

# Design Philosophy of Rijndael

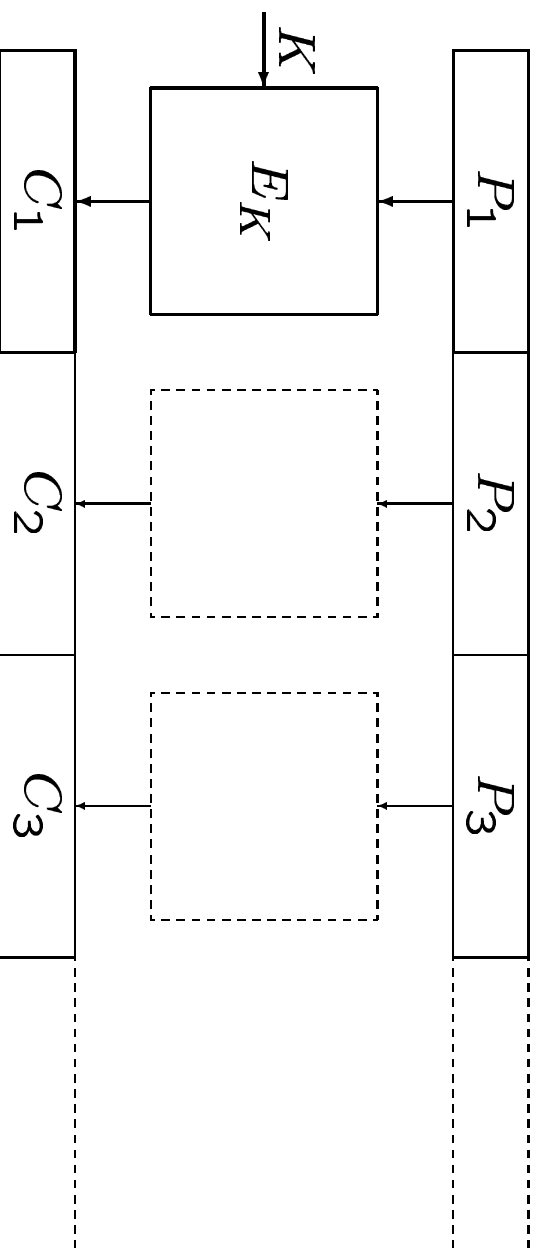
Vincent Rijmen

Dept. ESAT, K.U.Leuven

# Motivations

1. Resistance against attacks:  
differential, linear cryptanalysis
2. Efficiently implementable
3. Simplicity

# Block Cipher Model



Transformation  $E_K$  takes one block at a time,  
is identical for all blocks

# Design

$$E : \{0, 1\}^n \times \{0, 1\}^k \rightarrow \{0, 1\}^n$$

$$E_K : \{0, 1\}^n \rightarrow \{0, 1\}^n$$

Block cipher: family of permutations in the block space

Every key selects one permutation

Block length  $n \Rightarrow 2^n!$   $\approx 2^{(n-1)2^n}$  permutations

Key length  $k \Rightarrow 2^k$  selectable permutations

# What is Security

*(Alice in Wonderland)*

...

'But that's not security,' said Alice, 'security means something else.'

'Security means what I choose it to mean,' said the Queen.

# Taxonomy of Approaches

1. Information theoretic: unconditional security  
Problem solved !  
The Vernam scheme (1917) is the best we can do.
2. Complexity theoretic: reduce security to hard problems  
Instance problem
3. Probabilistic: unconditional security, but with probability of failure
4. Cryptanalytical: secure against state-of-the-art cryptanalysis

# Block Cipher Design Goals

Provide (at least) cryptanalytical security

At a low cost (key length, performance, chip area, ... )

## Practical Security: the Theory

Given  $(P_1, C_1), (P_2, C_2), (P_3, C_3), \dots$ , it should be 'difficult'

1. to recover the used key
2. to predict other  $(P_i, C_i)$  pairs
3. to characterize the family of  $2^k$  permutations, or distinguish them from the other permutations

