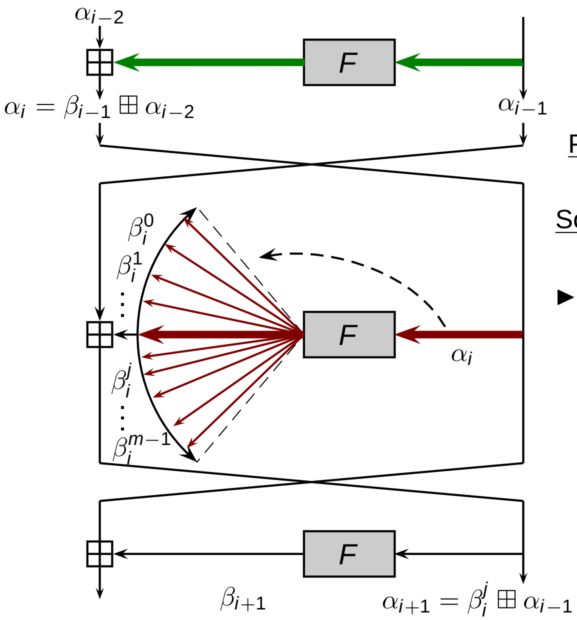
- Terminate and return $\perp$



Problem: $\nexists \beta : (\alpha_i, \beta) \in D$

Solution 2: Greedy choice:

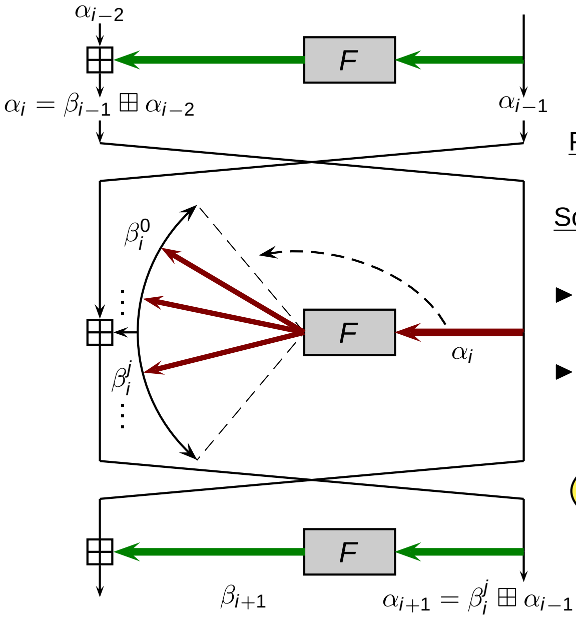
- $\beta_i^{\max} : p_i = \max_\beta p(\alpha_i \to \beta)$



<u>Problem:</u> $\nexists \beta : (\alpha_i, \beta) \in D$

<u>Solution 3:</u> Explore all β_i^j:

- $p(\alpha_i \to \beta_i^j) \geq \frac{\overline{B_n}}{p_1 p_2 \ldots p_{i-1} B_{n-i}}$



<u>Problem:</u> $\nexists \beta : (\alpha_i, \beta) \in D$

<u>Solution 4:</u> Explore some β_i^j:

- $p(\alpha_i \to \beta_i^j) \geq \frac{\overline{B_n}}{p_1 p_2 \ldots p_{i-1} B_{n-i}}$
- $\beta_i^j : (\alpha_{i+1}, \beta_{i+1}) \in D$

back-to-the-highway trick

Threshold Search

Application of Matsui's algorithm to ARX (threshold search):

1. Derive an expression for computing the DP of F.
2. Compute the pDDT of F (the highways table).
3. Execute the modified Matsui's algorithm with the pDDT as input.

Outline

UNIVERSITÉ DU LUXEMBOURG

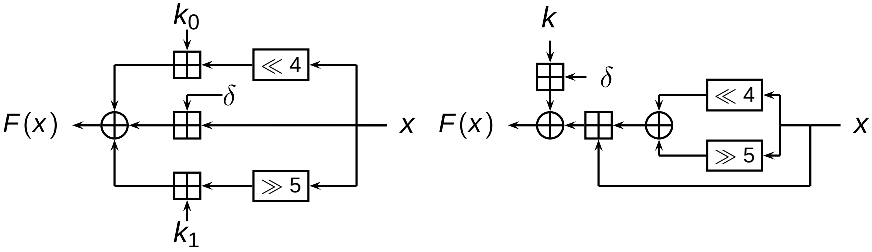


Figure: The F-functions of TEA (left) and XTEA (right).

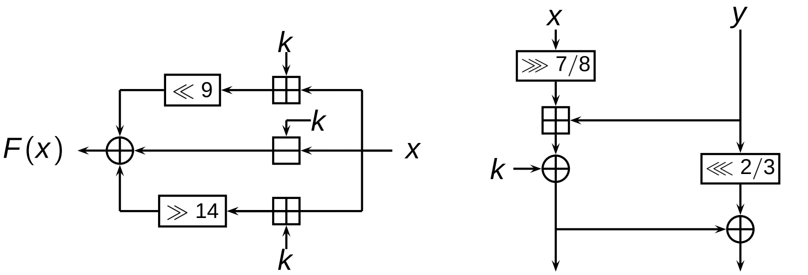


Figure: The F-functions of RAIDEN (left) and SPECK (right).

