

# ECE 646: Lecture 9

## Advanced Encryption Standard

1

---

---

---

---

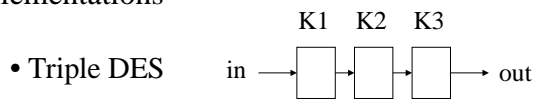
---

---

### Why a new standard?

1. Old standard insecure against brute-force attacks

2. Straightforward fixes lead to inefficient implementations



3. New trends in fast software encryption

- use of basic instructions of the microprocessor

4. New ways of assessing cipher strength

- differential cryptanalysis
- linear cryptanalysis

2

---

---

---

---

---

---

### Why a contest?

- Focus the effort of cryptographic community

Small number of specialists in the open research

- Stimulate the research on methods of constructing secure ciphers
- Avoid backdoor theories
- Speed-up the acceptance of the standard

3

---

---

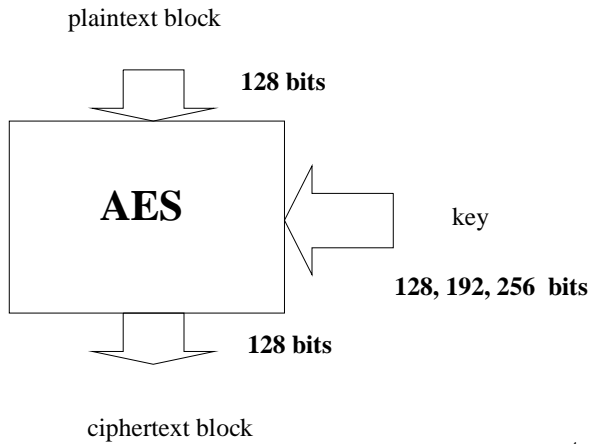
---

---

---

---

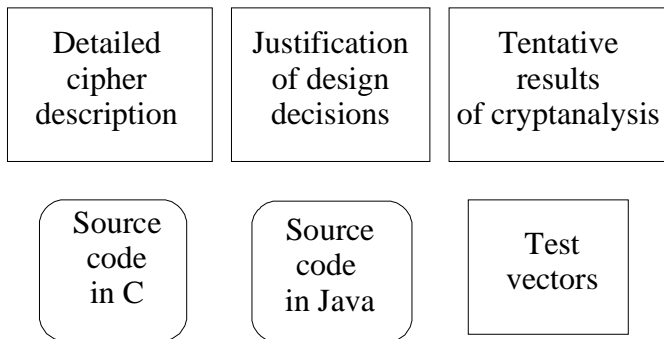
## External format of the AES algorithm



4

## Rules of the contest

*Each team submits*



5

## AES Contest Effort

**June 1998**

**15 Candidates**  
from USA, Canada, Belgium,  
France, Germany, Norway, UK, Isreal,  
Korea, Japan, Australia, Costa Rica

**Round 1**

**Security**  
**Software efficiency**

**August 1999**

**5 final candidates**  
Mars, RC6, Rijndael, Serpent, Twofish

**Round 2**

**Security**  
**Hardware efficiency**

**October 2000**

**1 winner: Rijndael**  
Belgium

6

## AES: Candidate algorithms

### North America (8)

#### Canada:

CAST-256  
Deal

#### USA:

Mars  
RC6  
Twofish  
Safer+  
HPC

#### Costa Rica:

Frog

### Europe (4)

#### Germany:

Magenta

#### Belgium:

Rijndael

#### France:

DFC

#### Israel, UK,

#### Norway:

Serpent

### Asia (2)

#### Korea:

Crypton

#### Japan:

E2

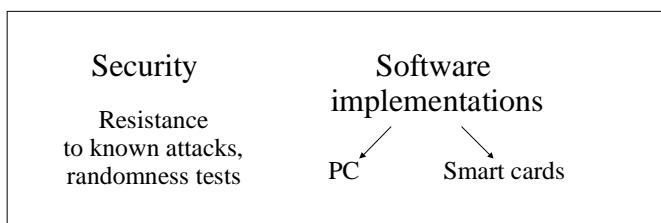
### Australia (1)

#### Australia:

LOKI97

7

## First round June 1998-August 1999



8

## Survey filled by 104 participants of the Second AES Conference in Rome, March 1999

1. Rijndael	+76	
2. RC6	+73	
3. Twofish	+61	Overwhelming YES
4. Mars	+52	
5. Serpent	+45	
6. E2	+14	Mild YES
7. CAST-256	-2	
8. Safer+	-4	Middle-of-the-Road
9. DFC	-5	
10. Crypton	-15	
11. DEAL	-70	Mild NO
12. HPC	-77	
13. Magenta	-83	
14. Loki97	-85	Overwhelming NO
15. Frog	-85	

9

## AES Finalists (1)

### USA

#### Mars - IBM

C. Burwick, D. Coppersmith, E. D'Avignon,  
R. Gennaro, S. Halevi, C. Jutla, S. M. Matyas,  
L. O'Connor, M. Peyravian, D. Safford,  
N. Zunic

#### RC6 - RSA Data Security, Inc.

R. Rivest - MIT  
M. Robshaw, R. Sidney, Y. L. Yin - RSA

#### Twofish - Counterpane Systems

B. Schneier, J. Kelsey, C. Hall, N. Ferguson  
- Counterpane, D. Whiting - Hi/fn,  
D. Wagner - Berkeley

10

---

---

---

---

---

---

## AES Finalists (2)

### Europe

**Rijndael** - J. Daemen, V. Rijmen  
**Katholieke Universiteit Leuven**  
Belgium

**Serpent** - R. Anderson, **Cambridge, England**  
E. Biham - **Technion, Israel**  
L. Knudsen, **University of Bergen, Norway**

11

---

---

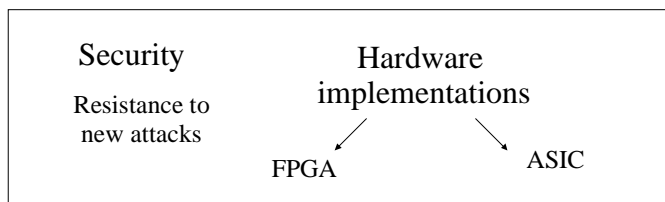
---

---

---

---

## Second round August 1999-August 2000



12

---

---

---

---

---

---

## AES contest: Second Round

- 13-14 April 2000  
3rd AES Conference in New York
- 15 May 2000  
End of the comment period for Round II
- **2 October 2000 Winner announced**
- November 2001 FIPS-197: AES announced
- May 2002 Standard becomes effective

13

## How NIST has made a final decision?

**BASIC CRITERIA =**

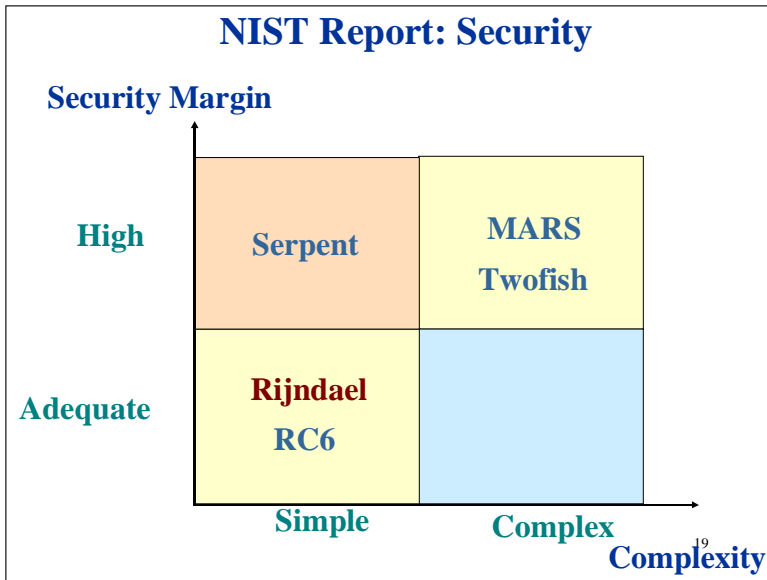
security  
software efficiency  
hardware efficiency  
flexibility