Important design criteria (Shannon)

1. Nonlinearity (in plaintext and in key input) ('confusion')
2. Spreading of statistics over all bits ('diffusion')

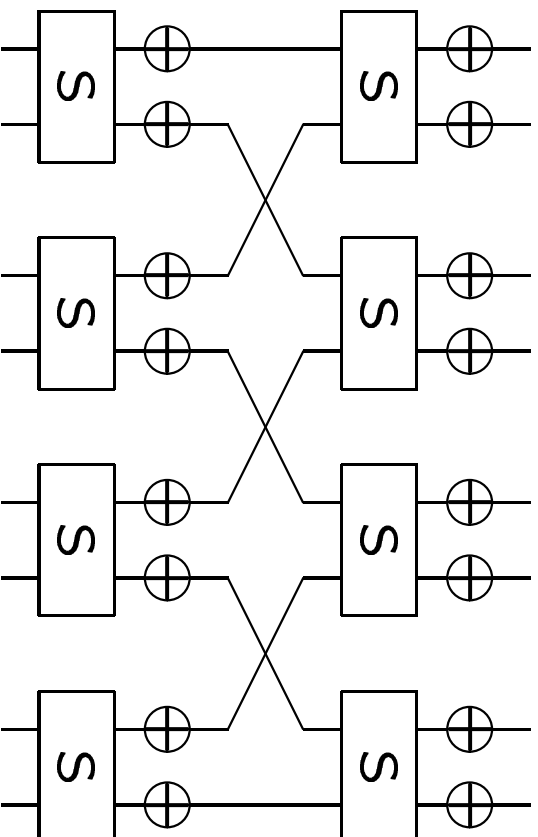
## Practical Security: Practical Constraints

Block cipher should be realizable  
in hardware and/or software

It should be possible to change the key easily

⇒ a designer has to use relatively small components, with  
a limited number of input bits, output bits

# Iterated Block Ciphers



Small code size, hardware area (important ??)

Iteration of a (weak) round transformation can make a strong cipher

# Rijndael structure

10/12/14 iterations of the round transformation

uniform and parallel round transformation, composed of:

- byte substitution
- shift rows
- mix columns
- round key addition

# Data Representation

Input: byte string

$p_0 p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12} p_{13} p_{14} p_{15}$

Key: byte string

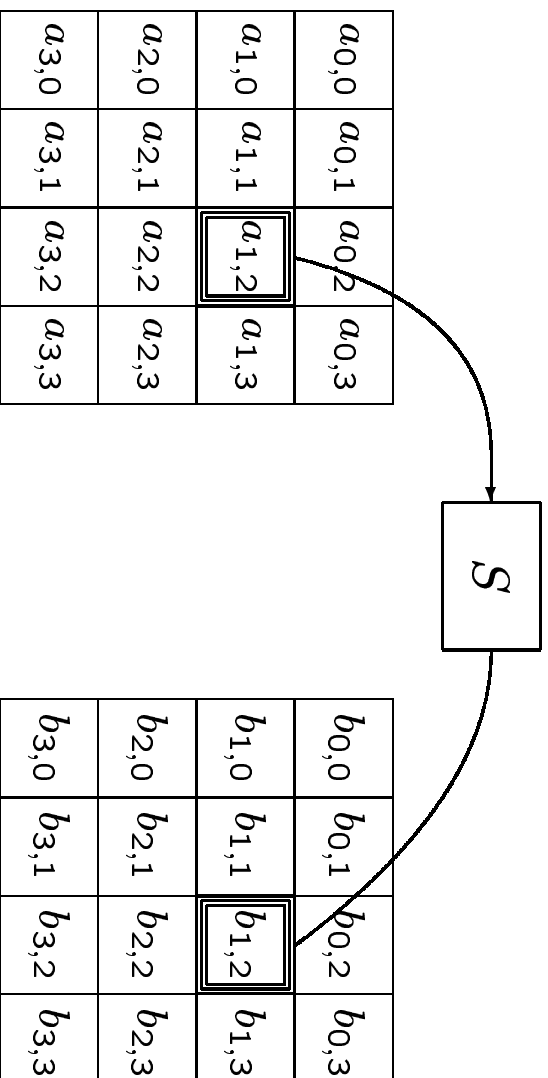
$k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7 k_8 k_9 k_{10} k_{11} k_{12} k_{13} k_{14} k_{15} k_{16} k_{17} k_{18} k_{19} k_{20} k_{21} k_{22} k_{23}$