Results

Cipher	Type of Trail	#Rounds Covered	#Rounds Total	Ref.
TEA	Trunc.	5	64	[Moon02+]
	Trunc.	7		[Chen12+]
	Trunc.	8		[Hong03+, Bogdanov12+]
	Full	**18**		[**Sect. 8**]
XTEA	Trunc.	6	64	[Moon02+]
	Trunc.	7		[Chen12+]
	Trunc.	8		[Bogdanov12+]
	Full	**14**, 14		[**Sect. 8**], [Hong03+]
SPECK32	**Full**	**9**, 8^*	22	[**Sect. 8**],[Abed13+]
SPECK48	**Full**	**10**, 10^*	$22/23$	[**Sect. 8**], [Abed13+]
SPECK64	**Full**	**13**, 13^*	$26/27$	[**Sect. 8**], [Abed13+]
RAIDEN	**Full**	**32**	32	**[Sect. 8]**

$^{(*)}$ differentials

Results: Update on SPECK (FSE 2014)

Cipher	Type of Trail	#Rounds Covered	#Rounds Total	Ref.
TEA	Trunc.	5	64	[Moon02+]
	Trunc.	7		[Chen12+]
	Trunc.	8		[Hong03+, Bogdanov12+]
	Full	**18**		[**Sect. 8**]
XTEA	Trunc.	6	64	[Moon02+]
	Trunc.	7		[Chen12+]
	Trunc.	8		[Bogdanov12+]
	Full	**14**, 14		[**Sect. 8**], [Hong03+]
SPECK32	**Full**	**9**, 8*	22	[**FSE ’14**],[Abed13+]
SPECK48	**Full**	**11**, 10*	22/23	[**FSE ’14**], [Abed13+]
SPECK64	**Full**	**14**, 13*	26/27	[**FSE ’14**], [Abed13+]
RAIDEN	**Full**	**32**	32	**[Sect. 8]**

(*) differentials

Takeaway Message

Threshold search: first application of Matsui's algorithm to ARX.

The idea is very simple. It is the technique that's difficult.
– James Joyce on Ulysses

- Simple idea:
  - Matsui + pDDT + Highways and Country roads.
- Difficult technique:
  - **Choice of parameters**: p_{thres}, $\mathrm{HW}_{\text{thres}}$, size of pDDT.
  - **pDDT:** complete vs. incomplete; pre-computed vs. dynamic update.
  - **Search strategy:** top-to-bottom vs. start-from-the-middle.
  - **Limit #CR:** back-to-highway (TEA) vs. limit-by-HW (SPECK).
- Depends on the cipher: **not a black-box tool to be applied as-is.**

YAARX: Yet Another ARX Toolkit

General toolkit for analysis of ARX:

`https://github.com/vesselinux/yaarx`

Documentation:

`http://vesselinux.github.io/yaarx/index.html`

- Complements Gaëtan Leurent's **ARX Toolkit**.
- Extends **The S-function Toolkit** by Mouha et al.

Thank you for your attention!

uni.lu
UNIVERSITÉ DU LUXEMBOURG

Questions


A
B
Questions?
Luxemburg City
Trier
Konz
Wadern
Schweich
Echternach
Larochette
Mersch
Junglinster
Lorentzweiler
Betzdorf
Niederanven
Nittel
Wiltingen
Pluwig
Waldrach
Kell am See
Freudenburg
Mettlach
Remich
Perl
Sierck-les-Bains
Saarburg
Palzem
Nunkirchen
Thalfang
Malborn
Heidenburg
Fell
Pfalzel
Naurath
Leiwen
Newel
Ralingen
Beggen

Backup Slides

Backup Slides

Monotonicity of the DP of XOR and ADD

Proposition

The differential probabilities (DP) of XOR and ADD (resp. xdp^{+} *and* $\mathrm{adp}^{\oplus}$*) are monotonously decreasing with the word size n of the differences* α, β, γ*:*

$$p_n \leq \ldots \leq p_{k+1} \leq p_k \leq p_{k-1} \leq \ldots \leq p_1 \quad ,$$

where $p_k = \mathrm{DP}(\alpha_k, \beta_k \rightarrow \gamma_k) : \ n \geq k \geq 1$ *and* x_k *denotes the k LSB-s of the difference x.*

Corollary

For fixed p_{thres} *the pDDT of XOR (ADD) can be computed bitwise over the words of the differences from LSB to MSB.*

UNIVERSITÉ DU LUXEMBOURG

Bitwise Computation of pDDT for XOR and ADD

Algorithm 1 Compute pDDT for XOR (ADD).

Input: n, p_{thres}, k, p_k, α_k, β_k, γ_k.

Output: Partial DDT D: $(\alpha, \beta, \gamma) \in D : \mathrm{DP}(\alpha, \beta \to \gamma) \geq p_{\mathrm{thres}}$.

if $n = k$ **then**
    Add $(\alpha, \beta, \gamma) \leftarrow (\alpha_k, \beta_k, \gamma_k)$ to D
    return
for $x, y, x \in \{0, 1\}$ **do**
    $\alpha_{k+1} \leftarrow x|\alpha_k$, $\beta_{k+1} \leftarrow y|\beta_k$, $\gamma_{k+1} \leftarrow z|\gamma_k$.
    $p_{k+1} = \mathrm{DP}(\alpha_{k+1}, \beta_{k+1} \to \gamma_{k+1})$
    if $p_{k+1} \geq p_{\mathrm{thres}}$ **then**
        Procedure 1$(n, p_{\mathrm{thres}}, k+1, p_{k+1}, \alpha_{k+1}, \beta_{k+1}, \gamma_{k+1})$

UNIVERSITÉ DU LUXEMBOURG

Computation of Partial DDT: Timings, $n = 32$

	ADD		XOR	
p_{thres}	**DDT size**	**Time**	**DDT size**	**Time**
0.1	$252,940$	36, *sec.*	$3,951,388$	2.29, *min.*
0.07	$361,420$	37, *sec.*	$3,951,388$	1.23, *min.*
0.05	$3,038,668$	5.35, *min.*	$167,065,948$	44.36, *min.*
0.01	$2,715,532,204$	17.46, *hours.*	–	–

UNIVERSITÉ DU LUXEMBOURG

TEA r	β		α	$\log_2 p$
1	F	$\leftarrow$	FFFFFFFF	−3.62
2	0	$\leftarrow$	0	−0.00
3	F	$\leftarrow$	FFFFFFFF	−2.87
4	0	$\leftarrow$	F	−7.90
5	FFFFFFF1	$\leftarrow$	FFFFFFFF	−3.60
6	0	$\leftarrow$	0	−0.00
7	FFFFFFF1	$\leftarrow$	FFFFFFFF	−2.78
8	2	$\leftarrow$	FFFFFFF1	−8.66
9	F	$\leftarrow$	1	−3.57
10	0	$\leftarrow$	0	−0.00
11	FFFFFFF1	$\leftarrow$	1	−2.87
12	FFFFFFFE	$\leftarrow$	FFFFFFF1	−7.90
13	F	$\leftarrow$	FFFFFFFF	−3.59
14	0	$\leftarrow$	0	−0.00
15	11	$\leftarrow$	FFFFFFFF	−2.79
16	0	$\leftarrow$	11	−8.83
17	FFFFFFEF	$\leftarrow$	FFFFFFFF	−3.61
18	0	$\leftarrow$	0	−0.00
$\sum_r \log_2 p_r$				−62.6
$\log_2 p_{\text{thres}}$				−4.32
#hways				68
Time:				21.36 min.

