

Cryptografie

Cyberboswachters

Tim Dams

Deze cursus werd gemaakt met hulp van AI-assistentie. Sommige afbeeldingen zijn door AI gegenereerd.





AI GENERATED

Wat leer je in dit hoofdstuk?

Na dit hoofdstuk kan je:

- ✓ het principe van **Kerckhoffs** uitleggen
- ✓ symmetrisch vs asymmetrisch duiden (snelheid ↔ sleuteldistributie)
- ✓ voor ECB/CBC/CTR uitleggen waarom **ECB onveilig** is + de rol van IV/nonce
- ✓ hashing, digitale handtekeningen en certificaten situeren
- ✓ de werking van HTTPS/TLS beschrijven + de impact van een MITM-aanval inschatten

CRYPTOGRAFIE

Informatie versleutelen zodat **enkel zender en ontvanger** het bericht kunnen lezen.

Meer weten? Lees **The Codebreakers** van David Kahn: 1200 pagina's die lezen als een thriller.

CIA



Confidentiality

Enkel leesbaar voor
geautoriseerde partijen.

Focus van dit hoofdstuk, via
encryptie.



Integrity

Niet gewijzigd tijdens de
overdracht.



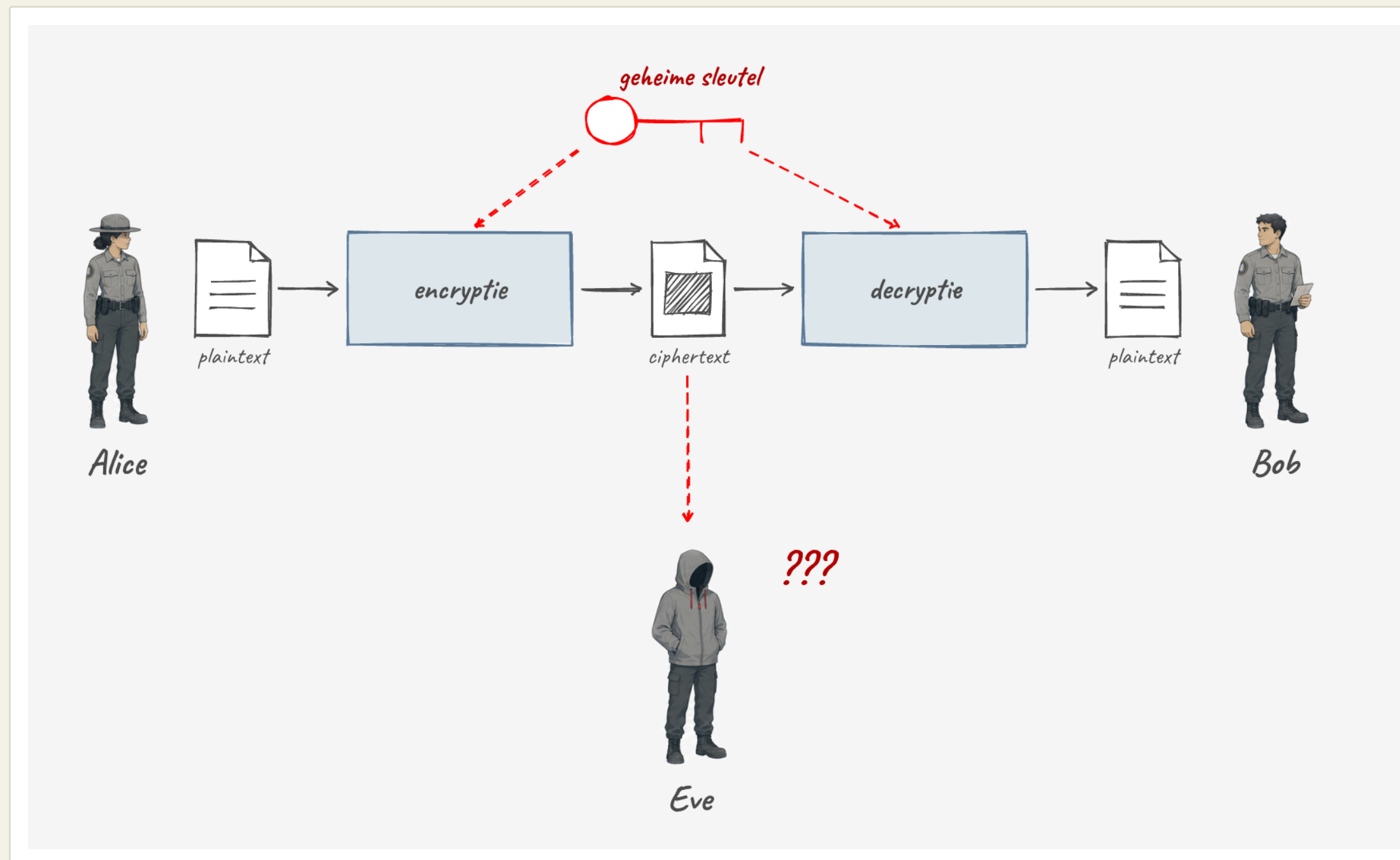
Availability

Toegankelijk wanneer nodig.

Ook **integrity** los je vaak op met cryptografische technieken, zoals hashing en digitale handtekeningen.

Encryptie en decryptie

Data versleutelen (**encrypteren**) met een **geheime sleutel**. Zonder sleutel is de data onleesbaar.



Enkele termen

Plaintext	de originele data
Ciphertext	de versleutelde data
Cryptanalyse	ontcijferen zonder de sleutel
Cryptografie	de wetenschap van het versleutelen
Cryptologie	cryptografie + cryptanalyse samen

ENCRYPTIE DOORHEEN DE TIJD

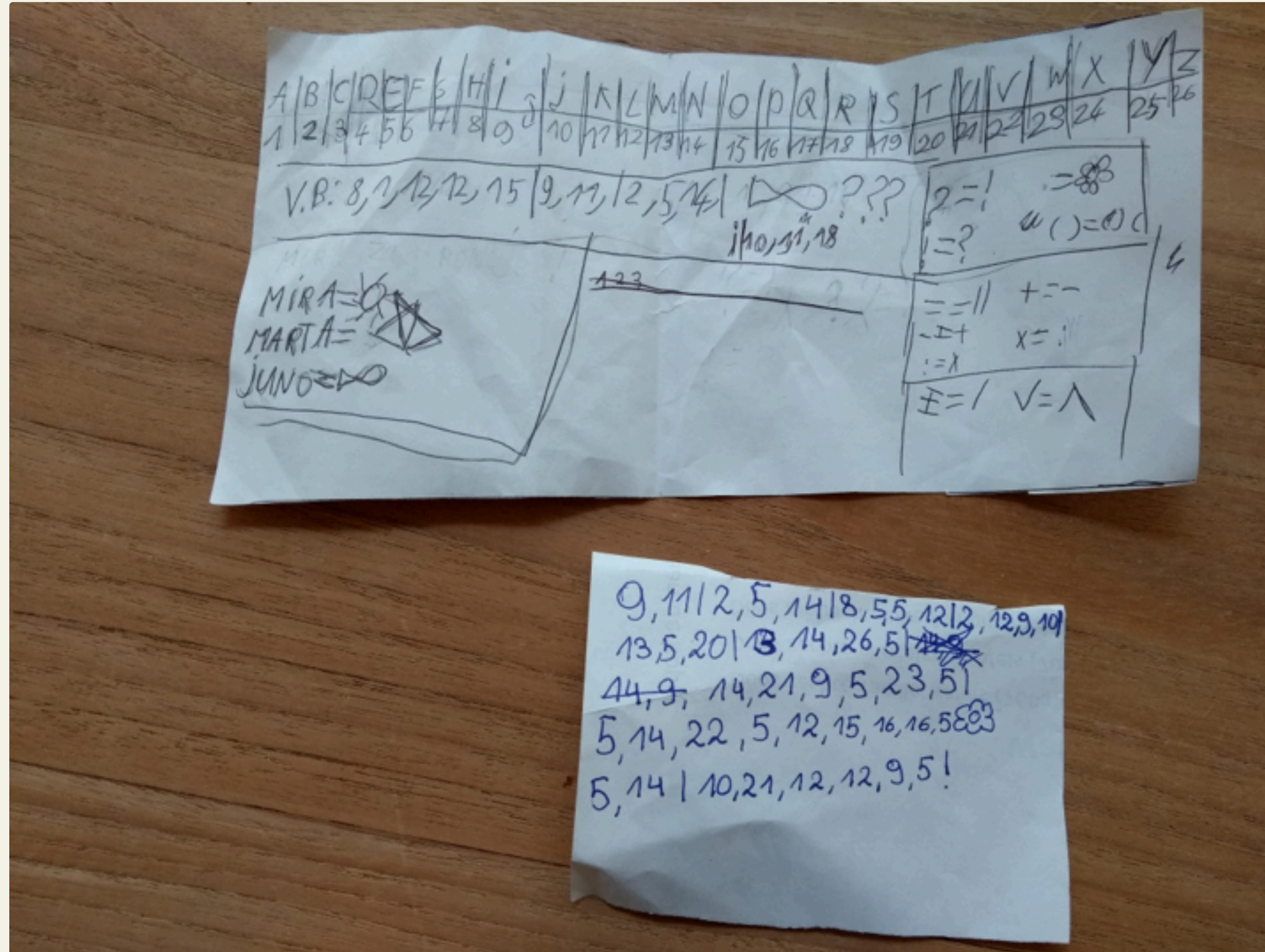
Hoe sterker de computers, hoe **sterker** de algoritmes moeten zijn.

Al sinds de **Romeinen** wordt er aan cryptografie gedaan: een wedloop tussen stropers en boswachters.

Crypto volgens m'n kids



Crypto volgens m'n kids, deel 2



Classificatie van cryptosystemen



Actie op de data

- **Substitutie:** teken door een ander teken vervangen
- **Transpositie:** teken op een andere plek zetten
- **Product:** combinatie van beide



Aantal sleutels

- **1 sleutel:** symmetrisch (*secret key encryption*)
- **2 sleutels:** asymmetrisch (*public key encryption*)



Verwerking

- **Blok:** blok per blok
- **Stream:** teken per teken



August Kerckhoffs

“

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

— Principe van Kerckhoffs, 19e eeuw

Het algoritme mag publiek zijn

Publiek algoritme

- Getest door **duizenden experts**
- Kans op (bewuste) fouten veel kleiner
- De meest gebruikte algoritmes zijn publiek

vs

Security through obscurity

- Geheim, *proprietary* systeem
- Geheimhouden is geen garantie op veiligheid
- **Vals gevoel** van veiligheid



Alles draait om het geheimhouden van je **sleutel.**

Sleutel == wachtwoord. Zonder de sleutel mag niemand aan de data kunnen, ook al weet iedereen welk algoritme je gebruikt.

SLEUTELS KRAKEN

Cryptanalyse

Ontcijferen zonder de sleutel te kennen.

BRUTEFORCE

Gewoon **alle** mogelijke sleutels testen.

Gemiddeld vind je de sleutel al in de **helft** van de maximale tijd: de eerste sleutel die je test, kan evengoed de juiste zijn als de laatste.

Hoe lang duurt bruteforcen?

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

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

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

~29 uur

gemiddeld gevonden

62 dagen

maximaal, met **één teken extra**

Bruteforce: de tabel

# 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

Ter vergelijking: het universum is $1,38 \times 10^{10}$ jaar oud. Per verdubbeling van het aantal GPU's halveert de tijd.

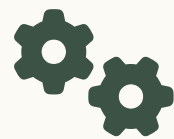
Dictionary attack

Bruteforce slimmer maken: test sleutels uit een **woordenboek**.



Woordenboek

De meest gebruikte wachtwoorden: `123456`, `password`, `12345678`, `qwerty`, ...



Variaties

Tools zoals **John The Ripper** testen ook *god1*, *god2*, ...



Nooit gebruiken

Wachtwoorden uit bekende lijsten, of varianten ervan.

Check zelf: [SecLists](#)

Soorten cryptanalytische aanvallen

Afhankelijk van de informatie die de cryptanalist heeft:



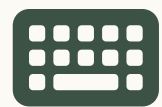
Enkel ciphertext

Alleen versleutelde data. Het **beste scenario** voor cyberboswachters.



Gekende plaintext

Zowel de ciphertext als de bijhorende plaintext zijn gekend.



Gekozen plaintext

De analist kiest de plaintext en ziet de resulterende ciphertext.



Gekozen ciphertext

De analist kiest de ciphertext en ziet de resulterende plaintext.

Hoe kraak je deze kluis?



AI GENERATED

DE BESTE "CRYPTANALYSE" IS GÉÉN CRYPTANALYSE

Vraag de sleutel gewoon aan de gebruiker.

Veel makkelijker dan sleutels kraken: `phishing` `helpdesk-scams`

`post-its onder het toetsenbord`

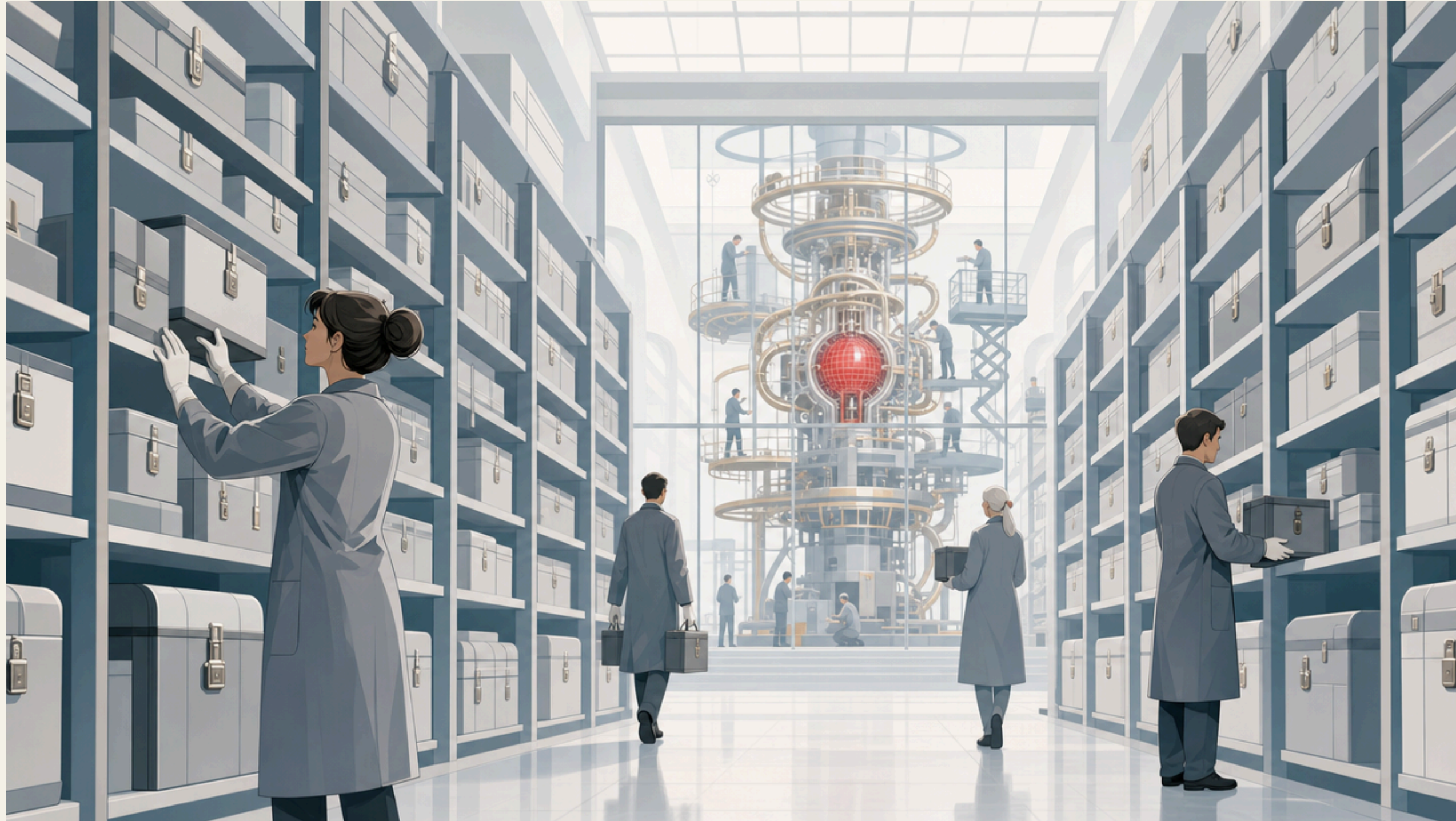
Cryptografie is **noodzakelijk**, maar zelden **voldoende**. Meer hierover in het hoofdstuk **Social Engineering**.

