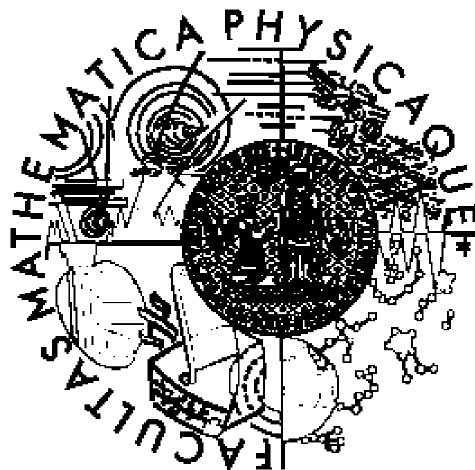
Matematicko-fyzikální fakulta
Univerzita Karlova v Praze

BAKALÁŘSKÁ PRÁCE



Jan Říha

Konstrukce a kryptoanalýza AES (Advanced Encryption Standard)

Katedra algebry

Vedoucí bakalářské práce: Doc. RNDr. Jiří Tůma, DrSc.

Studijní program: Matematika,
Obecná matematika,
Matematické metody informační bezpečnosti

2006

Na tomto místě bych chtěl poděkovat především vedoucímu mé bakalářské práce, Doc. RNDr. Jiřímu Tůmovi, DrSc., za zadání tohoto tématu a za mnoho přínosných rad, kterými přispěl k tvorbě tohoto textu. Dále bych rád poděkoval matematickému oddělení Knihovny MFF UK v Karlíně za zapůjčení stěžejní literatury a také celé Matematicko-fyzikální fakultě za poskytnuté zázemí k tvorbě této práce. V neposlední řadě bych rád poděkoval všem provozovatelům internetových stránek uvedených ke konci tohoto textu za poskytnutí cenných informací.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne Jan Říha

Obsah

1 Vznik AES **5**
- 1.1 Vypsání soutěže 5
- 1.2 První kolo 6
- 1.3 Hodnotící kritéria 6
- 1.4 Druhé kolo 7

2 Konstrukce AES **8**
- 2.1 Rozdíl mezi Rijndaelem a AES 8
- 2.2 Matematický úvod 8
- 2.3 Vstup a výstup algoritmu AES 11
- 2.4 Struktura šifry. 12
- 2.5 Transformace SubBytes() 12
- 2.6 Transformace ShiftRows() 14
- 2.7 Transformace MixColumns(). 14
- 2.8 Transformace AddRoundKey(). 15
- 2.9 Prodloužení klíče. 16
- 2.10 Přímá inverzní šifra 17
- 2.11 Ekvivalentní inverzní šifra 19
- 2.12 Zvláštní poděkování 20

3 Implementace AES **21**
- 3.1 8-bitové platformy 21
- 3.2 32-bitové platformy 22
- 3.3 Hardware určený pro šifrování 24
- 3.4 Multiprocesorové platformy 25

4 Kryptoanalýza AES **26**
- 4.1 Zkrácené rozdíly 26
- 4.2 Saturační útoky 26
- 4.3 Gilbert-Minierův útok 27
- 4.4 Interpolační útoky 27
- 4.5 Symetrické vlastnosti a slabé klíče jako u šifry DES 27
- 4.6 Slabé klíče jako u šifry IDEA 28
- 4.7 Related-Key útoky 28
- 4.8 Implementační útoky 28

Literatura **31**

Název práce: Konstrukce a kryptoanalýza AES
(Advanced Encyption Standard)
Autor: Jan Říha
Katedra: Katedra Algebry
Vedoucí bakalářské práce: Doc. RNDr. Jiří Tůma, DrSc.
E-mail vedoucího bakalářské práce: Jiri.Tuma@mff.cuni.cz

Abstrakt: V předložené práci studujeme nejnovější symetrickou blokovou šifru AES. Nejprve se zabýváme vývojem a vznikem šifry od vypsání soutěže až po vyhlášení vítězného kandidáta. Poté se věnujeme její konstrukci, ve které se využívá některých netriviálních poznatků algebry při práci s polynomy nad konečným tělesem. V této kapitole je též popsána přímá inverzní šifra a ekvivalentní inverzní šifra sloužící k dešifrování zašifrovaných dat. Ve třetí kapitole zkoumáme navrhované implementace šifry AES na jednotlivé platformy a nakonec rozebíráme možné útoky a odolnost šifry AES vůči nim.

Klíčová slova: AES, šifra, implementace, kryptoanalýza

Title: The design and cryptanalysis of the AES
(Advanced Encyption Standard)
Autor: Jan Říha
Department: Department of Algebra
Supervisor: Doc. RNDr. Jiří Tůma, DrSc.
Supervisor's e-mail address: Jiri.Tuma@mff.cuni.cz

Abstract: In the present work we study the newest symetric block cipher AES. At first we consider development and creation of the cipher from the start of selection proces till announcement of winning candidate. Then we turn to its design, in which we use some non-trivial algebraic knowledge at work with polynomials with coefficients in finite field. In this chapter there is also described straightforward inverse cipher and equivalent inverse cipher make for decryption of encrypted dates. In chapter three we investigate proposed implementations of the cipher AES on individual platforms and in the end we analyse posible attacks and how the cipher AES is resistant against them.

Key words: AES, cipher, implementation, cryptoanalysis

Kapitola 1

Vznik AES

1.1 Vypsání soutěže

V lednu roku 1997 americký National Institute of Standards and Technology (NIST) ohlásil počátek vývoje nového šifrovacího standardu Advanced Encryption Standard (AES), který by nahradil starší, již nedostatečně bezpečný Data Encryption Standard (DES) a triple-DES.

Na rozdíl od výběrového řízení na DES, hašovací funkci SHA-1 nebo na algoritmus pro digitální podpis DSA pojal NIST výběrové řízení na AES jako otevřenou soutěž. Tedy kdokoliv mohl předložit svého kandidáta na šifru AES. NIST se rozhodl nehodnotit bezpečnost a efektivitu předkládaných kandidátů sám, nýbrž přizval k projektu kryptologickou komunitu a každého, kdo se o tento obor zajímá, aby zkusili provést množství útoků a ohodnotit cenu implementací jednotlivých šifer. Všechny výsledky a posudky mohli být posílány na webové stránky NISTu ve formě veřejných komentářů.

V září roku 1997 byly publikovány konečné požadavky na EAS. Minimální požadavky pro symetrickou blokovou šifru byly: podporovaná délka bloku 128 bitů a podporované délky klíče 128, 192 a 256 bitů. NIST dále prohlašoval, že hledá symetrickou blokovou šifru bezpečnou alespoň jako triple-DES, ale efektivnější a požadoval též, aby soutěžící souhlasil s tím, že jeho šifra bude volně dostupná, pokud bude vybrána jako AES. Dále soutěžící museli poskytnout:

1. Kompletní specifikaci blokové šifry popsanou ve formě algoritmu.
2. Referenční implementaci v ANSI C, a matematicky optimalizované implementace v ANSI C a Javě.
3. Implementace sérií known-answer a Monte Carlo testů, jakožto i očekávané výstupy těchto testů pro korektní implementaci své blokové šifry.
4. Prohlášení zahrnující očekávanou hardwarovou i softwarovou efektivitu, předpokládanou odolnost vůči kryptoanalytickým útokům a výhody a omezení šifry v různých aplikacích.
5. Analýzu odolnosti šifry proti známým kryptoanalytickým útokům.

