Sprawozdanie - Zaawansowane architektury procesorów

Borys Jeleński, Jakub Postępski

13 czerwca 2017

1 Wstęp

Celem było porównanie metod szyfrowania symetrycznego algorytmem AES-128-CBC[1] na mikrokontrolerze z serii STM32L1. Porównano następujące implementacje:

- autorską, programową
- sprzętową

2 Algorytm AES-128-CBC

W algorytmie AES-128-CBC mamy do czynienia z szyfrowaniem symetrycznym. Klucz, tak jak blok danych, ma długość 128 bitów. Dłuższe wiadomości są rozbijane na bloki tej długości. Bloki przedstawiane są jako macierze rozmiaru 4 x 4 bajtów, nazywanych macierzami stanu. Dla każdego z bloków generowany jest podklucz początkowy, oraz po jednym podkluczu dla każdej rundy szyfrującej. Dla rundy inicjującej, każdy bajt z bloku jest sumowany operacją XOR z odpowiednim bajtem podklucza początkowego. Następnie wykonywanych jest 9 rund szyfrujących. Dla każdej z rund wykonywane są następujące operacje:

- operacja SB - zastępowanie bajtów innym bajtami (zgodnie ze zdefiniowanymi tabelami)
- operacja SR - przesunięcie bitowe trzech ostatnich wierszy macierzy stanu, o zdefiniowane wartości
- operacja MC - mnożenie macierzy stanu przez zdefiniowaną macierz rozmiaru 4x4 bajtów
- operacja AR - wykonanie operacji XOR na podkluczu dla danej rundy oraz danych w macierzy stanu

Na koniec wykonywana jest runda kończąca, która różni się od wcześniejszych rund brakiem operacji MC.
Pierwsze bajty klucza są oryginalnym kluczem. Klucz jest rozszerzany do 176 bajtów. Generowanie czterech kolejnych bajtów klucza polega na:

- skopiowaniu ostatnich 4 bajtów klucza
- zrotowaniu bajtów o jedną pozycję w lewo
- wymieszaniu bajtów zgodnie ze zdefiniowanym porządkiem
- zsumowanie XOR lewego bajtu z dwójką podniesioną do potęgi numeru iteracji pomniejszoną o jeden

- zsumowaniu XOR ostatnich 4 bajtów ze zdefiniowanym blokiem klucza.

W trybie CBC szyfrowany blok jest sumowany w trybie XOR z poprzednim blokiem, przed rozpoczęciem szyfrowania. Do odszyfrowywania wykonywane są analogiczne operacje.

3 Środowisko testowe

Do testów wykorzystano mikrokontroler STM32L162RD[2], który posiada sprzętowe wsparcie dla algorytmu AES-128. Taktowanie rdzenia Cortex-M3 zostało ustawione na 16MHz. Mikrokontroler posiada pamięć flash typu ECC oraz 48KB pamięci RAM. Mikrokontroler wlutowano w płytkę Nucleo 64 (posiada wbudowanyprogramator ST-Link). Użyto środowiska programistycznego Keil uVision 5.
Do obsługi mikrokontrolera wykorzystano biblioteki HAL. Do komunikacji z użytkownikiem wykorzystano połączenie szeregowe, z prędkością 115200 b/s. Zastosowano autorską bibliotekę do parsowania poleceń.
Do pamięci flash wgrano blok z niezaszyfrowanymi danymi o długości 64kB, w taki sposób aby nie przesłaniały kodu programu. Szyfrowane dane zawsze pobierane są z pamięci flash. Istnieje możliwość zapisu zaszyfrowanych danych do pamięci flash, pamięci RAM oraz nie zapisaniu ich nigdzie.

