POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AIR)
SPECJALNOŚĆ: Komputerowe sieci sterowania (ARK)

PRACA DYPLOMOWA MAGISTERSKA

Badanie wydajności algorytmów z kluczem symetrycznym w bibliotece kryptograficznej WolfSSL na platformie STM32F2

Performance analysis of symmetric-key algorithms in WolfSSL cryptographic library on STM32F2 platform

AUTOR:
Michał Stroka

PROWADZĄCY PRACĘ:
dr inż. Grzegorz Budzyń, I-6

OCENA PRACY:

WROCŁAW 2017

Spis treści

Rozdział 1

Wstęp

Już w czasach starożytnych ludzie zaczeli dostrzegać potrzebę komunikacji między dwiema osobami bez możliwości przechwycenia i odczytania jej przez osoby trzecie. Najstarszym znanym szyfrem jest szyfr Cezara, którym Juliusz Gajusz Cezar w pierwszym wieku p.n.e. szyfrował korespondencję. Był to tak zwany szyfr przesuwający z przesunięciem 3. Każda litera „a" była zamieniana na „d", „b" na „e" itd. Oczywiście taki szyfr był bardzo łatwy do złamania, jednak w czasach gdy umiejętność czytania była rzadkością, zwłaszcza wśród przeciwników Rzymu, zupełnie wystarczał. Później przez całą historię trwał ciągły rozwój kryptologii - nauki o przekazywaniu informacji w sposób zabezpieczony przed niepowołanym dostęp. Ponieważ ludzie zdawali sobie sprawę że np. w czasie wojny zdobycie informacji o ruchach przeciwnika może przesądzić o losie bitwy, to szeregi matematyków wynajdowały nowe metody szyfrowania i jednocześnie usiłowały złamać te już istniejące. Prawdopodobnie najbardziej znaną maszyną szyfrującą była Enigma - niemiecka maszyna szyfrująca, która została złamana przez 3 polskich matematyków: Mariana Rejewskiego, Henryk Zygalski i Jerzego Różyckiego. Równolegle do zatajania treści wiadomości rozwijały się techniki ukrywania samego faktu wysyłania wiadomości. Na początku XX. wieku Niemcy dysponowali techniką, która pozwalała na sfotografowanie kartki rozmiaru A4 i pomniejszenie zdjęcia do rozmiaru pojedynczej kropki w tekście, a następnie mogła być za pomocą mikroskopu z powrotem powiększona. Tak przekazana książka z kilkoma tajnymi kropkami nie budziła podejrzeń nawet gdy zostałą przechwycona przez przeciwników.
W dzisiejszym świecie, gdy wraz z rozwojem i rozpowszechnieniem się komputerów tysiące informacji są przekazywane tajność korespondencji wydaje się być kluczowa. Obecnie problem dotyczy nie tylko najbardziej wpływowych osób na świecie i wymiany informacji pomiędzy nimi, ale każdego z nas. Niezależnie czy logujemy się do banku, wysyłamy maile/SMSy, przesyłamy pliki po sieci czy wpisujemy gdzieś nasze dane osobowe każdy z nas ma nadzieję, że te wrażliwe dane będą dostępne tylko dla adresata końcowego i nie zostaną zrozumiane przez osoby postronne. Dodatkowo wraz z rozwojem elektroniki coraz częściej mamy doczynienia z komunikacją między dwoma urządzeniami np. komunikacja między kartą płatniczą, a terminalem, czy wymiana informacji między podzespołami w autonomicznych pojazdach. Tutaj również ważne jest, aby komunikacja była tajna i nie była podmieniona, gdyż może spowodować to zmianę kwoty i/lub rachunku obciążanego w przypadku kart, i katastrofę w ruchu drogowym/lotniczym w przypadku pojazdów. Obecnie istnieje wiele standardowych szyfrów kryptograficznych, które według obecnej wiedzy są nie do złamania w rozsądnym czasie. Wydajność niektórych z nich zostanie zbadana w tej pracy.

Rozdział 2

Cel pracy

Celem pracy magisterskiej jest zbadanie wydajności algorytmów symetrycznych w bibliotece kryptograficznej WolfSSL na platformie STM32F2. Praca zakłada kompilację biblioteki na wybraną platformę, zaprojektowanie i stworzenie środowiska testowego pozwalającego na przeprowadzenie pomiarów w sposób łatwy i automatyczny, porównanie czasów wykonywania algorytmów tylko w procesorze oraz z użyciem wbudowanych procesorów kryptograficznych, a także próba poprawienia implementacji niektórych algorytmów w celu polepszenia ich wydajności.
Do pracy dołączona jest płyta z całym projektem.

Rozdział 3

Konfiguracja środowiska

Platformą testową, na której badana jest wydajność biblioteki WolfSSL to płytka testowa STM3221G-EVAL z mikroprocesorem STM32F217IG. Procesor ten posiada specjalnie dedykowane hardware'owe jednostki kryptograficzne, które pozwalają na znaczne przyspieszenie niektórych algorytmów dzięki sprzętowemu wykonywaniu części obliczeń. Algorytmy symetryczne, dla których wsparcie sprzętowe posiada wyżej wymieniony procesor, to AES, DES, MD5 i SHA-1. Wszystkie one mogą być wykorzystywane przez bibliotekę WolfSSL. W pracy zostaną porównane wydajności algorytmów z i bez użycia kryptojednostek oraz porównanie czasów z własną implementacją algorytmów.

Rysunek 3.1 Płytka testowa stm3221g-eval

3.1 Struktura katalogów w projekcie

Struktura katalogów w projekcie jest następująca

- ldscript
- libs
  - Drivers
    - CMSIS
    - STM32F2_HAL_Driver
  - wolfssl-3.9.10
- misc
- test_vectors
- workspace
  - src
  - inc
  - build

W folderze ldscripts znajdują się skrypty linkera. To one definiują m.in. ile pamięci RAM/flash jest dostępne w procesorze oraz definiują podział pamięci na stertę i stos.
W folderze libs, podkatalogu Drivers znajdują się dwie bilioteki procesora STM32F2. Stanowią one swoisty interfejs do części hardwerowej procesora. Pozwalają na zarządzanie np. tatktowania zegarem procesora, konfiguracją timerów, obsługą krpytoprocesorów (badany procesor ma specjalnie dedykowane kryptojednostki wykonujące niektóre algorytmy np. AES, SHA-1) czy portów GPIO.
W folderze libs, podkatalogu wolfssl-3.9.10 znajdują się pliki źródłowe badanej biblioteki wolfSSL oraz zbudowana biblioteka.
W folderze misc znajdują się różne pliki, które nie mają swoich predefiniowanych katalogów. Są tam pliki opisujące sposób biblioteki wolfSSL, czy konfiguracji połączenia między komputerem klasy PC, a płytką testową z mikrokontrolerem STM32.
Folder test_vectors zawiera dane testowe, które służą do badania wydajnośći poszczególnych implementacji algorytmów. Dzięki wspólnym danym wejściowym, wyniki mogą być bezpośrednio porównywane.
W Folderze workspace, w poczególnych podkatalogach znajdują się pliki źródłowe z moją własną implementacją poszczególnych algorytmów, pliki nagłówkowe, a w katalogu pliki powstające w momencie budowania całego projektu (pliki obiektowe oraz plik ELF, który jest wgrywany do procesora).