Dané podmínky splnilo těchto 15 soutěžících:

Šifra	Soutěžící
CAST-256	Entrust (CA)
Crypton	Future Systems (KR)
DEAL	Outerbridge, Knudsen (USA-DK)
DFC	ENS-CNRS (FR)
E2	NTT (JP)
Frog	TecApro (CR)
HPC	Schroeppel (USA)
LOKI197	Brown et al. (AU)
Magenta	Deutsche Telekom (DE)
Mars	IBM (USA)
RC6	RSA (USA)
Rijndael	Daemen and Rijmen (BE)
SAFER+	Cylink (USA)
Serpent	Anderson, Biham, Knudsen (UK-IL-DK)
Twofish	Counterpane (USA)

Obrázek 1.1: Soutěžící, kteří prošli do prvního hodnotícího kola

1.2 První kolo

Všech 15 soutěžících se svými šiframi (Obr. 1.1), kteří uspěli v přijímacím období končícím 15. května 1998, bylo představeno na konferenci konající se v Californii ve městě Ventura ve dnech 20.-22. srpna 1998. Tímto bylo oficiálně zahájeno první hodnotící kolo, ve kterém byla mezinárodní kryptografická komunita požádána, aby se vyjadřovala k jednotlivým kandidátům na AES.

1.3 Hodnotící kritéria

Hodnotící kritéria pro první kolo byla rozdělena do tří hlavních kategorií:

Bezpečnost: Bezpečnost byla nejdůležitější, avšak asi nejobtížněji ohodnotitelnou kategorií. Jen pár kandidátů vykazovalo některé teoretické konstrukční vady. Většina spadala do skupiny ‘no weakness demostrated’, nebo-li u těchto šifer nebyla prokázána žádná slabina.

Cena: Cena kandidáta byla ještě rozdělena do několika podkategorií. První z nich byla tvořena cenou spojenou s náklady na vývoj šifry a oceněním vítězného tvůrce, který se předem musel vzdát všech vlastnických práv a nároků. Druhou podkategorii tvořila cena spojená s implementací a zavedením dané šifry.

Algoritmus a implementační charakteristiky: Také zde bylo několik podkategorií

Všestrannost: Hodnotilo se, jak efektivně lze šifru implementovat na různé platformy. Na jedné straně bylo třeba, aby šifra fungovala na 8-bitových mikroprocesorech a smart kartách, na straně druhé byla požadována efektivní implementace na hardware zaměřený přímo na šifrování a dešifrování, kde komunikace probíhá řádově v gigabitech za sekundu. Mezitím je samozřejmě široká škála procesorů, které jsou používány v serverech, domácích počítačích, laptopech, palmtopech, atd. Prominentní roli v této oblasti hrály procesory společnosti Pentium, díky přítomnosti ve většině PC.

Hbitost klíče: Pro blokové šifry zabere určitý výpočetní čas nastavení klíče. V aplikacích, kde je používán ten samý klíč pro šifrování velkého množství dat, je tato doba relativně nepodstatná. Ale některé aplikace vyžadují častou změnu klíče, například šifrování Internet Protocol (IP) packetů v Internet Protocol Security (IPSEC). V těchto aplikacích hraje nastavení klíče podstatnou roli a je tedy velkou výhodou, pokud toto nastavení proběhne rychle.

Jednoduchost: NIST považoval jednoduchost za výhodu hlavně z důvodu snadné implementace a také kvůli důvěře v bezpečnost šifry.

1.4 Druhé kolo

V srpnu roku 1999 NIST vyhlásil 5 finalistů, kteří postoupili do druhého kola. Byly to: MARS, RC6, Rijndael, Serpent a Twofish. Ostatní kandidáti buď nesplnili zadaná kritéria, nebo se v mnoha charakteristikách shodovali s pěti finalisty, ale vždy byli po určité stránce horší. Například byli srovnatelní s jedním z finalistů, ale měli menší bezpečnost, vyšší cenu implementace, nebo byli pomalejší.

Na konferenci v New York City, konané v dubnu roku 2000, byly shledány určité problémy v otázkách implementací šifer MARS a RC6. Naopak veřejným favoritem se stala šifra Rijndael.

2. října roku 2000 NIST oficiálně vyhlásil, že Rijndael bez modifikací zvítězil ve výběrovém řízení na AES. Svůj výběr odůvodnil velmi dobrou výkonností šifry v oblasti hardwaru i softwaru, širokým spektrem možného výpočetního prostředí, excelentním nastavováním klíče a dobrou hbitostí klíče. Dále měl Rijndael velmi nízké nároky na paměť a vykazoval výborné výkony v prostředích omezených na malý prostor. Také úroveň bezpečnosti byla velmi dobrá.

Kapitola 2

Konstrukce AES

2.1 Rozdíl mezi Rijndaelem a AES

Jediným rozdílem mezi šifrou Rijndael a AES je spektrum podporovaných délek bloků a klíčů.

Rijndael je blokovou šifrou s proměnnou délkou bloku i klíče. Délka bloku i klíče může být libovolným násobkem 32 bitů v rozmezí 128 bitů až 256 bitů.

AES má pevnou délku bloku rovnou 128 bitům a podporuje 3 délky klíče: 128, 192 a 256 bitů

2.2 Matematický úvod

Reprezentace bytů

Jak již víme, vstupem pro AES je blok 128 bitů, což je 16 bytů. Tedy každý z 16 bytů je reprezentován uspořádanou osmicí $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$, kde každé b_i nabývá hodnoty 0 nebo 1. Takovýto byte ovšem odpovídá též polynomu nejvýše 7. stupně ve tvaru:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \qquad (2.1)$$

Například byte {01100011} odpovídá v polynomiální reprezentaci
$x^6 + x^5 + x + 1$.

Každý byte ale také odpovídá nějakému prvku tělesa GF(2^8). Tyto prvky zapisujeme ve tvaru {x,y}, kde x a y jsou prvky šestnáctkové soustavy . Reprezentaci čtveřic bitů prvky této soustavy zachycuje obrázek 2.1.

Čtveřice bitů	v 16-tkové s.	Čtveřice bitů	v 16-tkové s.	Čtveřice bitů	v 16-tkové s.	Čtveřice bitů	v 16-tkové s.
0000	0	0100	4	1000	8	1100	c
0001	1	0101	5	1001	9	1101	d
0010	2	0110	6	1010	a	1110	e
0011	3	0111	7	1011	b	1111	f

Obrázek 2.1: Hexadecimální reprezentace čtveřic bitů

Odtud vidíme, že byte {01100011} může být reprezentován prvkem {63}.

Matematické operace

Sčítání: Ve dvojkové soustavě provádíme sčítání pomocí operace XOR (značení $\oplus$), kde platí $0 \oplus 0 = 0$, $1 \oplus 0 = 0 \oplus 1 = 1$ a $1 \oplus 1 = 0$. Podobně provádíme sčítání v polynomiální reprezentaci - sečteme pomocí operace XOR koeficienty u stejných mocnin.

Například následující výrazy jsou ekvivalentní:

$\{01010111\} \oplus \{10000011\} = \{11010100\}$	v binární notaci
$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = (x^7 + x^6 + x^4 + x^2)$	v polynomiální notaci
$\{57\} \oplus \{83\} = \{d4\}$	v hexadecimální notaci

Násobení: Polynomiální reprezentaci násobení v $GF(2^8)$ (značení $\bullet$) koresponduje s násobením polynomů modulo *ireducibilní polynom* stupně 8. Ireducibilním polynomem rozumíme takový polynom, který je dělitelný jen jednotkou a sám sebou. Pro AES je používán polynom

$$m(x) = x^8 + x^4 + x^3 + x + 1 \qquad (2.2)$$

Díky modulární redukci polynomem m(x) je zajištěno, že výsledný polynom bude stupně menšího než 8 a že tedy bude moci reprezentovat byte. Narozdíl od sčítání zde neexistuje jednoduchá operace na úrovni bytů, která by odpovídala násobení.