Representation: rectangular byte arrays

$p_0$	$p_4$	$p_8$	$p_{12}$
$p_1$	$p_5$	$p_9$	$p_{13}$
$p_2$	$p_6$	$p_{10}$	$p_{14}$
$p_3$	$p_7$	$p_{11}$	$p_{15}$

$k_0$	$k_4$	$k_8$	$k_{12}$	$k_{16}$	$k_{20}$
$k_1$	$k_5$	$k_9$	$k_{13}$	$k_{17}$	$k_{21}$
$k_2$	$k_6$	$k_{10}$	$k_{14}$	$k_{18}$	$k_{22}$
$k_3$	$k_7$	$k_{11}$	$k_{15}$	$k_{19}$	$k_{23}$

# Byte Substitution

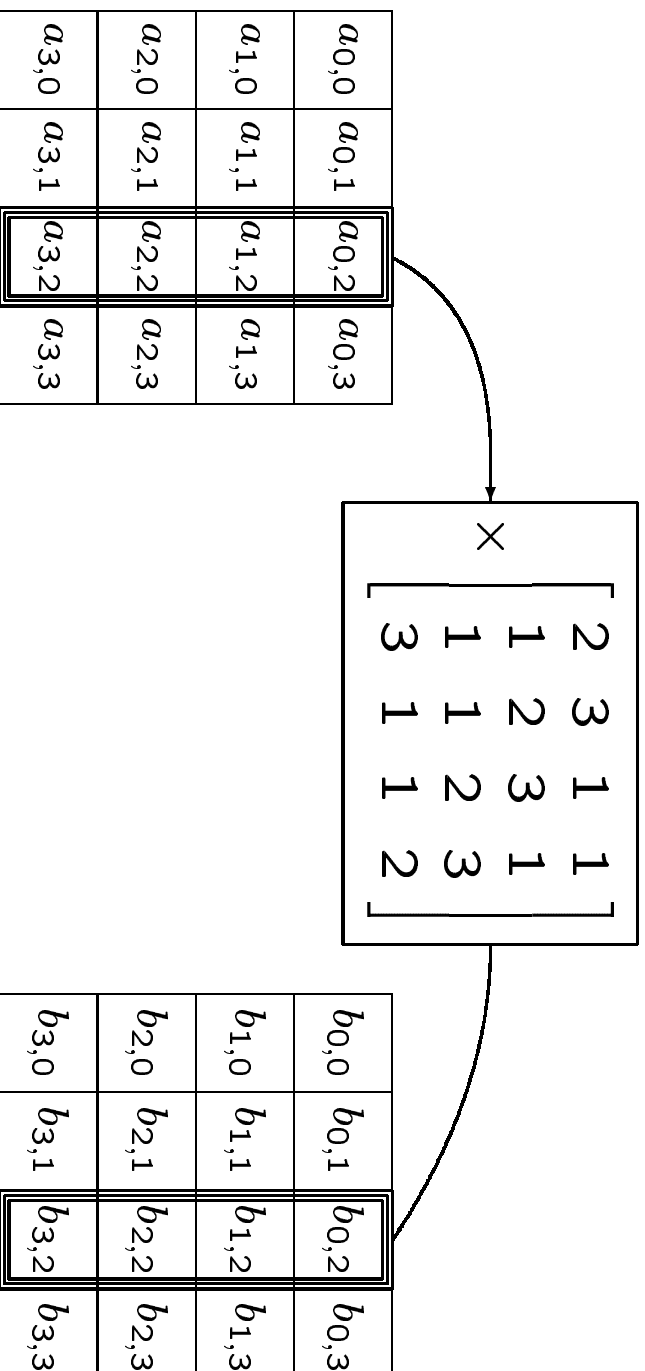


Transformation is byte by byte

Invertible S-box

1 S-box for the whole cipher (simplicity)

# Mix Columns



Bytes in columns are combined linearly

Good diffusion properties

Based on Maximal Distance codes

# Mathematical interludium

Modular integer arithmetic:

- fast on many processors
- slow/expensive in hardware and small processors (carry)
- not every operation is invertible (overflow)

Finite field operations:

- more mathematical structure (always invertible)
- reasonably fast on most platforms
- except mathematicians, nobody likes it
- linear codes

# Finite fields $\text{GF}(p)$

$\langle \mathbb{Z}, + \rangle$  is a group

$\langle \mathbb{Z}, + \bmod n \rangle$  is a group

$\langle \mathbb{Z}_0, \times \bmod p \rangle$  is a group, iff  $p$  prime

Example:  $p = 7,$

$\times$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

$\langle \mathbb{Z}, + \bmod p, \times \bmod p \rangle$  is a finite field, iff  $p$  prime

## Finite fields, $\text{GF}(p^n)$

polynomials of degree  $n - 1$  with coefficients in  $\text{GF}(p)$   
 $p = 2$ : convenient on computers

Example:  $0x57 = 01010111 \rightarrow x^6 + x^4 + x^2 + x + 1$

Addition: XOR

Example:

$$\begin{aligned} 0x57 + 0x83 &= (x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) \\ &= x^7 + x^6 + x^4 + x^2 \\ &= 0xD4 \end{aligned}$$

# Finite fields, $\text{GF}(p^n)$

Multiplication:

modulo an irreducible polynomial  $m(x)$  of degree  $n$

Example:

$$\begin{aligned} & (x^7 + x + 1) \times (x^6 + x^4 + x^2 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1) \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod (x^8 + x^4 + x^3 + x + 1) \\ &= x^7 + x^6 + 1 \end{aligned}$$

End-of-mathematics

# Shift Rows

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>m</i>	<i>n</i>	<i>o</i>	<i>p</i>



<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>f</i>	<i>g</i>	<i>h</i>	<i>e</i>
<i>k</i>	<i>l</i>	<i>i</i>	<i>j</i>
<i>p</i>	<i>m</i>	<i>n</i>	<i>o</i>

Rows shifted over different offsets  
Diffusion over the columns

# Round key addition

$$\begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ \hline k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ \hline k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ \hline k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \\ \hline \end{array}$$

Bit-wise XOR

# Rijndael Key Schedule

Different key scheduling for different key lengths (extendible for non-AES key lengths)

Fast and light, because in many applications, the key scheduling is a bottleneck: short messages, hashing, ...

## Rijndael Key Schedule (2)

$k_0$	$k_1$	$k_2$	$k_3$	$k_4$	$k_5$	$k_6$	$k_7$	$k_8$	$k_9$	$k_{10}$	$k_{11}$	$k_{12}$	$k_{13}$	$k_{14}$	$k_{15}$	...
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	-----

Round key 0	Round key 1	Round key 2	...
-------------	-------------	-------------	-----

$$k_{6n} = k_{6n-6} + f(k_{6n-1})$$

$$k_i = k_{i-6} + k_{i-1}$$

Cipher key expansion can be done just-in-time

No extra storage required

# Modern Cryptanalysis

Differential cryptanalysis: Propagation of differences

$$\begin{array}{l} P_1 \quad \rightarrow \quad C_1 \\ P_1 + \delta \quad \rightarrow \quad C_1 + \epsilon \end{array}$$