14

**Security**

15






---

---

---

---

---

---

---

---

## Efficiency - What's more important: software or hardware?

---

---

---

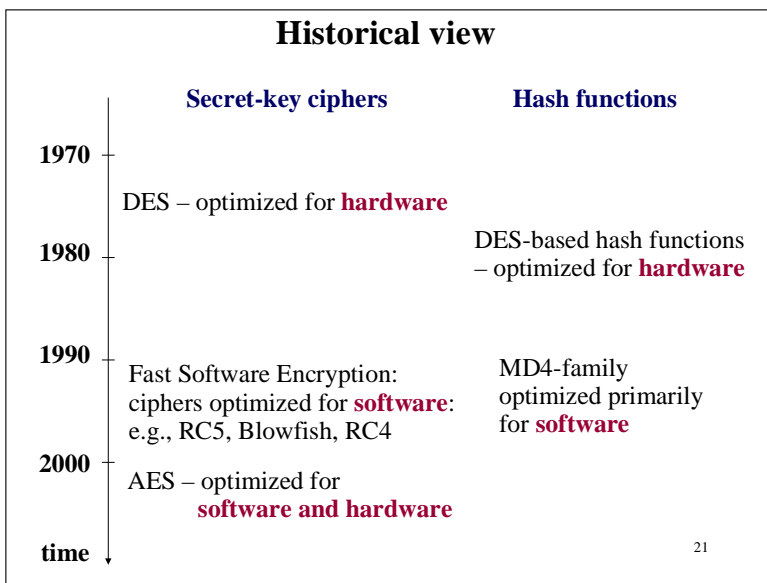
---

---

---

---

---




---

---

---

---

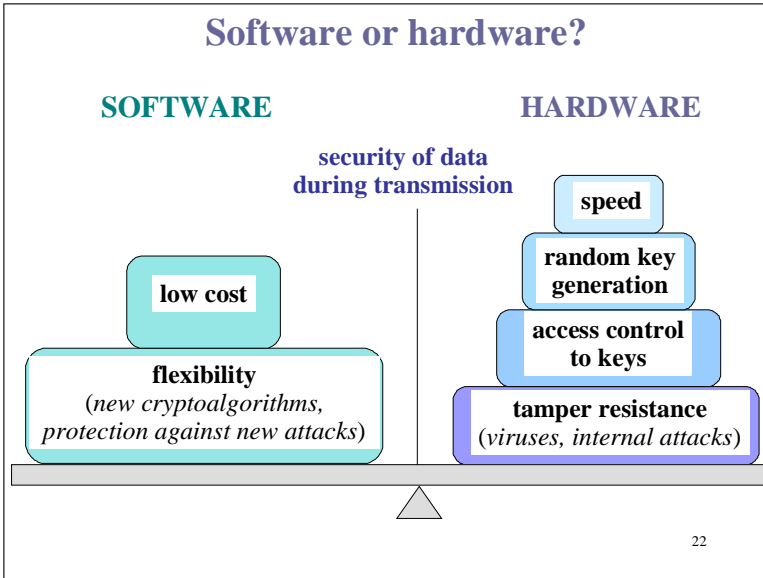
---

---

---

---

## Software or hardware?



---

---

---

---

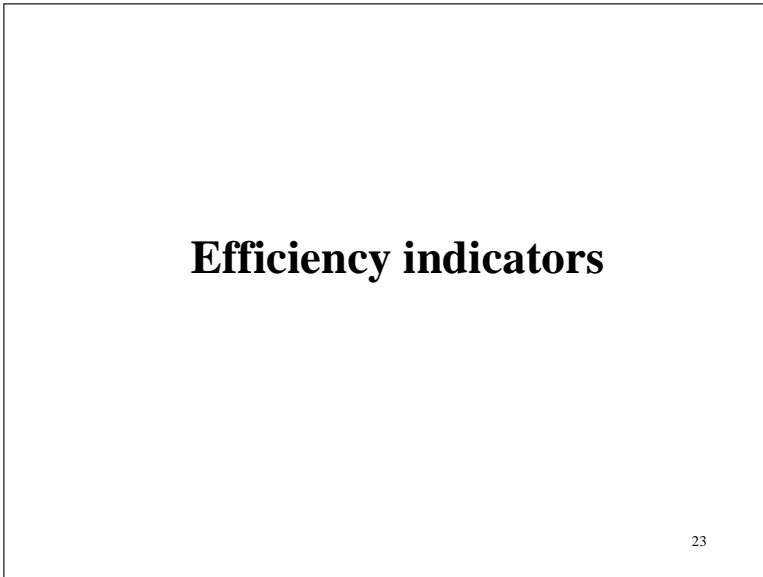
---

---

---

---

## Efficiency indicators



---

---

---

---

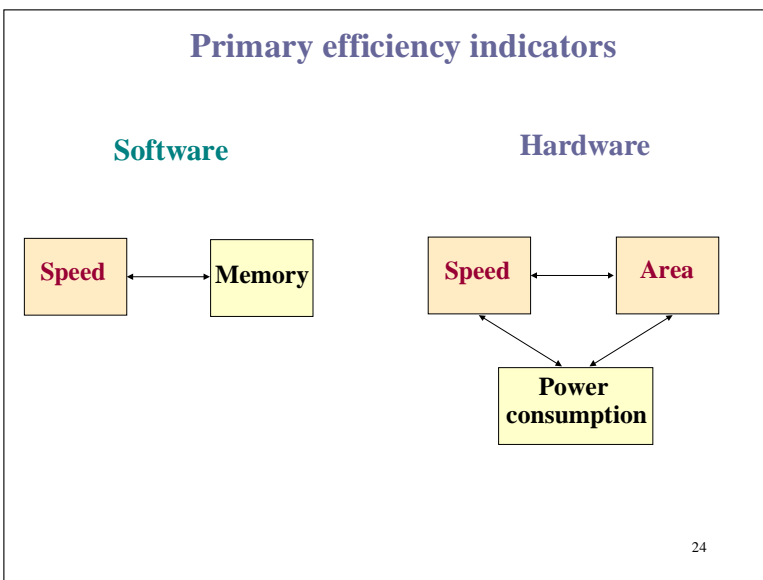
---

---

---

---

## Primary efficiency indicators



---

---

---

---

---

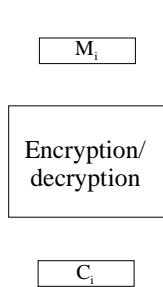
---

---

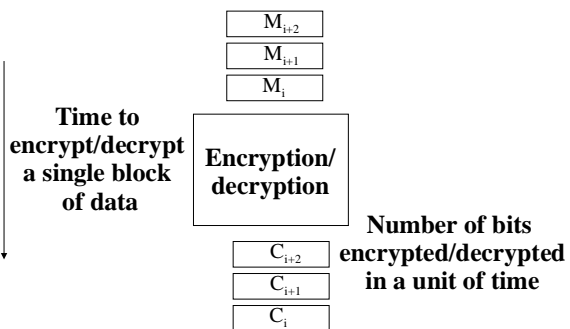
---

## Efficiency parameters

### Latency



### Throughput = Speed



$$\text{Throughput} = \frac{\text{Block\_size} \cdot \text{Number\_of\_blocks\_processed\_simultaneously}}{\text{Latency}}$$

25

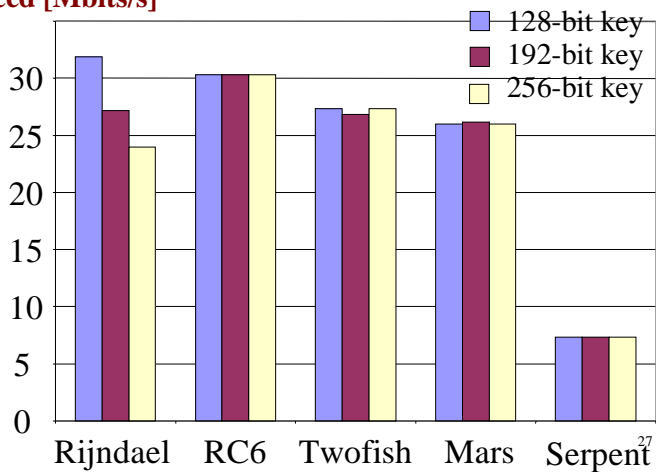
## Efficiency in software

26

### Efficiency in software: Code submitted by authors

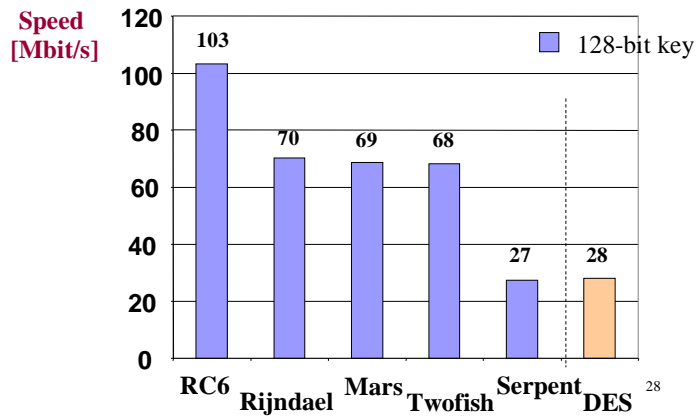
200 MHz Pentium Pro, Borland C++

Speed [Mbits/s]



