Cryptografie

Cyberboswachters

Tim Dams

Cryptografie

> Cryptografie: het versleutelen van informatie zodat enkel zender en ontvanger het bericht kunnen lezen.

Tip

Meer weten? Lees “The Codebreakers” van David Kahn — 1200 pagina’s die lezen als een thriller.

CIA

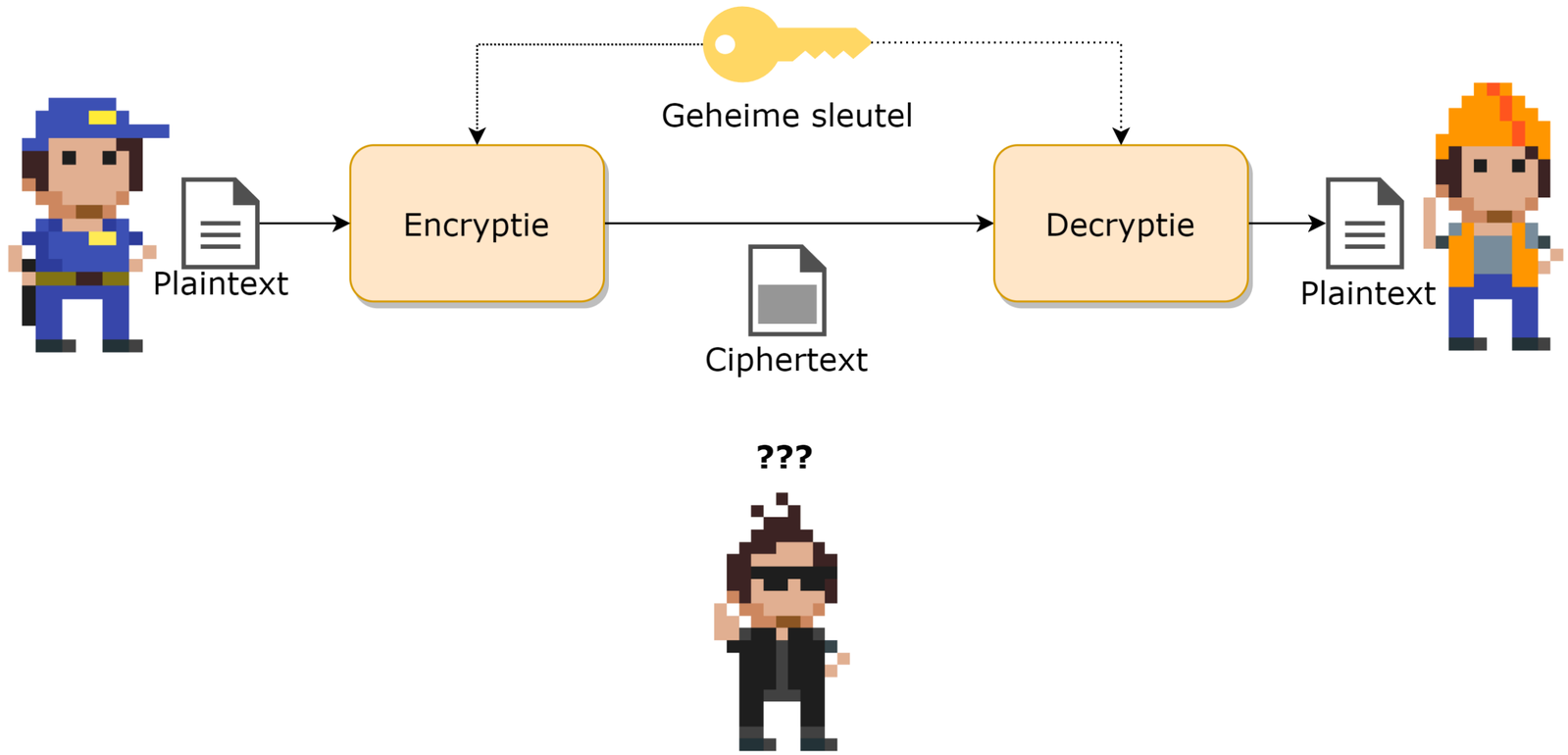
Aspect	Beschrijving
Confidentiality	Informatie is alleen leesbaar voor geautoriseerde partijen
Integrity	Informatie is niet gewijzigd tijdens overdracht
Availability	Informatie is toegankelijk wanneer nodig

Dit hoofdstuk: focus op **Confidentiality** via encryptie.

> **Tip**
>
> Integrity wordt ook vaak opgelost m.b.v. cryptgrafische technieken, zoals **hashing** en **digitale handtekeningen**.

Encryptie en decryptie

- Data versleutelen (**encrypteren**) met een **geheime sleutel**
  - → Data is onleesbaar zonder sleutel



Alice gebruikt encryptie om een beveiligd bericht naar Bob te sturen, zodat Eve deze niet kan lezen.

Enkele termen

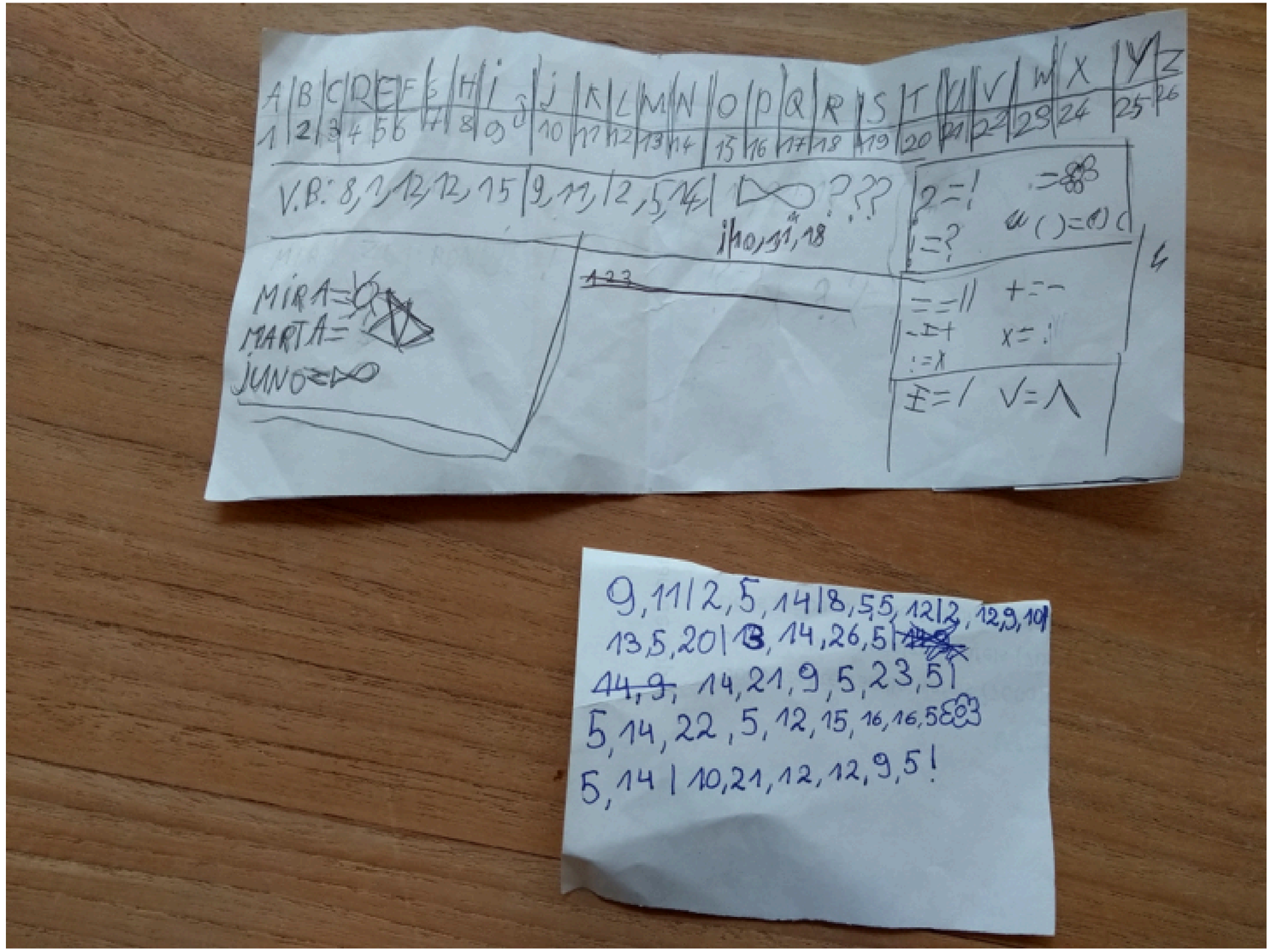
- **Plaintext** : originele data
- **Ciphertext** : versleutelde data
- **Cryptanalyse**: ontcijferen zonder de sleutel
- **Cryptografie**: ontwerpen van encryptiesystemen
- **Cryptologie**: combinatie cryptografie + cryptanalyse

Encryptie doorheen de tijd

- Al sinds de **Romeinen** wordt er aan cryptografie gedaan
- Wedloop tussen stropers en boswachters
  - → Sterkere computers → sterkere algoritmes nodig

Crypto volgens m’n kids

Crypto volgens m’n kids, deel 2



Classificatie van cryptosystemen

- **Acties** op de data:
  - → **Substitutie**: teken door ander teken vervangen
  - → **Transpositie**: teken op andere plek zetten
  - → **Product**: combinatie van beiden
- **Aantal sleutels**:
  - → **1 sleutel** → symmetrisch / private encryption
  - → **2 sleutels** → asymmetrisch / public encryption
- **Verwerking**:
  - → **Blok**: blok per blok
  - → **Stream**: teken per teken

Kerckhoffs principe

August Kerckhoffs

> *“Het kennen van het algoritme door de tegenstander is geen probleem. Enkel de geheime sleutel moet uit diens handen blijven.”*

- De **sleutel** moet geheim blijven (sleutel == wachtwoord)
- Het **algoritme** mag publiek zijn
  - → Publieke algoritmes worden door duizenden experts getest → veiliger
  - → *Security through obscurity* == **vals gevoel van veiligheid**

> ⚠ **Waarschuwing**
>
> **Alles draait dus om het geheimhouden van je sleutel!**

Sleutellengtes en bruteforcen

- **Bruteforce** = alle mogelijke sleutels testen
- Gemiddelde kraaktijd = is maar de **helft** van de maximale tijd

Sleutellengtes en bruteforcen, deel 2

$$MaximaleTijd = \frac{AantalMogelijkeTekens^{SleutelLengte}}{pogingen/seconde}$$

Voorbeeld: 8 tekens (a-z), 1M pogingen/s:

$$\frac{26^8}{1\,000\,000} = 208\,827 \text{ sec} \approx 58 \text{ uur}$$

Gemiddeld gevonden in **~29 uur**.

Eén teken extra → **62 dagen**!

Bruteforce: de tabel

GeForce GTX 1080 (~30M pogingen/s):

# tekens	nummers (10)	kleine letters (26)	alle tekens (95)
6	33 ms	10 s	6,8 uur
8	3,3 s	1,9 uur	**7 jaren**
10	5,6 min	54 dagen	$6,3 \times 10^4$ jaren
12	9,3 u	100 jaren	$5,7 \times 10^8$ jaren
16	11 jaar	$4,6 \times 10^7$ jaren	$4,7 \times 10^{16}$ jaren

Opmerking

Leeftijd universum: $1,38 \times 10^{10}$ jaar. Per verdubbeling GPU's halveert de tijd.

Dictionary attack

- Bruteforce verbeteren door een **woordenboek** te gebruiken
- Tools zoals **John The Ripper** testen variaties: *god1*, *god2*, ...
- Meest gebruikte wachtwoorden: `123456`, `password`, `qwerty`, `master`, ...

> **Tip**
>
> Gebruik nooit wachtwoorden uit bekende lijsten! Zie: SecLists

Soorten cryptanalytische aanvallen

Type	Beschrijving
Enkel ciphertext	Alleen geëncrypteerde data beschikbaar (beste scenario voor cyberboswachters)
Gekende plaintext	Zowel ciphertext als bijhorende plaintext gekend
Gekozen plaintext	Analist kiest plaintext en ziet resulterende ciphertext
Gekozen ciphertext	Analist kiest ciphertext en ziet resulterende plaintext

De beste “cryptanalyse” is géén cryptanalyse

- Veel makkelijker dan sleutels kraken: **vraag** de sleutel gewoon aan de gebruiker
  - → Phishing, helpdesk-scams, post-its onder toetsenbord, …
- *“There is no patch for human stupidity”*
- → Hoofdstuk **Social Engineering** behandelt dit uitgebreid

Belangrijk

Cryptografie is **noodzakelijk** maar zelden **voldoende** voor veiligheid.

