KRY03 - MNG

Model 2019

Kryptografie

Část 3 Symetrické algoritmy

Post 19/20

Souhrnné materiály

Ver 0.1

KRY

Kryptografie

Upd 2007/8
Post 2010/11
Pre 2015/16
Pre 2018/19
2020 MNGprep

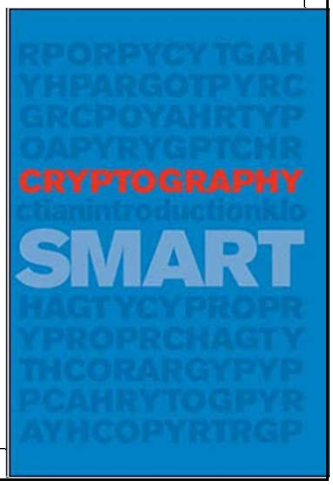
KRY 3

Symetrická kryptografie

Učebnice

3

- **Nigel Smart: Cryptography - An Introduction, 3rd Edition,**
  - Mcgraw-Hill College, 3rd Edition, 2013
  - ISBN-10: 0077099877
- **Kapitoly**
  - **Kapitola 8**
    - » Zajímavá je pro nás celá kapitola 8
  - **Kapitola 7**
    - » Zajímavá je pro nás celá kapitola 7



The third edition is now online. You may make copies and distribute the copies of the book as you see fit, as long as it is clearly marked as having been authored by N.P. Smart.

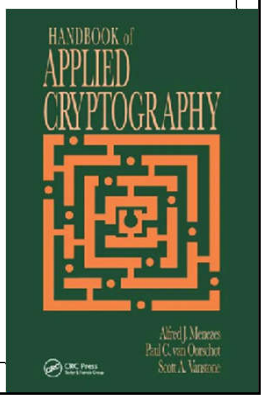
Učebnice je v dokumentovém skladu

©Petr Hanáček

Učebnice

3

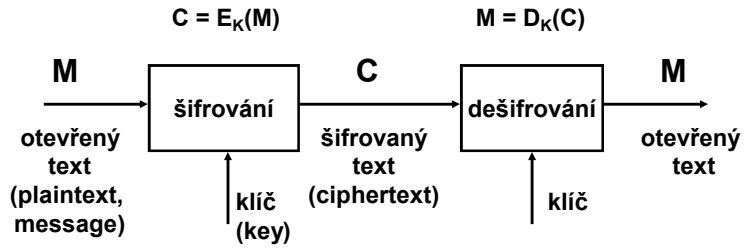
- Menezes, Van Oorschot, Vanstone: Handbook of Applied Cryptography, CRC Press, Hardcover, 816 pages, CRC Press, 1997.
- Kapitoly
  - Kapitola 7
    - » Zajímavá je pro nás celá kapitola 7
  - Kapitola 6
    - » Zajímavá je pro nás celá kapitola 6



http://www.cacr.math.uwaterloo.ca/hac/

Učebnice je v dokumentovém skladu

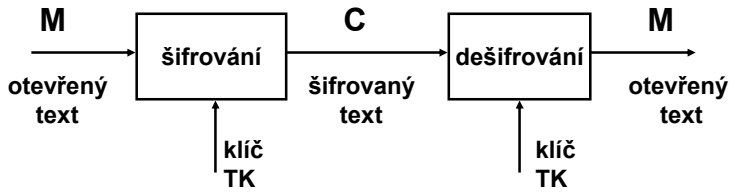
Kryptografie



- podle klíčů
  - symetrické X asymetrické
  - tajný klíč X veřejný klíč, soukromý klíč

KRY

Symetrický algoritmus



- S tajným klíčem
  - Uživatelé se dohodnou na stejném tajném klíči
  - Sdílené tajemství
- Útoky (COA, KPA)
- „Bezpečný algoritmus“
- Útok silou

Důvěrnost
Autentizace
Integrita
Nepopiratelnost

©Petr Hanáček CLACRYPT Slide 5

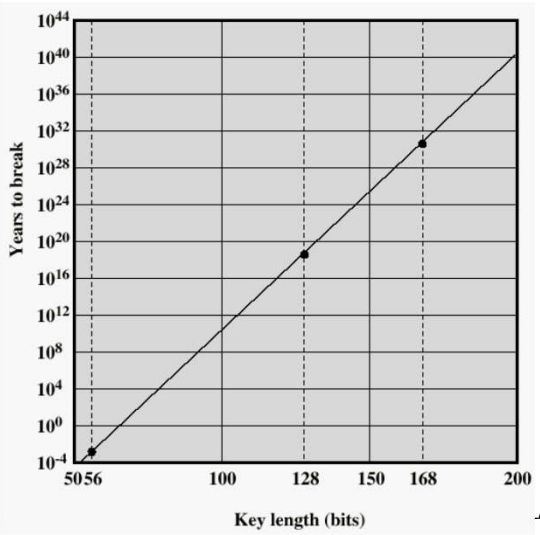
Útok silou

Délka klíče	1 test / uS	10^6 procesorů
32	35.8 m	2.15 ms
40	6.4 d	550 ms
48	4.46 r	2.35 m
56	>100 r	10.0 h
64		107 d

Typy útoků
- 1 počítač
- Paralelní superpočítač
- Celý svět (Čínská loterie)

©Petr Hanáček CLACRYPT Slide 6

Paralelní počítač – 10^6 decryptions/μs



Reálné útoky silou

Date	Key length	Time	# Computers	Rate (keys/sec)
8/95	40	8 days	120 + 2 super	0.5 M
1/97	40	3.5 hours	250	27 M
2/97	48	13 days	3,500	440 M
6/97	56	4 months	78,000	7 B
2/98	56	39 days	22,000 people	34 B
7/98	56	56 hours	1 with 1,728 chips	90 B
1/99	56	22 hours	100,000 + 1	250 B

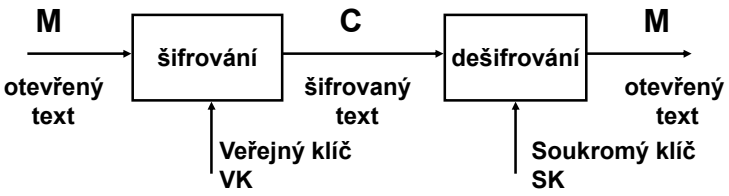
Symetrické algoritmy

- DES – 56 bit
- 3DES – 112 bit
- IDEA – 128-bit keys, PGP used in early versions
- RC2 – "Ron's code" (Ron Rivest), variable size key
- RC5 – variable size key
- Skipjack – 80-bit key, 32 rounds, NSA initially classified
- AES – variable size key

- Možnost vytvořit nový

 CLACRYPT Slide 9

Asymetrický algoritmus 1

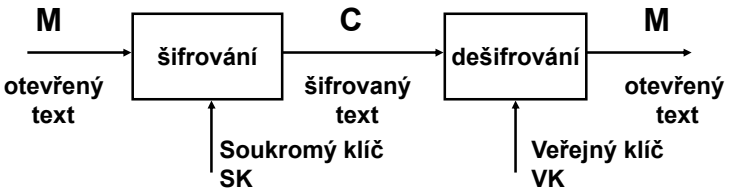


- Uživatel tají soukromý klíč, zveřejní veřejný klíč

Důvěrnost
Autentizace
Integrita
Nepopiratelnost

 CLACRYPT Slide 10

Asymetrický algoritmus 2

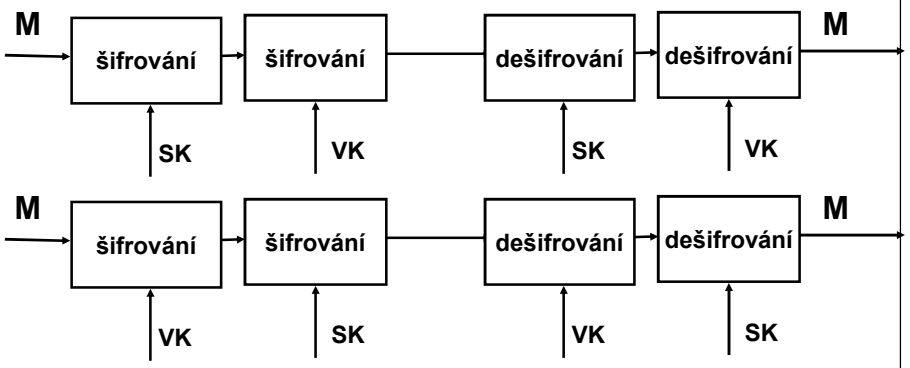


- Uživatel tají soukromý klíč, zveřejní veřejný klíč
- Elektronický podpis

Důvěrnost
Autentizace
Integrita
Nepopiratelnost

©Petr Hanáček CLACRYPT Slide 11

Asymetrický algoritmus 3



- Pořadí ?

Důvěrnost
Autentizace
Integrita
Nepopiratelnost

©Petr Hanáček CLACRYPT Slide 12

Asymetrické algoritmy

- RSA (Rivest – Shamir - Adleman)
- DSS/DSA (Digital Signature Standard)
- DH (Diffie-Hellman)
- Knapsack
- EC – Elliptic Curves

- Délky klíčů – 768, 1024, 2048 bitů
- Nemožnost vytvořit nový

Knapsack
Faktorizace čísel
Diskrétní logaritmus
Eliptické křivky

 CLACRYPT Slide 13

Porovnání vlastností

.	Tajný klíč	Veřejný klíč
kopií / tajemství	2	1
tajemství / uživatele	mnoho	1
rozšiřovatelnost	špatná	dobrá
rychlost	dobrá	malá

 CLACRYPT Slide 14

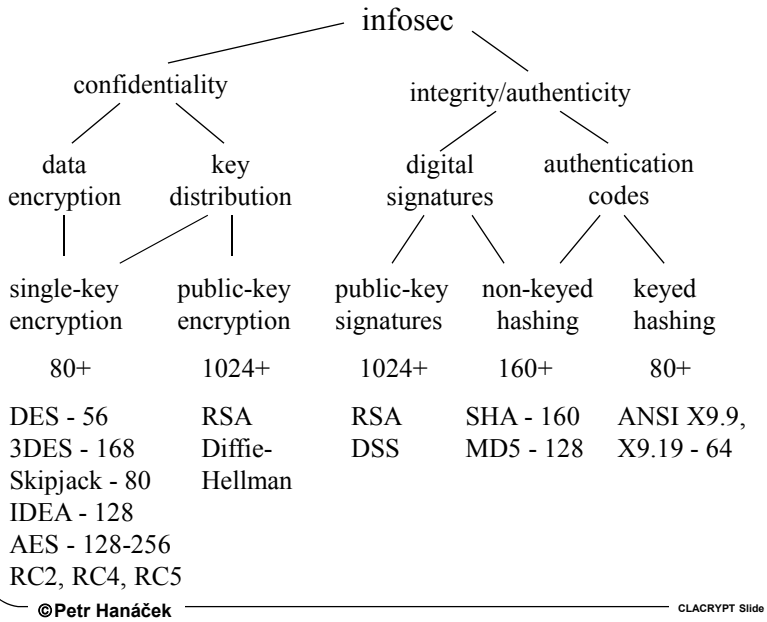
Hašovací funkce

- Hašovací funkce, charakteristika zprávy, jednocestná funkce, message digest, digest, hash, hash function, one way function
- je to funkce F taková, že
  - je aplikovatelná na argument libovolné velikosti
  - její výstupní hodnota má konstantní délku (zpravidla 128, 160 nebo 256 bitů)
  - lze rychle spočíst F(x)
  - pro dané *y* je výpočetně nezvládnutelné nalézt takové *x*, aby platilo *F(x)=y* *(first preimage resistance)*
  - pro dané *x* je výpočetně nezvládnutelné nalézt takové *x'<>x*, aby platilo *F(x')=F(x)* *(second preimage resistance)*
  - je výpočetně nezvládnutelné nalézt takové *x' a x, x'<>x*, aby platilo *F(x')=F(x)* *(collision resistance)*
- implementace
  - MD2, MD4, MD5
  - SHS (Secure Hash Standard), SHA

Birthday attack

- Birthday paradox:
  $r1, \ldots, rn \in [0,1,\ldots,B]$ indep. random integers.
  When $n = 1.2\sqrt{B}$ then
  $Pr[\exists i \neq j : r_i = r_j] > \frac{1}{2}$
- msg-digest only 64 bits long $\Rightarrow$
  can find collision in 2^{32} tries.
- Typical digest size = 160 bits. (e.g. SHA-1)
  $\Rightarrow$collision time is 2^{80} tries.

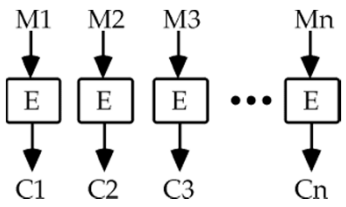
Základní schéma



Blokové šifry a DES

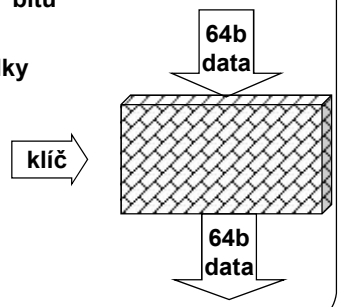
Blokové versus proudové šifry

- **Proudová šifra**
  - **Bere zprávu po bajtech (případně po bitech)**
- **Bloková šifra**
  - **Bere zprávu po větších blocích (64, 128 nebo 256 bitů)**
  - **Před zašifrováním musí být celý blok k dispozici**
  - **Větší velikost bloku zabraňuje**
    - » **Slovníkovým útokům**
    - » **Statistickým útokům**



Bloková šifra

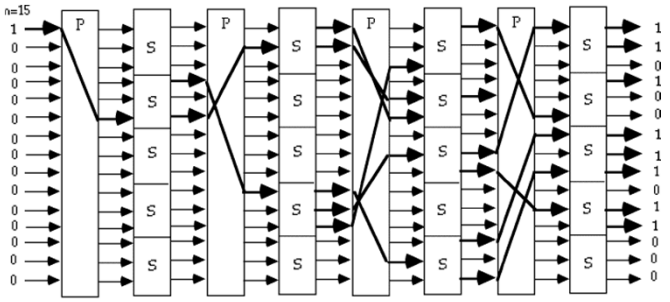
- 64 bitové bloky dat (nyní 128 až 256)
- Pro 64 bitový blok se jedná o 2^{64} možných bloků otevřeného textu a alespoň 2^{64} odpovídajících bloků zašifrovaného textu
  - Existuje $2^{64}!$ možných zobrazení
- Proč nevytvořit náhodné zobrazení?
  - Byla by třeba $2^{64} * 64$-bitová tabulka $\approx 10^{21}$ bitů
  - $14 quadrillion
  - Přenos klíče znamená přenos nové tabulky
- Ideální náhodné zobrazení aproximujeme pomocí několika komponent, řízených hodnotou klíče



©Petr Hanáček CLACRYPT Slide 21

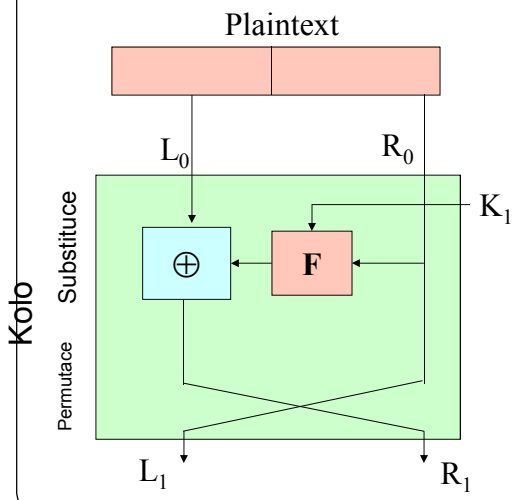
Feistelova šifra

- Základ některých symetrických šifer
  - Horst Feistel pracoval pro IBM v roce 1973
  - IBM's *Lucifer* algoritmus, založený na Feistelově principu, byl základem pro algpritmus DES v roce 1977
- Mnoho jiných algoritmů používá Feistelův princip
  - Mimo Feistelův princip jsou však i jiné iterativní principy
- Jde o substituční/permutační síť



©Petr Hanáček CLACRYPT Slide 22

Feistelova šifra



L_0 = levá polovina bloku
R_0 = pravá polovina bloku

$L_i = R_{i-1}$
$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

$C = R_n \| L_n$
n je počet kol

©Petr Hanáček CLACRYPT Slide 23

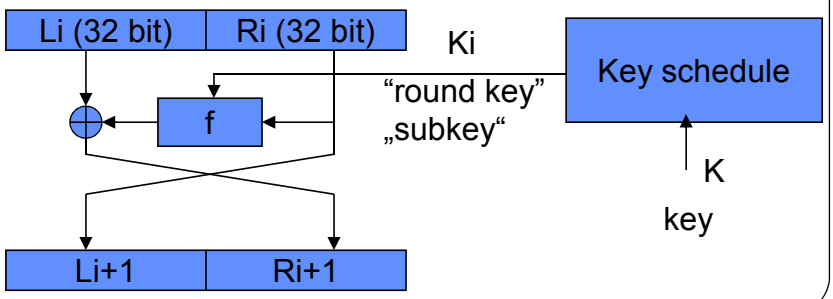
Poznámky

- **Proces je reverzibilní**
  - **$R_{i-1}=L_i$**
  - **$L_{i-1}=R_i \oplus F(R_{i-1},K_{i-1})$**
  - **Použije se stejný algoritmus ale pořadí subklíčů je opačné**
- **Jaké jsou požadavky na F?**
  - **Aby fungovalo dešifrování : žádné!**
  - **Aby byl algoritmus bezpečný:**
    - » **Skrytí vlastností zprávy**
    - » **Skrytí vlastností klíče**
    - » **Vytvořit dobrou F je obtížné**
- **Bezpečnost**
  - **Větší velikost bloku znamená méně bloků a větší bezpečnost**
  - **Větší velikost klíče znamená větší bezpečnost**
  - **Více kol znamená větší bezpečnost (?)**
  - **Větší složitost generace subklíčů může znamenat větší bezpečnost**
  - **Větší složitost F může znamenat větší bezpečnost**

©Petr Hanáček CLACRYPT Slide 24

Subklíče

- Vytvářejí se v bloku “Key schedule“
- Tvorba subklíčů má velmi výrazný vliv na bezpečnost algoritmu
  - Možnost vytvoření slabých subklíčů („00000000“ ??)
  - Možnost slabých klíčů



©Petr Hanáček CLACRYPT Slide 25

DES

©Petr Hanáček CLACRYPT Slide 26

Algoritmus DES

- vyvinut IBM v r. 1976 na zakázku NBS (nyní NIST)
- Na základě algoritmu Lucifer od IBM
- Modifikován NSA
  - Změna S-Boxů
  - Redukovaná délka klíče ze 128 na 56 bitů
- Přijat jako standard v r. 1976
- Zašifroval nejvíce bitů ze všech algoritmů

Požadavky na DES

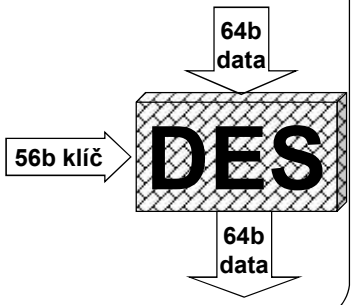
- musí zajišťovat vysokou bezpečnost
- musí být přesně specifikovaný
- bezpečnost nesmí záviset na utajení algoritmu
- musí být realizovatelný pomocí hardware
- musí být rychlý

NBS - National Bureau of Standards

 CLACRYPT Slide 27

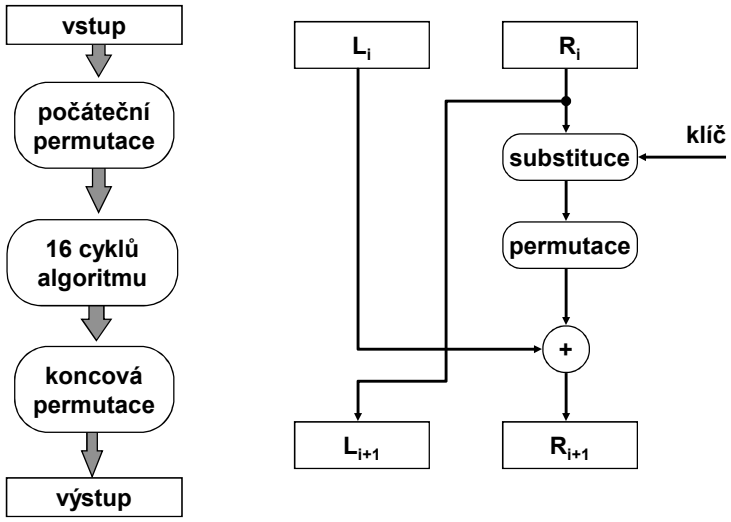
DES

- <u>symetrický</u> šifrovací algoritmus <u>tajným</u> klíčem
- šifruje bloky dat o šířce 64 bitů klíčem o velikosti 56 bitů
- Feistelova šifra s dodatečnou počáteční permutací IP
- 16 kol
- 56-bitový klíč, posuvy a permutace vytvářejí 48-bitové subklíče pro každé kolo
- Počet klíčů $2^{56} = 7.2 \times 10^{16}$