## Mean encryption & decryption in software Code developed by Brian Gladman, UK

200 MHz Pentium Pro, Microsoft Visual C++, ver. 6




---

---

---

---

---

---

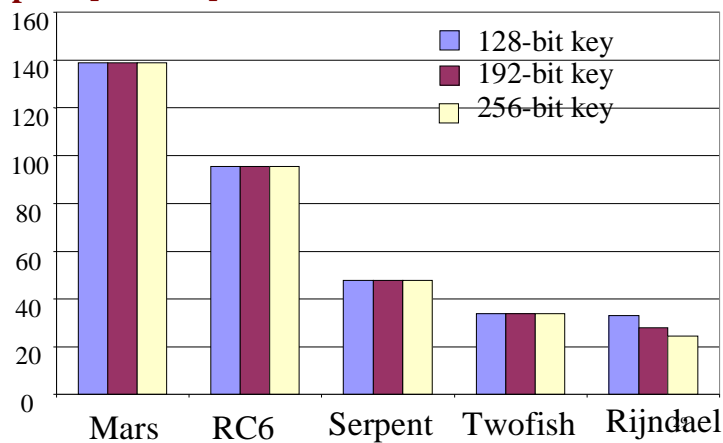
---

---

## Efficiency in software: NIST tests

450 MHz Pentium II, DJGPP gcc

Speed [Mbits/s]




---

---

---

---

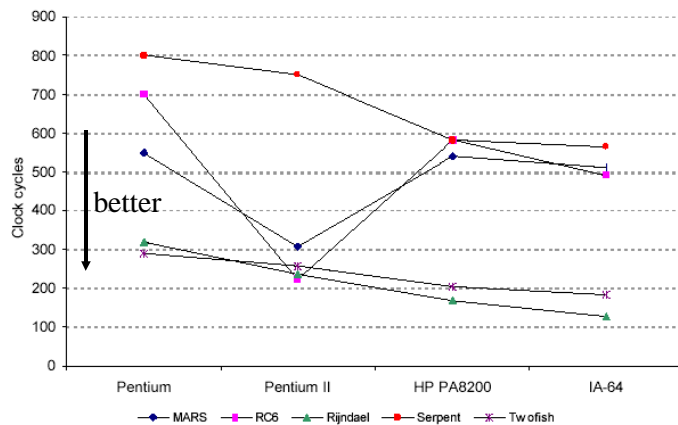
---

---

---

---

## Encryption time in clock cycles on various platforms Twofish team: Bruce Schneier & Doug Whiting




---

---

---

---

---

---

---

---

### Efficiency in software: Ranking of encryption speeds for various platforms

	Intel						Alpha			Sun-Sparc	H-P
Mars	4	4	2	4	3	2	3	4	4	3	3
RC6	<b>1</b>	3	<b>1</b>	<b>1</b>	4	<b>1</b>	4	3	3	5	4
Twofish	2	<b>1</b>	3	2	<b>1</b>	4	2	2	2	2	2
Rijndael	3	2	4	3	2	3	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
Serpent	5	5	5	5	5	5	5	5	5	4	5

---

---

---

---

---

---

---

---

### NIST Report: Software Efficiency

#### Encryption and Decryption Speed

	32-bit processors	64-bit processors	DSPs
high	RC6	Rijndael Twofish	Rijndael Twofish
medium	Rijndael Mars Twofish	Mars RC6	Mars RC6
low	Serpent	Serpent	Serpent

---

---

---

---

---

---

---

---

### NIST Report: Software Efficiency

#### Encryption and decryption speed in software on smart cards

	8-bit processors	32-bit processors
high	Rijndael	Rijndael RC6
medium	RC6 Mars Twofish	Mars
low	Serpent	Twofish Serpent

---

---

---

---

---

---

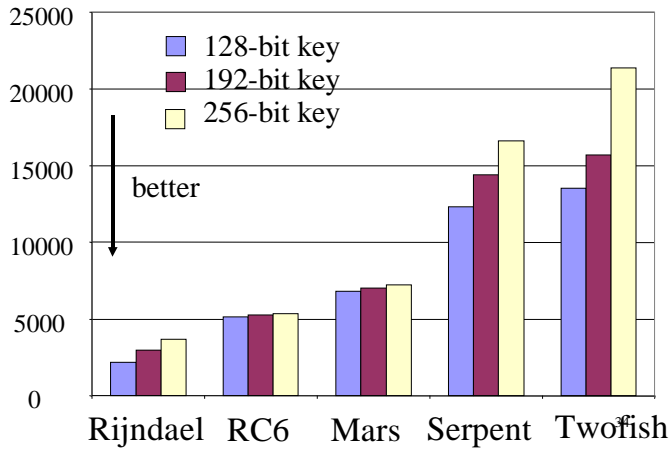
---

---

### Efficiency in software: Key setup, author codes

200 MHz Pentium Pro, Borland C++

Time [clock cycles]




---

---

---

---

---

---

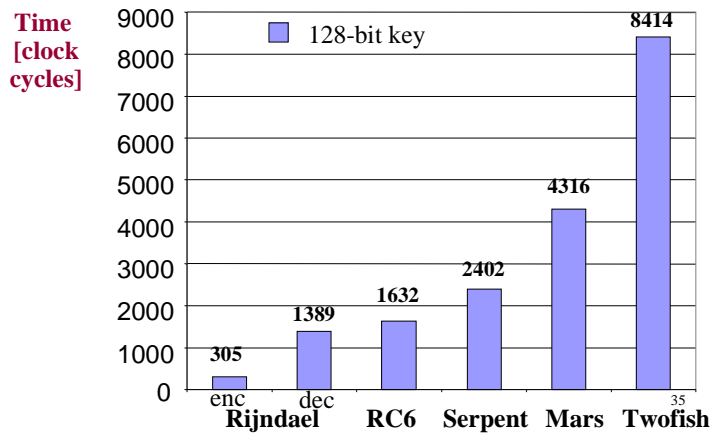
---

---

### Key set-up time in software

Code developed by Brian Gladman, UK

200 MHz Pentium Pro, Microsoft Visual C++, ver. 6




---

---

---

---

---

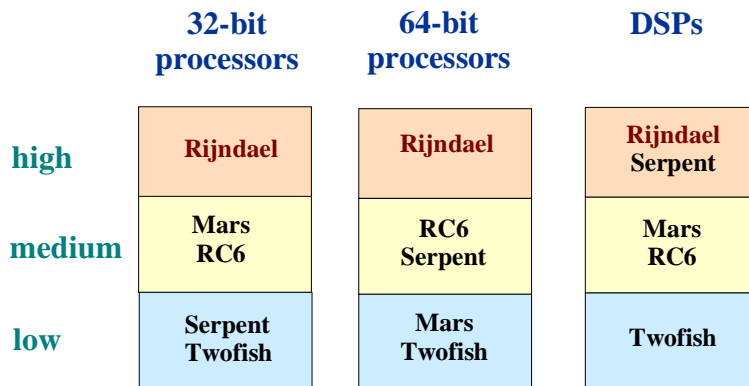
---

---

---

### NIST Report: Software Efficiency

Key scheduling




---

---

---

---

---

---

---

---

## NIST Report: Software Efficiency

### Key scheduling on smart cards

8-bit  
processors

high

Rijndael

medium

Mars  
Twofish

low

RC6  
Serpent

37

---

---

---

---

---

---

## Efficiency in software

### Strong dependence on:

1. Instruction set architecture  
(e.g., variable rotations)
2. Programming language  
(assembler, C, Java)
3. Compiler
4. Programming style

38

---

---

---

---

---

---

## Efficiency in software: Conclusions

### Encryption/decryption

Strong variation of results

Serpent the worst for majority of platforms

### Key setup

Moderate variation of results

Rijndael and RC6 the best for majority of platforms

Twofish and Serpent the worst for majority of  
platforms

39

---

---

---

---

---

---

## Efficiency in hardware

40

### Primary ways of implementing cryptography in hardware

#### ASIC Application Specific Integrated Circuit

- designs must be sent for expensive and time consuming **fabrication** in semiconductor foundry
- designed all the way from behavioral description to **physical layout**

#### FPGA Field Programmable Gate Array

- bought **off the shelf** and reconfigured by designers themselves
- no physical layout design; design ends with a **bitstream** used to configure a device

41

### Which way to go?

#### ASICs

High performance

Low power

Low cost (but only in high volumes)