Násobení definované výše je asociativní a prvek {01} je multiplikativní jednotkou. Pro jakýkoli binární polynom b(x) stupně menšího než 8 existuje inverzní polynom $b^{-1}(x)$. Ten lze najít rozšířeným Eukleidovým algoritmem – spočtením polynomů a(x) a c(x) tak, že

$$b(x)a(x) + m(x)c(x) = 1 \qquad (2.3)$$

a tedy

$$b^{-1}(x) = a(x) \bmod m(x) \qquad (2.4)$$

Polynomy s koeficienty v $GF(2^8)$

Polynomy nejvýše třetího stupně s koeficienty z konečného tělesa lze definovat:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \qquad (2.5)$$

a můžeme je zaznamenávat ve tvaru $[a_0, a_1, a_2, a_3]$.

Sčítaní a násobení takovýchto polynomů probíhá na koeficientech těchto polynomů za pomocí výše zadefinovaných operací $\oplus$ a $\bullet$. Nechť máme nyní dva polynomy $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$, $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$.
Takto tedy vypadá součet a součin polynomů a(x) a b(x):

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0) \quad (2.6)$$

$$c(x) = a(x) \bullet b(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \quad (2.7)$$

kde

$c_0 = a_0 \bullet b_0$, $c_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1$,
$c_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2$, $c_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$,
$c_4 = a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$, $c_5 = a_3 \bullet b_2 \oplus a_2 \bullet b_3$,
$c_6 = a_3 \bullet b_3$, (2.8)

Výsledný polynom c(x) ale nereprezentuje čtveřici bytů, proto musíme provést jeho modulární redukci polynomem stupně 4. Algoritmus AES používá k této redukci polynom $x^4 + 1$ a tedy platí

$$x^i \mod(x^4 + 1) = x^{i \bmod 4} \quad (2.9)$$

Operaci násobení polynomů a(x), b(x) modulo $x^4 + 1$ značíme $a(x) \otimes b(x)$ a jejím výsledkem je polynom stupně menšího než 4, definovaný následovně:

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 \quad (2.10)$$

kde

$$\begin{aligned} d_0 &= (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \\ d_1 &= (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \\ d_2 &= (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \\ d_3 &= (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3) \end{aligned} \quad (2.11)$$

Je-li a(x) pevně daný polynom, potom můžeme operaci definovanou výše v rovnosti (2.10) zapsat v maticovém tvaru:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (2.12)$$

Protože $x^4 + 1$ není ireducibilní polynom nad $GF(2^8)$, tak toto násobení pevně daným polynomem není vždy nutně invertibilní. Nicméně algoritmus AES používá specifický pevně daný polynom, který má inverz:

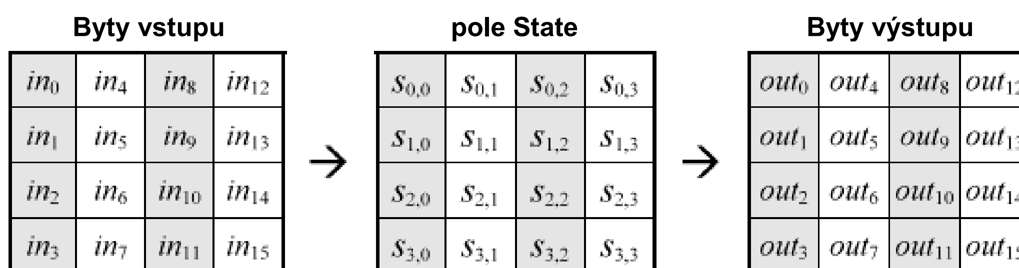
$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (2.13)$$
$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \quad (2.14)$$

Druhý polynom použitý v algoritmu AES má $a_0 = a_1 = a_2 = 0$, $a_3 = \{01\}$, což odpovídá polynomu x^3. Prohlédneme-li si rovnost (2.12), zjistíme, že

výsledek násobení takovýmto polynomem odpovídá pouhé rotaci bytů na vstupu. Tedy [b_0, b_1, b_2, b_3] přejde na [b_1, b_2, b_3, b_0].

2.3 Vstup a výstup algoritmu AES

Jak již víme, vstupem algoritmu AES je blok délky 128 bitů. Těchto 128 bitů se nejprve rozdělí po osmi do posloupnosti 16 bytů, ze kterých se vytvoří pole 4 x 4 bytů nazývané *State.* State potom vstupuje do samotného algoritmu, kde se pomoci níže popsaných operací substituují a míchají jednotlivé byty, ale struktura 4 x 4 zůstává. Výsledný State se opět rozloží do posloupnosti 16 bytů, ty pak do bloku 128 bitů. Vytvoření pole State je ukázáno na obrázku 2.2.



Obrázek 2.2: Vstup, State, Výstup

Počet sloupců pole state značíme Nb. Ve standardní verzi AES je tato hodnota pevně daná: Nb = 4 (Obr. 2.3).

Podobně upravíme bity klíče. Počet sloupců v poli klíče značíme Nk. Pole je tvaru 4 x Nk, kde Nk nabývá hodnot 4, 6 a 8, v závislosti na délce klíče (Obr. 2.3).

Druhou hodnotou závislou na délce klíče je počet rund, které budou v cyklu algoritmu vykonány. Značíme ji Nr a nabývá hodnot 10, 12 a 14 (Obr.2.3).

	Nk	Nb	Nr
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Obrázek 2.3: Hodnoty Nk, Nb a Nr v závislosti na délce klíče

Čtveřici bytů ve sloupci pod sebou nazýváme *slovo.* Hodnota Nb tedy udává počet slov bloku, Nk potom počet slov klíče.

2.4 Struktura šifry

Na začátku se otevřený text rozdělí do bloků po 128 bitech a z každého takového bloku se potom vytvoří State způsobem popsaným v části 2.3. Po přičtení klíče následuje cyklus 10,12 nebo 14 rund v závislosti na délce klíče. Poslední runda se ovšem mírně liší od předchozích Nr – 1 rund. Každá tato runda je kruhovou transformací, skládající se z posloupnosti čtyř funkcí: *SubBytes(), ShiftRows(), MixColumns()* a *AddRoundKey*, které jsou popsány v dalších částech této kapitoly. Poslední runda se liší od předchozích absencí funkce MixColumns() (Obr 2.4).