“

There is no patch for human stupidity.

— Vaak gehoord cliché in de security-wereld

Wat ligt hier te wachten?



AI GENERATED

Quantum-computers en crypto



RSA en ECC in gevaar

Met het algoritme van **Shor** kraakt een quantum-computer publieke crypto (RSA, DH, ECC). Voor AES is de winst beperkt: een lange sleutel (AES-256) volstaat.



Store now, decrypt later

Veel veiligheidsdiensten hanteren die strategie al jaren: nu bewaren, later ontcijferen.



Post-quantum

Onderzoek naar **post-quantum cryptografie** is volop bezig.

Impact op cryptocurrencies e-commerce wachtwoorden ...

“

Encryption is everywhere in modern day life - everything will be vulnerable!

— uclftr.com

SUBSTITUTIE EN TRANSPOSITIE

De eerste algoritmes

Onderverdeeld naar actie



Substitutie

Teken door een ander teken
vervangen.

Bv. **Caesar**



Transpositie

Teken op een andere plek
zetten.

Bv. **Scytale**



Combinatie

Substitutie én transpositie.

Bv. **AES**

🔧 Handige tool om crypto te leren: CrypTool

Caesar encryptie

Sleutel = getal **1-25**: iedere letter schuift evenveel plaatsen op. **A** =0, **B** =1, ... **Z** =25



Sleutel 3

A → **D** , **B** → **E** , ... en **Z** → **C** (overflow: terug naar voor)



Rot13

Ook wel **Rot** (rotatie) genoemd: Rot4, Rot13, ...
Rot13 2x toepassen = originele tekst.

Caesar met de modulo-operator

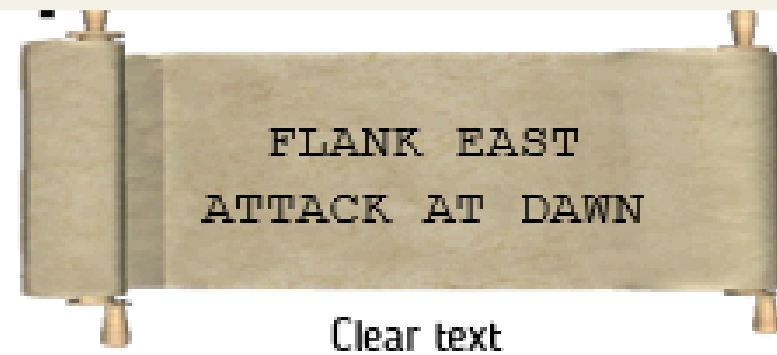
Formule: $(\text{teken} + \text{sleutel}) \% 26 = \text{nieuw teken}$

Voorbeeld: **Y** met sleutel 7



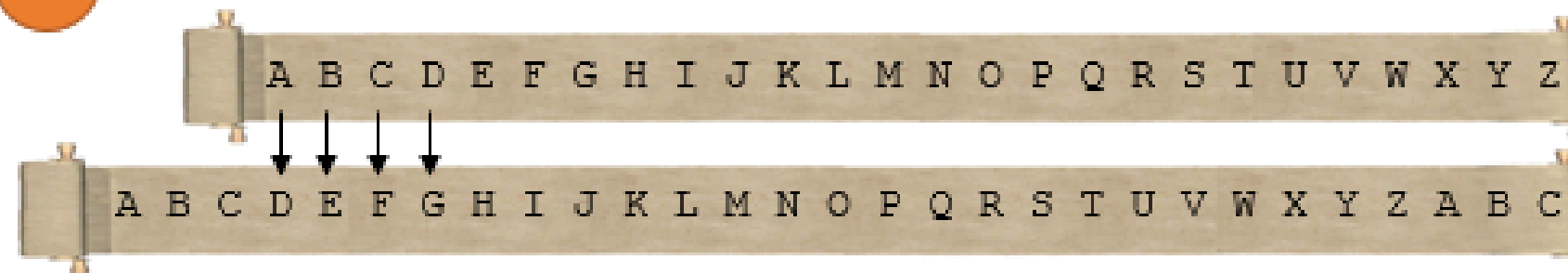
Caesar voorbeeld

1



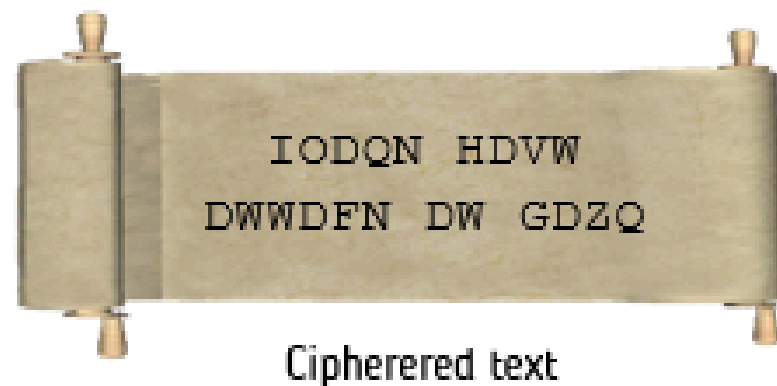
The clear text message would be encoded using a key of 3.

2



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

3



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

Caesar wiel



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



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



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

Andere manier om Caesar encryptie te visualiseren

Caesar kraken



Bruteforce

Slechts **25** mogelijke sleutels.

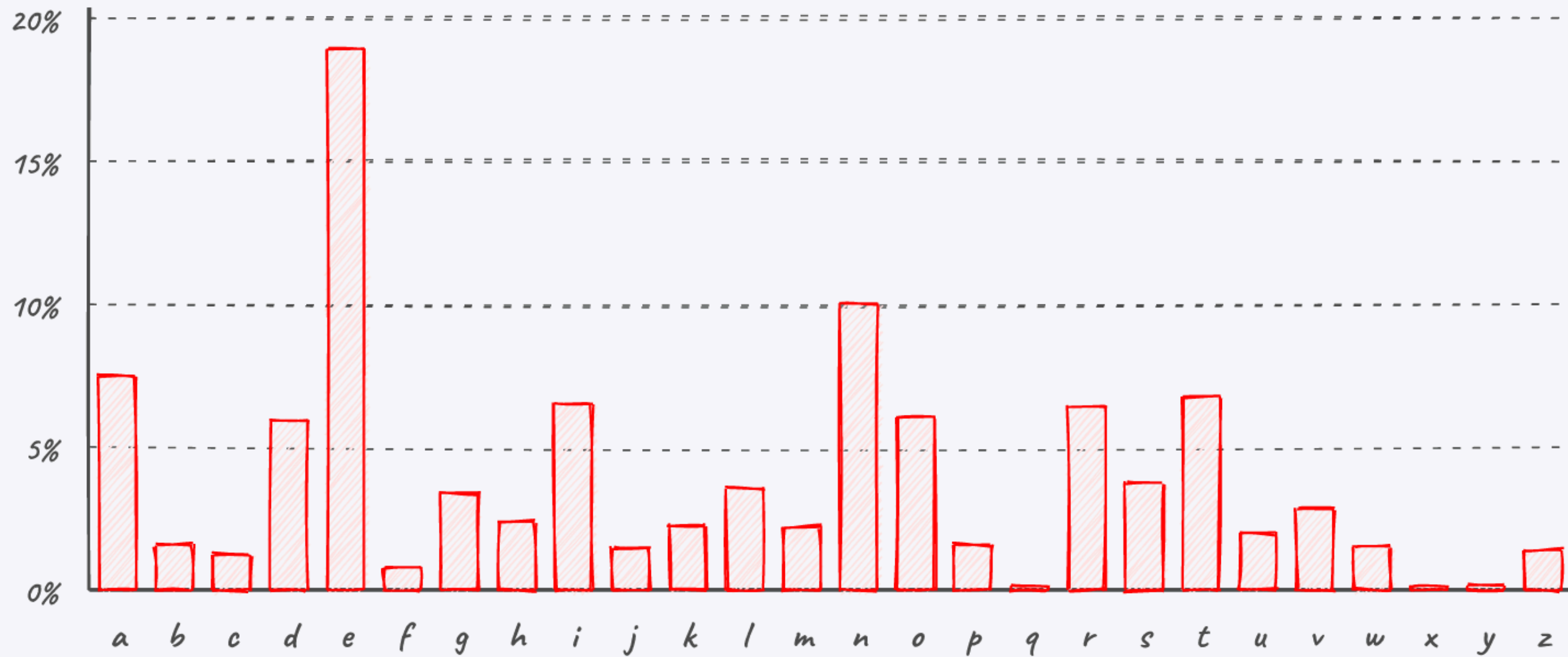


Frequentieanalyse

Statistische verdeling van letters in een taal: de **e** komt in het Nederlands veel vaker voor dan de **v**.

De meest voorkomende letter in de ciphertext is dus waarschijnlijk een **e**.

Frequentieanalyse



Frequentie-analyse Nederlandstalige tekst (cijfers: Wikipedia).

Vigenère

Caesar

- Iedere letter krijgt **dezelfde** shift
- Daardoor werkt frequentieanalyse

vs

Vigenère (16e eeuw)

- Een **sleutelwoord** i.p.v. één getal
- Iedere sleutelletter = shift voor de bijhorende plaintext-letter
- Sleutelwoord herhaald over de tekst: meerdere Caesars na elkaar

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
- De e wordt niet telkens dezelfde letter (R, S, G, S)
- 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

LE CHIFFRE INDÉCHIFFRABLE

Vigenère heette eeuwenlang **onkraakbaar**.

In de 19e eeuw kraakte **Kasiski** het: eens je de sleutellengte kent, heb je meerdere parallelle Caesars.

```
graph LR; A[1. Wikkel het lint rond de stok] --> B[2. Elk volgend teken komt op een andere zijde]; B --> C[3. Lees per zijde uit: dat is de ciphertext];
```

1 Wikkel het **lint** rond de stok

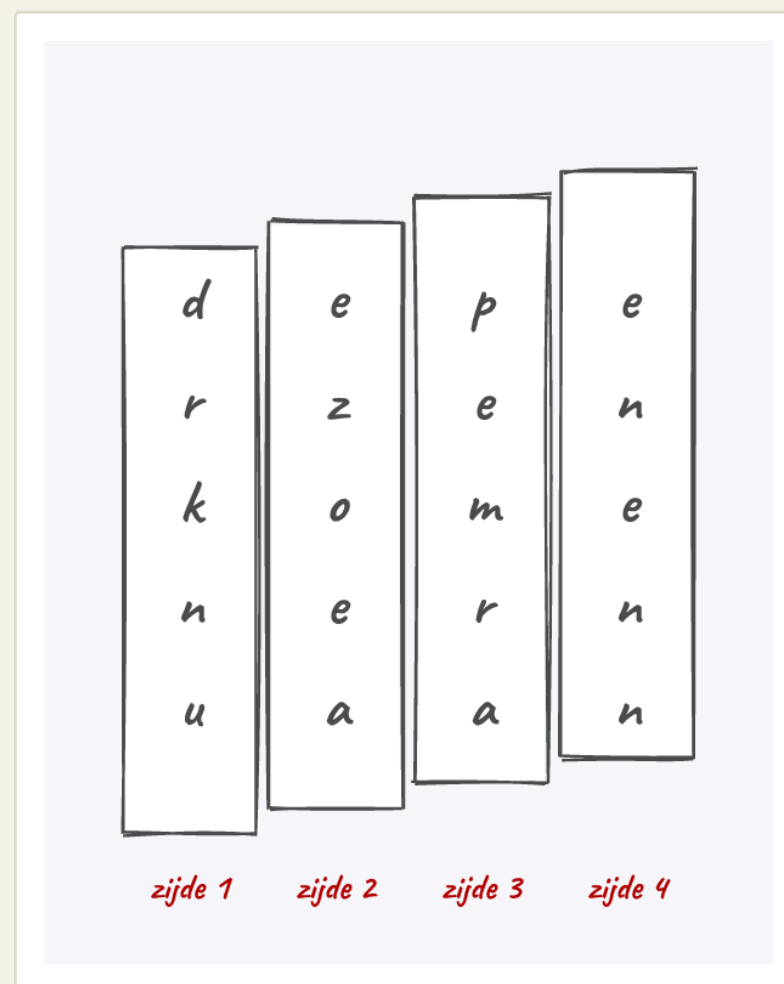
2 Elk volgend teken komt op een **andere zijde**

3 Lees **per zijde** uit: dat is de ciphertext

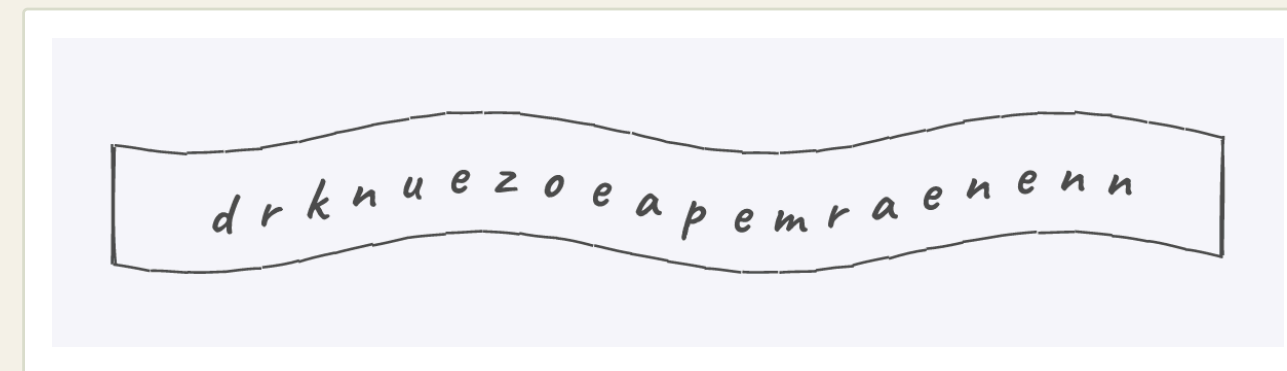


Scytale: voorbeeld

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



Ciphertext na afwikkelen:



COMBINATIE: SUBSTITUTIE + TRANSPOSITIE

Een combinatie **versterkt** de encryptie.