#### FPGAs

Off-the-shelf

Low development costs

Short time to the market

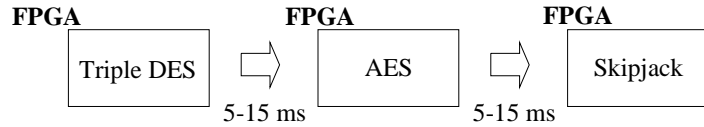
Reconfigurability

42

## Capabilities of reconfiguration (1)

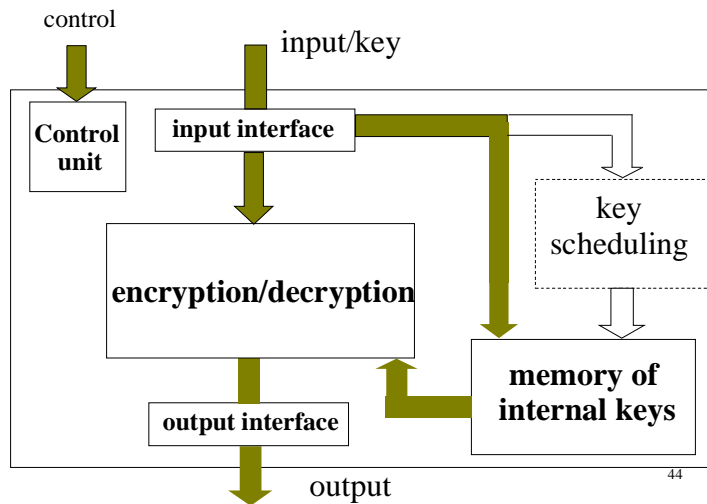
*External ROM and microprocessor enables changing the FPGA function in several milliseconds*

### Various algorithms



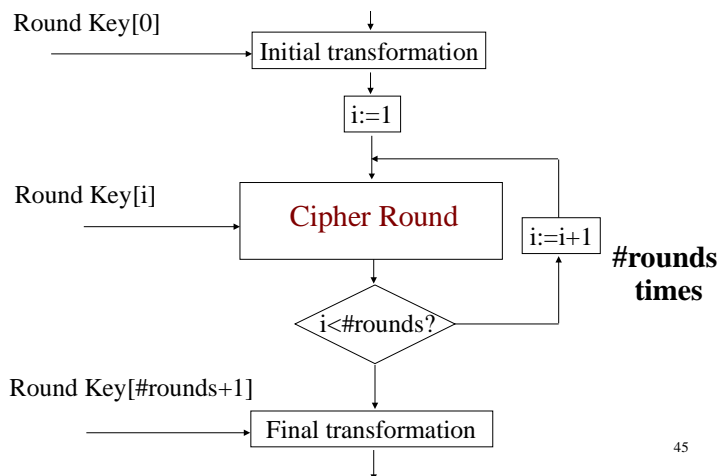
43

### Top level block diagram



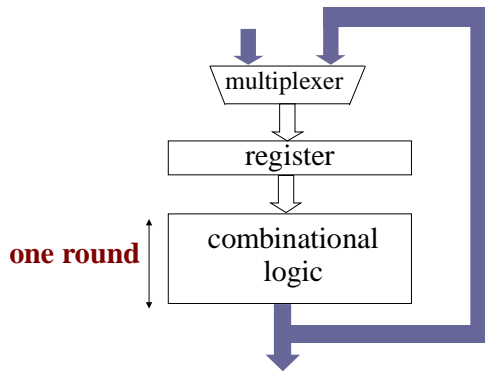
44

### Typical Flow Diagram of a Secret-Key Block Cipher



45

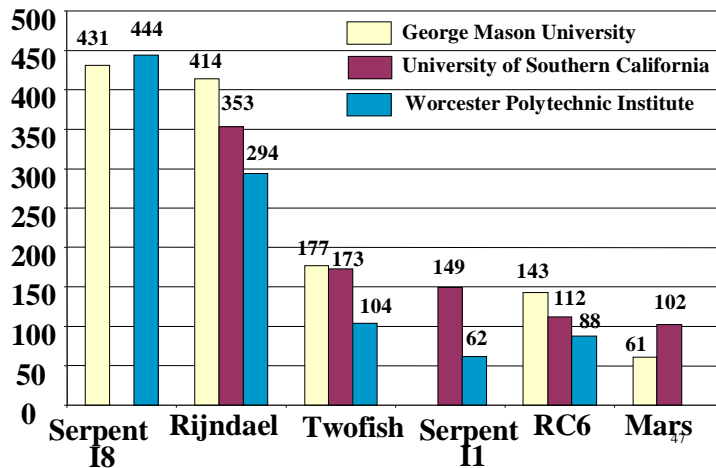
## Basic iterative architecture



46

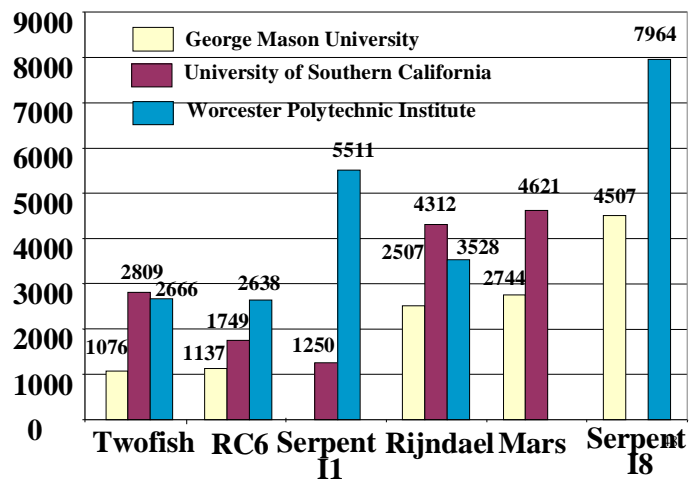
## Efficiency in hardware: FPGA Virtex 1000: Speed

### Throughput [Mbit/s]

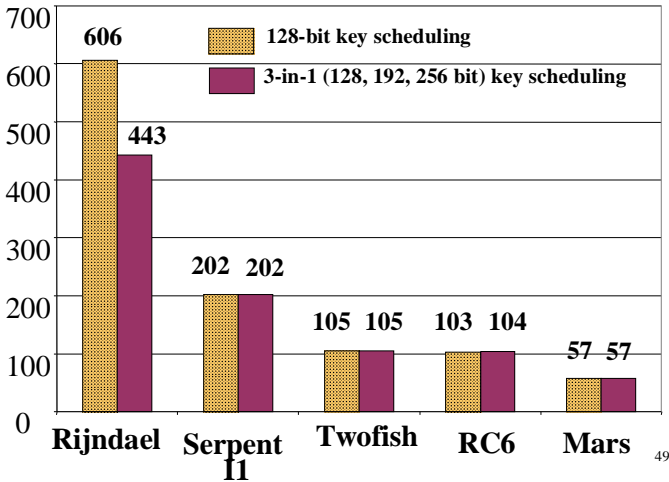


## Efficiency in hardware: FPGA Virtex 1000: Area

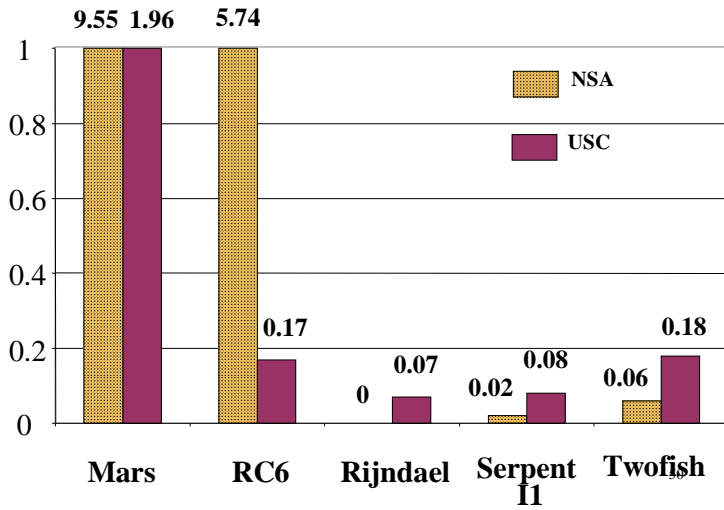
### Area [CLB slices]



