Obsah

1 Úvod

Komunikace, ať už písemná nebo ústní, je stará jak lidstvo samo. Jakmile se lidé začali mezi sebou dorozumívat, vznikla potřeba přenášet některé informace takovým způsobem, aby se dostaly pouze ke konkrétní osobě. Tak vzniklo šifrování, z kterého se vyvinul obor zvaný kryptografie. Samozřejmě se způsoby i metody šifrování s novými poznatky a rozvojem myšlení hodně měnily, ale asi nejmarkantnější změna nastala s rozvojem počítačové technologie a nárůstem počítačové komunikace. V sedmdesátých letech se navíc začal používat elektronický bankovní systém a to bylo hlavním důvodem pro rozhodnutí ministerstva obchodu USA k vytvoření standardu pro ochranu dat zpracovávaných, ukládaných a přenášených počítači. Tento šifrový standard dostal jméno DES (Data Encryption Standard) a vstoupil v platnost roku 1977. Stal se nejrozšířenějším kryptografickým systémem na světě. Byl zárukou bezpečnosti ve veřejném i soukromém sektoru. Ve finančním sektoru se používal k ochraně komunikace po síti a dá se říct, že se stal mezinárodním standardem. Bohužel nic nevydrží věčně a tedy ani standard DES nevydržel technický pokrok a neodolal dešifrovacím útokům. Přestal být dostačující, jelikož byly vynalezeny přístroje, kterými se podařilo tuto šifru prolomit. Bylo tudíž nutné najít nový šifrový algoritmus, který by nahradil DES. Tento nový standard byl nazván AES (Advanced Encryption Standard) a je platný od roku 2002. Nyní doufáme, že bude spolehlivým ochráncem dat alespoň 10-15 let. Čas ukáže...

2 Definice

2.1 Slovník výrazů a akronymů

AES – Advanced Encryption Standard (zdokonalený šifrovací standard)

Afinní transformace – tranformace spočívající v násobení maticí s následným přičtením vektoru

Bit – binární číslice mající hodnotu 1 nebo 0

Blok – sled bitů, který zahrnuje vstup, výstup, stav a rundovní klíč. Délka sledu je počet bitů, které obsahuje. Bloky jsou také interpretovány jako pole bytů

Byte – skupina 8 bitů, která je brána jako samostatná jednotka nebo jako pole 8 jednotlivych bitů

DES – Data Encryption Standard (datový šifrovací standard)

Dešifrování – série transformací konvertující zašifrovaný text na otevřený text používaje šifrovací klíč

Expanze klíče – postup používaný k vygenerování série rundovních klíču ze šifrovacího klíče

Pole – očíslované seskupení identických jednotek (př.pole bitů)

Rijndael – kryptografický algoritmus předepsaný v AES

Rundovní klíč – rundovní klíče jsou hodnoty odvozené ze šifrovacího klíče pomocí procesu expanze, jsou aplikovány na stav při šifrování i dešifrování

S-box – nelineární substituční tabulka používaná v některých transformacích nahrazujících byty a v procesu expanze klíče k provedení substituce bytové hodnoty

Slovo – skupina 32 bitů, která je chápána jako samostatná jednotka nebo jako pole 4 bytů

Stav – mezivýsledek šifrování, který může být znázorněn jako obdelníková matice bytů, která má 4 řádky a Nb sloupců

Šifrovací klíč – tajný, kryptografický klíč, který je používaný v procesu expanze klíče k vygenerování množiny rundovních klíču; může být znázorněn jako obdelníkové pole bytů mající 4 řádky a Nk sloupců

Šifrování – sled transformací, které konvertují otevřený text na zašifrovaný text používaje přitom šifrovací klíč

Zašifrovaný text – výstupní data šifry nebo vstupní data inverzní šifry

2.2 Parametry algoritmů, symboly a funkce

AddRoundKey() – transformace při šifrování i dešiffování, ve které je rundovní klíč operací XOR přidán do stavu. Délka náhodného klíče se rovná velikosti stavu (např. pro $Nb = 4$, rundovní klíč se rovná 128 bitů/16 bytů)

InvMixColumns() – transformace při dešifrování, která je inverzní k MixColumns()

InvshiftRows() – transformace při dešifrování, která je inverzní k ShiftRows()

InvSubBytes() – transformace při dešifrování, která je inverzní k SubBytes()

MixColumns() – transformace při šifrování, která vezme všechny sloupce stavu, zamíchá jejich data (nezávisle jeden na druhém) a vede k vytvoření nových sloupců

Nb – počet sloupců (32-bitová slova) ve stavu. Pro tento standard $Nb = 4$

Nk – počet 32-bitových slov v šifrovacím klíči. Pro tento standard $Nk = 4, 6,$ nebo 8

Nr – počet cyklů, které jsou funkcí Nk a Nb (je fixní). Pro tento standard $Nr = 10, 12, 14$

Rcon[] – rundovní konstantní pole slov

RotWord() – funkce používaná v procesu expanze klíče, která bere 4-bytové slovo a provádí s ním cyklickou permutaci

ShiftRows() – transformace při šifrování, která zpracovává stav cyklickým přesouváním posledních 3 řádků stavu různými ofsety

SubBytes() – transformace při šifrování, která zpracovává stav za použití nelineárních bytových substitučních tabulek (S-boxy), které fungují na každém stavu bytově nezávisle

SubWord() – funkce používaná v procesu expanze klíče, která bere 4-bytové slovo na vstupu a aplikuje S-box na každý ze 4 bytů, aby vytvořila výstupní slovo

XOR – operace Exclusive-OR

$\oplus$ – operace Exclusive-OR

$\otimes$ – násobení dvou polynomů (každý se stupněm < 4) modulo $x^4 + 1$.

$\bullet$ – konečné prostorové násobení

3 Standard DES

3.1 Popis algoritmu

DES je blokovou šifrou, která rozdělí vstupní text na části po 64 bitech a tyto 64-bitové bloky M otevřeného textu zpracovává na 64-bitové bloky C šifrového textu. Píšeme

$$C = E_K(M) \quad \text{a} \quad M = D_K(C)$$

K šifrování se používá klíč K o velikosti 56 bitů, který vznikne vynecháním paritních bitů z osmibytového slova.
Algoritmus probíhá v 16-ti krocích. V každém z nich se vytvoří pracovní klíč K_i, kde $i = 0, 1, \ldots, 15$ následujícím způsobem. 56-bitový klíč K je permutován permutací $PC1$ a poté vložen do dvou 28-bitových registrů. Obsah obou registrů je

cyklicky posouván v každém kroku o p_i bitů, kde $p_i = 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1$. Označme toto posunutí SHL. Jejich výsledné zřetězení, které má opět 56 bitů, je podrobeno další permutaci $PC2$, která zároveň redukuje počet bitů na 48. Výstup této permutace je již pracovní klíč K_i. Postup vytváření pracovních klíčů je koncipován tak, aby každý bit klíče K byl obsažen v $K_0, \ldots, K_{15}$ dohromady 12 krát až 15 krát a probíhá buď před začátkem šifrovacího algoritmu nebo v jeho průběhu.
Vlastní algoritmus zpracovávající blok M nejdříve provede počáteční permutaci IP, která permutuje všech 64 bitů otevřeného textu. Poté je rozdělen na pravou R_i a levou L_i polovinu každou o velikosti 32 bitů. Pak probíhá 16 identických kroků, které vytvářejí dvojici (L_{i+1}, R_{i+1}) z dvojice (L_i, R_i) za pomocí pracovního klíče K_i. R_i je nejdříve rozšířen na 48 bitů prostřednictvím expanze E tak, že

$$\begin{aligned} E(R_i) &= E(r_1, r_2, \ldots, r_{31}, r_{32}) \\ &= (r_{32}, r_1, r_2, r_3, r_4, r_5, r_4, r_5, r_6, r_7, r_8, r_9, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{12}, r_{13}, \\ &\quad r_{14}, r_{15}, r_{16}, r_{17}, r_{16}, r_{17}, r_{18}, r_{19}, r_{20}, r_{21}, r_{20}, r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{24}, \\ &\quad r_{25}, r_{26}, r_{27}, r_{28}, r_{29}, r_{28}, r_{29}, r_{30}, r_{31}, r_{32}, r_1) \end{aligned} \tag{3.1}$$

a k výsledku je přičten klíč K_i modulo 2 (Toto sčítání je popsáno v části 7.1). Výstup $E(R_i) \oplus K_i$ je rozdělen do 8 částí po 6 bitech, které potom prochází přes S-boxy $S_1, \ldots, S_8$. Tyto S-boxy jsou 6×4-bitové, což znamená, že výstup těchto S-boxů je $4 \times 8 = 32$-bitový. Tyto boxy se většinou zadávají tak, že 1. a 6. bit každé z osmi vstupních částí vybírá, jeden ze 4 možných 4×4-bitových boxů (odpovídá jednomu řádku v S-boxu). Vstupní i výstupní 4-bitová hodnota se pro jednoduchost zapisuje dekadicky jako 0 až 15 (viz. obr.1). Výstup S-boxů je upraven permutací P. Vznikne 32-bitové slovo, které je značeno $f(R_i, K_i)$, poněvadž je vytvořeno z R_i a je funkcí klíče K_i. Poslední operace v každém kroku je

$$L_{i+1} = R_i \qquad R_{i+1} = L_i \oplus f(R_i, K_i) \tag{3.2}$$

Po skončení 16.kroku proběhne záměna L_{16} a R_{16} a blok o 64 bitech je permutován závěrečnou permutací IP^{-1} (inverzní k IP) na výsledný šifrový text C.

		(x_2, x_3, x_4, x_5)															
x_1	x_6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1	0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1	1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Obr.1 Příklad S-boxu (S-box S1)

Dešifrování probíhá stejným způsobem jako šifrování, aby nemuselo být použito zcela jiné hardwarové schéma. Pouze pořadí výběru klíčů K_i je obrácené. Vytváříme-li klíče postupně, pak musíme výše popsané posunutí provádět doprava místo doleva a $p_i = 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1$. Vše ostatní zůstane stejné.