3.2 Instalowanie IDE do programowania STM32

W pracy zdecydowano się użyć toolchain'u dedykowanego do STM32 dla aplikacji bare metalowych (bez nadrzędnego systemu operacyjnego) GNU ARM Embedded Toolchain. Posiada on kompilator arm-gcc-none-eabi, który wspiera kompilację na większość platform z rodzin Cortex-M i Cortex-R, w szczególności na interesujący nas Cortex-M7. Posiada też

wbudowany debugger GDB (GNU Debugger), który pozwala podejrzeć co dzieje się wewnątrz programu (aktualne wartości zmiennych, rejestrów procesora, adresów w pamięci, licznik programu) oraz pozwala na sterowanie jego przebiegiem przez zatrzymywanie cyklu programu w kluczowych momentach przez brakepointy, czy przechodzenie „krok po kroku" między instrukjami.
Do zarządzania procesem budowania został użyty program make. Pozwala on na łatwe ustawianie reguł w jaki sposób należy budować poszczególne pliki, ustawia ścieżki do dołączania plików, czy wybór wersji biblioteki z której chcemy korzystać. Dzięki dobremu plikowi Makefile można posiadać wiele konfiguracji do budowania programu i wybierać odpowiednią za pomocą opcji wywołania programu make np. *make all LIBRARY=wolfssl_no_hardwarecrypto*

Do wgrywania zbudowanego program został użyty programator SEGGER, który jest już wbudowany w płytkę testową.

Ostantim używanym programem jest OpenOCD (Open On Chip Debugger). Za pomocą interfejsu JTAG dostępnego na płytce komunikuje się on z procesorem, umożliwia semihosting czyli przekazywanie danych z/do komputera w czasie pracy mikroprocesora.

3.3 Kompilowanie biblioteki

Pierwszym krokiem do zbudowania biblioteki jest wybór platformy, na której kod będzie wykonywany. W tym celu edytujemy plik *settings.h* znajdujące się w katalogu *wolfssl-3.9.10/wolfssl/wolfcrypt/*. Ponieważ naszą platformą jest STM32F2, to w pliku odkomentowujemy #define WOLFSSL_STM32F2 (linia 92). W późniejszej fazie pracy z biblioteką, okazało się że w pliku trzeba też usunąć #define KEIL_INTRINSICS (linia 766). WolfSSL w wersji 3.9.10 jawnie zakłada, że jedynym kompilatorem służącym do zbudowania biblioteki jest płatny kompilator Keil'a. Po usunięciu tej instrukcji dodającej funkcje kompilatora Keil, biblioteka kompiluje się również przy użyciu arm-gcc-none-eabi oraz dowolnych innych kompilatorów języka C na platformę STM32F2. W celu dalszej konfiguracji biblioteki powstał następujący skrypt bashowy:

```
CC=/c/GNU_Tools_ARM_Embedded/4.9_2015q3/bin/arm-none-eabi-gcc
AR=/c/GNU_Tools_ARM_Embedded/4.9_2015q3/bin/arm-none-eabi-ar
RANLIB=/c/GNU_Tools_ARM_Embedded/4.9_2015q3/arm-none-eabi/bin/ranlib
LDFLAGS=-lc -specs=nosys.specs"
CFLAGS=-mcpu=cortex-m3 -march=armv7-m -mthumb -mlittle-endian -DNO_WRITEV
-DWOLFSSL_USER_IO -DWOLFCRYPT_ONLY
-I/Drivers/CMSIS/Include
-I/Drivers/CMSIS/Device/ST/STM32F2xx/Include
-I/Drivers/STM32F2xx_StdPeriph_Lib_V1.1.0/Libraries/STM32F2xx_StdPeriph_Driver/src
-I/Drivers/STM32F2xx_HAL_Driver/Inc
-I/Drivers/STM32F2xx_HAL_Driver/Src
-I/Drivers/STM32F2xx_StdPeriph_Lib_V1.1.0/Libraries/STM32F2xx_StdPeriph_Driver/inc"
./configure -host=arm-none-eabi CC=$CC AR=$AR RANLIB=$RANLIB
LDFLAGS="$LDFLAGSĆFLAGS="$CFLAGS"
```

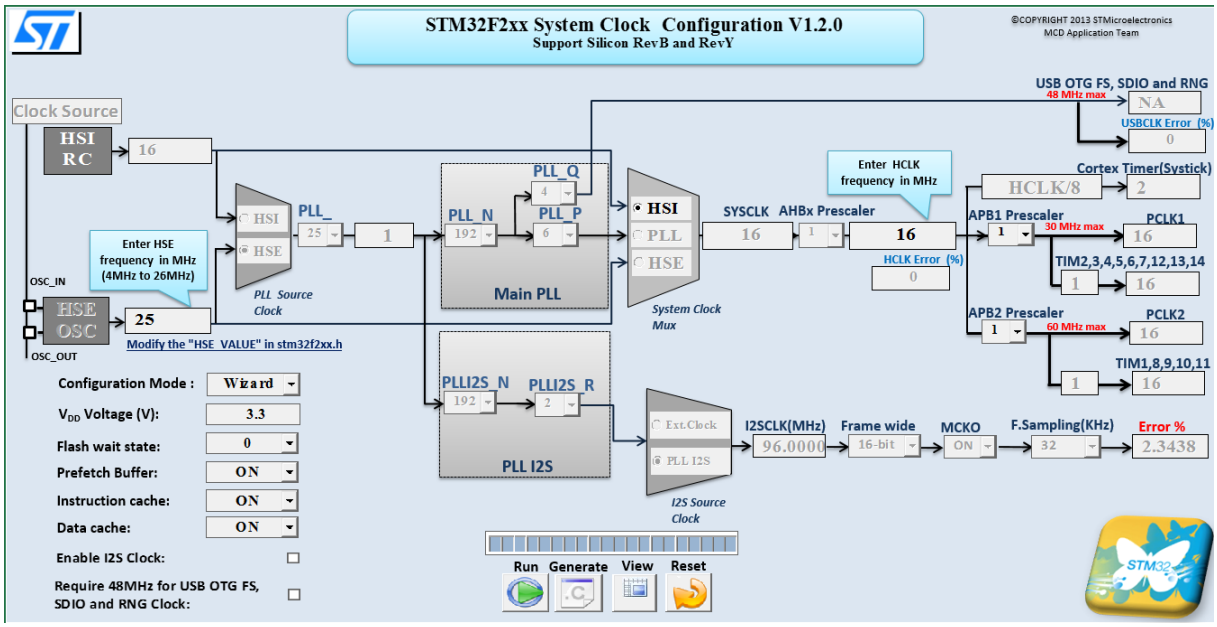
Zmienna CC przechowuje ścieżkę do kompilatora arm-none-eabi-gcc
Zmienna AR przechowuje ścieżkę do programu ar, służącego do tworzenia/modyfikowania/wyciągania danych z archiwum
Zmienna RANLIB przechowuje ścieżkę do programu ranlib, który tworzy indeksowaną listę symboli w archiwum