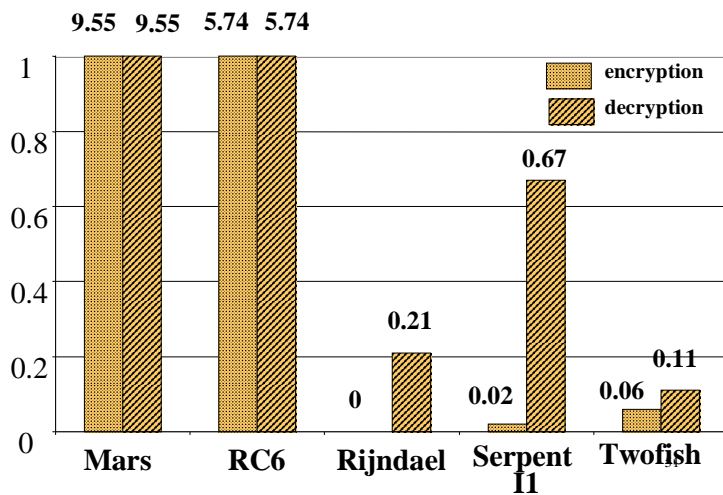
### ASIC implementations: NSA group



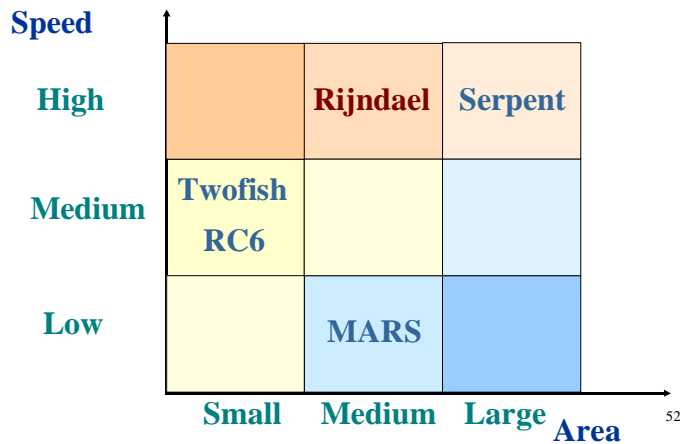
### Encryption Key Setup Latency [ $\mu$ s]



### Encryption vs. Decryption Key Setup Latency [ $\mu$ s]



**NIST Report + GMU Report:  
Hardware Efficiency**  
Feedback cipher modes: CBC, CFB




---

---

---

---

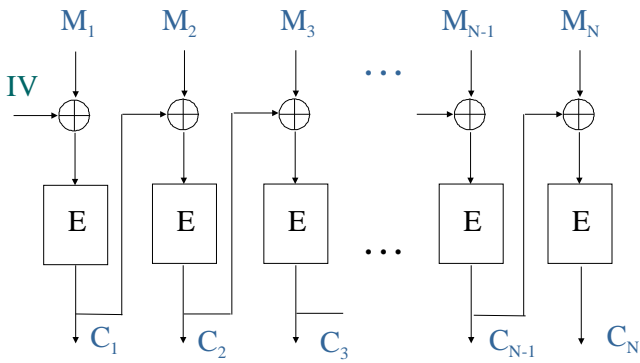
---

---

---

---

**Feedback cipher modes - CBC**



$$C_1 = \text{AES}(M_1 \oplus IV)$$

$$C_i = \text{AES}(M_i \oplus C_{i-1}) \quad \text{for } i=2..N$$

53

---

---

---

---

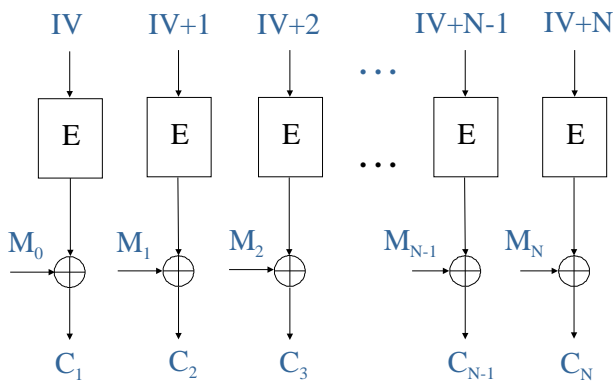
---

---

---

---

**Non-feedback Counter Mode - CTR**



$$C_i = M_i \oplus \text{AES}(IV+i) \quad \text{for } i=0..N$$

54

---

---

---

---

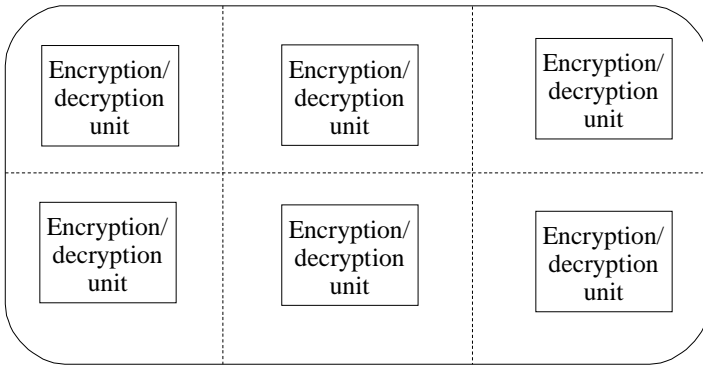
---

---

---

---

## Increasing speed by parallel processing



55

---

---

---

---

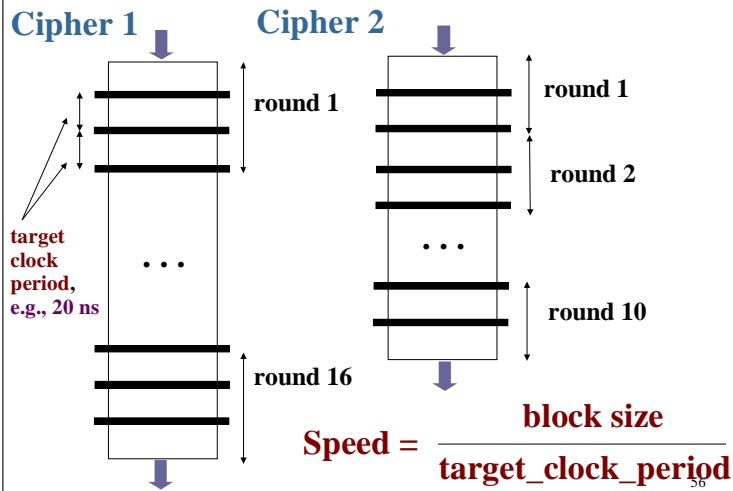
---

---

---

---

## Increasing speed using pipelining




---

---

---

---

---

---

---

---

## Pipelined operation of the encryption unit

clock cycle	1	2	3	4	5	6	7	8
	B1	B2	B3	B4	B5	B6	B7	B8
		B1	B2	B3	B4	B5	B6	B7
			B1	B2	B3	B4	B5	B6
				B1	B2	B3	B4	B5
clock cycle	9	10	11	12	13	14	15	16
	B9	B10	B11	B12	B13	B14	B15	B16
	B8	B9	B10	B3	B4	B5	B6	B7
	B7	B8	B9	B2	B3	B4	B5	B6
	B6	B7	B8	B9	B10	B11	B12	B13