3.2 Vlastnosti DES

Počáteční a koncové permutace IP a IP^{-1} jsou kryptograficky zanedbatelné. Ta první se používá pouze k rozprostření vlivu bitů z jedněch bytů otevřeného textu M do ostatních. Permutace IP^{-1} zase napravuje vliv permutace IP. Závěrečná výměna slov L a R je zde taky pouze kvůli tomu, aby dešifrovací proces byl shodný jako šifrovací.
Základem algoritmu je transformace f. Skládá se ze substituce (S-boxy), permutace P a přitom zajišťuje vliv klíče na šifrování textu. Klíč je přičítán k proměnné R v modulu 2 jako heslo u proudových šifer, permutace P připomíná transpoziční systémy a S-Boxy substituční systémy. DES je jejich součinová šifra.
DES je celkově substitučním systémem, ve kterém se pracuje se slovy délky 64 bitů. Je to tedy kódová kniha o 2^{64} kódových výrazech. Bez znalosti klíče by mělo jít o neřešitelnou úlohu najít souvislost mezi klíčem K, otevřeným textem M a šifrovým textem C.
Jednou z vlastností DESu je také vliv změny jednoho bitu v otevřeném textu M (respektivě klíči K), na změnu každého bitu šifrového textu C. Pravděpodobnost této změny šifrového textu by měla být asi jedna polovina. To nám zajistí, aby dvě velmi podobné zprávy měli úplně jinou zašifrovanou podobu.
Dalšími vlastnostmi, které požadujeme, jsou konfúze a difúze. Jde o to, aby každý bit klíče K a otevřeného textu M měl komplikovaný vliv na každý bit šifrového textu. Složitost nejvíce ovlivňují S-boxy. Každý výstupní bit S-boxu je nelineární funkcí všech vstupních bitů. Tato funkce musí být v každém případě nelineární, protože jinak bychom mohli schéma okamžitě rozluštit prostým vyřešením soustavy lineárních rovnic. Nelineárním vlastnostem DESu se věnovala velká pozornost, poněvadž zajišťují požadovanou konfúzi, ale bohužel kritéria tvorby S-boxů jsou doposud tajná, přestože právě zde byla objevena spousta slabin.

3.3 Módy DES

DES pracuje na základě 4 módů, které vznikly podle různých potřeb. vycházíme z vlastního blokového algoritmu, tj. ze zobrazení $E_K : X \to X$ kde X je množina všech 64-bitových bloků. Počet prvků množiny X je 2^{64} ($X = \{0,1\}^{\{1,2,\ldots,64\}}$). Tento algoritmus je zároveň prvním módem.

ECB – (Electronic Code Book) Elektronická kódová kniha, která každý blok šifruje

zvlášť. Píšeme klasicky

$$C = E_K(M),$$

kde M je otevřený text a C je šifrový text. Když budeme stejné bloky otevřeného textu opakovat, jejich zašifrovaná podoba bude též stejná. To způsobuje, že bez problému můžeme zfalšovat zprávy pouhým zopakováním šifrového bloku (tímto způsobem např. z částky 1 000 Kč jednoduše uděláme částku 1 000 000 Kč, aniž bychom znali klíč). Proto se tento mód nepoužívá k šifrování zpráv, ale jen k šifrování klíčů.

CBC – (Cipher Block Chaining) Zřetězení šifrového textu, zapisujeme

$$C_n = E_K(M_n \oplus C_{n-1}),$$

kde M_n resp. C_n je n-tý blok otevřeného resp. šifrového textu. Tento mód používá výstup jednoho kroku šifrování k zašifrování následujícího bloku.

CFB – (Cipher Feedback) Zpětná vazba ze šifrového textu, píšeme

$$C_n = M_n \oplus E_K(I_n).$$

I_n je vstupní registr, který je posunutý o k bitů doleva a zprava zase k bity C_{n-1} doplněný. M_n je proud k-bitových znaků otevřeného textu. Tento mód se používá, když je potřeba zpracovat data po znacích nebo po částech menších než 64 bitů. Zpětná vazba je vedena ze šifrového textu, ale jen v délce k bitů, přičemž u výstupu z blokového algoritmu je vyžíváno také jen k bitů. Tento systém zachovává vlastnost závislosti šifrového textu na předchozím otevřeném textu i předchozím šifrovém textu.

0FB – (Output Feedback) Zpětná vazba z výstupu, symbolicky zapsáno jako

$$H_n = E_K(J_n), \qquad C_n = M_n \oplus H_n.$$

J_n je vstupní registr, který je o k bitů posunutý doleva a zprava doplněný k bity H_{n-1}. M_n je opět proud k-bitových znaků otevřeného textu a H_n je vytvořený proud hesla. Je to mód vhodný tam, kde je nežádoucí, aby se chyby vzniklé na komunikačním kanálu rozšiřovaly působením šifrového algoritmu do otevřeného textu. Jde například o přenosy dat s vysokou rychlostí a redundancí.

U módů CBC, CBF a OFB je nezbytné vstupní registr na začátku naplnit nějakou hodnotou. Ta se nazývá inicializační vektor a odesílá se většinou na začátku zprávy.

4 Nedostatky šifry DES a její prolomení

4.1 Komplementárnost

Vlastnost komplementárnosti je dána vztahem

$$C = E_K(M) \qquad \text{implikuje} \qquad nonC = E_{nonK}(nonM), \tag{4.1}$$

kde *non* značí negaci bit po bitu. To je pravidelnost, která by se rozhodně neměla objevovat. Je umožněna tím, že negace klíče i vstupu se při jejich součtu mod 2 ve funkci f zruší:

$$f(R, K) = f(nonR, nonK) \tag{4.2}$$

Totiž označíme-li $R'_0 = nonR_0$, $L'_0 = nonL_0$, a šifrujeme-li zprávu $M' = (L'_0, R'_0)$ klíčem $K' = nonK$ obdržíme postupně: $M' = nonM$, platí-li, že $(L'_i, R'_i) = non(L_i, R_i)$, je i $(L'_{i+1}, R'_{i+1}) = non(L_{i+1}, R_{i+1})$, protože

$$\begin{aligned}(L'_{i+1}, R'_{i+1}) &= (R'_i, L'_i \oplus f(R'_i, K'_i)) \\ &= (nonR_i, (nonL_i) \oplus f(R_i, K_i)) = (nonR_i, nonR_{i+1}) \\ &= non(L_{i+1}, R_{i+1})\end{aligned} \tag{4.3}$$

Tedy $(L'_{16}, R'_{16}) = non(L_{16}, R_{16})$ tj. $nonE_K(M) = E_{nonK}(nonM)$. To se dá pak využít i k luštění v případě, že hledáme klíč K a máme k dispozici dvojice (M, C_1) a $(nonM, C_2)$, které vznikly při šifrování tímto neznámým klíčem. Proveďme zašifrování $E_K(M)$. Není-li tento výraz roven C_1, vylučujeme klíč K. Kdyby byl při šifrování *non* M použit klíč $nonK$, obdrželi bychom

$$C_2 = E_{nonK}(nonM) = nonE_K(M) \tag{4.4}$$

Nejsou-li si oba krajní výrazy, které máme k dispozici, rovny, můžeme vyloučit i klíč *non* K. Tím jsme nahradili jedno šifrování (klíčem $nonK$) za porovnávání výrazů, což je proti šifrování časově nesrovnatelně rychlejší. Prostor klíčů je tedy poloviční a hledání trvá polovinu času.

4.2 Nevhodný návrh S-boxů

Dalším problémem šifry DES je velká korelovanost a nedostatečná nelinearitu výstupních bitů S-boxů. Například box S4 má pětasedmdesátiprocentní redundanci.

S-box S4				(x_1, x_6)															
(x_2, x_3, x_4, x_5)				00				01				10				11			
0	0	0	0	0	1	1	1	1	1	0	1	1	0	1	0	0	0	1	1
0	0	0	1	1	1	0	1	1	0	0	0	0	1	1	0	1	1	1	1
0	0	1	0	1	1	1	0	1	0	1	1	1	0	0	1	0	0	0	0
0	0	1	1	0	0	1	1	0	1	0	1	0	0	0	0	0	1	1	0
0	1	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1	0
0	1	0	1	0	1	1	0	1	1	1	1	1	0	1	1	0	0	0	0
0	1	1	0	1	0	0	1	0	0	0	0	0	1	1	1	1	1	0	1
0	1	1	1	1	0	1	0	0	0	1	1	1	1	0	1	1	0	0	1
1	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0	1
1	0	0	1	0	0	1	0	0	1	1	1	0	0	0	1	0	1	0	0
1	0	1	0	1	0	0	0	0	0	1	0	0	0	1	1	0	1	0	1
1	0	0	1	0	1	0	1	1	1	0	0	1	1	1	0	1	0	1	1
1	1	0	0	1	0	1	1	0	0	0	1	0	1	0	1	1	1	0	0
1	1	0	1	1	1	0	0	1	0	1	0	0	0	1	0	0	1	1	1
1	1	1	0	0	1	0	0	1	1	1	0	1	0	0	0	0	0	1	0
1	1	1	1	1	1	1	1	1	0	0	1	0	1	0	1	1	1	1	0

Jeho 4 boxy 4×4 (při hodnotách krajních bitů $x_1, x_6 = 00, 01, 10, 11$) jsou jednoduše odvoditelné jen od prvního z nich (00). Platí dokonce vztah

$$S4(x \oplus 000001) = (12)(34)S4(x) \oplus (x_6, nonx_6, nonx_6, x_6), \tag{4.5}$$

kde x = $(x_1, x_2, x_3, x_4, x_5, x_6)$ je vstup a symbolický zápis (1,2) znamená výměnu 1. a 2. bitu. Označíme-li $y = (y_1, y_2, y_3, y_4)$ jako výstup, potom součet $(y_1 \oplus y_2)$ je na x_6 závislý pouze lineárně a $(y_1 \oplus y_2 \oplus y_3 \oplus y_4)$ na něm nezávisí vůbec. Taková pravděpodobnost je nežádoucí.

4.3 Slabé a poloslabé klíče

Slabosti standardu se objevily též při studiu klíčů. Budou-li oba registry, používané při tvorbě pracovních klíčů, na počátku obsahovat konstantní bity (0 nebo 1), pak je operace SHL a PC2 nijak nezmění a klíče K_i si budou rovny. Tím nastane i nežádoucí rovnost $E_K = D_K$. Celkem existují 4 tyto klíče (podle toho, zda jeden z registrů obsahuje nuly nebo jedničky). Podobnou vlastnost má 6 dvojic tzv. poloslabých klíčů K_1 a K_2, pro něž platí

$$E_{K1}E_{K2} = Id \qquad neboli \qquad E_{K1} = D_{K2}. \tag{4.6}$$

Takové klíče jsou shodné, ale pořadí jejich použití je opačné. Tak se nám objeví efekt, že při šifrování druhým klíčem probíhá postupně dešifrování klíčem prvním.

