V L Mahesh (CS16M046)
Muhammad Arsath K F (CS16S035)

Journey of MAD

What does it stand for?

- Is it the creator‘s names?
- Is it the names of the processes involved in it?
- Is it a reference in pop culture?

What does it stand for?

Currently Nothing

(But, we want it to mean something)

Presentation Agenda

M

A

D

Presentation Agenda

M istakes

A pproach

D eployment

Mistakes

Mistakes

Goals

- Complex (imitating/inspired by nature)
- Fast
- Solid (no compromise)
- <10 rounds
- Grand, Unique, Fresh

Mistakes

- Can we do it without Sboxes?

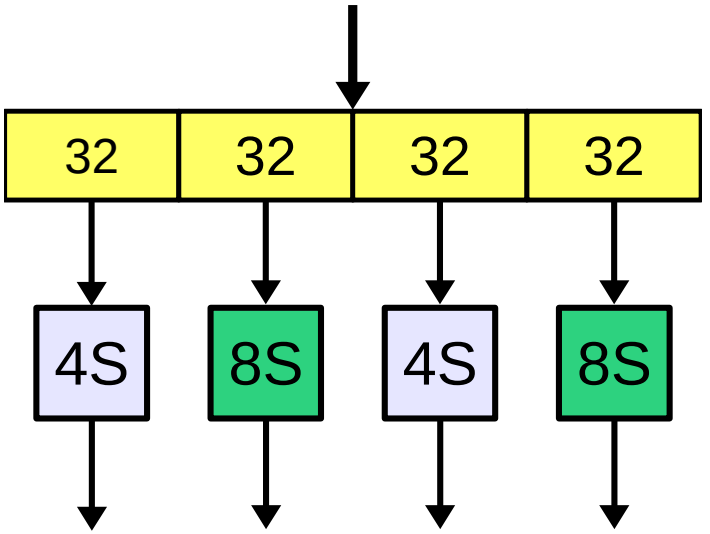
  (like SHACAL-2)

Mistakes

- Can we do it without Sboxes? (Obviously Not - MtoQ)
  (like SHACAL-2)
- Complicated S-layer, Simple P-layer
- Sbox Selection

Mistakes | Complicated S-layer

- Different patterns and sizes of Sboxes
- 4x4, 8x8 and a combination of both
- 4-8-4 blocks
- Groups of 4x4 sboxes and 8x8 sboxes



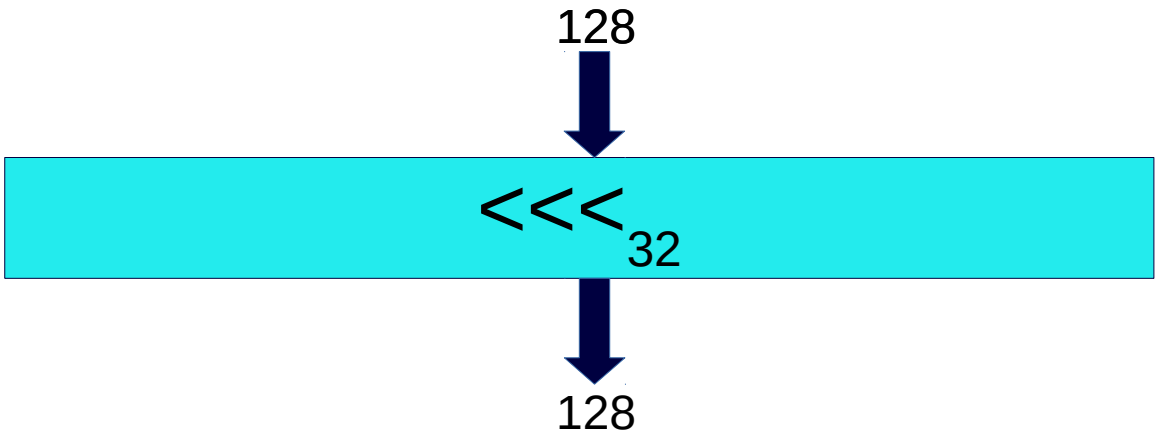
Mistakes | Complicated S-layer

- Different patterns and sizes of Sboxes
- 4x4, 8x8 and a combination of both
- 4-8-4 blocks
- Groups of 4x4 sboxes and 8x8 sboxes