Zmienna LDFLAGS zawiera dodatkowe flagi dla linkera. Flaga -lc dodaje bibliotekę libc.a do kompilacji, flaga -specs=nosys.specs oznacza, że aplikacja będzie działała na platformie bez systemu operacyjnego i funkcje systemowe jak np. malloc muszą być dodane przez kompilator
Zmienna CFLAGS zawiera dodatkowe zmienne dla kompilatora. Zmienna -mcpu=cortex-m3 oznacza że program będzie wykonywany na platformie Cortex-M3, -march=armv7-m określa architekturę procesora jako ARMv7-M, -mthumb dodaje rozszerzony zestaw instrukcji asemblerowych Thumb, -mlittle-endian oznacza że zmienne w procesorze są przechowywane w formacie little endian (najmniej znaczący bit jest najmłodszy). Flagi -NO_WRITEV -WOLFSSL_USER_IO oznaczają, że w systemie nie ma żadnych instrukcji wejścia/wyjścia, a użytkownik posiada swoje funkcje IO (eng. input/output). Kolejne flagi zaczynające się od I/ zawierają ścieżki do plików nagłóWkowych i źródłowych bibliotek STM32F2.
Ostatnia linia skryptu wywołuje inny skrypt dostarczany przez WolfSSL, który przetwarza wszystkie dane i tworzy własny plik Makefile. Tak przygotowana biblioteka jest już prawie gotowa do kompilacji. Ostatnim krokiem jest poprawienie stworzonego pliku Makefile. Z flag kompilacji należy usunąć -fpie. Jest to flaga Position Independent Executable, która normalnie powoduje że kod biblioteki może być wpisany w dowolne miejsce w pamięci. Niestety w przyapdku platformy STM32F2 powodowało to, że w czasie wykonywania programu procesor próbował odczytywać nieistniejące rejestry pamięci, co koóńczyło się zawieszeniem procesora, przez uruchomienie procedury HardFault_handler. Znalezione rozwiązanie zostało przedstawione supportowi WolfSSL i została wydana nowa wersja biblioteki 3.10.0, która domyślnie nie wstawia tej flagi. Po dokonaniu zmiany poleceniem *make src/libwolfssl.la* kompilujemy bibliotekę do pliku libwolfssl.la.

Rozdział 4

Pomiar czasu

Pomiary czasu można wykonać na dwa podstawowe sposoby: przez zastosowanie portów GPIO, lub przez użycie wewnętrznych timerów procesora. Pierwszy sposób zakłada zmianę stanu napięcia w chwilach rozpoczęcia i zakończenia wykonywania algorytmu, a następnie dzięki podpięciu oscyloskopu pomiar częstotliwości zmian napięcia na wybranym pinie. Drugi sposób wykorzystuje timery procesora, który jest w stanie zliczać impulsy z częstotliwością równą częstotliwości taktowania procesora, czyli 96 MHz. W pracy zdecydowano się na drugi sposób z uwagi na łatwiejszą konfigurację projektu, mniej złożone stanowisko testowe oraz pełną automatyzację wykonywania pomiarów.



Rysunek 4.1 Schemat konfiguracji licznika

Konfiguracja wewnętrznych timerów została przeprowadzona za pomocą biblioteki HAL. Został wybrany 32-bitowy licznik zliczający przejścia zegara procesora z logicznego „0” na „1”. W ten sposób uzyskany został pomiar z dokładnością do $\frac{1}{48000000}$ sekundy i maksymalną mierzoną wartością $2^{32} * \frac{1}{48000000} \approx 90$ sekund. Do pomiaru czasu służą 3 proste funkcje: inicjalizująca timer, rozpoczynająca oraz kończąca pomiar.
Funkcja inicjująca najpierw wybiera timer2 (32-bitowy), ustawia preskaler oraz dzielnik

częstotliwości na 1 aby osiągnąć maksymalną dokładność pomiaru oraz ustawia kierunek zliczania impulsów w górę i makysmalną wartość timera na 0xFFFFFFFF:

```
void TimerInit( TIM_HandleTypeDef *timerHandle)
{
     timerHandle->Instance = TIM2;
     __TIM2_CLK_ENABLE();
     timerHandle->Init.Prescaler = 1;
     timerHandle->Init.CounterMode = TIM_COUNTERMODE_UP;
     timerHandle->Init.Period = 0xFFFFFFFF;
     timerHandle->Init.ClockDivision = 1;
}
```

Funkcja rozpoczynająca pomiar, wołana po inicjacji timera ustawia aktualną wartość zliczany impulsów:

```
void TimerStart(TIM_HandleTypeDef *timerHandle)
{
     timerHandle->Instance->CNT = 0;
}
```

Funkcja kończąca pomiar, wołana po inicjacji timera odczytuje aktualną ilość zliczonych impulsów i zwraca jej wartość:

```
uint32_t TimerStop(TIM_HandleTypeDef *timerHandle)
{
     return timerHandle->Instance->CNT;
}
```

Rozdział 5

Pomiary czasów poszczególnych algorytmów z biblioteki WolfSSL

Poszczeólne algorytmy były wykonywane dla reprezentacyjnych danych testowych. Taktowanie zegara wynosiło 96 MHz. Przezntowane wyniki zawierają dane każdego z 30 testów jak i czasy średnich wykonań algorytmów.

5.1 Algorytm SHA-1

SHA-1 to powstały w 1993 roku algorytm hashujący. Jako dane wejściowe pobiera wiadomość o maksymalnej długości 2^{64} bitów i zamienia je w 160-bitowy skrót. Tak jak wszystkie algorytmy haszujące, głównym zastosowaniem algorytmu SHA-1 jest weryfikacja inegralności plików lub wiadomości. Wraz z każdą wiadomością może być wysyłany jej skrót SHA-1. W momencie gdy adresat odbiera wiadomość oblicza jej skrót SHA-1. Jeżeli skróty są od siebie różne, to mamy pewność że któryś z bitów został źle odebrany i wiadomość nie może dalej być przetwarzana. Niestety w 2005 roku zostały opublikowane pierwsze ataki pozwalające złamać skrót w czasie krótszym niż atakiem typu BruteForce (sprawdzanie każdej możlwiości po kolei) i używanie tego algorytmu hashującego w nowych systemach nie jest zalecane.

5.1.1 Software'owa implementacja algorytmu

Algorytm SHA-1 składa się z trzech części: inicjalizacja danych, przetwarzanie wszystkich danych poza ostatnią częsią, przetworzenie ostatniej części. Dane przetwarzane są partiami po 32 bajty w każdym cyklu. W każdym cyklu na bajtach dokonywane są operacje logiczne: and i xor, a także operacja przesuwania bitowego. Dokładny opis algorytmu znajduje się w Dodatku A.
Badania przeprowadzono dla 30 wywołań algorytmu dla danych testowych o długościach: 32 bajtów, 64 bajtów i 1 kilobajta. Wyniki są dostępne na kolejnej stronie:

5.1.2 Hardware'owa implementacja algorytmu

Użyty procesor STM32f217IGH6 posiada specjalnie dedykowane kryptoprocesory do wykonania algorytmu SHA-1. Można się do niego dostać dzięki bibliotece STM32F2xx_StdPeriph_Driver, a konkretnie interfejsowi hardware'owemu dostępnemu

