FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Semestrální práce KIV/BIT

Implementace šifry AES

Pavel Třeštík
A17B0380P

13. května 2020

Obsah

1 Zadání **1**

2 Princip šifry AES **2**
2.1 KeyExpansion 3
2.2 AddRoundKey 4
2.3 SubBytes 4
2.4 ShiftRows 4
2.5 MixColumns 4
2.6 Zdroje 5
2.6.1 Wikipedia 5
2.6.2 GitHub 5
2.6.3 Jiné 5

3 Implementace **5**
3.1 Moduly 6
3.2 aes.c 6
3.2.1 Důležité konstanty 6
3.2.2 Funkce 6
3.3 main.c 7

4 Uživatelská dokumentace **8**
4.1 Překlad zdrojových souborů 8
4.2 Spuštění programu 8

5 Závěr **9**

1 Zadání

Úkolem je implementovat symetrickou blokovou šifru AES. Práce musí splňovat následující podmínky.

- Výsledek bude v hexadecimálním formátu.
- Použití šifrovacího módu ECB, tzn. aplikovat šifrovací algoritmus přímo na vstupní data. Inicializační vektor není potřeba. Dle nutnosti bude poslední blok dat zarovnán nulami z prava.
- Velikost bloku a klíče bude 128 bitů (16 Bytů). Klíč může zvolit uživatel
- Testovaní bude provedeno na poskytnutých souborech **Shea.jpg** a **message.txt** s klíčem **josefvencasladek**

2 Princip šifry AES

Jedná se o symetrickou blokovou šifru. To znamená, že vstupní data jsou šifrována po blocích a jsou šifrována i dešifrována pomocí stejného klíče.

Šifrování se provádí nad blokem dat, pro který se provádí několik transformací v určitém počtu kol. Velikost bloku je shodná s velikostí klíče. Podle velikosti klíče (bloku) je potom určen počet kol, kolikrát se provedou transformace. Pro náš algoritmus je velikost klíče 128 bitů a počet kol pro tuto velikost je 10.

Algoritmus se potom skládá ze 4 částí, jejichž podrobnější vysvětlení následuje:

1. `KeyExpansion` – rozšíření klíče pomocí "Rijndael's key schedule". Jedná se o algoritmus, který vypočte nový klíč pro každé kolo AES šifrování. Každý nový klíč je odvozen z předchozího a úplně první klíč je klíč zvolen k šifrování. Pro jednotlivé klíče je potom využíván výraz **round key**.

2. `AddRoundKey` – aplikování prvního **round key** na blok dat, které budou šifrovány, před zahájením transformací.

3. `9 kol transformací`:

   (a) `SubBytes` – nahradí každý Byte stavu za odpovídající Byte ve vyhledávací tabulce.

   (b) `ShiftRows` – posune pořadí prvků v řádkách.

   (c) `MixColumns` – kombinuje Byty ve sloupcích a výsledkem je sloupec jiných Bytů.

   (d) `AddRoundKey` – aplikování **round key** pro dané kolo.

4. `Poslední (10.) kolo transformací`:

   (a) `SubBytes`

   (b) `ShiftRows`

   (c) `AddRoundKey`

   Jedná se o stále stejné transformace jako v kolech 1-9, ale už bez provedení kroku `MixColumns`. Tento krok je vynechán, aby mělo šifrování a dešifrování podobnější strukturu a také proto, že síla šifry už se nezmění i bez provedení tohoto kroku.

2.1 KeyExpansion

Klíč je pro každé kolo šifrování jiný. Tyto klíče mohou být předem vygenerovány pro všechna kola následujícím způsobem:

$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus rcon_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod N \\ W_{i-N} \oplus \text{SubWord}(W_{i-1}) & \text{if } i \geq N, N > 6, \text{ and } i \equiv 4 \pmod N \\ W_{i-N} \oplus W_{i-1} & \text{otherwise.} \end{cases}$$

Obrázek 1: Způsob generování následujícího klíče
Zdroj: `https://en.wikipedia.org/wiki/AES_key_schedule`

N – počet 32-bitových slov v klíči
K_i – i-té slovo šifrovacího klíče
W_i – i-té slovo generovaného klíče

Z Obrázku 1 lze vidět, že se při generování slov klíče používají funkce `SubWord` a `RotWord` a konstanta `rcon`.