Příklad:

$$(01FE01FE01FE01FE, FE01FE01FE01FE01) \tag{4.7}$$

4.4 Příliš krátký klíč

Všechny předešlé nedostatky jsou závažné, ale příčinou pádu standardu DES se nakonec stala právě délka šifrovacího klíče. To, co nezvládli kryptologové svými analytickými útoky, docílil rozvoj výpočetní techniky. Vyluštit šifrový text tzv. hrubou silou znamená, že odzkoušíme všechny možné klíče. Právě malý objem klíče - 56 bitů se stal pro DES osudným. Již v roce 1975 Diffie a Hellman ze Stanfordské univerzity uvažovali, že by pomocí tehdejší technologie byli schopni sestrojit stroj na dešifrování DES algoritmu, který by stál 20 000 000 dolarů. Tento stroj by vyzkoušel cca 10^{17} klíčů za den. Nicméně dokud takový stroj nebyl na světě, stále bylo slyšet argumenty, že to a)není možné, b)bylo by to příliš drahé, c)stroj by se musel přehřívat, d)nikdo nebude investovat miliony dolarů do nečeho tak nejistého atd. Dnes je tento stroj na světě. V roce 1995 se na veřejnost dostává informace, že NSA (National Security Agency) vlastní stroj, který je schopen DES rozluštit do 5 minut. Toto zařízení sestrojila firma The Harris Corporation. Pro ty, kteří ještě stále pochybovali, bylo komerčně sestrojeno a předvedeno speciální zařízení DES-cracker(1998), které je schopno rozluštit všech 2^{56} klíčů do 9 dnů a nalézt tak příslušné řešení. DES musel být nahrazen jiným standardem. Prozatímně jej NIST (National Institute of Standards and Technology) nahradil implementací TripleDES. V podstatě se jedná o opakované použití algoritmu DES. Kryptologické veřejnosti však bylo jasné, že řešení není optimální (především pro nižší rychlost), a proto v roce 1997 NIST vypisuje veřejnou soutěž na vytvoření nového komerčního standardu pro symetrické šifrování.

5 Nový standard AES

Pro název tohoto nového algoritmu se vžilo označení AES (Advanced Encryption Standard). Vybraný standard měl být velice flexibilní, lehce implementovatelný, měl pracovat s 32-bitovým mikroprocesorem, 64-bitovým procesorem, ale i 8-bitovým (v tzv. režimu smart card). V červnu 1998, kdy byla stanovena uzávěrka pro podání návrhu, bylo celkem předloženo 15 kandidátů. Z nich bylo v květnu následujícího roku do dalšího kola vybráno pět: MARS, RC6, Rijndael, Serpent a Twofish. V říjnu 2000 byl vybrán vítěz a 26.11. roku 2001 byl vyhlášen nový šifrový symetrický standard pro šifrování senzitivních informací s platností od 26.5.2002. Tímto výtězem se stal algoritmus Rijndael, navržený týmem belgických kryptologů - Vincentem Rijnmenem a Joanem Deamenem. Rijndealův algoritmus je symetrická bloková šifra, která umí zpracovat datové bloky o 128 bitech.

Používá šifrovací klíče s délkami 128, 192 a 256 bitů. Rijndeal je navržený tak, aby používal pouze celobytové operace.
AES se na první pohled diametrálně liší od DES, ale opak je pravdou. AES ve skutečnosti vychází z těch teoretických principů, které byly použity u DES a za celých 25 let existence DES nikdy nebyly zpochybněny. Přejděme už ale k přesnému popisu nového standardu.

6 Označení a konvence AES

6.1 Vstupy a výstupy

Každý vstup i výstup AES algoritmu se skládá z posloupnosti 128 bitů. Tato posloupnost bude nazývána bloky stejně jako u standardu DES a počet bitů, které obsahuje, bude nazýván jejich délkou. Šifrovací klíč AES algoritmu je posloupnost 128, 192 nebo 256 bitů. Vstup, výstup nebo klíč jiné délky není tímto standardem povolen.
Bity v takovýchto posloupnostech budou očíslovány od nuly do čísla o jedno menšího než je délka posloupnosti (bloku nebo klíče). Číslo i odpovídající danému bitu je známo jako index a bude v rozsahu $0 \leq i < 128$, $0 \leq i < 192$, $0 \leq i < 256$.

6.2 Byty

Základní jednotka v AES algoritmu je byte, sekvence 8 bitů braná jako samostatná jednotka. Vstup, výstup nebo šifrovací klíč jsou zpracovávány jako pole bytů, které je vytvořeno rozdělením této posloupnosti do skupin po osmi bitech, které tvoří toto pole bytů (viz.6.3). Pro vstup, výstup nebo klíč značený a budou byty ve výsledném poli značeny a_n nebo $a[n]$, kde n je v jednom z následujících rozsahů:

$$\begin{aligned} \text{Délka klíče} = 128 \text{ bitů}, &\quad 0 \leq i < 16 \\ \text{Délka klíče} = 192 \text{ bitů}, &\quad 0 \leq i < 24 \\ \text{Délka klíče} = 256 \text{ bitů}, &\quad 0 \leq i < 32 \\ \text{Délka bloku} = 128 \text{ bitů}, &\quad 0 \leq i < 16 \end{aligned}$$

Všechny hodnoty bytů v AES algoritmu budou prezentovány jako zřetězení jednotlivých hodnot bitů (0 nebo 1) v závorkách v pořadí $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. Tyto byty jsou interpretovány jako konečné prvky pole používaje polynomiální reprezentace:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0 = \sum_{i=0}^{7} b_i x^i \qquad (6.1)$$

Například, {0100011} identifikuje určitý konečný prvek pole $x^6 + x^5 + x + 1$.
To také vede k označení hodnot bytů pomocí hexadecimální notace. Každá ze dvou skupin čtyř bitů je označena jedním symbolem jako na obr.2.

Bitový vzor	symbol
0000	0
0001	1
0010	2
0011	3

Bitový vzor	symbol
0100	4
0101	5
0110	6
0111	7

Bitový vzor	symbol
1000	8
1001	9
1010	a
1011	b

Bitový vzor	symbol
1100	c
1101	d
1110	e
1111	f

Obr.2 Hexadecimální reprezentace bitových vzorů

Proto {01100011} může být reprezentováno jako {63}, kde symbol označující 4-bitovou skupinu obsahující vyšší očíslované bity je vlevo.
Některé konečné operace pole zapojí jeden součtový bit (b_8). Tento extra bit je reprezentován '{01}' a přímo předchází 8-bitovému bytu, například 9-bitová posloupnost bude reprezentována jako {01}{1b}.

6.3 Pole bytů

Pole bytů bude reprezentován následujícím způsobem:

$$a_0 a_1 a_2 \dots a_{15}$$

Byty a bitové řazení je odvozeno od 128-bitové vstupní sekvence

$$input_0 input_1 input_2 \dots input_{127}$$

následovně

$$\begin{aligned} a_0 &= \{input_0, input_1 \dots, input_7\} \\ a_1 &= \{input_8, input_9 \dots, input_{15}\} \\ &\vdots \\ a_{15} &= \{input_{120}, input_{121} \dots, input_{127}\} \end{aligned}$$

Vzor může být roztažen na delší posloupnosti (např. 192-bitové a 256-bitové klíče), zapíšeme-li to tedy obecně

$$a_n = \{input_{8n}, input_{8n+1}, \dots, input_{8n+7}\} \tag{6.2}$$

Obr.3 nám ukáže, jak jsou bity v každém bytu číslovány.

Vstupní bitová sekvence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
Číslo bytu	0								1								2								...
Čísla bitů v bytu	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	

Obr.3 Indexy pro byty a bity.

6.4 Stav

Interně jsou operace AES algoritmu prováděny na 2-dimenzionálním poli bytů, které se nazývá **stav**. Stav se skládá ze 4 řádků bytů obsahujících Nb bytů, kdeNb je velikost bloku dělitelná 32. Ve stavovém poli označeném symbolem s má každý jednotlivý byte 2 indexy: číslo řádku r v rozmezí $0 \leq r < 4$ a číslo sloupce c v rozmezí $0 \leq c < Nb$. To umožňuje jednotlivému bytu stavu, aby byl značen buď $s_{c,r}$ nebo $s[c, r]$.
Na začátku šifrovacího i dešifrovacího algoritmu je vstup - pole bytů $in_0, in_1, \ldots, in_{15}$ - kopírován do stavu, jak je nakresleno na obr.4. Šifrovací a dešifrovací operace jsou poté prováděny v tomto stavovém poli a nakonec jejich konečná hodnota je kopírována na výstup - pole bytů $out_0, out_1, \ldots, out_{15}$.

Vstupní byty			
in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

$\rightarrow$

Stavové pole			
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$\rightarrow$

Výstupní byty			
out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

Obr.4 Stavové pole, vstup a výstup

Proto na začátku šifrování nebo dešifrování je vstupní pole in kopírováno do stavového pole podle následujícího schématu:

$$s[c, r] = in[r + 4c] \quad \text{pro} \quad 0 \leq r < 4 \text{ a } 0 \leq c < Nb \tag{6.3}$$

6.5 Stav jako pole sloupců

Čtyři byty v každém sloupci stavového pole tvoří 32-bitová slova, kde číslo řádku r stanovuje index pro 4 byty v každém slově. Stav proto může být interpretován jako jednodimenzionální pole 32-bitových slov (sloupců), $w_0 \ldots w_3$, kde číslo sloupce c stanovuje index v tomto poli. Proto například v obr.4 může být stav považován za pole 4 slov následovně:

$$\begin{aligned} w_0 &= s_{0,0}s_{1,0}s_{2,0}s_{3,0} \quad & w_1 &= s_{0,2}s_{1,2}s_{2,2}s_{3,2} \\ w_1 &= s_{0,1}s_{1,1}s_{2,1}s_{3,1} \quad & w_2 &= s_{0,3}s_{1,3}s_{2,3}s_{3,3} \end{aligned} \tag{6.4}$$

7 Matematické operace potřebné pro AES

Všechny byty v AES algoritmu jsou interpretovány jako konečné pole prvků s použitím notace zavedené v 6.2. Následující paragrafy zavádějí základní matematické pojmy potřebné pro pochopení algoritmu.