Look for special values of  $\epsilon$

Probabilistic attack

Linear transformations:  $\delta$  determines  $\epsilon$  uniquely

Nonlinear transformations: several  $\epsilon$ 's for each  $\delta$

→ unpredictability

# Difference propagation (I)

BytesSub: differences remain contained

*	*		
	*		

BytesSub  
→

*	*		
	*		

Difference byte values: unpredictable

Difference pattern: fixed

## Difference propagation (II)

MixColumn: differences spread in the columns

*	*		
	*		

MixColumn  
→

*	*		
*			
*	*		
*	*		

Difference byte values: predictable

Difference pattern: use of MDS codes guarantees 'wide trails'

## Difference propagation (III)

ShiftRow: spreading over columns

Every '\*' adds unpredictability

The S-box is optimal in this respect (design by K. Nyberg)

Linear components:

Theorem: in every 4 rounds, at least 25 '\*'s

Similar theorem for linear cryptanalysis

# The S-box

K. Nyberg ('94):

Use  $x \rightarrow x^{-1}$  over  $\text{GF}(2^8)$

1. optimal spreading of  $(\delta, \epsilon)$
2. minimal input-output bit correlations
3. maximal nonlinear order of the output
4. 'easy' description

→ Add affine mapping to remediate (4)

# Implementation

Exors and small bit permutations can be implemented quite efficiently

MDS code based mappings require operations in a finite field

Efficient implementation on 32-bit CPU: combine non-linear substitution and mixing layer in 1 set of table look-ups

# Non-Linear Substitution and Mixing

Substitution

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \rightarrow \begin{bmatrix} S[a_0] \\ S[a_1] \\ S[a_2] \\ S[a_3] \end{bmatrix}$$

Mixing

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \rightarrow \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

## Combined Table Look-Up

$b$  = mixing (substitution( $a$ )):

$$\begin{aligned}
 b_0 &= c_{00} \cdot S[a_0] + c_{01} \cdot S[a_1] + c_{02} \cdot S[a_2] + c_{03} \cdot S[a_3] \\
 b_1 &= c_{10} \cdot S[a_0] + c_{11} \cdot S[a_1] + c_{12} \cdot S[a_2] + c_{13} \cdot S[a_3] \\
 b_2 &= c_{20} \cdot S[a_0] + c_{21} \cdot S[a_1] + c_{22} \cdot S[a_2] + c_{23} \cdot S[a_3] \\
 b_3 &= c_{30} \cdot S[a_0] + c_{31} \cdot S[a_1] + c_{32} \cdot S[a_2] + c_{33} \cdot S[a_3]
 \end{aligned}$$

$$T_0[x] = \begin{bmatrix} c_{00} \cdot S[x] \\ c_{10} \cdot S[x] \\ c_{20} \cdot S[x] \\ c_{30} \cdot S[x] \end{bmatrix}, T_1[x] = \begin{bmatrix} c_{01} \cdot S[x] \\ c_{11} \cdot S[x] \\ c_{21} \cdot S[x] \\ c_{31} \cdot S[x] \end{bmatrix}, \dots$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = T_0[a_0] + T_1[a_1] + T_2[a_2] + T_3[a_3]$$

## 8-bit processors

Implementation on processor with small RAM:

No place for the  $T$ -tables

Direct implementation of steps (simple coefficients)

About 1 K of code

Approximately 2 milliseconds (4 MHz. clock)