Moderne algoritmen zijn een sequentie van basisvormen. **AES** (de facto standaard): substituties (**sub**), transposities (**shiftrows**) en het mengen van kolommen (**mixcolumns**) in meerdere rondes. Dat zien we later.

ZWAARDER GESCHUT

Moderne cryptosystemen

Klassieke ciphers (Caesar, Vigenère, scytale) zijn ondertussen allemaal gekraakt.

Wat maakt crypto "modern"?



Grote sleutels

128-256 bits: brute force wordt onhaalbaar.



Vele rondes

Substitutie en transpositie, telkens opnieuw.



Publiek getest

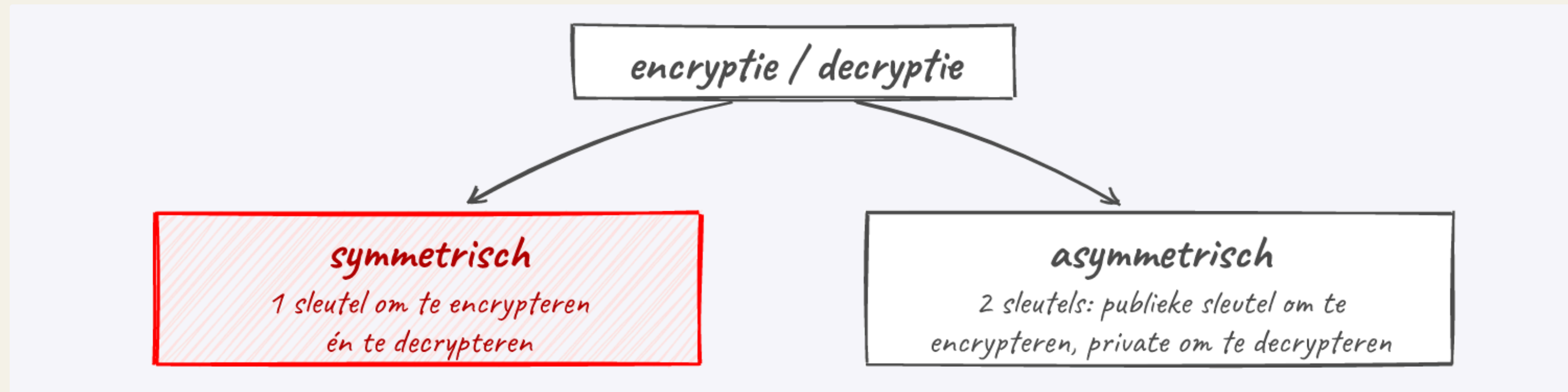
Kerckhoffs in actie: geen *security through obscurity*.



Wisselende subkeys

Per ronde een andere subkey: Vigenère op steroïden.

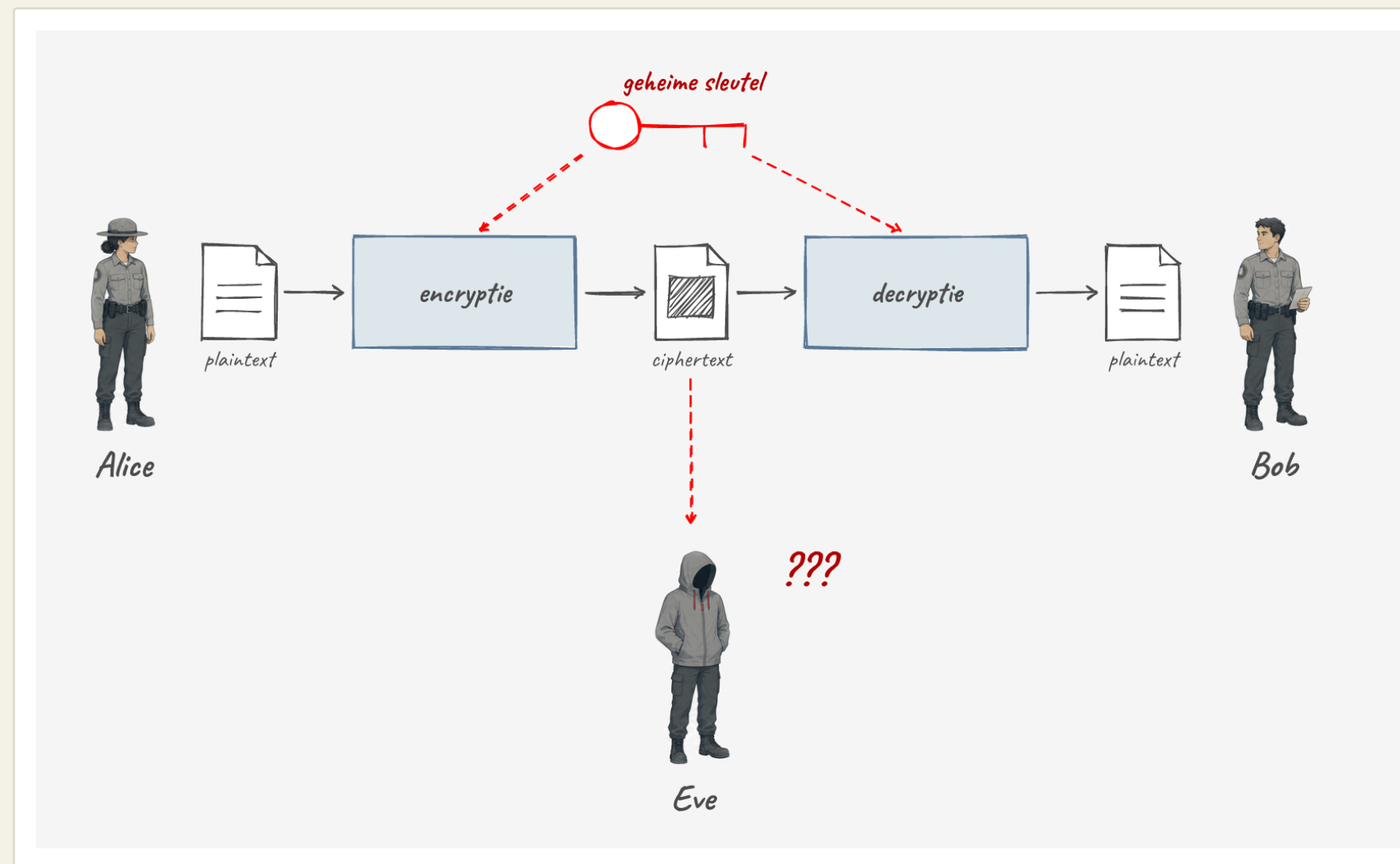
Families volgens aantal sleutels



We starten met **symmetrisch**, de oudste van de twee.

Symmetrische encryptie

Eén sleutel voor encryptie én decryptie: zender en ontvanger gebruiken dezelfde sleutel.



Sleuteloverdracht: het grote probleem

Zender en ontvanger hebben dezelfde sleutel nodig. Hoe wissel je die **veilig** uit?



Via asymmetrische encryptie

Dat systeem kan wél veilig sleutels uitwisselen (zie later).



Via een ander beveiligd kanaal

Fysiek afgeven, zeggen, via een ander kanaal,
...

Block- en streamciphers

Streamcipher

- **Teken per teken**
- Sneller
- Bv. RC4

vs

Blockcipher

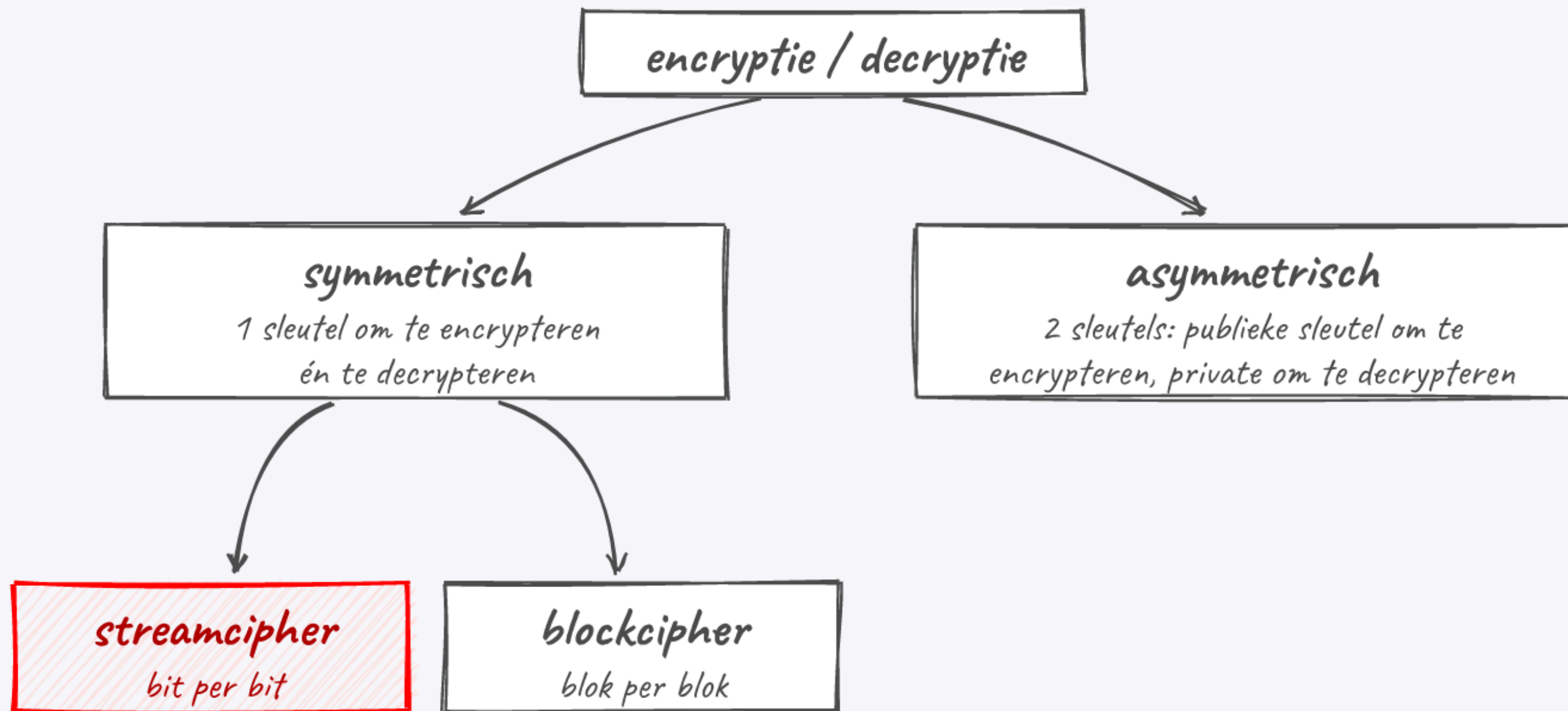
- **Blok per blok** (bv. 128 bits)
- Trager
- Bv. AES, DES, 3DES

SYMMETRISCH

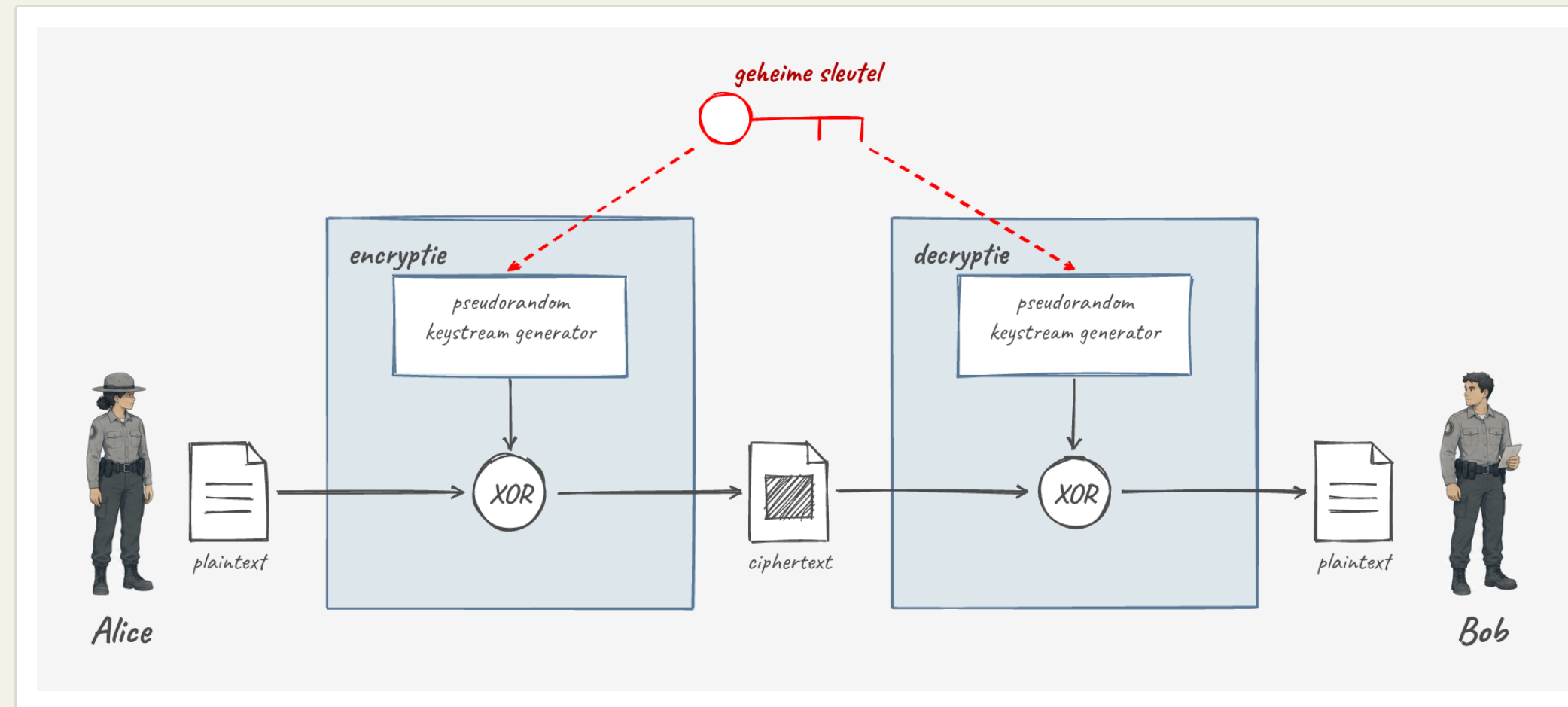
Streamciphers

Teken per teken versleutelen.

Waar staan we?



Streamciphers: werking



Keystream generator expandeert de sleutel naar een **keystream**, even lang als de data

XOR-functie $\text{plaintext} \oplus \text{keystream} = \text{ciphertext}$

BELANGRIJK INZICHT

Zelfde sleutel → zelfde keystream → originele plaintext.

Enkel een ontvanger met dezelfde sleutel kan het bericht dus decrypteren.

De XOR-functie: hart van cryptografie!

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

Encrypteren

$$1010 \oplus 1101 = 0111$$

Dezelfde XOR met de keystream (**1101**)
decrypteert weer:

$$0111 \oplus 1101 = 1010$$

Keystream generator

Genereert een **pseudo-random** keystream, gebaseerd op een PRNG (Pseudo-Random Number Generator).



Seed

Zelfde sleutel (**seed**) → zelfde reeks. Reproduceerbaar.