 CLACRYPT Slide 28

DES



©Petr Hanáček CLACRYPT Slide 29

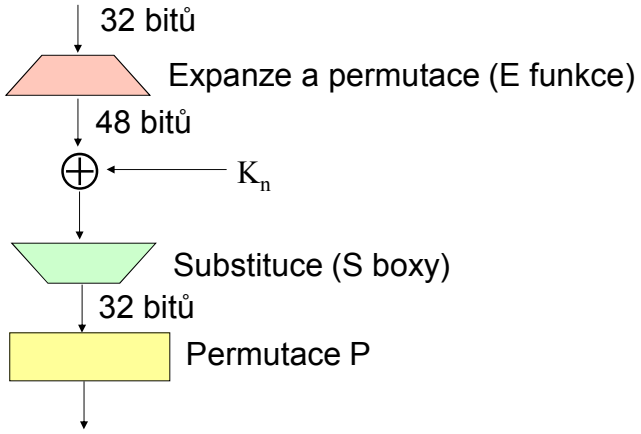
Počáteční permutace IP

IP

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

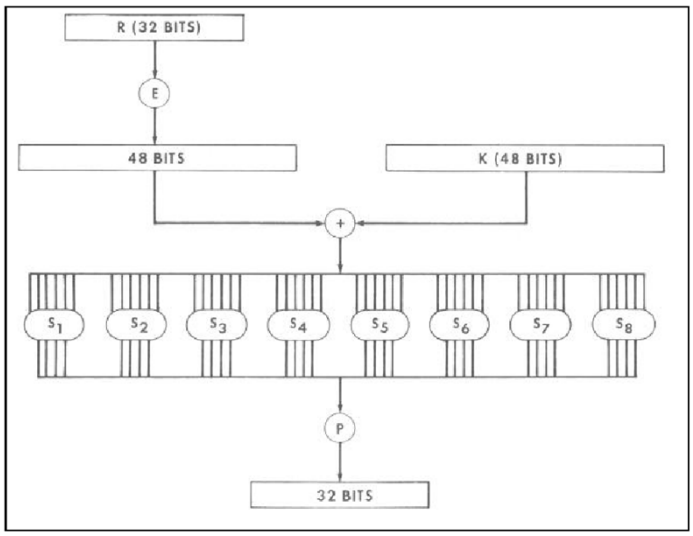
©Petr Hanáček CLACRYPT Slide 30

Funkce F



©Petr Hanáček CLACRYPT Slide 31

Funkce F



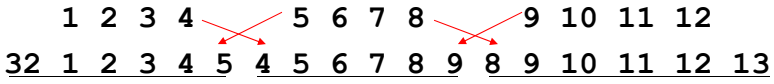
©Petr Hanáček CLACRYPT Slide 32

Funkce E

- Expanduje vstup ze 32 na 48 bitů
- Duplikace některých bitů

E BIT-SELECTION TABLE

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



 CLACRYPT Slide 33

Permutace P

- Permutuje 32 bitový výstup z S-boxů

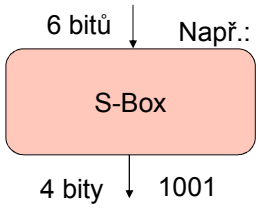
P

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

 CLACRYPT Slide 34

S-Boxy

- **Původně Selection Box, později Substitution Box**
- **Zásadní pro bezpečnost**
- **NSA změnila obsah S-Boxů**
- **Jediný nelineární krok DESu**



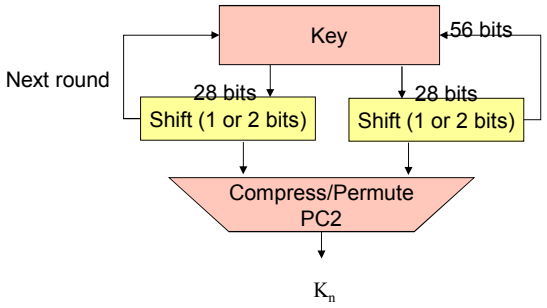
Column Number

Row No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

©Petr Hanáček CLACRYPT Slide 35

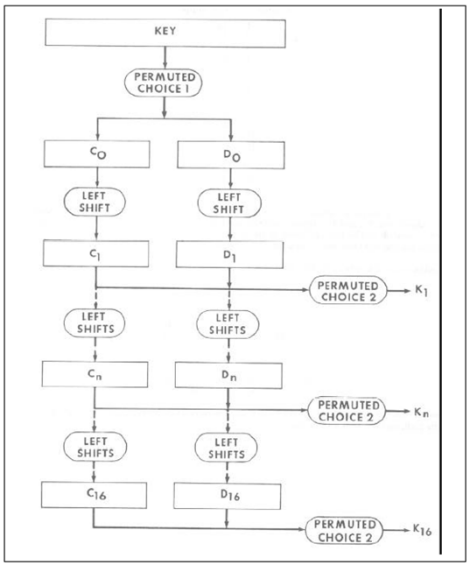
Key Schedule

- **Je třeba 16 48-bitových klíčů**
  - **Nejlepší by bylo mít 16 nezávislých klíčů -> 768 bitů**
- **Používá se 56-bitový klíč (64 s paritou)**
- **Otázky**
  - **Dešifrování**
  - **Slabé klíče**



©Petr Hanáček CLACRYPT Slide 36

Key Schedule



©Petr Hanáček CLACRYPT Slide 37

PC1, PC2 - Permuted Choice

PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36

$C_{()}$

63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

$D_{()}$

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

CLACRYPT Slide 38

Left Shift Schedule

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Pozn.: Ve skutečnosti se nejedná o posuvy (shifts) ale o rotace

©Petr Hanáček CLACRYPT Slide 39

Jedno kolo algoritmu DES

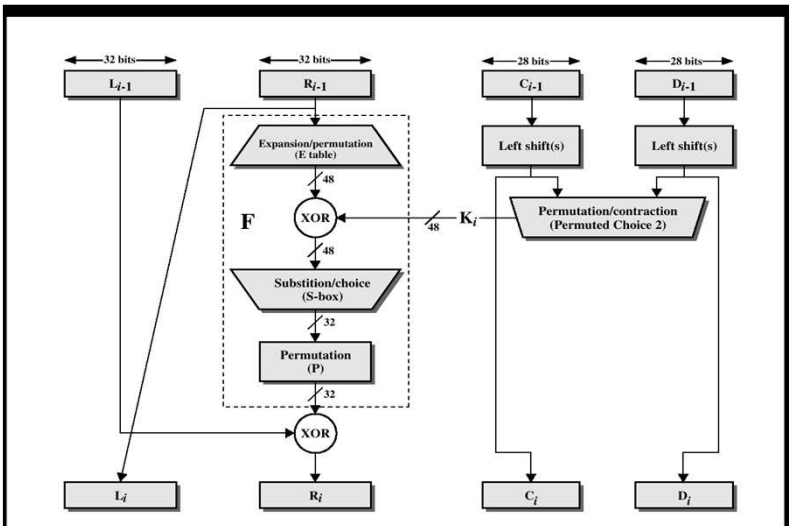
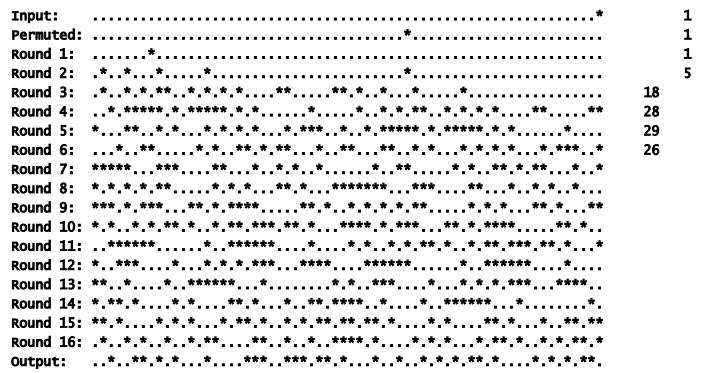


Figure 2.4 Single Round of DES Algorithm

©Petr Hanáček

Bezpečnost DES

DES Avalanche Efekt



Source: *Willem de Graaf,* http://www-groups.dcs.st-and.ac.uk/~wdg/slides/node150.html

DES Avalanche Efekt

- Změna v otevřeném textu
  - Počet výstupních bitů které se změní při změně jednoho bitu na vstupu

• Round:	0	1	2	3	4	5	6
• Bits:	1	6	21	35	39	34	32
• Round:	7	8	9	10	11	12	13
• Bits:	31	29	42	44	32	30	30
• Round:	14	15	16				
• Bits:	26	29	34				

- Stejná vlastnost se zkoumá pro změnu jednoho bitu klíče

©Petr Hanáček CLACRYPT Slide 43

DES - slabiny a pochybnosti

- Velikost bloku a klíče
  - Je více bloků než klíčů
  - Pro jeden blok 2^{64} zpráv > 2^{56} klíčů
  - Zašifrovaný konstantní blok nemůže nabýt všech 2^{64} možných hodnot
- 56-bitový klíč je příliš krátký
  - úspěšnost útoku silou
- Komplementární klíče
- Mezi klíči existují tzv. slabé klíče
  - při generování klíče je třeba kontrolovat, zda nejde o slabý nebo poloslabý klíč
- Není zcela jasné, zda 16 cyklů je postačující pro bezpečné zašifrování
- Návrh S-boxů nebyl zveřejněn - možnost "zadních vrátek"
  - pokud by S-boxy byly nějakou lineární funkcí, autor S-boxů může snadno šifru rozbít
  - 2011 - NSA prohlašuje, že v DESu zadní vrátka nebyla

http://gcn.com/articles/2011/02/16/rsa-11-nsa--no-des-backdoor.aspx?s=gcndaily_170211

©Petr Hanáček CLACRYPT Slide 44

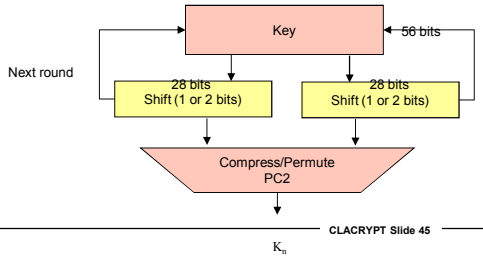
Slabé klíče

Slabé klíče (4)
- Levá a pravá polovina jsou samé nuly nebo jedničky
- Všech 16 subklíčů je stejných

Poloslabé klíče (12)
- Generují pouze 2 různé subklíče

Pravděpodobně slabé klíče (48)
- Generují pouze 4 různé subklíče

Celkem 64 "slabých" klíčů ze 72,000 trilionů
- malá pravděpodobnost vygenerování



©Petr Hanáček CLACRYPT Slide 45

Zadní vrátka v S-boxech

- NSA nezveřejnila podrobnosti o návrhu
- Nejsou náhodné, ale mají jisté pravidelnosti
- Úplně náhodné S-boxy by byly méně bezpečné
- Pokud by obsahovaly lineární funkci, prolomitelné pro toho, kdo je navrhoval

©Petr Hanáček CLACRYPT Slide 46

Útoky na DES

- **Využívající slabin**
  - **Slabé (poloslabé klíče)**
  - **Komplementární klíče (z krácení z 56 bitů na 55 bitů)**
  - **Diferenciální kryptoanalýza – není prakticky použitelná**
  - **Lineární kryptoanalýza – není prakticky použitelná**
- **Útok silou**
  - **Nejlepší známý útok**

- **1990, Eli Biham a Adi Shamir, diferenciální kryptoanalýza**
- **1993, Mitsuru Matsui, lineární kryptoanalýza**

 CLACRYPT Slide 47

Diferenciální kryptoanalýza

- **Eli Biham & Adi Shamir, 1990**
  - **NSA ji asi znala dříve**
- **Umí rozlomit 8-round DES ve dvou minutách (chosen-plaintext attack)**
- **Pro 16-round není rychlejší než útok silou**
- **Potřebuje 2^{47} zvolených zpráv (CPA) nebo 2^{55} známých zpráv (KPA)**
- **Není pro DES prakticky použitelná**
- **Útok je možný proto, že S-boxy nemají rovnoměrně rozložené výstupy xorovaných výstupů pro xorované vstupy**
- **Např. pro S1, pokud xor vstupů je 110100, pak XOR výstupů je 0010 s pravděpodobností 16/64 = 1/4 (měla by být 1/16)**

 CLACRYPT Slide 48

Diferenciální kryptoanalýza

- Předpokládejme DES
- Celý DES je lineární vyjma S-boxů
- Diferenciální útok se soustředí na nelinearitu
- Myšlenka je porovnávat rozdíly vstupu a výstupu S-boxů
- Pro jednoduchost předpokládejme jednokolový DES s jedním S-boxem

 CLACRYPT Slide 49

Diferenciální kryptoanalýza

- Předpokládejme DES s S-boxem který převádí 3 bity na 2 bity
- *Sbox(abc)* označuje prvek v řádku *a* a sloupci *bc*
- Příklad: Sbox(010) = 11

	column			
row	00	01	10	11
0	10	01	11	00
1	00	10	01	11

- Zavedeme klíč:
  - Nechť $X_1 = 110$, $X_2 = 010$, $K = 011$
  - Pak $X_1 \oplus K = 101$ a $X_2 \oplus K = 001$
  - $Sbox(X_1 \oplus K) = 10$ a $Sbox(X_2 \oplus K) = 01$

 CLACRYPT Slide 50

Diferenciální kryptoanalýza

- **Předpokládejme**
  - Neznámé: **K**
  - Známé: **X = 110, X = 010**
  - Známé: **Sbox(X ⊕ K) = 10, Sbox(X ⊕ K) = 01**
- **Víme že X ⊕ K ∈ {000,101}, X ⊕ K ∈ {001,110}**
- **Potom K ∈ {110,011} ∩ {011,100} ⇒ K = 011**
- **Jde v podstatě „known plaintext attack“ na S-box**

	column			
row	**00**	**01**	**10**	**11**
0	**10**	**01**	**11**	**00**
1	**00**	**10**	**01**	**11**

©Petr Hanáček CLACRYPT Slide 51

Diferenciální kryptoanalýza

- **Útok na jeden S-box není moc užitečný**
  - **Útočník nezná jeho vstup a výstup**
- **Aby to bylo k něčemu, musíme dokázat dvě věci:**
  - **Rozšířit útok na celé jedno kolo**
    - » **V něm je několk S-boxů**
    - » **Zvolíme vstupy tak, aby pouze jeden S-box byl “aktivní”**
  - **Rozšířit útok na (skoro) všechna kola**
    - » **Výstup jednoho kola je vstupem dalšího kola**
    - » **Je nutné zvolit vstupy tak, aby výstup byl „dobrý“ pro další kolo**

©Petr Hanáček CLACRYPT Slide 52

Diferenciální kryptoanalýza

- **Budeme pracovat s *rozdíly* vstupu a výstupu**
- **Předpokládejme že známe vstupy X a X**
  - **Pro vstup X je vstup S-boxu X ⊕ K a pro vstup X je vstup S-boxu X ⊕ K**
  - **Klíč K je neznámý**
  - Rozdíl vstupů: **(X ⊕ K) ⊕ (X ⊕ K) = X ⊕ X**
  - **Rozdíl vstupů je tedy nezávislý na klíči K**
- Rozdíl výstupů: **Y ⊕ Y je (téměř) rozdílem vstupů dalšího kola („téměř“ kvůli expanzi)**
- **Cílem je „zřetězit“ rozdíly přes několik kol**
- **Pokud dostaneme známý rozdíl vstupů ze známého rozdílu výstupů…**
  - **Můžeme být schopni řetězit rozdíly přes několik kol**
  - **Funguje jenom pokud to nastává s jistou pravděpodobností**
- **Pokud rozdíl vstupů je 0…**
  - **…rozdíl výstupů je 0**
  - **Umožňuje „zneaktivnit“ některé S-boxy vzhledem k rozdílům**

 CLACRYPT Slide 53

Diferenciální kryptoanalýza

row	column 00	01	10	11
0	10	01	11	00
1	00	10	01	11

- Rozdíl vstupů 000 není zajímavý
- Rozdíl vstupů 010 vždy dává rozdíl výstupů 01
- Čím nerovnoměrnější, tím lepší pro útok

X ⊕ X

	Sbox(X)⊕Sbox(X) 00	01	10	11
000	8	0	0	0
001	0	0	4	4
010	0	8	0	0
011	0	0	4	4
100	0	0	4	4
101	4	4	0	0
110	0	0	4	4
111	4	4	0	0

 CLACRYPT Slide 54

Lineární kryptoanalýza

- **Mitsuru Matsui, 1993.**
- **Pro 16-round DES je třeba 2^{43} známých zpráv, získá se 26 bitů klíče, zbylých 30 bitů útokem silou**
  - **$2^{43} = 9 \times 10^{12}$ bloků zpráv**
- **Podobně jako u diferenciální kryptoanalýzy se soustředíme na nelineární část šifry**
- **Snažíme se nelinearitu aproximovat lineárními rovnicemi**
- **Pro DES chceme aproximovat S-boxy lineárními funkcemi**
  - **Opět využívá slabin v S-boxech, které nebyly navrženy jako odolné proti tomuto přístupu**
- **Pro každé kolo se hledá lineární aproximace (skutečné funkce nejsou lineární)**
- **Hledají se takové aproximace, které platí s pravděpodobností jinou než je ½**

Lineární kryptoanalýza

- Vstup je $x_0x_1x_2$ kde x_0 je řádek a x_1x_2 je sloupec
- Výstup je y_0y_1
- Počítáme jedničky a nuly pro y_0, y_1 a $y_0 \oplus y_1$ v závislosti na vstupu
- Nejlepší je 0 nebo 8, 4 je nepoužitelná

row	column 00	01	10	11
0	10	01	11	00
1	00	10	01	11

		output y_0	y_1	$y_0 \oplus y_1$
	0	4	4	4
i	x_0	4	4	4
n	x_1	4	6	2
p	x_2	4	4	4
u	$x_0 \oplus x_1$	4	2	2
t	$x_0 \oplus x_2$	0	4	4
	$x_1 \oplus x_2$	4	6	6
	$x_0 \oplus x_1 \oplus x_2$	4	6	2

Lineární kryptoanalýza

- Například
  $y_1 = x_1$
  s pravd. 3/4
- nebo
  $y_0 = x_0 \oplus x_2 \oplus 1$
  s pravd. 1
- nebo
  $y_0 \oplus y_1 = x_1 \oplus x_2$
  s pravd. 3/4

row	column 00	01	10	11
0	10	01	11	00
1	00	10	01	11

input		output y_0	y_1	$y_0 \oplus y_1$
	0	4	4	4
i	x_0	4	4	4
n	x_1	4	6	2
p	x_2	4	4	4
u	$x_0 \oplus x_1$	4	2	2
t	$x_0 \oplus x_2$	0	4	4
	$x_1 \oplus x_2$	4	6	6
	$x_0 \oplus x_1 \oplus x_2$	4	6	2

 CLACRYPT Slide 57

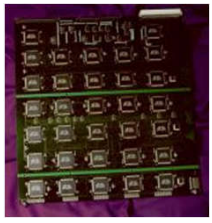
Teoretické útoky silou

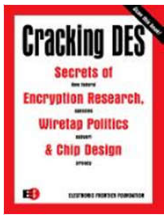
- Diffie a Hellman - 1976
  - Stroj za $20 M s 1 miliónem procesorů, každý s rychlostí 1 M DES keys/sec, zkouší 10^{12} keys/sec a všechny klíče za jeden den
  - Zaplatí se mi ten stroj???
- Michael Wiener - 1993
  - $1 M stroj s 57600 čipy, každý testuje 50 M keys/s, hledá klíč průměrně za 3.5 hodiny (polovina všech klíčů)

 CLACRYPT Slide 58

EFF DES Cracker

- Speciální stroj v ceně 250 000 $
- 27 desek plošných spojů, každá s 64 čipy - 1,728 čipů
- Každý čip obsahuje 24 „procesorů“ (search engine) = 41 472 procesorů
- Při hodinové frekvenci 40 MHz – otestuje 2.5 miliónu klíčů/sec





 CLACRYPT Slide 59

DES Cracker

90B klíčů za sekundu
Cena < $250K (v roce 1998)

 CLACRYPT Slide 60

Praktické útoky silou

Date	Key length	Time	# Computers	Rate (keys/sec)
8/95	40	8 days	120 + 2 super	0.5 M
1/97	40	3.5 hours	250	27 M
2/97	48	13 days	3,500	440 M
6/97	56	4 months	78,000	7 B
2/98	56	39 days	22,000 people	34 B
7/98	56	56 hours	1 EFF with 1,728 chips	90 B
1/99	56	22 hours	100,000 + 1 EFF	250 B

©Petr Hanáček CLACRYPT Slide 61

Jak dlouho bude bezpečný 100bitový klíč?