RAM + registers: 36 bytes

## Rijndael: hardware and multiprocessors

### Hardware:

No carry delays, simple design

Very flexible in area/speed tradeoff

### Multiprocessors:

Large degree of parallelism

5 ALU's at work simultaneously

Memory access is bottleneck

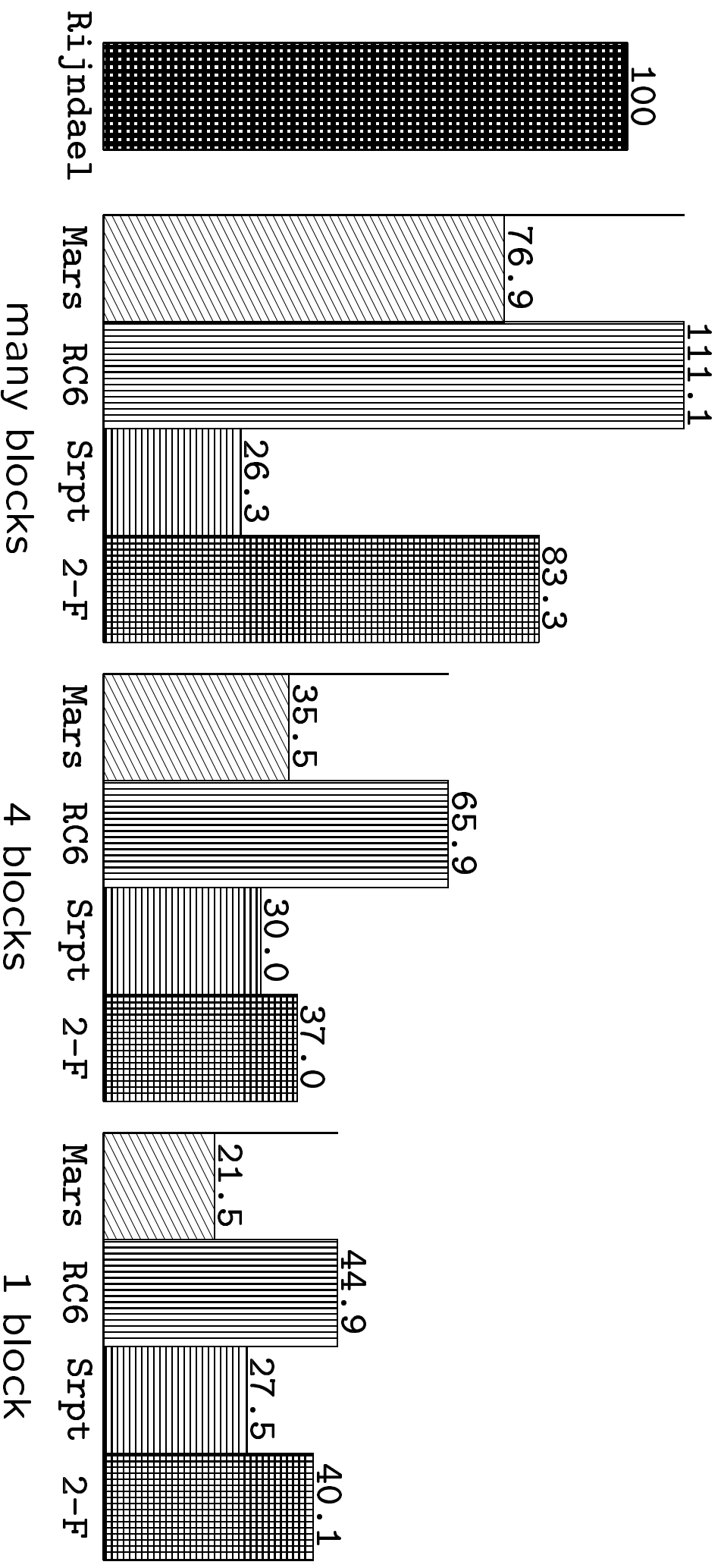
# Performance-Security Trade-off

(Serpent designers:) Security weighted performance

→ Performance-weighted Security