Copyright

UNIVERSITÉ DU LUXEMBOURG

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

Author: Markku-Juhani O. Saarinen

Presented by: Jean-Philippe Aumasson

KUDELSKI SECURITY

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

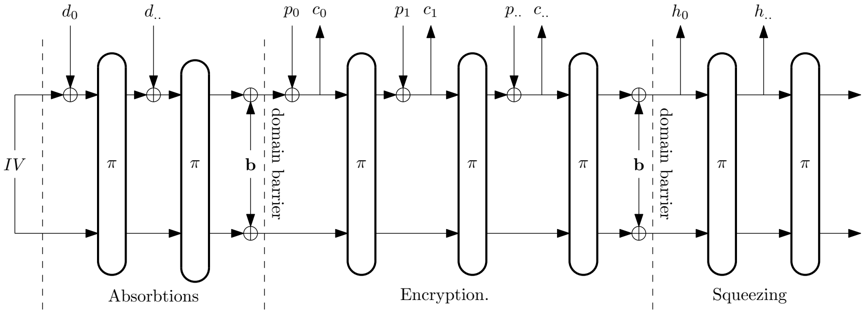
Are based on some keyless cryptographic permutation π.

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.

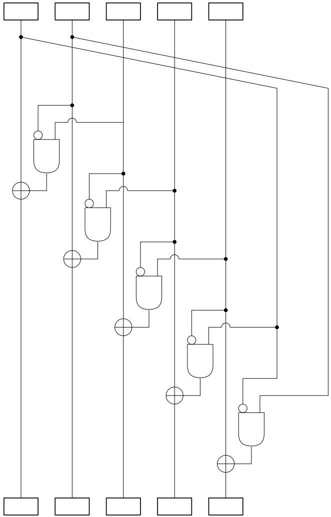
About π in Sponge Constructions

- 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 $64\times$ **bit-slicing** SIMD.

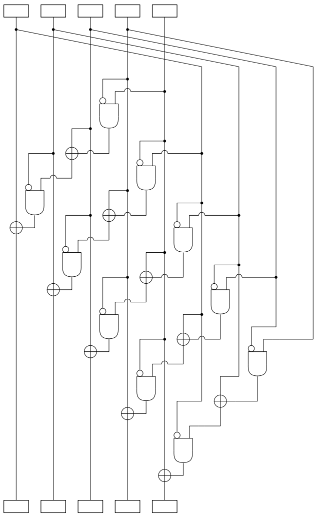
A **rotation-invariant** ϕ function:

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

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**.

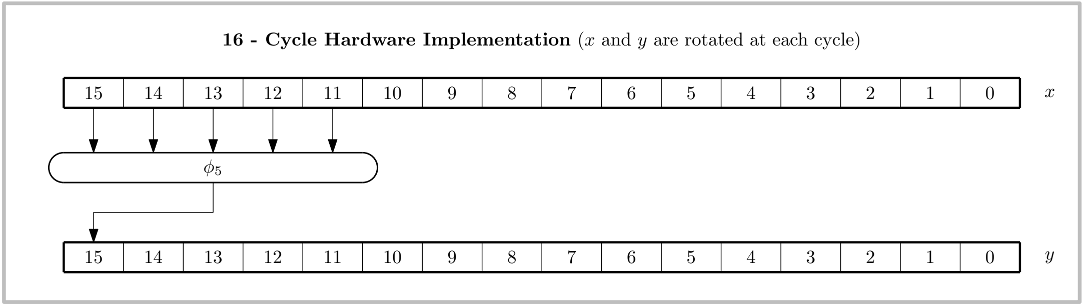
What about it's inverse ϕ_{16}^{-1} ?

- 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} ?

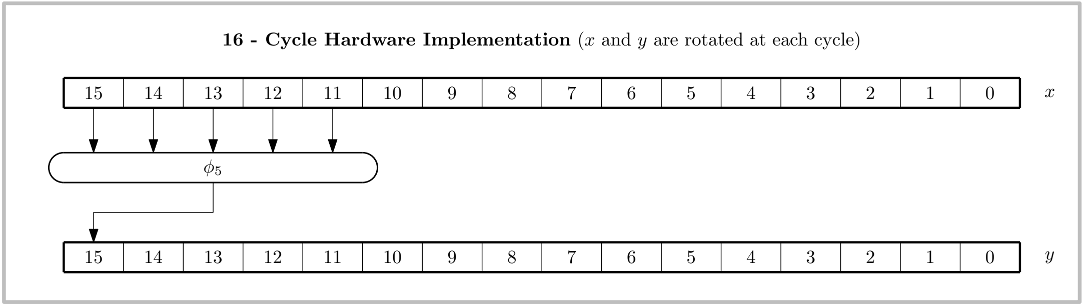
- First of all, there is an inverse, which is by no means obvious.
- We tested all 2^{32} 5-input Boolean functions to find ϕ_5.
- **ϕ_{16}^{-1} has degree 11 for each output bit and 13465 monomials in its ANF**
- Each output bit depends on all input bits (only $\frac{5}{n}$ in case of ϕ_n.)
- The degree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

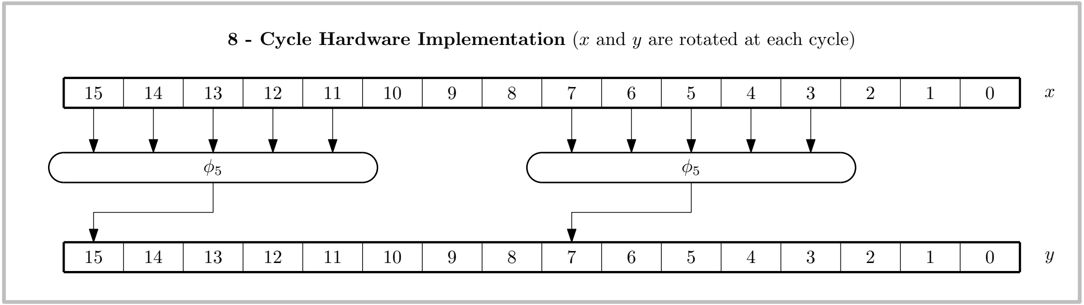
Implementation Technique 1: 16-Cycle Hardware