- **Moorův zákon**
  - **Každých 18 měsíců se zdvojnásobí rychlost a paměťová kapacita**
  - **Každého 1.5 roku získáváme schopnost luštit navíc jeden bit klíče**
- **Pokud dnes je 56bitový klíč rozlomen distribuovaným útokem za asi jeden den, 100bitový bude rozlomen za jeden den za cca 70 let**

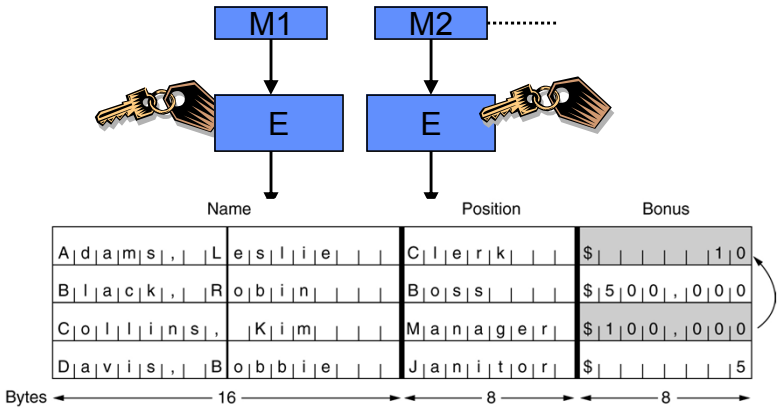
©Petr Hanáček CLACRYPT Slide 62

Režimy blokových šifer

Režimy blokových šifer

- Původně definovány ve FIPS PUB 81
- 4 základní režimy
  - Electronic Codebook (ECB)
  - Cipher Block Chaining (CBC)
  - Output Feedback (OFB)
  - Cipher Feedback (CFB)
- Mohou být použity pro jakoukoli blokovou šifru

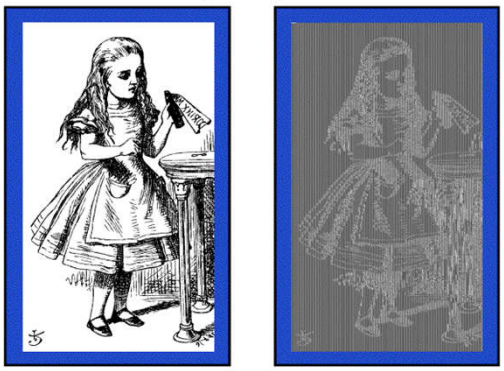
ECB (Electronic Code Book)



- **Umožňuje**
  - **Slovníkové útoky**
  - **Repetici bloků**
  - **Přeskládání bloků**

Vlastnosti ECB

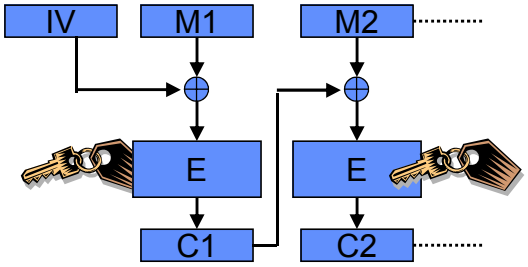
- **Nezkomprimovaný obrázek, zašifrovaný blokovou šifrou v režimu ECB (TEA)**



- **Proč?**
- **Stejný blok otevřeného textu ⇒ stejný blok zašifrovaného textu!**

Cipher Block Chaining (CBC)

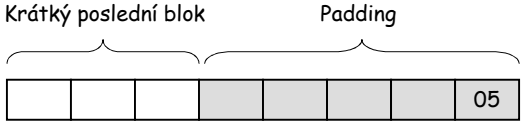
- Každý blok zprávy je xorován z předchozím zašifrovaným blokem
  - $C_i = E(K, (M_i \text{ XOR } C_{i-1}))$
- První blok je xorován s inicializačním vektorem IV



©Petr Hanáček CLACRYPT Slide 67

Vlastnosti CBC

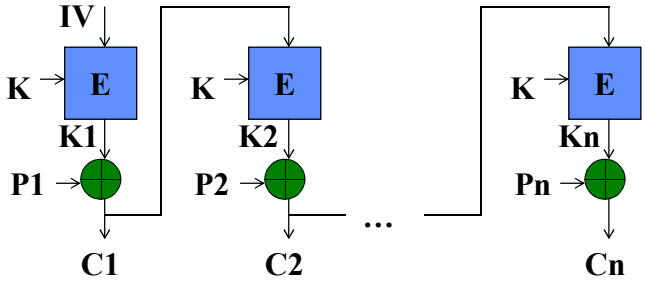
- IV nemusí být tajný, ale je třeba chránit jeho integritu
  - Modifikace IV útočníkem umožňuje útočníkovi cíleně změnit první blok
- Zarovnávání (padding)
  - V případě, že délka zprávy není násobkem délky bloku
  - Doplnění dodatečných bloků na konec
  - Musí být jednoznačné!
  - Jednoznačné zarovnání obvykle znamená vždy prodloužení aspoň o 1 byte
    - » Měl by nést délku zarovnání



©Petr Hanáček CLACRYPT Slide 68

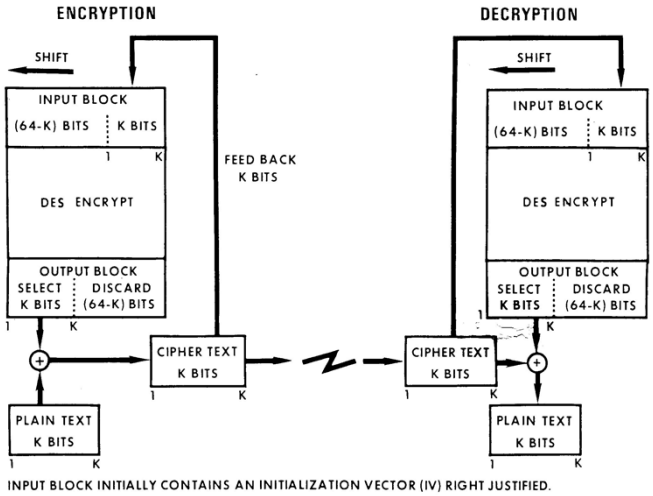
Cipher Feedback (CFB)

- Samosynchronizující proudová šifra
  - DES je použit pro generování pseudonáhodné posloupnosti (key stream)
- DES je inicializován inicializačním vektorem IV
- Vstupní blok DESu je posunut doleva o k bitů a zprava je doplněn k bity ciphertextu
- Typicky k = 8 nebo 64



©Petr Hanáček CLACRYPT Slide 69

FIGURE 3: K-BIT CIPHER FEEDBACK (CFB) MODE

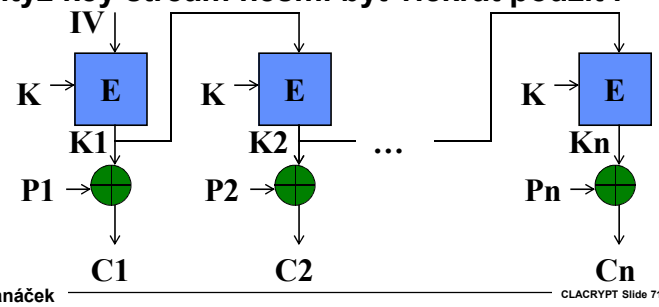


INPUT BLOCK INITIALLY CONTAINS AN INITIALIZATION VECTOR (IV) RIGHT JUSTIFIED.

©Petr Hanáček CLACRYPT Slide 70

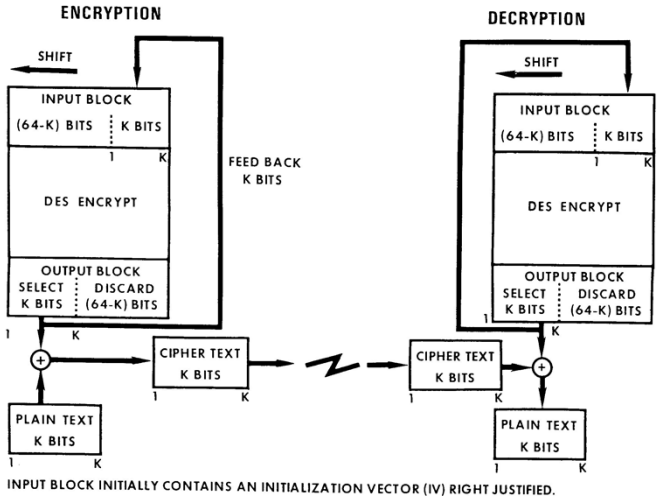
Output Feedback (OFB)

- Synchronní proudová šifra
  - DES je použit pro generování pseudonáhodné posloupnosti (key stream)
- DES je inicializován inicializačním vektorem IV
- Vstupní blok DESu je posunut doleva o k bitů a zprava je doplněn k bity výstupního bloku
  - Typicky k = 8 nebo 64
- Tentýž key stream nesmí být víckrát použit !



©Petr Hanáček CLACRYPT Slide 71

FIGURE 4: K-BIT OUTPUT FEEDBACK (OFB) MODE

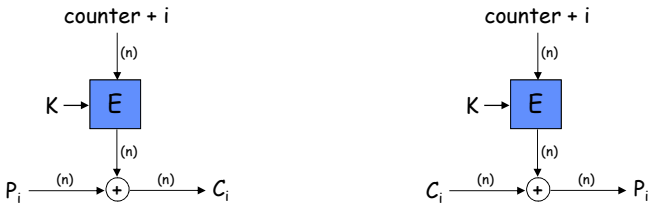


INPUT BLOCK INITIALLY CONTAINS AN INITIALIZATION VECTOR (IV) RIGHT JUSTIFIED.

©Petr Hanáček CLACRYPT Slide 72

Counter (CTR)

- Synchronní proudová šifra
  - DES je použit pro generování pseudonáhodné posloupnosti (key stream)
- Podobný OFB
- Čítač se *nesmí* pro stejný klíč opakovat
- Každý blok je šifrován nezávisle na ostatních
  - Paralelizovatelný



 CLACRYPT Slide 73

Vliv chyby v jednotlivých režimech

Mode	Effect of Bit Errors in C_j or $C^{\#}_j$
ECB	RBE in P_j
CBC	RBE in P_j SBE in P_{j+1}
CFB	SBE in $P^{\#}_j$ RBE in $P^{\#}_{j+1}, P^{\#}_{j+2}, \ldots P^{\#}_{j+w}$
OFB	SBE in P_j
CTR	SBE in P_j

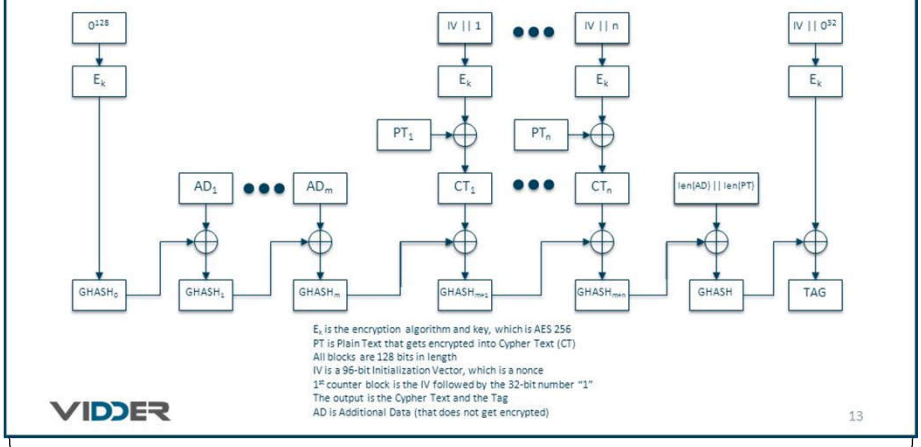
In the Table, SBE (specific bit errors) means that an individual error bit $c_{i,j}$ or $c^{\#}_{i,j}$ produces in the appropriate plaintext block $P_j = (p_{1,j}, p_{2,j}, \ldots, p_{b,j})$ or $P^{\#}_j = (p^{\#}_{1,j}, p^{\#}_{2,j}, \ldots, p^{\#}_{s,j})$ an individual error bit $p_{i,j}$ or $p^{\#}_{i,j}$. In other words, bit errors in the plaintext occur in the same bit positions as the bit errors in the cryptogram.

 CLACRYPT Slide 74

https://www.researchgate.net/publication/235934999_Error_Propagation_in_Various_Cipher_Block_Modes

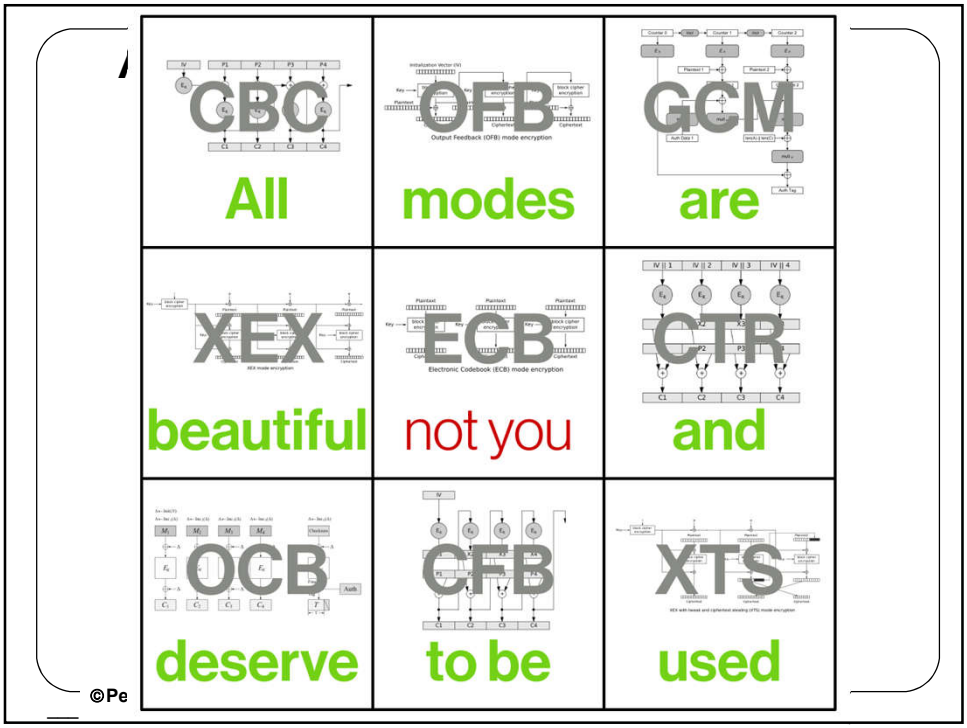
GCM

- **Galois/Counter Mode**



 CLACRYPT Slide 75

https://en.wikipedia.org/wiki/Galois/Counter_Mode

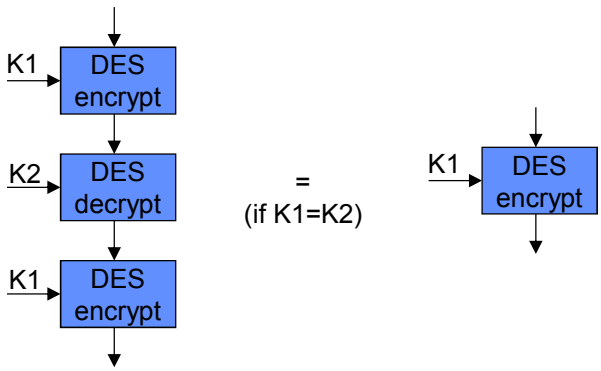


EDE

 CLACRYPT Slide 77

3DES, DES-EDE

- **Velikost bloku 112 bitů**
- **Velikost bloku 64 bitů**
- **ANSI X9.17, ISO 8732 standard**
- **Double DES není bezpečný**
  - **KPA útok Merkle-Hellman meet-in-the-middle - 2^{n+1} pokusů**



 CLACRYPT Slide 78

Double DES?

- Na dvojitou aplikaci šifrovacího algoritmu se dá útočit KPA meet-in-the-middle
- Složitost 2N+1
- Předpoklady:
  - N bitový klíč
  - Známé P a C takové, že C = enc (K_2, enc (K_1, P))
- To znamená, že
  - Existuje M takové, že M = enc(K_1,P) a C = enc(K_2,M)
- Útok
  - Spočti 2^N hodnot X_i = enc (K_i, P)
  - Spočti 2^N hodnot Y_j = dec (K_j, C)
  - Najdi hodnoty K_i a K_j takové, že $X_i = Y_j$
  - "false positives" mohou být snadno odhaleny pomocí další dvojice (P,C)

 CLACRYPT Slide 79

Další blokové šifry

 CLACRYPT Slide 80

International Data Encryption Algorithm (IDEA)

- Šifruje 64-bitové bloky pomocí 128-bitového klíče
- Podobně jako DES:
  - Pracuje v kolech (rounds)
  - Operace je stejná pro šifrování i dešifrování
- Od DESu se liší:
  - Navržena, aby byla efektivní v software
  - Nepoužívá S-boxy a P-boxy
- Základní operace
  - ⊕ XOR
  - \+ sčítání mod 2^{16}
  - x násobení mod $2^{16}+1$

 CLACRYPT Slide 81

IDEA

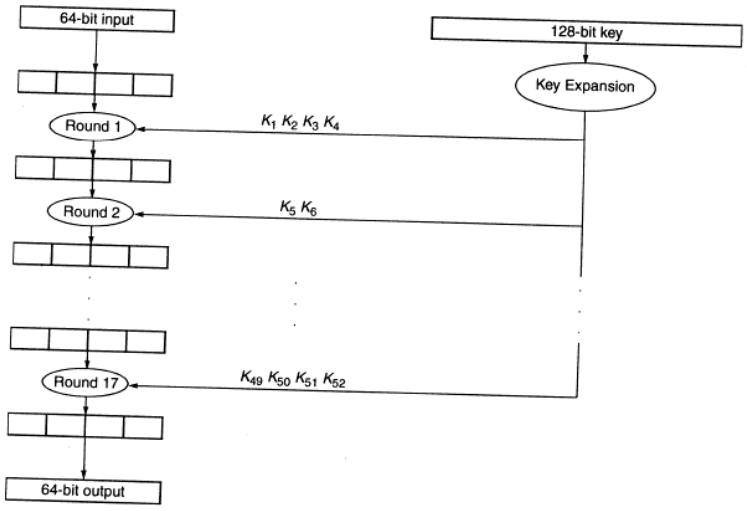


Figure 3-18. Basic Structure of IDEA

 CLACRYPT Slide 82

[From Ravi Mukkamala]

Blowfish

- Šifruje 64-bitové bloky
- Klíč má proměnnou délku, až do 448 bitů
- Vytvořená Bruce Schneierem
- Téměř Feistelova šifra
  - $R_i = L_{i-1} \oplus K_i$
  - $L_i = R_{i-1} \oplus F(L_{i-1} \oplus K_i)$
- Funkce F používá 4 S-boxy
  - Každý S-box převádí 8 bitů na 32 bitů
- Klíčově závislé S-boxy
  - Obsah S-boxu je určen hodnotou klíče

 CLACRYPT Slide 83

TEA, Tiny Encryption Algorithm

- 64 bitový blok, 128 bitový klíč
- Využívá 32-bitovou aritmetiku
- Proměnný počet kol (obvykle 32)
- Používá „slabou“ funkci v každém kole, proto je třeba větší počet kol

 CLACRYPT Slide 84

TEA

- **Algoritmus (předpokládáme 32 kol):**
  - **(K[0],K[1],K[2],K[3]) = 128 bitový klíč**
  - **(L,R) = plaintext (64-bitový blok)**

```
delta = 0x9e3779b9
sum = 0
for i = 1 to 32
     sum += delta
     L += ((R<<4)+K[0])^(R+sum)^((R>>5)+K[1])
     R += ((L<<4)+K[2])^(L+sum)^((L>>5)+K[3])
next I

ciphertext = (L,R)
```

 CLACRYPT Slide 85

TEA

- **Téměř Feistelova šifra**
  - **Používí + a – namísto ⊕ (XOR)**
- **Jednoduchá, snadno implementovatelná, rychlá, paměťově nenáročná**
- **Existuje tzv. „related key attack“**
  - **eXtended TEA (XTEA) eliminuje tento útok**

 CLACRYPT Slide 86

AES: The Next Generation

- **Advanced Encryption Standard (FIPS PUB 197)**
  - Má odstranit nedostatky DES
  - Kandidáti:
    - » Serpent, Rijndael, Twofish, Mars, RC6
  - Založený na algoritmu Rijndael algorithm
    - » Joan Daemen and Vincent Rijmen, Belgie
  - U. S. od Nov. 26, 2001, platný od May 26, 2002
  - Délky klíčů 128, 192 a 256 bitů
  - Velikost bloku 128 bitů
    - » Platí pro AES, Rijndael dovoluje i jiné velikosti
- **Délky klíčů a počet kol**
  - AES-128 – 10 kol
  - AES-192 – 12 kol
  - AES-256 – 14 kol