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

   state = in

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

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

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

   out = state
end
```

Obrázek 2.4: Zdrojový pseudokód šifry

V obrázku 2.4 se vyskytuje pole w[ ]. To představuje prodloužený klíč, který je popsán v části 2.9.

2.5 Transformace SubBytes()

Transformace SubBytes je nelineární substitucí bytů v poli State nezávisle na sobě. K tomuto se využívá substituční tabulky (S-box) (Obr. 2.6), která je invertibilní a je vytvořena složením dvou operací:

1. Spočteme mutiplikativní inverz daného bytu v tělese GF(2^8) (popsaný v části 2.2, odstavci násobení). Prvek {00} zůstává sám sebou.

2. Použijeme nasledující afinní transformaci (nad GF(2^8)):

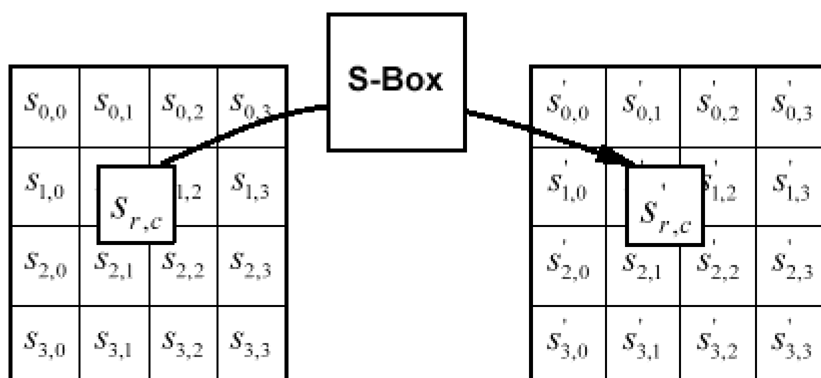
$$b_i' = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \qquad (2.15)$$

pro i od 0 do 7, kde b_i je i-tý bit daného bytu a c_i je i-tý bit bytu s hodnotou {63}, nebo-li {01100011}.

Chceme-li tuto afinní transformaci vyjádřit v maticovém zápisu, vypadá takto:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \qquad (2.16)$$

Obrázek 2.5 ukazuje efekt transformace SubBytes() na pole State.



Obrázek 2.5: SubBytes() aplikuje S-box na každý jednotlivý byte v State

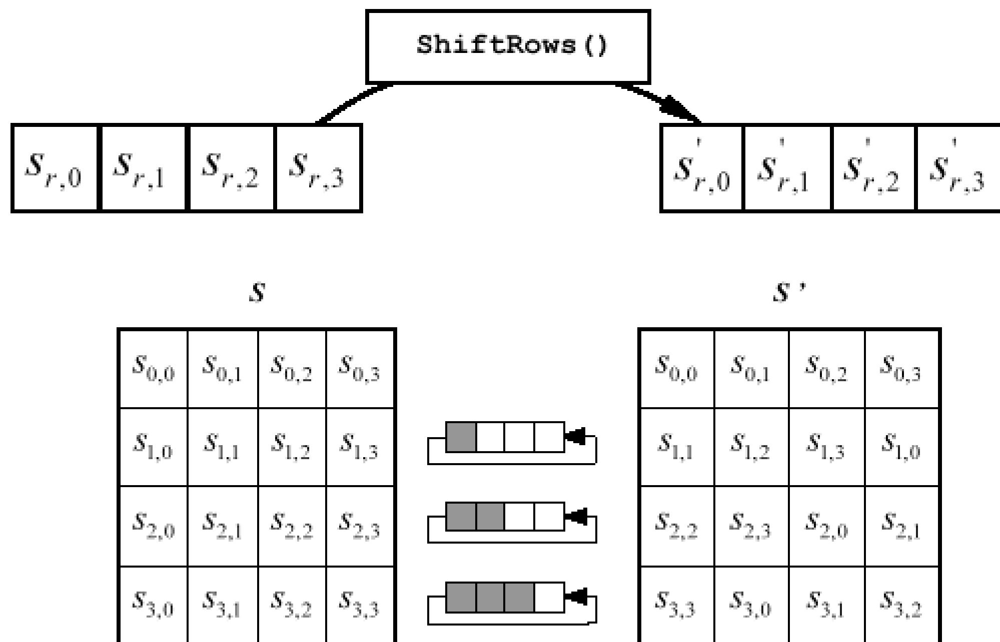
S-box používaný transformací SubBytes() používá zápisu v šestnáctkové soustavě(Obr 2.6)

x \ y	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

Obrázek 2.6: S-box: Substituční hodnoty pro byte xy

2.6 Transformace ShiftRows()

V transformaci ShiftRows() se byty v dolních třech řádcích pole State cyklicky posouvají o určitý počet políček doleva. Konkrétně druhý řádek o 1, třetí o 2 a čtvrtý o 3 políčka. Toto cyklické posunutí bytů v řádcích zachycuje obrázek 2.7.



Obrázek 2.6: ShiftRows() cyklicky posouvá byty v dolních 3 řádcích ve State

2.7 Transformace MixColumns()

Transformace MixColumns() se provádí na pole State sloupec po sloupci, kde každý ze sloupců uvažujeme jako polynom stupně menšího nebo rovno 3 s koeficienty v GF(2^8), jak je popsáno v části 2.2, v odstavci Polynomy s koeficienty v GF(2^8). Násobení se provádí modulo $x^4 + 1$, jako pevně daný polynom používáme

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{2.17}$$

Toto násobení lze vyjádřit v maticovém zápisu takto:

$s'(x) = a(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{pro c = 0,1,...,Nb-1}$$

Výsledkem tohoto násobení je nahrazení čtveřice bytů ve sloupci následující čtveřicí

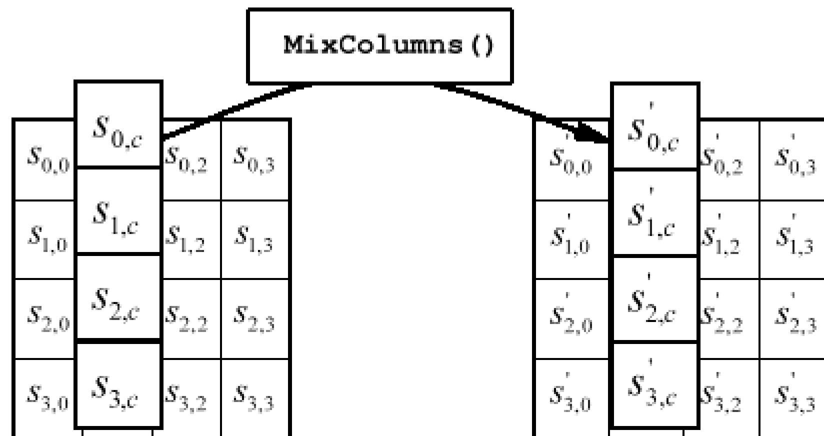
$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

Obrázek 2.7 ukazuje použití transformace MixColumns()



Obrázek 2.7: MixColumns() se provádí na State sloupec po sloupci

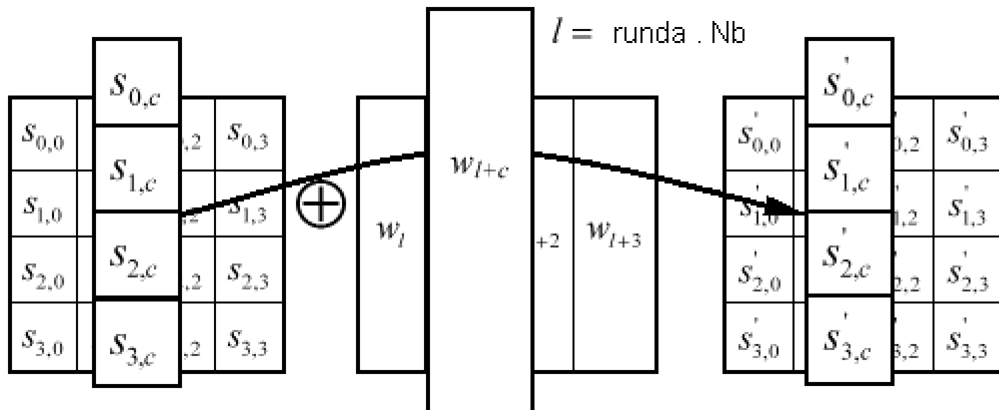
2.8 Transformace AddRoundKey()

V transformaci AddRoundKey() je přičítána operací XOR k poli State vždy část prodlouženého klíče (popsaného v části 2.9) příslušná dané rundě:

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{\text{runda.Nb} + c}] \qquad (2.18)$$

Kde $[w_i]$ jsou slova prodlouženého klíče (popsaná v části 2.9) a c probíhá hodnoty 0,1,..., Nb – 1

Obrázek 2.8 znázorňuje přičtení klíče, kde l = runda.Nb. Čtveřicím bytů ve sloupci v klíči říkáme slovo (viz část 2.3).



Obrázek 2.8: AddRoundKey sčítá pomocí opetace XOR jednotlivé sloupce ze State se slovy prodlouženého klíče

2.9 Prodloužení klíče

Jak již z předchozího víme, mají klíče pro AES délku 128, 192 a 256 bitů. To je však málo. My potřebujeme nagenerovat klíč, který obsahuje celkem Nb(Nr + 1) slov. K tomu v algoritmu AES slouží procedura *KeyExpansion(),* která z daného klíče vytvoří lineární pole 4-bytových slov, značíme w[i], kde i = 0,1,..., Nb(Nr + 1) – 1. Zdrojový pseudokód procedury KeyExpansion() zachycuje obrázek 2.9

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

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

Obrázek 2.9: Zdrojový pseudokód pro KeyExpansion()

Funkce *SubWord()* aplikuje na každý byte vstupního slova zvlášť substituci S-box (část 2.5).
Funkce *RotWord()* provede na vstupním slově $[a_0, a_1, a_2, a_3]$ cyklickou permutaci, jejímž výsledkem je slovo $[a_1, a_2, a_3, a_0]$.
Rcon[i] je pole rundovních konstant, obsahující hodnoty $[x^{i-1}$,{00},{00},{00}], kde x^{i-1} jsou mocniny x v GF(2^8). Tedy x^0 = {01}, x^1 = {02}, x^2 = {04}, atd. (viz. část 2.2 odstavec Reprezentace bytů).

Důležitá poznámka: Pokud je délka klíče 256 bitů (Nk = 8), je průběh prodloužení klíče mírně odlišný od průběhu při délkách 128 a 192 bitů. Je-li Nk = 8 a i – 4 je násobkem Nk, potom SubWord() je aplikován na w[i – 1] dříve, než XOR.

2.10 Přímá inverzní šifra

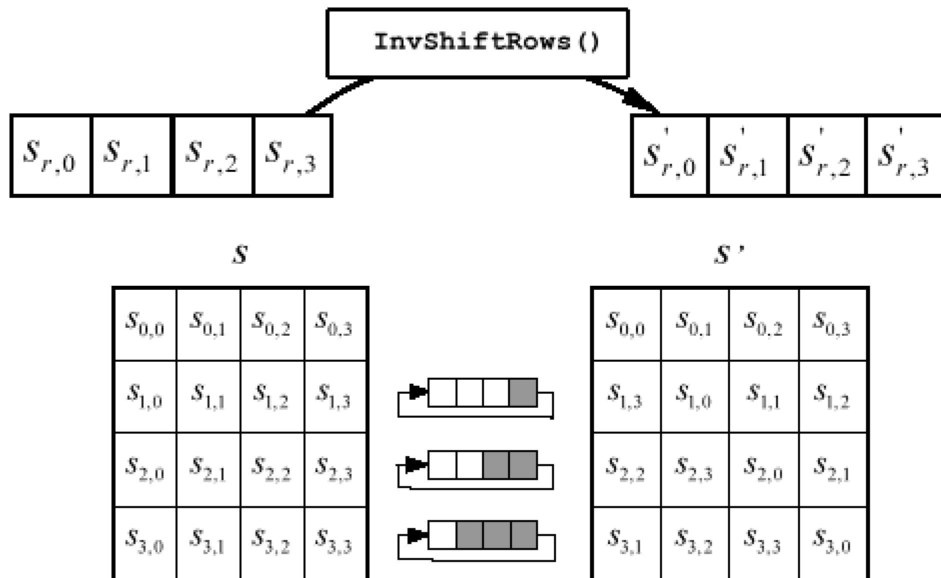
Jednotlivé transformace šifry (části 2.5, 2.6, 2.7, 2.8) mohou být invertovány a poté implementovány v obráceném pořadí. Tím se vytvoří *Přímá inverzní šifra* pro AES algoritmus. Jednotlivé inverzní transformace – InvShiftRows(), InvSubBytes(), InvMixColumns() a AddRoundKey() – budou popsány v následujících odstavcích.

Obrázek 2.10 ukazuje zdrojový pseudokód inverzní šifry. Pole w[i] obsahuje prodloužený klíč, jehož vytvoření bylo popsáno v části 2.9.

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

   state = in

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

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for

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

   out = state
end
```