7.1 Sčítání

Sčítání dvou prvků v konečném poli je prováděno sečtením koeficientů pro příslušné mocniny polynomů v těchto 2 prvcích. Sčítání je prováděno XOR operací (značenou $\oplus$) - např. modulo 2 - tedy $1 \oplus 1 = 0, 1 \oplus 0 = 1$ a $0 \oplus 0 = 0$. Proto odčítání polynomů je identické jako sčítání polynomů.
Alternativně může být sčítání prvků konečného pole popsáno jako sčítání modulo 2 odpovídající bitům v bytu. Pro 2 byty $\{a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0\}$ a $\{b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0\}$ je součet $\{c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0\}$, kde každé $c_i = a_i \oplus b_i$.
Například, následující výrazy jsou ekvivalentní jeden druhému.

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \quad \text{(polynomický zápis)}$$
$$\{01010111\} \oplus \{10000011\} = \{11010100\}) \quad \text{(binární zápis)}$$
$$\{57\} \oplus \{83\} = \{d4\} \quad \text{(hexadecimální zápis)}$$

7.2 Násobení

V polynomické reprezentaci odpovídá násobení v $\mathrm{GF}(2^8)$ (značené $\bullet$) násobení polynomů modulo nějaký ireducibilní polynom stupně 8. Polynom je ireducibilní, jestliže je nekonstantní a jeho jediní dělitelé jsou jednička a on sám. Pro AES algoritmus je tento ireducibilní polynom

$$m(x) = x^8 + x^4 + x^3 + x + 1 \tag{7.1}$$

nebo $\{01\}\{1b\}$ v hexadecimální notaci.
Například, $\{57\} \bullet \{83\} = \{c1\}$, protože

$$\begin{aligned}(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = {} & x^{13} + x^{11} + x^9 + x^8 + x^7 + \\ & x^7 + x^5 + x^3 + x^2 + x + \\ & x^6 + x^4 + x^2 + x + 1 \\ = {} & x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1\end{aligned}$$

a

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \quad \text{modulo} \quad (x^8 + x^4 + x^3 + x + 1) = x^7 + x^6 + 1$$

Modulová redukce polynomem $m(x)$ zaručí, že výsledek bude binární polynom stupně menšího než 8 a tedy může být reprezentován jako byte. Naproti tomu u sčítání neexistuje jednoduchá operace na bytové úrovni, která odpovídá tomuto násobení.

Násobení definované výše je asociativní a prvek $\{01\}$ je jednotkovým prvkem.

Pro jakýkoli nenulový binární polynom $b(x)$ stupně menšího než 8 může být nalezena násobná inverze (značená $b^{-1}(x)$) pomocí rozšířeného euklidova algoritmu, který počítá polynomy $a(x)$ a $c(x)$takové,že

$$b(x)a(x) + m(x)c(x) = 1 \tag{7.2}$$

odtud $a(x) \bullet b(x) \mod m(x) = 1$, což znamená

$$b^{-1}(x) = a(x) \mod m(x) \tag{7.3}$$

Nadto pro jakékoli $a(x), b(x)$ a $c(x)$ v poli platí

$$a(x) \bullet (b(x) + c(x)) = a(x) \bullet b(x) + a(x) \bullet c(x)$$

Z toho plyne, že množina 256 možných hodnot bytu s operací XOR použitou jako sčítání a s násobením již definovaným, má strukturu konečného pole GF(2^8).

7.2.1 Násobení proměnnou x

Násobení binárního polynomu definovaného v rovnici (6.1) polynomickým x dává výsledek

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \tag{7.4}$$

Výsledek $x \bullet b(x)$ dostaneme zkrácením předešlého výsledku modulo m(x), definovaného v rovnici (7.1). Je-li $b_7 = 1$, krácení je ekvivalentní odčítání (např. XOR operací) polynomu $m(x)$. Je patrno, že násobení proměnnou x (např. {00000010} nebo {02}) může být implementován na úrovni bytu levým posunem a následovnou podmíněnou bitovou XOR operací s {1b}. Tato operace na bytech je značena *xtime()*. Násobení vyššími mocninami x může být implementováno opakováným použitím *xtime()*. Přidáním mezivýsledků můžeme implementovat násobení nějakou konstantou.

Například, $\{57\} \bullet \{13\} = \{fe\}$ protože

$$\begin{aligned}
\{57\} \bullet \{02\} &= xtime(\{57\}) = \{ae\} \\
\{57\} \bullet \{04\} &= xtime(\{ae\}) = \{47\} \\
\{57\} \bullet \{08\} &= xtime(\{47\}) = \{8e\} \\
\{57\} \bullet \{10\} &= xtime(\{8e\}) = \{07\}
\end{aligned}$$

tedy,

$$\begin{aligned}
\{57\} \bullet \{13\} &= \{57\} \bullet (\{01\} \oplus \{02\} \oplus \{10\}) \\
&= \{57\} \oplus \{ae\} \oplus \{07\} \\
&= \{fe\}.
\end{aligned}$$

7.3 Polynomy s koeficienty v GF(2^8)

4-členné polynomy mohou být definovány - s koeficienty, které jsou prvky konečného pole - takto:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \tag{7.5}$$

Budeme ho označovat jako slovo ve tvaru $[a_0, a_1, a_2, a_3]$. Uvědomme si, že polynomy v této části se chovají trochu jinak, než polynomy používané v definici konečného pole prvků, i když oba typy polynomů používají stejnou neznámou x. Koeficienty v této části jsou samy členy konečného pole, např. bytů místo bitů; také násobení 4-členými polynomy používá jiné polynomické krácení, definované dále. Rozdíl můžeme vždy poznat z kontextu.
Pro ilustraci sčítání a násobení nechť

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0 \tag{7.6}$$

definuje druhý 4-členný polynom. Sčítání se provádí sečtením koeficientů konečného pole stejné mocniny x. Toto sčítání odpovídá XOR operaci mezi odpovídajícími byty v každém slově - v ostatních slovech XOR operaci kompletních hodnot slov.
Použijeme tedy rovnice (7.5) a (7.6):

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0) \tag{7.7}$$

Násobení je docíleno ve dvou krocích. V prvním je součin polynomů $c(x) = a(x) \bullet b(x)$ algebraicky rozšířen, stejné mocniny jsou seskupeny a dají nám

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \tag{7.8}$$

kde

$$\begin{aligned}
c_0 &= a_0 \bullet b_0 \\
c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \\
c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \\
c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \\
c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\
c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\
c_6 &= a_3 \bullet b_3
\end{aligned} \tag{7.9}$$

Výsledek $c(x)$ nereprezentuje 4-bytové slovo. Proto tedy druhý krok násobení je zkrátit $c(x)$ modulo polynom stupně 4; výsledek však může být krácen i polynomem nižšího stupně. Pro AES algoritmus je toto prováděno polynomem $x^4 + 1$, aby

$$x^i \mod (x^4 + 1) = x^{i \bmod 4} \tag{7.10}$$

Modulový součin $a(x)$ a $b(x)$, značený $a(x) \otimes b(x)$, je daný 4-členným polynomem $d(x)$ definovaným následovně:

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 \tag{7.11}$$

s

$$\begin{aligned} d_0 &= (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \\ d_1 &= (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \\ d_2 &= (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \\ d_3 &= (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3) \end{aligned} \tag{7.12}$$

Když *a(x)* je fixní polynom, operace popsaná v rovnici (7.11) může být napsaná v maticové formě jako:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{7.13}$$

Protože $x^4 + 1$ není ireducibilní polynom nad GF(2^8), násobení pevným 4-členným polynomem není nezbytně invertibilní. Ačkoliv AES algoritmus specifikuje pevný 4-člený polynom, k tomuto polynomu inverze existuje (viz.8.1.3 a 8.3.3):

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{7.14}$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{7.15}$$

Jiný polynom používaný v AES algoritmu (viz funkce $RotWord()$ v 8.2) má $a_0 = a_1 = a_2 = \{00\}$ a $a_3 = \{01\}$, tedy je to polynom x^3. Zkoumání rovnice (7.13) nám ukáže, že jeho účel je vytvoření výstupního slova rotací bytů ve vstupním slově. To znamená, že $[b_0, b_1, b_2, b_3]$ je transformováno na $[b_1, b_2, b_3, b_0]$

8 Specifikace algoritmu AES

Pro AES algoritmus je velikost vstupního bloku, výstupního bloku a stavu 128 bitů. To je reprezentováno $Nb = 4$, což odráží číslo 32-bitovéhoslova (počet sloupců) ve stavu. Pro AES algoritmus, délka klíče K je 128, 192 nebo 256 bitů. Délka klíče je representována $Nk = 4$, 6 nebo 8, což odráží číslo 32-bitového slova (počet sloupců) v klíči.
Počet cyklů, které provádí AES algoritmus během realizace algoritmu, záleží na délce

klíče. Počet cyklů je reprezentován Nr, kde

$$\begin{aligned} Nr = 10 \quad & \text{když} \quad Nk = 4 \\ Nr = 12 \quad & \text{když} \quad Nk = 6 \\ Nr = 14 \quad & \text{když} \quad Nk = 8 \end{aligned}$$

Jediné kombinace klíč-blok-cyklus, která se řídí tímto standardem jsou dány v obr.5.

	Délka klíče (Nk slov)	**Velikost bloku (Nb slov)**	**Počet cyklů (Nr)**
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Obr.5 Kombinace klíč-blok-cyklus

Pro šifrování i dešifrování používají AES algoritmus rundovní funkce, které jsou složeny ze 4 odlišných bytově-orientovaných transformací:
1) bytová substituce používá substituční tabulku (S-box)
2) posouvání řádku stavového pole různými ofsety
3) míchání dat v každém sloupci stavového pole
4) přidání náhodného klíče do stavu.
Tyto transformace (a jejich inverze) budou popsány dále.