 CLACRYPT Slide 87

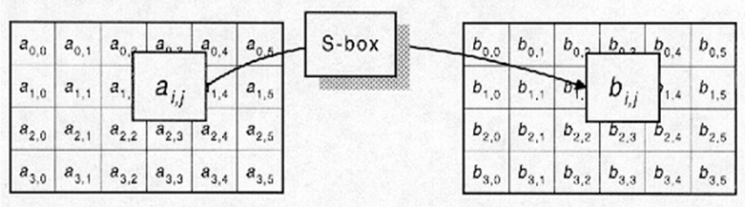
AES

1. Provedení kroku Add Round Key (XOR subklíče s blokem)
2. Provedení kol:
   1. Byte Sub
      - (každý bajt bloku je nahrazen jiným bajtem podle S-boxu)
   2. Shift Row
      - Bajty jsou uspořádány do matice a posunuty
   3. Mix Column
      - Násobení matic
   4. Add round key (XOR subklíče)

 CLACRYPT Slide 88

AES ByteSub

- **Předpokládejme 192-bitový blok, 4x6 bajtů**



- **ByteSub je něco "S-box"**
- **Je to nelineární krok (ale invertovatelný)**

 CLACRYPT Slide 89

AES S-box

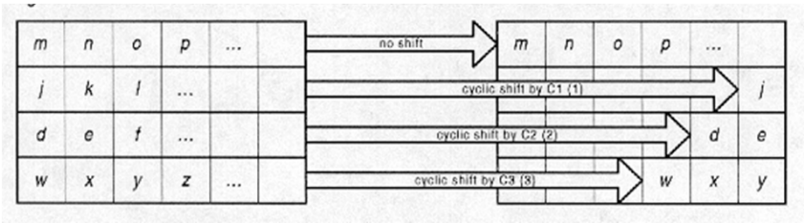
Druhé 4 bity vstupu

První 4 bity vstupu	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

 CLACRYPT Slide 90

AES ShiftRow

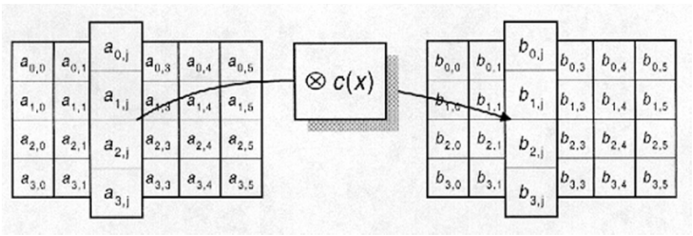
- **Cyklický posuv řádků**



AES MixColumn

Násobení maticí *c* v GF(2^8)
Přes polynom
$x^8+x^4+x^3+x+1$ kde c = 2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2

- **Nelineární, invertovatelná operace, aplikovaná na každý sloupec**



- **Obvykle implementovaná jako velká tabulka**

AES AddRoundKey

- **XOR subklíče s blokem**

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \end{bmatrix} \oplus \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} & k_{04} & k_{05} \\ k_{10} & k_{11} & k_{12} & k_{13} & k_{14} & k_{15} \\ k_{20} & k_{21} & k_{22} & k_{23} & k_{24} & k_{25} \\ k_{30} & k_{31} & k_{32} & k_{33} & k_{34} & k_{35} \end{bmatrix} = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} & b_{05} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \end{bmatrix}$$

Blok Subklíč

- **RoundKey (subklíč) je vytvořen algoritmem** key schedule

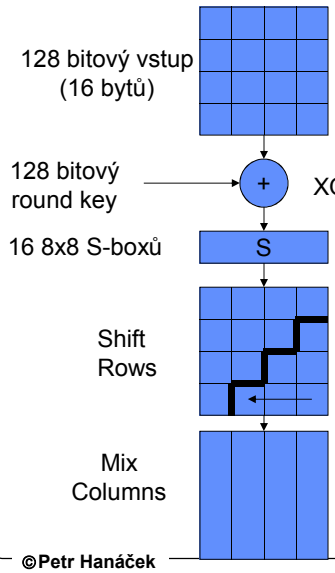
©Petr Hanáček CLACRYPT Slide 93

AES - Dešifrování

- **Proces musí být invertovatelný**
- **Inverze MixAddRoundKey je jednoduchá, ⊕ je inverzní samo k sobě**
- **MixColumn je invertovatelný (pomocí jiné tabulky)**
- **Inverze ShiftRow je jednoduchá (cyklický posuv v opačném směru)**
- **ByteSub je také invertovatelný (jinou tabulkou)**

©Petr Hanáček CLACRYPT Slide 94

AES



S(i) = k xor rol(k,4) xor rol(k,5) xor rol(k,6) xor rol(k,7) xor 0x63, where k = i^{-1} in GF(2^8), rol = rotate byte left

Multiply by M in GF(2^8) over polynomial $x^8+x^4+x^3+x+1$ where M=

$$\begin{matrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{matrix}$$

©Petr Hanáček CLACRYPT Slide 95

Proudové šifry

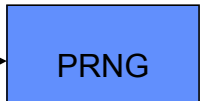
©Petr Hanáček CLACRYPT Slide 96

Definice

- **Bloková šifra**
  - Data jsou rozdělena na bloky pevné délky a po blocích zašifrována
  - V případě potřeby jsou bloky zarovnány (padding)
- **Proudová šifra**
  - Data jsou šifrována bit po bitu (bajt po bajtu), tak, jak jsou předkládána šifrovacímu mechanismu
- **Většina algoritmů jsou blokové šifry**

 CLACRYPT Slide 97

Proudová šifra



Key stream

xor

M → C

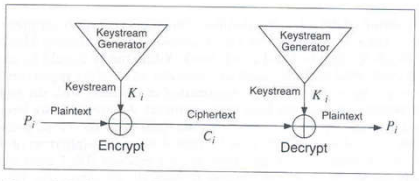


Figure 9.6 Stream cipher.

 CLACRYPT Slide 98

Synchronnní proudové šifry

- Pseudonáhodná posloupnost (key stream) je nezávislá na zprávě
- Zašifrovaný text je obvykle spočten jako XOR otevřeného textu a hodnoty „key stream“
- Např. Vernamova šifra, DES v Output Feedback (OFB) režimu

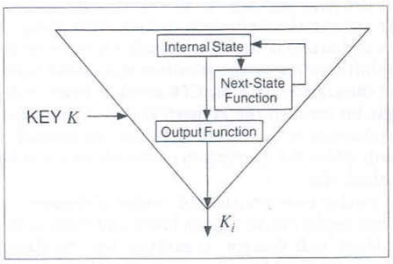


Figure 9.7 Inside a keystream generator.

©Petr Hanáček CLACRYPT Slide 99

Samosynchronizující proudové šifry

- Každý bajt pseudonáhodné posloupnosti je závislý na pevném počtu (např. 1) předcházejících bajtů šifrovaného textu
- Umí se resynchronizovat
- Např.: Autokey, DES v Cipher Feedback (CFB) režimu

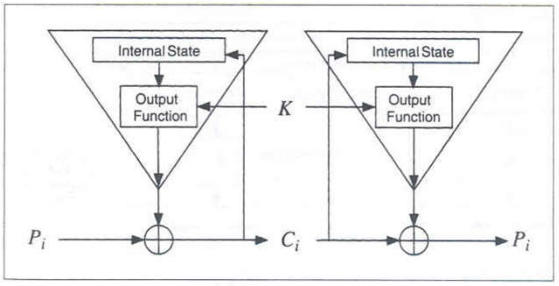


Figure 9.8 A self-synchronizing keystream generator.

©Petr Hanáček CLACRYPT Slide 100

Generátory PRNG

- Generují pseudonáhodnou posloupnost (key stream) z malého klíče a inicializačního vektoru / semínka
- Metody
  - Blokové šifry v režimu OFB
    - » Pomalé (jedno šifrování/byte)
  - Lineární kongruenční generátory
    - » Ne příliš bezpečné
  - LFSR (Linear feedback shift registers)
    - » Rychlé, ne vždy bezpečné
  - Jiné

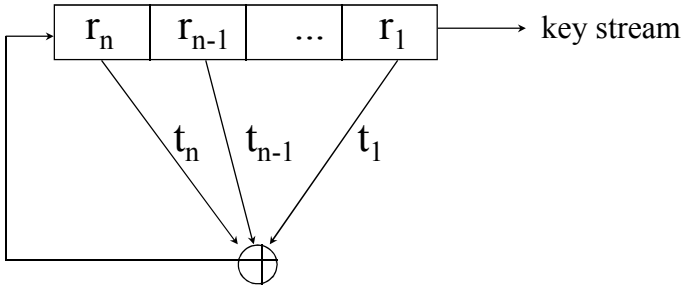
Lineární kongruenční generátory

- Generují pseudonáhodné posloupnosti
  - $x_0, x_1, \ldots, x_{i-1}, x_i, \ldots$
  - kde $x_i = (a * x_{i-1} + b) \bmod n$
- x_0 je „semínko“ (seed)
  - Perioda je menší než n
  - Generátor s maximální periodou má periodu n-1
- Kryptograficky slabé, dokonce i s použitím kvadratické nebo kubické funkce

```
#define MODMULT(a,b,c,m,s) q=s/a; s=b*(s-a*q)-c*q; if(s<0)s+=m;
Double combined LCG( void)
{
    long q ;      long z ;
    MODMULT(53668,40014,12211,2147483563L,s1)
     MODMULT(52774,40692,3791,2147483399L, s2)
     z= s1-s2;
     if( z<1)
       z+=2147483562;
     return z*4.656613e-10;
}
```

LFSR - Linear Feedback Shift Registers

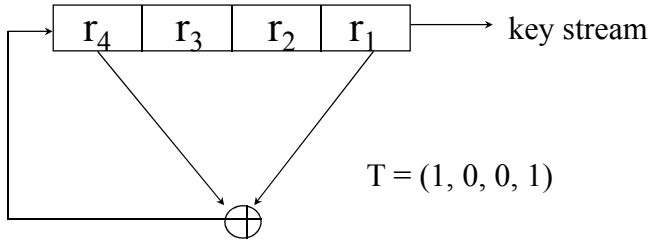
- **Shift registr (binárně) R = $(r_n, r_{n-1}, ..., r_1)$**
- **Tap sequence (binárně) T = $(t_n, t_{n-1}, ..., t_1)$ (zpětné vazby)**
- **Generování bitu:**
  - **r_1 se dá na výstup**
  - **Ostatní bity se posunou doprava**
  - **r_n = TR mod 2 = t_1r_1 XOR t_2r_2 XOR ... XOR t_nr_n.**



©Petr Hanáček CLACRYPT Slide 103

Maximální perioda

- **Maximální perioda je 2^n - 1. Dosáhneme jí, je-li polynom**
  - **$T(x) = t_n x^n + t_{n-1} x^{n-1} + ... + t_1x + 1$**
- **primitivní**
- **Existují tabulky primitivních polynomů**
- **Příklad:**
  - $T(x) = x^4 + x + 1$ je primitivní



©Petr Hanáček CLACRYPT Slide 104

Útoky

- Dá se rozlomit útokem KPA, je třeba 2n bitů plaintext/ciphertext
- Je třeba xorovat M a C a tím získáme 2n bitů key streamu
- 2n bitů key streamu jedinečně určují zpětné vazby (tap sequence) – určí se lineární funkcí (matice)
- Na útok je třeba čas $O(n^3)$

 CLACRYPT Slide 105

Kombinované generátory

- Kombinují několik LFSR pro zvýšení bezpečnosti
- Mohou mít různé délky a různé zpětné vazby
- Mohou mít různé hodiny – hodinový vstup jednoho může záviset na výstupu jiného (feed forward) nebo sebe sama (feedback)
- Způsoby kombinace
  - Nejjednodušší – XOR
  - Multiplexor (Geffe Generator) – jeden LFSR (měl by běžet $\log_2 t$ rychleji než ostatní) určuje, ze kterého z *t* ostatních se vezme následující bit
  - Threshold
    - » Použije se t LFSR (t je liché)
    - » Výstupem generátoru je bit odpovídající majoritě bitů jednotlivých generátorů

 CLACRYPT Slide 106

Kombinované generátory

- Geffe generátor
  - Dva LFSR
    - » Jejich výstupy se přepínají multiplexorem, řízeným dalším LFSR
- Obecný Geffe generátor
  - Několik LFSR
    - » Jejich výstupy se přepínají multiplexorem, řízeným dalším LFSR

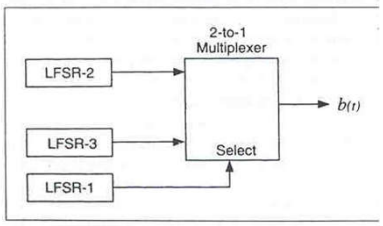


Figure 16.6 Geffe generator.

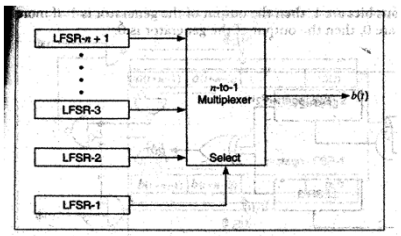


Figure 16.7 Generalized Geffe generator.

Kombinované generátory

- Stop-and-Go generátory
  - Beth-Piper Stop-and-Go generátor
  - Alternující Stop-and-Go generátor

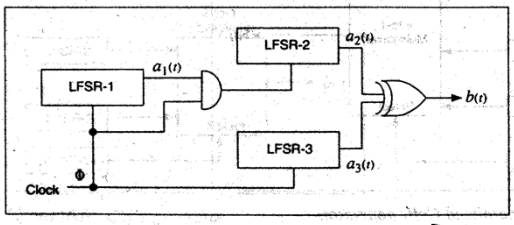


Figure 16.9 Beth-Piper stop-and-go generator.

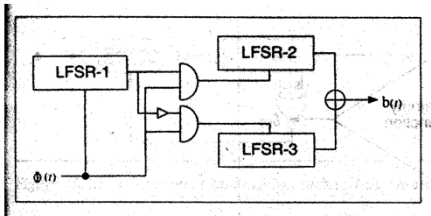


Figure 16.10 Alternating stop-and-go generator.

Kombinované generátory

- **Prahové (threshold) generátory**

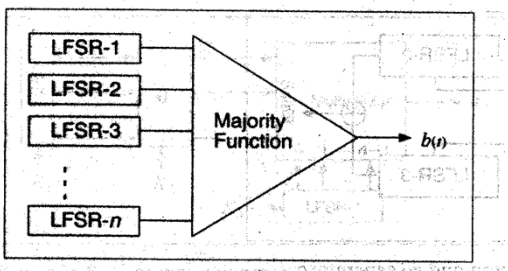
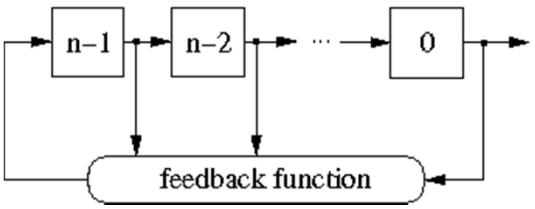


Figure 16.12 Threshold generator.

NLFSR

- **Nelineární FSR**

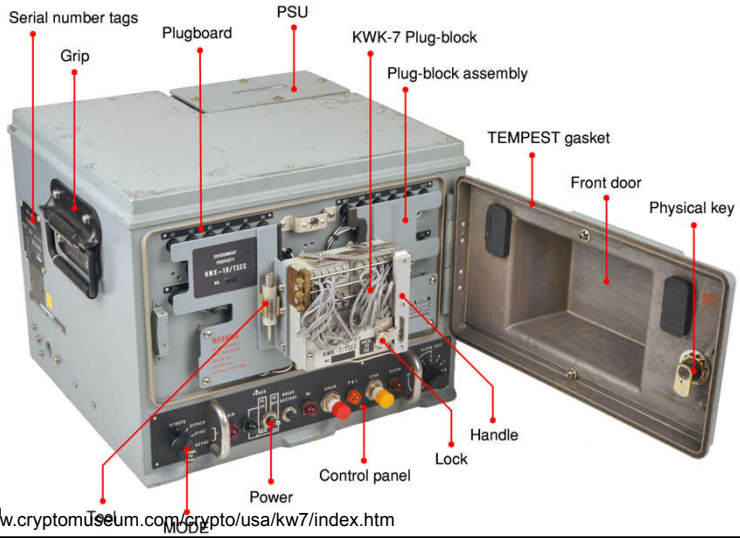


- **Příklady zpětnovazebních funkcí**
  - $x_0 \oplus x_a \oplus x_b \oplus x_c \cdot x_d$
    $x_0 \oplus x_a \oplus x_b \cdot x_c \oplus x_d \cdot x_e$
    $x_0 \oplus x_a \oplus x_b \oplus x_c \oplus x_d \oplus x_e \cdot x_f$
    $x_0 \oplus x_a \oplus x_b \oplus x_c \cdot x_d \cdot x_e$

- **Příklad aplikace**
  - **KEELOQ**

KW-7 Orestes

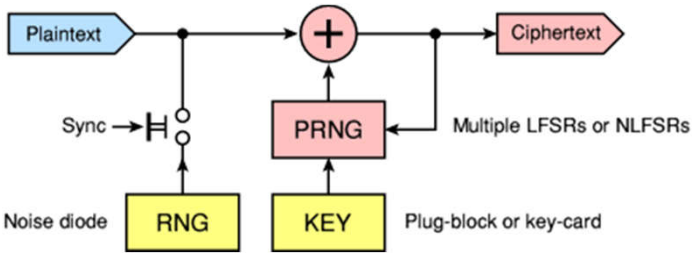
- Vytvořen NSA, používán cca 1960-1990
- Asi první rozšířený elektronický (nerotorový) šifrátor



©Pe
http://www.cryptomuseum.com/crypto/usa/kw7/index.htm

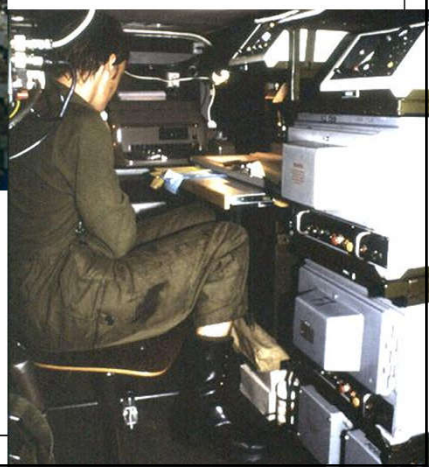
KW-7 Orestes

- Rozluštěn Sověty
  - The most famous spying case is that of John Anthony Walker, born 1937, who worked for the US Navy and successfully spied for the Russians for nearly 17 years. Walker joined the US Navy in 1955 and started spying for the Soviets in December 1967, when he had financial difficulties.
  - From that moment, until his retirement from the navy in 1983, he supplied the Russians with the key lists and other critical cipher material of the KL-47, the KW-7 and other encryption systems.