Obrázek 2.10: Zdrojový pseudokód inverzní šifry

InvShiftRows() je inverzí k transformaci ShiftRows() (viz. část 2.6). To znamená, že byty v dolních třech řádcích ve State jsou cyklicky posouvány doprava místo doleva, jak je zachyceno na obrázku 2.11



Obrázek 2.11: InvShiftRows() cyklicky posouvá byty v dolních 3 řádcích v State

InvSubBytes() je inverzí substituce bytů, ve které je inverze S-box (viz. část 2.5) provedena na každý jednotlivý byte ve State. Tedy skládá se z inverze afinní transformace a následného spočtení multiplikativního inverzu v GF(2^8). Inverzní S-box používaný v InvSubBytes() je na obrázku 2.12.

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

Obrázek 2.12: Inverzní S-box: Substituční hodnoty pro byte xy

InvMixColumns() je inverzí k transformaci MixColumns(), která je prováděna na jednotlivých sloupcích ve State (viz část 2.7). Sloupce jsou opět uvažovány ve formě polynomů nad GF(2^8) a násobeny pevně daným polynomem $a^{-1}(x)$ modulo $x^4 + 1$, kde

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{2.19}$$

V maticovém zápisu tato operace vypadá takto:
$s'(x) = a^{-1}(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \textbf{pro c = 0,1,...,Nb} \tag{2.20}$$

Výsledkem tohoto násobení je nahrazení čtveřice bytů ve sloupci následující čtveřicí

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

Inverze k **AddRoundKey()**: Transformace AddRoundKey() je sama svou inverzí, neboť dvojí provedení operace XOR dává identitu.

2.11 Ekvivalentní inverzní šifra

V přímé inverzní šifře uvedené v části 2.10 a na obrázku 2.10 je použita opačná posloupnost transformací (inverzních), ale využívá se stejný prodloužený klíč, jako při vlastním šifrování. Nicméně některé vlastnosti algoritmu AES umožňují použít *Ekvivalentní inverzní šifru*, která naopak zachovává posloupnost transformací (inverzních). To ovšem vyžaduje jistou změnu prodlouženého klíče.
Dvě vlastnosti umožňující použití ekvivalentní inverzní šifry jsou:

1. Transformace SubBytes() a ShiftRows() komutují. To znamená, že SubBytes() následovaná ShiftRows() je ekvivalentní ShiftRows() následované SubBytes(). Totéž platí i pro jejich inverze InvSubBytes() a InvShiftRows().

2. Transformace MixColumns() a InvMixColumns() jsou lineární vzhledem ke sloupcům vstupu, což znamená, že

InvMixColumns(State XOR RoundKey) =
InvMixColumns(State) XOR InvMixColumns(RoundKey)

Tyto vlastnosti dovolují prohození pořadí tranformací InvSubBytes() a InvShiftRows() a taktéž prohození transformací AddRoundKey() a InvMixColumns().

Takovéto změny vedou k tomu, že ekvivalentní inverzní šifra má efektivnější strukturu než přímá inverzní šifra.

Zdrojový pseudokód ekvivalentní inverzní šifry je na obrázku 2.13. Pole slov dw[i] obsahuje pozměněný prodloužený klíč, jehož vytvoření zachycuje také obrázek 2.13.

```
EqInvCipher(byte in[4*Nb], byte out[4*Nb], word dw[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvSubBytes(state)
      InvShiftRows(state)
      InvMixColumns(state)
      AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
   end for

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

   out = state
end
```

Při použití ekvivalentní inverzní šifry je následující zdrojový pseudokód přidán na konec procesu vytvoření prodlouženého klíče (viz část 2.9)

```
   for i = 0 step 1 to (Nr+1)*Nb-1
      dw[i] = w[i]
   end for

   for round = 1 step 1 to Nr-1
      InvMixColumns(dw[round*Nb, (round+1)*Nb-1])     //  note  change  of
type
   end for
```

Obrázek 2.13: Zdrojový pseudokód pro ekvivalentní inverzní šifru

2.12 Zvláštní poděkování

Na tomto místě bych zvláště rád poděkoval americkému National Institute of Standards and Technology (NIST), na jehož stránkách jsem nalezl nejlepší a nejpřehlednější obrázky a schémata týkající se problematiky konstrukce AES, z nichž jsem většinu s drobnými úpravami převzal do své práce.

