FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KIV/ZOS

# IMPLEMENTACE SYMETRICKÉ BLOKOVÉ ŠIFRY AES

Jakub Vítek (viteja) - A19B0222P

Květen 2020

# Obsah

# Implementace symetrické blokové šifry AES

## KIV/BIT - Standardní zadání semestrální práce pro ak. rok 2019/20

Ve zvoleném programovacím jazyce implementujte symetrickou blokovou šifru AES. Při řešení budou splněny následující podmínky:

- Výsledek bude v hexadecimálním formátu.
- Použijte šifrovací mód ECB (Electronic Codebook), tzn. šifrovací algoritmus aplikujte přímo na bloky vstupních dat. Inicializační vektor není potřeba. Bude-li to nutné poslední blok zarovnejte zprava pomocí nul.
- Velikost (délka) klíče, resp. bloku bude 128 bitů, tzn. očekáván je klíč délky 16 Bytů. Klíč si může zvolit uživatel.
- Otestujte na souborech **Shea.jpg** a **message.txt** (oba dostupné z: http://home.zcu.cz/~jimar/BIT/). a jako klíč použijte: **josefvencasladek**

**Poznámka**: Pro kontrolu můžete využít např. nástroj:
http://aes.online-domain-tools.com/

# 1 Advanced Encryption Standard

Advanced Encryption Standard je standardizovaným algoritmem pro šifrování dat informatice. Jedná se se o symetrickou šifru, šifrování i dešifrování se provádí stejným klíčem.

Standardně, šifrování i dešifrování probíha nad bloky dat ve velikosti 128 bit (16 Byte). U klíče jsou pak vyžadovány bloky velikostí 128 bit (16 Byte), 196 bit (24 Byte) či 256 bit (32 Byte). Algoritmus může pracovat ve dvou základních režimech, kdy v režimu ECB (Electronic Codebook) jsou všechny bloky dat při šifrování na ostatních blocích nezávislé a v režimu CBC (Cipher block chaining) je šifrování následujícího bloku parametrizováno výsledkem šifrování bloku předchozího.

AES má relativně jednoduchý algebraický popis, z tohoto důvodu v minulosti proběhlo velké množství útoků na tuto šifru, výsledkem těchto útoků bylo snížení složitosti při prolomení metodou brute force. I tak však algoritmus zůstává dostatečně bezpečný a je v současné době používán pro zabepečení Wi-Fi sítě (WPA2). Výkon algoritmu je pak zajištěn podporou pro jeho vnitřní operace v dnešním hardware.

# 2 Analýza

Algoritmus AES pracuje s bloky dat ve velikosti 128 bit (16 Byte). Tyto bytes jsou uspořádány do matice o rozměrech 4x4 byte. Zadáním semestrální práce je dána nutnost použití klíče o velikosti 128 bit (16 byte). Standard algoritmu pak říká, že pro klíč o velikosti 128 bit použijeme 10 iterací algoritmu. V rámci práce budeme používat šifrovací režim ECB (Electronic Codebook), kdy budeme s každým blokem dat pracovat nezávisle na všech ostatních blocích dat.

V základu algoritmus pracuje na bázi substitucí, kdy nahrazujeme vstupní data výstupními a permutací, kdy měníme pozici dat v logické struktuře (4x4 byte matice). V algoritmu jsou obsaženy matematické operace, mezi které patří například bitová operace XOR či rotace části dat (words). Nejnáročnější operací je násobení (a obecně aritmetika) nad Galoisovým tělesem GF($2\hat{8}$). Pro zvýšení výkonu algoritmu je mnoha autory doporučováno předpočítat hodnoty daného násobení a pouze provádět vyhledávání v předpočítaných datech.

# 3 Implementace

Samotný algoritmus AES je standardizován, při jeho implementaci musíme dodržovat standardizovaný postup, který lze shrnout jako:

1. Expanze klíče - Vytvoření poklíčů z počátečního klíče
2. Inicializace - XOR prvního klíče s daty
3. Jednotlivé iterace od prvního k poslednímu podklíči
   (a) Substitice bytes pro stav
   (b) Prohození řádků ve stavu
   (c) Kombinace všech bytes sloupců ve stavu
   (d) Přidání podklíče (XOR)
4. Závěrečná část
   (a) Finální substituce bytes
   (b) Finální prohození řádků
   (c) Přidání finálního podklíče (XOR)