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

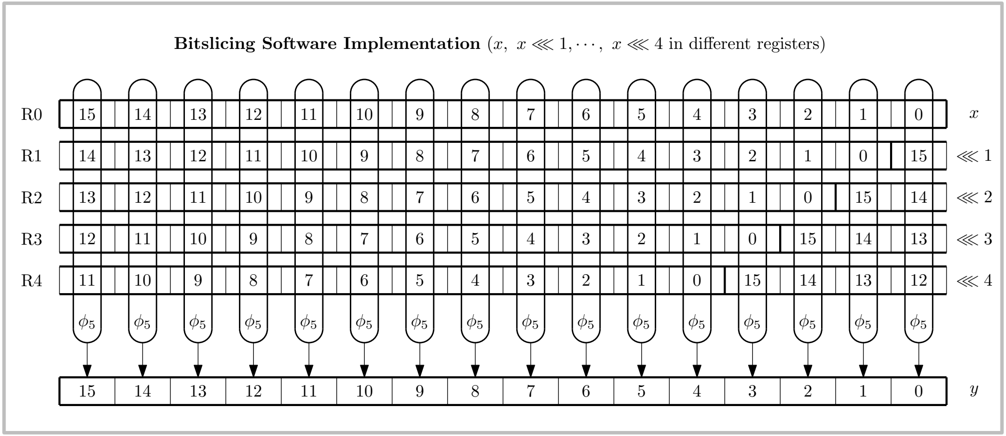


Again two cyclic shift registers x and y but two nonlinear ϕ_5 functions.

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

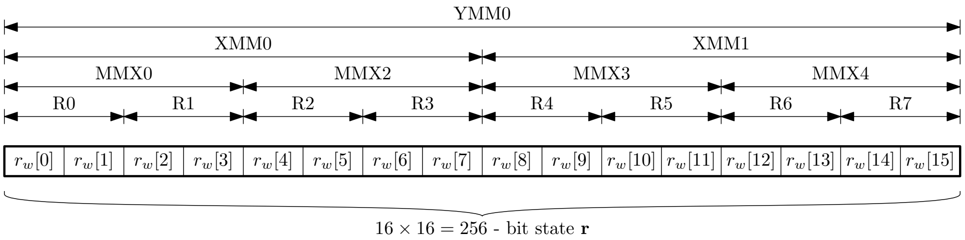
This way we can have speed/area trade-offs for $1, 2, 4, 8, 16$ cycles.

Implementation Technique 3: Rotational Bit-Slicing



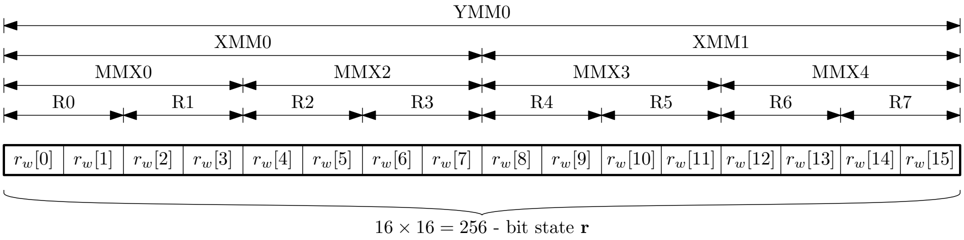
Get cyclic rotations of input word $x_i = x \lll i$ for $0 \leq i \leq 4$ into five 16-bit registers Ri: R0, R1, R2, R3, R4. Then compute 16× ϕ_5 **in parallel** using **bitwise logic**.

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.

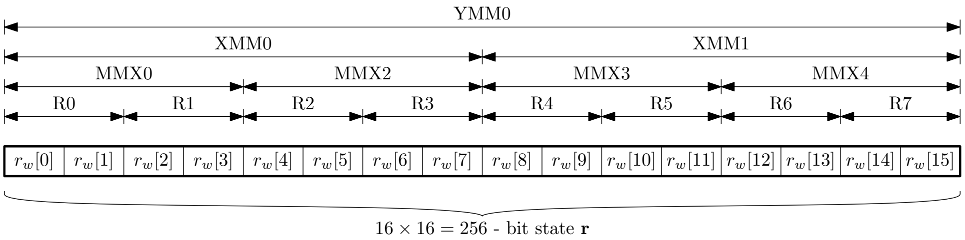
Implementation Technique 4: Massive Parallelism



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 $\mathsf{mx}^6 = \pi$, 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:

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

- $\cdot^T$ 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 $K \stackrel{\$}{\leftarrow} \{0,1\}^k$.

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.

Beyond Modes: Building a Secure Record Protocol from a Cryptographic Sponge Permutation

Author: Markku-Juhani O. Saarinen

Presented by: Jean-Philippe Aumasson

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

Background: Complex, Insecure Legacy Protocols

All of the RFC / de facto standard networking security protocols—SSL3, SSH2, TLS, IPSEC, PPTP, and wireless WPA2 (together with its predecessors)—consist of two largely independent protocols:

1. The **handshake / authentication protocol** which establishes a shared secret K.
2. The **transport / record protocol** which provides communications security.

Background: Complex, Insecure Legacy Protocols

All of the RFC / de facto standard networking security protocols—SSL3, SSH2, TLS, IPSEC, PPTP, and wireless WPA2 (together with its predecessors)—consist of two largely independent protocols:

1. The **handshake / authentication protocol** which establishes a shared secret K.
2. The **transport / record protocol** which provides communications security.

In addition to the plaintext P, data items required by record protocols to perform authenticated encryption at **each direction** usually include at least the following:

- S Incremental message sequence number.
- IV Initialization vector for block ciphers.
- K_e Key for the symmetric encryption algorithm.
- K_a Key for the message authentication algorithm.