Quantum-computers en crypto

- Quantum-computers kunnen bruteforce **drastisch versnellen**
  - → Veel veiligheidsdiensten hanteren al jaren **“Store now, decrypt later”** strategie
- Alle huidige encryptie potentieel kwetsbaar
- Impact op **cryptocurrencies**, e-commerce, wachtwoorden, …
- Onderzoek naar **post-quantum cryptografie** is bezig

> **Opmerking**
>
> *“Encryption is everywhere in modern day life — everything will be vulnerable!”*

De eerste algoritmes

Encryptie-algoritmes onderverdeeld naar actie:

1. **Substitutie** (bv. Caesar)
2. **Transpositie** (bv. Scytale)
3. **Combinatie** van beiden (bv. AES)

> **Tip**
>
> Handige tool om crypto te leren: CrypTool

Caesar encryptie (substitutie)

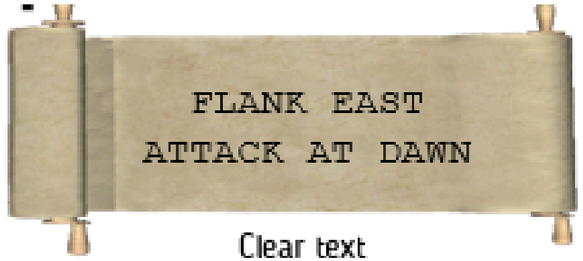
- Sleutel = getal 1-25 → iedere letter schuift op
  - → A=1, B=2, … Z=26
  - → Sleutel 3: A → D, B → E, … Z → C (overflow)
- Ook wel **Rot** (rotatie) genoemd: Rot4, Rot13, …
  - → **Rot13** is speciaal: 2x toepassen = originele tekst

Formule: `(teken + sleutel) % 26 = nieuw teken`

Voorbeeld: Y (24) met sleutel 7: `(24+7) % 26 = 5` → **F**

Caesar voorbeeld

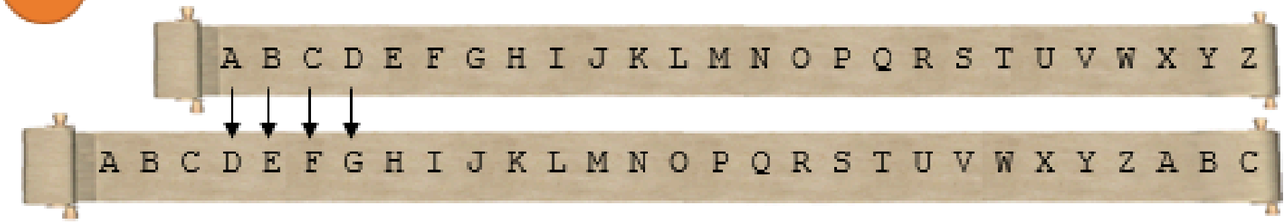
1



Clear text

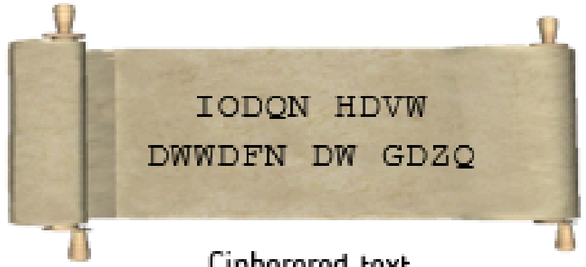
The clear text message would be encoded using a key of 3.

2



Shift the top scroll over by three characters (key of 3), an A becomes D, B becomes E, and so on.

3

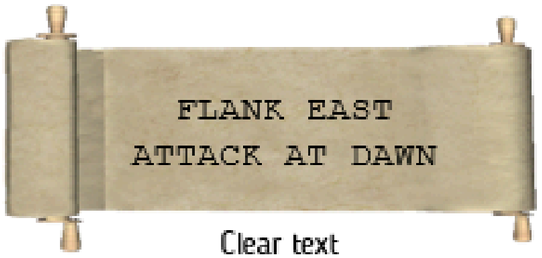


Ciphererred text

The clear text message would be encrypted as follows using a key of 3.

Caesar wiel

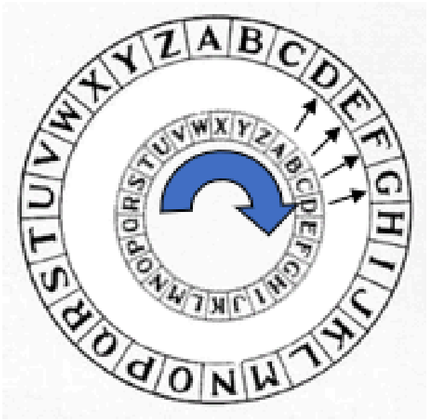
1



Clear text

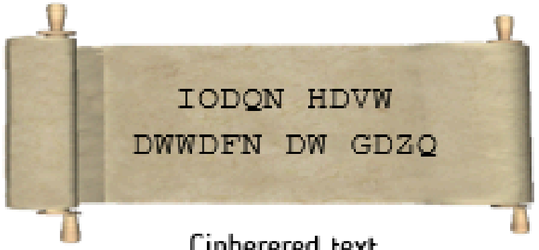
The clear text message would be encoded using a key of 3.

2



Shifting the inner wheel by 3, then the A becomes D, B becomes E, and so on.

3



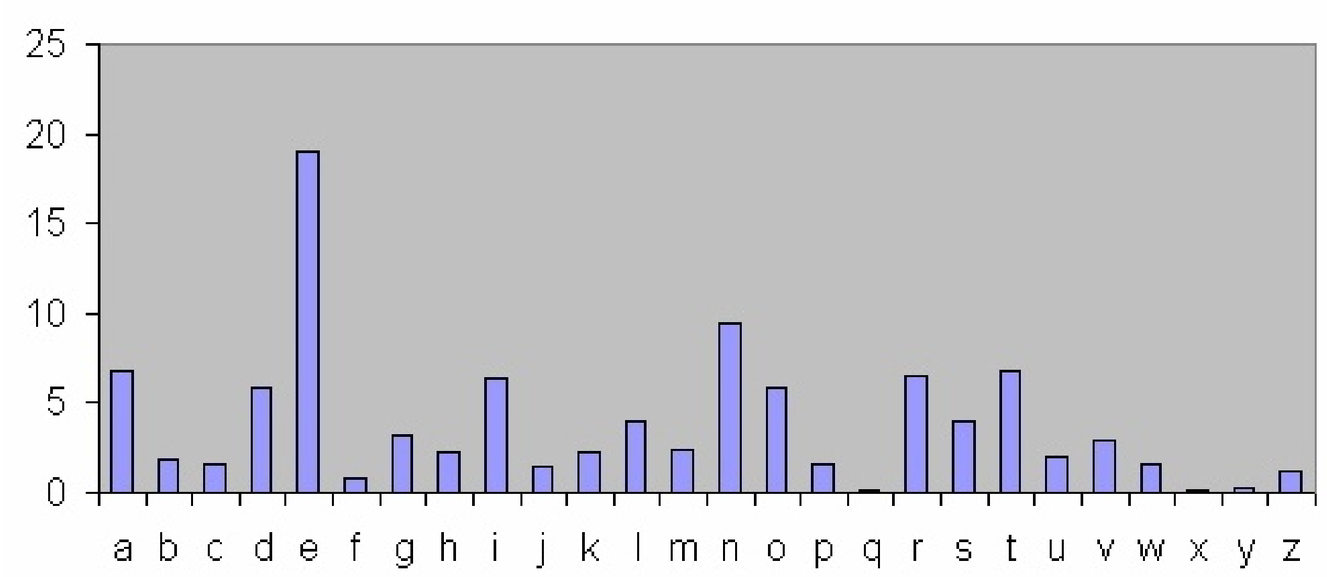
Ciphererred text

The clear text message would appear as follows using a key of 3.

Andere manier om Caesar encryptie te visualiseren

Caesar kraken

- Via bruteforce: Slechts **25 mogelijke sleutels**
- Via **Frequentieanalyse**: statistische verdeling van letters in een taal
  - → Letter e komt veel vaker voor dan v in NL
  - → Meest voorkomende letter in ciphertext → waarschijnlijk e



Frequentie-analyse Nederlandstalige tekst (Bron: Wikipedia).

Vigenère: substitutie 2.0

- Zwakte Caesar: iedere letter → **zelfde** shift → frequentieanalyse werkt
- **Vigenère** (16e eeuw): gebruik een **sleutelwoord** i.p.v. één getal
  - → Iedere letter van de sleutel = shift voor bijhorende plaintext-letter
  - → Sleutelwoord wordt herhaald over de tekst
- = “Combinatie van meerdere Caesars na elkaar”

> **Tip**
>
> Eeuwenlang **onkraakbaar** genoemd: *le chiffre indéchiffrable*

Vigenère: voorbeeld

Sleutel: `COUNTON`, plaintext:

`vigenerecipher`

```
C O U N T O N C O U N T O N
v i g e n e r e c i p h e r
X W A R G S E G Q C C A S E
```

- `v` + sleutel `C` (shift 2) → `X`
- Iedere `e` → *andere* letter (R, S, G, S, E)
- → frequentieanalyse werkt **niet meer**

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Tabula recta

Opmerking

19e eeuw: Kasiski kraakte het. Sleutellengte bekend → meerdere parallelle Caesars.

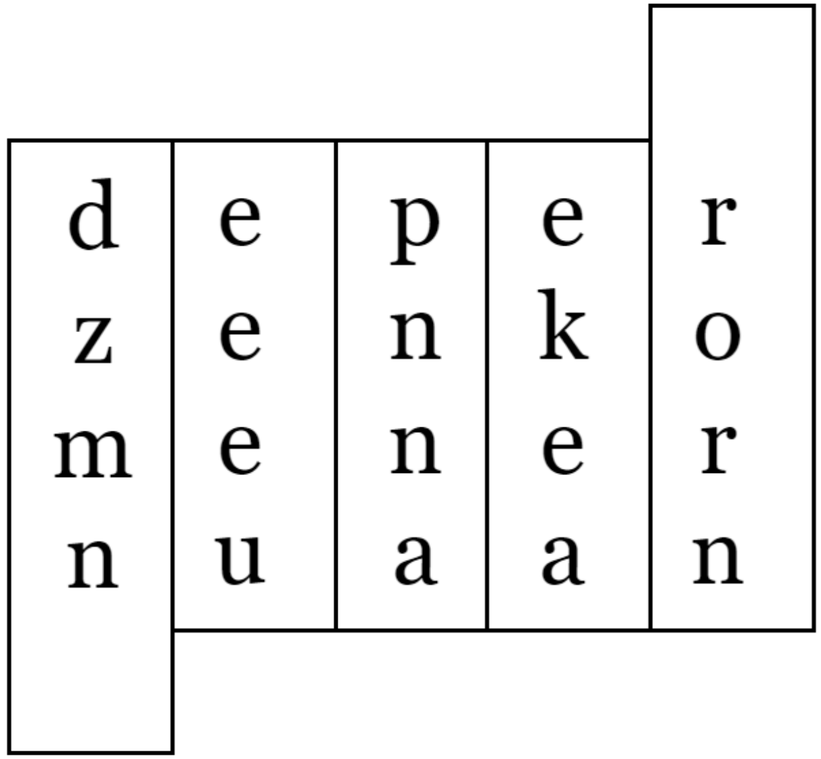
Scytale encryptie (transpositie)

- Oude Grieken: lint wikkelen rond stok met meerdere zijden
- **Sleutel** = aantal vlakken van de scytale
- Letters komen op verschillende zijden → per zijde uitlezen = ciphertext

Een authentieke scytale (Bron: Wikipedia).