Onvoorspelbaar

Voor buitenstaanders is de reeks niet te voorspellen.



Zwakke generator

→ **kwetsbare encryptie** (bv. WEP!)

Middle-square: een van de eerste PRNG's



Voorbeeld (seed = 1111):

```
11112 = 01 2343 21 → 2343
23432 = 05 4896 49 → 4896
48962 = 23 9708 16 → 9708
```

Reeks: 2343, 4896, 9708, ...

Ondertussen onveilig (korte cycli), maar de basis van alle moderne PRNG's.

PRNG in de praktijk: Random() in C#

```
1 int key = 666;  
2  
3 Random s = new Random(key); // Zender  
4 Random r = new Random(key); // Ontvanger  
5 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**. Een van de meest gebruikte streamciphers ooit:

WEP

SSL/TLS

40-2048 bits

sleutellengte

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

RC4 in actie

1

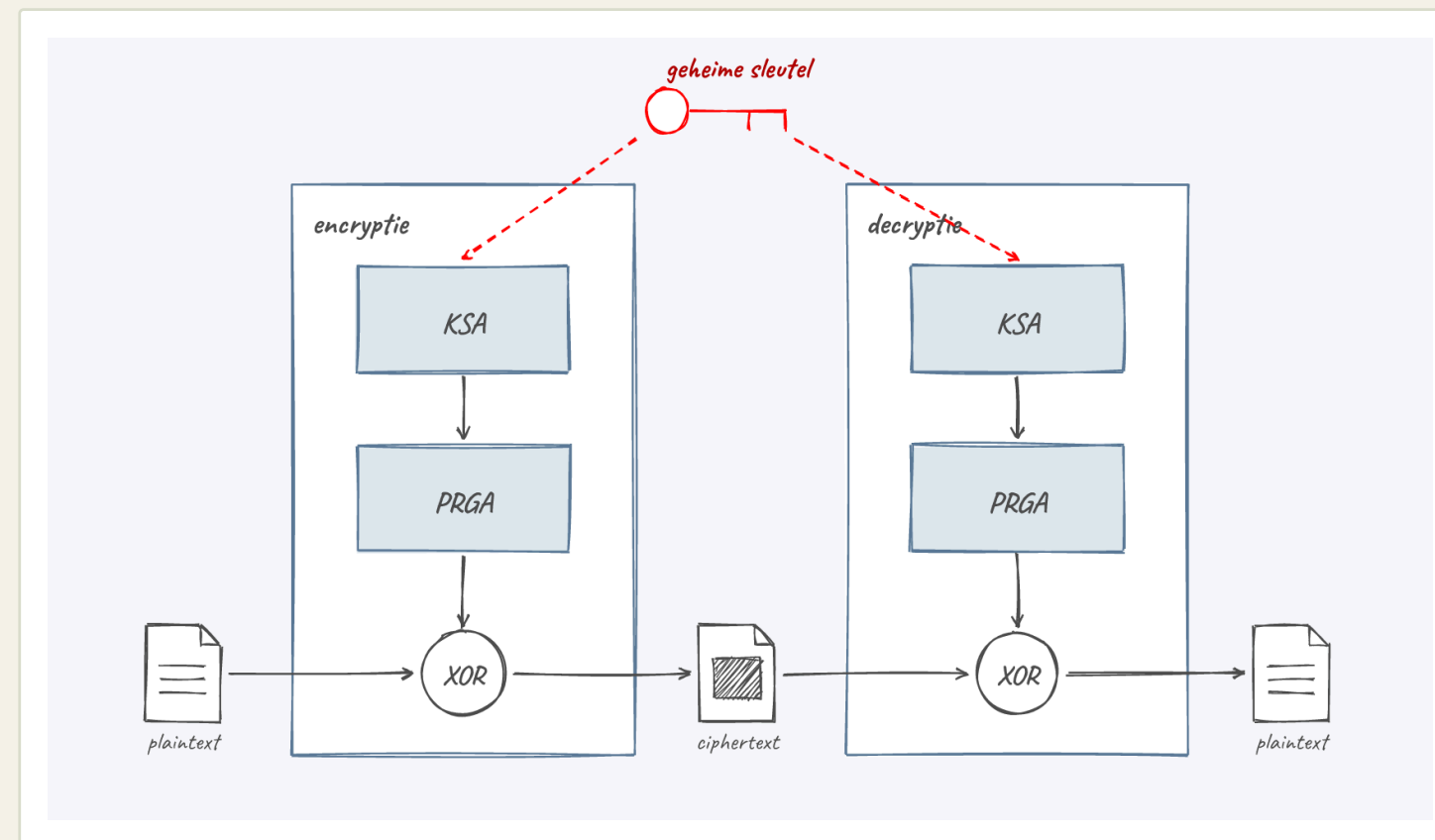
KSA (Key Scheduling Algorithm): sleutel → werksleutel

2

PRGA (Pseudo-Random Generation Algorithm): werksleutel → keystream

3

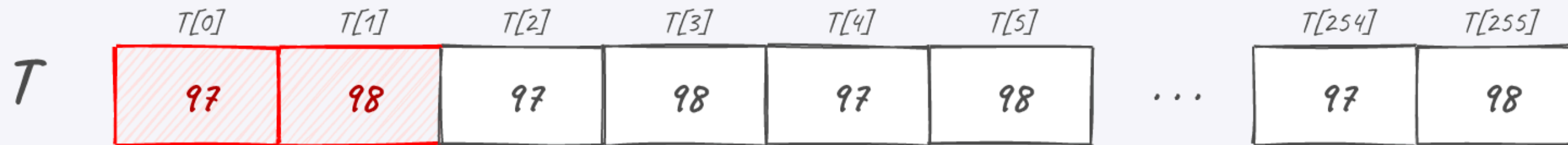
XOR van de keystream met de plaintext



Stap 1 en 2: de arrays T en S

Stap 1: zet de sleutel in array T en herhaal hem tot alle 256 plaatsen vol zijn

 sleutel "ab": $a = 97$ en $b = 98$ (ASCII)



Stap 2: maak array S met de waarden 0 tot en met 255



Stap 3: S door elkaar schudden

```
j = 0  
for i = 0 tot 255
```

```
{
```

```
  j = (j + S[i] + T[i]) % 256
```

```
  Verwissel(S[i], S[j])
```

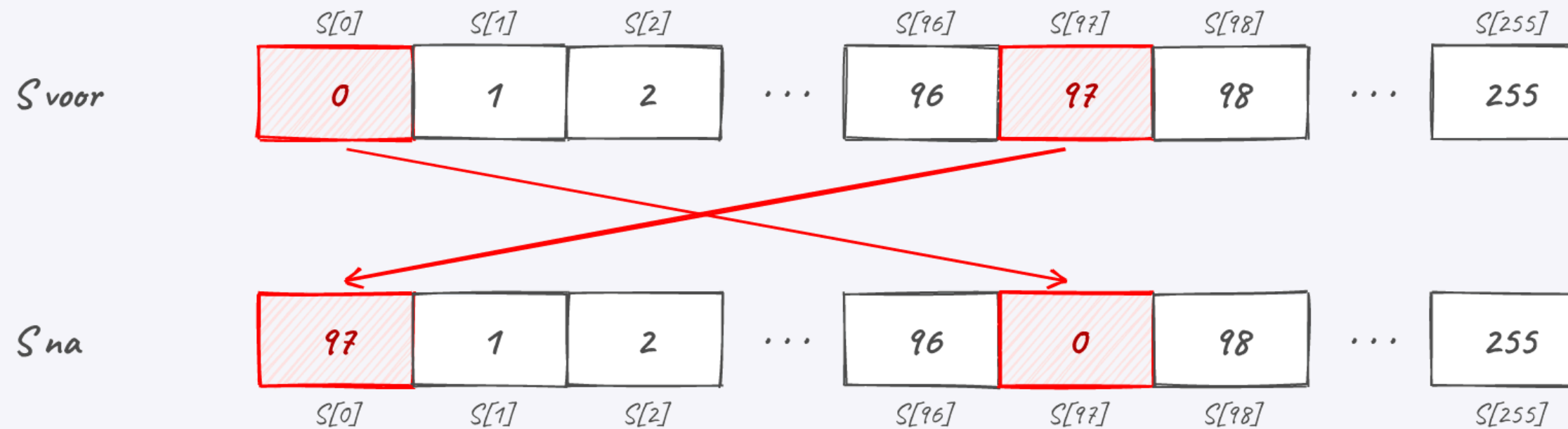
```
}
```

Eerste iteratie: i = 0

$$j = (0 + S[0] + T[0]) \% 256$$
$$= (0 + 0 + 97) \% 256 = 97$$

Verwissel(S[0], S[97])

en dat nog 255 keer, voor i = 1 tot en met 255



Het resultaat: de werksleutel S

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

sleutel "ab"

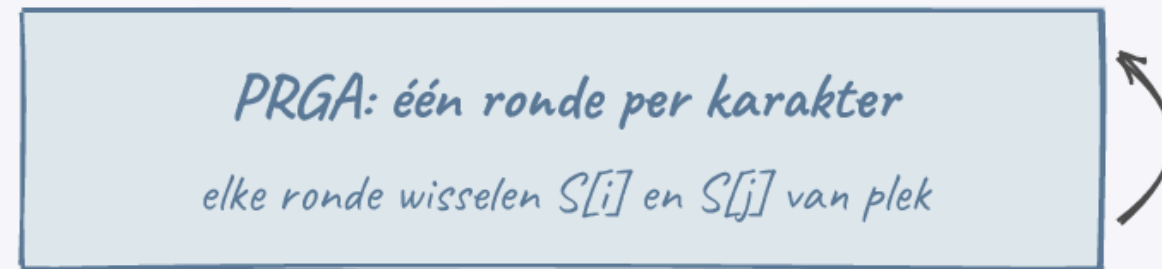
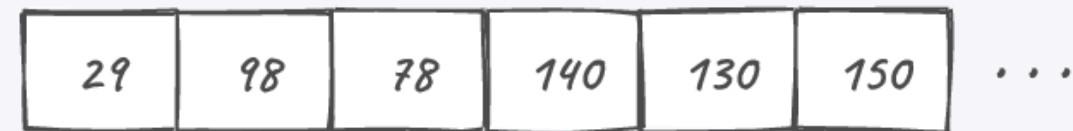
elk getal van
0 tot en met 255
staat er één keer
in, op een
willekeurige plek

andere sleutel:
totaal andere S

Van werksleutel naar keystream