4 Instrukcja użytkowania

Użytkownikowi zostały udostępnione funkcje szyfrowania danych, oraz odszyfrowania danych wcześniej zaszyfrowanych. Do pamięci flash zostały wcześniej wpisane przykładowe dane szyfrujące. Wykonanie zadań jest wynikiem wysłania odpowiednich komend przez port szeregowy. Dostępne są:

- **setclkres** [**res_symbol**] ustawia sposób odmierzania czasu testu, gdzie **res_symbol** to **cc**, **us** lub **ms**, co oznacza odpowiednio cykle zegarowe, mikrosekundy oraz milisekundy.
- **enc** [**size**] [**size_unit**] [**impl**] [**memtype**] gdzie **size** odpowiada za rozmiar, **size_unit** rozmiar bloku, do wyboru **block** oraz **kB**. Pole **impl** wybiera rodzaj algorytmu odpowiednio **acc**, **mbedtls** oraz **custom** odpowiada implementacji sprzętowej, implementacji z biblioteki mbedTLS oraz własnej implementacji. Pole **memtype** jest opcjonalne (domyślnie **discard**). Do wyboru mamy **flash**, **ram** oraz **discard** co daje zapis odpowiednio do pamięci RAM, do pamięci flash oraz brak zapisu. Przed zapisem do pamięci flash następuje wymazanie odpowiednich bloków tej pamięci.
- **dec** [**size**] [**size_unit**] [**impl**] [**memtype**] odszyfrowuje wcześniej zaszyfrowany tekst. Parametry jak wcześniej.
- **printplain** [**size**] [**size_unit**] pozwala na odczytanie tekstu niezaszyfrowanego. Parametry jak wcześniej.
- **printcipher** [**size**] [**size_unit**] pozwala na odczytanie zaszyfrowanego tekstu. Parametry jak wcześniej.
- **printdecipher** [**size**] [**size_unit**] pozwala na odczytanie tekstu odszyfrowanego. Parametry jak wcześniej.

Tablica 1: Porównanie dla zapisu do RAMu (cc - cykle zegara)

implementacja	1 blok	16 bloków
programowa	8674 cc	3895 us
sprzętowa	1080 cc	297 us

Tablica 2: Porównanie dla braku zapisu (cc - cykle zegara)

implementacja	1 blok	16 bloków	16kB	64kB
programowa	8662 cc	3889 us	246992 us	987880 us
sprzętowa	1069 cc	290 us	17584 us	70288 us

5 Implementacje

5.1 Programowe

Kod autorskiej implementacji algorytmu opisany jest w pliku *my_aes.c*. Większość instrukcji wykonywana jest na 8 bitowych słowach. Kod w czytelny sposób odwzorowuje algorytm AES-128-CBC, lecz nie jest tak dobrze zoptymalizowany jak implementacje dostępne np. w bibliotece mbedTLS. Implementacja została oparta o założenia z sekcji 2.

5.2 Sprzętowa

Wbudowany moduł szyfrowania sprzętowego wymaga, aby przed rozpoczęciem szyfrowania zapisać w rejestrach AES_KEYRx klucz i ustawić opcję szyfrowania (np. szyfrowanie, odszyfrowywanie). Następnie należy ustawić odpowiedni tryb szyfrowania (bity CHMOD, rejestr AES_CR) i uruchomić moduł (bit EN, rejest AES_CR). Moduł jest wtedy gotowy na przyjęcie danych do szyfrowania bądź deszyfrowania (rejestr AES_DINR). Stan szyfrowania jest dostępny w bicie CCF rejestru AES_SR. Moduł może też zgłaszać przerwania. Po zakończeniu obliczeń wynik dostępny jest w rejestrze AES_DOUTR. Obsługi użyto funkcji **HAL_CRYP_Init()**, *ACC_AES_RESET()* oraz *HAL_CRYP_AESECB_Encrypt()*.

6 Porównanie

W celu mierzenia czasu, jaki potrzebny jest na wykonanie szyfrowania stosowany jest timer. W zmiennej zapamiętywana jest wartość timera, w momencie rozpoczęcia procedury i porównywana z wartością timera w momencie zakończenia procedury. Porównanie zostało przeprowadzone przy optymalizacji O0.
Wykonano porównanie dla zapisu do pamięci RAM (tabela 1) oraz dla braku zapisu (tabela 2).

Jak widać przy wykorzystaniu akceleracji sprzętowej można uzyskać około ośmiokrotny wzrost wydajności dla szyfrowania jednego bloku, oraz około czternastokrotny wzrost wydajności dla tekstów dłuższych. Zapis do pamięci RAM nie powoduje drastycznego zmniejszenia wydajności.

Literatura

[1] "ADVANCED ENCRYPTION STANDARD (AES)." `https://doi.org/10.6028/NIST.FIPS.197`.

[2] "STM32L162RD)." https://my.st.com/content/my_st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32l1-series/stm32l162/stm32l162rd.html.