Scytale: voorbeeld

“De perzen komen er nu aan” op scytale met 4 zijden:



Ciphertext na afwikkelen:

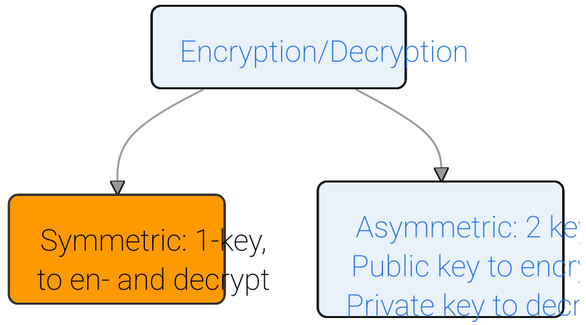
dzmneeeupnnaekearorn

Combinatie: substitutie + transpositie

- Combinatie versterkt encryptie
- Moderne algoritmen = sequentie van basisvormen

  → **AES** (de facto standaard): substituties (*sub*) + transposities (*mixcolumns*, *shiftrows*) in meerdere rondes (zien we later)

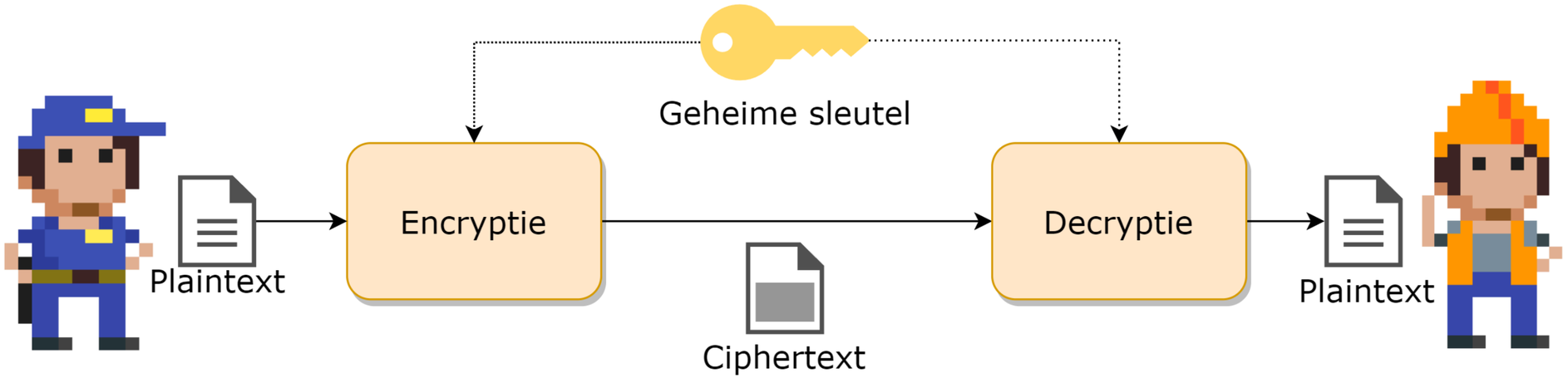
Moderne cryptosystemen



Symmetrische encryptie

- **Eén sleutel** voor zowel encryptie als decryptie
  - → Zender en ontvanger gebruiken **dezelfde sleutel**
- Oudste vorm van encryptie
- Voorbeelden: **AES**, DES, 3DES, RC4, Blowfish, IDEA

Symmetrische encryptie basismodel



Sleuteloverdracht == moeilijk in symmetric crypto

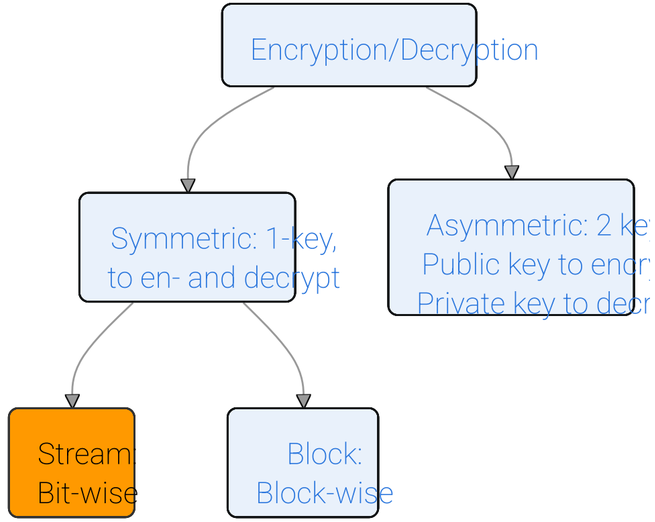
Het grote probleem: hoe wissel je de sleutel **veilig** uit?

- Via **asymmetrisch** encryptiesysteem (zie later)
- Via een **ander beveiligd kanaal** (fysiek, ander kanaal, …)

Block- en streamciphers

	Streamcipher	**Blockcipher**
Verwerking	Teken per teken	Blok per blok (bv. 128 bits)
Snelheid	Sneller	Trager
Voorbeelden	RC4	AES, DES, 3DES

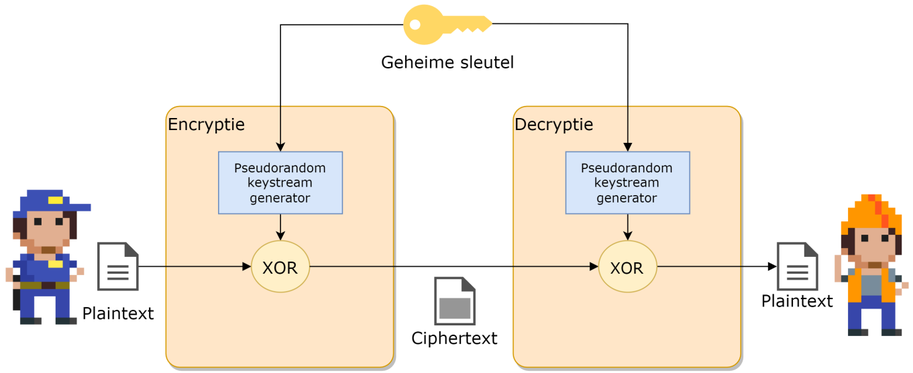
Streamciphers



Streamciphers: werking

Twee onderdelen:

1. **Keystream generator**: expandeert sleutel naar keystream (zelfde lengte als data)
2. **XOR-functie**: plaintext $\oplus$ keystream = ciphertext



Belangrijk

Ontvanger met **zelfde sleutel** → zelfde keystream → originele plaintext

De XOR functie : hart van cryptografie!

Plaintext	Keystream	Ciphertext
1	0	**1**
0	1	**1**
0	0	**0**
1	1	**0**

Voorbeeld: `1010` $\oplus$ `1101` = `0111` (ciphertext)

Dezelfde XOR met keystream decrypteert weer!

Keystream generator

- Genereert **pseudo-random** keystream
- Zelfde sleutel (**seed**) → zelfde reeks (reproduceerbaar)
- Onvoorspelbaar voor buitenstaanders
- Zwakke generator → **kwetsbare encryptie** (bv. WEP!)
- Gebaseerd op **PRNG** (Pseudo-Random Number Generator)

Middle-square: de eerste PRNG

Von Neumann (1946): eenvoudig maar krachtig idee

1. Start met **seed**
2. **Kwadrateer**
3. Neem **middelste cijfers** → nieuw getal
4. Herhaal

Voorbeeld (seed = 1111):

```
1111² = 01234321  →  2343
2343² = 05489649  →  4896
4896² = 23970816  →  9708
```

Reeks: `2343, 4896, 9708, ...`

> **Opmerking**
>
> Ondertussen onveilig (korte cycli), maar basis van alle moderne PRNGs!

PRNG in de praktijk: Random() in C

```
int key = 666;

Random s = new Random(key);    // Zender
Random r = new Random(key);    // Ontvanger
Random e = new Random(123);    // Eve (andere seed)
```

Partij	Output (10 getallen)
Zender	`6337963648`
Ontvanger	`6337963648`
Eve	`9978711226`

Zelfde seed → zelfde reeks. Andere seed → totaal andere reeks.

> ⚠ **Waarschuwing**
>
> C# `Random` is **niet** cryptografisch veilig! Gebruik `RandomNumberGenerator` uit `System.Security.Cryptography`.

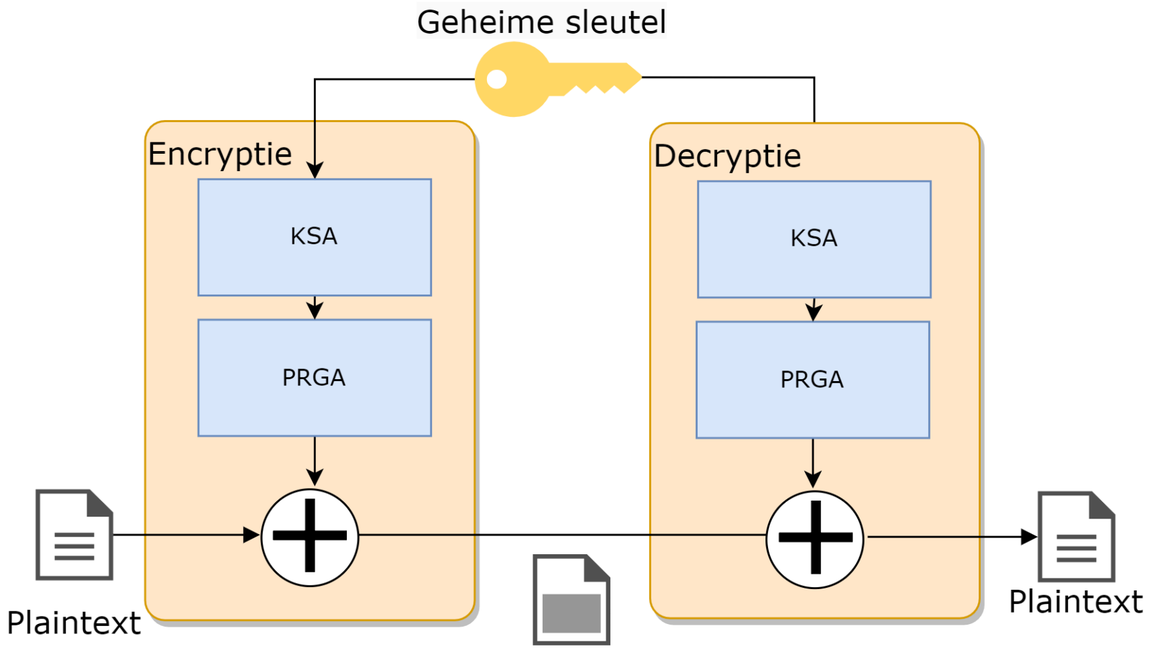
RC4: streamcipher

- Ontwikkeld door Ron Rivest (*Ron's Cipher 4*)
- Sleutel: 40 tot 2048 bits
- Eén van de meest gebruikte streamciphers ooit (WEP, SSL/TLS)

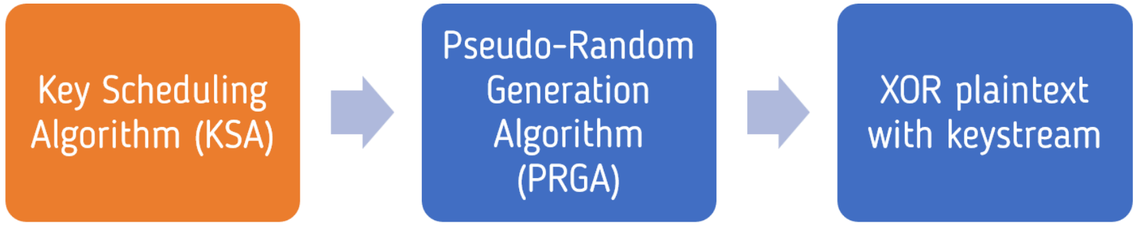
  → Nu verouderd en onveilig, maar nog steeds een goed voorbeeld van een streamcipher

RC4 in actie

- Twee fasen:
  1. **KSA** (Key Scheduling Algorithm): sleutel → werksleutel
  2. **PRGA** (Pseudo-Random Generation Algorithm): werksleutel → keystream



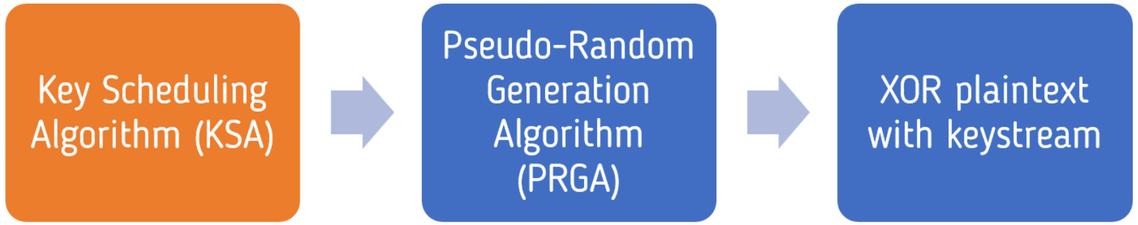
KSA



- Generate a RC4 compatible key (S) based on the user key (T)

```
for i from 0 to 255
    S[i] := i
    T[i] := key[i mod keylength]
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] +T[i]) mod 256
    swap values of S[i] and S[j]
endfor
```