Algoritmus pro dešifrování je velice podobný. Podklíče se však iterují opračným způsobem, s inverzními operacemi k těm ze šifrování a v lehce jiném pořadí. Dešifrování lze shrnout následovně:

1. Expanze klíče - Vytvoření poklíčů z počátečního klíče
2. Inicializace - XOR posledního podklíče s daty
3. Jednotlivé iterace od poslední podklíče k prvnímu
   (a) Inverzní prohození řádků ve stavu
   (b) Inverzní substitice bytes pro stav
   (c) Přidání podklíče (XOR)
   (d) Inverze ke kombinaci všech bytes sloupců ve stavu
4. Závěrečná část
   (a) Inverzne k prohození řádků
   (b) Inverze k substituci bytes
   (c) Přidání první podklíče (XOR)

## 3.1 Expanze klíče

Jedná se o operaci, při které z počátečních 128 bit klíče (16 byte) vytvoříme, v našem případě 11 dalších poklíčů (12 dohromady - první, 10 pro iteraci a poslední). Další podklíč je vždy vytvářen z klíče předchozího. Předchozí klíč je rotován tak, že z původního klíče [a0, a1, a2, a3] vznikne [a1,a2,a3,a0]. Následně je proveda substituce bytes a nakonec je tvorba následující podklíče dokončena provedením operace XOR s konstantou rcon, jež je závislá na aktuálním pořadí generovaného klíče. Ve zdrojovém kódu se jedná o funkci *expand_key*.

## 3.2 Inicializace

V tomto kroku je provedena pouhá operace XOR se vstupními daty a aktuálním podklíčem (prvním či posledním, na základě zda šifrujeme či dešifrujeme). V semestrální práci tuto část zajišťuje funkce *add_round_key*.

## 3.3 Substituce a inverzní substituce bytes

Tato operace nahrazuje původní blok o velikost 16 byte. V případě, že šifrujeme, využíváme pro nahrazení přístup do dvourozměné tabulky ve třídě AES128 *AES128.sbox* a v případě dešifrování přistupujeme do dvourozměrné tabulky *AES128.rsbox*. Pro jeden byte provádíme přístup tak, že horní 4 bits nám určují řádek a dolní 4 bits nám určují sloupec v dané tabulce. Šifrování zajišťují funkce *sub_bytes_16B*, *sub_word* a dešifrování naopak funkce *inv_sub_bytes_16B* a *inv_sub_word*.

## 3.4 Prohození řádků

Vstupem funkce jsou data uspořádaná do matice 4x4 byte. První řádek matice se nemění. Druhý řádek matice je cyklickly posunut doleva, takže řádek b: [b0,b1,b2,b3] je transformován na [b1,b2,b3,b0]. Podobným způsobem je posunut třetí řádek o dvě pozice doleva -> c: [c0,c1,c2,c3] se změní na [c2, c3, c0, c1]. Obdobným způsobem změníme i poslední řádek o 3 místa doleva, takže výsledek pro d: [d0, d1, d2, d3] bude [d3, d1, d2, d3]. U dešifrování se řádky cyklickly posouvají pro změnu doprava. Funkcionalitu zajišťují funkce *shift_rows* a *inv_shift_rows*.

## 3.5 Kombinace slopců

V této části se provádí součit konstantní matice a vektoru (jeden sloupec dat). Operace sčítání při násobení matic je nahrazena operací XOR a násobení je

$$\begin{bmatrix} result[0] \\ result[1] \\ result[2] \\ result[3] \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} * \begin{bmatrix} column[0] \\ column[1] \\ column[2] \\ column[3] \end{bmatrix} \quad (1)$$

Obrázek 1: Rovnice pro kombinaci sloupců při šifrování

$$\begin{bmatrix} result[0] \\ result[1] \\ result[2] \\ result[3] \end{bmatrix} = \begin{bmatrix} 0x0e & 0x0b & 0x0d & 0x09 \\ 0x09 & 0x0e & 0x0b & 0x0d \\ 0x0d & 0x09 & 0x0e & 0x0b \\ 0x0b & 0x0d & 0x09 & 0x0e \end{bmatrix} * \begin{bmatrix} column[0] \\ column[1] \\ column[2] \\ column[3] \end{bmatrix} \quad (2)$$