Kapitola 3

Implementace AES

3.1 8-bitové platformy

Dobrý výkon na 8-bitových platformách je důležitý od dob, kdy se takovéto procesory začaly vyskytovat na smart kartách a kdy mnoho kryptografického softwaru na smart kartách funguje.

V algoritmu AES, popsaném v předcházející kapitole, probíhá násobení hodnotou {02}, které značíme *xtime().* Polynom asociovaný s {02} je x. Implementace takovéto operace byla popsána v části 2.2. xtime() může být implementována posunutím a XOR operací. Abychom předcházeli časovým útokům (část 4.8), je třeba dávat pozor na to, aby xtime() implementovaný tímto způsobem zabral pevný počet cyklů nezávisle na argumentu této funkce. To lze provést přidáním falešných instrukcí na správná místa. Nicméně tento přístup bude pravděpodobně představovat slabinu vůči energetické analýze (část 4.8). Lepší přístupem je vytvoření tabulky M, kde M[a] = {02} . a. Pak může být průběh xtime() implementován jako vyhledávání v tabulce M.

Protože všechny prvky $GF(2^8)$ lze napsat ve tvaru součtu mocnin {02}, můžeme násobení jakoukoliv konstantou implementovat jako opakované použití funkce xtime().

Šifrování

Na 8-bitových procesorech lze šifrování naprogramovat jednoduchou implementací každého kroku. Implementace ShiftRows() a AddRoundKey() je přímo daná jejich popisem. Implementace SubBytes() vyžaduje substituční tabulku všech 256 bytů. MixColumns() lze algoritmicky realizovat takto:

Nechť `t = a[0] ⊕ a[1] ⊕ a[2] ⊕ a[3]`, kde a je daný sloupec. Pak postupujeme takto:

```
u = a[0];
v = a[0]⊕a[1];     v = xtime(v);     a[0] = a[0]⊕v⊕t
v = a[1]⊕a[2];     v = xtime(v);     a[1] = a[1]⊕v⊕t
v = a[2]⊕a[3];     v = xtime(v);     a[2] = a[2]⊕v⊕t
v = a[3]⊕u;        v = xtime(v);     a[3] = a[3]⊕v⊕t
```

Implementování prodloužení klíče je jednorázová operace, která zabírá asi nejvíce paměti RAM na smart kartě. Kromě toho ve většině aplikacích běžících na smart kartách se šifruje a dešifruje jen pár bloků při jednom běhu. Z tohoto důvodu je velký výkon věnován novému vygenerování klíče pro každou aplikaci, namísto jeho ukládání do paměti. Prodloužení klíče lze tedy implementovat použitím cyklického bufferu o velikosti 4.Nk bytů. Po použití všech bytů bufferu dojde k aktualizaci jeho obsahu.

Dešifrování
Při implementaci na 8-bitové platformy nepřináší užití ekvivalentní inverzní šifry žádnou výhodu. Proto se používá přímá inverzní šifra. Z toho vyplývá, že jedinou obtížněji implementovatelnou inverzní transformací je InvMixColumns().

Zatímco MixColumns() operuje s koeficienty {01}, {02} a {03}, u InvMixColumns() to jsou koeficienty {09}, {0E}, {0B} a {0D}.

P. Barreto si všiml následující relace mezi polynomem c(x) v MixColumns() a polynomem d(x) v InvMixColumns().

$$d(x) = (\{04\}\, x^2 + \{05\})c(x) \pmod{x^4 + \{01\}} \tag{3.1}$$

V maticovém zápisu tato relace vypadá takto:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{bmatrix} \tag{3.2}$$

Důsledkem toho je, že InvMixColumns() může být implementována jako jednoduchý předzpracovávající krok, který je poté následován transformací MixColumns(). Tento krok lze realizovat takto:

```
u = xtime(xtime(a[0] ⊕ a[2]);        kde a je opět daný sloupec
v = xtime(xtime(a[1] ⊕ a[3]);
a[0] = a[0] ⊕ u;
a[1] = a[1] ⊕ v;
a[2] = a[2] ⊕ u;
a[3] = a[3] ⊕ v;
```

3.2 32-bitové platformy

Různé kroky kruhové transformace lze zkombinovat do jedné sady vyhledávacích tabulek, dovolujících velmi rychlou implementaci pro procesory s délkou slova 32 a více.

Nechť a je vstupem kruhové transformace a b je výstupem transformace SubBytes() (dále značené SUB):

$$b_{i,j} = SUB[a_{i,j}], \quad 0 \le i < 4,\ 0 \le j < Nb \tag{3.3}$$

Nechť dále c je výstupem transformace ShiftRows() a d je výstupem transformace MixColumns():

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j+0 \bmod Nb} \\ b_{1,j+1 \bmod Nb} \\ b_{2,j+2 \bmod Nb} \\ b_{3,j+3 \bmod Nb} \end{bmatrix} \qquad 0 \le j < Nb \tag{3.4}$$

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} \qquad 0 \le j < Nb \tag{3.5}$$

Rovnosti (3.3), (3.4) a (3.5) lze nyní zkombinovat do:

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} SUB[a_{0,j+0 \bmod Nb}] \\ SUB[a_{1,j+1 \bmod Nb}] \\ SUB[a_{2,j+2 \bmod Nb}] \\ SUB[a_{3,j+3 \bmod Nb}] \end{bmatrix} \qquad 0 \le j < Nb \tag{3.6}$$

Násobení matic lze nyní přepsat do lineární kombinace čtyř sloupcových vektorů:

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} SUB[a_{0,j+0 \bmod Nb}] \oplus \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} SUB[a_{1,j+1 \bmod Nb}] \oplus \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} SUB[a_{2,j+2 \bmod Nb}] \oplus \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} SUB[a_{3,j+3 \bmod Nb}] \qquad 0 \le j < Nb \tag{3.7}$$

Nyní definujeme čtyři T-tabulky: T_0, T_1, T_2, T_3:

$$T_0[a] = \begin{bmatrix} 02.SUB[a] \\ 01.SUB[a] \\ 01.SUB[a] \\ 03.SUB[a] \end{bmatrix} \qquad T_1[a] = \begin{bmatrix} 03.SUB[a] \\ 02.SUB[a] \\ 01.SUB[a] \\ 01.SUB[a] \end{bmatrix}$$

$$T_2[a] = \begin{bmatrix} 01.SUB[a] \\ 03.SUB[a] \\ 02.SUB[a] \\ 01.SUB[a] \end{bmatrix} \qquad T_3[a] = \begin{bmatrix} 01.SUB[a] \\ 01.SUB[a] \\ 03.SUB[a] \\ 02.SUB[a] \end{bmatrix} \tag{3.8}$$

Každá tato tabulka obsahuje 256 4-bytových slov a zabírají dohromady 4 kB místa.