57

---

---

---

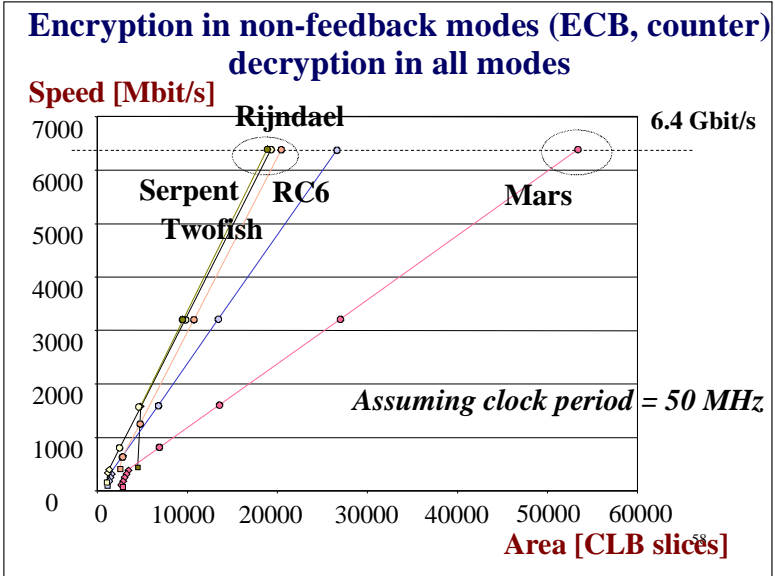
---

---

---

---

---




---

---

---

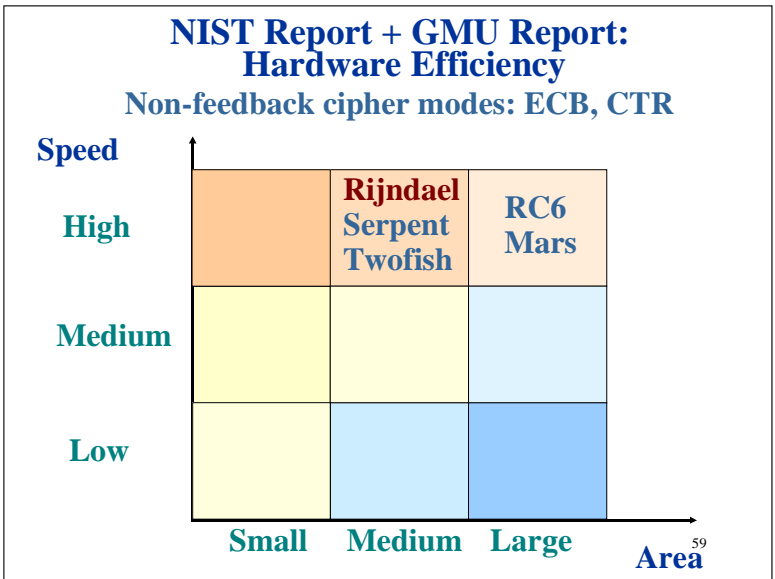
---

---

---

---

---




---

---

---

---

---

---

---

---

**Flexibility**

---

---

---

---

---

---

---

---

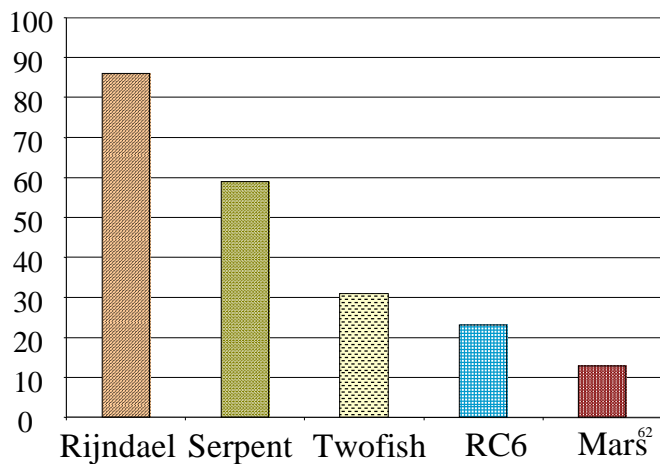
## Flexibility: Criteria

- Additional key-sizes and block-sizes
- Ability to function efficiently and securely in a wide variety of platforms and applications
  - low-end smartcards, wireless - memory requirements
  - IPSec, ATM - key setup time in hardware
  - B-ISDN, satellite communication - encryption speed

61

## Survey filled by 167 participants of the Third AES Conference, April 2000

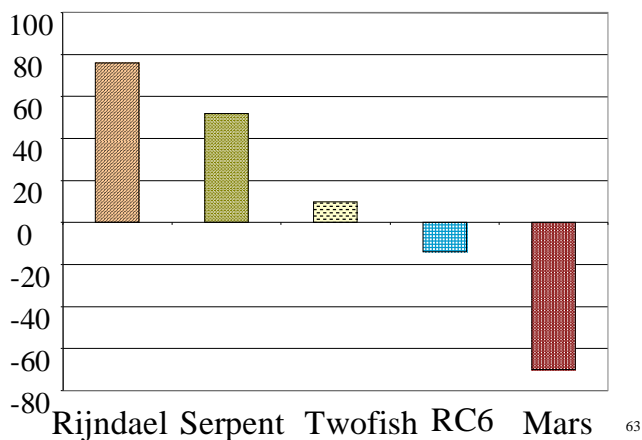
# votes



## Ranking by participants of the AES3 Conference

# votes

Positive votes – negative votes



63

### Leading candidates (1)

#### Rijndael

+

- fastest in hardware
- close to the fastest in software
- very high flexibility

—

- security margin

novel ideas

64

---

---

---

---

---

### Leading candidates (2)

#### Serpent

+

- large security margin
- conservative construction
- very fast in hardware
- cryptanalytical reputation of authors

—

- slow in software
- moderate flexibility

65

---

---

---

---

---

### Leading candidates (3)

#### Twofish

+

- good security margin
  - fast encryption/decryption in software
- 
- American
  - strongly advertized

—

- moderately fast in hardware
- slow key setup in software
- moderate flexibility

66

---

---

---

---

---

## Major operations

67

---



---



---



---



---

### Major operations of AES finalists

	Serpent	Rijndael	Twofish	RC6	Mars
S-boxes					
Multiplication in $GF(2^m)$					
Integer multiplication					
Variable rotation					

68

---



---



---



---



---

### Auxiliary operations of AES finalists

	Serpent	Twofish	Rijndael	RC6	Mars
Boolean					
Fixed rotation					
Addition/subtraction					
Permutation					

