06. Akcelerace AES pomocí dedikovaných instrukcí AES-NI

- Implementujte AES s využitím AES-NI pomocí C/C++ intrinsických funkcí
- Implementace bude bez smyček a volání funkcí (kromě inline)
- Implementujte expanzi klíče nejprve odděleně od zpracování rund (podobně jako minule, ale s AES-NI)
- Implementujte expanzi klíče na místě (on-the-fly) při zpracování rund, bez ukládání do pole
- Změřte čas zpracování za stejných podmínek jako minule a porovnejte zjištěné časy

Odkazy

- Intel AES-NI White Paper (https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf) – pro inspiraci, ale nekopírujte řešení
- Intel Intrinsics Guide (https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html)
- Intel® 64 and IA-32 Architectures Software Developer's Manual - Instruction Set Reference, A-Z (https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf)

Použité instrukce

Instrukce pro akceleraci výpočtu šifry AES jsou z rodiny SSE (Streaming SIMD Extension). Používají 128bitové XMM registry (xmm0 až xmm15), které mohou být zpracovávány různými instrukcemi jako 4 32bitové, nebo 2 64bitové, nebo 8 16bitových, nebo 16 osmibitových hodnot. Pro základní přehled existujících registrů platformy x86 viz např. stránku na Wikipedii (https://en.wikipedia.org/wiki/X86#x86_registers) .

AESENC xmm1, xmm2/m128 – Vypočti 1 rundu AES (SubBytes, ShiftRows, MixColumns, AddRoundKey)

- xmm1 … vstup/výstup hodnota stavového pole (128 bitů)
- xmm2/m128 … hodnota rundovního klíče
- C/C++: `__m128i _mm_aesenc_si128 (__m128i a, __m128i RoundKey)`

AESENCLAST xmm1, xmm2/m128 – Vypočti poslední rundu AES (SubBytes, ShiftRows, AddRoundKey)

- xmm1 … vstup/výstup hodnota stavového pole (128 bitů)
- xmm2/m128 … hodnota rundovního klíče
- C/C++: `__m128i _mm_aesenclast_si128 (__m128i a, __m128i RoundKey)`

AESKEYGENASSIST xmm1, xmm2/m128, imm8 – Připrav hodnoty pro výpočet rundovního klíče

- xmm1 … výstupní hodnota pomocné hodnoty k tvorbě rundovního klíče
- xmm2/m128 … vstupní hodnota předchozího rundovního klíče, skládá se ze 4 slov

- imm8 … hodnota z pole Rcon
- C/C++: `__m128i _mm_aeskeygenassist_si128 (__m128i a, const int imm8)`

Input xmm2	**w3**	**w2**	**w1**	**w0**
Output (xmm1)	RotWord(SubWord(w3)) xor imm8	SubWord(w3)	RotWord(SubWord(w1)) xor imm8	SubWord(w1)

PSHUFD xmm1, xmm2/m128, imm8

- zkopíruje vybraná 32bitová slova ze vstupu na výstup
- pro každé výstupní 32bitové slovo lze vybrat jedno za 4 vstupních slov pomocí 2 bitů operandu imm8
- C/C++: `__m128i _mm_shuffle_epi32 (__m128i a, int imm8)`
- Popis z Intel Intrinsics Guide: (https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#text=_mm_shuffle_epi32&expand=224,4916,4738,221,224,4738,221,4738,4738&ig_expand=6317)

```
SELECT4(src, control){
        CASE(control[1:0])
        0:      tmp[31:0] := src[31:0]
        1:      tmp[31:0] := src[63:32]
        2:      tmp[31:0] := src[95:64]
        3:      tmp[31:0] := src[127:96]
        ESAC
        RETURN tmp[31:0]
}

dst[31:0] := SELECT4(a[127:0], imm8[1:0])
dst[63:32] := SELECT4(a[127:0], imm8[3:2])
dst[95:64] := SELECT4(a[127:0], imm8[5:4])
dst[127:96] := SELECT4(a[127:0], imm8[7:6])
```

PSLLDQ xmm1, imm8

- Posune logicky doleva operand xmm1 o imm8*8 bitů
- VPSLLDQ* xmm1, xmm2, imm8
- Posune logicky doleva operand xmm2 o imm8*8 bitů a výsledek uloží do xmm1
- Obě instrukce pro posun mají společný intrinsický tvar pro C/C++:
    - `__m128i _mm_slli_si128 (__m128i a, int imm8)`

PXOR xmm1, xmm2/m128

- provede logický XOR po bitech (xmm1 = xmm1 xor xmm2)
- `__m128i _mm_xor_si128 (__m128i a, __m128i b)`

Použití instrukcí v C/C++

Pro použití speciálních instrukcí je v překladači podpora tzv. intrinsics, tj, funkcí a typů, které odpovídají jednotlivým instrukcím. Seznam intrinsických funkcí pro Intel CPU je např. v [Intel Intrinsics Guide (https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html)](https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html) s možností vyhledávání.