8.1 Šifrování

Na začátku šifrování je vstup kopírován do stavového pole. Po počátečním přičtení náhodného klíče je stavové pole transformováno implementací rundovní funkce 10, 12 nebo 14krát (podle délky klíče) s posledním cyklem lišícím se jen nepatrně od předešlých $Nr - 1$ cyklů. Konečný stav je potom kopírován na výstup.
Zaokrouhlovací funkce je parametrizována použitím klíčové tabulky, která je složena z 1-rozměrného pole 4-bytových slov odvozených pomocí procesu expanze klíče popsaného v paragrafu 8.2
Šifra je popsána v pseudokódu na obr.6. Jednotlivé transformace - SubBytes(), shiftRows(), MixColumns() a AddRoundKey() - které zpracovává stav, jsou popsány v následujících paragrafech. V obr.6 pole $w[]$ se rovná klíčové tabulce, která je popsána v 8.2.
Všech Nr cyklů je stejných s vyjímkou posledního cyklu, který nezahrnuje transformaci MixColumns().

```
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])    viz.5.1.4

  for round = 1 step 1 to Nr-1
    SubBytes(state)     viz.5.1.1
    shiftRows(state)    viz.5.1.1
    MixColumns(state)   viz.5.1.1
    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
```

Obr.6 Pseudokód pro šifrování

8.1.1 Transformace SubBytes()

Transformace SubBytes() je nelineární bytová substituce, která pracuje nezávisle na každém bytu stavu používaje substituční tabulku (S-box). Tento S-box (obr.7), který je invertibilní, je sestrojen složením 2 transformací:

1. Vezměte násobnou inverzi v konečném poli GF(2^8), popsanou v paragrafu 7.2., prvek {00} je značen sám na sebe.

2. Aplikujte následující afinní transformaci (nad GF(2)):

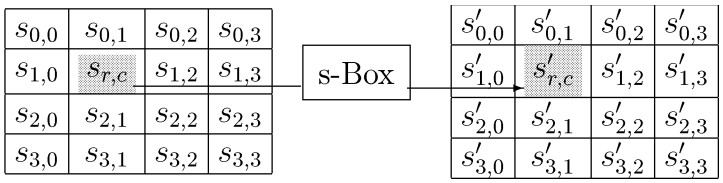
$$b'_i = b_i \oplus b_{(\mathrm{i+4}) \bmod 8} \oplus b_{(\mathrm{i+5}) \bmod 8} \oplus b_{(\mathrm{i+6}) \bmod 8} \oplus b_{(\mathrm{i+7}) \bmod 8} \oplus c_i \qquad (8.1)$$

   pro $0 \leq i < 8$, kde b_i je i-tý bit v bytu a c_i je i-tý bit bytu c s hodnotou {63} neboli {01100011}. Zde i kdekoli jinde bude čárka nad proměnnou značit, že proměnná je aktualizována hodnotou vpravo.

V maticovém tvaru může být afinní transformace prvku z S-boxu vyjádřena jako:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{8.2}$$

Matice použitá v (8.2) je zřejmě regulární, protože její inverzní podoba je použita v inverzní transformaci InvSubBytes při dešifrování. Obr.7 ilustruje efekt transformace SubBytes() na stav.



Obr.7 SubBytes() aplikuje S-box na každý byte stavu

S-box používaný v transformaci SubBytes() je reprezentován v hexadecimálním tvaru na obr.7. Například je-li $s_{1,1} = \{53\}$, potom substituční hodnota může být určena jako průsečík řádku s indexem '5' a sloupce s indexem '3'. Toto nám dá výsledek $s'_{1,1} = \{ed\}$.

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

Obr.7 S-box: Substituční hodnoty pro byte xy (v hexadecimálním tvaru)

8.1.2 Transformace ShiftRows()

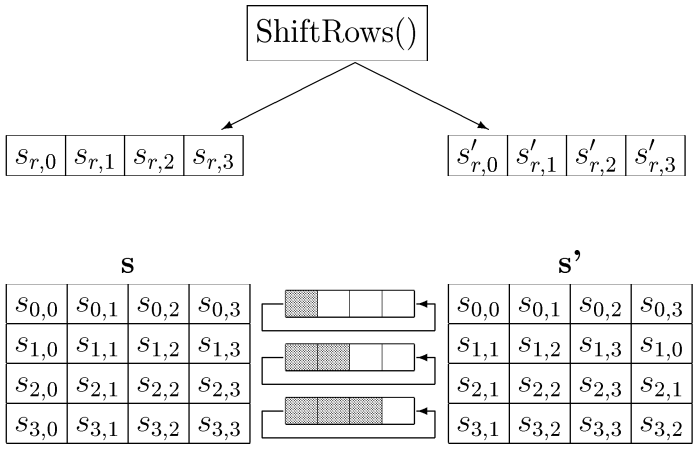
V transformaci ShiftRows() jsou byty v posledních 3 řádcích stavu cyklicky zaměňovány přes odlišné počty bytů (ofsety). První řádek $r = 0$ není zaměňován. Přesněji transformace ShiftRows() postupuje následovně:

$$s'_{r,c} = s_{r,(c+shift(r,Nb))\mathrm{mod}Nb} \quad \text{pro} \quad 0 < r < 4 \quad a \quad 0 \leq Nb \tag{8.3}$$

kde hodnota $shift(r, Nb)$ záleží na čísle řádku r následovně (připomeňme, že $Nb = 4$):

$$shift(1, 4) = 1; \quad shift(2, 4) = 2; \quad shift(3, 4) = 3. \tag{8.4}$$

Toto má efekt pohyblivých bytů do "nižších" pozic v řádků. Nejnižší byty dáme na konec řádku.
Obr.8 ilustruje transformaci ShiftRows().



Obr.8 ShiftRows() cyklicky posouvá poslední 3 řádky stavu

8.1.3 Transformace MixColumns()

Transformace MixColumns() pracuje na stavu sloupec po sloupci, zpracovává každý sloupec jako 4-členný polynom. Sloupce jsou považovány za polynomy nad GF(2^8) a násobeny modulo $x^4 + 1$ pevným polynomem *a(x)*

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{8.5}$$

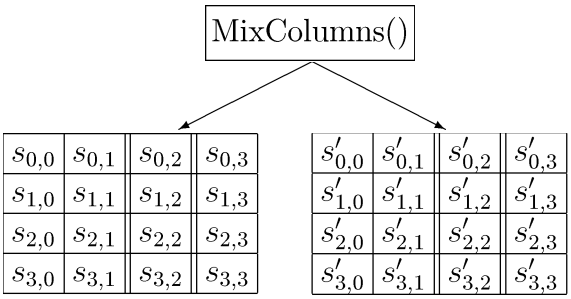
Jak popisuje paragraf 7.3, může to být zapsáno jako maticové násobení. Nechť $s'(x) = a(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{pro } 0 \leq c < Nb \tag{8.6}$$

Jako výsledek tohoto násobení jsou 4 byty ve sloupci přemísťovány následovně:

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

Obr.10 ilustruje transformaci MixColumns().



Obr.10 MixColumns() pracuje na stavu sloupec po sloupci

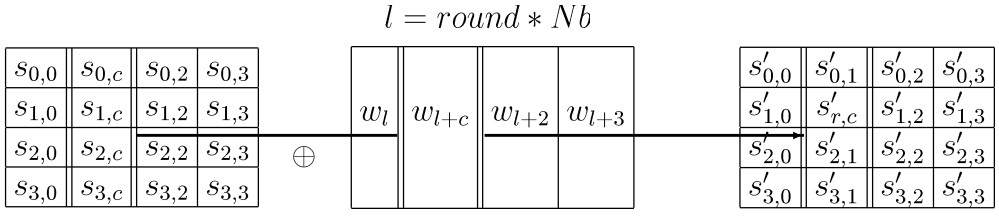
8.1.4 Transformace AddRoundKey()

V transformaci AddRoundKey() je rundovní klíč přidán do stavu jednoduchou bitovou XOR operací. Každý rundovní klíč se skládá z Nb slov z tabulky klíčů (je popsaná v paragrafu 8.2). Všechny tyto Nb slova jsou přidávány do sloupců stavu tak, že

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}] \quad \text{pro } 0 \leq c < Nb \qquad (8.7)$$

$kde[w_i]$ jsou slova tabulky klíčů a $round$ je hodnota v řádku $0 \leq round < Nr$. Při šifrování se provádí počáteční přičtení rundovního klíče (když $round = 0$) ještě před první aplikací rundovní funkce. Aplikace AddRoundKey() transformace na Nr cyklů šifry proběhne, když $1 \leq round \leq Nr$.

Průběh této transformace je ilustrován na obr.11, kde $l = round * Nb$.



Obr.11 AddRoundKey()"XORuje" každý sloupec stavu slovem z tabulky klíčů

8.2 Expanze klíče

AES algoritmus vezme šifrovací klíč K a provede postup expanze klíče pro vygenerování tabulky klíčů. Expanze klíče generuje celkově $Nb(Nr + 1)$ slov. Čtveřice

slov tvoří jeden rundovní klíč K_i, kde $0 \leq i \leq 10$. Začneme klíčem K, roztáhneme ho a dostaneme lineární pole w. Toto pole je délky $Nb(Nr + 1)$. Ukažme si, jak to funguje pro $Nb = 4$ a $Nr = 10$, tedy w má $4 * (10 + 1) = 44$ slov po 32 bitech. Každý blok 4 slov má 128 bitů a je to podklíč K_i. První rundovní klíč je $K_0 = (w_0, w_1, w_2, w_3)$, který je zkopírován ze šifrovacího klíče K. Další podklíč jsou získány z K_0. Pravidlo je následující:

$$\begin{aligned} w_4 &= w_0 \text{ xor } temp_1 \\ w_5 &= w_1 \text{ xor } w_4 \\ w_6 &= w_2 \text{ xor } w_4 \\ w_7 &= w_3 \text{ xor } w_4 \\ w_8 &= w_4 \text{ xor } temp_2 \\ &\vdots \\ w_{43} &= w_{39} \text{ xor } w_{42} \\ w_{44} &= w_{40} \text{ xor } temp_{11} \end{aligned}$$

Podívejme se na případy w_i, kde $(i \mod Nk \neq 0)$. Pravidlo je

$$w_i = w_{i-Nk} \text{ xor } w_{i-1}$$
$$\text{s } Nr = 4.$$

Když $(i \mod Nk = 0)$, pak máme

$$w_i = w_{i-Nk} \text{ xor } temp_k$$

kde $temp_k = SubWord(S_1 w_{i-1} \text{ xor } Rcon[k])$. $SubWord$ je aplikován na w_{i-1} s posunutým bytem a $Rcon[k]$ je definován jako $Rcon[k] = (RC_k, 00, 00, 00)$ s $RC_1 = 1$, $RC_1 = X \times RC_{k-1} = X^{k-1}$ a $RC_k \in GF(2^8)$ for $k = i/4$ a $i = 4, 8, 12, \ldots, 44$.