That is $2 \times 4 = 8$ separate cryptovariables and at least two different algorithms (HMAC and block cipher) in addition to PRFs that derive these.

Motivation for BLINKER

Legacy protocols are unsuited for ultra-lightweight applications.

Academic research has focused on lightweight **primitives**, and suitable lightweight, **general purpose communications protocols** have not been proposed.

Motivation for BLINKER

Legacy protocols are unsuited for ultra-lightweight applications.

Academic research has focused on lightweight **primitives**, and suitable lightweight, **general purpose communications protocols** have not been proposed.

We need a generic **short-distance lightweight link layer** security provider that can function independently from upper layer application functions.

- Design with mathematical and legal **provability** in mind.
- Aim at simplicity and small footprint: use a **single** sponge permutation for key derivation, confidentiality, integrity, etc. (Instead of distinct algorithms.)
- Use a **single state variable** in both directions, instead of 8+ cryptovariables.
- Ideally this protocol would be realizable with semi-autonomous integrated hardware, without much CPU or MCU involvement.

Two-party Synchronization

Legacy protocols use two independent channels: one from Alice to Bob ($A \to B$) and another from Bob to Alice ($B \to A$).

Two-party Synchronization

Legacy protocols use two independent channels: one from Alice to Bob ($A \to B$) and another from Bob to Alice ($B \to A$).

Example. Consider the following three transcripts:

$$T1: \quad B \to A: M_2, \quad A \to B: M_1, \quad A \to B: M_3$$

$$T2: \quad A \to B: M_1, \quad B \to A: M_2, \quad A \to B: M_3$$

$$T3: \quad A \to B: M_1, \quad A \to B: M_3, \quad B \to A: M_2$$

These three exchanges have precisely the **same valid representation** on the two channels when sent over IPSEC, TLS, SSL, or SSH protocols.

The same authentication codes will match, etc.

The Synchronization Problem of Two-Channel Protocols.

Despite individual message authentication, the **interwoven order** of the sequence of back-and-forth messages cannot be unambiguously determined and authenticated with legacy protocols.

The Synchronization Problem of Two-Channel Protocols.

Despite individual message authentication, the **interwoven order** of the sequence of back-and-forth messages cannot be unambiguously determined and authenticated with legacy protocols.

This is why transaction records are often authenticated on the **application level as well**, adding an another layer of complexity.

Issue also affects basic **end-user interactive security** as portions of server messaging can be maliciously delayed, encouraging the user to react to partial information.

The Synchronization Problem of Two-Channel Protocols.

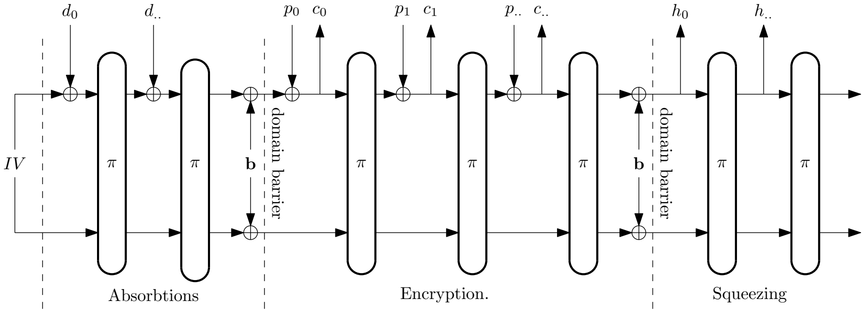
Despite individual message authentication, the **interwoven order** of the sequence of back-and-forth messages cannot be unambiguously determined and authenticated with legacy protocols.

This is why transaction records are often authenticated on the **application level as well**, adding an another layer of complexity.

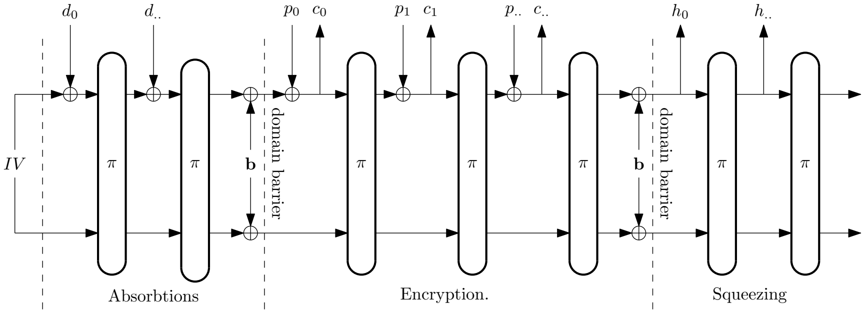
Issue also affects basic **end-user interactive security** as portions of server messaging can be maliciously delayed, encouraging the user to react to partial information.

Legal perspective on unambiguous session transcripts:
Steven J. Murdoch and Ross Anderson: "Security Protocols and Evidence: Where Many Payment Systems Fail." Financial Cryptography and Data Security 2014, 3 – 7 March 2014, Barbados.

Recap: Sponge-based Authenticated Encryption



Recap: Sponge-based Authenticated Encryption



1. Key, nonce, and associated data (d_i) are absorbed in state.
2. Plaintext p_i is used to produce ciphertext c_i (or vice versa).
3. MAC h_i is squeezed from the state.
4. **Why not use that final state as IV for reply and go straight to Step 2 ?**