`SubWord` – jedná se v podstatě o `SubBytes` použité na Byty slova.
`RotWord` – posune slovo o pozici doleva.
`RotWord`$[b_0 b_1 b_2 b_3]$ `=` $[b_1 b_2 b_3 b_0]$
`rcon` – může být spočítáno následujícím způsobem (viz Obrázek 2).

$$rc_i = \begin{cases} 1 & \text{if } i = 1 \\ 2 \cdot rc_{i-1} & \text{if } i > 1 \text{ and } rc_{i-1} < 80_{16} \\ (2 \cdot rc_{i-1}) \oplus 11\text{B}_{16} & \text{if } i > 1 \text{ and } rc_{i-1} \geq 80_{16} \end{cases}$$

Obrázek 2: Počítání konstanty rcon
Zdroj: `https://en.wikipedia.org/wiki/AES_key_schedule`

rc_i – první Byte slova klíče generovaného pro i-té kolo.
i – i-té kolo

2.2 AddRoundKey

Každý Byte bloku je nahrazen výsledkem operace XOR nad původním Bytem pozice a odpovídajícím Bytem klíče pro dané kolo.

Na příklad 7-mý Byte bloku je výsledkem XOR mezi 7-mým Bytem bloku a 7-mým Bytem klíče pro dané kolo.

2.3 SubBytes

Nahradí každý Byte bloku za odpovídající hodnotu ve vyhledávací tabulce. Jedná se o "**Rijndael S-box**"(dále jen **S-box**) substituční tabulku. Hodnota nahrazovaného Bytu se použije jako hodnota klíče (indexu) v **S-box** a nahradí Byte za hodnotu nacházející se na této pozici.

S-box může být vygenerován pomocí jednoduchého algoritmu, ale ten jsem se rozhodl v rámci této semestrální práce neimplementovat.

2.4 ShiftRows

První řádku nemění.[1]
Pro druhou řádku posune každý Byte o 1 do leva.
Pro třetí řádku posune každý Byte o 2 do leva.
Pro čtvrtou řádku posune každý Byte o 3 do leva.

2.5 MixColumns

Transformuje prvky tak, že vynásobí sloupec bloku konstantní maticí a výsledkem je nový sloupec. Sloupec je násoben maticí:

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

[1] Blok uvažujeme jako matici 4x4 pro náš 16 Bytový blok

2.6 Zdroje

2.6.1 Wikipedia

- Advanced Encryption Standard
  `https://en.wikipedia.org/wiki/Advanced_Encryption_Standard`
- Rijndael S-box
  `https://en.wikipedia.org/wiki/Rijndael_S-box`
- AES key schedule
  `https://en.wikipedia.org/wiki/AES_key_schedule`
- Rijndael MixColumns
  `https://en.wikipedia.org/wiki/Rijndael_MixColumns`
- Finite field arithmetic
  `https://en.wikipedia.org/wiki/Finite_field_arithmetic`

2.6.2 GitHub

- GitHub projekt uživatele kokke
  `https://github.com/kokke/tiny-AES-c`
- GitHub projekt uživatele dhuertas
  `https://github.com/dhuertas/AES`

2.6.3 Jiné

- Příklad AES šifry
  `https://kavaliro.com/wp-content/uploads/2014/03/AES.pdf`
- Why is MixColumns omitted from the last round of AES?
  `https://crypto.stackexchange.com/questions/1346/why-is-mixcolumns-omitted-from-the-last-round-of-aes`

3 Implementace

Práci jsem se rozhodl implementovat v jazyce C. Jazyk jsem si vybral především proto, že se velká část činností provádí lépe na nižší úrovni a také pro jeho rychlost.