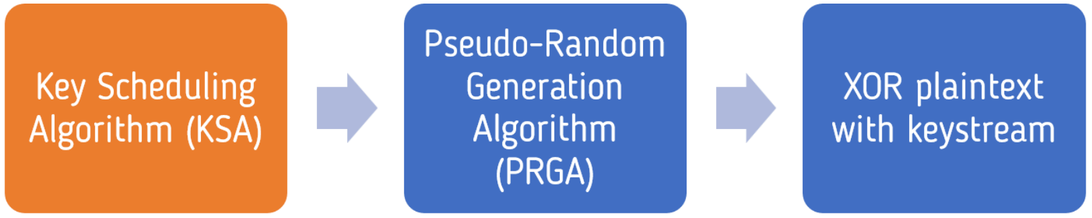
KSA



- Generate a RC4 compatible key (S) based on the user key (T)

```
for i from 0 to 255
    S[i] := i
    T[i] := key[i mod keylength]
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] +T[i]) mod 256
    swap values of S[i] and S[j]
endfor
```

KSA: Step 1

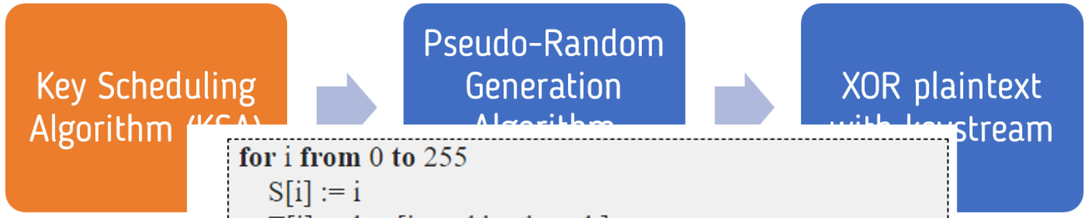


- Fill 256 array with the key, repeat if needed.
- Example, key = “pwd12” => 112 119 100 49 50 in decimal

```
for i from 0 to 255
    S[i] := i
    T[i] := key[i mod keylength]
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] +T[i]) mod 256
    swap values of S[i] and S[j]
endfor
```

Array Indices **i**	**T[i]**	Binary Representation of **T[i]**
0	112	01110000
1	119	01110111
2	100	01100100
3	49	00110001
4	50	00110010
5	112	01110000
6	119	01110111
. . .	. . .	. . .
253	49	00110001
254	50	00110010
255	112	01110000

KSA: Step 2



- Permute key array T with array S using:

```
for i from 0 to 255
    S[i] := i
    T[i] := key[i mod keylength]
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] +T[i]) mod 256
    swap values of S[i] and S[j]
endfor
```

S

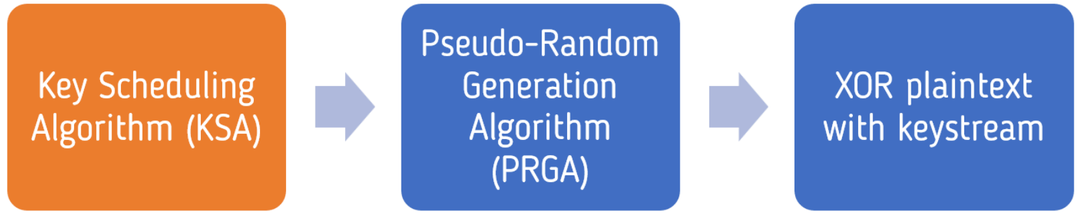
Array Indices i	S[i]
0	0
1	1
2	2
3	3
4	4
5	5
6	6
.	.
.	.
.	.
253	253
254	254
255	255

T

Array Indices i	T[i]	Binary Representation of T[i]
0	112	01110000
1	119	01110111
2	100	01100100
3	49	00110001
4	50	00110010
5	112	01110000
6	119	01110111
.	.	.
.	.	.
.	.	.
253	49	00110001
254	50	00110010
255	112	01110000

Iteration No.	i	j	S[i]	T[i]	New j (j + S[i] + T[i]) mod 256		Swapped elements S[i] and S[j]	New swapped S elements	
					Calculation	New j		Element	New Value
1	0	0	0	112	(0+0+112)mod256	112	S[0], S[112]	S[0]	112
								S[112]	0
2	1	112	1	119	(112+1+119)mod 256	232	S[1], S[232]	S[1]	232
								S[232]	1
3	2	232	2	100	(232+2+100)mod256	78	S[2], S[78]	S[2]	78
								S[78]	2
4	3	78	3	49	(78+3+49)mod256	130	S[3], S[130]	S[3]	130
								S[130]	3
5	4	130	4	50	(130+4+50)mod256	184	S[4], S[184]	S[4]	184
								S[184]	4
6	5	184	5	112	(184+5+112)mod256	45	S[5], S[45]	S[5]	45
								S[45]	5
7	6	45	6	119	(45+6+119)mod256	170	S[6], S[170]	S[6]	170
								S[170]	6
8	7	170	7	100	(170+7+100)mod256	21	S[7], S[21]	S[7]	21
								S[21]	7
9	8	21	8	49	(21+8+49)mod256	78	S[8], S[78]	S[8]	78
								S[78]	8
10	9	78	9	50	(78+9+50)mod256	117	S[9], S[117]	S[9]	117
								S[117]	9

KSA: Final result

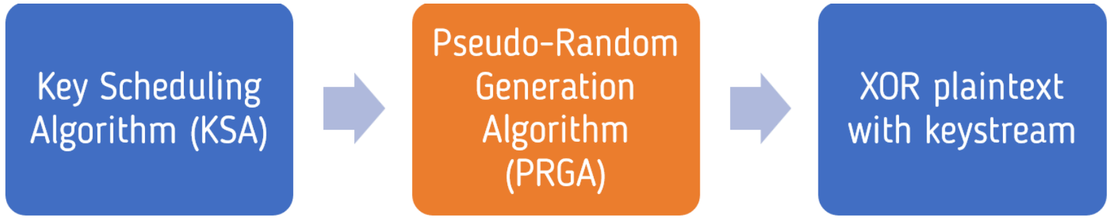


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
101	124	172	10	166	26	46	91	2	137	39	243	253	25	3	30
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
47	238	196	38	94	149	15	32	248	51	158	150	106	183	67	219
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
95	177	138	152	13	188	118	108	207	151	41	142	236	103	55	72
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
20	244	216	14	168	90	4	42	153	64	250	129	97	225	87	199
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
204	100	16	249	191	82	43	131	24	169	69	54	96	0	255	84
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
1	143	242	123	21	93	61	102	224	107	109	79	80	23	229	6
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
156	181	105	159	33	141	0	104	9	56	233	178	127	111	135	206
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
202	128	31	71	211	222	45	66	163	189	167	201	232	17	251	198
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
170	155	115	57	228	98	190	76	59	239	37	147	180	240	197	200
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
19	0	213	99	125	44	195	164	176	121	220	212	86	186	34	214
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
230	254	40	203	194	231	162	226	187	116	208	22	68	88	192	140
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
205	234	119	83	136	63	12	112	217	154	184	81	70	35	174	78
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
241	179	210	215	49	144	130	48	133	7	209	92	73	193	28	75
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
117	223	50	113	114	148	173	29	53	160	8	139	246	65	252	161
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
221	185	27	36	11	110	237	165	5	182	145	171	120	157	134	175
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
122	58	235	52	62	126	85	60	132	74	245	227	218	89	247	146

- KSA algorithm

```
for i from 0 to 255
    S[i] := i
    T[i] := key[i mod keylength]
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] +T[i]) mod 256
    swap values of S[i] and S[j]
endfor
```

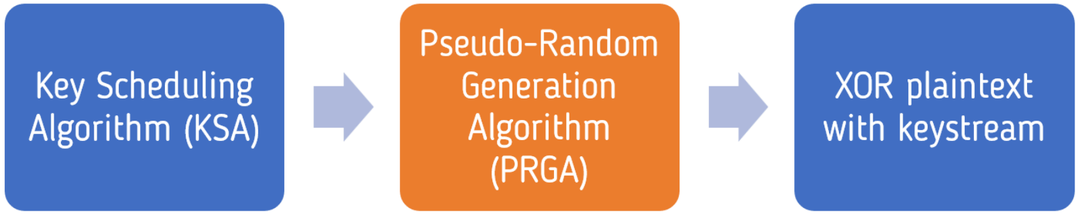
PRGA



- Generate keystream, based on generated key

```
i = 0
j = 0
Until the required keystream bytes have been generated:
   i = (i + 1) mod 256
   j = (j + S[i]) mod 256
   swap ( S[i], S[j] )
   k = S[(S[i] + S[j]) mod 256]
  output the keystream byte k
end
```

PRGA



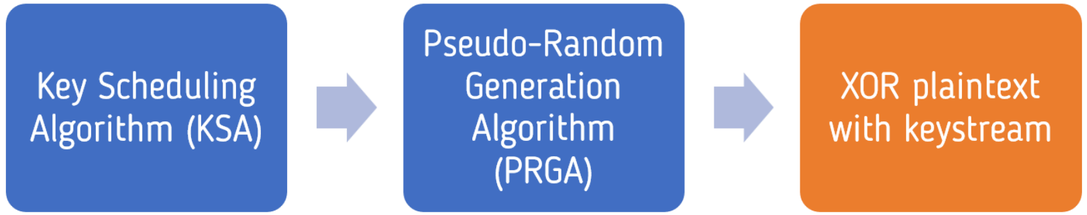

```
i = 0
j = 0
Until the required keystream bytes have been generated:
   i = (i + 1) mod 256
   j = (j + S[i]) mod 256
   swap ( S[i], S[j] )
   k = S[(S[i] + S[j]) mod 256]
  output the keystream byte k
end
```

Iteration No.	Previous **i**	Previous **j**	New i **(i+1)mod256**	New j **(j+S[i])mod256**	Swapped Values of **S[i], S[j]**		k = **S[(S[i] + S[j]) mod 256]**	
							S[i] + S[j]	**k**
1	0	0	1	124	S[1]	232	356	33
					S[124]	124		
2	1	124	2	40	S[2]	172	379	201
					S[225]	207		
3	2	225	3	50	S[3]	10	226	27
					S[50]	216		
4	3	50	4	216	S[4]	166	219	139
					S[216]	53		
5	4	216	5	242	S[5]	26	261	235
					S[242]	235		
6	5	242	6	32	S[6]	46	141	240
					S[32]	95		
7	6	32	7	123	S[7]	91	292	13
					S[123]	201		
And so on until the required length of keystream is fetched.								

Index	K[index]	Binary Representation
0	33	00100001
1	201	11001001
2	27	00011011
3	139	10001011
4	235	11101011
5	240	11110000
6	13	00001011
and so on	.....	.....

Generated Keystream

PRGA



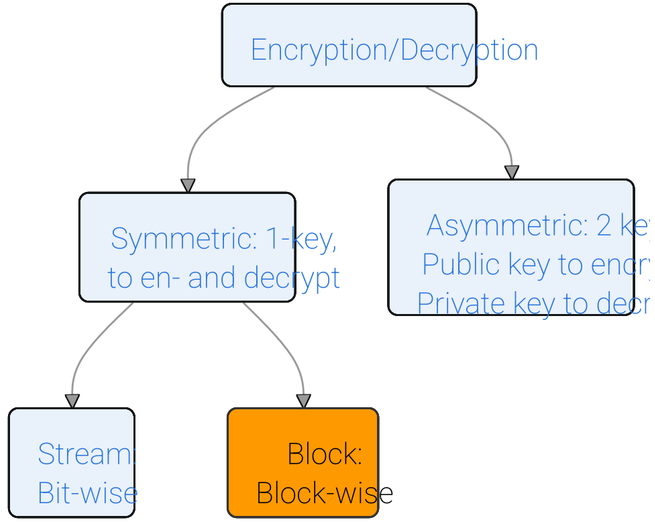
- Suppose first byte of plaintext = 1111 0101

Index	K[index]	Binary Representation
0	33	00100001
1	201	11001001
2	27	00011011
3	139	10001011
4	235	11101011
5	240	11110000
6	13	00001011
and so on	.....	.....

Generated Keystream

$$11110101 \oplus 00100001 = 11010100$$

Blockciphers

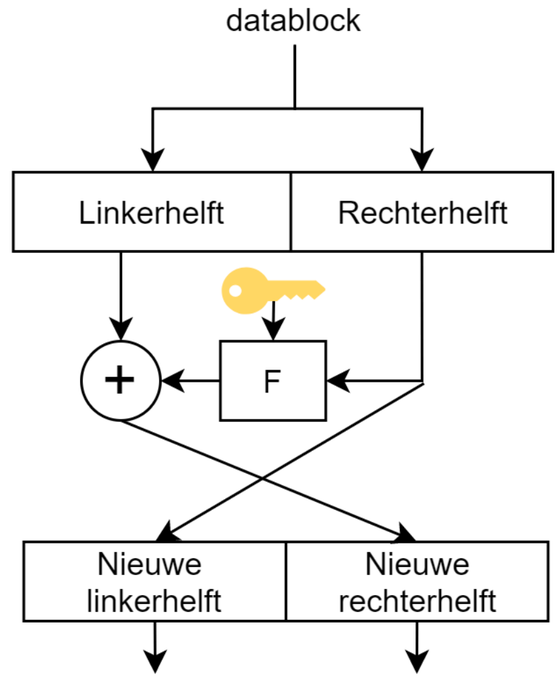


Blockciphers

- Plaintext opdelen in **blokken** (bv. 128 bits)
- Blok per blok encrypteren
- Laatste blok eventueel **padding** nodig

Intermezzo: Feistel-structuren

- Nieuwe soort operatie (naast XOR, substitutie, transpositie): **Feistel-structuur**
- Data splitsen in **twee helften** (L en R)
- F-operatie op één helft → XOR met andere helft
- **Meerdere rondes** herhalen
- Elke ronde een andere **subkey** (via key scheduling)



Een enkele Feistel-structuur.