Simplification

Legacy protocol encryption of P to C with 4 cryptovariables:

$$C = f_{\mathrm{cs}}(P, S, IV, K_e, K_a).$$

Simplification

Legacy protocol encryption of P to C with 4 cryptovariables:

$$C = f_{\mathsf{cs}}(P, S, IV, K_e, K_a).$$

Decryption can fail in authentication (auth tag is in C):

$$f_{\mathsf{cs}}^{-1}(C, S, IV, K_e, K_a) = P \; or \; \mathsf{FAIL}.$$

Simplification

Legacy protocol encryption of P to C with 4 cryptovariables:

$$C = f_{\mathrm{cs}}(P, S, IV, K_e, K_a).$$

Decryption can fail in authentication (auth tag is in C):

$$f_{\mathrm{cs}}^{-1}(C, S, IV, K_e, K_a) = P \; or \; \mathrm{FAIL}.$$

In sponges we have a state S_i, plaintext P_i, and some padding info that produces a new state and ciphertext (including a MAC):

$$(S_{i+1}, C_i) = \mathrm{enc}(\ S_i, P_i, pad\ ).$$

Simplification

Legacy protocol encryption of P to C with 4 cryptovariables:

$$C = f_{\text{cs}}(P, S, IV, K_e, K_a).$$

Decryption can fail in authentication (auth tag is in C):

$$f_{\text{cs}}^{-1}(C, S, IV, K_e, K_a) = P \; or \; \text{FAIL}.$$

In sponges we have a state S_i, plaintext P_i, and some padding info that produces a new state and ciphertext (including a MAC):

$$(S_{i+1}, C_i) = \text{enc}(\; S_i, P_i, pad \;).$$

The decoding function dec() produces the same S_{i+1} and P_i from the ciphertext and equivalent S_i and padding, synchronizing the state between sender and receiver:

$$(S_{i+1}, P_i) = \text{dec}(\; S_i, C_i, pad \;) \; or \; \text{FAIL}.$$

Security Goals

Protocol designers should have provable bounds on these three goals:

priv The ciphertext result C of $\text{enc}(S, P, \textit{pad})$ must be indistinguishable from random when S is random and P may be chosen by the attacker.

Security Goals

Protocol designers should have provable bounds on these three goals:

priv The ciphertext result C of enc(S, P, pad) must be indistinguishable from random when S is random and P may be chosen by the attacker.

auth The probability of an adversary of choosing a message C that does not result in a FAIL in dec(S, C, pad) without knowledge of S is bound by a function of the authentication tag size t and number of trials.

Security Goals

Protocol designers should have provable bounds on these three goals:

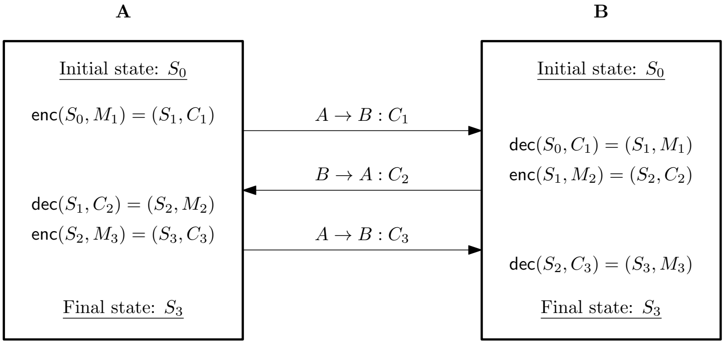
priv The ciphertext result C of enc(S, P, pad) must be indistinguishable from random when S is random and P may be chosen by the attacker.

auth The probability of an adversary of choosing a message C that does not result in a FAIL in dec(S, C, pad) without knowledge of S is bound by a function of the authentication tag size t and number of trials.

sync Each party can verify that all previous messages of the session have been correctly received and the absolute order in which messages were sent.

First two are standard Authentication Encryption requirements, the **last one is new**.

Solution: Just continue to use the state in reply!



Simplified interchange of three messages whose plaintext equivalents are $A \to B : M_1$, $B \to A : M_2$, $A \to B : M_3$, utilizing a synchronized secret state variables S_i.

The order of messages cannot be modified and hence this exchange is **sync**-secure !

So .. it's Half-Duplex ?

Half-duplex links may not seem ubiquitous to developers due to the use of the socket programming paradigm. Full-duplex illusion is often achieved by time-division duplexing.

So .. it's Half-Duplex ?

Half-duplex links may not seem ubiquitous to developers due to the use of the socket programming paradigm. Full-duplex illusion is often achieved by time-division duplexing.

- Half-duplex is physically prevalent on sensor networks, IoT and last-hop radio links: **Bluetooth** and **IEEE 802.15.4 ZigBee** are half-duplex.
- In addition to wireless last-hop transports, most **RFID, smart card** [ISO 7816-4, ISO 18000-63], and **industrial control** [MODBUS] communications are implemented under a query-response model and are therefore effectively half-duplex.

So .. it's Half-Duplex ?

Half-duplex links may not seem ubiquitous to developers due to the use of the socket programming paradigm. Full-duplex illusion is often achieved by time-division duplexing.

- Half-duplex is physically prevalent on sensor networks, IoT and last-hop radio links: **Bluetooth** and **IEEE 802.15.4 ZigBee** are half-duplex.
- In addition to wireless last-hop transports, most **RFID, smart card** [ISO 7816-4, ISO 18000-63], and **industrial control** [MODBUS] communications are implemented under a query-response model and are therefore effectively half-duplex.
- Half-duplex links can be established wirelessly with unpaired frequencies (same frequency in both directions), or with (twisted-wire / single contact) serial links. These are a typical scenarios in lightweight time-divide communications, our **specific targets**.

Unambiguous Session Transcripts via Better Domain Separation

- **Keccak** only has domain separation between data input and hash output.

Unambiguous Session Transcripts via Better Domain Separation

- **Keccak** only has domain separation between data input and hash output.
- **Keccak**-160/256/512 are distinguished from each other via different rates r, not via padding or IV; different hardware for different hash sizes?!

Unambiguous Session Transcripts via Better Domain Separation

- **Keccak** only has domain separation between data input and hash output.
- **Keccak**-160/256/512 are distinguished from each other via different rates r, not via padding or IV; different hardware for different hash sizes?!
- **SpongeWrap** extended this to domain separation with *frame bits* between key material, payload data, and message authentication tag.

Unambiguous Session Transcripts via Better Domain Separation

- **Keccak** only has domain separation between data input and hash output.
- **Keccak**-160/256/512 are distinguished from each other via different rates r, not via padding or IV; different hardware for different hash sizes?!
- **SpongeWrap** extended this to domain separation with *frame bits* between key material, payload data, and message authentication tag.
- **Sakura** added further frame bits yet again to facilitate tree hashing.