Ukažme si to na příkladu. Uvažujme klíč

$$K_0 = K = 00\ 01\ 02\ 03\ 04\ 05\ 06\ 07\ 08\ 09\ 0A\ 0B\ 0C\ 0D\ 0E\ 0F$$

Výpočet w_i, když i je násobek 4, dostaneme za použití pravidla

$$w_i = w_{i-4} \text{ xor } SubWord(S_1 w_{i-1}) \text{ xor } Rcon[k],$$

výpočet w_i, když i není násobek 4, dostaneme za použití pravidla

$$w_i = w_{i-4} \text{ xor } w_{i-1}$$

Počítané hodnoty jsou:

$$w_4 = w_0 \text{ xor } SubWord(S_1 w_3)$$

$$
\begin{aligned}
D7 &= 00 \text{ xor } D7 \\
AA &= 01 \text{ xor } AB \\
74 &= 02 \text{ xor } 76 \\
FD &= 03 \text{ xor } FE \\
w_4[0] &= w_4[0] \text{ xor } rcon[1] \\
D6 &= D7 \text{ xor } 02
\end{aligned}
$$

$w_5 = w_1$ xor w_4 $\quad$ $w_6 = w_2$ xor w_5 $\quad$ $w_7 = w_3$ xor w_6

$$
\begin{aligned}
D2 &= 04 \text{ xor } D6 & DA &= 08 \text{ xor } D2 & D6 &= 0C \text{ xor } DA \\
AF &= 05 \text{ xor } AA & A6 &= 09 \text{ xor } AF & AB &= 0D \text{ xor } A6 \\
72 &= 06 \text{ xor } 74 & 78 &= 0A \text{ xor } 72 & 76 &= 0E \text{ xor } 78 \\
FA &= 07 \text{ xor } FD & F1 &= 0B \text{ xor } FA & FE &= 0F \text{ xor } F1
\end{aligned}
$$

Rundovní klíč $K_1 = D6\ AA\ 74\ FD\ D2\ AF\ 72\ FA\ DA\ A6\ 78\ F1\ D6\ AB\ 76\ FE$

Stejným způsobem určíme i další rundovní klíče $K_2, \ldots, K_{10}$

8.3 Dešifrování

Šifrovcí transformace v paragrafu 8.1 mohou být převráceny a poté implementovány v opačném pořadí, aby přímo vytvořily inverzní šifru pro AES algoritmus. Jednotlivé transformace použité při dešifrování - **InvShiftRows()**, **InvSubBytes()**, **InvMixColumns()** a **AddRoundKey()** - zpracovávají stav a jsou popsány v následujících odstavcích.
Dešifrování je popsáno v pseudokódu na obr.12. Pole **w**[] obsahuje tabulku klíčů.

```
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])    viz.5.1.4

  for round = Nr-1 step -1 downto 1
    InvshiftRows(state)    viz.5.3.1
    InvSubBytes(state)     viz.5.3.1
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    InvMixColumns(state)   viz.5.3.1
  end for

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

  out = state
end
```

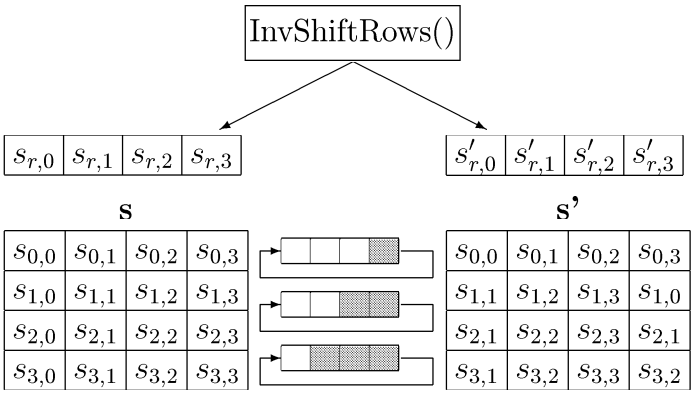
Obr.12 Pseudokód pro dešifrování

8.3.1 Transformace InvShiftRows()

InvShiftRows() je inverze od **ShiftRows()** transformace. Byty v posledních 3 řádcích stavu jsou cyklicky posouvány přes různý počet bytů (ofsety). První řádek $r = 0$ není posouván. Dolní tří řádky jsou cyklicky posouvány o *Nb-shift(r,Nb)* bytů, kde hodnota *shift(r,Nb)* záleží na čísle řádku a je dosazena do rovnice (8.4). Speciálně **InvShiftRows()** transformace postupuje následovně:

$$s'_{r,(c+shift(r,Nb))\mathrm{mod}Nb} = s_{r,c} \quad \text{pro} \quad 0 < r < 4 \quad a \quad 0 \leq c < Nb \tag{8.8}$$

Obr.13 ilustruje **InvShiftRows()** transformaci.



Obr.13 InvShiftRows() cyklicky posouvá poslední 3 řádky stavu

8.3.2 Transformace InvSubBytes()

InvSubBytes() je inverze bytové substituční transformace, ve které inverzní S-box je aplikován na každý byte stavu. Toto dostaneme použitím inverze k afinní transformaci (8.1) s následným nalezením násobné inverze v GF (2^8).
Inverzní S-box používaný v této transformaci je prezentován na obr.14:

X \ Y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

Obr.12 Inverzní S-box: substituční hodnoty pro byte xy

8.3.3 Transformace InvMixColumns()

Transformace **InvMixColumns()** je inverzí k transformaci **MixColumns()**. **InvMixColumns()** pracuje na stavu sloupec po sloupci, zpracovává každý sloupec jako 4-členný polynom. Sloupce jsou považovány za polynomy nad GF(2^8) a násobeny modulo $x^4 + 1$ pevným polynomem $a^{-1}(x)$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{8.9}$$

Jak popisuje paragraf 7.3., může to být zapsáno jako maticové násobení. Nechť $s'(x) = a^{-1}(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0d \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{pro } 0 \leq c < Nb \tag{8.10}$$

Jako výsledek tohoto násobení jsou 4 byty ve sloupci přemísťovány následovně:

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

8.3.4 Inverze AddRoundKey() transformace

AddRoundKey(), která je popsaná v paragrafu 8.1.4, je svojí vlastní inverzí, protože obsahuje pouze aplikaci XOR operace.

8.3.5 Ekvivalentní dešifrování

V přímém dešifrování, popsaném v paragrafu 8.3 a na obr.12, se posloupnost transformací liší od těch šifrovacích, i když podle klíčové tabulky pro šifrování i dešifrování se zdají stejné. Ale některé vlastnosti AES algoritmu připouští ekvivalentní dešifrování, které má stejnou posloupnost transformací jako šifrování (s transformacemi vyměněnými za jejich inverze). To je doprovázeno změnami v tabulce klíčů. Dvě vlastnosti, které berou v úvahu toto ekvivalentní dešifrování, jsou následující:

1. Transformace **SubBytes()** a **ShiftRows()** komutují. To stejné platí pro jejich inverze **InvSubBytes()** a **InvShiftRows()**.

2. Operace míchající sloupce **MixColumns()** a **InvMixColumns()** jsou lineární se zachováním sloupcových vstupů, což znamená

   ```
   InvMixColumns(state XOR Round Key) =
   InvMixColumns (state) XOR InvMixColumns(Round Key).
   ```

Tyto vlastnosti dovolují, aby pořadí **InvSubBytes()** a **InvShiftRows()** transformací bylo opačné. Pořadí **AddRoundKey()** a **InvMixColumns()** může též být opačné, za předpokladu že sloupce (slova) klíčové tabulky pro dešifrování jsou modifikovány použitím **InvMixColumns()** transformace.
Ekvivalentní dešifrování je definované změnou pořadí **AddRoundKey()** a **InvMixColumns()** transformací používaných v rundovní cyklus po první modifikaci klíčové tabulky pro dešifrování pro round = 1 až Nr-1 používaje **InvMixColumns()** transformace. Prvních a posledních Nb slov dešifrovací klíčové tabulky nemůže být modifikováno tímto způsobem
Když uděláme tyto změny, výsledné ekvivalentní dešifrování nabízí účinější strukturu než dešifrování popsané v paragrafu 8.3 a na obr.12. Pseudokód pro ekvivalentní dešifrování je popsán na obr.15.(Pole slov **dw**[] obsahuje modifikovanou dešifrovací tabulku klíčů.)

```
EqInvCipher(byte in[4*Nk], byte out[4*Nr+1)], 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

Pro ekvivalentní dešifrování, následující pseudokód je přidán
na konec procesu expanze klíče(viz.8.2)
  for i = 0 step 1 (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
```

Obr.15 Pseudokód pro ekvivalentní dešifrování

9 Implementace AES

9.1 Požadavky na délku klíče

Implementace AES algoritmu by měla podporovat přinejmenším jednu z délek klíče: 128, 192 nebo 256 bitů (tj. $N_K = 4, 6$ nebo 8). Implementace mohou také

podporovat dvě nebo tři délky klíče, které mohou umožňovat propojení algoritmových implementací.

9.2 Klíčová omezení

Neredukované nebo poloredukované klíče byly určeny pro AES algoritmus a neexistuje zde žádné omezení pro výběr klíče.

9.3 Parametrizace délky klíče, rozměru bloku a počtu cyklů

Tento standard explicitně definuje povolené hodnoty pro délku klíče Nk, rozměr bloku Nb a počet cyklů Nr. Ačkoli budoucí upřesnění tohoto standardu si může vyžádat změny nebo dodatky k povoleným hodnotám daných parametrů.