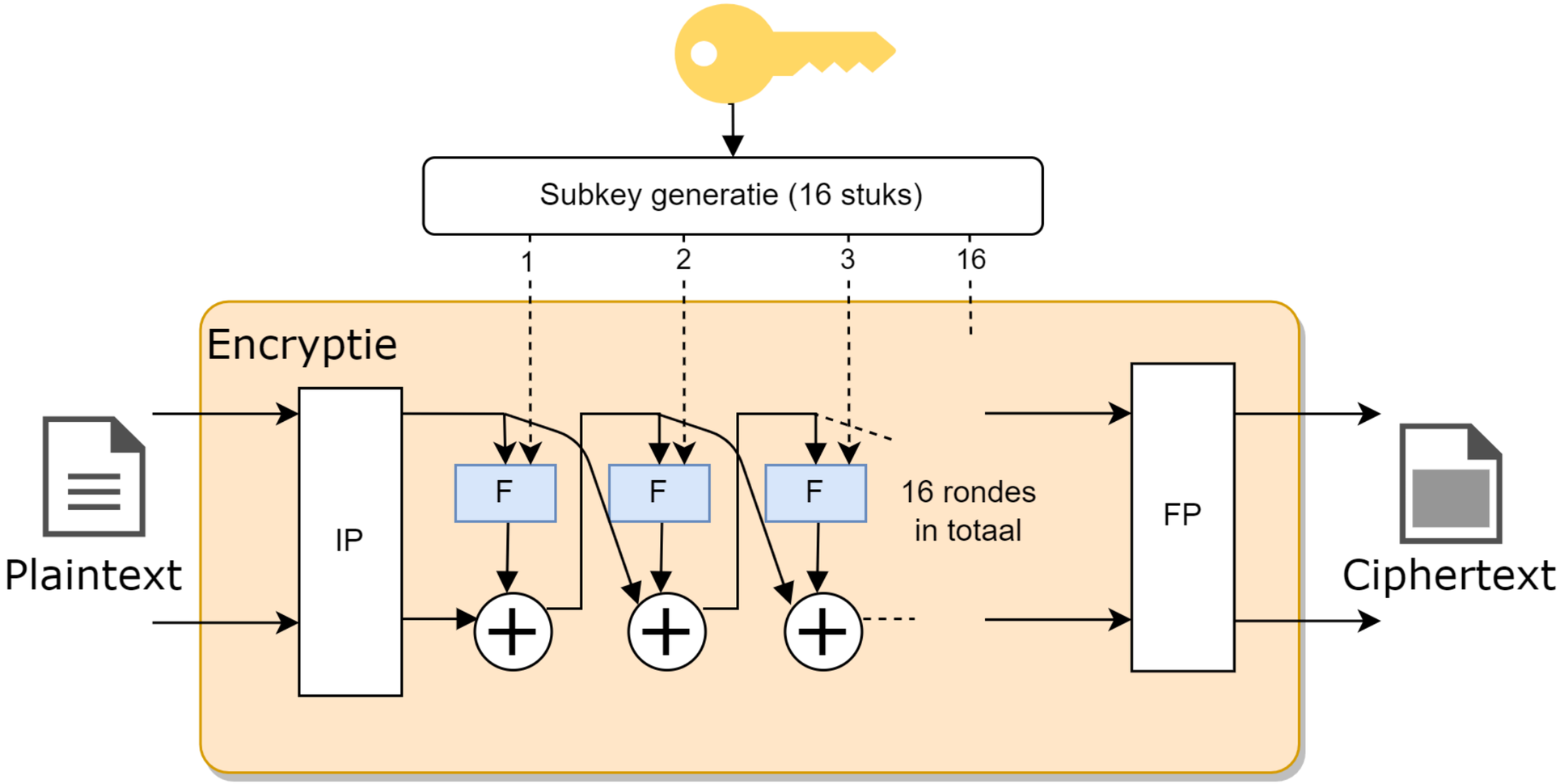
DES (Data Encryption Standard)

- **1977**, blokken van 64 bits, sleutel van 56 bits
- Outdated maar belangrijk voor begrip
- Gebruikt in **bankwezen** (financiële transacties)
- Gebaseerd op IBM’s Lucifer cipher (NSA halveerde sleutellengte!)

Twee onderdelen:

1. **Versleuteling**: 16 Feistel-rondes
2. **Subkey generatie**: 16 subkeys uit 56-bit sleutel

DES: versleuteling

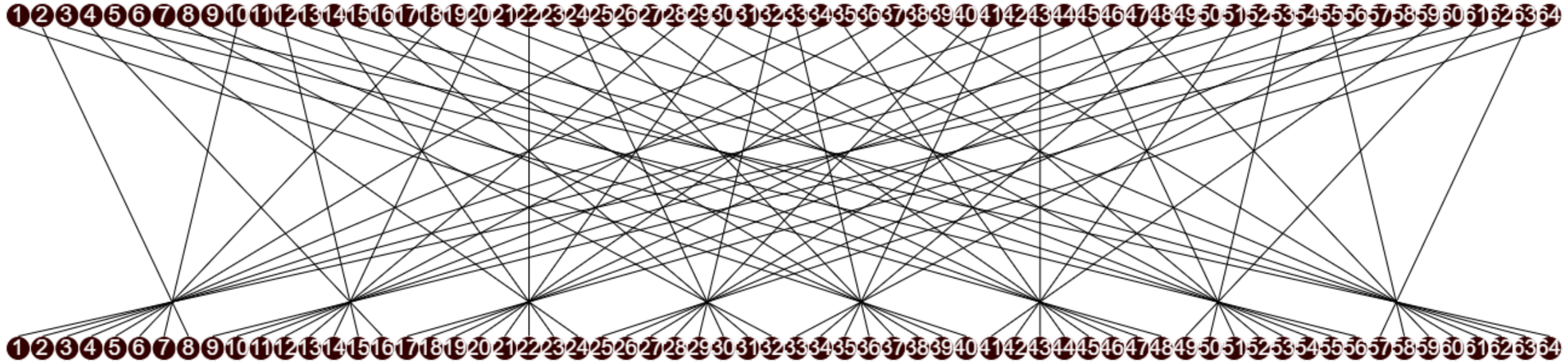


DES encryptie: 16 Feistel-structuren.

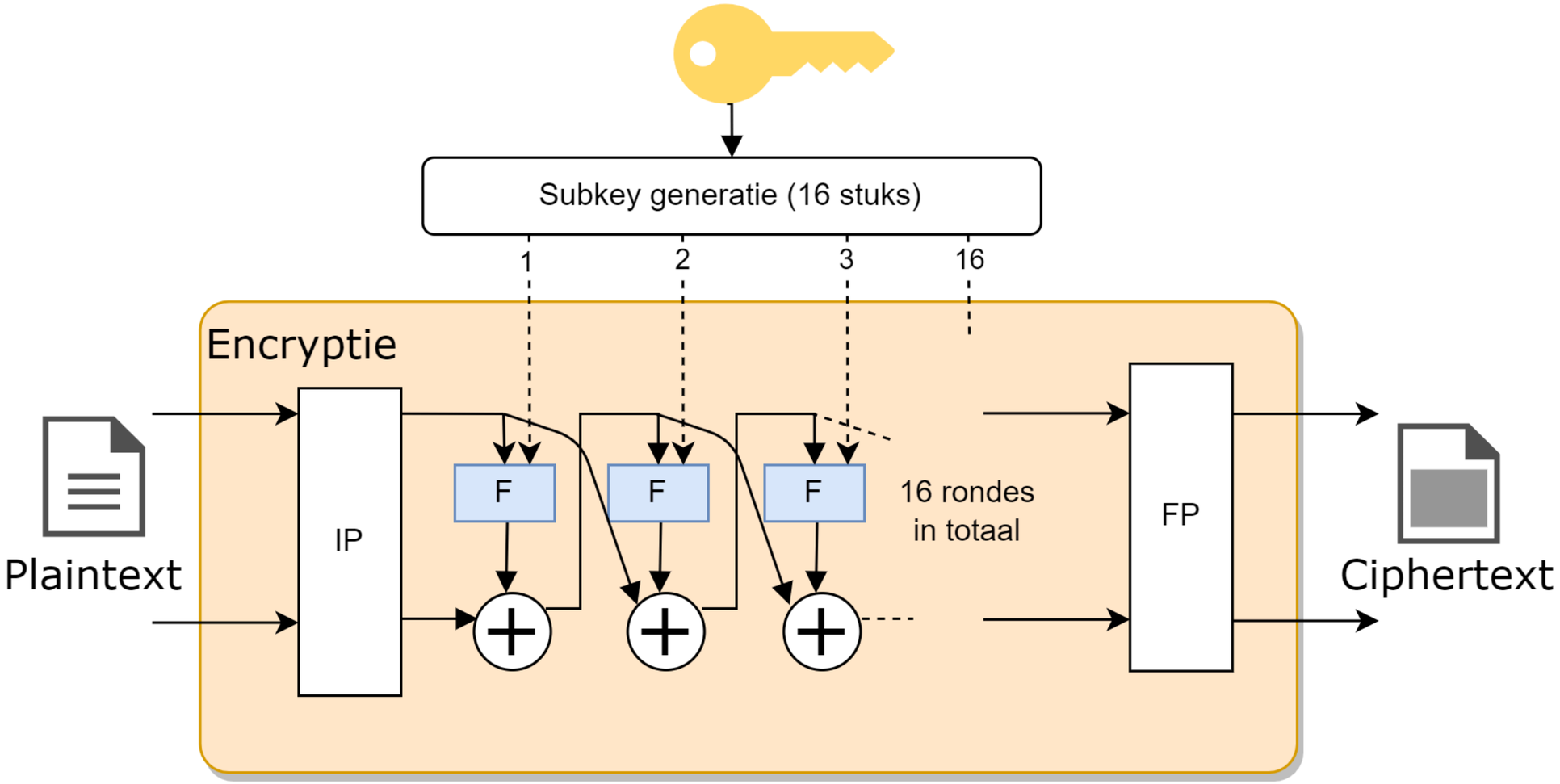
DES: stappen per ronde

1. **Initiële Permutatie** (IP): bits herschikken
2. **16 Feistel-rondes**: splitsen, F-operatie, XOR
3. **Finale Permutatie** (FP)

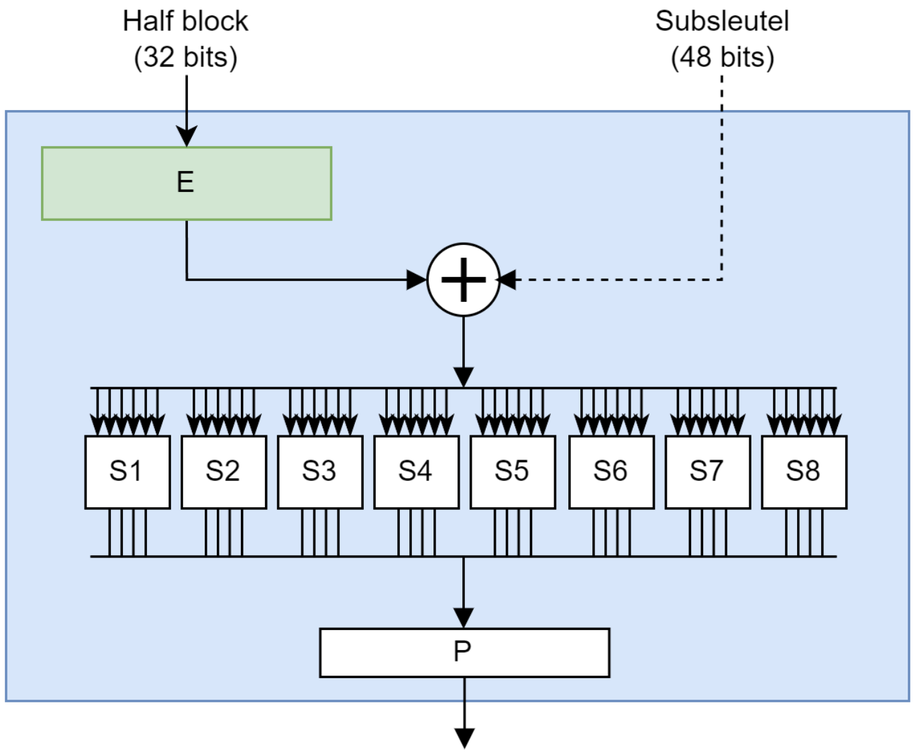
DES: De Initiële Permutatie.