```
i = 0
j = 0
Herhaal telkens keystream karakter nodig is
{
    i = (i+1) % 256
    j = (j + S[i]) % 256
    Verwissel(S[i], S[j])
    k = S[(S[i] + S[j]) % 256]
    output k naar keystream
}
```

S uit de KSA



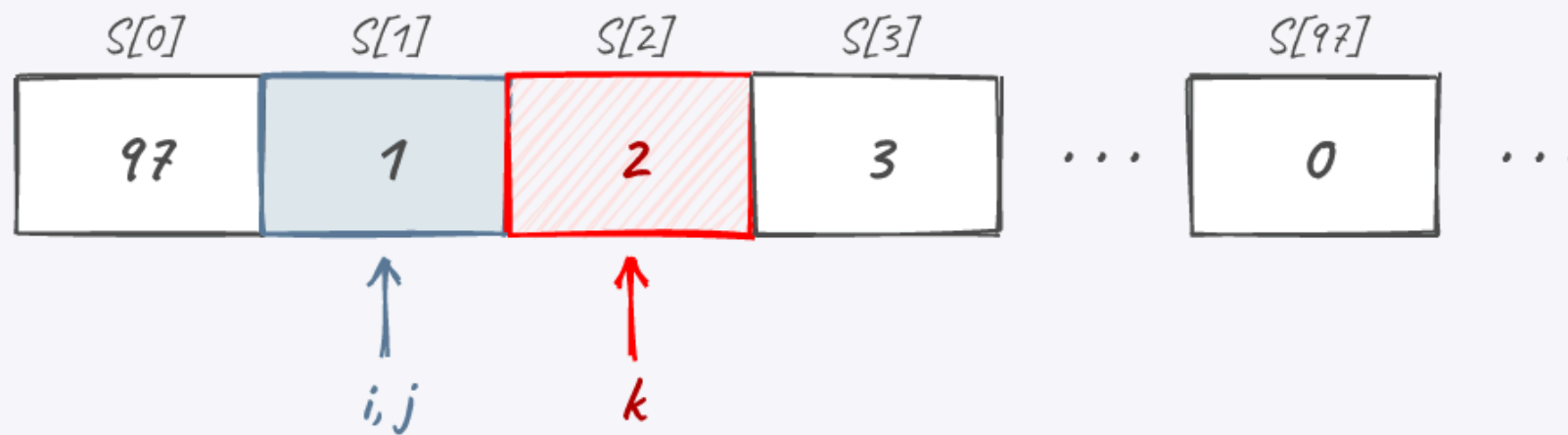
keystream



elke k is één waarde tussen 0 en 255

Uitgewerkt voorbeeld

S na de eerste KSA-iteratie (sleutel "ab"), start met $i = 0$ en $j = 0$



$$i = (0 + 1) \% 256 = 1$$

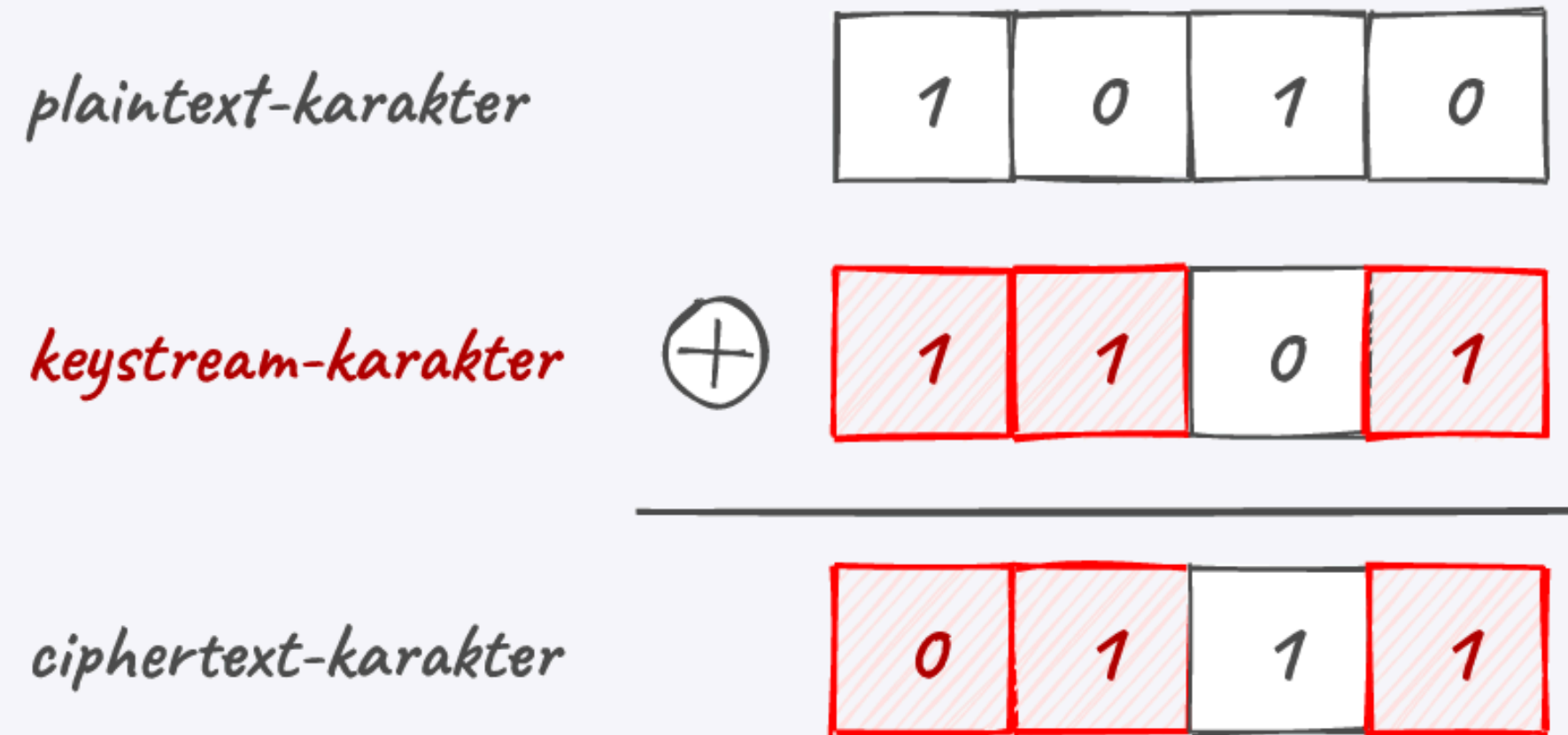
$$j = (0 + S[1]) \% 256 = (0 + 1) \% 256 = 1$$

Verwissel($S[1]$, $S[1]$): geen wissel, want i en j vallen samen

$$k = S[(S[1] + S[1]) \% 256] = S[2] = 2$$

*eerste keystream-karakter
 $k = 2$*

Keystream XOR plaintext



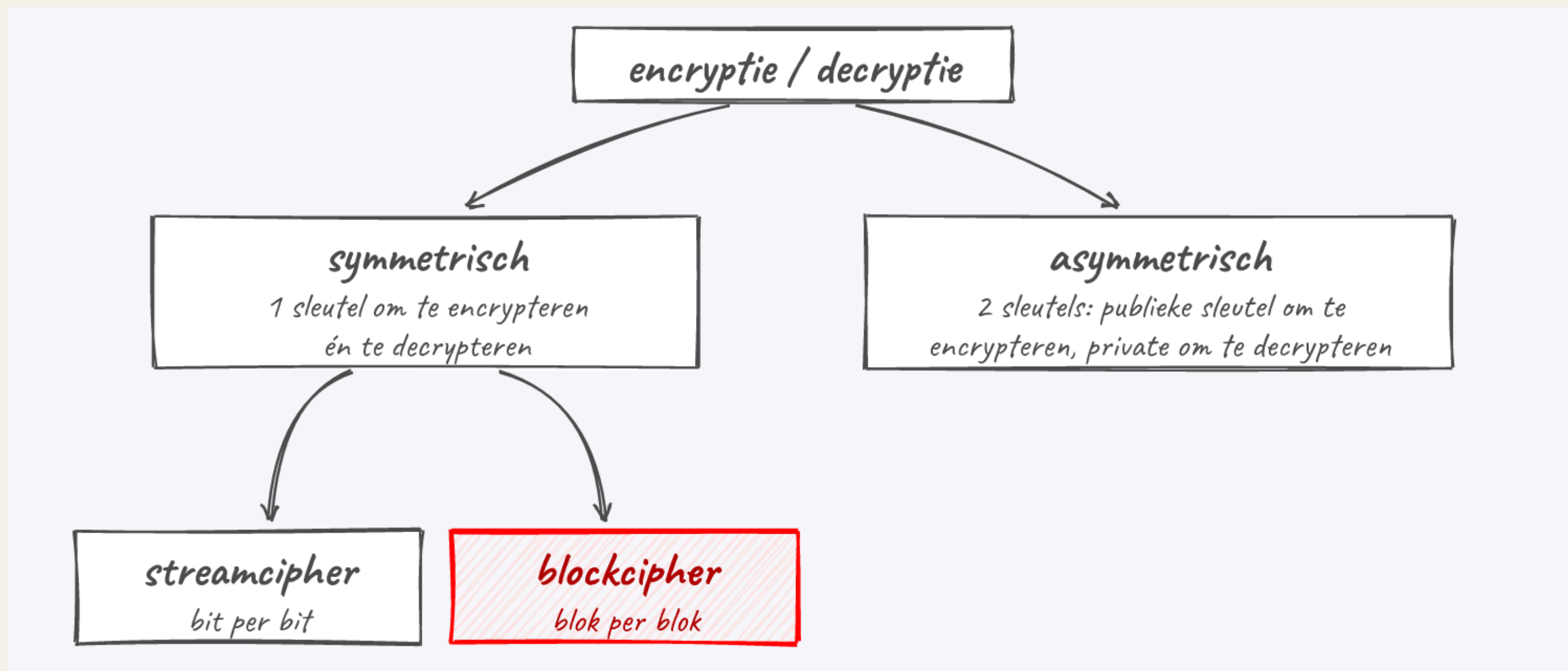
volgend plaintext-karakter: de PRGA levert de volgende k

SYMMETRISCH

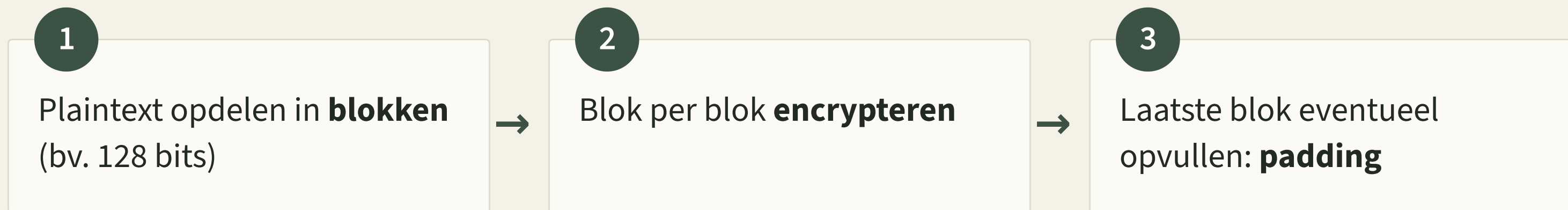
Blockciphers

Blok per blok versleutelen.

Waar staan we?



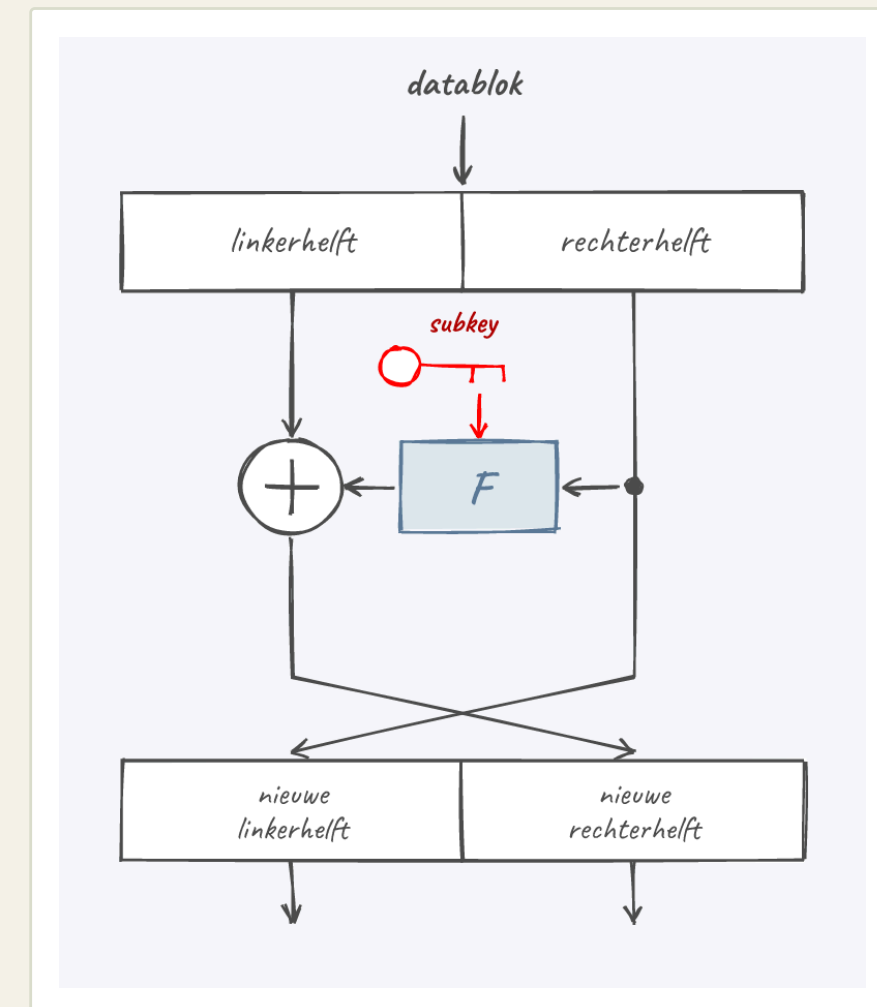
Blockciphers



Feistel-structuren

Een nieuwe soort operatie, naast XOR, substitutie en transpositie:

1. Data splitsen in **twee helften** (L en R)
2. F-operatie op één helft
3. Resultaat **XOR** met de andere helft
4. **Meerdere rondes**, elk met een andere subkey (via key scheduling)



Een enkele Feistel-structuur.

DES

1977

standaard

64 bits

per blok

56 bits

sleutel

- Outdated, maar belangrijk voor begrip
- Gebruikt in het **bankwezen** (financiële transacties)
- Gebaseerd op IBM's Lucifer cipher: de NSA liet de sleutellengte inkorten van 128 naar 56 bit!

DES: hoe werkt het?



Versleuteling

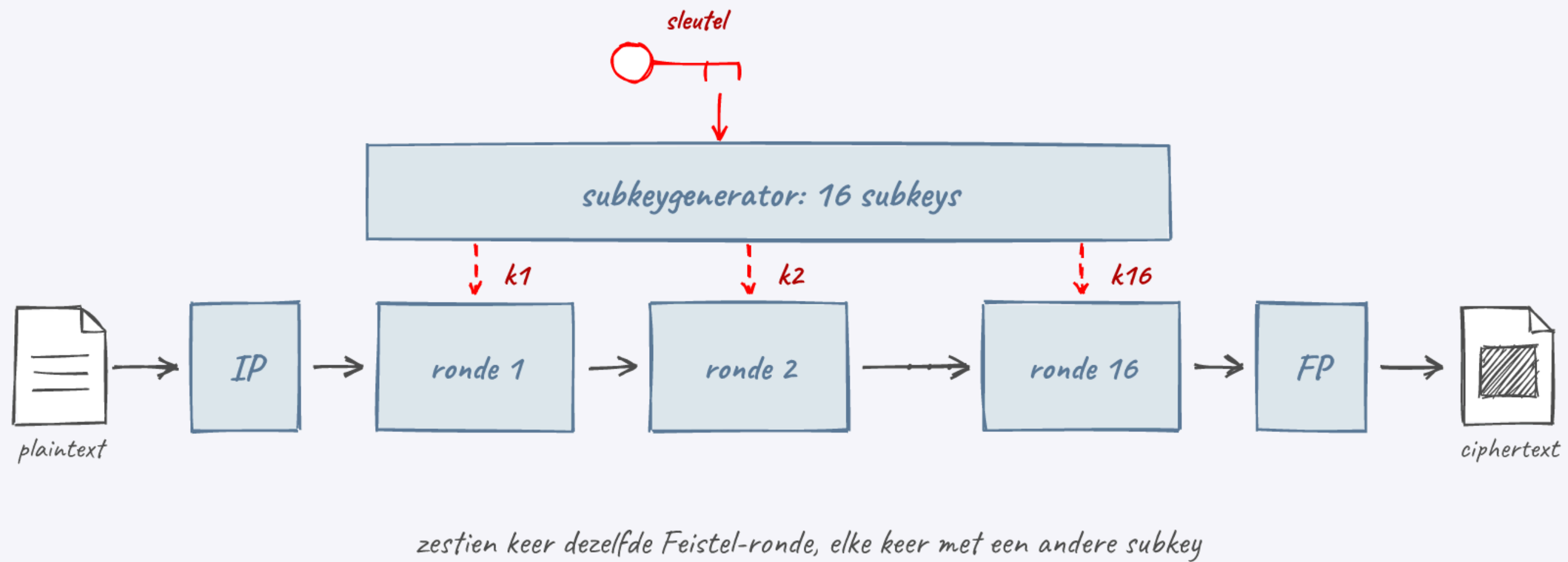
16 Feistel-rondes die de data blok per blok versleutelen.



Subkeys maken

16 subkeys uit de 56-bit sleutel, één per ronde.

DES: versleuteling



DES encryptie: 16 Feistel-structuren.

DES: stappen

1

Initiële permutatie (IP): bits herschikken



2

16 Feistel-rondes: splitsen, F-operatie, XOR



3

Finale permutatie (FP)