- Made analysis very difficult and error prone

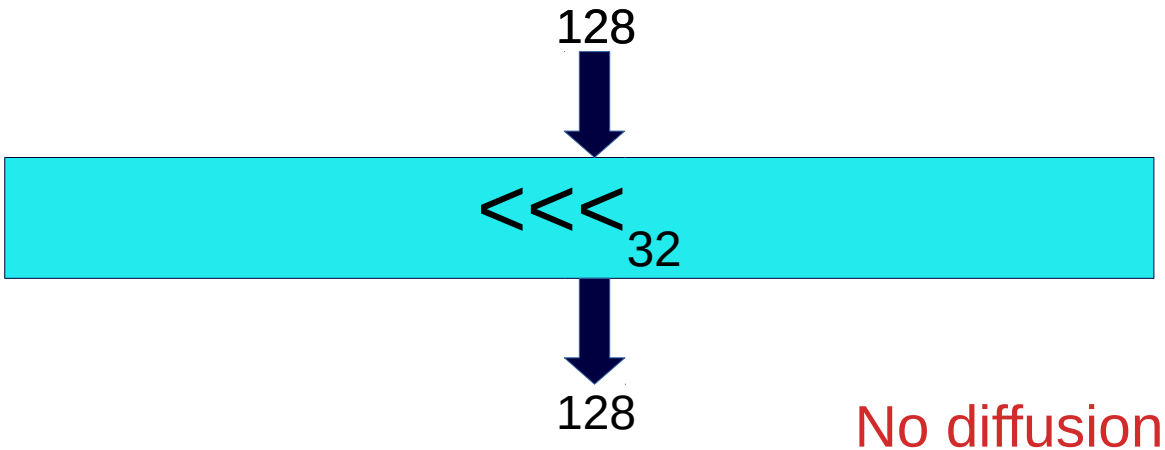
Mistakes | Simple P-layer

- What is wrong with this? (In conjunction with previous layer)



Mistakes | Simple P-layer

- What is wrong with this? (In conjunction with previous layer)



No diffusion !

Mistakes | Sbox Selection

- First, we used the **PRESENT** Sbox

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Mistakes | Sbox Selection

- First, we used the **PRESENT** Sbox
- We got into a problem while doing LC
  - It‘s output bit 0 had a linear dependence with input bits {0,3}

Algorithm & Source	**S-Box** 0123456789ABCDEF	**Bit 0** LS	**Bit 0** deg	**Bit 1** LS	**Bit 1** deg	**Bit 2** LS	**Bit 2** deg	**Bit 3** LS	**Bit 3** deg
Lucifer S0 [37]	CF7AEDB026319458	{}	3	{}	3	{}	3	{}	3
Lucifer S1 [37]	72E93B04CD1A6F85	{}	3	{}	3	{}	3	{}	3
Present [9]	C56B90AD3EF84712	{0,3}	2	{}	3	{}	3	{}	3