10 Módy AES

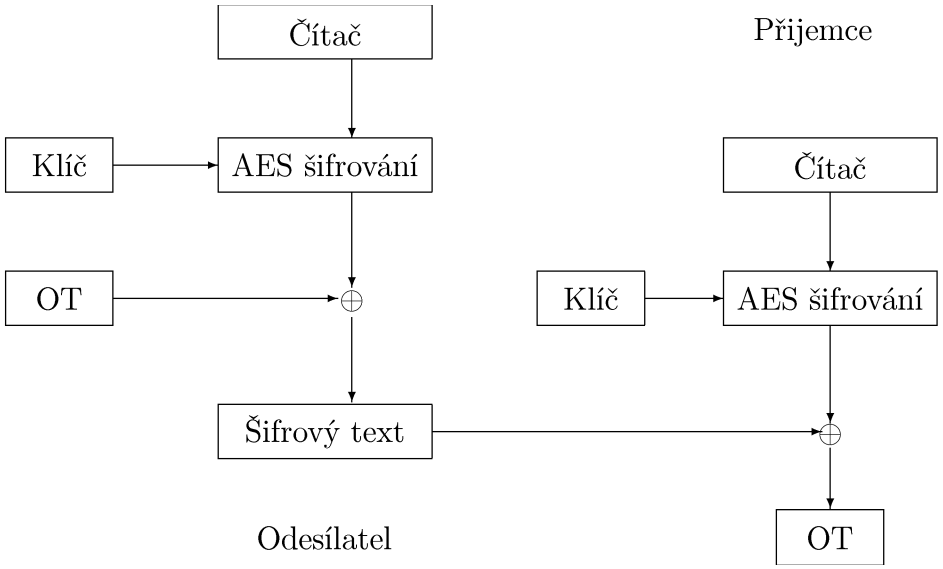
AES používá 4 módy stejné jako u DES: ECB, CBC, CFB a OFB (viz.3.3) s jinou délkou bloku. Pro větší výkonnost byl však u AES zaveden nový mód CTR (Counter=čítač). Tento mód je velmi spolehlivý mód, který klade důraz na aplikaci přední šifry na množinu vstupních bloků, zvaných čítače, aby vytvořily posloupnost výstupních bloků tak, že jsou spojeny s otevřeným textem(OT) operací XOR, aby vytvořily šifrový(ŠT) text a naopak. Posloupnost čítačů musí mít vlastnost, že každý blok v posloupnosti je odlišný od každého jiného bloku. Tato podmínka není omezena na jednotlivou šifru: na druhé straně všechny zprávy, které jsou šifrovány daným klíčem, musí mít všechny čítače odlišné. Čítače pro danou zprávu jsou značeny $T_1, T_2, \ldots, T_n$ a mód je definovaný následovně:

CTR šifrování :	$O_j = E_K(T_j)$	pro j $= 1, 2, \ldots, n$
	$C_j = M_j \oplus O_j$	pro j $= 1, 2, \ldots, n-1$
	$C_j^* = M_n \oplus S_u(O_n)$	
CTR dešifrování :	$O_j = E_K(T_j)$	pro j $= 1, 2, \ldots, n$
	$M_j = C_j \oplus O_j$	pro j $= 1, 2, \ldots, n-1$
	$M_n = C_n \oplus S_u(O_n)$	

$O_j \ldots$ j-tý výstupní blok
$T_j \ldots$ j-tý blok čítače
$b \ldots$ velikost bloku (v bitech)
$M_j \ldots$ j-tý blok OT
$C_j \ldots$ j-tý blok ŠT
$n \ldots$ počet bloků v OT

u ... počet bitů v posledním bloku otevřeného nebo šifrového textu
$S_u(X)$ ... řetězec složený z u nejvýznačnějších bitů řetězce X
$*$ ... poslední blok

V CRT šifrování (resp.dešifrování) je přední šifrovací funkce volána na každý čítač a výsledné výstupní bloky "XORovány" s odpovídajícími bloky otevřeného (resp.šifrového)textu pro vytvoření bloků šifrového (resp.otevřeného)textu. V posledním bloku, který může být částečným blokem u bitů, je těchto u bitů použito pro XOR operaci; zbývajících $u - b$ bitů posledního výstupního bloku je vyřazeno. V CRT šifrování i dešifrování mohou být přední šifrovací funkce prováděny paralelně; podobně bloky otevřeného textu, které odpovídají nějakému konkrétnímu bloku šifrového textu, mohou být nahrazeny nezávisle z jiných bloků otevřeného textu, jestliže odpovídající čítač je určen.



Obr.16 Schéma CTR módu

11 Útoky na AES

V této části budeme diskutovat útoky, které vychází ze strukturálních vlastností blokových šifer (=kryptoanalýza) a v posledním odstavci útoky hrubou silou. Kryptoanalytický útok prolomí šifru, jestliže jeho očekávaná zátěž je nižší než hledání klíče hrubou silou. Takový útok se jmenuje *shortcut* útok. Paradoxní význam této definice je, že šifra s delším klíčem je snadněji rozluštitelná. Vzhledem k tomu, že očekávaná složitost k objevení klíče hrubou silou je s jeho délkou rostoucí, je

jednodušší najít útok, který má nižší očekávanou složitost. Proto existence shortcut útoku pro danou šifru nutně neznamená, že šifra nemá žadnou další bezpečnost, kterou by nabídla, protože většina shortcut útoků popsaných v kryptologické literatuře stále má nepropustně vysokou složitost nebo nemůže být vůbec implementována.
Dešifrování hrubou silou vyžaduje pouze malou část dvojice otevřený text - šifrový text (OT/ŠT), zatímco většina shortcut útoků je mnohem náročnější. Některé vyžadují obrovské množství otevřeného a jemu odpovídajícího šifrového textu *útok s nešifrovaným textem (known plaintext)*. Při jiných útocích musí kryptoanalytik mít hodnoty šifrového textu odpovídající otevřenému textu, který si vybral *útok s vybraným textem (chosen plaintext)*. V útocích nazývaných *related*, je kryptoanalytik v pozici dešifrovat otevřený text s různými neznámými hodnotami klíče, které mají určité vztahy vybrané kryptoanalytikem.
Nicméně přítomnost nebo absence shortcut útoků na šifru je kritérium kvality, které je obecně akceptováno kryptografickou společností. Tedy nejpřednější kritérium k vybrání mezi finalisty soutěže na AES byla právě absence shortcut útoků. Pro mnoho moderních šifer žádný shortcut útok není znám. Přesto však rezistence iterativních blokových šifer s ohledem na specifické kryptografické metody může být vyčíslena tím, že ji aplikujeme na šifru s redukovaným počtem cyklů. Útoky na tuto verzi dovolují získat informaci o *bezpečnostním rozmezí* šifry. Jestliže pro šifru s R cykly existuje shortcut útok proti cyklově redukované verzi s $R - r$ cykly, šifra má absolutní bezpečnostní rozmezí r cyklů nebo relativní bezpečnostní rozmezí r/R. Poznamenejme, že objevení útoku na cyklově redukovanou verzi s $R/2$ cykly neznamená, že je šifra poloprolomitelná. Ovšem složitost většiny teoretických útoků roste exponenciálně s počtem cyklů.
Protože pokroky v kryptoanalýze šifer směřují k umožnění prolomení více a více cyklů, bezpečnostní rozmezí ukazuje odolnost proti přírůstku známých typů kryptoanalýzy. Nicméně to nic neříká o pravděpodobnosti těchto pokroků v kryptoanalýze nebo o rezistenci šifry proti neznámým útokům.
Často pro nové druhy kryptoanalýzy není jednoduché přesně odhadnout složitost útoku. V těchto případech můžeme získat více informací o složitosti implementací útoku na cyklově redukovanou verzi, když je neproveditelné implementovat útok na úplnou šifru.

11.1 Diferenciální a lineární kryptoanalýza

Diferenciální a lineární kryptoanalýzy jsou dva nejsilnější cíle kryptografických útoků, které jsou dnes známé. Zajistit nižší hranice složitosti těchto útoků bylo hlavním kritériem při tvorbě algoritmu Rijndael.
V jejich základní formě oba útoky nahradí informace o klíči z posledního cyklu statistickou analýzou velkého množství dvojic OT/ŠT. Získaná informace o klíči

je poté použita k nalezení více a více bitů klíče až je nalezen celý. **Diferenciální kryptoanalýza** je *chosen − plaintext* útok, kde části otevřeného textu jsou dány do dvojic, které mají fixní rozdíly. Diferenciální kryptoanalýza využívá velkého zvětšení pravděpodobnosti mezi rozdíly vzorů v otevřeném textu a rozdíly vzorů na vstupu posledního cyklu. **Lineární kryptoanalýza** je *known−plaintext* útok, který využívá velké korelace mezi paritami (lineární kombinace bitů) na vstupu posledního cyklu, s kterým je paritní otevřený text. Mnoho chytrých technik bylo publikováno, aby učinily tyto základní útoky efektivnějšími. Ačkoliv ve všech případech způsobilost předvídat velkou pravděpodobnost rozdílu přenášení a velké korelace nad vícenásobnými cykly jsou důležité pro jejich úspěšnost.
Pro Rijndael je dokázána horní mez 2^{-150} pro pravděpodobnost nějakého čtyřcyklového diferenciálního záznamu a mez 2^{-75} pro nějaký čtyřcyklový lineární záznam. V kombinaci s počtem cyklů v algoritmu Rijndael tyto hranice zajišťují velké bezpečnostní rozmezí proti diferenciálnímu i lineárnímu útoku.
Po jejich publikaci se lineární a diferenciální útoky rozšířily několika směry a byly publikovány nové útoky, které byly s nimi spojeny. Nejlepší známé rozšíření se jmenuje *zkrácená diference (truncated differential)*. Tvůrci Rijndael algoritmu vzali tyto útoky v potaz již při jeho navrhování.
Ostatní útoky používají odlišnost šíření a korelaci různými směry. To zahrnuje *bumerangové útoky* a *obdelníkové útoky*. Díky horní hranici pro 4-cyklový záznam a aktuální počet cyklů, žádná z těchto metod kryptoanalýzy nevede k *shortcut* útoku v algoritmu Rijndael.