Pro GCC zapneme podporu AES instrukcí přepínačem `-m aes`, případně `-maes`

V programu je potřeba zpřístupnit tyto funkce např. pomocí

```
#include <immintrin.h>
```

Dále budeme potřebovat zejména typ, který odpovídá 128bitovým hodnotám – bloku a klíči pro AES.

```
__m128i
```

Data z paměti lze do proměnných tohoto typu nahrávat pomocí

```
__m128i _mm_loadu_si128 (__m128i const* mem_addr)
```

Ukládat lze pomocí

```
void _mm_storeu_si128 (__m128i* mem_addr, __m128i a)
```

Tyto funkce jsou důležité pro zajištění správné sémantiky (abychom nenarazili na nedefinované chování / undefined behavior) a vyhnuli se případným problémům s nezarovnaným přístupem (unaligned access).

Expanze klíče

Pro AES-128 je expanze klíče nejjednodušší, protože délka klíče i délka bloku jsou stejné.

Expanzi provádíme po 128bitových rundovních klíčích, každý 128b operand se vejde do jednoho xmm registru (proměnné typu __m128i).

Mějme předchozí rundovní klíč v 128bitové proměnné, která se skládá ze čtyř 32bitových slov od nejvyššího po nejnižší: **{w3, w2, w1, w0}**. Chceme vytvořit nový rundovní klíč **{w7, w6, w5, w4}**.

Nechť $RSX(w3, rcon) = RotWord(SubWord(w3)) \oplus rcon$. Jednotlivá slova nového klíče jsou

$$
\begin{aligned}
w4 &= w0 \oplus RSX(w3, rcon) \\
w5 = w4 \oplus w1 &= w0 \oplus w1 \oplus RSX(w3, rcon) \\
w6 = w5 \oplus w2 &= w0 \oplus w1 \oplus w2 \oplus RSX(w3, rcon) \\
w7 = w6 \oplus w3 &= w0 \oplus w1 \oplus w2 \oplus w3 \oplus RSX(w3, rcon)
\end{aligned}
$$

Problém můžeme rozložit na dvě části

- Vytvoření pomocné proměnné $tmp1 = \{RSX(w3, rcon), RSX(w3, rcon), RSX(w3, rcon), RSX(w3, rcon)\}$
- Vytvoření pomocné proměnné $tmp2 = \{w0 \oplus w1 \oplus w2 \oplus w3, w0 \oplus w1 \oplus w2, w0 \oplus w1, w0\}$
- Nový rundovní klíč $\{w7, w6, w5, w4\} = tmp1 \oplus tmp2$

Pomocná proměnná tmp1 s obsahem RotWord(SubWord(w3)) xor rcon

Instrukce AESKEYGENASSIST nám vrátí slovo $RSX(w3, rcon) = RotWord(SubWord(w3)) \oplus rcon$ jako nejvyšší slovo ve výstupním operandu (viz výše). Toto slovo musíme nakopírovat do všech slov proměnné tmp1. Toho můžeme dosáhnout pomocí instrukce PSHUFD s řídicím bajtem 11111111 (bin), čímž do všech slov výsupu nakopírujeme nejvyšší, tedy třetí slovo vstupu (adresa 3 = 11 bin).

Původní klíč	**w3**	**w2**	**w1**	**w0**
AESKEYGENASSIST output	RSX(w3, rcon)	SubWord(w3)	RSX(w1, rcon)	SubWord(w1)
Output of PSHUFD = tmp1	**RSX(w3, rcon)**	**RSX(w3, rcon)**	**RSX(w3, rcon)**	**RSX(w3, rcon)**

Pomocná proměnná tmp2

Potřebujeme operace logického posuvu doleva (PSLLDQ) a bitového XORu (PXOR). Opakovanými posuvy o 4 bajty a xorováním dostaneme hodnotu tmp2.

Původní klíč	**w3**	**w2**	**w1**	**w0**
$\oplus$	w2	w1	w0	0
$\oplus$	w1	w0	0	0
$\oplus$	w0	0	0	0
= tmp2	$w0 \oplus w1 \oplus w2 \oplus w3$	$w0 \oplus w1 \oplus w2$	$w0 \oplus w1$	$w0$