Použitím těchto tabulek můžeme rovnost (3.7) přepsat do tvaru:

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = T_0[a_{0,j+0 \bmod Nb}] \oplus T_1[a_{1,j+1 \bmod Nb}] \oplus T_2[a_{2,j+2 \bmod Nb}] \oplus T_3[a_{3,j+3 \bmod Nb}] \qquad (3.9)$$

$$0 \le j < Nb$$

Připočteme-li, že transformaci AddRoundKey() lze implementovat jako přídavnou 32-bitovou XOR operaci na sloupec, dostneme implementaci pomocí vyhledávací tabulky zabírající 4 kB, která se skládá ze čtyř vyhledávacích tabulek a čtyř XOR operací na sloupec za jednu rundu.

Navíc tabulky T_0, T_1, T_2 a T_3 jsou pouze rotované verze jedné a té samé tabulky pro všechny hodnoty a. Tudíž lze používat jen jednu tabulku, čímž se velikost celkové tabulky zmenší na 1kB.

V poslední rundě se neprovádí operace MixColumns() a je tedy potřeba samotná tabulka pro SUB. Tato potřeba může být vyřešena extrahováním SUB-tabulky z T-tabulky v posledním kole.

Většina operací v KeyExpansion() jsou 32-bitové XOR operace. Zbylé transformace jsou aplikací SUB a cyklického posunu o 8 bitů. To lze implementovat velmi efektivně.

Dešifrování je možno implementovat podobnými vyhledávacími tabulkami, ale pro algoritmus ekvivalentní inverzní šifry. Transformace KeyExpansion() ve spojení s algoritmem pro ekvivalentní inverzní šifru je ale pomalejší.

3.3 Hardware určený pro šifrování

AES je vhodný k implementování na hardware určený pro šifrování. Je zde však několik kompromisů mezi velikostí čipu a možnou rychlostí. Protože softwarová implementace na běžné procesory je velmi rychlá, budou nároky na hardwarovou implementaci rozděleny do dvou případů:

1. Extrémně rychlý čip bez jakýchkoliv omezení na velikost: T-tabulky budou “hardwired” a XOR operace se zapojí paralelně.

2. Kompaktní coprocesor na smart kartě: pro tuto platformu budou typicky “hardwired” transformace SUB a xtime (nebo kompletní MixColums()).

3.4 Multiprocesorové platformy

U multiprocesorových platforem můžeme uvažovat o paralelismu při kruhové transformaci. Všechny čtyři kroky rundy působí paralelním způsobem na byty, řádky, nebo sloupce pole State. V implementaci pomocí vyhledávacích tabulek lze v principu zapojit všechna vyhledávaní v tabulkách paralelně. XOR operaci můžeme většinou také zapojit paralelně.

KeyExpansion() má jinou povahu. Hodnota w[i-1] je potřeba pro vypočtení hodnoty w[i]. Nicméně v aplikacích, kde je rychlost průběhu šifry rozhodující, se transformace KeyExpansion() provádí vždy jednou pro velké množství dat. V aplikacích, kde se klíče mění často, lze KeyExpansion() a rundy šifry zapojit paralelně.

Kapitola 4

Kryptoanalýza AES

4.1 Zkrácené rozdíly

Koncept zkrácených rozdílů popsal L. Knudsen v *Truncated and higher order differentials.* Příslušná třída útoků využívá faktu, že v některých šifrách mají stopy odlišností tendenci se shlukovat. Shlukování nastává, jestliže je pro určité vstupní vzory odlišností a výstupní vzory odlišností počet stop odlišností obzvláště velký. Očekávaná pravděpodobnost, že stopa odlišnosti zůstane v mezích shluku lze spočítat nezávisle na pravděpodobnostech jednotlivých stop odlišností. Šifry, ve kterých působí všechny jejich kroky na State ve svazcích, jsou náchylné na takovéto útoky. Protože toto je případ šifry AES, která operuje s celými byty na místo jednotlivých bitů, byla prověřována její odolnost vůči zkráceným rozdílům. Pro šest a více rund nebyl nalezen žádný útok, který by byl rychlejší než vyzkoušení všech možných klíčů.

4.2 Saturační útok

L. Knudsen též útočil na bolokovou šifru Square, tzv. "čtvercový" útok. Tento útok využívá bytově orientovanou strukturu Square a je též aplikovatelný na šifru AES. N. Ferguson v *Improved cryptanalysis of Rijndael* navrhnul určité optimalizace, jak redukovat pracovní faktor tohoto útoku. V *The saturation attack – a bait for Twofish* navrhuje S. Lucks pro tento typ útoku jméno Saturační útok. Dříve se tento typ útoku názýval Strukturální útok.

Saturační útok je útok s předem vybraným otevřeným textem na šifru s kruhovou strukturou. Lze ho nasadit nezávisle na výběru substituce S-box v nelineárním kroku, nebo na klíči. Jedna z verzí útoku je, že krok MixColumns() má maximální počet větvení (číslo, dané určitým vzorcem) a že bytová transformace MixColumns() je difuzionálně optimální. Jestliže jedna z těchto podmínek není úplně splněna, je sice útok mírně odlišný, ale má srovnatelnou složitost.

Do šesti rund je tento útok rychlejší, než vyzkoušení všech možných klíčů.

4.3 Gilbert-Minierův útok

Saturační útok na AES redukovaný na šest rund je založen na faktu, že tři rundy AES lze rozpoznat od náhodné permutace. H. Gilbert a M. Minier vynalezli čtyř-rundový rozpoznávač, se kterým lze útočit na AES redukovaný na sedm rund. Díky zvýšenému pracovnímu faktoru je tento útok účinnější než vyzkoušení všech možných klíčů jen pro některé délky klíčů (pro klíč délky 128bitů je rychlejší, pro klíč délky 192 bitů jsou ekvvalentní a pro klíč délky 256 bitů je pomalejší).

4.4 Interpolační útok

V The *interpolation attack on block ciphers* představili T. Jakobsen a L. Knudsen nový útok na blokové šifry. Útočník při tomto druhu útoku konstruuje polynomy s pomocí párů vstup-výstup. Útok je možné provádět, jestliže má daná šifra kompaktní algebraické vyjádření. Lze ho kombinovat tak, aby vedl k výrazům, jejichž složitost je zvladatelná. Základem útoku je fakt, že pokud má zkonstruovaný polynom malý stupeň, pak je třeba najít jen několik párů vstup-výstup k odvození koeficientů polynomu.

Transformace SUB tvoří z bytů vstupu byty výstupu. Tedy může být vyjádřena pomocí polynomu nad GF(2^8). Polynomiální vyjádření transformace SUB můžeme nalézt například Lagrangeovou interpolační metodou. Toto vyjádření je dáno:

$$\mathrm{SUB}[x] = \{63] + \{8F\}\, x^{127} + \{B5\}\, x^{191} + \{01\}\, x^{223} + \{F4\}\, x^{239} + \{25\}\, x^{247} + \{F9\}\, x^{251} + \{09\}\, x^{253} + \{05\}\, x^{254} \qquad (4.1)$$

Toto komplikované vyjádření transformace SUB v kombinaci s efektem mixujících a transformačních kroků zamezuje použití interpolačního útoku na více než jen několik rund.

4.5 Symetrické vlastnosti a slabé klíče jako u šifry DES

Navzdory velkému množství symetrie, kterou šifra obsahuje, byla věnována velká péče eliminaci jejího symetrického chování. Ta byla docílena rundovními konstantami, které se pro jednotlivé rundy liší. Fakt, že šifra i její inverze používají odlišné transformace, prakticky eliminuje možnost slabých a poloslabých klíčů, které u šifry DES popsal D. Davies v *Some Regular Properties of the DES*. Nelinearita transformace KeyExpansion() prakticky vylučuje možnost ekvivalentních klíčů.

4.6 Slabé klíče jako u šifry IDEA

Nyní se budeme zabývat klíči, jež mají za následek permutaci s detekovatelnými slabostmi. Nejznámějším takovým případem jsou slabé klíče šifry IDEA. Tato slabost se typicky vyskytuje, pokud je nelinearita šifry silně závislá na aplikaci klíče. U šifry AES je nelinearita zajištěna transformací SubBytes(), tudíž AES nevykazuje žádné známky takovýchto slabých klíčů.