DES

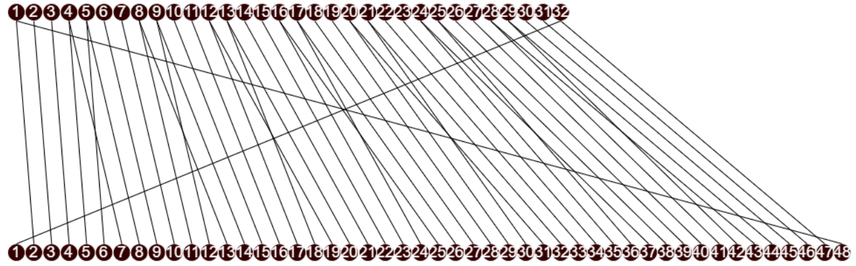

Subkey generatie (16 stuks)
1
2
3
16
Encryptie
Plaintext
IP
F
F
F
16 rondes
in totaal
FP
Ciphertext

DES: de F-operatie



Binnenin de F-operatie.

1. **Expansie**: 32 bits → 48 bits



2. **XOR** met subkey (48 bits)
3. **S-boxen**: 6 bits → 4 bits (substitutie) (zie volgende slide)
4. **P-box**: permutatie

DES: S-boxen

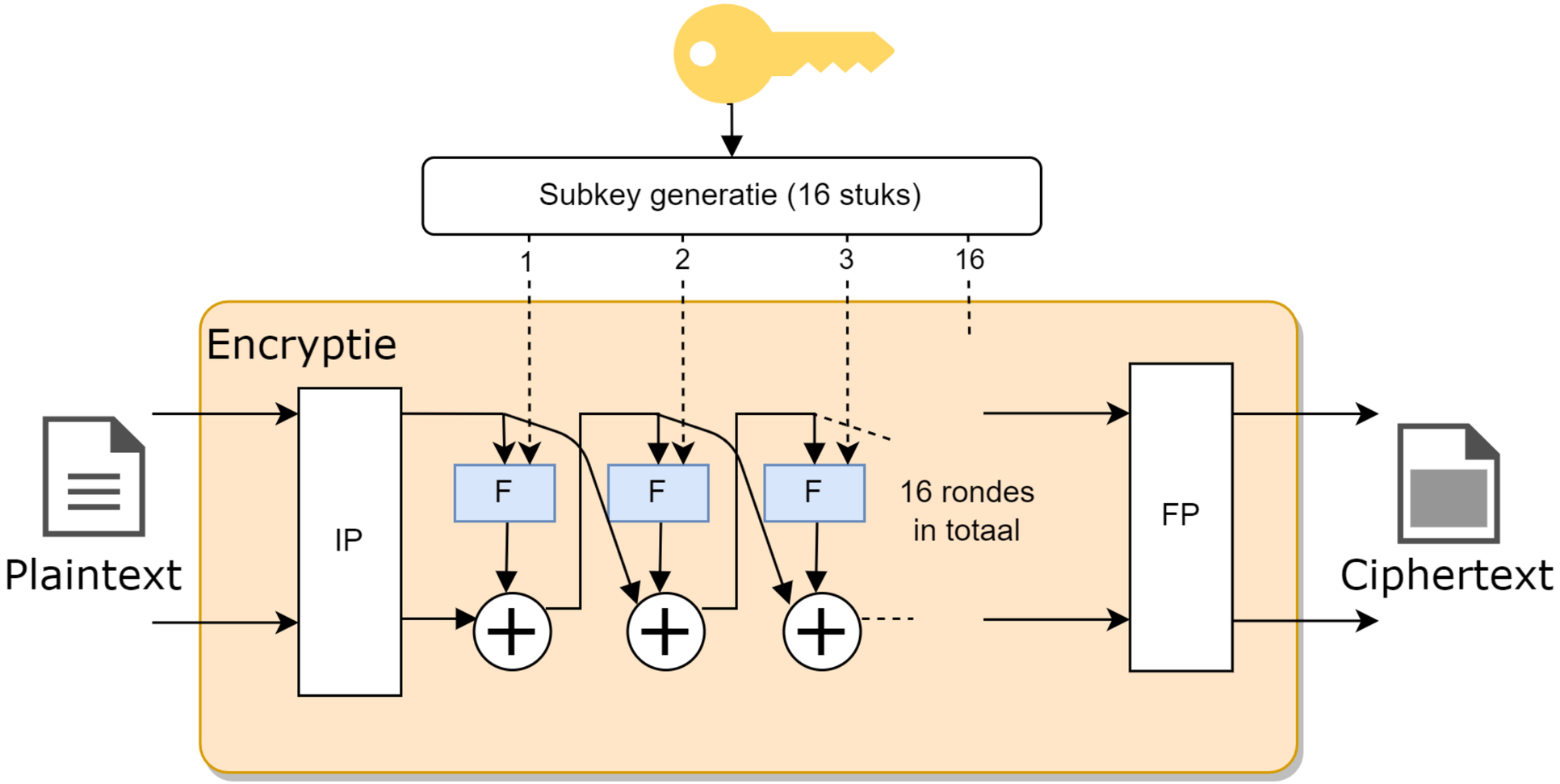
- 8 S-boxen, elk 6 bits in → 4 bits uit
- 2 outer bits = rij, 4 inner bits = kolom

Voorbeeld: **0**11101**1** → 1001

S_5		Middle 4 bits of input														
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

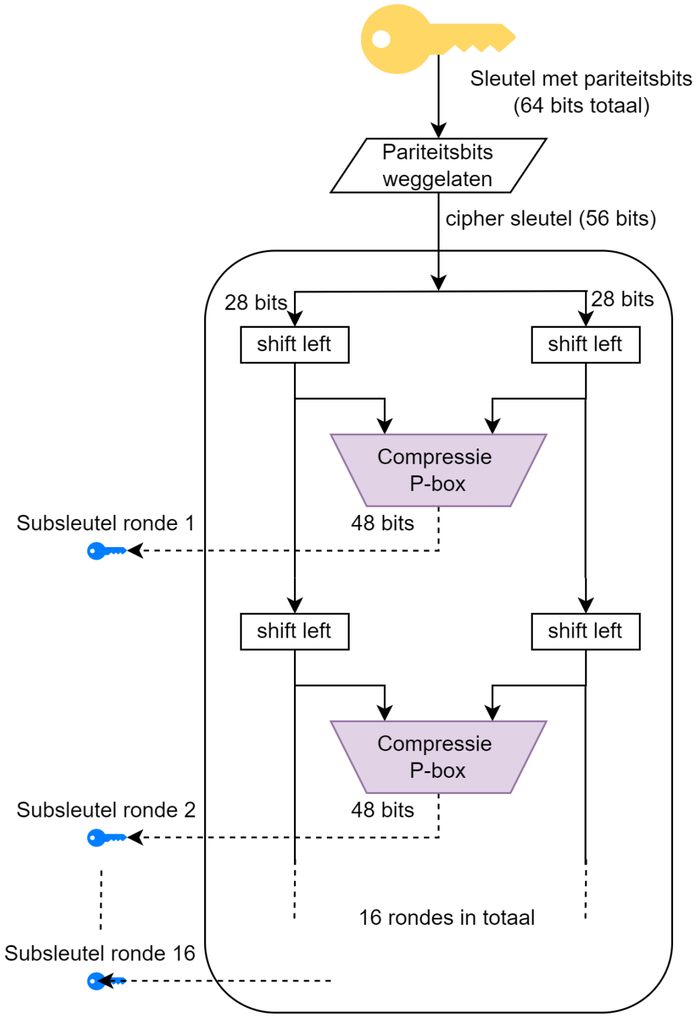
Waarheidstabel van het S5-blok (Bron: Wikipedia).

DES


Subkey generatie (16 stuks)
1
2
3
16
Encryptie
Plaintext
IP
F
F
F
16 rondes
in totaal
FP
Ciphertext

DES: subkeys maken

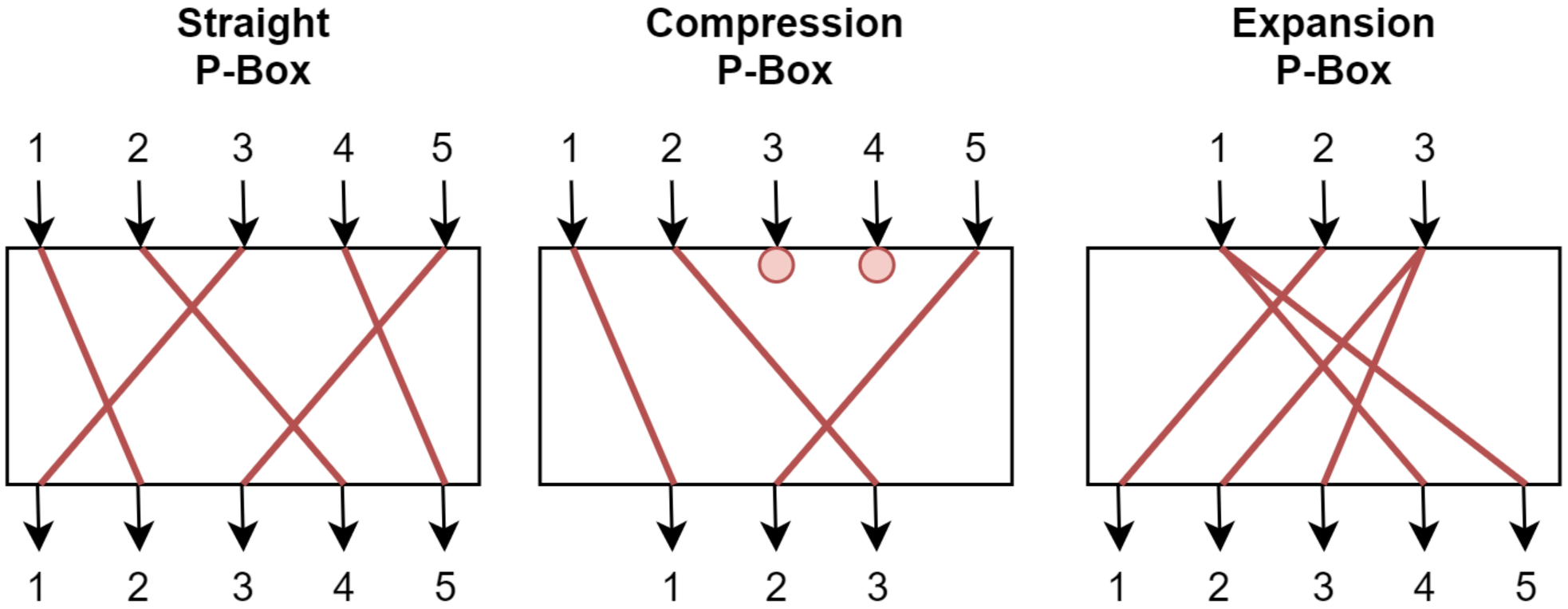
- 56-bit sleutel → splitsen in 2×28 bits
- Per ronde: **bit-shift** + **compressie P-box**
  → 48-bit subkey
- 16 rondes = 16 unieke subkeys



De 16 rondes die telkens 1 subkey maken.