Approach

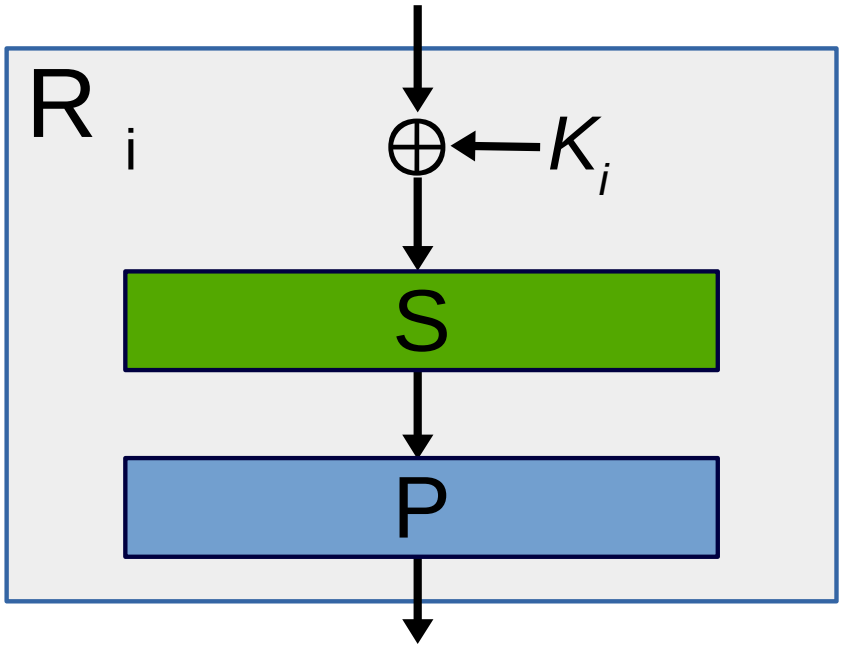
Approach

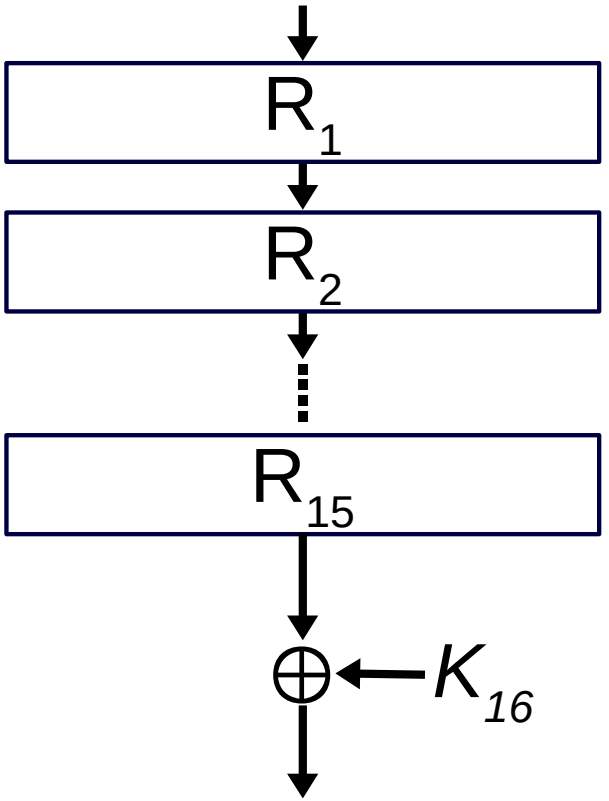
New Goals

- Simple
- Efficient
- Scalable
- <20 rounds
- Learn something

Approach | General Structure

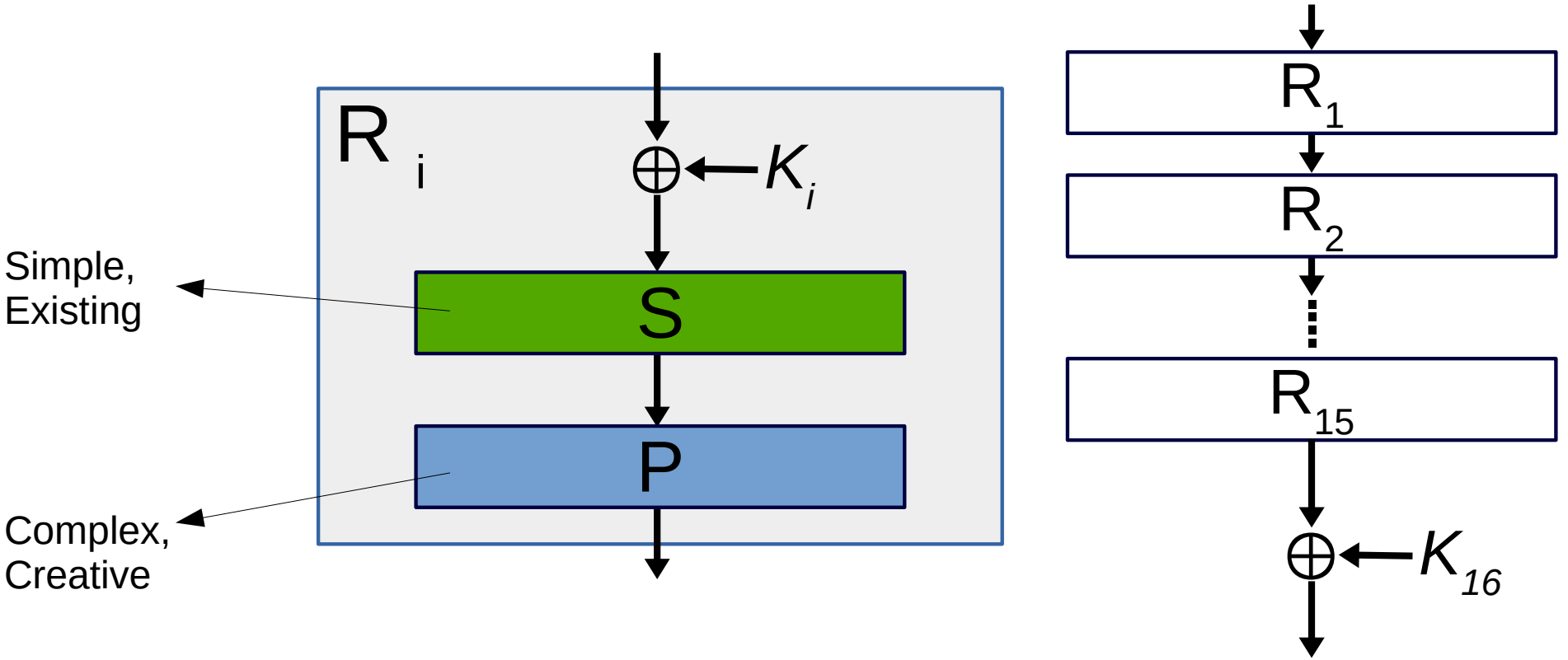
- Straightforward SPN





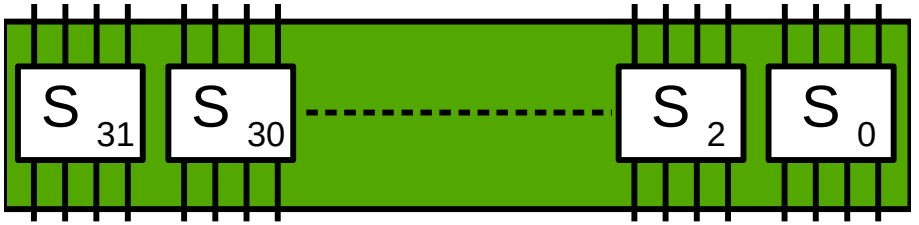
Approach | General Structure

- Straightforward SPN



Approach | S-layer

- Extremely Simple
- Uses the **SERPENT S3** Sbox
  - 4x4
  - Said to be Golden
  - Very nice properties