Nr	32 bajty			64 bajty			1024 bajty		
	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	690	0.014	2.226	1261	0.026	2.436	10004	0.208	4.913
2	691	0.014	2.223	1261	0.026	2.436	10015	0.209	4.908
3	691	0.014	2.223	1261	0.026	2.436	10004	0.208	4.913
4	690	0.014	2.226	1260	0.026	2.438	10015	0.209	4.908
5	691	0.014	2.223	1261	0.026	2.436	10005	0.208	4.913
6	690	0.014	2.226	1261	0.026	2.436	10014	0.209	4.908
7	690	0.014	2.226	1260	0.026	2.438	10005	0.208	4.913
8	690	0.014	2.226	1260	0.026	2.438	10015	0.209	4.908
9	691	0.014	2.223	1261	0.026	2.436	10004	0.208	4.913
10	690	0.014	2.226	1260	0.026	2.438	10015	0.209	4.908
11	690	0.014	2.226	1260	0.026	2.438	10004	0.208	4.913
12	690	0.014	2.226	1261	0.026	2.436	10015	0.209	4.908
13	691	0.014	2.223	1260	0.026	2.438	10004	0.208	4.913
14	691	0.014	2.223	1260	0.026	2.438	10014	0.209	4.908
15	691	0.014	2.223	1260	0.026	2.438	10005	0.208	4.913
16	691	0.014	2.223	1260	0.026	2.438	10015	0.209	4.908
17	690	0.014	2.226	1261	0.026	2.436	10005	0.208	4.913
18	690	0.014	2.226	1260	0.026	2.438	10015	0.209	4.908
19	690	0.014	2.226	1271	0.026	2.417	10005	0.208	4.913
20	691	0.014	2.223	1260	0.026	2.438	10015	0.209	4.908
21	691	0.014	2.223	1260	0.026	2.438	10005	0.208	4.913
22	690	0.014	2.226	1260	0.026	2.438	10015	0.209	4.908
23	690	0.014	2.226	1260	0.026	2.438	10005	0.208	4.913
24	690	0.014	2.226	1260	0.026	2.438	10016	0.209	4.907
25	690	0.014	2.226	1261	0.026	2.436	10004	0.208	4.913
26	690	0.014	2.226	1260	0.026	2.438	10016	0.209	4.907
27	690	0.014	2.226	1263	0.026	2.432	10004	0.208	4.913
28	690	0.014	2.226	1261	0.026	2.436	10007	0.208	4.912
29	690	0.014	2.226	1260	0.026	2.438	10004	0.208	4.913
30	690	0.014	2.226	1260	0.026	2.438	10016	0.209	4.907
Średnia	**690**	**0.014**	**2.225**	**1261**	**0.026**	**2.437**	**10010**	**0.209**	**4.911**

Tablica 5.1 Czasy dla algorytmu SHA-1

w pliku stm32f2xx_hash.c. Posiada on funkcje, które obsługują krpytoprocesor i biblioteka WolfSSL z nich korzysta. Czasy uzyskane przy pomocy wsparcia hardware'owego dla tych samych danych testowych są na kolejenj stronie.

5.1.3 Poprawiona hardware'owa implementacja algorytmu

Podczas analizy kodu biblioteki WolfSSL okazało się, że do zmiany końcowych wartośsci wyniku z formatu big-endian na little-endian wykorzystuje ona funkcję, która po kolei wyciąga bajty z liczby 32-bitowej i zamienia ich kolejność. Dzięki zastosowaniu asemblerowej instrukcji rev Rev Rn, Rm dokonującej operacji zamiany bajtów udało się lekko poprawić uzyskiwane czasy biblioteki

Nr	32 bajty			64 bajty			1024 bajty		
	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	254	0.005	6.047	318	0.007	9.660	2282	0.048	21.539
2	253	0.005	6.071	318	0.007	9.660	2292	0.048	21.445
3	254	0.005	6.047	318	0.007	9.660	2282	0.048	21.539
4	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
5	254	0.005	6.047	318	0.007	9.660	2282	0.048	21.539
6	253	0.005	6.071	318	0.007	9.660	2282	0.048	21.539
7	253	0.005	6.071	317	0.007	9.691	2282	0.048	21.539
8	253	0.005	6.071	317	0.007	9.691	2282	0.048	21.539
9	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
10	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
11	253	0.005	6.071	317	0.007	9.691	2282	0.048	21.539
12	253	0.005	6.071	317	0.007	9.691	2282	0.048	21.539
13	253	0.005	6.071	318	0.007	9.660	2282	0.048	21.539
14	253	0.005	6.071	317	0.007	9.691	2282	0.048	21.539
15	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
16	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
17	254	0.005	6.047	318	0.007	9.660	2282	0.048	21.539
18	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
19	253	0.005	6.071	318	0.007	9.660	2282	0.048	21.539
20	253	0.005	6.071	317	0.007	9.691	2282	0.048	21.539
21	254	0.005	6.047	317	0.007	9.691	2292	0.048	21.445
22	254	0.005	6.047	318	0.007	9.660	2282	0.048	21.539
23	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
24	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
25	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
26	253	0.005	6.071	318	0.007	9.660	2282	0.048	21.539
27	254	0.005	6.047	317	0.007	9.691	2282	0.048	21.539
28	254	0.005	6.047	318	0.007	9.660	2282	0.048	21.539
29	254	0.005	6.047	318	0.007	9.660	2282	0.048	21.539
30	254	0.005	6.047	318	0.007	9.660	2282	0.048	21.539
Średnia	**254**	0.**005**	**6.056**	**317**	**0.007**	**9.678**	**2283**	**0.048**	**21.533**

Tablica 5.2 Czasy dla algorytmu SHA-1 ze wsparciem sprzętowym

Nr	32 bajty			64 bajty			1024 bajty		
	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	249	0.005	6.169	308	0.006	9.974	2204	0.046	22.301
2	248	0.005	6.194	308	0.006	9.974	2214	0.046	22.201
3	249	0.005	6.169	308	0.006	9.974	2204	0.046	22.301
4	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
5	249	0.005	6.169	308	0.006	9.974	2204	0.046	22.301
6	248	0.005	6.194	308	0.006	9.974	2204	0.046	22.301
7	248	0.005	6.194	307	0.006	10.007	2204	0.046	22.301
8	248	0.005	6.194	307	0.006	10.007	2204	0.046	22.301
9	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
10	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
11	248	0.005	6.194	307	0.006	10.007	2204	0.046	22.301
12	248	0.005	6.194	307	0.006	10.007	2204	0.046	22.301
13	248	0.005	6.194	308	0.006	9.974	2204	0.046	22.301
14	248	0.005	6.194	307	0.006	10.007	2204	0.046	22.301
15	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
16	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
17	249	0.005	6.169	308	0.006	9.974	2204	0.046	22.301
18	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
19	248	0.005	6.194	308	0.006	9.974	2204	0.046	22.301
20	248	0.005	6.194	307	0.006	10.007	2204	0.046	22.301
21	249	0.005	6.169	307	0.006	10.007	2214	0.046	22.201
22	249	0.005	6.169	308	0.006	9.974	2204	0.046	22.301
23	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
24	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
25	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
26	248	0.005	6.194	308	0.006	9.974	2204	0.046	22.301
27	249	0.005	6.169	307	0.006	10.007	2204	0.046	22.301
28	249	0.005	6.169	308	0.006	9.974	2204	0.046	22.301
29	249	0.005	6.169	308	0.006	9.974	2204	0.046	22.301
30	249	0.005	6.169	308	0.006	9.974	2204	0.046	22.301
Średnia	**249**	**0.005**	**6.178**	**307**	**0.006**	**9.992**	**2205**	**0.046**	**22.295**

Tablica 5.3 Czasy poprawionego algorytmu SHA-1 ze wsparciem sprzętowym

5.2 Algorytm DES

Algorytm DES (eng. Data Encryption Standard) jest symetrycznym szyfrem blokowym zapoczątkowanym przez IBM w 1975 r. Algorytm mając klucz i wektor inicjujący zamienia 8 bajtów wejściowych w 8 bajtów wyjściowych, które mogą następnie być z powrotem odszydrowane o ile adresat zna klucz użyty do zaszyfrowania wiadomości. Z matematycznego punktu widzenia DES jest funkcją różnowartościową która każdej wartości z dziedziny funkcji $D \in (0, 2^{64} - 1)$ przypisuje jedną wartość z jej przeciwdziedziny $Y \in (0, 2^{64} - 1)$.
Z uwagi na małą długość klucza - tylko 56 bitów algorytm ten uznawany jest za podatny na ataki typu brute force.