DES: de initiële permutatie

initiële permutatie (IP)

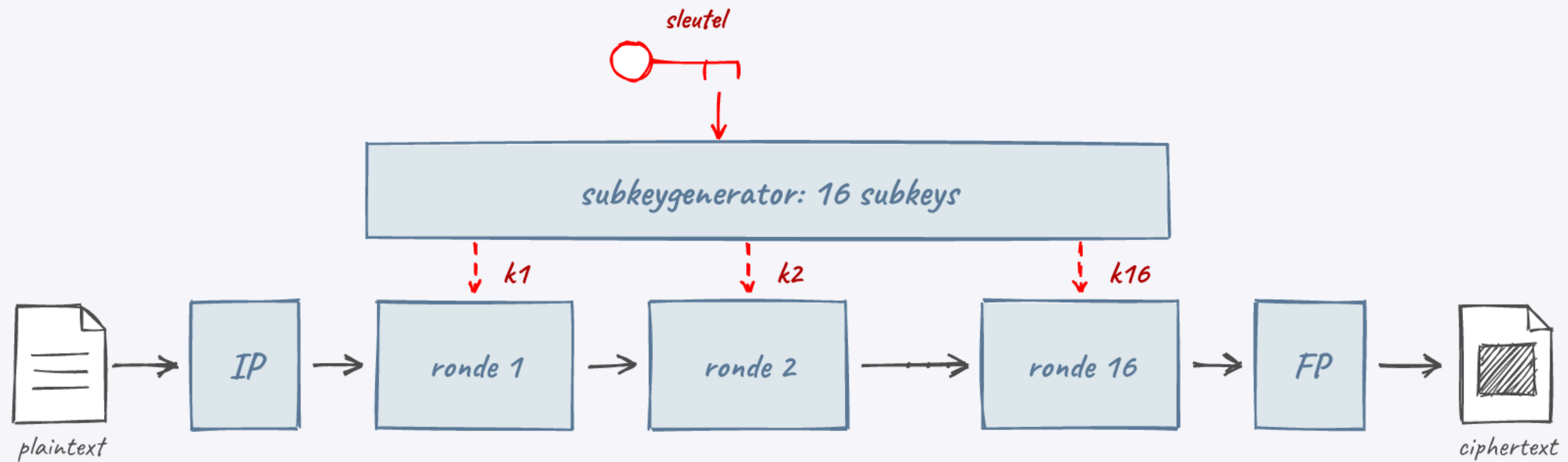
uitgangsbij 1 krijgt de waarde van bit 58



58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

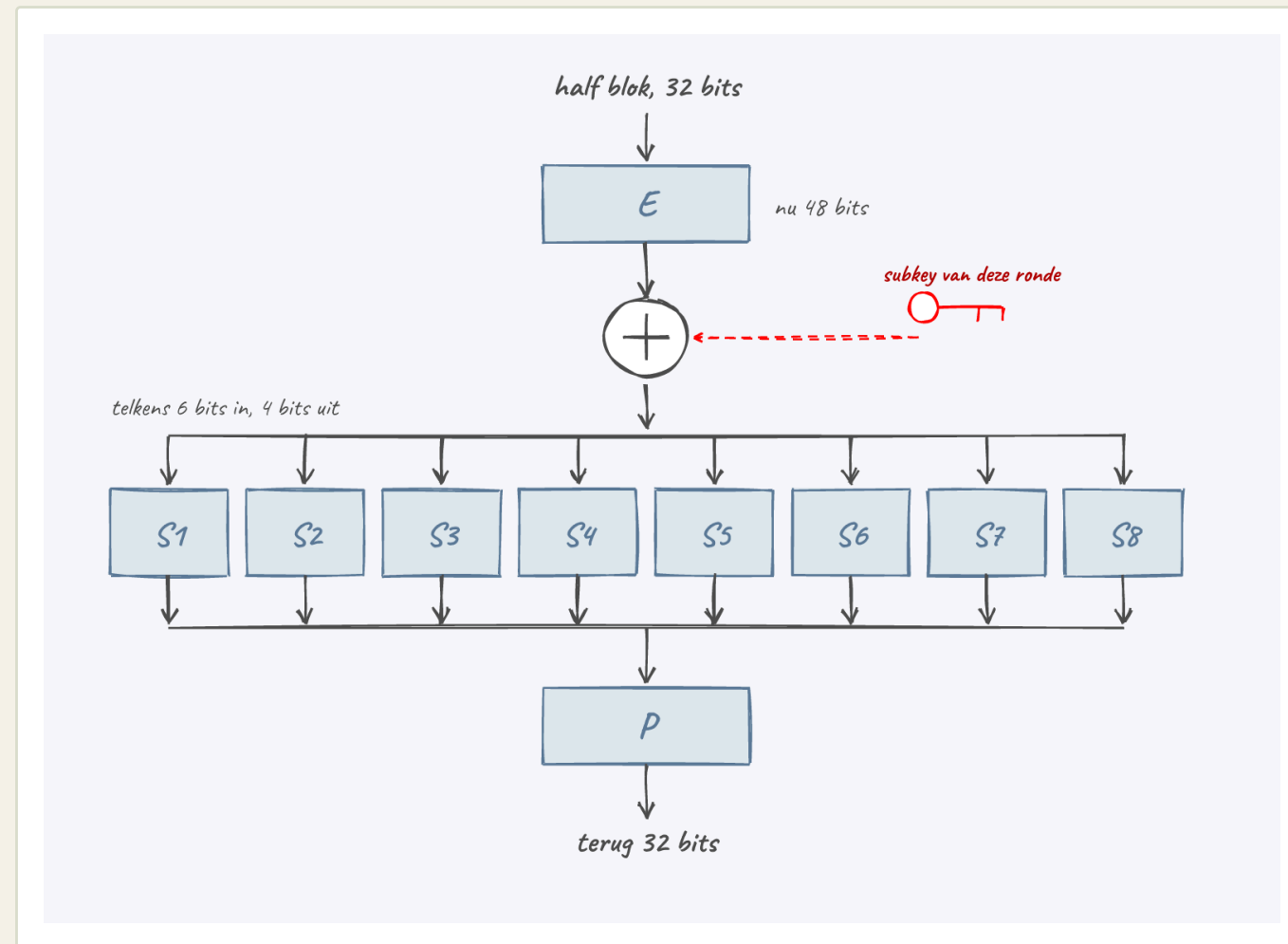
zo krijgt elk van de 64 bits een vaste nieuwe plek, elke keer dezelfde

DES



zestien keer dezelfde Feistel-ronde, elke keer met een andere subkey

DES: de F-operatie



Binnenin de F-operatie.

1. Expansie: 32 bits $\rightarrow$ 48 bits

expansie (E): 32 bits worden er 48

acht rijen van zes bits, samen achtenveertig

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

elke rij begint met de laatste twee bits van de rij erboven

acht rijen maal twee herhaalde bits: $32 + 16 = 48$

2. XOR met subkey (48 bits)

3. S-boxen: 6 bits $\rightarrow$ 4 bits (substitutie) (zie volgende slide)

DES: S-boxen

8 S-boxen, elk 6 bits in → 4 bits uit.

Outer bits (eerste + laatste) rij

Inner bits (middelste 4) kolom

Voorbeeld: **011011** → 1001

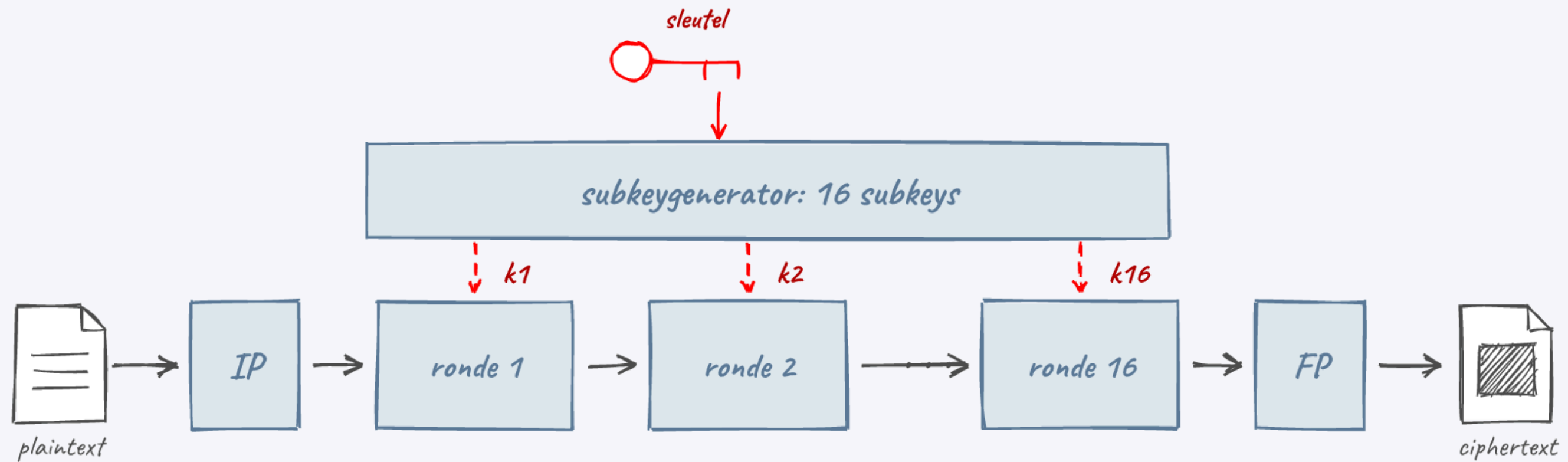
het S5-blok: zes bits in, vier bits uit
de twee outer bits kiezen de rij, de vier inner bits de kolom

outer bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
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

011011 komt binnen: outer bits 01 is rij 1, inner bits 1101 is kolom 13
daar staat 1001, en dat is wat het blok teruggeeft

Waarheidstabel van het S5-blok.

DES

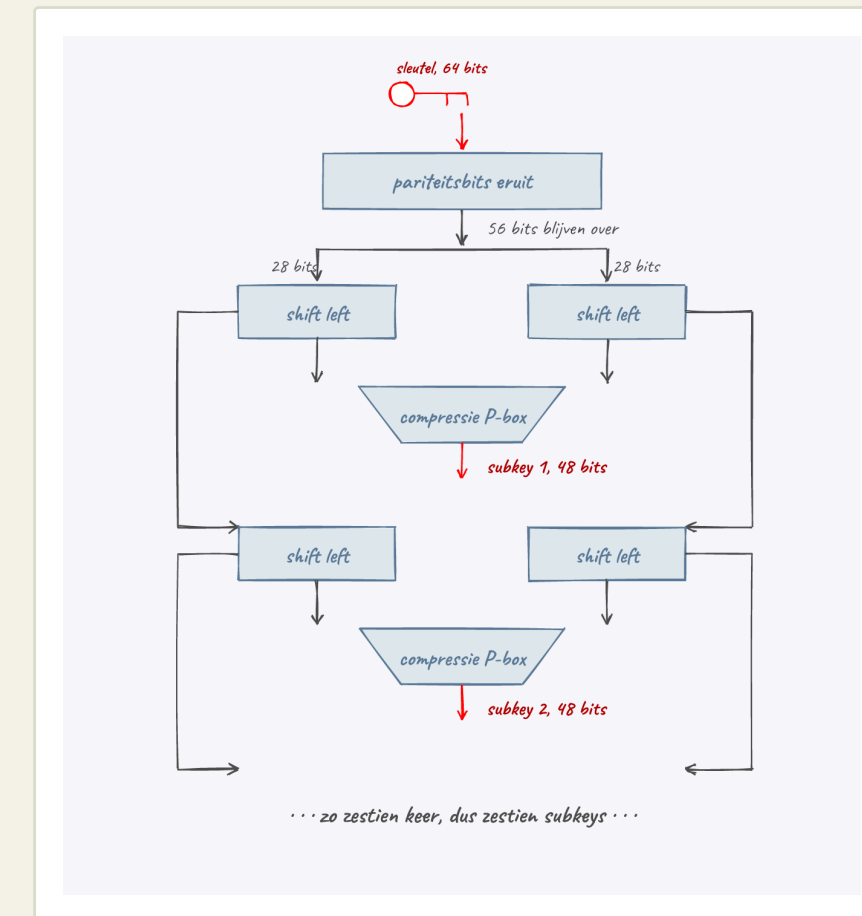


zestien keer dezelfde Feistel-ronde, elke keer met een andere subkey

DES: subkeys maken

1. 56-bit sleutel splitsen in **2 × 28 bits**
2. Per ronde: **bit-shift**
3. **Compressie P-box** → 48-bit subkey

16 rondes = **16 unieke subkeys**

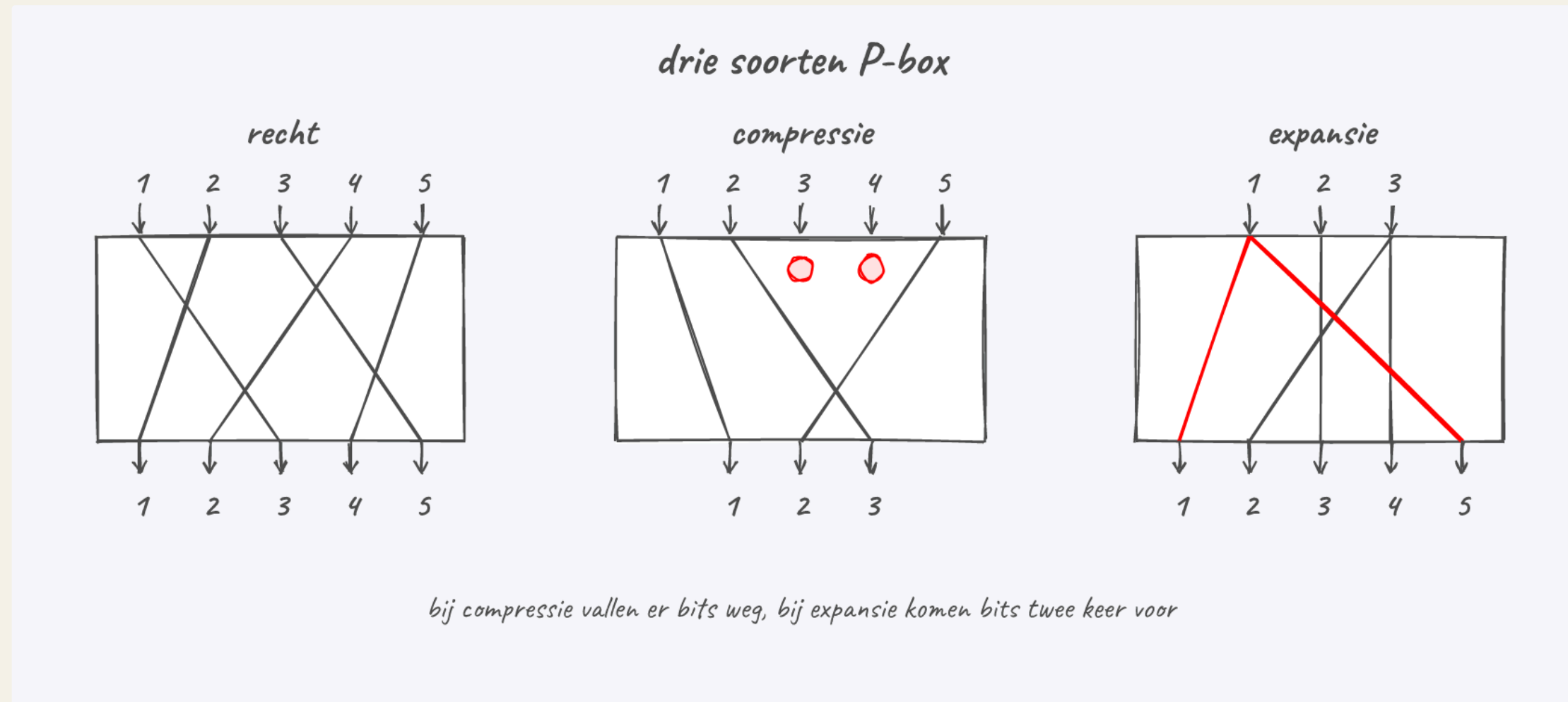


De 16 rondes die telkens 1 subkey maken.

DES: P-boxen

Compressie 56 bits → 48 bits (bits vallen weg)

Permutatie bits veranderen van plek



Drie types P-Boxes.

3DES

1995

de DES-sleutel (56 bit) bleek te kort

168 bits