©Petr Hanáček CLACRYPT Slide 112
http://www.cryptomuseum.com/crypto/usa/kw7/index.htm

KW-7 Orestes



©Petr Hanáček

http://www.cryptomuseum.com/crypto/usa/kw7/index.htm

A5

- Použita v GSM -- Global System Mobile (or Group Special Mobile)
- A5/1 původně pro Evropu A USA
  - A5/1 vytvořen v roce 1987
- Utajované
  - Obecný popis unikl v roce 1994
  - Zreverzovány v roce 1999 z GSM telefonu
- A5/2 úmyslně oslabená verze pro méně důvěryhodné země
  - A5/2 vytvořen v roce 1989
- A5/3
  - Málo rozšířený
    - » Specifikován 2002
    - » Testy interoperability 2009

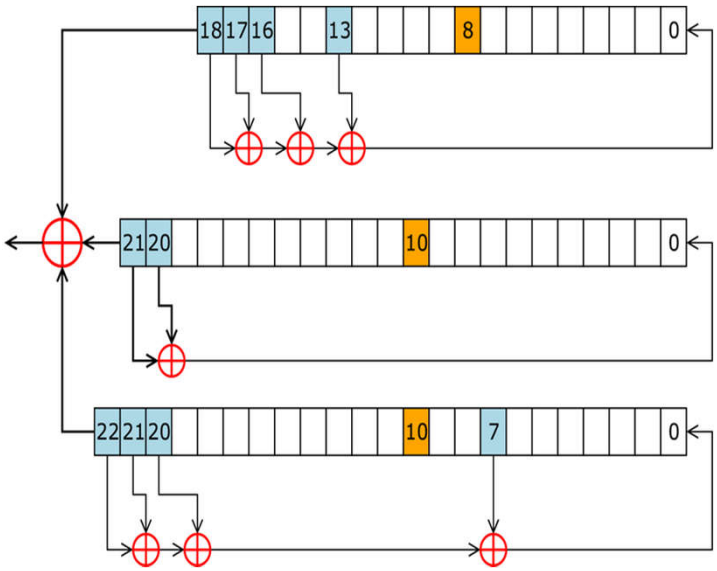
©Petr Hanáček CLACRYPT Slide 114

A5

- Pro zašifrování dat přenášených rádiovým kanálem (over-the-air link)
- 3 LFSR s délkami 19, 22, 23 – výstupy jsou XORovány
- Každý register má hodiny ze svého prostředního bitu XORovaného výstupem funkce (threshold) prostředních bitů všech tří registrů
- Registry nejsou příliš dlouhé

©Petr Hanáček CLACRYPT Slide 115

A5/1



©Petr Hanáček CLACRYPT Slide 116

Bezpečnost A5/1

- **Útoky**
  - **Útok silou 2^{64} (nebo spíše 2^{54})**
  - **Útok na dva menší registry 2^{45}**
- **Skutečný útok**
  - **Jsou třeba 2 sekundy plaintextu**
  - **Je třeba 2^{48} předvýpočtů, 148 GB prostoru**
  - **Doba útoku 1 minuta na PC**

 CLACRYPT Slide 117

A5/2

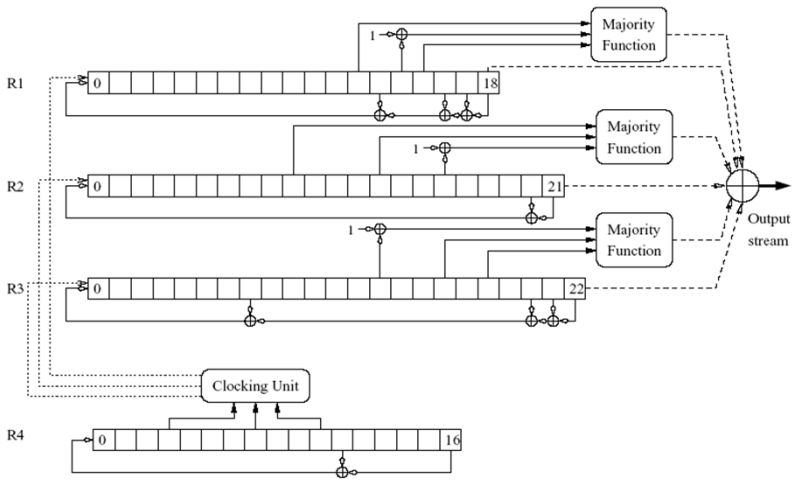


Fig. 1. The A5/2 internal structure

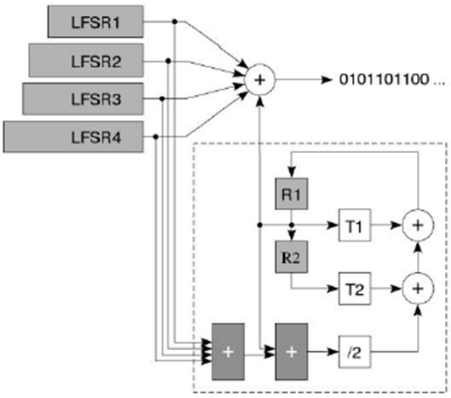
 CLACRYPT Slide 118

A5/2

- A5/2 byl zreverzován na konci 90 let
- Je luštitelný v reálném čase
- V roce 2003 konečně ETSI/3GPP/GSMA uznaly, že je zde problém
- Problém je ještě horší:
  - Jelikož generování klíče pro A5/1 a A5/2 je stejné, je možné si nechat vygenerovat klíč pro A5/2 a rozluštit s ním zachycenou komunikaci A5/1 (downgrading attack)
  - Jediným řešení je odstranění A5/2 ze všech zařízení
  - Teprve 2004 byl A5/2 vyjmut ze standardu
- Pozn.: Generování klíče pro A5/3 a A5/1 je stejné, opět půjde provést downgrading attack

©Petr Hanáček CLACRYPT Slide 119

Proudová šifra Bluetooth



- Délka klíče 128 bitů
- Nejlepší známý útok 2^{70}

©Petr Hanáček CLACRYPT Slide 120

Celulární kryptografie

	Confidentiality	Authentication	Keying
US Analog	None	None	None
US Digital	XOR mask & CMEA (ORYX)	CAVE	CAVE
GSM	A5/0, A5/2, or A5/1 (soon: A5/3)	COMP128 (COMP128-2, 3DES-CBC-MAC)	COMP128 (same)

Key: = insecure

PKZIP

- Proudová šifra, bere text po bajtech
- 32 bitové hodnoty
- 3 magická čísla

  K_0 = 305419896
  K_1 = 591751049
  K_2 = 878082192
- Bezpečnost PKZIPu
  - Nedostatečná
  - Časová složitost 2^{27}
  - Nevhodné k použití

$C_i = P_i \wedge K_3$

$K_0 = \text{crc32}(K_0, P_i)$
$K_1 = K_1 + (K_0 \,\&\, 0x000000ff)$
$K_1 = K_1 * 134775813 + 1$
$K_2 = \text{crc32}(K_2, K_1 >> 24)$
$K_3 = ((K_2 \mid 2) * ((K_2 \mid 2) \wedge 1)) >> 8$

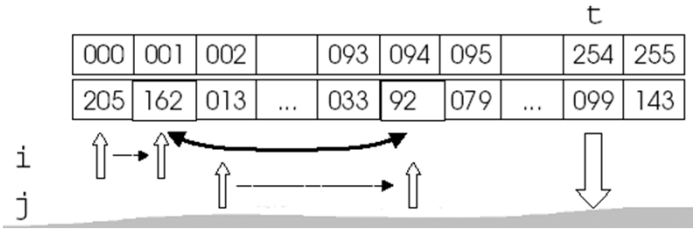
RC4

- Ron Rivest, 1987
- Proměnná délka klíče
- Používá S-box s prvky S_0, ..., S_{255} které jsou permutací osmibitových čísel 0 ... 255
- Inicializace
  - Set $S_i = i$ for i = 0, ..., 255
  - Fill K_0, ..., K_{255} with the key bits - repeat as necessary (key is variable length) - each K_i is one byte (8 bits)
  - j = 0;
  - for i = 0 to 255 do begin "scramble the S_i"
    j = (j + S_i + K_i) mod 256;
    swap S_i and S_j
    end

 CLACRYPT Slide 123

RC4 – generování posloupnosti

- i = 0; j = 0
- next byte of key stream is K where
  i = (i + 1) mod 256
  j = (j + S_i) mod 256
  swap S_i and S_j
  t = (S_i + S_j) mod 256
  K = S_t



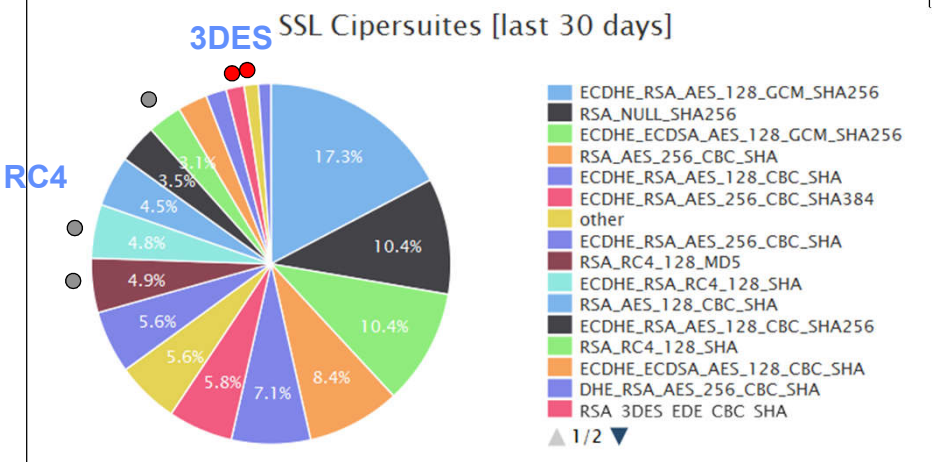
 CLACRYPT Slide 124

Slabiny RC4

- Nezajišťuje integritu
  - Je třeba ji zajistit jinak (např. MAC)
- Problém při zašifrování dvou zpráv stejným klíčem
  - Modifikace klíče
- Útok: Fluhrer, Mantin a Shamir
  - Počáteční keystream není dostatečně náhodný
  - Pokud je klíč pouze konkatenován s nágodným číslem, lze jej zrekonstruovat
  - Použito pro rozlomení šifrování WEP použitého pro WiFi sítě podle standardu 802.11b
- RC4 se používá
  - SSL TLS
  - WPA TKIP

 CLACRYPT Slide 125

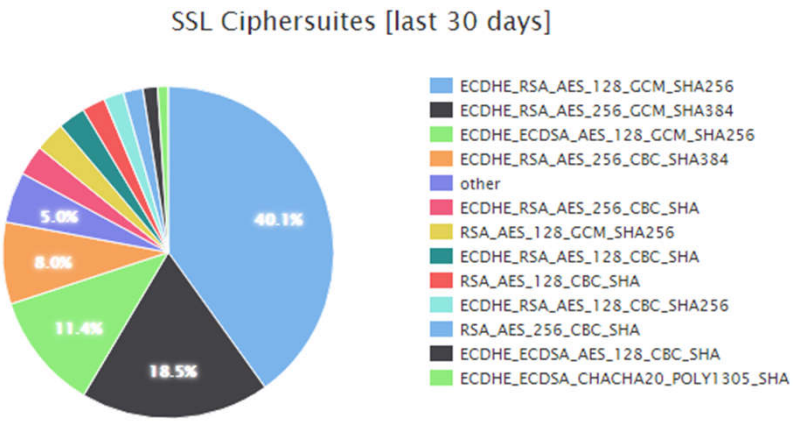
SSL algoritmy



Podzim 2015
http://notary.icsi.berkeley.edu/

 CLACRYPT Slide 126

SSL algoritmy



RC4
+3DES
+xxx

Jaro 2017
http://notary.icsi.berkeley.edu/

©Petr Hanáček CLACRYPT Slide 127

Slabiny RC4 - 2015

- Článek „All your RC4 Ciphers Are Belong to Us“
  - http://www.rc4nomore.com/vanhoef-usenix2015.pdf
- Which weaknesses in RC4 does the attack abuse?
  - It relies on two types of statistical biases present in the keystream.
    - » The first one is that two consecutive bytes are biased towards certain values. These are commonly called the Fluhrer-McGrew biases.
      - http://www.mindspring.com/~dmcgrew/rc4-03.pdf
    - » The second type of biases is that a pair of consecutive bytes is likely to repeat itself. These are called the Mantin's ABSAB biases. Both types of biases are combined in our attack.
      - http://link.springer.com/chapter/10.1007/11426639_29
  - These biases allow us to decrypt repeated plaintext such as cookies.
- Our attack against WPA-TKIP takes only an hour to execute, and allows an attacker to inject and decrypt arbitrary packets.

©Petr Hanáček CLACRYPT Slide 128

Odstranění RC4

18. března 2016

Microsoft oznámil , že v dohledné době výchozím odstraní podporu pro RC4 ve svých internetových prohlížečích Edge a Internet Explorer (11). Jde o algoritmus, který se využívá například při šifrovaném přenosu u webových stránek či při zabezpečení bezdrátových sítí. Historie šifry RC4 sahá ještě do roku 1987.

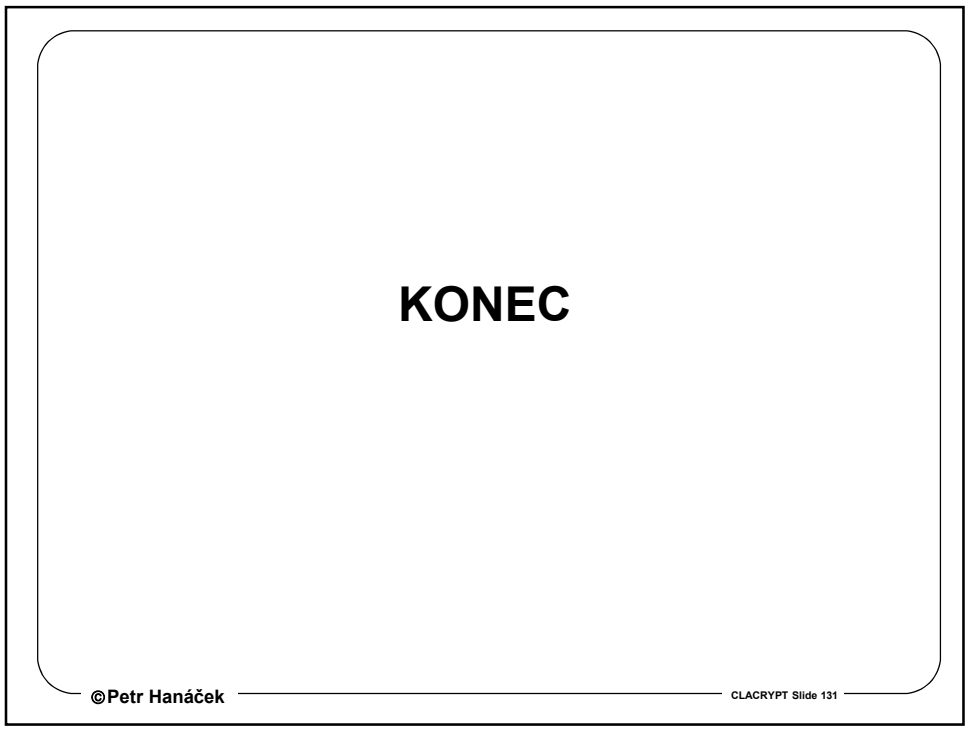
RC4

Šifra široce používaná u běžně používaných protokolů SSL/TLS pro HTTPS nebo WEP a WPA u Wi-Fi sítí. Byla oblíbená pro svou rychlost, jednoduchost a snadnou implementaci. Aktuálně už je ale překonána modernějšími alternativami.

Problémy proudových šifer

- **Je třeba dobře volit IV**
  - **Nesmí se použít stejný klíč pro šifrování dvou zpráv**
  - **Hodně algoritmů nepodporuje explicitní IV !!!**
  - **C1 = P1 xor Keystream**
  - **C2 = P2 xor Keystream**
  - **C1 xor C2 = P1 xor P2**
- **Je třeba dát pozor na integritu**
  - **Útočník může změnit : “PAY ALICE $1M” na “PAY NIKITA$1M” jenom překlopením bitů**

 CLACRYPT Slide 130


KONEC
©Petr Hanáček
CLACRYPT Slide 131

CHAPTER 8

Block Ciphers

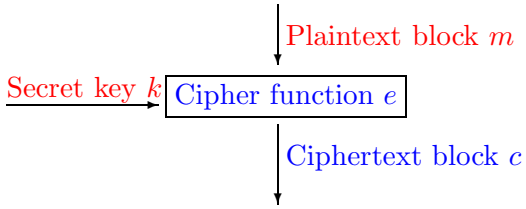
Chapter Goals

- To introduce the notion of block ciphers.
- To understand the workings of the DES algorithm.
- To understand the workings of the Rijndael algorithm.
- To learn about the various standard modes of operation of block ciphers.

1. Introduction To Block Ciphers

The basic description of a block cipher is shown in Fig. 1. Block ciphers operate on blocks

FIGURE 1. Operation of a block cipher



of plaintext one at a time to produce blocks of ciphertext. The main difference between a block cipher and a stream cipher is that block ciphers are stateless, whilst stream ciphers maintain an internal state which is needed to determine which part of the keystream should be generated next. We write

$$c = e_k(m),$$
$$m = d_k(c)$$

where

- m is the plaintext block,
- k is the secret key,
- e is the encryption function,
- d is the decryption function,
- c is the ciphertext block.

The block sizes taken are usually reasonably large, 64 bits in DES and 128 bits or more in modern block ciphers. Often the output of the ciphertext produced by encrypting the first block is used to help encrypt the second block in what is called a *mode of operation*. These modes are used to avoid certain attacks based on deletion or insertion by giving each ciphertext block a context within the overall message. Each mode of operation offers different protection against error propagation due to transmission errors in the ciphertext. In addition, depending on the mode of operation (and the application) message/session keys may be needed. For example, many modes require a per message

initial value to be input into the encryption and decryption operations. Later in this chapter we shall discuss modes of operation of block ciphers in more detail.

There are many block ciphers in use today, some which you may find used in your web browser are RC5, RC6, DES or 3DES. The most famous of these is DES, or the Data Encryption Standard. This was first published in the mid-1970s as a US Federal standard and soon become the de-facto international standard for banking applications.

The DES algorithm has stood up remarkably well to the test of time, but in the early 1990s it became clear that a new standard was required. This was because both the block length (64 bits) and the key length (56 bits) of basic DES were too small for future applications. It is now possible to recover a 56-bit DES key using either a network of computers or specialized hardware. In response to this problem the US National Institute for Standards and Technology (NIST) initiated a competition to find a new block cipher, to be called the Advanced Encryption Standard or AES.

Unlike the process used to design DES, which was kept essentially secret, the design of the AES was performed in public. A number of groups from around the world submitted designs for the AES. Eventually five algorithms, known as the AES finalists, were chosen to be studied in depth. These were

- MARS from a group at IBM,
- RC6 from a group at RSA Security,
- Twofish from a group based at Counterpane, UC Berkeley and elsewhere,
- Serpent from a group of three academics based in Israel, Norway and the UK,
- Rijndael from a couple of Belgian cryptographers.

Finally in the fall of 2000, NIST announced that the overall AES winner had been chosen to be Rijndael.

DES and all the AES finalists are examples of *iterated* block ciphers. The block ciphers obtain their security by repeated use of a simple *round function*. The round function takes an n-bit block and returns an n-bit block, where n is the block size of the overall cipher. The number of rounds r can either be a variable or fixed. As a general rule increasing the number of rounds will increase the level of security of the block cipher.

Each use of the round function employs a round key

$$k_i \text{ for } 1 \leq i \leq r$$

derived from the main secret key k, using an algorithm called a *key schedule*. To allow decryption, for every round key the function implementing the round must be invertible, and for decryption the round keys are used in the opposite order that they were used for encryption. That the whole round is invertible does not imply that the functions used to implement the round need to be invertible. This may seem strange at first reading but will become clearer when we discuss the DES cipher later. In DES the functions needed to implement the round function are not invertible, but the whole round is invertible. For Rijndael not only is the whole round function invertible but every function used to create the round function is also invertible.

There are a number of general purpose techniques which can be used to break a block cipher, for example: exhaustive search, using pre-computed tables of intermediate values or divide and conquer. Some (badly designed) block ciphers can be susceptible to chosen plaintext attacks, where encrypting a specially chosen plaintext can reveal properties of the underlying secret key. In cryptanalysis one needs a combination of mathematical and puzzle-solving skills, plus luck. There are a few more advanced techniques which can be employed, some of which apply in general to any cipher (and not just a block cipher).

- **Differential Cryptanalysis:** In differential cryptanalysis one looks at ciphertext pairs, where the plaintext has a particular difference. The exclusive-or of such pairs is called a

differential and certain differentials have certain probabilities associated to them, depending on what the key is. By analysing the probabilities of the differentials computed in a chosen plaintext attack one can hope to reveal the underlying structure of the key.

- **Linear Cryptanalysis:** Even though a good block cipher should contain non-linear components the idea behind linear cryptanalysis is to approximate the behaviour of the non-linear components with linear functions. Again the goal is to use a probabilistic analysis to determine information about the key.

Surprisingly these two methods are quite successful against some ciphers. But they do not appear that successful against DES or Rijndael, two of the most important block ciphers in use today.

Since DES and Rijndael are likely to be the most important block ciphers in use for the next few years we shall study them in some detail. This is also important since they both show general design principles in their use of substitutions and permutations. Recall that the historical ciphers made use of such operations, so we see that not much has changed. Now, however, the substitutions and permutations used are far more intricate. On their own they do not produce security, but when used over a number of rounds one can obtain enough security for our applications.

We end this section by discussing which is best, a block cipher or a stream cipher? Alas there is no correct answer to this question. Both have their uses and different properties. Here are just a few general points.

- Block ciphers are more general, and we shall see that one can easily turn a block cipher into a stream cipher.
- Stream ciphers generally have a more mathematical structure. This either makes them easier to break or easier to study to convince oneself that they are secure.
- Stream ciphers are generally not suitable for software, since they usually encrypt one bit at a time. However, stream ciphers are highly efficient in hardware.
- Block ciphers are suitable for both hardware and software, but are not as fast in hardware as stream ciphers.
- Hardware is always faster than software, but this performance improvement comes at the cost of less flexibility.