3.1 Moduly

Projekt má pouze dva moduly. Modul šifrovacího algoritmu `aes.c` (a příslušný header file) a modul obsluhy `main.c`.

3.2 aes.c

3.2.1 Důležité konstanty

`const u_char Rcon[11]` – rcon konstanty pro generování **round key**[2]
`const u_char s_box[256]` – vyhledávací tabulka **S-box**

3.2.2 Funkce

- `void add_round_key(u_char state[MAGICAL_FOUR] [MAGICAL_FOUR], u_char round_key[MAGICAL_SIXTEEN * ROUND_COUNT], short round)`

  Přidá (XOR) klíč k bloku. Parametry jsou **state** = blok 4x4, **round_key** = buffer s klíči pro všechny kola, **round** = kolikáté kolo se provádí.

- `void sub_bytes(u_char state[MAGICAL_FOUR] [MAGICAL_FOUR])`

  Nahradí Byty **state** příslušným Byty z **S-box**.

- `void shift_rows(u_char state[MAGICAL_FOUR] [MAGICAL_FOUR])`

  Posune řádky **state** způsobem popsaným v principu.

- `u_char gmul(u_char a, u_char b)`

  Pomocná funkce k **MixColumns**. Vynásobí **a** s **b** v **GF(2)**[3].

- `void mix_columns(u_char state[MAGICAL_FOUR] [MAGICAL_FOUR])`

  Transformuje sloupce **state** způsobem popsaným v principu.

- `void rot_word(u_char word[MAGICAL_FOUR])`

  Pomocná funkce k **KeyExpansion**. Posune Byty slova z parametru o 1 do leva.

[2] u_char je typedef pro unsigned char, který jsem používal jako 1B strukturu
[3] Galois Field nad 2-ma prvky. Označován také jako Finite field

- `void sub_word(u_char word[MAGICAL_FOUR])`

  Pomocná funkce ke **KeyExpansion**. Zamění Byty slova z parametru za příslušný Byty z **S-box**.

- `void key_expansion(u_char key[MAGICAL_SIXTEEN], u_char round_key[MAGICAL_SIXTEEN * ROUND_COUNT])`

  Vygeneruje klíče pro všechna kola podle algoritmu popsaném v principu. Jako parametry bere **key** – originální klíč a **round_key** – buffer uchovávající klíče všech kol.

- `void append_state_to_output(u_char state [MAGICAL_FOUR][MAGICAL_FOUR], int where)`

  Přidá zašifrovaný blok do **output** bufferu. Mimo **output** bere jako parametr **where** pozici, kam blok zapíše.

- `void encrypt(u_char state[MAGICAL_FOUR][MAGICAL_FOUR], u_char round_key[MAGICAL_FOUR * ROUND_COUNT])`

  Funkce šifrující blok předán parametrem. Parametr **state** – blok dat k šifrování a **round_key** – klíče pro šifrování.

- `void print_output(int length)`

  Vypíše zašifrované data v hexadecimálním tvaru (pro lidi čitelné) a mezerou po každých 4 číslicích (2 Byte). Jako parametr **length** – bere délku zašifrovaných dat.

- `u_char *get_output()`

  Vrací pointer na zašifrované data, aby se s nimi dalo pracovat z jiného souboru.

3.3 main.c

- `void print_help()`

  Vypíše nápovědu ke spuštění programu.

- `int write_output_to_file(char *file_name, u_char *output, int output_size)`

Zapíše zašifrovaná data do souboru. Na rozdíl od výpisu do konzole jsou do souboru znaky psány svými hodnotami a pro lidi nečitelné. Parametry jsou **file_name** – název souboru do kterého se výstup zapíše, **output** – buffer se zašifrovanými daty a **output_size** – velikost zašifrovaných dat.