5.2.1 Software'owa implementacja algorytmu

Pierwszym krokiem jest generowanie 16 rund kluczy na podstawie jednego klucza głównego. Następnie algorytm przetwarza dane w 8-bajtowych porcjach. Każda porcja jest podawana operacjom matematycznym takim jak permutacja bitów, operacja bitowa xor, dokonywanie podstawień na podstawie S-boxy i P-boxy (są to takie „czarne skrzynki" które zamieniają liczby S-box z liczby z zakresu 0-63 na liczbę z zakresu 0-15, a P-box z 0-31 na liczbę 0-31, oraz funcje Feistela.
Dokładny opis algorytmu znajduje się w dodatku B. Czasy szyfrowania danych z użyciem algorytmu DES przedstawia tabela na kolejnej stronie.

5.2.2 Hardware'owa implementacja algorytum

Podobnie jak w przypadku SHA-1 także DES posiada swój koprocesor kryptograficzny na pokładzie STM32f217IGH6. Dzięki zastosowaniu funkcji znajdujących się w plikach stm32f2xx_cryp.c i stm32f2xx_cryp_des.c uzyskujemy interfejs pozwalający na korzystanie z koprocesora. Biblioteka WolfSSL oferuje wsparcie dla koprocesora (przez ustawienie flagi STM32_CRYPT przy kompilacji) co pozwala na znaczące przyspieszenie wykonywanych obliczeń. Wyniki pomiarów czasu zaprezentowano w tablei na kolejnej stronie.

Nr	Generowanie kluczy		8 bajtów			16 bajtów			1024 bajty		
	licznik	czas [ms]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	10155	0.212	434	0.009	0.885	843	0.018	0.911	52261	1.089	0.941
2	10155	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
3	10155	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
4	10155	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
5	10155	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
6	10155	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
7	10155	0.212	435	0.009	0.883	844	0.018	0.910	52262	1.089	0.940
8	10155	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
9	10165	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
10	10155	0.212	435	0.009	0.883	855	0.018	0.898	52260	1.089	0.941
11	10167	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
12	10155	0.212	435	0.009	0.883	843	0.018	0.911	52260	1.089	0.941
13	10166	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
14	10155	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
15	10166	0.212	435	0.009	0.883	844	0.018	0.910	52262	1.089	0.940
16	10155	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
17	10168	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
18	10155	0.212	435	0.009	0.883	844	0.018	0.910	52262	1.089	0.940
19	10169	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
20	10155	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
21	10169	0.212	435	0.009	0.883	844	0.018	0.910	52272	1.089	0.940
22	10155	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
23	10168	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
24	10155	0.212	435	0.009	0.883	844	0.018	0.910	52273	1.089	0.940
25	10169	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
26	10155	0.212	435	0.009	0.883	844	0.018	0.910	52260	1.089	0.941
27	10168	0.212	435	0.009	0.883	844	0.018	0.910	52271	1.089	0.940
28	10155	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
29	10167	0.212	435	0.009	0.883	844	0.018	0.910	52261	1.089	0.941
30	10155	0.212	435	0.009	0.883	844	0.018	0.910	52271	1.089	0.940
Średnia	**10160**	**0.212**	**435**	**0.0091**	**0.883**	**844**	**0.018**	**0.910**	**52262**	**1.089**	**0.940**

Tablica 5.4 Czasy dla algorytmu DES

Nr	Generowanie kluczy		8 bajtów			16 bajtów			1024 bajty		
	licznik	czas [ms]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	34	0.001	250	0.005	1.536	298	0.006	2.577	6220	0.130	7.902
2	34	0.001	251	0.005	1.530	298	0.006	2.577	6230	0.130	7.890
3	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
4	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
5	34	0.001	251	0.005	1.530	309	0.006	2.485	6230	0.130	7.890
6	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
7	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
8	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
9	34	0.001	251	0.005	1.530	298	0.006	2.577	6229	0.130	7.891
10	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
11	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
12	34	0.001	251	0.005	1.530	298	0.006	2.577	6230	0.130	7.890
13	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
14	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
15	34	0.001	251	0.005	1.530	298	0.006	2.577	6230	0.130	7.890
16	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
17	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
18	34	0.001	251	0.005	1.530	298	0.006	2.577	6230	0.130	7.890
19	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
20	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
21	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
22	34	0.001	251	0.005	1.530	309	0.006	2.485	6220	0.130	7.902
23	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
24	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
25	34	0.001	251	0.005	1.530	298	0.006	2.577	6230	0.130	7.890
26	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
27	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
28	34	0.001	251	0.005	1.530	298	0.006	2.577	6230	0.130	7.890
29	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
30	34	0.001	251	0.005	1.530	298	0.006	2.577	6219	0.130	7.904
Średnia	**34**	**0.001**	**251**	**0.005**	**1.530**	**299**	**0.006**	**2.571**	**6222**	**0.130**	**7.900**

Tablica 5.5 Czasy dla algorytmu DES ze wsparciem sprzętowym

5.3 Algorytm AES

AES (eng. Advanced Encryption Standard) jest symetrycznym szyfrem blokowym, który przetwarza dane w paczkach po 16 bajtów każda. Podobnie jak DES też składa się z części obliczającej klucze rundy i z części kodującej wiadomość. Algorytm jest uznawany za bezpieczny i jest zalecany zamiast algorytmu DES, który przez krótki klucz jest łatwy do złamania.

5.3.1 Software'owa wydajność algorytmu

Pierwszym krokiem jest wygenerowanie kluczy 11/13/15 kluczy rund, następnie dane są kodowane w 10/12/14 rundach (pierwszy klucz dodawany jest na samym początku). Na danych dokonywane są takie operacje jak logiczny xor, cykliczne obracanie bajtów/bitów, podstawianie bajtów. Dokładny opis algorytmu przedstawiony jest w dodatku C. Czasy szyfrowania danych z użyciem algorytmu AES przedstawaiają tabele na kolejnych stronach.