Obrázek 2: Rovnice pro kombinaci sloupců při dešifrování

nahrazeno násobením nad Galoisovým tělesem GF($2\hat{8}$). Pro šifrování operace vypadá následovně:

Pro dešifrování pak operace vypadá takto:

Funkcionalita je zajištěna funkcemi *mix_columns*, *inv_mix_columns*, samotné násobení matic je součástí funkce *matrix_mul* a násobení nad Galoisovým tělesem je implementováno jako vyhledávání v tabulkách *AES128.galois_mulX*, kde X je číslo, kterým se nasobí. Vyhledávání zajištuje funkce *galois_lookup*. Součástí práce je i původní funkce *galois*, která prováděla výpočet bez vyhledávání, vzhledem k její náročnosti jí však v práci nadále nevyužívám.

## 3.6 Přidání podklíče

Funkce *add_round_key* zajištuje XOR mezi aktuálním podklíčem a aktuálními daty.

# 4 Uživatelská příručka

Program je strikně terminálový a vyžaduje základní znalost práce s terminálem. Je vyžadována instalace podpory pro jazyk Python 3 v minimální verzi 3.6.8 a dostupnost knihovny Numpy.

## 4.1 Parametry programu

| Krátký přepínač | Přepínač | Popis |
|---|---|---|
| -h | –help | Vypíše návod k použití programu |
| -e | –encrypt | Šifrovací režim programu |
| -d | –decrypt | Dešifrovací režim programu |
| -k | –key | Specifikuje AES128 klíč |

Tabulka 1: Tabulka parametrů programu

## 4.2 Vstup a výstup

Jediný způsob zadání klíče je jeho předání jako argument programu. V případě, že žádný klíč nebyl zadán, bude použit implicitní klíč *josefvencasladek*. Povolené znaky pro klíč jsou znaky z ASCII tabulky. Při spuštění programu je provedena validace klíče. Klíč musí být 16 byte dlouhý už při zadávání.

Data k šifrování či dešifrování se v zásadě dají programu poskytnou dvěma způsoby. Prvním způsobem je předání textu přes "pipe"(viz. příklady spouštění programu), v případě, že program nedostane při spuštění žádná data, je možné je zadat jako textový vstup v terminálu. V takovémto případě program čeká na než získá blok data k šifrování (16 byte). Program čeká do nekonečna na vstup, uživatel sám ukončuje program ve chvíli, kdy získal výsledek šifrování posledního bloku který chtěl šifrovat.

Standardně se výstup vypisuje na stadardní výstup programu. Mezi každým bytem je mezera pro zvýšení čitelnosti výstupu, z toho důvodu nelze výstup šifry vložit jako vstup dešifrování jelikož mezery jsou také data. Pokud uživatel vyžaduje data, která právě zašifroval, dešifrovat, musí sám odstranit mezery. Bytes jsou vypisovány v hexadecimálním formátu bez počátečního "0x".

## 4.3 Příklady spuštění programu

```
# Spuštění programu v šifrovacím režimu
# bez zadaného klíče a vstupu
python3 ./aes.py --encrypt

# Spuštění programu v šifrovacím režimu
# se vstupem přes "pipe" s jiným než standardním klíčem
cat cesta/k/obrazku.jpg | python3 \
./aes.py --encrypt --key ahojahojahojahoj

# Spuštění programu v dešifrovacím režimu se
# vstupem přes pipe a přesmerováním výstupu
cat soubor.txt | python3 ./aes.py \
--decrypt > vystup.txt

# Měření výkonu algoritmu
cat /dev/urandom | python3 \
./aes.py | pv > /dev/null
```

# 5 Závěr

V rámci semestrální práce jsem úspěšně implementoval blokovou symetrickou šifru AES se 128 bitovým klíčem. Tato implementace podporuje šifrování i dešifrování. Neoptimalizovaná verze dosahovala na testovacím souboru Shea.jpg průchodnosti dat okolo *5 KiB/s*. Po optimalizaci dosahuje program průchodnosti přibližně *20 KiB/s*.

I přesto, že je práce plně funkční nedosahuje průchodnosti, kterou poskytuje například openssl. Program by bylo možné dále optimalizovat paralelním zpracováním bloků, efektivnější prací s daty či využitím nízkoúrovňových instrukcí podporovaných drtivou většinou dnešních procesorů.

# Seznam rovnic

# Seznam tabulek