2. Feistel Ciphers and DES

The DES cipher is a variant of the basic Feistel cipher described in Fig. 2, named after H. Feistel who worked at IBM and performed some of the earliest non-military research on encryption algorithms. The interesting property of a Feistel cipher is that the round function is invertible regardless of the choice of the function in the box marked F. To see this notice that each encryption round is given by

$$\begin{aligned} L_i &= R_{i-1}, \\ R_i &= L_{i-1} \oplus F(K_i, R_{i-1}). \end{aligned}$$

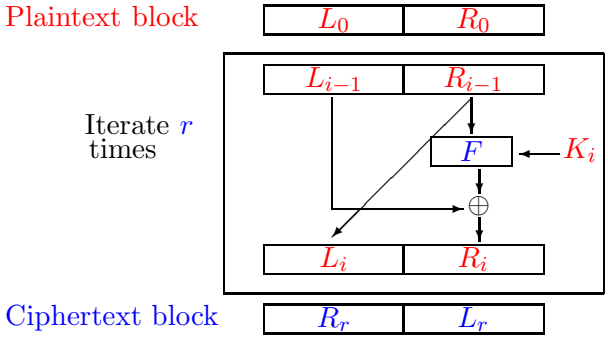
Hence, the decryption can be performed via

$$\begin{aligned} R_{i-1} &= L_i, \\ L_{i-1} &= R_i \oplus F(K_i, L_i). \end{aligned}$$

This means that in a Feistel cipher we have simplified the design somewhat, since

- we can choose any function for the function F, and we will still obtain an encryption function which can be inverted using the secret key,

FIGURE 2. Basic operation of a Feistel cipher



- the same code/circuitry can be used for the encryption and decryption functions. We only need to use the round keys in the reverse order for decryption.

Of course to obtain a secure cipher we still need to take care with

- how the round keys are generated,
- how many rounds to take,
- how the function F is defined.

Work on DES was started in the early 1970s by a team in IBM which included Feistel. It was originally based on an earlier cipher of IBM's called Lucifer, but some of the design was known to have been amended by the National Security Agency, NSA. For many years this led the conspiracy theorists to believe that the NSA had placed a trapdoor into the design of the function F. However, it is now widely accepted that the modifications made by the NSA were done to make the cipher more secure. In particular, the changes made by the NSA made the cipher resistant to differential cryptanalysis, a technique that was not discovered in the open research community until the 1980s.

DES is also known as the Data Encryption Algorithm DEA in documents produced by the American National Standards Institute, ANSI. The International Standards Organisation ISO refers to DES by the name DEA-1. It has been a world-wide standard for well over twenty years and stands as the first publicly available algorithm to have an 'official status'. It therefore marks an important step on the road from cryptography being a purely military area to being a tool for the masses.

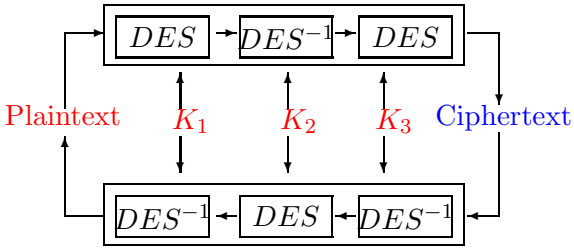
The basic properties of the DES cipher are that it is a variant of the Feistel cipher design with

- the number of rounds r is 16,
- the block length n is 64 bits,
- the key length is 56 bits,
- the round keys $K_1, \ldots, K_{16}$ are each 48 bits.

Note that a key length of 56 bits is insufficient for many modern applications, hence often one uses DES by using three keys and three iterations of the main cipher. Such a version is called Triple DES or 3DES, see Fig. 3. In 3DES the key length is equal to 168. There is another way of using DES three times, but using two keys instead of three giving rise to a key length of 112. In this two-key version of 3DES one uses the 3DES basic structure but with the first and third key being equal. However, two-key 3DES is not as secure as one might inititally think.

2.1. Overview of DES Operation. Basically DES is a Feistel cipher with 16 rounds, as depicted in Fig. 4, except that before and after the main Feistel iteration a permutation is performed. Notice how the two blocks are swapped around before being passed through the final inverse permutation. This permutation appears to produce no change to the security, and people have often wondered why it is there. One answer given by one of the original team members was that this permutation was there to make the original implementation easier to fit on the circuit board.

FIGURE 3. Triple DES

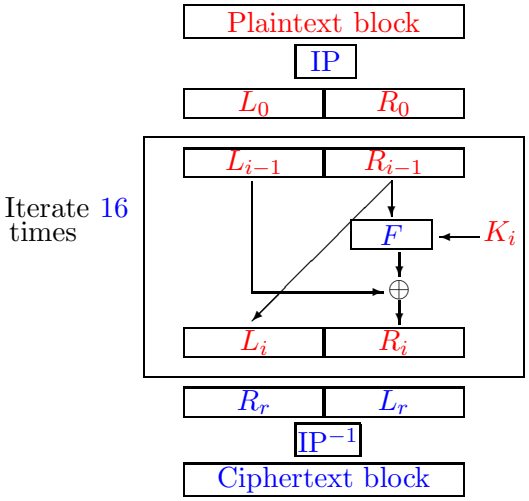


In summary the DES cipher operates on 64 bits of plaintext in the following manner:

- Perform an initial permutation.
- Split the blocks into left and right half.
- Perform 16 rounds of identical operations.
- Join the half blocks back together.
- Perform a final permutation.

The final permutation is the inverse of the initial permutation, this allows the same hardware/software to be used for encryption and decryption. The key schedule provides 16 round keys of 48 bits in length by selecting 48 bits from the 56-bit main key.

FIGURE 4. DES as a Feistel cipher



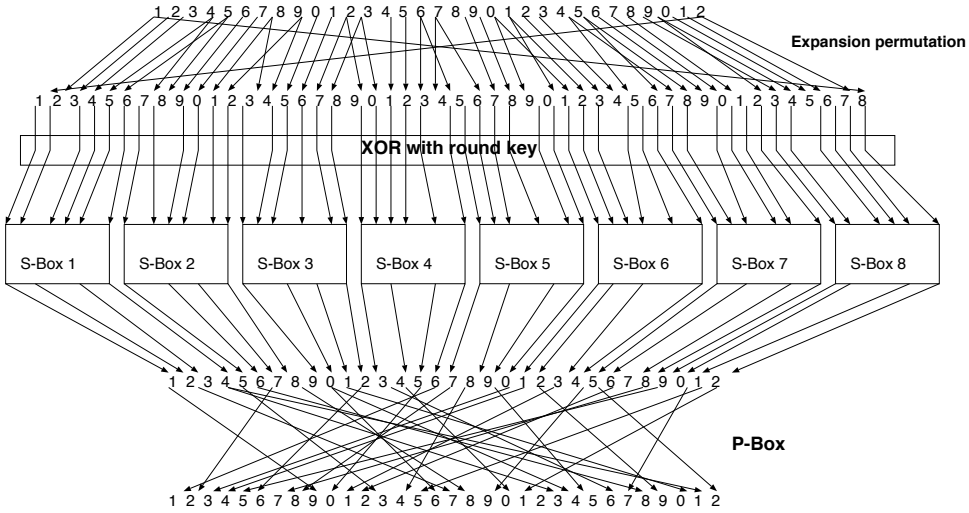
We shall now describe the operation of the function F. In each DES round this consists of the following six stages:

- **Expansion Permutation:** The right half of 32 bits is expanded and permuted to 48 bits. This helps the diffusion of any relationship of input bits to output bits. The expansion permutation (which is different from the initial permutation) has been chosen so that one bit of input affects two substitutions in the output, via the S-Boxes below. This helps spread dependencies and creates an avalanche effect (a small difference between two plaintexts will produce a very large difference in the corresponding ciphertexts).
- **Round Key Addition:** The 48-bit output from the expansion permutation is XORed with the round key, which is also 48 bits in length. Note, this is the only place where the round key is used in the algorithm.
- **Splitting:** The resulting 48-bit value is split into eight lots of six-bit values.

- **S-Box:** Each six-bit value is passed into one of eight different S-Boxes (Substitution Box) to produce a four-bit result. The S-Boxes represent the non-linear component in the DES algorithm and their design is a major contributor to the algorithms security. Each S-Box is a look-up table of four rows and sixteen columns. The six input bits specify which row and column to use. Bits 1 and 6 generate the row number, whilst bits $2, 3, 4$ and 5 specify the column number. The output of each S-Box is the value held in that element in the table.
- **P-Box:** We now have eight lots of four-bit outputs which are then combined into a 32-bit value and permuted to form the output of the function F.

The overall structure of DES is explained in Fig. 5.

FIGURE 5. Structure of the DES function F



We now give details of each of the steps which we have not yet fully defined.

2.1.1. *Initial Permutation, IP:.* The DES initial permutation is defined in the following table. Here the 58 in the first position means that the first bit of the output from the IP is the 58th bit of the input, and so on.

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

The inverse permutation is given in a similar manner by the following table.

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

2.1.2. *Expansion Permutation, E:.* The expansion permutation is given in the following table. Each row corresponds to the bits which are input into the corresponding S-Box at the next stage. Notice how the bits which select a row of one S-Box (the first and last bit on each row) are also used to select the column of another S-Box.

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

2.1.3. *S-Box:* The details of the eight DES S-Boxes are given in the Fig. 6. Recall that each box consists of a table with four rows and sixteen columns.

2.1.4. *The P-Box Permutation, P:.* The P-Box permutation takes the eight lots of four-bit nibbles, output of the S-Boxes, and produces a 32-bit permutation of these values as given by the following table.

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

2.2. DES Key Schedule. The DES key schedule takes the 56-bit key, which is actually input as a bitstring of 64 bits comprising of the key and eight parity bits, for error detection. These parity bits are in bit positions $8, 16, \ldots, 64$ and ensure that each byte of the key contains an odd number of bits.

We first permute the bits of the key according to the following permutation (which takes a 64-bit input and produces a 56-bit output, hence discarding the parity bits).

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

FIGURE 6. DES S-Boxes

S-Box 1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-Box 2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S-Box 3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S-Box 4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S-Box 5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S-Box 6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S-Box 7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S-Box 8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

The output of this permutation, called PC-1 in the literature, is divided into a 28-bit left half C_0 and a 28-bit right half D_0. Now for each round we compute

$$C_i = C_{i-1} \lll p_i,$$
$$D_i = D_{i-1} \lll p_i,$$

where $x \lll p_i$ means perform a cyclic shift on x to the left by p_i positions. If the round number i is $1, 2, 9$ or 16 then we shift left by one position, otherwise we shift left by two positions.

Finally the two portions C_i and D_i are joined back together and are subject to another permutation, called PC-2, to produce the final 48-bit round key. The permutation PC-2 is described below.

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

3. Rijndael

The AES winner was decided in fall 2000 to be the Rijndael algorithm designed by Daemen and Rijmen. Rijndael is a block cipher which does not rely on the basic design of the Feistel cipher. However, Rijndael does have a number of similarities with DES. It uses a repeated number of rounds to obtain security and each round consists of substitutions and permutations, plus a key addition phase. Rijndael in addition has a strong mathematical structure, as most of its operations are based on arithmetic in the field $\mathbb{F}_{2^8}$. However, unlike DES the encryption and decryption operations are distinct.

Recall that elements of $\mathbb{F}_{2^8}$ are stored as bit vectors (or bytes) representing binary polynomials. For example the byte given by $0x83$ in hexadecimal, gives the bit pattern

$$1, 0, 0, 0, 0, 0, 1, 1$$

since

$$0x83 = 8 \cdot 16 + 3 = 131$$

in decimal. One can obtain the bit pattern directly by noticing that 8 in binary is $1, 0, 0, 0$ and 3 in 4-bit binary is $0, 0, 1, 1$ and one simply concatenates these two bit strings together. The bit pattern itself then corresponds to the binary polynomial

$$x^7 + x + 1.$$

So we say that the hexadecimal number $0x83$ represents the binary polynomial

$$x^7 + x + 1.$$

Arithmetic in $\mathbb{F}_{2^8}$ is performed using polynomial arithmetic modulo the irreducible polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Rijndael identifies 32-bit words with polynomials in $\mathbb{F}_{2^8}[X]$ of degree less than four. This is done in a big-endian format, in that the smallest index corresponds to the least important coefficient. Hence, the word

$$a_0 \| a_1 \| a_2 \| a_3$$

will correspond to the polynomial

$$a_3 X^3 + a_2 X^2 + a_1 X + a_0.$$

Arithmetic is performed on polynomials in $\mathbb{F}_{2^8}[X]$ modulo the reducible polynomial

$$M(X) = X^4 + 1.$$

Hence, arithmetic is done on these polynomials in a ring rather than a field, since $M(X)$ is reducible.

Rijndael is a parametrized algorithm in that it can operate on block sizes of 128, 192 or 256 bits, it can also accept keys of size 128, 192 or 256 bits. For each combination of block and key size a different number of rounds is specified. To make our discussion simpler we shall consider

the simpler, and probably more used, variant which uses a block size of 128 bits and a key size of 128 bits, in which case 10 rounds are specified. From now on our discussion is only of this simpler version.

Rijndael operates on an internal four-by-four matrix of bytes, called the state matrix

$$S = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix},$$

which is usually held as a vector of four 32-bit words, each word representing a column. Each round key is also held as a four-by-four matrix

$$K_i = \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix}.$$

3.1. Rijndael Operations. The Rijndael round function operates using a set of four operations which we shall first describe.

3.1.1. *SubBytes:* There are two types of S-Boxes used in Rijndael: One for the encryption rounds and one for the decryption rounds, each one being the inverse of the other. We shall describe the encryption S-Box, the decryption one will follow immediately. The S-Boxes of DES were chosen by searching through a large space of possible S-Boxes, so as to avoid attacks such as differential cryptanalysis. The S-Box of Rijndael is chosen to have a simple mathematical structure, which allows one to formally argue how resilient the cipher is from differential and linear cryptanalysis. Not only does this mathematical structure help protect against differential cryptanalysis, but it also convinces users that it has not been engineered with some hidden trapdoor.

Each byte $s = [s_7, \ldots, s_0]$ of the Rijndael state matrix is taken in turn and considered as an element of $\mathbb{F}_{2^8}$. The S-Box can be mathematically described in two steps:

(1) The multiplicative inverse in $\mathbb{F}_{2^8}$ of s is computed to produce a new byte $x = [x_7, \ldots, x_0]$. For the element $[0, \ldots, 0]$ which has no multiplicative inverse one uses the convention that this is mapped to zero.
(2) The bit-vector x is then mapped, via the following affine $\mathbb{F}_2$ transformation, to the bit-vector y:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

The new byte is given by y. The decryption S-Box is obtained by first inverting the affine transformation and then taking the multiplicative inverse. These byte substitutions can either be implemented using table look-up or by implementing circuits, or code, which implement the inverse operation in $\mathbb{F}_{2^8}$ and the affine transformation.

3.1.2. *ShiftRows:* The ShiftRows operation in Rijndael performs a cyclic shift on the state matrix. Each row is shifted by different offsets. For the version of Rijndael we are considering this

is given by

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \longmapsto \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}.$$

The inverse of the ShiftRows operation is simply a similar shift but in the opposite direction. The ShiftRows operation ensures that the columns of the state matrix 'interact' with each other over a number of rounds.

3.1.3. *MixColumns:* The MixColumns operation ensures that the rows in the state matrix 'interact' with each other over a number of rounds; combined with the ShiftRows operation it ensures each byte of the output state depends on each byte of the input state.

We consider each column of the state in turn and consider it as a polynomial of degree less than four with coefficients in $\mathbb{F}_{2^8}$. The new column $[b_0, b_1, b_2, b_3]$ is produced by taking this polynomial

$$a(X) = a_0 + a_1 X + a_2 X^2 + a_3 X^3$$

and multiplying it by the polynomial

$$c(X) = 0x02 + 0x01 \cdot X + 0x01 \cdot X^2 + 0x03 \cdot X^3$$

modulo

$$M(X) = X^4 + 1.$$

This operation is conveniently represented by the following matrix operation in $\mathbb{F}_{2^8}$,

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}.$$

In $\mathbb{F}_{2^8}$ the above matrix is invertible, hence the inverse of the MixColumns operation can also be implemented using a matrix multiplication such as that above.

3.1.4. *AddRoundKey:* The round key addition is particularly simple. One takes the state matrix and XORs it, byte by byte, with the round key matrix. The inverse of this operation is clearly the same operation.

3.2. Round Structure. The Rijndael algorithm can now be described using the pseudo-code in Algorithm 8.1. The message block to encrypt is assumed to be entered into the state matrix S, the output encrypted block is also given by the state matrix S. Notice that the final round does not perform a MixColumns operation. The corresponding decryption operation is described in Algorithm 8.2.

Algorithm 8.1: Rijndael Encryption Outline

```
AddRoundKey(S, K_0)
for i = 1 to 9 do
    SubBytes(S)
    ShiftRows(S)
    MixColumns(S)
    AddRoundKey(S, K_i)
end
SubBytes(S)
ShiftRows(S)
AddRoundKey(S, K_10)
```

Algorithm 8.2: Rijndael Decryption Outline

AddRoundKey(S, K_{10})
InverseShiftRows(S)
InverseSubBytes(S)
for $i = 9$ **downto** 1 **do**
  AddRoundKey(S, K_i)
  InverseMixColumns(S)
  InverseShiftRows(S)
  InverseSubBytes(S)
end
AddRoundKey(S, K_0)

3.3. Key Schedule. The only thing left to describe is how Rijndael computes the round keys from the main key. Recall that the main key is 128 bits long, and we need to produce 11 round keys $K_0, \ldots, K_{11}$ all of which consist of four 32-bit words. Each word corresponding to a column of a matrix as described above. The key schedule makes use of a round constant which we shall denote by

$$RC_i = x^i \pmod{x^8 + x^4 + x^3 + x + 1}.$$

We label the round keys as $(W_{4i}, W_{4i+1}, W_{4i+2}, W_{4i+3})$ where i is the round. The initial main key is first divided into four 32-bit words (k_0, k_1, k_2, k_3). The round keys are then computed as in Algorithm 8.3, where RotBytes is the function which rotates a word to the left by a single byte, and SubBytes applies the Rijndael encryption S-Box to every byte in a word.

Algorithm 8.3: Rijndael Key Schedule

$W_0 = K_0, W_1 = K_1, W_2 = K_2, W_3 = K_3$
for $i = 1$ **to** 10 **do**
  $T =$RotBytes(W_{4i-1})
  $T =$SubBytes(T)
  $T = T \oplus RC_i$
  $W_{4i} = W_{4i-4} \oplus T$
  $W_{4i+1} = W_{4i-3} \oplus W_{4i}$
  $W_{4i+2} = W_{4i-2} \oplus W_{4i+1}$
  $W_{4i+3} = W_{4i-1} \oplus W_{4i+2}$
end

4. Modes of Operation

A block cipher like DES or Rijndael can be used in a variety of ways to encrypt a data string. Soon after DES was standardized another US Federal standard appeared giving four *recommended* ways of using DES for data encryption. These modes of operation have since been standardized internationally and can be used with any block cipher. The four modes are

- **ECB Mode:** This is simple to use, but suffers from possible deletion and insertion attacks. A one-bit error in ciphertext gives one whole block error in the decrypted plaintext.
- **CBC Mode:** This is the best mode to use as a block cipher since it helps protect against deletion and insertion attacks. In this mode a one-bit error in the ciphertext gives not only a one-block error in the corresponding plaintext block but also a one-bit error in the next decrypted plaintext block.

- **OFB Mode:** This mode turns a block cipher into a stream cipher. It has the property that a one-bit error in ciphertext gives a one-bit error in the decrypted plaintext.
- **CFB Mode:** This mode also turns a block cipher into a stream cipher. A single bit error in the ciphertext affects both this block and the next, just as in CBC mode.

Over the years various other modes of operation have been presented, probably the most popular of the more modern modes is

- **CTR Mode:** This also turns the block cipher into a stream cipher, but in enables blocks to be processed in parallel, thus providing performance advantages when parallel processing is available.

We shall now describe each of these five modes of operation in detail.

4.1. ECB Mode. Electronic Code Book Mode, or ECB Mode, is the simplest way to use a block cipher. The data to be encrypted m is divided into blocks of n bits:

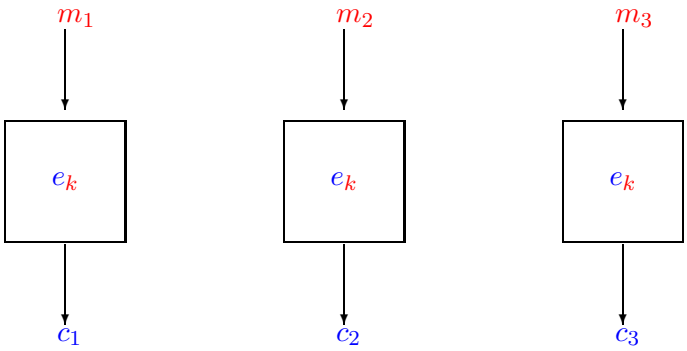
$$m_1, m_2, \ldots, m_q$$

with the last block padded if needed. The ciphertext blocks $c_1, \ldots, c_q$ are then defined as follows

$$c_i = e_k(m_i),$$

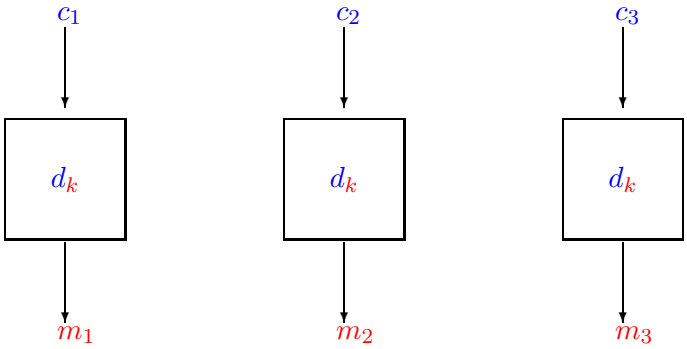
as described in Fig. 7. Decipherment is simply the reverse operation as explained in Fig. 8.