Algorithm & Source	S-Box 0123456789ABCDEF	BN #	DC p	DC n_d	LC ϵ	LC n_l	Bit 0 LS	Bit 0 deg	Bit 1 LS	Bit 1 deg	Bit 2 LS	Bit 2 deg	Bit 3 LS	Bit 3 deg
Serpent S0 [1]	38F1A65BED42709C	3	$^1/_4$	24	$^1/_4$	36	{}	3	{}	3	{}	3	{1,2}	2
Serpent S1 [1]	FC27905A1BE86D34	3	$^1/_4$	24	$^1/_4$	36	{}	3	{}	3	{2,3}	2	{}	3
Serpent S2 [1]	86793CAFD1E40B52	3	$^1/_4$	24	$^1/_4$	36	{1,3}	2	{}	3	{}	3	{}	3
Serpent S3 [1]	0FB8C963D124A75E	3	$^1/_4$	18	$^1/_4$	32	{}	3	{}	3	{}	3	{}	3

(We did calculate the above properties)

Approach | P-layer

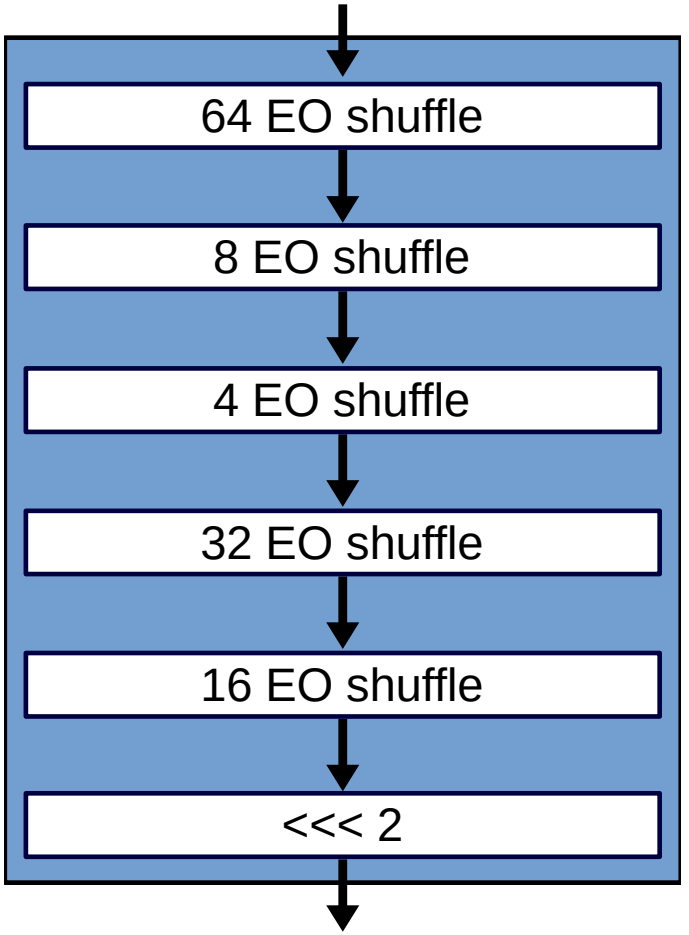
- Concept of EO Shuffle

EO

$$c = (a_{odd} >> 1) \mid b_{odd}$$
$$d = (b_{even} << 1) \mid a_{even}$$

EO^{-1}