sleutel, compatibel met bestaande DES-hardware

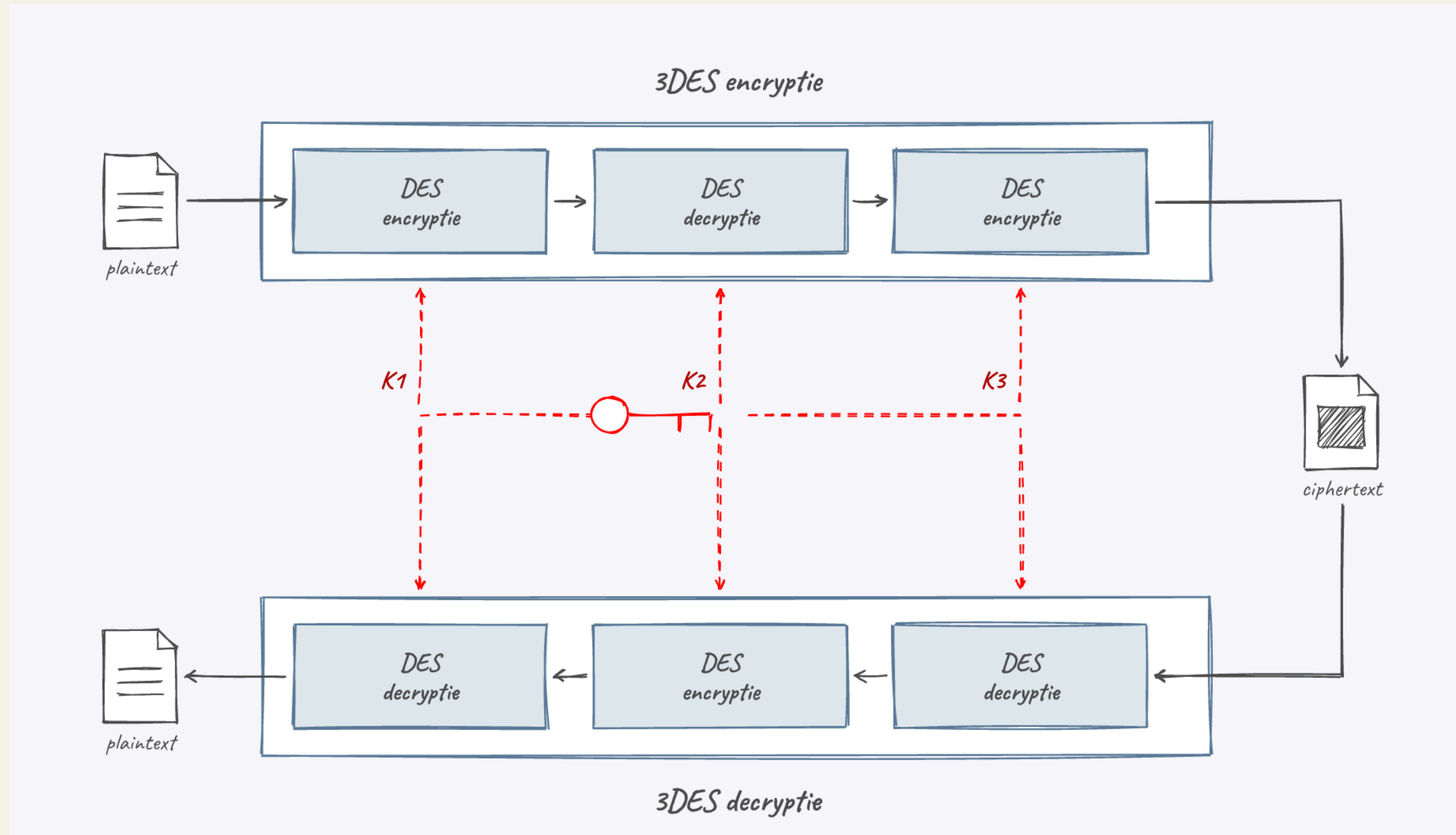
3x DES

Encrypt → Decrypt → Encrypt, met 3 verschillende sleutels

Verlengde de levensduur van DES, maar is nu ook verouderd.

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

Triple DES



3DES.

1998, ELECTRONIC FRONTIER FOUNDATION

DES gekraakt in de praktijk

Deep Crack kraakt een DES-sleutel in **dagen**.

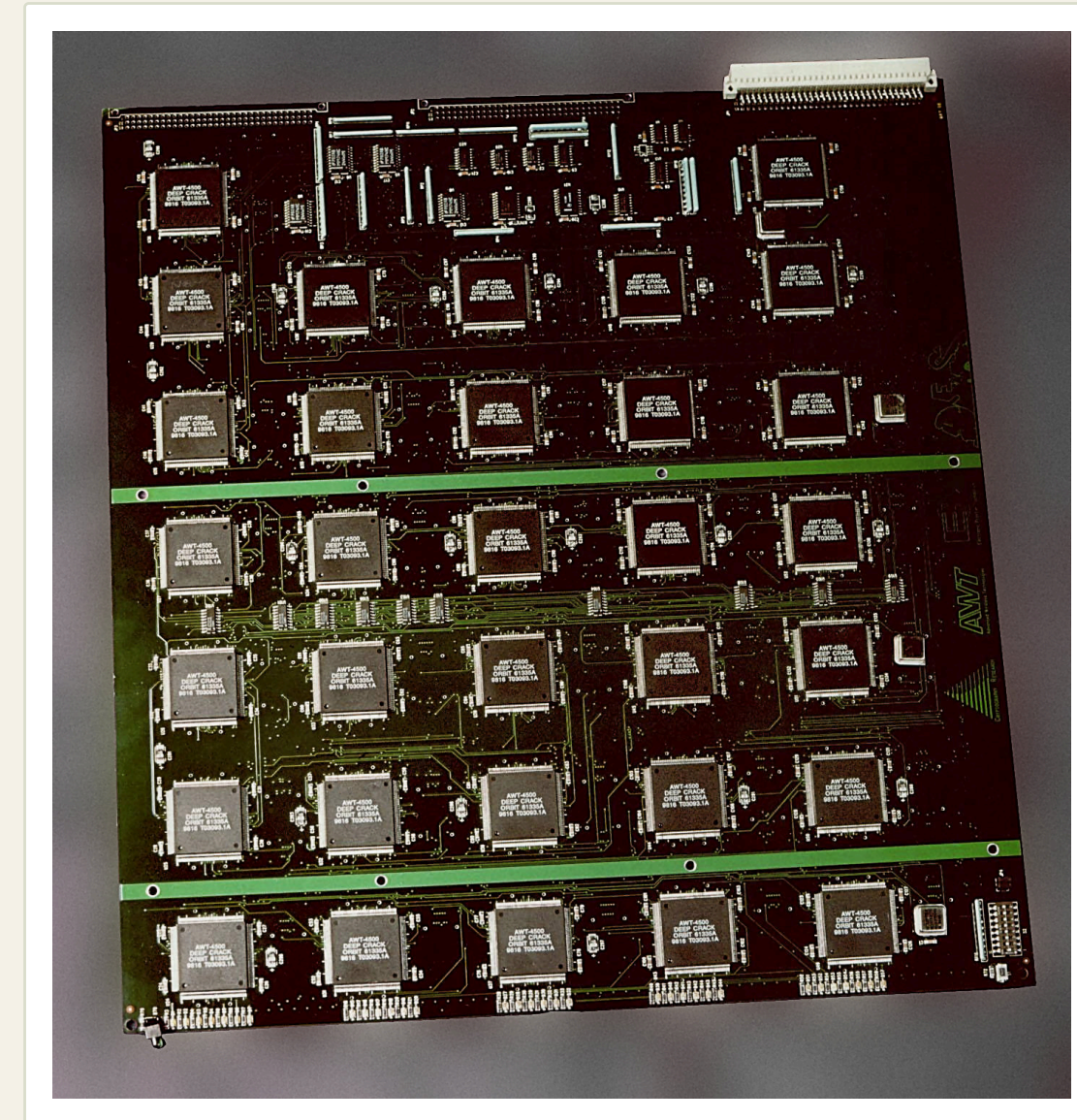
\$250.000 **1.856**

kostprijs

custom chips

< 24 uur

later, samen met distributed.net



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

“

It's time for those 128-, 192-, and 256-bit keys.

— Het bericht dat Deep Crack bij de publieke aankondiging kraakte

DES is **dood** voor gevoelige data. Dat bevestigt 3DES (1995) als de juiste keuze.

BLOCKCIPHERS

Block cipher modes

Hoe versleutel je data die groter is dan één blok?

HET PROBLEEM

Zelfde plaintext-blok + zelfde sleutel = **zelfde ciphertext.**

Dat lekt informatie over de plaintext. Daarom bedacht men verschillende **modes of operation.**

Block cipher modes: overzicht

ECB	blokken onafhankelijk (onveilig!)
------------	-----------------------------------

CBC	output van het vorige blok als extra input
------------	--

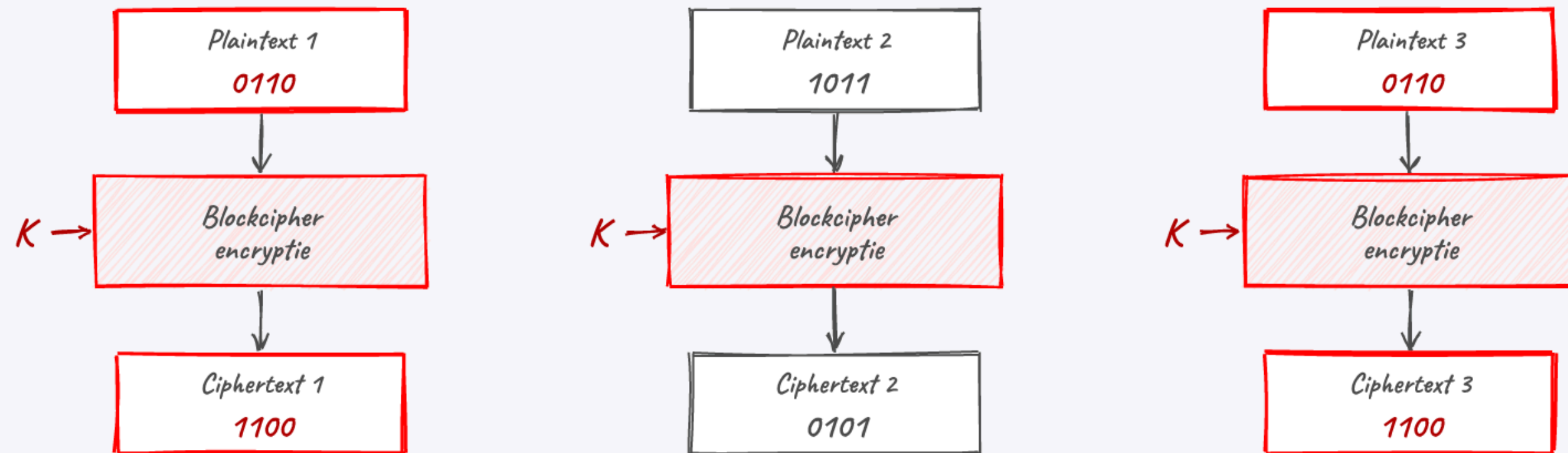
CFB	vergelijkbaar met CBC
------------	-----------------------

OFB	blockcipher als streamcipher
------------	------------------------------

CTR	teller (start bij een <i>nonce</i>) als extra input (veel gebruikt: WPA2, IPSEC)
------------	---

Block cipher modes: ECB

Electronic Codebook (ECB): elk blok onafhankelijk encrypteren

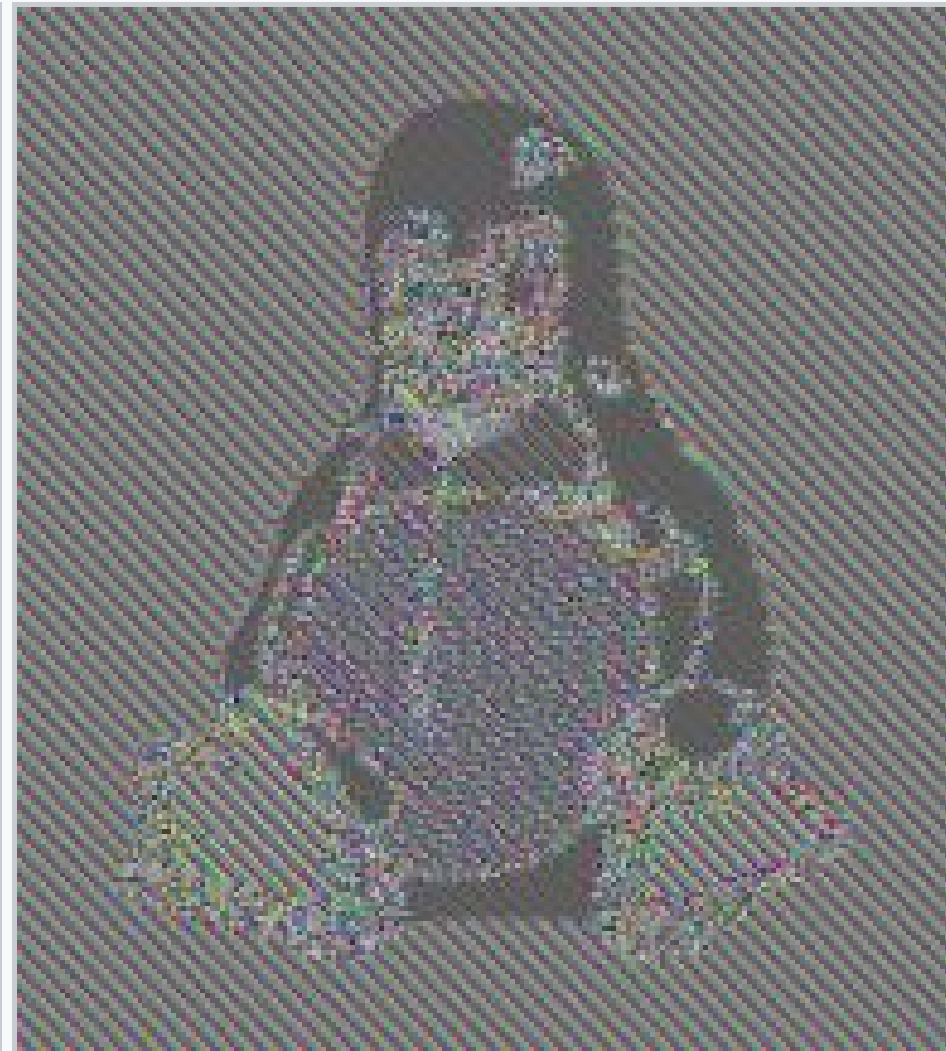


blok 1 en blok 3: zelfde plaintext geeft zelfde ciphertext

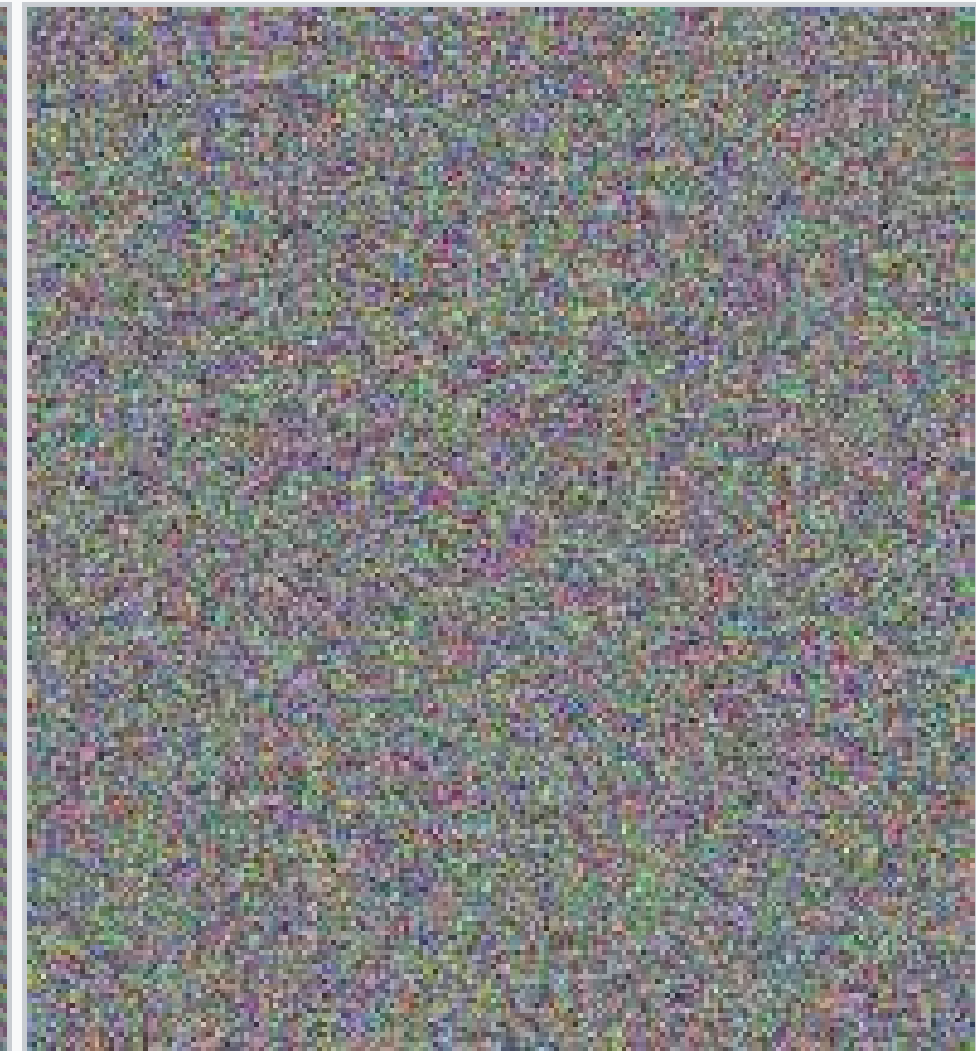
Probleem met ECB



Original image



Encrypted using ECB mode

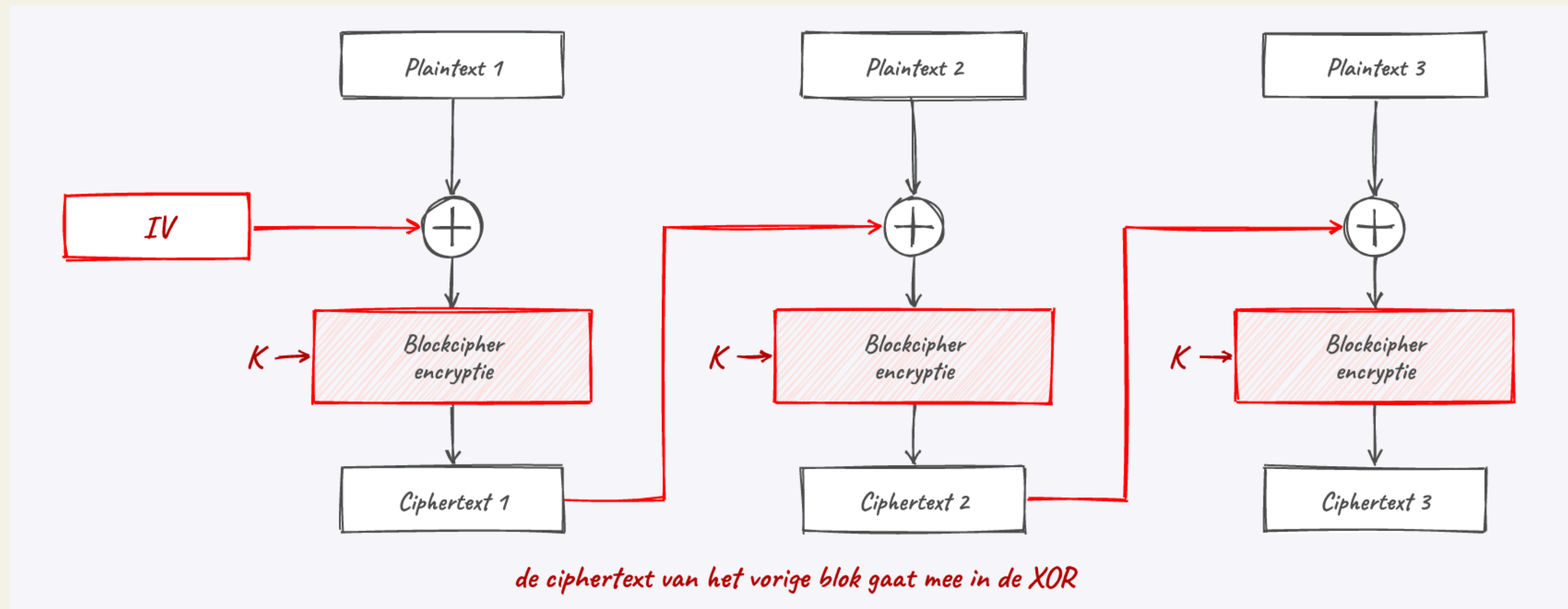


Modes other than ECB result in pseudo-randomness

ECB lekt patronen (Bron: Wikipedia).