Nr	Generowanie kluczy		16 bajtów			32 bajtów			1024 bajty		
	licznik	czas [ms]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	260	0.0054	461	0.0096	1.666	890	0.019	1.726	27645	0.576	1.778
2	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
3	261	0.0054	462	0.0096	1.662	890	0.019	1.726	27647	0.576	1.778
4	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27656	0.576	1.777
5	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
6	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
7	260	0.0054	462	0.0096	1.662	900	0.019	1.707	27645	0.576	1.778
8	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
9	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
10	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
11	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
12	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
13	260	0.0054	461	0.0096	1.666	890	0.019	1.726	27657	0.576	1.777
14	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
15	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
16	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
17	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
18	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
19	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27647	0.576	1.778
20	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
21	261	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
22	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27656	0.576	1.777
23	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
24	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
25	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
26	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
27	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27657	0.576	1.777
28	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
29	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27645	0.576	1.778
30	260	0.0054	462	0.0096	1.662	890	0.019	1.726	27646	0.576	1.778
Średnia	**260**	**0.0054**	**462**	**0.0096**	**1.663**	**890**	**0.019**	**1.725**	**27647**	**0.576**	**1.778**

Tablica 5.6 Czasy dla algorytmu AES dla 10-ciu rund

Nr	Generowanie kluczy		16 bajtów			32 bajtów			1024 bajty		
	licznik	czas [ms]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
2	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
3	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
4	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32176	0.670	1.528
5	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
6	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32176	0.670	1.528
7	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32166	0.670	1.528
8	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
9	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32175	0.670	1.528
10	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32166	0.670	1.528
11	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32175	0.670	1.528
12	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
13	258	0.0054	533	0.011	1.441	1039	0.022	1.478	32166	0.670	1.528
14	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32175	0.670	1.528
15	249	0.0052	533	0.011	1.441	1050	0.022	1.463	32165	0.670	1.528
16	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32175	0.670	1.528
17	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
18	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
19	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32175	0.670	1.528
20	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
21	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32175	0.670	1.528
22	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32166	0.670	1.528
23	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
24	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32175	0.670	1.528
25	249	0.0052	544	0.011	1.412	1040	0.022	1.477	32165	0.670	1.528
26	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32176	0.670	1.528
27	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32165	0.670	1.528
28	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32164	0.670	1.528
29	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32175	0.670	1.528
30	249	0.0052	533	0.011	1.441	1039	0.022	1.478	32166	0.670	1.528
Średnia	**249**	**0.0052**	**533**	**0.011**	**1.440**	**1039**	**0.022**	**1.478**	**32169**	**0.670**	**1.528**

Tablica 5.7 Czasy dla algorytmu AES dla 12-stu rund

Nr	Generowanie kluczy		16 bajtów			32 bajtów			1024 bajty		
	licznik	czas [ms]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	320	0.0067	603	0.013	1.274	1178	0.025	1.304	36683	0.764	1.340
2	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36695	0.764	1.339
3	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36685	0.764	1.340
4	320	0.0067	604	0.013	1.272	1178	0.025	1.304	36696	0.765	1.339
5	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36696	0.765	1.339
6	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
7	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36695	0.764	1.339
8	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
9	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36695	0.764	1.339
10	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
11	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
12	320	0.0067	603	0.013	1.274	1179	0.025	1.303	36694	0.764	1.340
13	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
14	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36696	0.765	1.339
15	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
16	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
17	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36695	0.764	1.339
18	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
19	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36695	0.764	1.339
20	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
21	320	0.0067	604	0.013	1.272	1189	0.025	1.292	36684	0.764	1.340
22	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36695	0.764	1.339
23	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
24	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36695	0.764	1.339
25	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
26	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36683	0.764	1.340
27	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36694	0.764	1.340
28	320	0.0067	604	0.013	1.272	1179	0.025	1.303	36684	0.764	1.340
29	319	0.0066	604	0.013	1.272	1179	0.025	1.303	36694	0.764	1.340
30	319	0.0066	604	0.013	1.272	1189	0.025	1.292	36684	0.764	1.340
Średnia	320	0.0067	604	0.013	1.272	1180	0.025	1.302	36689	0.764	1.340

Tablica 5.8 Czasy dla algorytmu AES dla 14-stu rund

Nr	Generowanie kluczy		16 bajtów			32 bajtów			1024 bajty		
	licznik	czas [ms]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	70	0.0015	294	0.006	2.612	365	0.008	4.208	4752	0.099	10.343
2	70	0.0015	294	0.006	2.612	365	0.008	4.208	4763	0.099	10.320
3	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
4	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
5	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
6	70	0.0015	294	0.006	2.612	365	0.008	4.208	4764	0.099	10.317
7	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
8	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
9	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
10	70	0.0015	294	0.006	2.612	365	0.008	4.208	4764	0.099	10.317
11	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
12	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
13	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
14	70	0.0015	294	0.006	2.612	365	0.008	4.208	4767	0.099	10.311
15	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
16	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
17	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
18	70	0.0015	294	0.006	2.612	365	0.008	4.208	4767	0.099	10.311
19	70	0.0015	294	0.006	2.612	376	0.008	4.085	4753	0.099	10.341
20	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
21	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
22	70	0.0015	294	0.006	2.612	365	0.008	4.208	4766	0.099	10.313
23	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
24	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
25	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
26	70	0.0015	294	0.006	2.612	365	0.008	4.208	4767	0.099	10.311
27	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
28	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
29	70	0.0015	294	0.006	2.612	365	0.008	4.208	4753	0.099	10.341
30	70	0.0015	294	0.006	2.612	365	0.008	4.208	4767	0.099	10.311
Średnia	70	0.0015	294	0.006	2.612	365	0.008	4.204	4756	0.099	10.334

Tablica 5.9 Czasy dla algorytmu AES dla 10-ciu rund ze wsparciem sprzętowym

Nr	Generowanie kluczy		16 bajtów			32 bajtów			1024 bajty		
	licznik	czas [ms]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	80	0.0017	295	0.006	2.603	367	0.008	4.185	4754	0.099	10.339
2	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
3	80	0.0017	295	0.006	2.603	378	0.008	4.063	4755	0.099	10.337
4	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
5	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
6	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
7	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
8	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
9	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
10	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
11	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
12	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
13	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
14	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
15	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
16	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
17	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
18	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
19	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
20	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
21	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
22	80	0.0017	295	0.006	2.603	367	0.008	4.185	4754	0.099	10.339
23	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
24	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
25	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
26	80	0.0017	295	0.006	2.603	367	0.008	4.185	4754	0.099	10.339
27	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
28	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
29	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
30	80	0.0017	295	0.006	2.603	367	0.008	4.185	4755	0.099	10.337
Średnia	80	0.0017	295	0.006	2.603	367	0.008	4.181	4755	0.099	10.337

Tablica 5.10 Czasy dla algorytmu AES dla 12-stu rund ze wsparciem sprzętowym

Nr	Generowanie kluczy		16 bajtów			32 bajtów			1024 bajty		
	licznik	czas [ms]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]	licznik	czas [ms]	prędkość [MB/s]
1	92	0.0019	295	0.006	2.603	366	0.008	4.197	4768	0.099	10.309
2	91	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
3	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
4	91	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
5	92	0.0019	295	0.006	2.603	366	0.008	4.197	4768	0.099	10.309
6	91	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
7	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
8	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
9	92	0.0019	295	0.006	2.603	366	0.008	4.197	4766	0.099	10.313
10	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
11	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
12	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
13	91	0.0019	295	0.006	2.603	366	0.008	4.197	4765	0.099	10.315
14	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
15	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
16	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
17	91	0.0019	295	0.006	2.603	366	0.008	4.197	4764	0.099	10.317
18	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
19	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
20	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
21	91	0.0019	295	0.006	2.603	366	0.008	4.197	4765	0.099	10.315
22	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
23	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
24	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
25	92	0.0019	295	0.006	2.603	366	0.008	4.197	4765	0.099	10.315
26	91	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
27	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
28	92	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
29	92	0.0019	295	0.006	2.603	366	0.008	4.197	4765	0.099	10.315
30	91	0.0019	295	0.006	2.603	366	0.008	4.197	4754	0.099	10.339
Średnia	92	0.0019	295	0.006	2.603	366	0.008	4.197	4757	0.099	10.332