4.7 Related-Key útoky

V *New Types of Cryptoanalytic Attacks Using Related Keys* představil E. Biham related-key útok. J. Kelsey později ukázal, že několik šifer vykazuje related-key slabosti. Při takovém útoku se předpokládá, že máme přístup k zašifrovanému textu, jenž je výsledkem šifrování za pomocí odlišných (neznámých nebo částečně neznámých) klíčů s určitými souvislostmi. Použití klíče v AES s jeho velkou difuzí a nelinearitou činí použití takovéhoto útoku velmi obtížným. N. Ferguson popsal v *Improved cryptoanalysis of Rijndael* Key-Related útok na šifru Rijndael redukovanou na devět rund. Takovýto útok fungoval na verzi s 128 bitovým vstupem a 256 bitovým klíčem. Vyžadoval 2^{72} vybraných otevřených textů a jeho pracovní faktor byl 2^{227} šifrování, což je přece jen rychlejší, než vyzkoušení všech možných klíčů jen pro některé délky klíčů

4.8 Implementační útoky

Implementační útoky jsou založeny nejen na matematických vlastnostech šifry, ale i na fyzikálních charakteristikách její implementace. Typickým příkladem jsou časové útoky a energetická analýza, které ve svých pracích představil P. Kocher. Při časových útocích se využívá měření celkového trvání zašifrování k odvození informací o klíči. Při energetické analýze se měří množství energie spotřebované šifrovacím zařízením a to pak slouží k odvození informací o klíči. Mezi útoky pomocí energetické analýzy lze zahrnout i útoky, které využívají jiné měřitelné hodnoty, jako například radiaci nebo vyzařované teplo.

Časové útoky:
Časového útoku je možné použít za předpokladu, že doba běhu algoritmu závisí na hodnotách klíče. Předpokládejme, že máme implementaci šifry, ve které se vykoná určitá instrukce za podmínky, že jistý na klíči závislý mezivýsledek b nabude určité hodnoty. Potom tedy můžeme z pečlivě změřeného času určit, zda byla daná instrukce vykonána a tak odvodit hodnotu b. K tomu totiž stačí změřit dobu běhu algoritmu pro jednotlivé

hodnoty b, pokud ovšem ostatní parametry ovlivňující tuto dobu zabírají konstantní nebo v průměru stejný čas.

Implementaci lze ochránit proti časovým útokům zajištěním nezávislosti doby běhu na hodnotách klíče. Toho můžeme dosáhnout vložením přídavných instrukcí do nejkratších průběhů šifry tak, aby tento průběh trval pro všechny hodnoty klíče stejně dlouho. Takové řešení však může vést k tomu, že šifra pozbude ochrany před útoky pomocí energetické analýzy.

AES má jen jednu možnou slabinu vzhledem k časovým útokům. Tou je implementace násobení v konečné tělese GF(2^8) v transformaci MixColumns(), jmenovitě jde o funkci xtime() . Ostatní operace v AES jsou implementovány tak, že zabírají vždy konstantní čas. Slabinu v xtime() lze jednoduše odstranit definováním 256-bytové tabulky a implementováním funkce xtime() jako vyhledávací tabulky.

Energetická analýza:

Jednoduchá energetická analýza je útokem, při kterém se útočník snaží změřit množství energie spotřebované daným zařízením během šifrovacího procesu. Tento druh útoku je typicky možné použít pro zařízení, která potřebují externí zdroj energie, jako například smart karty. Jestliže množství spotřebované energie závisí na provedených instrukcích, pak může útočník zkusit odvodit pořadí prováděných instrukcí. Pokud pořadí nebo typ instrukce závisí na hodnotách klíče, pak naměřené spotřeby energie vypovídají o hodnotách klíče. AES lze snadno implementovat tak, že má pevné pořadí instrukcí a tím je chráněna vůči tomuto útoku.

U většiny procesorů závisí spotřeba energie při vykonání určité operace na hodnotě operandů. Obvykle je však rozdíl ve spotřebě energie pro rozdílné operandy tak malý, že zanikne v šumu a není tak odhalen při měření. Nicméně kombinace měření velké spousty běhů šifrovacího algoritmu dává útočníkovi možnost zjistit průměrný šum a tak může získat informace o hodnotách operandů. Tento druh útoku se nazývá *Diferenciální energetická analýza.* Ochránit implementaci vůči takto sofistikovanému útoku je mnohem obtížnější, než vůči časovému útoku nebo jednoduché energetické analýze. Tento druh útoku dělíme do tří tříd:

Ochrana jednotlivé instrukce: Je možné redukovat zranitelnost každé jednotlivé instrukce vůči energetické analýze. Prvním příkladem je takzvané *vyvážení zatížení*, kterého lze dosáhnout změnou designu hardwaru, abychom minimalizovali, nebo úplně eliminovali závislost spotřeby energie na hodnotách operandů. To můžeme také simulovat změnou softwaru tak, aby souvislost mezi spotřebou energie a hodnotami operandů byla malá. Bohužel to zatím vypadá, že tuto souvislost nelze úplně odstranit.

Další technikou je takzvané *maskování operandů.* Zde jsou instrukce operandu x nahrazeny instrukcemi operandu x^I a x^{II}, kde x^I a x^{II} jsou pro útočníka nepředvídatelné. Jen spojení znalostí o x^I a x^{II} nám prozradí informace o hodnotě x.

Nevýhodou ochrany jednodlivé instrukce je její opákování pro každou v algoritmu použitou instrukci. Fakt, že AES lze implementovat jen pomocí operace XOR a použití vyhledávacích tabulek je tedy jasnou výhodou.

Desynchronizace: Kromě zaměření se na ochranu každé jednotlivé instrukce zvlášť, můžeme také zkusit omezit vliv každé instrukce. Toho lze dosáhnout desynchronizací: změnou pořadí instrukcí pro každé šifrování,

nebo pro každou jeho část. To ztíží útočníkovi získání smysluplné statistiky. Paralelismus v kruhové transformaci AES dovoluje několik variant pořadí instrukcí. Počet různých pořadí je však omezen.
Složitost prodloužení klíče: V *A cautionary note regarding evaluation of AES candidates on smart cards* S. Chari tvrdí, že složité schéma prodloužení klíče pomáhá zvýšit odolnost vůči energetické analýze. Pokud znalost rundovního klíče neumožňuje rekonstrukci klíče šifry, pak bude muset útočník odhalit více (možná všechny) rundovní klíče, aby si mohl zprávu přečíst. Tvrdí se, že jednoduchost prodloužení klíče v případě AES je velkou nevýhodou. Je však zřejmé, že je-li možné odhalit jeden rundovní klíč, pak lze se stejným úsilím odhalit i ostatní rundovní klíče. Mimoto je velmi pravděpodobné, že další úsilí na odhalení více rundovních klíčů se projeví jen úměrně delší výpočetní dobou. Navíc vlastní výpočty prodlouženého klíče jsou cílem energetické analýzy. V tomto ohledu je složité prodlužování klíče nevýhodou.

Literatura

[1] AES Lounge of the ECRYPT – The European Network of Excellence in Cryptology http://www.iaik.tu-graz.ac.at/research/krypto/AES/index.php

[2] Daemon J., Rijmen V., *The Design of Rijndael*, Springer-Verlag 2002, ISBN 3-540-42580-2

[3] NESSIE – New European Schemes for Signatures, Integrity, and Encryption http://www.nessie.org

[4] NIST – National Institute of Standart and Technology http://crsc.nist.gov./publications/