FIGURE 7. ECB encipherment



ECB Mode has a number of problems: the first is due to the property that if $m_i = m_j$ then we have $c_i = c_j$, i.e. the same input block always generates the same output block. This is a problem since stereotyped beginnings and ends of messages are common. The second problem comes because

FIGURE 8. ECB decipherment



we could simply delete blocks from the message and no one would know. Thirdly we could replay known blocks from other messages. By extracting ciphertext corresponding to a known piece of plaintext we can then amend other transactions to contain this known block of text.

To see all these problems suppose our block cipher is rather simple and encrypts each English word as a block. Suppose we obtained the encryption of the sentences

Pay Alice one hundred pounds,
Don't pay Bob two hundred pounds,

which encrypted were

the horse has four legs,
stop the pony hasn't four legs.

We can now make the recipient pay Alice two hundred pounds by sending her the message

the horse hasn't four legs,

in other words we have replaced a block from one message by a block from another message. Or we could stop the recipient paying Alice one hundred pounds by inserting the encryption stop of don't onto the front of the original message to Alice. Or we can make the recipient pay Bob two hundred pounds by deleting the first block of the message sent to him.

These threats can be countered by adding checksums over a number of plaintext blocks, or by using a mode of operation which adds some 'context' to each ciphertext block.

4.2. CBC Mode. One way of countering the problems with ECB Mode is to chain the cipher, and in this way add context to each ciphertext block. The easiest way of doing this is to use Cipher Block Chaining Mode, or CBC Mode.

Again, the plaintext must first be divided into a series of blocks

$$m_1, \ldots, m_q,$$

and as before the final block may need padding to make the plaintext length a multiple of the block length. Encryption is then performed via the equations

$$\begin{aligned} c_1 &= e_k(m_1 \oplus IV), \\ c_i &= e_k(m_i \oplus c_{i-1}) \text{ for } i > 1, \end{aligned}$$

see also Fig. 9.

Notice that we require an additional initial value IV to be passed to the encryption function, which can be used to make sure that two encryptions of the same plaintext produce different ciphertexts. In some situations one therefore uses a random IV with every message, usually when the same key will be used to encrypt a number of messages. In other situations, mainly when the key to the block cipher is only going to be used once, one chooses a fixed IV, for example the all zero string. In the case where a random IV is used, it is not necessary for the IV to be kept secret and it is usually transmitted in the clear from the encryptor to the decryptor as part of the message. The distinction between the reasons for using a fixed or random value for IV is expanded upon further in Chapter 21.

Decryption also requires the IV and is performed via the equations,

$$\begin{aligned} m_1 &= d_k(c_1) \oplus IV, \\ m_i &= d_k(c_i) \oplus c_{i-1} \text{ for } i > 1, \end{aligned}$$

see Fig. 10.

With ECB Mode a single bit error in transmission of the ciphertext will result in a whole block being decrypted wrongly, whilst in CBC Mode we see that not only will we decrypt a block incorrectly but the error will also affect a single bit of the next block.

FIGURE 9. CBC encipherment

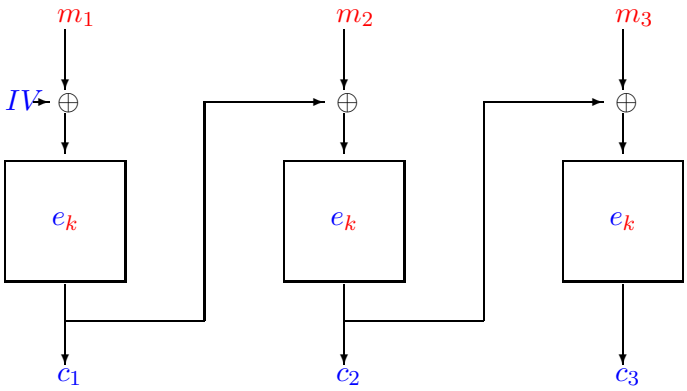
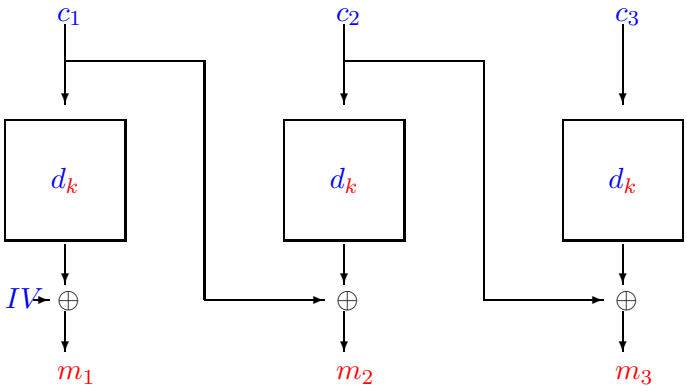


FIGURE 10. CBC decipherment



4.3. OFB Mode. Output Feedback Mode, or OFB Mode enables a block cipher to be used as a stream cipher. We need to choose a variable j $(1 \leq j \leq n)$ which will denote the number of bits output by the keystream generator on each iteration. We use the block cipher to create the keystream, j bits at a time. It is however usually recommended to take $j = n$ as that makes the expected cycle length of the keystream generator larger.

Again we divide plaintext into a series of blocks, but this time each block is j-bits, rather than n-bits long:

$$m_1, \ldots, m_q.$$

Encryption is performed as follows, see Fig. 11 for a graphical representation. First we set $X_1 = IV$, then for $i = 1, 2, \ldots, q$, we perform the following steps,

$$\begin{aligned} Y_i &= e_k(X_i), \\ E_i &= j \text{ leftmost bits of } Y_i, \\ c_i &= m_i \oplus E_i, \\ X_{i+1} &= Y_i. \end{aligned}$$

Decipherment in OFB Mode is performed in a similar manner as described in Fig. 12.

4.4. CFB Mode. The next mode we consider is called Cipher FeedBack Mode, or CFB Mode. This is very similar to OFB Mode in that we use the block cipher to produce a stream cipher. Recall

FIGURE 11. OFB encipherment

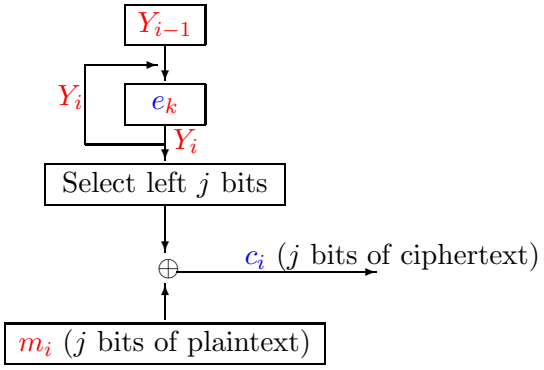
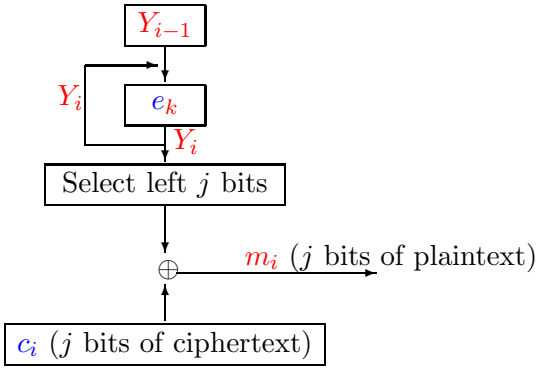


FIGURE 12. OFB decipherment



that in OFB Mode the keystream was generated by encrypting the IV and then iteratively encrypting the output from the previous encryption. In CFB Mode the keystream output is produced by the encryption of the ciphertext, as in Fig. 13, by the following steps,

$$\begin{aligned} Y_0 &= IV, \\ Z_i &= e_k(Y_{i-1}), \\ E_i &= j \text{ leftmost bits of } Z_i, \\ Y_i &= m_i \oplus E_i. \end{aligned}$$

We do not present the decryption steps, but leave these as an exercise for the reader.

4.5. CTR Mode. The next mode we consider is called Counter Mode, or CTR Mode. This combines many of the advantages of ECB Mode, but with none of the disadvantages. We first select a public IV, or counter, which is chosen differently for each message encrypted under the fixed key k. Then encryption proceeds for the ith block, by encrypting the value of $IV + i$ and then xor'ing this with the message block. In other words we have

$$c_i = m_i \oplus e_k(IV + i).$$

This is explained pictorially in Figure 14

CTR Mode has a number of interesting properties. Firstly since each block can be encrypted independently, much like in ECB Mode, we can process each block at the same time. Compare this to CBC Mode, OFB Mode or CFB Mode where we cannot start encrypting the second block until the first block has been encrypted. This means that encryption, and decryption, can be performed in parallel. However, unlike ECB Mode two equal blocks will not encrypt to the same ciphertext

FIGURE 13. CFB encipherment

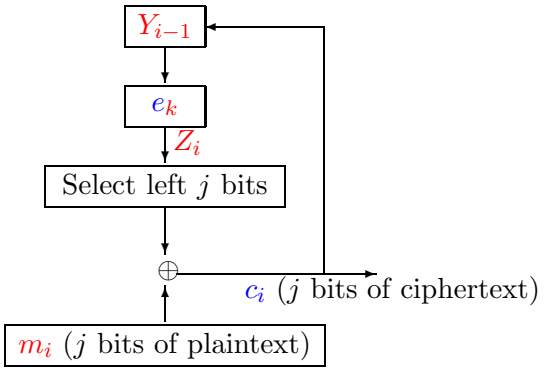
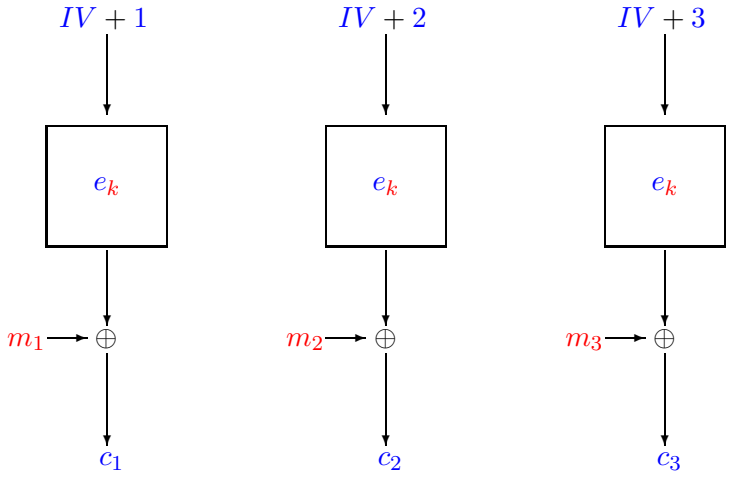


FIGURE 14. CTR encipherment



value. This is because each plaintext block is encrypted using a different input to the encryption function, in some sense we are using the block cipher encryption of the different inputs to produce a stream cipher. Also unlike ECB Mode each ciphertext block corresponds to a precise position within the ciphertext, as its position information is needed to be able to decrypt it successfully.

Chapter Summary

- The most popular block cipher is DES, which is itself based on a general design called a Feistel cipher.
- A comparatively recent block cipher is the AES cipher, called Rijndael.
- Both DES and Rijndael obtain their security by repeated application of simple rounds consisting of substitution, permutation and key addition.
- To use a block cipher one needs to also specify a mode of operation. The simplest mode is ECB mode, which has a number of problems associated with it. Hence, it is common to use a more advanced mode such as CBC or CTR mode.

- Some block cipher modes, such as CFB, OFB and CTR modes, allow the block cipher to be used in a stream cipher.

Further Reading

The Rijndael algorithm, the AES process and a detailed discussion of attacks on block ciphers and Rijndael in particular can be found in the book by Daemen and Rijmen. Stinson's book is the best book to explain differential cryptanalysis for students.

J. Daemen and V. Rijmen. *The Design of Rijndael: AES – The Advanced Encryption Standard.* Springer-Verlag, 2002.

D. Stinson. *Cryptography Theory and Practice.* CRC Press, 1995.

CHAPTER 7

Modern Stream Ciphers

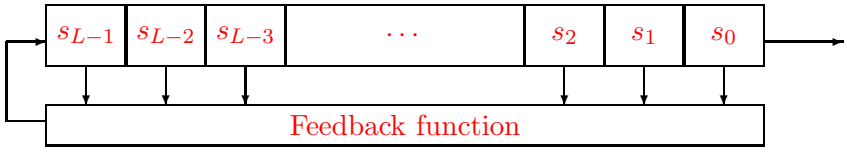
Chapter Goals

- To understand the basic principles of modern symmetric ciphers.
- To explain the basic workings of a modern stream cipher.
- To investigate the properties of linear feedback shift registers (LFSRs).

1. Linear Feedback Shift Registers

A standard way of producing a binary stream of data is to use a feedback shift register. These are small circuits containing a number of memory cells, each of which holds one bit of information. The set of such cells forms a register. In each cycle a certain predefined set of cells are 'tapped' and their value is passed through a function, called the *feedback function*. The register is then shifted down by one bit, with the output bit of the feedback shift register being the bit that is shifted out of the register. The combination of the tapped bits is then fed into the empty cell at the top of the register. This is explained in Fig. 1.

FIGURE 1. Feedback shift register



It is desirable, for reasons we shall see later, to use some form of non-linear function as the feedback function. However, this is often hard to do in practice hence usually one uses a linear feedback shift register, or LFSR for short, where the feedback function is a linear function of the tapped bits. In each cycle a certain predefined set of cells are 'tapped' and their value is XORed together. The register is then shifted down by one bit, with the output bit of the LFSR being the bit that is shifted out of the register. Again, the combination of the tapped bits is then fed into the empty cell at the top of the register.

Mathematically this can be defined as follows, where the register is assumed to be of length L. One defines a set of bits $[c_1, \ldots, c_L]$ which are set to one if that cell is tapped and set to zero otherwise. The initial internal state of the register is given by the bit sequence $[s_{L-1}, \ldots, s_1, s_0]$. The output sequence is then defined to be $s_0, s_1, s_2, \ldots, s_{L-1}, s_L, s_{L+1}, \ldots$ where for $j \geq L$ we have

$$s_j = c_1 \cdot s_{j-1} \oplus c_2 \cdot s_{j-2} \oplus \cdots \oplus c_L \cdot s_{j-L}.$$

Note, that for an initial state of all zeros the output sequence will be the zero sequence, but for a non-zero initial state the output sequence must be eventually periodic (since we must eventually

return to a state we have already been in). The period of a sequence is defined to be the smallest integer N such that

$$s_{N+i} = s_i$$

for all sufficiently large i. In fact there are $2^L - 1$ possible non-zero states and so the most one can hope for is that an LFSR, for all non-zero initial states, produces an output stream whose period is of length exactly $2^L - 1$.

Each state of the linear feedback shift register can be obtained from the previous state via a matrix multiplication. If we write

$$M = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ c_L & c_{L-1} & c_{L-2} & \dots & c_1 \end{pmatrix}$$

and

$$v = (1, 0, 0, \dots, 0)$$

and we write the internal state as

$$s = (s_1, s_2, \dots, s_L)$$

then the next state can be deduced by computing

$$s = M \cdot s$$

and the output bit can be produced by computing the vector product

$$v \cdot s.$$

The properties of the output sequence are closely tied up with the properties of the binary polynomial

$$C(X) = 1 + c_1 X + c_2 X^2 + \cdots + c_L X^L \in \mathbb{F}_2[X],$$

called the connection polynomial for the LFSR. The connection polynomial and the matrix are related via

$$C(X) = \det(XM - I_L).$$

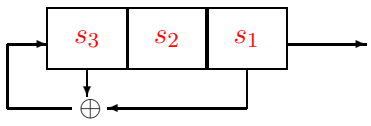
In some text books the connection polynomial is written in reverse, i.e. they use

$$G(X) = X^L C(1/X)$$

as the connection polynomial. One should note that in this case $G(X)$ is the characteristic polynomial of the matrix M.

As an example see Fig. 2 for an LFSR in which the connection polynomial is given by X^3+X+1 and Fig. 3 for an LFSR in which the connection polynomial is given by $X^{32} + X^3 + 1$.

FIGURE 2. Linear feedback shift register: $X^3 + X + 1$



Of particular importance is when the connection polynomial is primitive.

FIGURE 3. Linear feedback shift register: $X^{32} + X^3 + 1$

DEFINITION 7.1. *A binary polynomial $C(X)$ of degree L is primitive if it is irreducible and a root θ of $C(X)$ generates the multiplicative group of the field $\mathbb{F}_{2^L}$. In other words, since $C(X)$ is irreducible we already have*

$$\mathbb{F}_2[X]/(C(X)) = \mathbb{F}_2(\theta) = \mathbb{F}_{2^L},$$

but we also require

$$\mathbb{F}_{2^L}^* = \langle\theta\rangle.$$

The properties of the output sequence of the LFSR can then be deduced from the following cases.

- $c_L = 0$:
  In this case the sequence is said to be singular. The output sequence may not be periodic, but it will be eventually periodic.
- $c_L = 1$:
  Such a sequence is called non-singular. The output is always purely periodic, in that it satisfies $s_{N+i} = s_i$ for all i rather than for all sufficiently large values of i. Of the non-singular sequences of particular interest are those satisfying
  - $C(X)$ is irreducible:
    Every non-zero initial state will produce a sequence with period equal to the smallest value of N such that $C(X)$ divides $1 + X^N$. We have that N will divide $2^L - 1$.
  - $C(X)$ is primitive:
    Every non-zero initial state produces an output sequence which is periodic and of exact period $2^L - 1$.

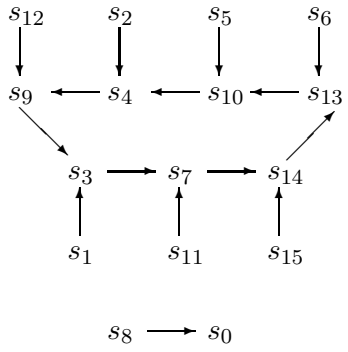
We do not prove these results here, but proofs can be found in any good textbook on the application of finite fields to coding theory, cryptography or communications science. However, we present four examples which show the different behaviours. All examples are on four bit registers, i.e. $L = 4$.

Example 1 : In this example we use an LFSR with connection polynomial $C(X) = X^3 + X + 1$. We therefore see that $\deg(C) \neq L$, and so the sequence will be singular. The matrix M generating the sequence is given by

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

If we label the states of the LFSR by the number whose binary representation is the state value, i.e. $s_0 = (0, 0, 0, 0)$ and $s_5 = (0, 1, 0, 1)$, then the periods of this LFSR can be represented by the transitions in Figure 4. Note, it is not purely periodic.

FIGURE 4. Transitions of the four bit LFSR with connection polynomial $X^3 + X + 1$

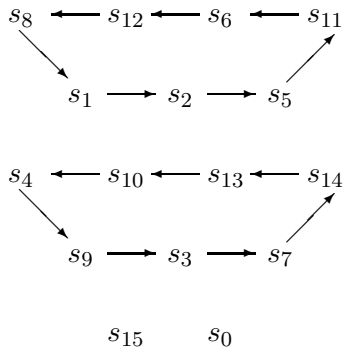


Example 2 : Now let the connection polynomial $C(X) = X^4+X^3+X^2+1 = (X+1)(X^3+X+1)$, which corresponds to the matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

The state transitions are then given by Figure 5. Note, it is purely periodic, but that there are different period lengths due to the different factorization properties of the connection polynomial modulo 2. One of length $7 = 2^3 - 1$ corresponding to the factor of degree three, and one of length $1 = 2^1 - 1$ corresponding to the factor of degree one. We ignore the trivial period of the zero'th state.

FIGURE 5. Transitions of the four bit LFSR with connection polynomial $X^4+X^3+X^2+1$

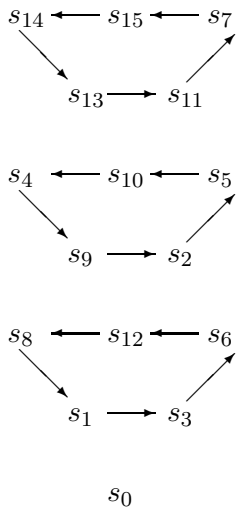


Example 3 : Now take the connection polynomial $C(X) = X^4 + X^3 + X^2 + X + 1$, which is irreducible, but not primitive. The matrix is now given by

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

The state transitions are then given by Figure 6. Note, it is purely periodic and all periods have same length, bar the trivial one.