11.2 Saturation útoky

Nejsilnější kryptoanalýza Rijndaelu je dnes *saturation útok*. (Dříve se tyto typy útoků nazývaly *strukturální útoky* a *integrální kryptoanalýza*). Saturation útok je typu *chosen − plaintext*. Využívá bytově-orientovanou strukturu šifry a pracuje na nějaké šifře s cyklickou strukturou podobné Rijndaelu.
Původní *saturation útok* umí prolomit cyklově redukované varianty Rijndaelu na 6 nebo 7 cyklech rychleji než hrubou silou. N.Ferguson navrhl nějaké optimalizace, které redukují pracovní faktor útoku.
H.Gilbert a M.Minier zdokonalili také útok, který využívá bytově orientovanou strukturu AES. Jejich útok umí prolomit redukovanou verzi AES se sedmi cykly, ale zdá se, že je pouze rychlejší než hledání 256-bitového klíče hrubou silou .

11.3 Algebraická struktura

Cyklická transformace Rijndaelu může být rozložena na sekvence kroků v několika odlišných směrech. S.Murphy zkoumal, že rozložení může být definované takovým způsobem, že kroky cyklické transformace mají nízký algebraický řád. Algebraický

řád transformace f se rovná počtu různých transformací, které mohou být konstruovány opakovanou aplikací f: $f, f \circ f, f \circ f \circ f \ldots$. Až do dneška tyto pozorování na některých komponentách cyklické transformace nevedli k žádnému kryptoanalytickému útoku. Naopak R.Wernsdorf nedávno dokázal, že úplná cyklická transformace Rijndaelu generuje střídavou skupinu. To ukazuje, že algebraický řád cyklické transformace není nízký. J.Fuller pozoroval strukturu S-boxů. S-box může být popsán jako 8 booleovských funkcí propojených mezi sebou:

$$f_i(x) \equiv f_j(g_{ij}(x)) + c_{ij} \tag{11.1}$$

Zde c_{ij} jsou konstanty a funkce g_{ij} jsou lineární. To vede k závěru, že Rijndael může být popsán průměrem 1 nelineární booleovské funkce a více lineárních/affiních operací. Toto pozorování je důsledkem matematické definice S-boxů. Stejně jako opakované použití jednoho S-boxu nevede k útoku, také toto pozorování není slabost, ale spíše ilustrace jednoduchosti návrhu šifry.

11.4 Algebraické útoky

Zřetelná algebraická struktura algoritmu Rijndael povzbudila některé týmy výzkumníků ke zkoumání bezpečnosti Rijndaelu proti algebraickým luštícím metodám. Typicky se algebraický útok skládá ze dvou kroků.

Shromažďující krok – Kryptoanalytik vyjádří šifru jako množinu *jednoduchých* rovnic v počtu proměnných. Tyto proměnné zahrnují bity (nebo byty) z otevřeného textu, šifrového textu a klíče a také z mezivýpočetních hodnot a rundovních klíčů. Výraz *jednoduchý* může být definovaný velmi volně jako *vhodný pro následující krok.*

Luštící krok – Kryptoanalytik používá některá vstupní data jako dvojice OT/ŠT, substituuje tyto hodnoty v odpovídajících proměnných v množině rovnic shromážděných v prvním kroku a zkouší řešit množinu rovnic, čímž objeví klíč.

Rijndael může být vyjádřen upravenými rovnicemi několika způsoby. Zatímco v mnoha jiných šifrovacích schématech je struktura skryta sčítáním mnoha komplexních operací, v Rijndaelu je vnitřní struktura velmi jednoduchá a zřejmá, jasně usnadňující vyjádření šifry jako množiny jednoduchých rovnic. Klíčový výsledek k posouzení je, zdali rovnice vypadající matematicky upraveně jsou také jednoduché k vyřešení. Některé pokusy byly vytvořeny ke konstruování algebraických útoků na Rijndael. Nikdo zatím nedospěl k shortcut útoku a většina dokumentů uzavírá, že je potřeba ještě mnoho výzkumů.

11.4.1 Řetězové zlomky

Ferguson odvodil uzavřenou formuli pro Rijndael, na kterou může být pohlíženo jako na zobecnění řetězových zlomků. Každý byte v mezivýsledku po 5 cyklech může být vyjádřen následovně:

$$x = K + \sum \cfrac{C_1}{K^* + \sum \cfrac{C_2}{K^* + \sum \cfrac{C_3}{K^* + \sum \cfrac{C_4}{K^* + \sum \cfrac{C_5}{K^* + p_*^*}}}}} \tag{11.2}$$

Zde každé K je nějaký rozšířený klíčový byte, každé C_i je známá konstanta a každá * je známý exponent nebo index, ale tyto hodnoty záleží na sumaci proměnných, které uzavírají výraz.
Úplně rozšířená verze (11.2) má 2^{25} členů. Za účelem prolomit 10-cyklový Rijndael by měl kryptoanalytik použít dvě rovnice tohoto typu. První by měla vyjadřovat mezivýsledné proměnné po pěti cyklech jako funkci bytů otevřeného textu. Druhá rovnice by měla pokrýt cykly 6-10 vyjádřením stejných mezivýsledných proměných jako funkci bytů šifrového textu. Kombinováním obou rovnic můžeme dospět k rovnici s 2^{26} neznámými. Opakováním této rovnice pro $2^{25}/16$ známých dvojic OT/ŠT můžeme získat dostatečnou informaci pro vyřešení neznámých v informačně-teoretickém smyslu. Zatím není známo, jak by praktický algoritmus pro řešení tohoto typu rovnic měl vypadat.

11.4.2 XL a XSL útoky

Courtois and Pieprzyk ukázali, že algoritmus Rijndael má zcela zvláštní algebraickou strukturu a může být zapsán jako systém předdefinovaných multivarietních kvadratických rovnic. Autoři předvedli, že např. pro 128-bitový Rijndael se problém nalezení tajného klíče při znalosti jednoho otevřeného textu dá popsat jako systém 8 tisíc kvadratických rovnic s 1600 binárními neznámými. Bezpečnost Rijndaelu tím však ohrožena nebyla. Není totiž znám dostatečně výkonný algoritmus, který by takovýto systém byl schopen řešit. Známý kryptolog Shamir popsal algoritmus XL, který řeší takovéto systémy v subexponenciálním čase. To by však znamenalo, že bezpečnost Rijndaelu se nezvyšuje exponenciálně s počtem cyklů. Připomeňme, že klasické útoky na blokové šifry jako např. lineární nebo diferenciální mají složitost exponenciální v závislosti na počtu cyklů. V praxi algoritmus XL není tak výkonný a nedokáže rozbít Rijndael. Systém kvadratických rovnic

popisující Rijndael není zcela náhodný, má mnoho speciálních vlastností: získaná matice je řídká a je výrazně strukturovaná. Právě tuto strukturu Curtois s Pieprzykem studovali a navrhli zlepšení obecné XL metody a její přizpůsobení právě této struktuře. Navrhli tedy novou třídu útoků, nazvanou XSL útok.
Objevili, že S-boxy použité v algoritmu Rijndael mohou být popsány množstvím implicitních kvadratických booleovských funkcí. Jestliže osm vstupních bitů je značeno $x_1, \ldots, x_8$ a osm výstupních bitů $y_1, \ldots, y_8$, potom existují rovnice ve tvaru

$$f(x_1, \ldots, x_8, y_1, \ldots, y_8) = 0 \tag{11.3}$$

kde se algebraický stupeň f rovná 2.
V podstatě 8 rovnic typu (11.3) stačí k definování S-boxu, ale Courtois a Pieprzyck vyzkoumali, že lze zkonstruovat více rovnic tohoto typu. Navíc tvrdili, že tyto extra rovnice mohou být použity k redukování složitosti řešení.
V prvním kroku XSL metody se vyberou rovnice, které popisují výstup každého podbloku šifry jako funkce stejných vstupních podbloků. Jako výsledek vezme kryptoanalytik systém již zmíněných 8000 kvadratických rovnic o 1600 neznámých, kde lineární kroky jsou ignorovány z důvodu jednoduchosti.
Nejobtížnější část XSL metody je najít účinější eliminační proces. Courtois a Pieprzyk předběžně spočítali, že složitost by byla 2^{255} kroků. Pro Rijndael s 256-bitovým klíči by složitost byla 2^{203} kroků v jejich nejoptimističtějších úvahách. Všechny tyto složitostní úvahy prováděny za předpokladu, že gausova eliminační metoda pro lineární rovnice může být implementována při složitosti $o(n^{2.4})$. Je stále předmětem debat, jestli úvahy Courtoise a Pieprzycka jsou platné a XSL útok bude vůbec fungovat.

11.5 Útoky hrubou silou

Vzhledem k tomu, že pád DES standardu způsobila právě nedostatečná délka klíče a tudíž útok hrubou silou, vyvstává otázka, nehrozí-li stejné nebezpečí u AES. Současné lidské technologie ani veškeré pozemské zdroje nestačí na to, aby mohly útočit hrubou silou na 128-bitový klíč, a žádné vhodné ani nejsou v dohledu a i kdyby byly, vždy je možné přejít na klíč delší. Pokud by došlo k pokroku například v oblasti kvantových počítačů nebo počítačů na bázi DNA, bude to jistě známo dostatečně dlouho (odhaduje se 10-15 let) před tím, než by taková technologie mohla být prakticky použitelná na lámání dlouhých klíčů AES.

Reference

[1] Paseka J.: Kryptografie (el.uč.texty,MU Brno)

[2] Klíma V.: AES - Nová šifra nastupuje (Chip, květen 2002, str.142-144)

[3] Archív článků *http://www.decros.cz/bezpecnost/kryptografie.html*

[4] domovská stránka AES od NIST *http://csrc.nist.gov/encryption/aes*

[5] Crypto-World (prosinec 2002) *http://www.mujweb.cz/veda/gcucmp*

MASARYKOVA UNIVERZITA V BRNĚ
Přírodovědecká fakulta

BAKALÁŘSKÁ PRÁCE

Nové šifrovací standardy

Lenka Vítovcová

Vedoucí bakalářské práce: doc.RNDr.Jan Paseka,CSc květen 2003

Prohlášení:
Prohlašuji, že jsem bakalářskou práci "Nové šifrovací standardy" vypracovala samostatně s využitím pramenů uvedených v seznamu literatury.

Lenka Vítovcová

Poděkování:
Ráda bych poděkovala vedoucímu bakalářské práce doc.RNDr.Janu Pasekovi,CSc. za metodické vedení a cenné rady při jejím vypracování.