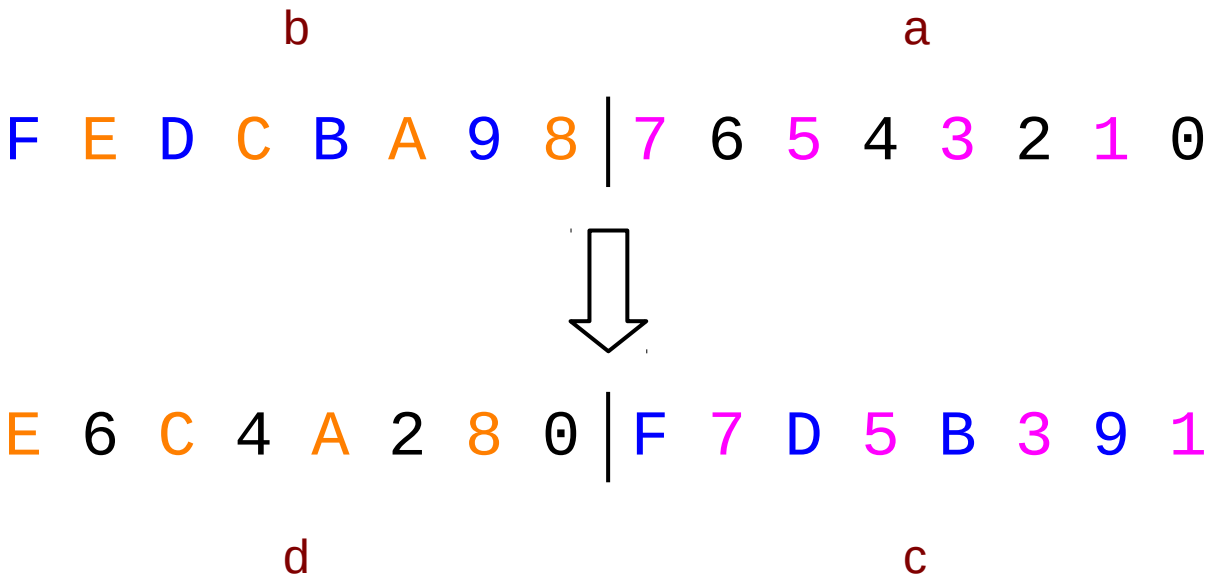
$$a = (c_{even} << 1) \mid d_{even}$$
$$b = (d_{odd} >> 1) \mid c_{odd}$$



- We tried different sequences
- More inversions → Better randomness

Approach | EO Shuffle

8 EO Shuffle on a 16-bit number



- Odd & Even bits are extracted using bit masks

Approach | Analysis

- Achieves Resistance to both Linear & Differential Cryptanalysis in 10 rounds
- Calculated Cumulative Bias & Propagation Ratios respectively (just like in the Stinson Textbook)
- Since Serpent S3 only had ¼ & ½ in LAT/DDT, we could easily calculate them w/o Multiprecision Libs.
- But, we‘ve used 15 rounds to be resistant to even advanced cryptanalysis techniques, just in case!