Tablica 5.11 Czasy dla algorytmu AES dla 14-stu rund ze wsparciem sprzętowym

Rozdział 6

Problemy przy realizacji zadania

Największym problemem przy realizacji zadania okazały się być kompilacja i uruchomienie biblioteki WolfSSL. Już na samym początku okazało się, że pomimo biblioteka wspiera wiele platdorm w tym STM32F2, to jednak jedynym oficjalnym komilatorem mogącym ją skompilować jest kompilator Keil'a. Dodanie opcji kompilacji przez arm-none-eabi-gcc wymagało dodatkowej modyfikacji plików nagłówkowych biblioteki.
Nawet po modyfikacji nagłówków, zkonfigurowaniu i zbudowaniu biblioteki a następnie uruchomieniu jej na płytce okazało się, że przy wywoływaniu wielu funkcji program przestaje działać. Po włączeniu trybu debuggowania i sprawdzeniu kodu asemblerowego okazywało się, że problem pojawia się w momencie wywoływania funkcji toupper. Jest to funkcja z poza biblioteki WolfSSL, która była w niej użyta to zamiany liter w kluczu z małych na wielkie (np. z klucza „1a2b3c4d" na klucz „1A2B3C4D"). W momencie wywoływania następował skok licznika programu na adres poza dozwoloną przestrzenia adresową. Rozwiązaniem problemu była edycja automatycznie generowanego pliku Makefile i usunięcia z niego flagi -fPIE.
Wszystkie te problemy były rozwiązywane we współpracy z zespołem wsparcia technicznego od WolfSSL, co z uwagi na różnice czasu między Polską a Stanami Zjednoczonymi było bardzo czasochłonne. Tutaj pojawia się druga słaba stona biblioteki - tydzień po znalezieniu rozwiązania została wypuszczona nowa wersja oprogramowania zawierająca tą i parę innych małych poprawek. Ogólnie cała struktura katalogowa biblioteki przypomina zbiór łat, które ktoś próbuje ze sobą połączyć. Zaskakuje nawet brak wspólnego systemu nazewnictwa funkcji. Istnieje np. funkcja wc_Des_SetKey - ustawiająca klucz algorytmu DES i funkcja wc_AesSetKey - bez podkreślenia między AES i Set - ustawiająca klucz algorytmu AES. Brak konsekwencji w nazewnictwie sprawia że korzystanie z biblioteki jest nieintuicyjne.
Ostatnim problemem z biblioteką jest to, że nawet zdefiniowanie flagi -DWOLFSSL_STM32F2 nie uruchamia całego wsparcia WolfSSL'a dla STM32F2. Dodatkowo konieczne było zdefiniowanie -DSTM32F2_HASH.
Kolejnym problemem był początkowy brak znajomości architektury badanej platformy. Na początku zadania nie zdawałem sobie sprawy, że badany procesor, tak jak wszystkie procesory z rodziny STM32F2, posiada krpytoprocesory znacząco przyspieszające działanie algorytmów szeregowych. Dopiero dodanie do projektu biblioteki Standard Peripheral Library pozwoliło na odwołanie się do specjalnych jednostek kryptograficznych przyspieszających obliczenia.

Rozdział 7

Zakończenie

Budowa środowiska testowego z właściwą strukturą katalogów nie jest sprawą prostą, jednak cel udało się osiągnąć. Dzięki przejrzstemu podziałowi projektu na poszczególne foldery i podfoldery badające poszczególne moduły, rozbudowanie środowiska o badanie kolejnych algorytmów jest możliwie proste.
Otrzymane wyniki czasów wykonań algorytmów są zgodne z tymi przedstawionymi w benchmarkach na stronie WolfSSL'a i pomimo że są absolutnie czołowe wśród bibliotek opensource'owych, ja nie poleciłbym jej większości użytkowników. Czasy biblioteki można minimalnie poprawić edytując jej kod źródłowy, jednak są te różnice na poziomie poniżej 5%
Biblioteka okazała się nie być dokońca przenaszalna na wszystkie platformy, a aby w pełni wykorzystać jej potencjał konieczne jest dokładne zgłębienie kodów źródłowych biblioteki. W szczególności opcja użycia koprocesorów hashujących była dobrze ukryta, w dokumentacji w rozdiale o kompilowaniu biblioteki nie było o tej opcji ani słowa.
Wśród wyników warto zwrócić uwagę na parę faktów. Po pierwsze prędkości hashowania/kodowania wiadomości maleją wraz ze wzrostem ilości danych. Dzieje się tak dlatego, że wsszelkie dane inicjalizujące wykonywane są tylko raz i większa część czasu może być przeznaczona na kodowanie wiadomości. Wnioski są takie, że przy zastosowaniach praktycznych algorytmów kryptograficznych dane należy najpierw zbierać, a gdy będzie ich możliwe dużo zakodować i wysłać.
Kolejnym faktem jest to, że w algorytmach szeregowych DES/AES stosunkowo dużo czasu zajmuje algorytm rozszerzania klucza. Z uwagi, że jest to operacja wykonywana tylko raz dla każdego klucza czas potrzebny na zakodowanie pierwszej partii danych może być znacząco większy niż czas na zakodowanie reszty danych. W szczególności w algorytmie DES możemy zauważyć, że czas na generowanie kluczy starczyłby na zakodowanie 200 bajtów danych.
Na koniec warto zauważyć ogromną przewagę kryptoprocesorów, pozwalającą im dokonywać obliczeń z 7-8 razy większą wydajnością, niż dla obliczeń w samym CPU. Różnica ta wynika z operacji jakie wykonują algorytmy szeregowe. Od samego początku były one projektowane tak, aby ich hardwareowa implementacja była możliwie prosta. Dzięki temu użytkownik może uruchmić procedurę hashującą jakąś strukturę danych o wielkości setek MegaBajtów, która w ogóle nie będzie zajmowała czasu procesora.