69

---



---



---



---



---

# Types of ciphers

70

---

---

---

---

---

---

## AES: Types of candidate algorithms

### Feistel Networks

Twofish      Deal  
E2            LOKI97  
DFC          Magenta

### Modified Feistel Network

RC6  
MARS  
CAST-256

### Substitution-Linear Transformation Networks

Rijndael      Safer+  
Serpent        Crypton

### Others

Frog  
HPC

71

---

---

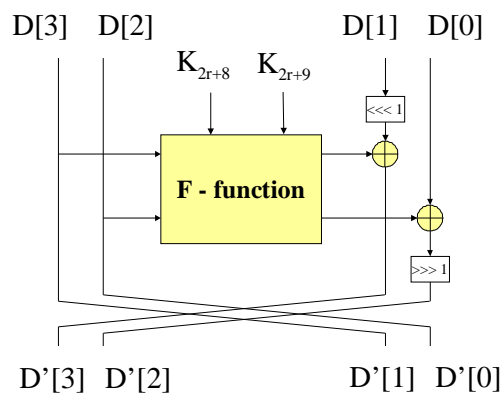
---

---

---

---

## Feistel Network: Single Round of Twofish



- units shared between encryption and decryption 72

---

---

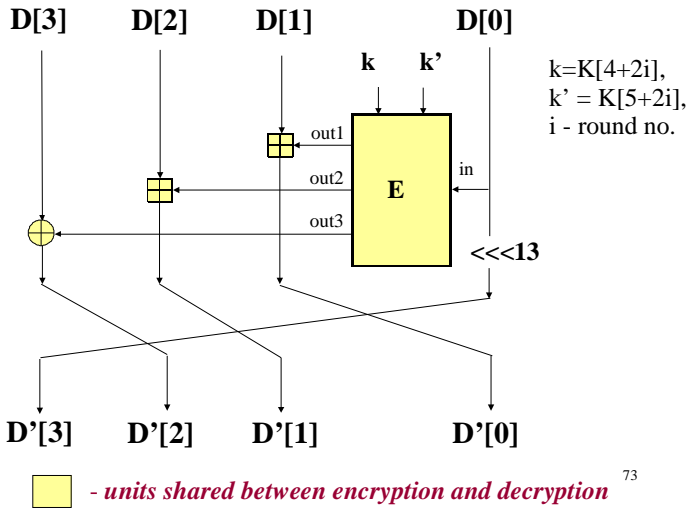
---

---

---

---

### Modified Feistel Network: Single Round of MARS




---

---

---

---

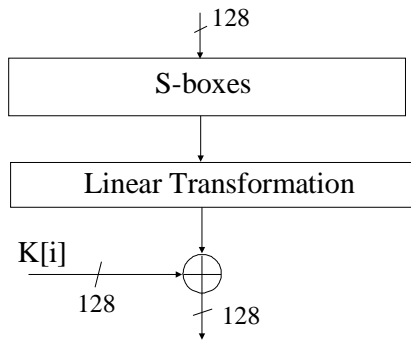
---

---

---

---

### Substitution-Linear Transformation Network: Single Round of Serpent



- units shared between encryption and decryption<sup>4</sup>

---

---

---

---

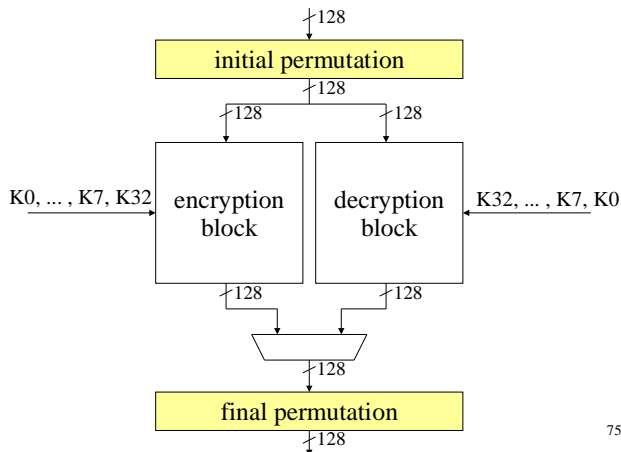
---

---

---

---

### Substitution-Linear Transformation Network: Serpent in Hardware



75

---

---

---

---

---

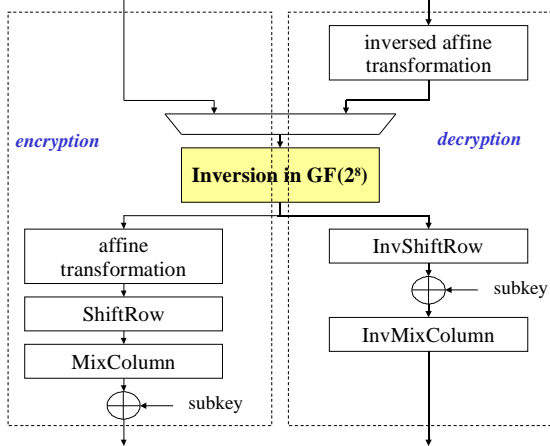
---

---

---

## Substitution-Linear Transformation Network: Rijndael in Hardware

■ - units shared between encryption and decryption

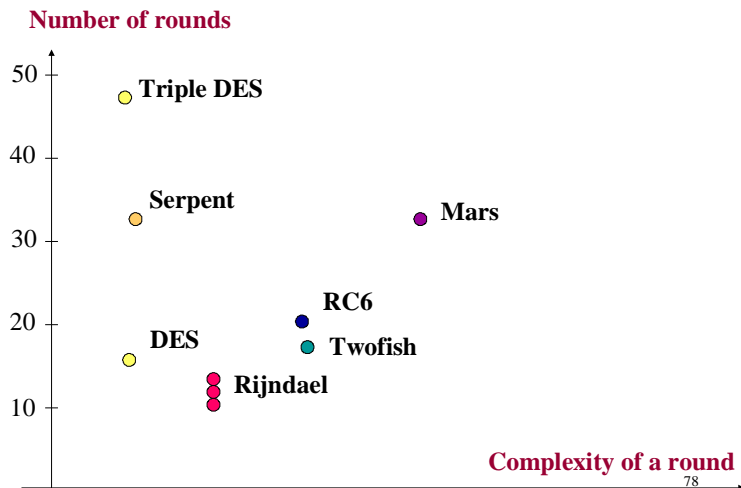


76

## Number and complexity of rounds

77

## Number vs. complexity of a round



78

# Rijndael

79

---

---

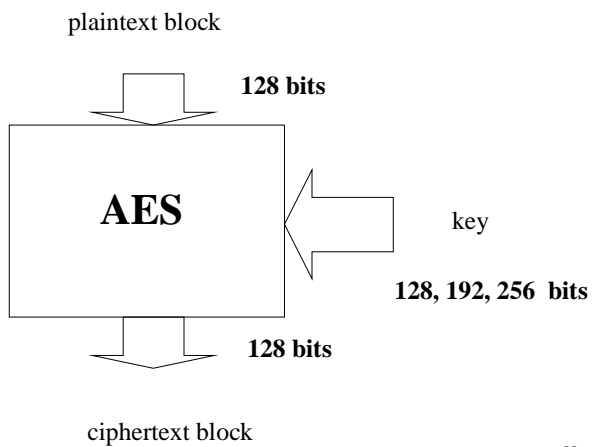
---

---

---

---

## External format of the AES algorithm



80

---

---

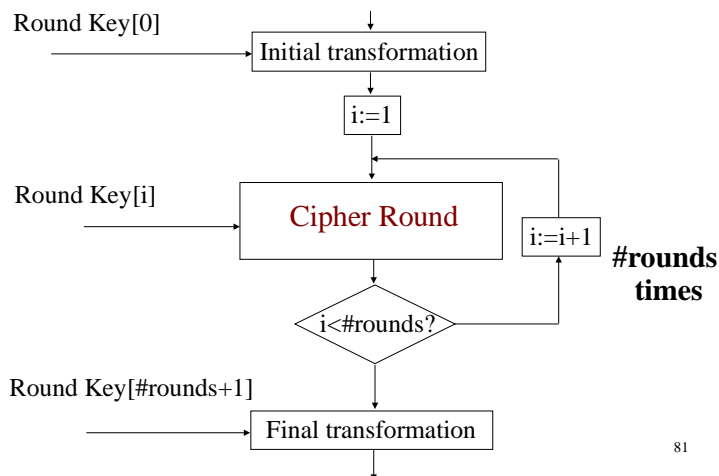
---

---

---

---

## Iterative cipher



81

---

---

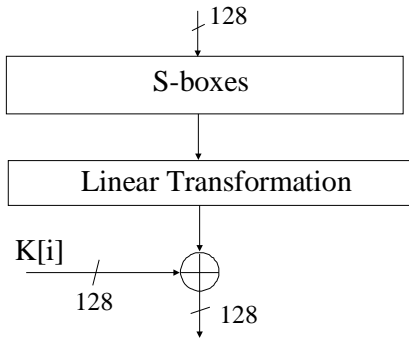
---

---

---

---

## One round of a Substitution-Linear Transformation Network cipher



82

---

---

---

---

---

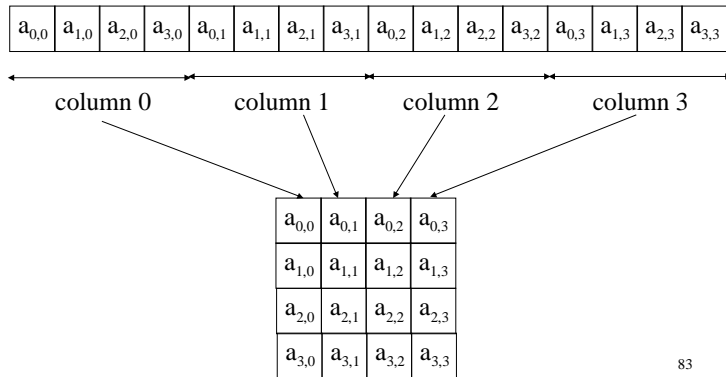
---

---

---

## Input, internal state, and output

**128 bits = 16 bytes**



83

---

---

---

---

---

---

---

---

## Order of bits within a byte

Input bit sequence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...
Byte number	0							1							2							...			
Bit numbers in byte	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	...

Figure 2. Indices for Bytes and Bits.

84

---

---

---

---

---

---

---

---

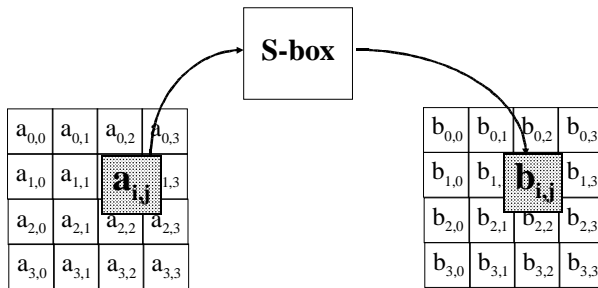


## Basic transformations

- Uniform, Parallel, Invertible
- No arithmetic operations = no carry propagation
- 4 basic transformations
  - ByteSub: nonlinearity
  - ShiftRow: intercolumn diffusion
  - MixColumn: inter-byte diffusion within columns
  - Round key addition: mixing key bits

88

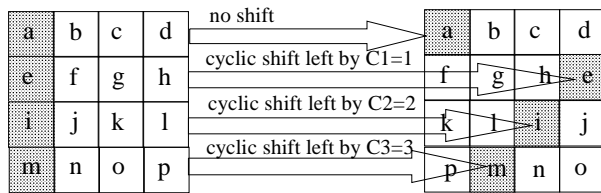
## ByteSub



- Bytes are transformed by applying an invertible S-box
- One single S-box for the complete cipher

89

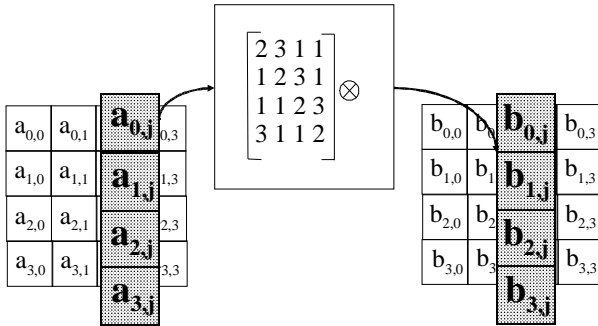
## ShiftRow



	Block size			
	128 bits	192 bits	256 bits	
C1	1	1	1	only in the initial specification, not supported by the standard
C2	2	2	3	
C3	3	3	4	

90

## MixColumn



### High diffusion

A difference in 1 input byte propagates to all 4 output bytes  
 A difference in 2 input bytes propagates to at least 3 output bytes  
 Any linear relation between input and output bits involves bits from  
 at least 5 different bytes (branch number = 5) <sup>91</sup>