Block cipher modes: CBC

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



CBC werkt met een “Initialisatie Vector” (IV)



Extra startwaarde

Naast de sleutel, die zelf **dezelfde** blijft. Zo geeft hetzelfde bericht telkens een andere ciphertext.



Onversleuteld mee

De IV gaat **ongeëncrypteerd** mee in de header. Dat is geen probleem.



IV-selectie

Bij CBC moet de IV **onvoorspelbaar** (willekeurig) zijn.

CBC uitgewerkt

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

Plaintext: 00 00 00 00 (vier gelijke blokken), IV: 10 . Met ECB: 01 01 01 01 .

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

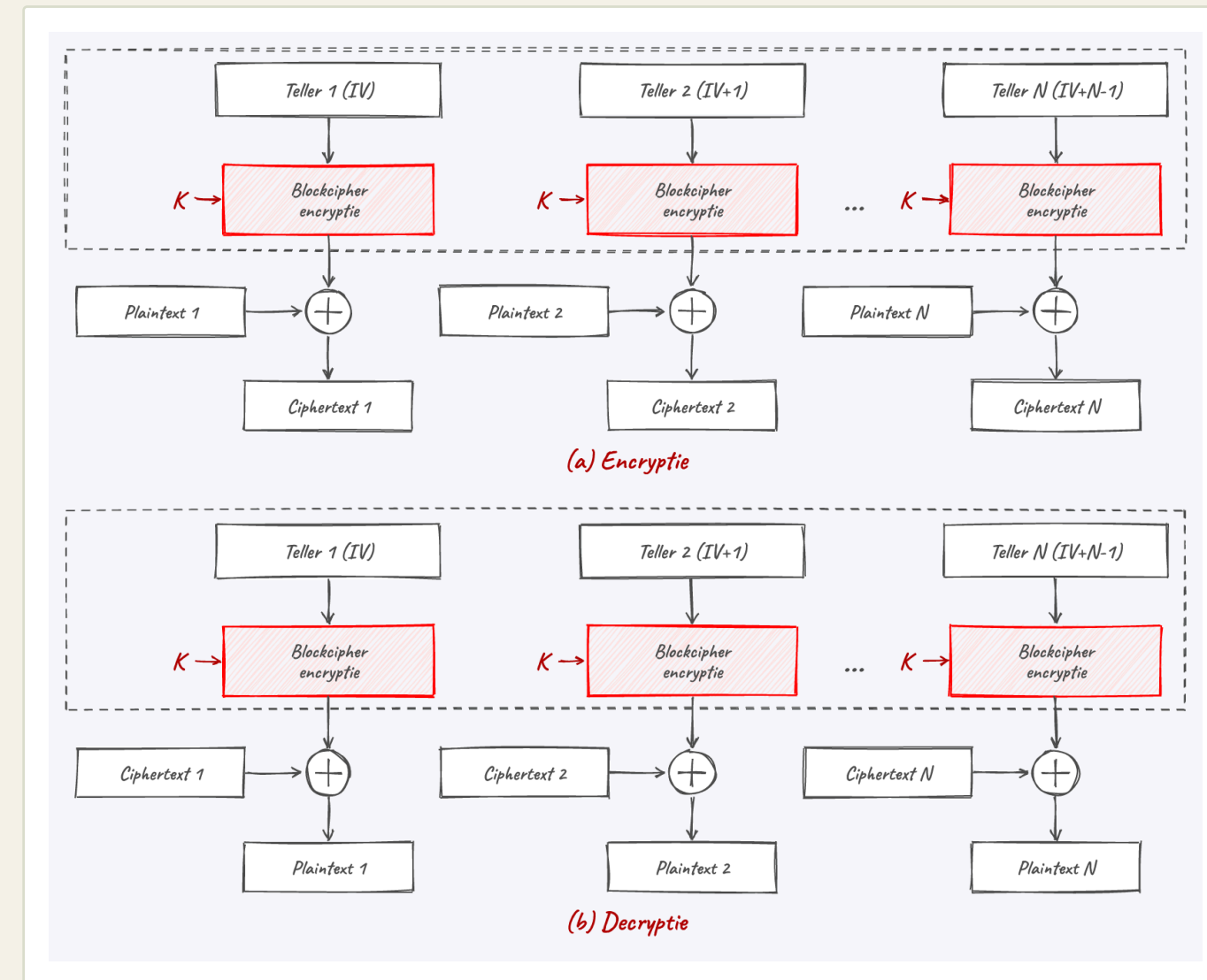
Ciphertext: 11 00 01 10

Vier **gelijke** plaintext-blokken, vier **verschillende** ciphertext-blokken.

Block cipher modes: CTR

Counter (CTR): de blockcipher encrypteert een **teller** die start bij een **nonce** (*number used once*).

- Blokken onafhankelijk van elkaar → **parallel** verwerken
- Gebruikt in WPA2 IPSEC, ... (bv. Netflix-streaming)
- **Nooit** dezelfde nonce hergebruiken met dezelfde sleutel: dan lekt de XOR van de plaintexts



CTR mode: encryptie en decryptie.

ADVANCED ENCRYPTION STANDARD

AES

De huidige standaard en meest gebruikte symmetrische blockcipher.

2001

nieuwe wereldwijde standaard

128 bits

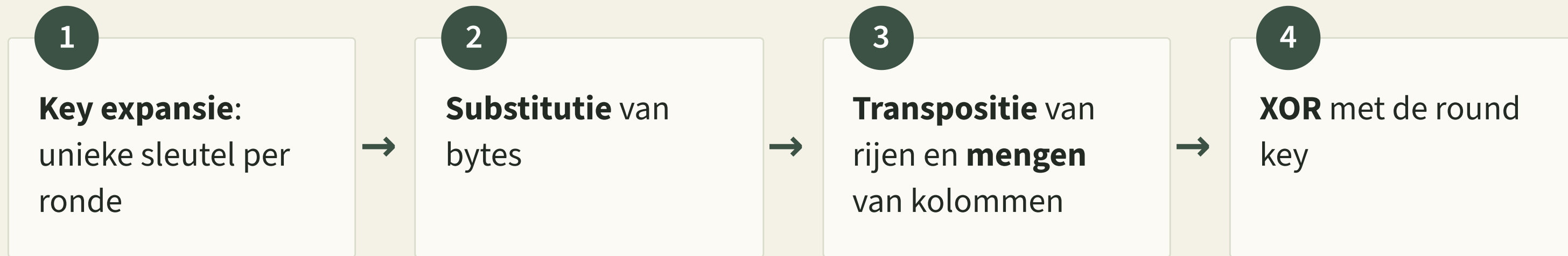
per blok

256 bits

maximale sleutellengte

Gebaseerd op **Rijndael**. Belgisch! Door Vincent **Rij**men en Joan **Da**emen.

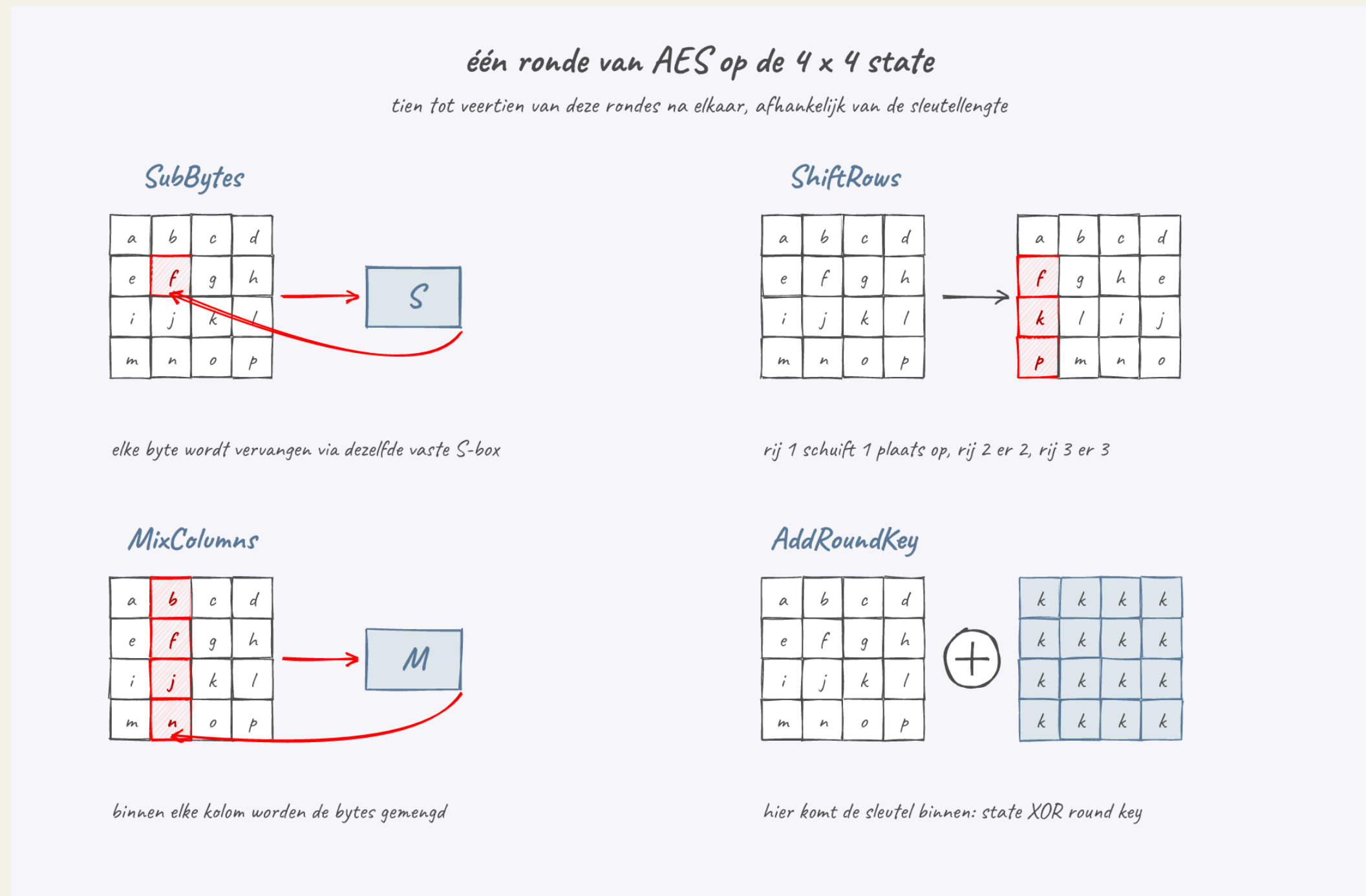
AES: werking in het kort



Stap 2 tot 4 herhaal je in **meerdere rondes**.

De XOR is ook hier de feitelijke encryptie.

AES encryptie



Eén ronde van AES.

Conclusie (deel 1)

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



Sleuteluitwisseling blijft het **grote probleem**.

Langere sleutels en sterke algoritmes geven betere bescherming. De oplossing voor de sleuteluitwisseling: **asymmetrische crypto**.

Up next?