DES: P-boxen

- **Compressie**: 56 bits → 48 bits (bits vallen weg)
- **Permutatie**: bits van plek veranderen



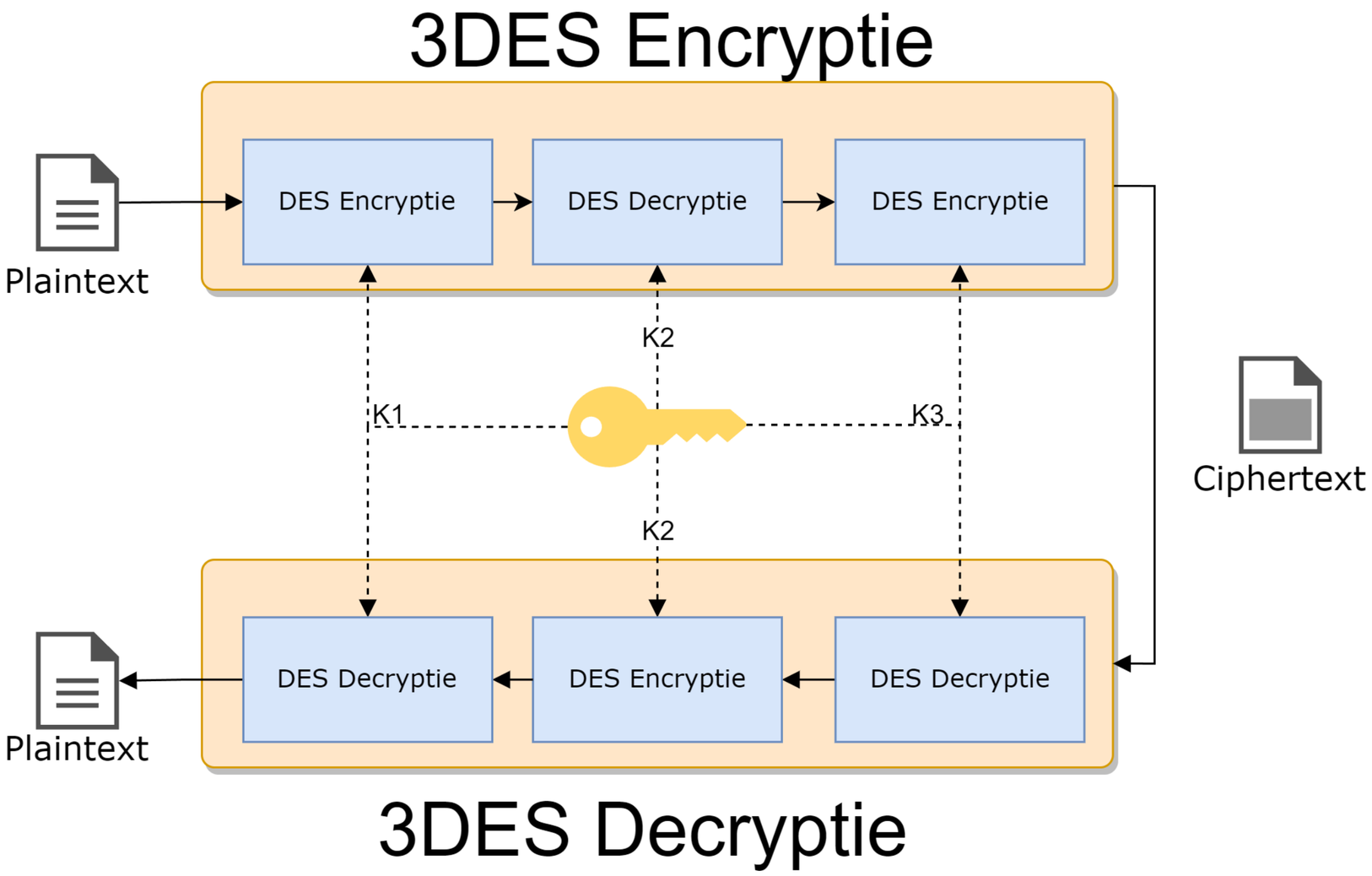
Drie types P-Boxes.

3DES

- DES sleutel (56 bit) te kort → 3DES (1995)
- Sleutel tot **168 bits**, compatibel met bestaande DES hardware
- 3x DES: **Encrypt** → **Decrypt** → **Encrypt** (met 3 verschillende sleutels)
- Verlengde de levensduur van DES, maar is nu ook verouderd

> **Opmerking**
>
> Het bankwezen gebruikt 3DES nog steeds voor o.a. Visa en Mastercard transacties.

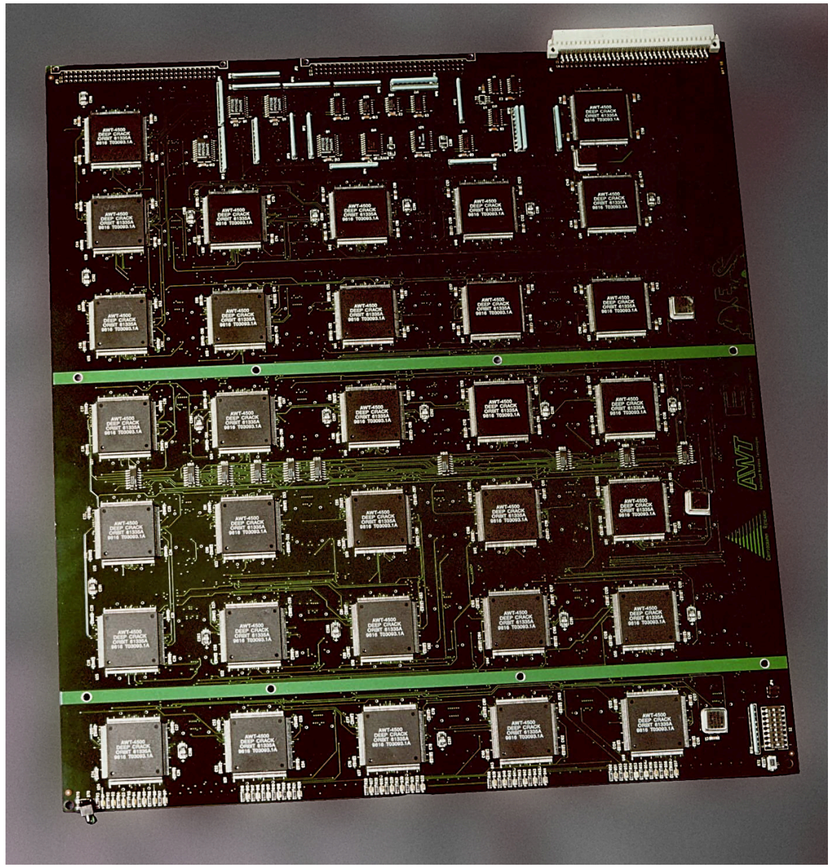
“Tripple DES”



3DES.

DES gekraakt in de praktijk

- **1998**: *Electronic Frontier Foundation* bouwt **Deep Crack**
  - → Kostprijs: **$250.000**
  - → **1.856 custom chips**
  - → DES-sleutel gekraakt in **dagen**
  - → Later (+distributed.net): **< 24 uur**
- DES is **dood** voor gevoelige data
- Bevestigt 3DES (1995) als juiste keuze



Deep Crack board (64 van de 1.856 chips).
Bron: EFF, CC-BY 3.0.

Opmerking

Block cipher modes

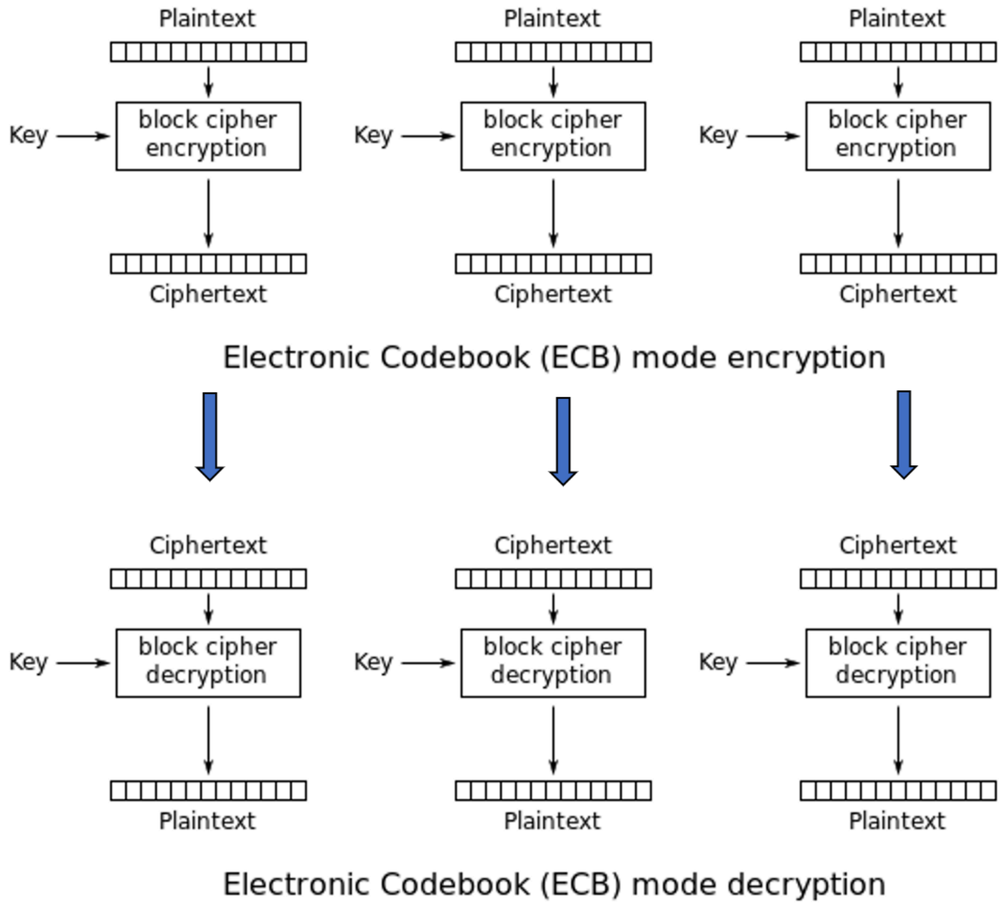
- Hoe decryptie toepassen op data groter dan 1 blok?
- Zelfde plaintext-blok + zelfde sleutel = **zelfde ciphertext**
- Lekt informatie! Maakt **replay attacks** mogelijk
- Verschillende **modes of operation** bedenkt om dit probleem te verhelpen.

Block cipher modes: overzicht

Mode	Werking
ECB	Blokken onafhankelijk (onveilig!)
CBC	Output vorig blok als extra input
CFB	Vergelijkbaar met CBC
OFB	Blockcipher als streamcipher
CTR	Teller (IV) als extra input (veel gebruikt: WPA2, IPSEC)