Unambiguous Session Transcripts via Better Domain Separation

- **Keccak** only has domain separation between data input and hash output.
- **Keccak**-160/256/512 are distinguished from each other via different rates r, not via padding or IV; different hardware for different hash sizes?!
- **SpongeWrap** extended this to domain separation with *frame bits* between key material, payload data, and message authentication tag.
- **Sakura** added further frame bits yet again to facilitate tree hashing.

We want to have a extensible padding mechanism that allow same hardware to be used for **any purpose**.

Unambiguous Session Transcripts via Better Domain Separation

- **Keccak** only has domain separation between data input and hash output.
- **Keccak**-160/256/512 are distinguished from each other via different rates r, not via padding or IV; different hardware for different hash sizes?!
- **SpongeWrap** extended this to domain separation with *frame bits* between key material, payload data, and message authentication tag.
- **Sakura** added further frame bits yet again to facilitate tree hashing.

We want to have a extensible padding mechanism that allow same hardware to be used for **any purpose**.

Key feature in BLINKER: **originator** bits; whether sponge input is from Alice, Bob, Both (e.g. DH Secret), or none in particular..

Multiplexing the Sponge

We retain **one d-bit word D in S^c for domain separation**; $S^c = (S^d \mid\mid S^{c'})$ with $c' = c - d$. The iteration for arbitrary absorption, squeezing, and encryption is now:

$$S_{i+1} = \pi(\ S_i^r \oplus M_i \mid\mid S_i^d \oplus D_i \mid\mid S_i^{c'}\ ).$$

Multiplexing the Sponge

We retain **one d-bit word D in S^c for domain separation**; $S^c = (S^d \mid\mid S^{c'})$ with $c' = c - d$. The iteration for arbitrary absorption, squeezing, and encryption is now:

$$S_{i+1} = \pi(\ S_i^r \oplus M_i \mid\mid S_i^d \oplus D_i \mid\mid S_i^{c'}\ ).$$

For decryption we have the following update function:

$$S_{i+1} = \pi(\ C_i \mid\mid S_i^d \oplus D_i \mid\mid S_i^{c'}\ ).$$

Multiplexing the Sponge

We retain **one d-bit word D in S^c for domain separation**; $S^c = (S^d \mid\mid S^{c'})$ with $c' = c - d$. The iteration for arbitrary absorption, squeezing, and encryption is now:

$$S_{i+1} = \pi(\ S_i^r \oplus M_i \mid\mid S_i^d \oplus D_i \mid\mid S_i^{c'}\ ).$$

For decryption we have the following update function:

$$S_{i+1} = \pi(\ C_i \mid\mid S_i^d \oplus D_i \mid\mid S_i^{c'}\ ).$$

In BLINKER, $d = 16$ bits. We estimate that the capacity suffers only by few bits.

Even hash and MAC outputs are padded (length padding + domain separation). This protects against length-extension.

Multiplex Word

Depending on protocol state and the intended usage of message block, multiple bits are set simultaneously.

Bit	Mask	When set
0	0x0001	This is a full input or output block (r bits).
1	0x0002	This is the final block of this data element.
4	0x0004	Block is an input to sponge ("absorption").
3	0x0008	Block is output from sponge ("squeezing").
4	0x0010	Associated Authenticated Data (in).
5	0x0020	Secret key (in).
6	0x0040	Nonce or sequence number (in).
7	0x0080	Encryption / Decryption (in and out).
8	0x0100	Hash block (out).
9	0x0200	Keyed Message Authentication Code (MAC) (out).
10	0x0400	Block for state storage or reloading (in or out).
11	0x0800	Pseudo Random Number Generator (PRNG) (feed or out).
12	0x1000	Originating from Alice – client / slave.
13	0x2000	Originating from Bob – server / master.
14	0x4000	Tree chaining Node.
15	0x8000	Tree final Node.

Example: Authentication and Record Protocol Flow (1)

We first **absorb and transmit the identities** I_a and I_b of Alice and Bob into the state. These are not encrypted as S_0 is the Initialization Vector.
We recommend identifiers I_a and $I_b$to be random strings of sufficient size (at least 128 bits).

Example: Authentication and Record Protocol Flow (1)

We first **absorb and transmit the identities** I_a and I_b of Alice and Bob into the state. These are not encrypted as S_0 is the Initialization Vector.
We recommend identifiers I_a and $I_b$to be random strings of sufficient size (at least 128 bits).

This is an **optional step** that helps both parties select the correct shared secret K.

$$
\begin{aligned}
(S_1, M_1) &= \mathsf{enc}(S_0, I_a, \mathtt{0x108C}) \mid A \rightarrow B : M_1 \\
(S_2, M_2) &= \mathsf{enc}(S_1, I_b, \mathtt{0x208C}) \mid B \rightarrow A : M_2 \\
S_3 &= \mathsf{enc}(S_2, K, \mathtt{0x3024}) \mid no\ transmission
\end{aligned}
$$

Example: Authentication and Record Protocol Flow (1)

We first **absorb and transmit the identities** I_a and I_b of Alice and Bob into the state. These are not encrypted as S_0 is the Initialization Vector. We recommend identifiers I_a and $I_b$to be random strings of sufficient size (at least 128 bits).

This is an **optional step** that helps both parties select the correct shared secret K.

$$(S_1, M_1) = \mathsf{enc}(S_0, I_a, \texttt{0x108C}) \mid A \to B : M_1$$
$$(S_2, M_2) = \mathsf{enc}(S_1, I_b, \texttt{0x208C}) \mid B \to A : M_2$$
$$S_3 = \mathsf{enc}(S_2, K, \texttt{0x3024}) \mid \textit{no transmission}$$

K may be derived with a lightweight asymmetric key exchange method such as Curve25519 [Bernstein 2006] or derived from passwords. It is **never transmitted**, but just absorbed in the secret state to produce S_3 from S_2.