Dodatek A

SHA-1

Pseudokod algorytmu hashującego wygląda następująco:
Wartości początkowe:
    h0 := 0x67452301
    h1 := 0xEFCDAB89
    h2 := 0x98BADCFE
    h3 := 0x10325476
    h4 := 0xC3D2E1F0
Przetwarzanie wstępne:
dopisz '1' do wiadomości;
dopisz k '0', gdzie k jest najmniejszą liczbą taką, że długość nowej wiadomości modulo 512 wynosi 448.
Dopisz długość wiadomości w bitach (przed wypełnieniem) jako 64-bitową liczbę całkowitą zakodowaną big endian (w ten sposób długość całej wiadomości w bitach jest podzielna przez 512).
Przetwarzaj wiadomość 512-bitowymi porcjami:
for (każda porcja)
    podziel porcję na 16 32-bitowych słów kodowanych big-endian w(i), 0 <= i <= 15
    Rozszerz 16 32-bitowych słów w 80 32-bitowych słów:
    for i from 16 to 79
        w(i) := (w(i-3) xor w(i-8) xor w(i-14) xor w(i-16)) «< 1
    Zainicjuj zmienne dla tej porcji:
    a := h0
    b := h1
    c := h2
    d := h3
    e := h4
    Główna pętla:
    for i from 0 to 79
        if 0 <= i <= 19 then
            f := (b and c) or ((not b) and d)
            k := 0x5A827999
        else if 20 <= i <= 39
            f := b xor c xor d
            k := 0x6ED9EBA1
        else if 40 <= i <= 59
            f := (b and c) or (b and d) or (c and d)

```
            k := 0x8F1BBCDC
        else if 60 <= i <= 79
            f := b xor c xor d
            k := 0xCA62C1D6
        temp := (a «< 5) + f + e + k + w(i)
        e := d
        d := c
        c := b «< 30
        b := a
        a := temp
    Dodaj skrót tej porcji do dotychczasowego wyniku:
        h0 := h0 + a
        h1 := h1 + b
        h2 := h2 + c
        h3 := h3 + d
        h4 := h4 + e
Wytwórz ostateczną wartość skrótu (zakodowaną big-endian):
    skrót = h0 dopisz h1 dopisz h2 dopisz h3 dopisz h4
```

Dodatek B

DES

Pseudokod algorytmu wygląda następująco:

1. Tworzenie szesnastu kluczy rund:
Z klucza 64-bitowego za pomocą tablicy PC-1 stwórz klucz 56-bitowy
Podziel klucz na dwie 28-bitowe części: C_0 i D_0
Stwórz C_i i D_i dla i = 1 do 15. Każda część C_i i D_i powstają odpowiednio z części C_{i-1} i D_{i-1} przesuniętej w lewo o 1 lub 2 bity (wartość podana w standardzie)
Z pary C_iD_i stwórz i-ty klucz rundy K_i. Klucz uzyskuje się przez permutację pary C_iD_i zgodnie z tablicą PC-2. Każdy klucz rundy ma 48 bitów.

2. Po wygenerowaniu kluczy można zacząć kodować dane w porcjach po 8 bajtów każda.
Przepermutuj bity wiadomości zgodnie z tabelą IP.
Podziel nową wiadomość an dwie 32-bitowe części: L_0 i R_0.
Oblicz części L_i i R_i dla 1 <= i <= 16 wg. wzoru:

$$L_n = R_{n-1},\ R_n = L_{n-1} \oplus \mathrm{f}(R_{n-1},\ K_n)$$

Po otrzymaniu wszystkich L i R łączymy w jedną całość $R_{16}L_{16}$, w tej kolejności, i dokonujemy ostatecznej permutacji bitów zgodnie z tabelą IP^{-1}.
Pozostaje jeszcze do opisania w jaki sposób oblicza się wartość funkcji Feistela: f(R_{n-1}, K_n).
Oblicz wartość E(R_{n-1}) dokonując przestawień bitów w R_{n-1} zgodnie z tabelą E. To przestawienie powoduje rozszerzenie z 32 bitów w R_{n-1} do 48 bitów w E(R_{n-1}).
Oblicz wartość E(R_{n-1}) $\oplus$ K_n.
Podziel otrzymaną wartość na osiem 6-bitowych bloków od B_1 do B_8.
Następnie korzystając z tabel zwanych S-Boxami dokonujemy zamiany wartości B_i na $S_i(S_i)$. Każda wartość w S-Boxach ma tylko 4 bity dlatego w rezultacie otrzymane wyrażenie $S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ ma tylko 32 bity.
Ostatnim krokiem jest dokonanie permutacji uzyskanych bitów za pomocą tabeli P.

Dodatek C

AES

Pseudokod algorytmu wygląda następująco:

1. Rozszerzenie klucza z N = 16/24/32 bajtów do B = 176/208/240 bajtów w zależności od ilości rund wygląda następująco:

    Pierwsze N bajtów jest normalnie przepisywanych
    iloscBajtow = N
    nrRundy = 1
    dopóki iloscBajtów < N:
    for (i = 0; i < 4; i++)
        t[a] = klucz[i + iloscBajtow - 4]
    if (iloscBajtow % (16/24/32) == 0)
        PrzeliczKluczRundy(t,nrRundy)
        nrRundy++
    for (i = 0; i < 4; i++)
        klucz[iloscBajtow] = klucz[iloscBajtow - 16] $\oplus$ t[i]
        iloscBajtow++

Operacja PrzeliczKluczRundy(t, nrRundy) wygląda następująco:

    Obróć t cyklicznie 8 bitów w lewo
    for (i = 0; i < 4; i++)
        t[i] = sBox[t[i]]
    t[0] = t[0] $\oplus$ rcon(i)

sBox i rcon to tablice z wartościami zawartymi w standardzie.

2. Po wygenerowaniu kluczy wszystkich rund zaczyna się kodowanie danych w porcjach po 16 bajtów każda:

    Operacja XOR danych wejściowych z pierwszym kluczem
    for (nrRundy = 1; nrRundy < (9/11/13); nrRundy++)
        Dane (16 bajtów) przedstaw jako tablicę dwuwymiarową o wymiarach 4 bajty na 4 bajty
        Dokonaj operacji podstawienia bajtów: bajt = f(bajt)
        Dokonaj operacji cyklicznego obracania wierszy:
            drugi wiersz 1 bajt w lewo
            trzeci wiersz 2 bajty w lewo
            czwarty wiersz 3 bajty w lewo

Każdą kolumnę tablicy pomnóż przez stałą macierz w polu Galoiss
Do wyniku dodaj modulo 2 klucz rundy

Na koniec wykonywana jest jeszcze jedna runda, w której pomijana jest operacja mnożenia przez macierz.

Bibliografia

[1] Instrukcja instalacji toolchain'a gcc,
http://gnuarmeclipse.github.io/toolchain/install/

[2] Opis dodawania instrukcji asemblerowych do kodu w C dla arm-none-eabi-gcc,
http://www.ethernut.de/en/documents/arm-inline-asm.html

[3] Poradnik do kompilacji WolfSSL na dowolną platformę,
https://www.wolfssl.com/wolfSSL/Docs-wolfssl-porting-guide.html

[4] Bencharki biblioteki WolfSSL,
https://www.wolfssl.com/wolfSSL/benchmarks-wolfssl.html

[5] Dokumentacja biblioteki Standard Peripheral Library,
http://www.st.com/content/ccc/resource/technical/document/user_manual
/59/2d/ab/ad/f8/29/49/d6/DM00023896.pdf/files/DM00023896.pdf

[6] Technische Universität Berlin: Opis algorytmu DES,
http://page.math.tu-berlin.de/ kant/teaching/hess/krypto-ws2006/des.htm

[7] National Institute of Standards and Technology: Opis algorytmu AES,
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[8] Szablon pracy ze strony dr Adama Ratajczaka,
http://rab.ict.pwr.wroc.pl/ ar/LaTeX/

Spis rysunków

Spis tablic