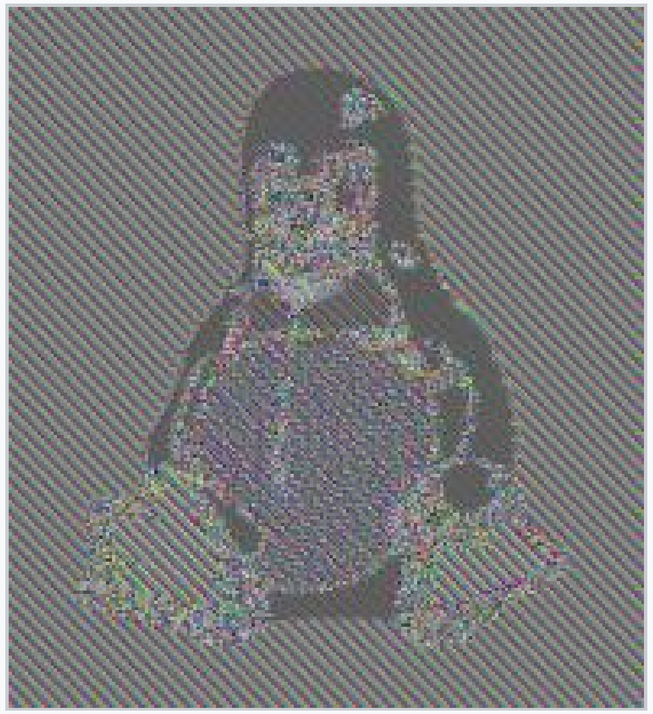
Block cipher modes: ECB

Electronic Codebook (ECB): elk blok onafhankelijk encrypteren

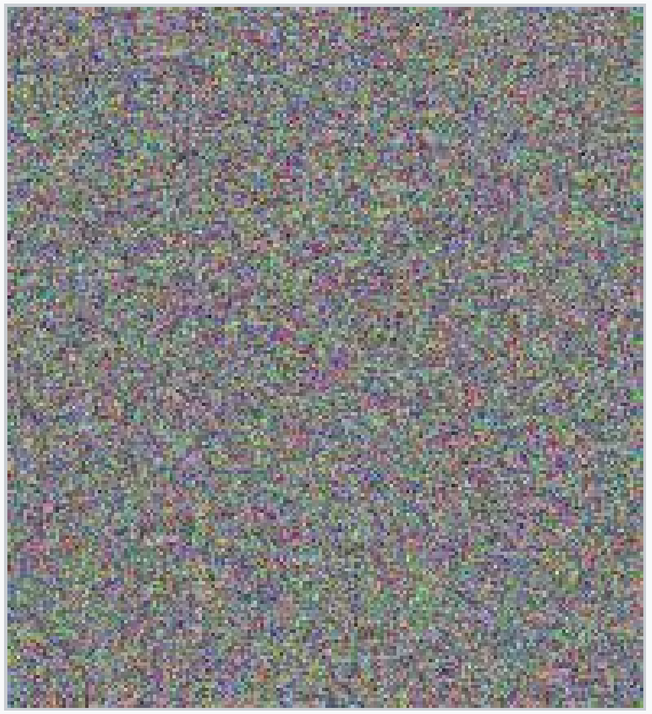


Probleem met ECB

Original image



Encrypted using ECB mode

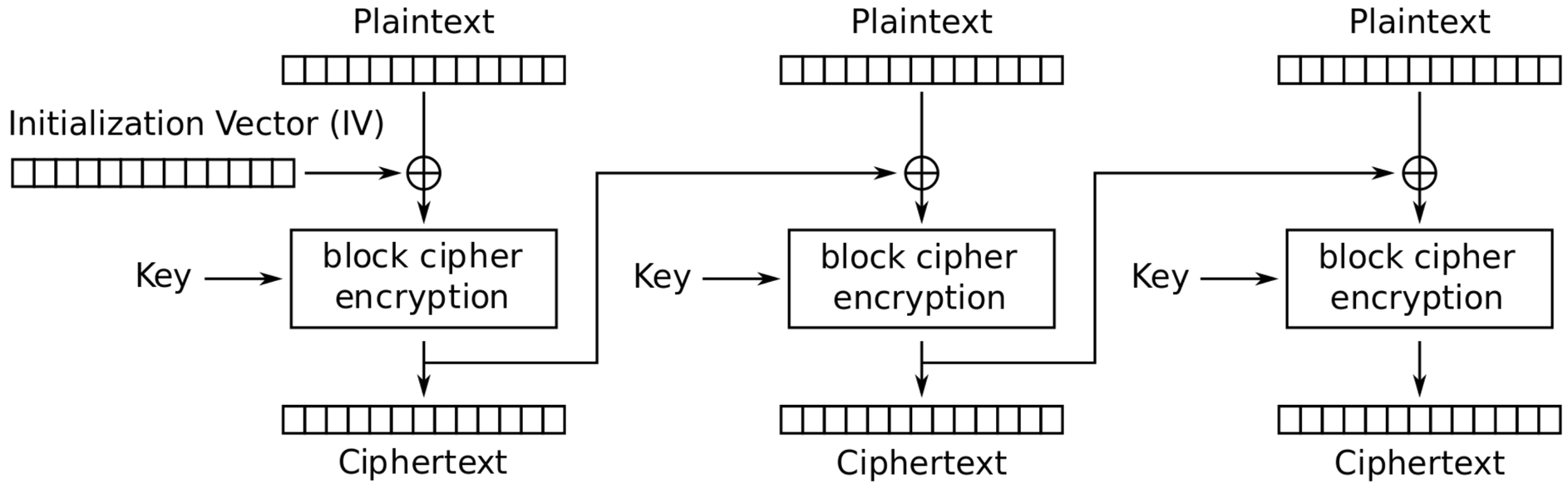


Modes other than ECB result in pseudo-randomness

ECB lekt patronen (Bron: Wikipedia).

Block cipher modes: CBC

Cipher Block Chaining: output van vorig blok als extra input voor volgende blok



Cipher Block Chaining (CBC) mode encryption

CBC encryptie (Bron: Wikipedia).

CBC werkt met een “Initialisatie Vector” (IV)

- **Extra seed** naast de sleutel → effectief steeds andere sleutel
- Wordt **ongeëncrypteerd** meegestuurd (in header)
- Goede IV-selectie algoritme is cruciaal

CBC uitgewerkt

Mini-cipher (2 bits/blok): `00→01, 01→10, 10→11, 11→00`

Plaintext: `00 01 10 11`, IV: `10`

Blok	Plaintext	XOR met	Resultaat	Cipher →
1	`00`	IV=`10`	`10`	**`11`**
2	`01`	`11`	`10`	**`11`**
3	`10`	`11`	`01`	**`10`**
4	`11`	`10`	`01`	**`10`**

Ciphertext: `11 11 10 10`

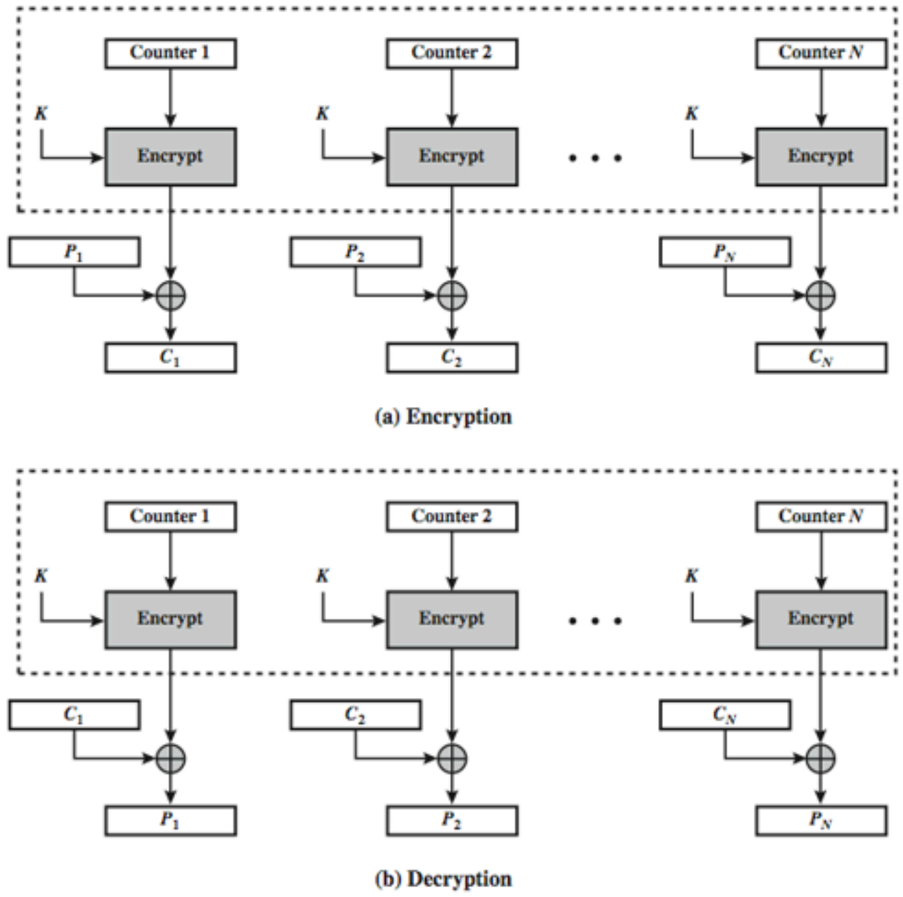
> **Tip**
>
> 4 **verschillende** plaintext-blokken → toch herhalingen in ciphertext. Patronen zijn **doorgeknipt**.

Block cipher modes: CTR

- **Counter (CTR)**: teller (IV) als extra input voor blockcipher (“we encrypten de teller”)

  → Nuttig voor parallelle verwerking (encryptie van blokken onafhankelijk van elkaar)

  → Gebruikt in WPA2, IPSEC, … (bv Netflix streaming)



CTR mode (Bron: Wikipedia).

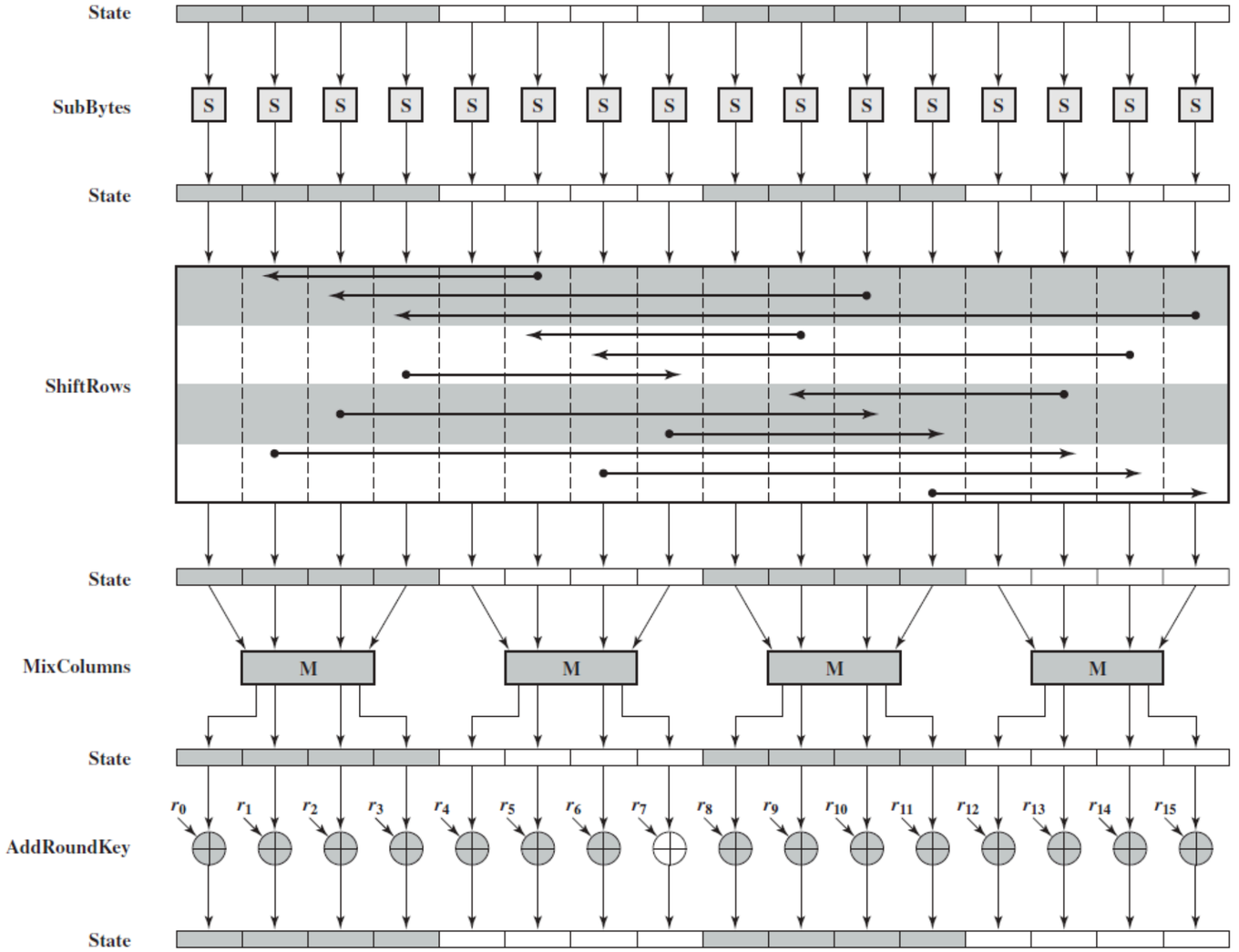
AES (Advanced Encryption Standard)

- Meest recente en meest gebruikte symmetric block cipher
- **2001**: nieuwe wereldwijde standaard
- Gebaseerd op **Rijndael** (Belgisch! Door Vincent Rijmen & Joan Daemen)
- Blokken van **128 bits**, sleutels tot **256 bits**

AES: werking in het kort

- **Key expansie**: unieke sleutel per ronde
- **Meerdere rondes** met:
  - → Substituties (bytes)
  - → Transposities (rijen/kolommen)
  - → XOR met round key
- XOR = wederom de feitelijke encryptie

AES encryptie



(Bron: Wikipedia).

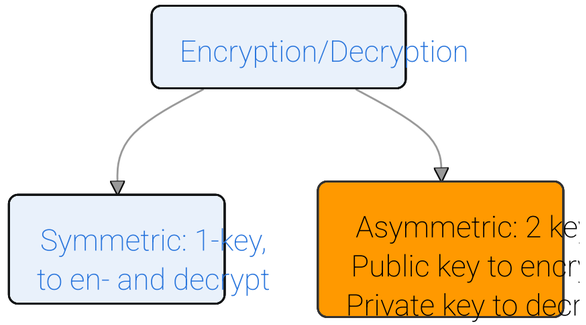
Conclusie

- **Cryptografie** beschermt vertrouwelijkheid (CIA) via encryptie met geheime sleutels
- **Kerckhoffs' principe**: de sleutel is het geheim, niet het algoritme
- **Symmetrische encryptie**: één sleutel, snel, maar sleuteluitwisseling is moeilijk
- **Stream-** (RC4) vs **blockciphers** (DES, AES) met verschillende modes (ECB, CBC, CTR)
- **AES** is de huidige standaard — Belgisch en ongebroken!

Belangrijk

Langere sleutels + sterke algoritmes = betere bescherming. Maar sleuteluitwisseling blijft het grote probleem → **asymmetrische crypto** biedt de oplossing!

Up next?


Encryption/Decryption
Symmetric: 1-key,
to en- and decrypt
Asymmetric: 2 ke
Public key to encr
Private key to decr