Example: Authentication and Record Protocol Flow (2)

Two **random nonces R_a and R_b are required** for challenge-response authentication and to make the session unique.

$$(S_4, M_3) = \mathsf{enc}(S_3, R_a, \texttt{0x10CC}) \mid A \rightarrow B : M_3$$
$$(S_5, M_4) = \mathsf{enc}(S_4, R_b, \texttt{0x20CC}) \mid B \rightarrow A : M_4$$

Example: Authentication and Record Protocol Flow (2)

Two **random nonces R_a and R_b are required** for challenge-response authentication and to make the session unique.

$$(S_4, M_3) = \text{enc}(S_3, R_a, \texttt{0x10CC}) \mid A \to B : M_3$$
$$(S_5, M_4) = \text{enc}(S_4, R_b, \texttt{0x20CC}) \mid B \to A : M_4$$

We may now perform **mutual authentication** with tags of t bits:

$$(S_6, M_5) = \text{enc}(S_5, 0^t, \texttt{0x1208}) \mid A \to B : M_5$$
$$(S_7, M_6) = \text{enc}(S_6, 0^t, \texttt{0x2208}) \mid B \to A : M_6$$

Checking M_5 and M_6 completes mutual authentication. By an inductive process we see that the session secret S_7 is now dependent upon randomizers from both parties and the original shared secret is not leaked if the sponge satisfies our security axioms.

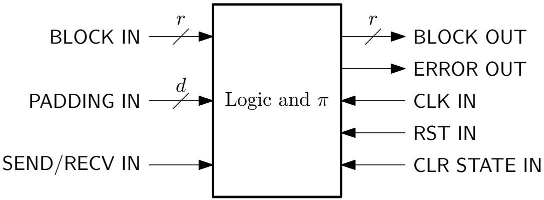
Example: Authentication and Record Protocol Flow (3)

After this, plaintexts P_a (for $A \to B$) and P_b (for $B \to A$) can be encrypted, transmitted and authenticated by repeating the following exchange:

$$
\begin{aligned}
(S_{i+1}, M_a) &= \mathrm{enc}(S_i, P_a, \mathtt{0x108C}) \mid A \to B : M_a \\
(S_{i+2}, T_a) &= \mathrm{enc}(S_{i+1}, 0^t, \mathtt{0x1208}) \mid A \to B : T_a \\
(S_{i+3}, M_b) &= \mathrm{enc}(S_{i+2}, P_b, \mathtt{0x208C}) \mid B \to A : M_b \\
(S_{i+4}, T_b) &= \mathrm{enc}(S_{i+3}, 0^t, \mathtt{0x2208}) \mid B \to A : T_b
\end{aligned}
$$

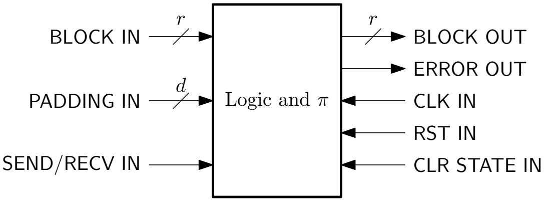
Due to explicit padding it is easy to show **inductively** that the entire message flow is authenticated if appropriate checks are made.

Semi-Autonomous Hardware and Lightweight Demo Software



If we incorporate K management in the comms hardware session secrets never have to leave (and cannot leave) a specific hardware component and are inaccessible to MCU/CPU app.

Semi-Autonomous Hardware and Lightweight Demo Software



If we incorporate K management in the comms hardware session secrets never have to leave (and cannot leave) a specific hardware component and are inaccessible to MCU/CPU app.

Such separation is very difficult (and costly) to achieve with SSL and other legacy protocols which generally require CPU/MCU interaction to create encryption and authentication keys from session secrets.

cblnk.com

https://www.cblnk.com/cb0cat/

cblnk.com

KUDELSKI SECURITY

cb0cat 0.20131216

1. Introduction and License

This is a quick tutorial to the cb0cat multi-use cryptographic tool, which can be used to hash, encrypt, and decrypt files and to establish secure communication links over TCP. cb0cat has been designed to be self-contained, portable, and extremely lightweight (currently only about 1500 lines).

Conclusions

- **Provably secure lightweight record protocols** can be built from a single π permutation. No need for separte PRF, HMAC, Block Cipher, and a mode of operation. Significantly reduces implementation ROM / Flash footprint.

Conclusions

- **Provably secure lightweight record protocols** can be built from a single π permutation. No need for separte PRF, HMAC, Block Cipher, and a mode of operation. Significantly reduces implementation ROM / Flash footprint.
- Working memory required to implement the entire two-way BLINKER protocol is only **slightly more than b bits** for the state. Legacy protocols require additional storage for two sequence counters / nonces, authenticators, cipher round keys, etc.

Conclusions

- **Provably secure lightweight record protocols** can be built from a single π permutation. No need for separte PRF, HMAC, Block Cipher, and a mode of operation. Significantly reduces implementation ROM / Flash footprint.
- Working memory required to implement the entire two-way BLINKER protocol is only **slightly more than b bits** for the state. Legacy protocols require additional storage for two sequence counters / nonces, authenticators, cipher round keys, etc.
- Explicit padding and continuous authentication **resolves synchronization issues** and allows straight-forward inductive security proofs based only on a single assumption.

Conclusions

- **Provably secure lightweight record protocols** can be built from a single π permutation. No need for separte PRF, HMAC, Block Cipher, and a mode of operation. Significantly reduces implementation ROM / Flash footprint.
- Working memory required to implement the entire two-way BLINKER protocol is only **slightly more than b bits** for the state. Legacy protocols require additional storage for two sequence counters / nonces, authenticators, cipher round keys, etc.
- Explicit padding and continuous authentication **resolves synchronization issues** and allows straight-forward inductive security proofs based only on a single assumption.
- Provable transcripts: final "state hash" **proves the integrity of an entire transaction** rather than an individual message.

Conclusions

- **Provably secure lightweight record protocols** can be built from a single π permutation. No need for separte PRF, HMAC, Block Cipher, and a mode of operation. Significantly reduces implementation ROM / Flash footprint.
- Working memory required to implement the entire two-way BLINKER protocol is only **slightly more than b bits** for the state. Legacy protocols require additional storage for two sequence counters / nonces, authenticators, cipher round keys, etc.
- Explicit padding and continuous authentication **resolves synchronization issues** and allows straight-forward inductive security proofs based only on a single assumption.
- Provable transcripts: final "state hash" **proves the integrity of an entire transaction** rather than an individual message.
- **BLINKER**: a class of lightweight half-duplex protocols. Especially suited for IoT, Smart Card, RFID, NFC, and other last-lap security.