---

---

---

---

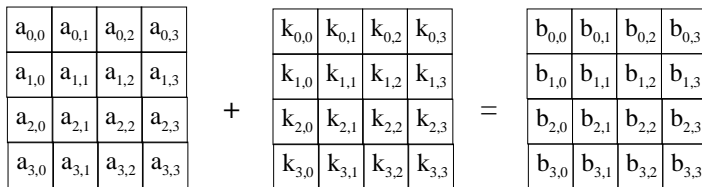
---

---

---

---

## Add Round Key



- simple bitwise addition (xor) of round keys

---

---

---

---

---

---

---

---

## Number of rounds

### Key length

Block length	128 bits Nk=4	192 bits Nk=6	256 bits Nk=8
128 bits Nb=4	10	12	14
required by the standard			
192 bits Nb=6	12	12	14
256 bits Nb=8	14	14	14

non-standard extensions

---

---

---

---

---

---

---

---

## Pseudocode for AES encryption

```

Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]

  state = in

  AddRoundKey(state, w[0, Nb-1])          // See Sec. 5.1.4

  for round = 1 step 1 to Nr-1
    SubBytes(state)                       // See Sec. 5.1.1
    ShiftRows(state)                      // See Sec. 5.1.2
    MixColumns(state)                     // See Sec. 5.1.3
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for

  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

  out = state
end

```

## S-box: substitution values for the byte xy (in hexadecimal notation)

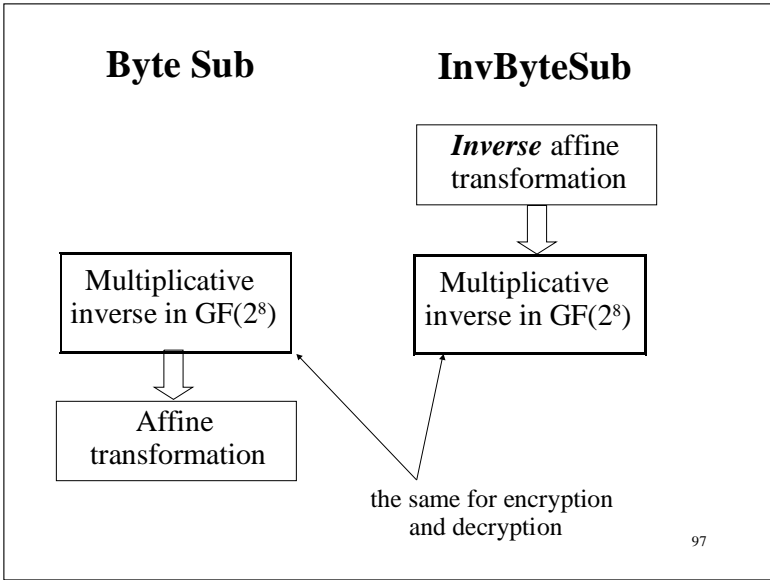
		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

95

## Inverse S-box: substitution values for the byte xy (in hexadecimal notation)

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

96




---

---

---

---

---

---

---

---

### Affine transformation

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

i.e.,  $b'_0 = b_0 + b_4 + b_5 + b_6 + b_7 + 1$   
 $b'_1 = b_0 + b_1 + b_4 + b_5 + b_6 + b_7 + 1$   
 $b'_2 = b_0 + b_1 + b_2 + b_4 + b_5 + b_6 + b_7 + 0$   
 $b'_3 = b_0 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + 0$   
 $b'_4 = b_0 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + 0$   
 $b'_5 = b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + 1$   
 $b'_6 = b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + 1$   
 $b'_7 = b_3 + b_4 + b_5 + b_6 + b_7 + 0$

In general:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad \text{98}$$


---

---

---

---

---

---

---

---

### Inverse affine transformation

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

i.e.,  $b'_0 = b_0 + b_4 + b_5 + b_6 + b_7 + 1$   
 $b'_1 = b_0 + b_1 + b_4 + b_5 + b_6 + b_7 + 1$   
 $b'_2 = b_0 + b_1 + b_2 + b_4 + b_5 + b_6 + b_7 + 0$   
 $b'_3 = b_0 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + 0$   
 $b'_4 = b_0 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + 0$   
 $b'_5 = b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + 1$   
 $b'_6 = b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + 1$   
 $b'_7 = b_3 + b_4 + b_5 + b_6 + b_7 + 0$

In general:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad \text{99}$$


---

---

---

---

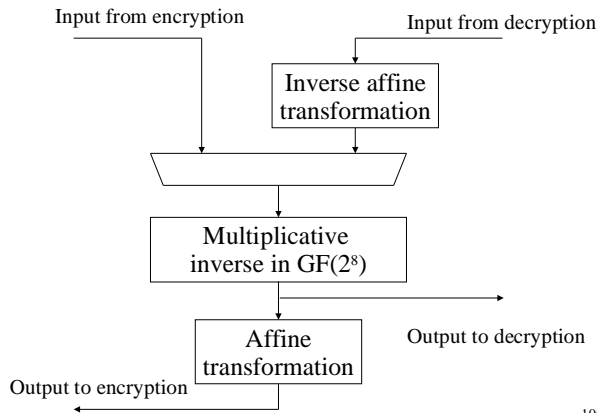
---

---

---

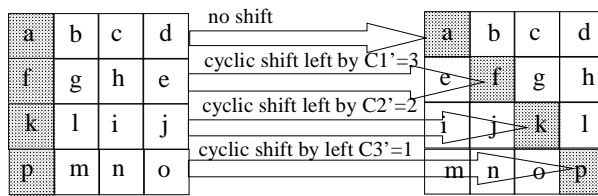
---

# ByteSub - Implementation in Hardware



100

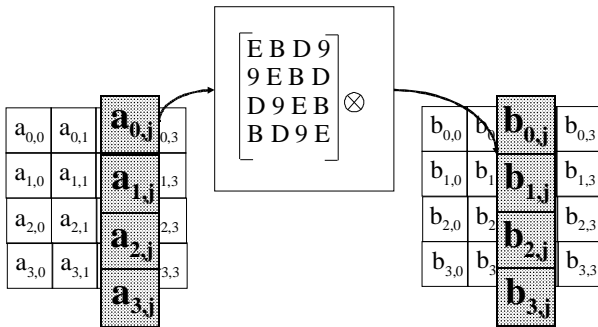
## InvShiftRow



	Block size		
	128 bits	192 bits	256 bits
C1'	3	5	7
C2'	2	4	5
C3'	1	3	4

101

## InvMixColumn



- more computationally intensive than MixColumn, especially on 8-bit smart card processors

102

## Pseudocode for AES decryption

```

InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]

  state = in

  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

  for round = Nr-1 step -1 downto 1
    InvShiftRows(state) // See Sec. 5.3.1
    InvSubBytes(state) // See Sec. 5.3.2
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    InvMixColumns(state) // See Sec. 5.3.3
  end for

  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, Nb-1])

  out = state
end

```

103

## Key scheduling

**Key size = 192 bits (Nk=6)**

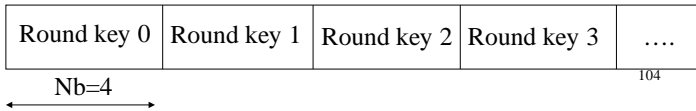
**Block size = 128 bits (Nb=4)**



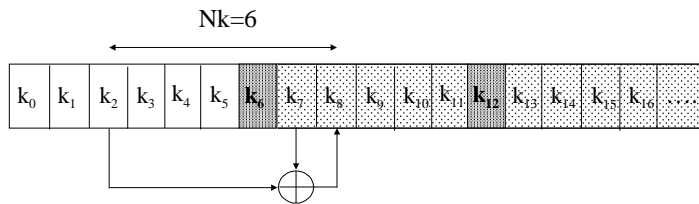
### Key expansion



### Round key selection



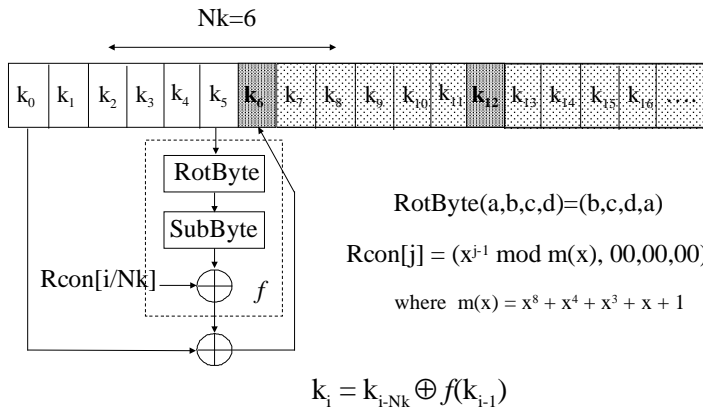
## Key expansion (1)



$$k_i = k_{i-Nk} \oplus k_{i-1}$$

105

## Key expansion (2)



106

---

---

---

---

---

---

## Pseudocode for Key Expansion

```

KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while
    i = Nk
    while (i < Nb * (Nr+1))
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
    
```

.J7

---

---

---

---

---

---

---

---

---

---

---

---