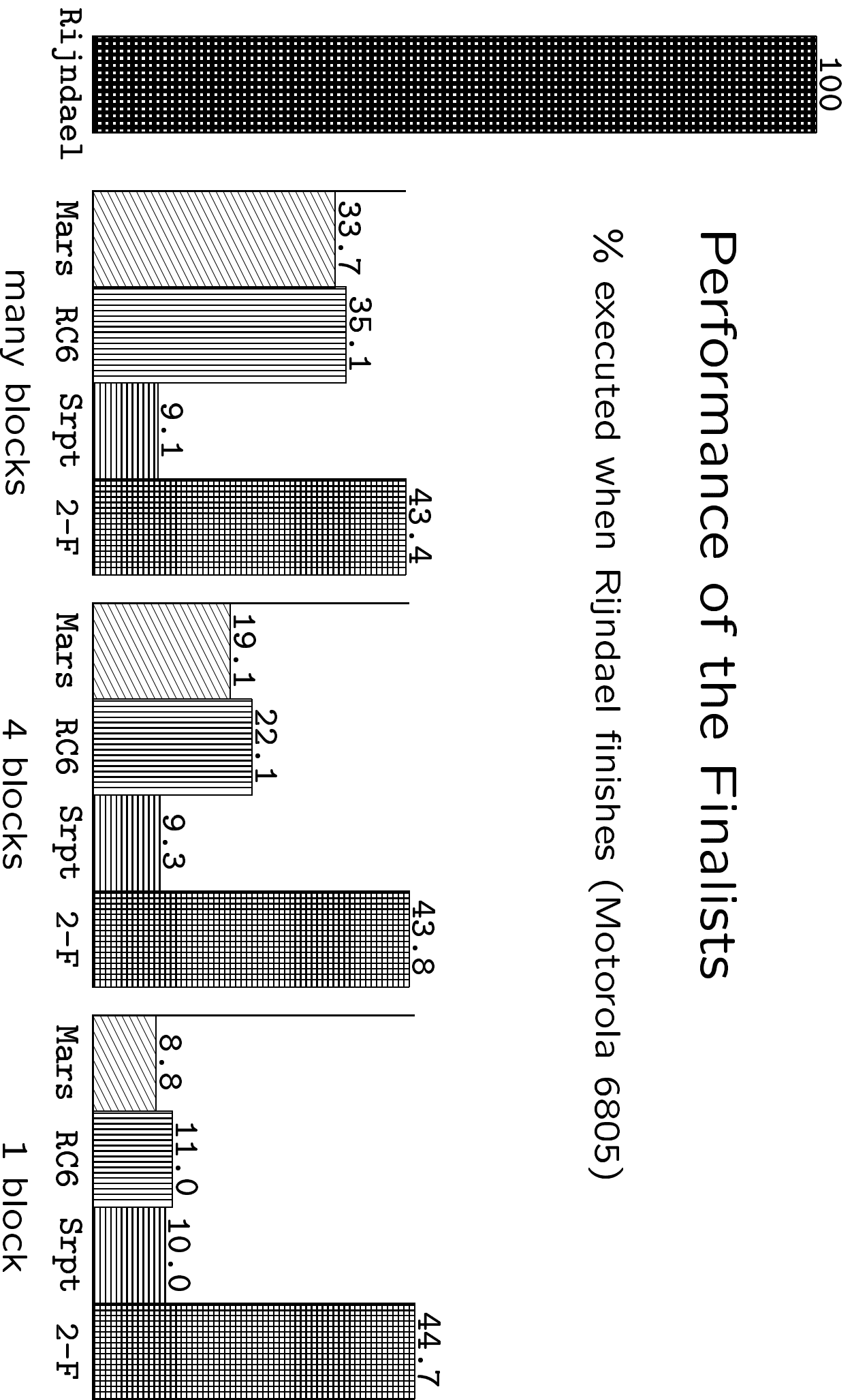
# Performance of the Finalists

% executed when Rijndael finishes (Pentium II/Pro)



# Performance of the Finalists

% executed when Rijndael finishes (Motorola 6805)



## Performance figures of Rijndael

- Pentium III @800 MHz. (assembly):  
232 cycles/encryption, or 412 Mbit/s
- IA-64: 124 cycles/encryption
- Motorola 6805 @4 MHz.:  
9464 cycles/encryption (+ key setup), or 54 kbit/s
- Hardware (estimated) 2.5 Gbit/s