FIGURE 6. Transitions of the four bit LFSR with connection polynomial $X^4+X^3+X^2+X+1$

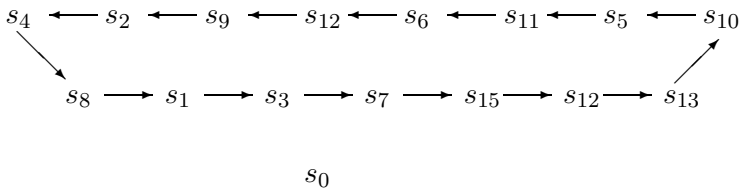


Example 4 : As our final example we take the connection polynomial $C(X) = X^4 + X + 1$, which is irreducible and primitive. The matrix M is now

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

and the state transitions are given by Figure 7.

FIGURE 7. Transitions of the four bit LFSR with connection polynomial X^4+X+1



Whilst there are algorithms to generate primitive polynomials for use in applications we shall not describe them here. We give some samples in the following list, where we give polynomials with a small number of taps for efficiency.

$$\begin{array}{ccc} x^{31}+x^3+1 & x^{31}+x^6+1 & x^{31}+x^7+1 \\ x^{39}+x^4+1 & x^{60}+x+1 & x^{63}+x+1 \\ x^{71}+x^6+1 & x^{93}+x^2+1 & x^{137}+x^{21}+1 \\ x^{145}+x^{52}+1 & x^{161}+x^{18}+1 & x^{521}+x^{32}+1 \end{array}$$

Although LFSRs efficiently produce bitstreams from a small key, especially when implemented in hardware, they are not usable on their own for cryptographic purposes. This is because they are essentially linear, which is after all why they are efficient.

We shall now show that if we know an LFSR has L internal registers, and we can determine $2L$ consecutive bits of the stream then we can determine the whole stream. First notice we need to determine L unknowns, the L values of the 'taps' c_i, since the L values of the initial state $s_0, \ldots, s_{L-1}$ are given to us. This type of data could be available in a known plaintext attack, where we obtain the ciphertext corresponding to a known piece of plaintext, since the encryption operation is simply exclusive-or we can determine as many bits of the keystream as we require.

Using the equation

$$s_j = \sum_{i=1}^{L} c_i \cdot s_{j-i} \pmod 2,$$

we obtain $2L$ linear equations, which we then solve via matrix techniques. We write our matrix equation as

$$\begin{pmatrix} s_{L-1} & s_{L-2} & \cdots & s_1 & s_0 \\ s_L & s_{L-1} & \cdots & s_2 & s_1 \\ \vdots & \vdots & & \vdots & \vdots \\ s_{2L-3} & s_{2L-4} & \cdots & s_{L-1} & s_{L-2} \\ s_{2L-2} & s_{2L-3} & \cdots & s_L & s_{L-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{L-1} \\ c_L \end{pmatrix} = \begin{pmatrix} s_L \\ s_{L+1} \\ \vdots \\ s_{2L-2} \\ s_{2L-1} \end{pmatrix}.$$

As an example, suppose we see the output sequence

$$1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, \ldots$$

and we are told that this sequence was the output of a four-bit LFSR. Using the above matrix equation, and solving it modulo 2, we would find that the connection polynomial was given by

$$X^4 + X + 1.$$

Hence, we can conclude that a stream cipher based solely on a single LFSR is insecure against a known plaintext attack.

An important measure of the cryptographic quality of a sequence is given by the linear complexity of the sequence.

DEFINITION 7.2 (Linear complexity). *For an infinite binary sequence*

$$s = s_0, s_1, s_2, s_3, \ldots,$$

we define the linear complexity of s as $L(s)$ where

- *$L(s) = 0$ if s is the zero sequence,*
- *$L(s) = \infty$ if no LFSR generates s,*
- *$L(s)$ will be the length of the shortest LFSR to generate s.*

Since we cannot compute the linear complexity of an infinite set of bits we often restrict ourselves to a finite set s^n of the first n bits. The linear complexity satisfies the following properties for any sequence s.

- For all $n \geq 1$ we have $0 \leq L(s^n) \leq n$.
- If s is periodic with period N then $L(s) \leq N$.
- $L(s \oplus t) \leq L(s) + L(t)$.

For a random sequence of bits, which is what we want from a stream cipher's keystream generator, we should have that the expected linear complexity of s^n is approximately just larger than $n/2$. But for a keystream generated by an LFSR we know that we will have $L(s^n) = L$ for all $n \geq L$. Hence, an LFSR produces nothing at all like a random bit string.

We have seen that if we know the length of the LFSR then, from the output bits, we can generate the connection polynomial. To determine the length we use the linear complexity profile, this is defined to be the sequence $L(s^1), L(s^2), L(s^3), \ldots$. There is also an efficient algorithm called the Berlekamp–Massey algorithm which given a finite sequence s^n will compute the linear complexity profile

$$L(s^1), L(s^2), L(s^3), \ldots, L(s^n).$$

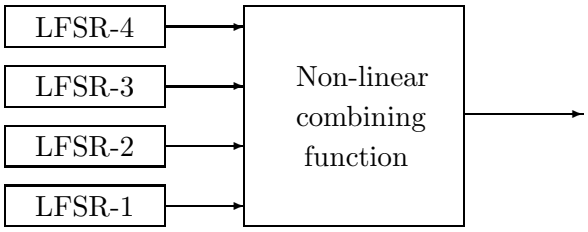
In addition the Berlekamp–Massey algorithm will also output the associated connection polynomial, if $n \geq L(s^n)/2$, using a technique more efficient than the prior matrix technique.

Hence, if we use an LFSR of size L to generate a keystream for a stream cipher and the adversary obtains at least $2L$ bits of this keystream then they can determine the exact LFSR used and so generate as much of the keystream as they wish. Therefore, one needs to find a way of using LFSRs in some non-linear way, which hides the linearity of the LFSRs and produces output sequences with high linear complexity.

2. Combining LFSRs

To use LFSRs in practice it is common for a number of them to be used, producing a set of output sequences $x_1^{(i)}, \ldots, x_n^{(i)}$. The key is then the initial state of all of the LFSRs and the keystream is produced from these n generators using a non-linear combination function $f(x_1, \ldots, x_n)$, as described in Fig. 8.

FIGURE 8. Combining LFSRs



We begin by examining the case where the combination function is a boolean function of the output bits of the constituent LFSRs. For analysis of this function we write it as a sum of distinct products of variables, e.g.

$$f(x_1, x_2, x_3, x_4, x_5) = 1 \oplus x_2 \oplus x_3 \oplus x_4 \cdot x_5 \oplus x_1 \cdot x_2 \cdot x_3 \cdot x_5.$$

However, in practice the boolean function could be implemented in a different way. When expressed as a sum of products of variables we say that the boolean function is in algebraic normal form.

Suppose that one uses n LFSRs of maximal length (i.e. all with a primitive connection polynomial) and whose periods $L_1, \ldots, L_n$ are all distinct and greater than two. Then the linear complexity of the keystream generated by $f(x_1, \ldots, x_n)$ is equal to

$$f(L_1, \ldots, L_n)$$

where we replace $\oplus$ in f with integer addition and multiplication modulo two by integer multiplication, assuming f is expressed in algebraic normal form. The non-linear order of the polynomial f is then equal to total the degree of f.

However, it turns out that creating a nonlinear function which results in a high linear complexity is not the whole story. For example consider the stream cipher produced by the Geffe generator.

This generator takes three LFSRs of maximal period and distinct sizes, L_1, L_2 and L_3, and then combines them using the non-linear function, of non-linear order 2,

$$z = f(x_1, x_2, x_3) = x_1 \cdot x_2 \oplus x_2 \cdot x_3 \oplus x_3. \tag{12}$$

This would appear to have very nice properties: It's linear complexity is given by

$$L_1 \cdot L_2 + L_2 \cdot L_3 + L_3$$

and it's period is given by

$$(2^{L_1} - 1)(2^{L_2} - 1)(2^{L_3} - 1).$$

However, it turns out to be cryptographically weak.

To understand the weakness of the Geffe generator consider the following table, which presents the outputs x_i of the constituent LFSRs and the resulting output z of the Geffe generator

x_1	x_2	x_3	z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

If the Geffe generator was using a "good" non-linear combining function then the output bits z would not reveal any information about the corresponding output bits of the constituent LFSRs. However, we can easily see that

$$\Pr(z = x_1) = 3/4 \text{ and } \Pr(z = x_3) = 3/4.$$

This means that the output bits of the Geffe generator are correlated with the bits of two of the constituent LFSRs. This means that we can attack the generator using a correlation attack.

This attack proceeds as follows, suppose we know the lengths L_i of the constituent generators, but not the connection polynomials or their initial states. The attack is desribed in Algorithm 7.1

Algorithm 7.1: Correlation attack on the Geffe generator

forall the *primitive connection polynomials of degree* L_1 **do**
  forall the *Initial states of the first LFSR* **do**
    Compute $2L_1$ bits of output of the first LFSR
    Compute how many are equal to the output of the Geffe generator
    A large value signals that this is the correct choice of generator and starting state.
  end
end
Repeat the above for the third LFSR
Recover the second LFSR by testing possible values using (12)

It turns out that their are a total of

$$S = \phi(2^{L_1} - 1) \cdot \phi(2^{L_2} - 1) \cdot \phi(2^{L_3} - 1)/(L_1 \cdot L_2 \cdot L_3)$$

possible connection polynomials for the three LFSRs in the Geffe generator. The total number of initial states of the Geffe generator is

$$T = (2^{L_1} - 1)(2^{L_2} - 1)(2^{L_3} - 1) \approx 2^{L_1 + L_2 + L_3}.$$

This means that the key size of the Geffe generator is

$$S \cdot T \approx S \cdot (2^{L_1+L_2+L_3}).$$

For a secure stream cipher we would like the size of the key space to be about the same as the number of operations needed to break the stream cipher. However, the above correlation attack on the Geffe generator requires roughly

$$S \cdot (2^{L_1} + 2^{L_2} + 2^{L_3})$$

operations. The reason for the reduced complexity is that we can deal with each constituent LFSR in turn.

To combine high linear complexity and resistance to correlation attacks (and other attacks) designers have had to be a little more ingenious as to how they have produced non-linear combiners for LFSRs. We now outline a small subset of some of the most influential:

Filter Generator: The basic idea here is to take a single primitive LFSR with internal state $s_1, \ldots, s_L$ and then make the output of the stream cipher be a non-linear function of the whole state, i.e.

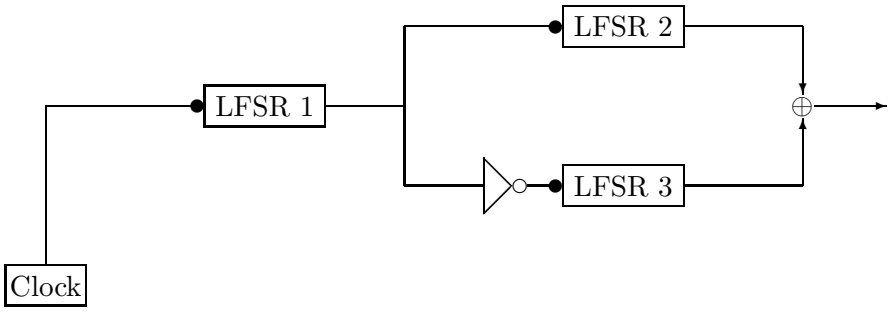
$$z = F(s_1, \ldots, s_L).$$

If F has non-linear order m then the linear complexity of the resulting sequence is given by

$$\sum_{i=1}^{m} \left(\begin{array}{c} L \\ i \end{array} \right).$$

Alternating Step Generator: This takes three LFSRs of size L_1, L_2 and L_3 which are pairwise coprime, and of roughly the same size. If the output sequence of the three LFSRs is denoted by x_1, x_2 and x_3, then one proceeds as follows: The first LFSR is clocked on every iteration. If its output x_1 is equal to one, then the second LFSR is clocked and the output of the third LFSR is repeated from its last value. If the output of x_1 is equal to zero, then the third LFSR is clocked and the output of the second LFSR is repeated from its last value. The output of the generator is the value of $x_2 \oplus x_3$. This operation is described graphically in Figure 9.

FIGURE 9. Graphical representation of the Alternating step generator



The alternating step generator has period

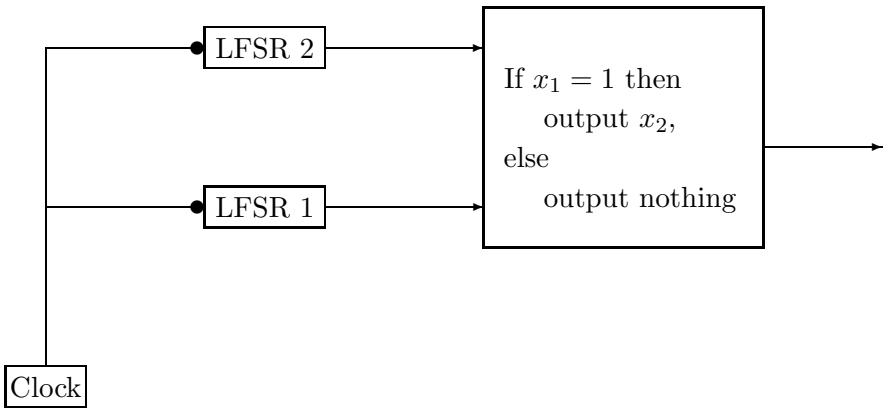
$$2^{L_1}(2^{L_2} - 1)(2^{L_3} - 1)$$

and linear complexity, approximately

$$(L_2 + L_3) \cdot 2^{L_1}.$$

Shrinking Generator: Here we take two LFSRs with output sequence x_1 and x_2, and the idea is to throw away some of the x_2 stream under the control of the x_1 stream. Both LFSRs are clocked at the same time, and if x_1 is equal to one then the output of the generator is the value of x_2. If x_1 is equal to zero then the generator just clocks again. Note, that this means that the generator does not produce a bit on each iteration. This operation is described graphically in Figure 10.

FIGURE 10. Graphical representation of the Shrinking Generator



If we assume that the two constituent LFSRs have size L_1 and L_2 with $\gcd(L_1, L_2)$ equal to one, then the period of the shrinking generator is equal to

$$(2^{L_2} - 1) \cdot 2^{L_1 - 1}$$

and its linear complexity is approximately

$$L_2 \cdot 2^{L_1}.$$

The A5/1 Generator: Probably the most famous of the recent LFSR based stream ciphers is A5/1. This is the stream cipher used to encrypt the on-air traffic in the GSM mobile phone networks in Europe and the US. This was developed in 1987, buts its design was kept secret until 1999 when it was reverse engineered. There is a weakened version of the algorithm called A5/2 which was designed for use in places where there were various export restrictions. In recent years various attacks have been published on A5/1 which has resulted in it no longer being considered a secure cipher. In the replacement for GSM, i.e. UMTS or 3G networks, the cipher has been replaced with the use of the block cipher KASUMI in a stream cipher mode of operation.

A5/1 makes use of three LFSRs of length 19, 22 and 23. These have characteristic polynomials

$$x^{18} + x^{17} + x^{16} + x^{13} + 1,$$
$$x^{21} + x^{20} + 1,$$
$$x^{22} + x^{21} + x^{20} + x^7 + 1.$$

Alternatively (and equivalently) their connection polynomials are given by

$$x^{18} + x^5 + x^2 + x^1 + 1,$$
$$x^{21} + x^1 + 1,$$
$$x^{22} + x^{15} + x^2 + x^1 + 1.$$

The output of the cipher is the exclusive-or of the three output bits of the three LFSRs.

To clock the registers we associate to each register a "clocking bit". These are in positions 10, 11 and 12 of the LFSR's (assuming bits are ordered with 0 corresponding to the output bit, other books may use a different ordering). We will call these bits c_1, c_2 and c_3. At each clock step the three bits are computed and the "majority bit" is determined via the formulae

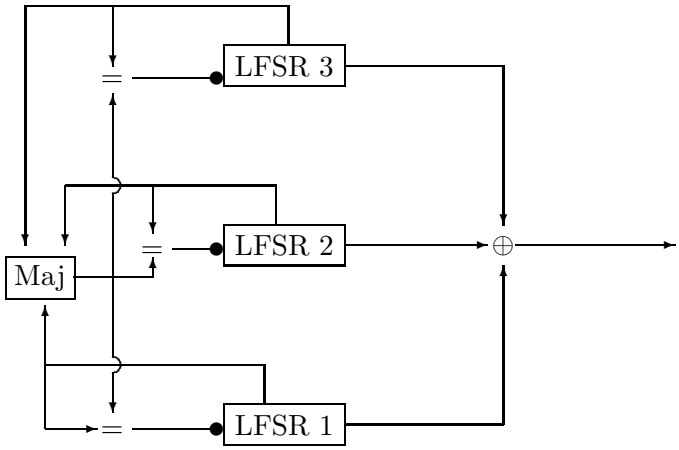
$$c_1 \cdot c_2 \oplus c_2 \cdot c_3 \oplus c_1 \cdot c_3.$$

The ith LFSR is then clocked if the majority bit is equal to the bit c_i. Thus clocking occurs subject to the following table

			Majority	Clock LFSR		
c_1	c_2	c_3	Bit	1	2	3
0	0	0	0	Y	Y	Y
0	0	1	0	Y	Y	N
0	1	0	0	Y	N	Y
0	1	1	1	N	Y	Y
1	0	0	0	N	Y	Y
1	0	1	1	Y	N	Y
1	1	0	1	Y	Y	N
1	1	1	1	Y	Y	Y

Thus we see in A5/1 that each LFSR is clocked with probability 3/4. This operation is described graphically in Figure 11.

FIGURE 11. Graphical representation of the A5/1 Generator



3. RC4

RC stands for Ron's Cipher after Ron Rivest of MIT. You should not think that the RC4 cipher is a prior version of the block ciphers RC5 and RC6. It is in fact a very, very fast stream cipher. It is very easy to remember since it is surprisingly simple.

Given an array S indexed from 0 to 255 consisting of the integers $0, \ldots, 255$, permuted in some key-dependent way, the output of the RC4 algorithm is a keystream of bytes K which is XORed with the plaintext byte by byte. Since the algorithm works on bytes and not bits, and uses very simple operations it is particularly fast in software. We start by letting $i = 0$ and $j = 0$, we then repeat the steps in Algorithm 7.2.

The security rests on the observation that even if the attacker knows K and i, he can deduce the value of S_t, but this does not allow him to deduce anything about the internal state of the

Algorithm 7.2: RC4 Algorithm

$i = (i + 1) \mod 256$
$j = (j + S_i) \mod 256$
$\text{swap}(S_i, S_j)$
$t = (S_i + S_j) \mod 256$
$K = S_t$

table. This follows from the observation that he cannot deduce the value of t, as he does not know j, S_i or S_j.

It is a very tightly designed algorithm as each line of the code needs to be there to make the cipher secure.

- $i = (i + 1) \mod 256$ **:**
  Makes sure every array element is used once after 256 iterations.
- $j = (j + S_i) \mod 256$ **:**
  Makes the output depend non-linearly on the array.
- **swap**(S_i, S_j) **:**
  Makes sure the array is evolved and modified as the iteration continues.
- $t = (S_i + S_j) \mod 256$ **:**
  Makes sure the output sequence reveals little about the internal state of the array.

The initial state of the array S is determined from the key using the method described by Algorithm 7.3.

Algorithm 7.3: RC4 Key Schedule

for $i = 0$ **to** 255 **do** $S_i = i$
Initialise K_i, for $i = 0, \ldots, 255$, with the key, repeating if neccesary
$j = 0$
for $i = 0$ **to** 255 **do**
  $j = (j + S_i + K_i) \mod 256$
  $\text{swap}(S_i, S_j)$
end

Although RC4 is very fast for a software based stream cipher, there are some issues with its use. In particular both the key schedule and the main algorithm do not produce as random a stream as one might wish. Hence, it should be used with care.

Chapter Summary

- Modern stream ciphers can be obtained by combining, in a non-linear way, simple bit generators called LFSRs, these stream ciphers are bit oriented.
- LFSR based stream ciphers provide very fast ciphers, suitable for implementation in hardware, which can encrypt real-time data such as voice or video.
- RC4 provides a fast and compact byte oriented stream cipher for use in software.

Further Reading

A good introduction to linear recurrence sequences over finite fields is in the book by Lidl and Neiderreiter. This book covers all the theory one requires, including examples and a description of the Berlekamp–Massey algorithm. Analysis of the RC4 algorithm can be found in the Master's thesis of Mantin.

R. Lidl and H. Neiderreiter. *Introduction to fintie field and their applications.* Cambridge University Press, 1986.

I. Mantin. *Analysis of the Stream Cipher RC4.* MSc. Thesis, Weizmann Institute of Science, 2001.