Deployment

Deployment

Implementation Guidelines

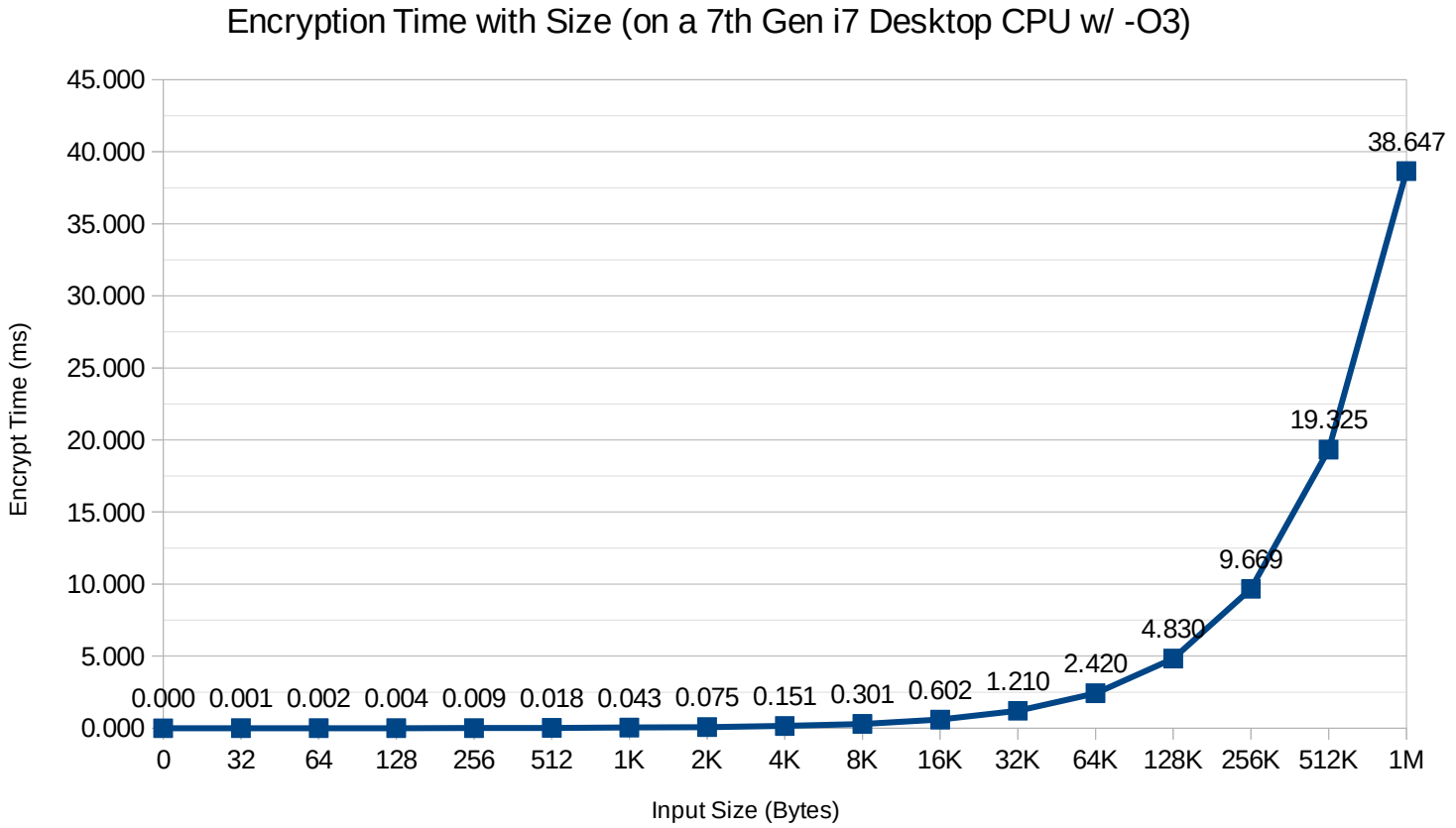
- Use previously learned concepts from different areas (Cryptography, Computer Architecture, Compilers)
- Readable
- Portable
- Auto-Optimized
- Efficient Hardware Implementation

Deployment | At-a-glance

- Stores the 128-bit block as two `uint64_t`
- Relies on a Compiler‘s matured toolset, rather than our own experience in assembly code optimization
- Separate set of functions for encrypt and decrypt to avoid branches (we need them because SPN)
- Cache Efficient

  (only 1 4x4 Sbox/R-Sbox = 32 Bytes)

Deployment | Performance



- As can be seen, 38.6 ms » ~26 MB/s
- Pretty good for a first implementation

Deployment | Details

How did we achieve this?

- No expensive operations (only shifts,and,or)

Deployment | Details

How did we achieve this?

- No expensive operations (only shifts,and,or)
- No function calls (Function Inlining)

Deployment | Details

How did we achieve this?

- No expensive operations (only shifts,and,or)
- No function calls (Function Inlining)
- Constant size loops (Loop Unrolling)

Deployment | Details

How did we achieve this?

- No expensive operations (only shifts,and,or)
- No function calls (Function Inlining)
- Constant size loops (Loop Unrolling)
- Auto-vectorized (SSE,AVX,etc…)

Deployment | Details

How did we achieve this?

- No expensive operations (only shifts,and,or)
- No function calls (Function Inlining)
- Constant size loops (Loop Unrolling)
- Auto-vectorized (SSE,AVX,etc…)

All of these without losing portability, readability