- `int read_input(char *file_name, int *size)`

  Čte vstup a rovnou ho šifruje. Tuto bylo původně myšleno, aby se celá nezakódovaná zpráva nedržela v paměti, ale to je zbytečné vzhledem k tomu že je známý klíč. Parametry jsou **file_name** – jméno vstupního souboru a **size** – pointer na proměnnou uchovávající velikost vstupu.

- `void run(int argc, char *argv[])`

  Zpracuje parametry a spustí s nimi šifrování.

4 Uživatelská dokumentace

4.1 Překlad zdrojových souborů

Projekt obsahuje **makefile**, takže překlad zdrojových souborů na systémech Linux s překladačem gcc je velmi snadný. Stačí pouze v kořenovém adresáři projektu spustit příkaz **make** a program se přeloží a vytvoří spustitelný soubor **aes**. Program nebyl dělaný s automatickým překladem pro systémy Windows, ale aby Windows měl přístup k překladači C je pravděpodobné, že budete muset nainstalovat program jako Cygwin nebo MinGW. S těmito programy by měl makefile fungovat také, popřípadě by bylo třeba doinstalovat do těchto programů modul make.

4.2 Spuštění programu

Po přeložení souborů je vytvořený spustitelný soubor **aes**. Tento soubor vyžaduje alespoň jeden parametr při spuštění. Celkově program má 1 povinný parametr a 2 nepovinné.

Program je tedy možno spustit následovně:
aes <input_file> -k [cipher_key] -o [output_file]

- **<input_file>** – název vstupního souboru, který bude šifrován. Toto je povinný parametr a musí být na první pozici.
- **-k** [**cipher_key**] – při použití přepínače "-k" program použije parametr následující tento přepínač jako klíč k šifrování. Pokud není zvolen je použit testovací klíč "**josefvencasladek**". Klíč musí být 16 písmen dlouhý, pokud je jiné délky, program vypisuje chybové hlášení.
- **-o** [**output_file**] – při použití přepínače "-o" program použije parametr následující tento přepínač jako cílový soubor, kam se zapíší zašifrovaná data. Pokud není zadán, výstup je vypsán na konzoli.

```
[pavel@cf-arch semestral]$ ./aes res/message.txt -o
Invalid argument count!
Usage:
aes <input_name> -k [cipher_key] -o [output_file]

Parameter <input_name> is the input file that is to be enrypted.
        This parameter is required.
If option -k is used it is expected to be followed by [cipher_key].
        Which is 16 custom letter encryption key.
If option -o is used it is expected to be followed by [output_file].
        This is target file where the encrypted data will be written.
```

Obrázek 3: Příklad spuštění s chybějícím parametrem po přepínači -o

5 Závěr

Program by měl bez problému fungovat. Byl otestován proti výstupům ze stránky `aes.online-domain-tools.com/` a vrací shodné výsledky. Program je také velice rychlý. Soubor o velikosti zhruba 3 MB zašifruje za zhruba 2.2 vteřiny a testovací soubor **Shea.jpg** kolem 0.1 vteřiny.

Pár detailů na zlepšení se ale určitě najde. Například by neškodilo zvětšit výstupní buffer, protože momentálně má program maximální výstupní buffer 5 MiB. Dále by určitě bylo vhodné přepsat můj **u_char** (typedef pro `unsigned char`) na jiný datový typ, který má pevně danou velikost 1 Byte. Začal jsem používáním `char`, protože jsem ze začátku načítal soubor do bufferu typu `char`, ale protože to způsobovalo potíže s přetékáním změnil jsem `char` na `unsigned char`. Toto není takový problém při použití překladače gcc

a většiny překladačů, kde je char 1 Byte, ale pokud by byl použit překladač, který bere char jako více Bytový typ nastal by problém.

Další potenciální problém, také s načítáním, je šifrování během načítání souboru. Výhodou je, že soubor je šifrován bez závislosti na velikosti souboru, ale nevýhodou a potenciálním nebezpečím je stream otevřený po většinu doby běhu programu.