STUDIA INFORMATICA 2011
Volume 32 Number 4A (100)

Jacek LACH
Politechnika Śląska, Instytut Informatyki

PRZEGLĄD ALGORYTMÓW SZYFRUJĄCYCH DLA SYSTEMÓW O OGRANICZONYCH ZASOBACH

Streszczenie. W artykule przedstawiono przegląd dostępnych algorytmów szyfrujących, które mogą być efektywnie wykorzystane w środowisku o ograniczonych zasobach systemowych, takich jak mikrokontrolery jednoukładowe. Nacisk położono na możliwość praktycznego zastosowania algorytmów podczas ich programowej implementacji. Dokonano również przeglądu algorytmów zorientowanych na implementację sprzętową pod kątem możliwości ich wykorzystania w rozwiązaniach programowych.

Słowa kluczowe: kryptografia, mikrokontrolery jednoukładowe

A SURVEY OF CRYPTOGRAPHY ALGORITHMS FOR SYSTEMS WITH LIMITED RESOURCES

Summary. This article surveys cryptography algorithms that can be effectively used in systems built with limited resources such as microcontrollers. The most important aspect taken into consideration was its practical usefulness for software implementation. The survey of algorithms designed for hardware implementation which can be effectively implemented in software was also conducted.

Keywords: cryptography, microcontrollers

1. Wprowadzenie

Bezpieczeństwo informacji odgrywa istotną rolę w procesie jej przetwarzania i składowania. Typowe zadania związane z zapewnieniem bezpieczeństwa informacji to: poufność, integralność, uwierzytelnianie oraz niezaprzeczalność. Zadania te są realizowane z wykorzystaniem mechanizmów kryptograficznych ogólnie znanych i powszechnie uważanych za

wolne od potencjalnych luk drastycznie obniżających poziom bezpieczeństwa oferowany przez konkretne rozwiązanie w przypadku wykorzystania różnych form ataku zarówno na sam algorytm, jak i na implementacje tego algorytmu. Rozwiązania obecne stosowane są odporne na ataki z wykorzystaniem kryptoanalizy liniowej, kryptoanalizy różnicowej, ataki na implementacje i pomiary mocy podczas wykonywania ataku (*ang. simple power attack, differential power attack*). W zależności od wartości chronionych danych i czasu ich ważności istotna jest również odporność na realizację ataku z użyciem przeszukiwania całej przestrzeni klucza (*ang. brute-force attack*)

Obecnie można zauważyć coraz intensywniejsze wykorzystanie systemów obliczeniowych w każdej dziedzinie codziennego życia. Jednocześnie rozwiązania te powoli zaczynają ze sobą współpracować tworząc sieci systemów, dla których istotnym elementem zaczyna być również ochrona informacji. Małe, wbudowane rozwiązania są stosowane na obszarach, które wykraczają poza obszar, który może być fizycznie odgraniczony od przypadkowych bądź celowych działań prowadzących do niepoprawnej realizacji podstawowych zadań związanych z bezpieczeństwem informacji. Rozpowszechnienie zastosowań urządzeń o niewielkich zasobach systemowych (szybkość procesora, rozmiar dostępnej pamięci) spowodowało intensywne zainteresowanie możliwymi zastosowaniami kryptografii również w tej dziedzinie.

Przy wyborze algorytmu kryptograficznego należy uwzględnić przede wszystkim elementy, mające wpływ na uzyskany poziom bezpieczeństwa oferowany przez końcowy produkt. Należy jednak również zwrócić uwagę na elementy bezpośrednio niezwiązane z bezpieczeństwem produktu, a wynikające z jego specyfiki. Istotnym czynnikiem decydującym o przydatności konkretnego rozwiązania może być również szybkość jednostki obliczeniowej czy rozmiar dostępnej pamięci. W pierwszym przypadku szybkość procesora może powodować, że algorytm pomimo jego implementacji nie ma praktycznego zastosowania (przykładem może być implementacja algorytmu RSA dla mikrokontrolera ATmega128 realizująca operacje z wykorzystaniem klucza o długości 1024 bitów w czasie 11 sekund [NGURA]). W przypadku ograniczeń związanych z rozmiarem pamięci kluczowym elementem będzie wielkość pamięci wymaganej do implementacji algorytmu; należy w tym przypadku rozważyć wykorzystanie algorytmów działających bez wykorzystania skrzynki podstawieniowej (ang. *S-box*), której przechowywanie może być kosztowne. Wiele ogólnie znanych rozwiązań powstało z myślą o szybkości działania, nie z myślą o ograniczeniach w rozmiarze pamięci wymaganej do zapisu implementacji algorytmu.

Podstawowymi elementami umożliwiającymi tworzenie rozwiązań oferujących ochronę danych są szyfry blokowe, funkcje skrótu oraz kody uwierzytelniające. W artykule tym uwagę skupiono na szyfrach blokowych. Platformą sprzętową, wykorzystywaną przez autora, z której wynikają ograniczenia, stanowiące podstawę do rozważań zawartych w niniejszym

artykule, jest rodzina mikrokontrolerów 8-bitowych PIC firmy Microchip. Wnioski dotyczą jednak dowolnej rodziny mikrokontrolerów o podobnych właściwościach.

Typowe ograniczenia związane z realizacją projektu opartego na mikrokontrolerach serii PIC18 firmy Microchip:

- CPU: 8-bitowy,
- zegar: do 64 MHz,
- rozmiar słowa rozkazowego: 16 bitów,
- pamięć programu: do 128 KB,
- pamięć danych: do 1 KB,
- pamięć RAM: do 4 KB.

Pomimo szerokiego zastosowania ogólnie przyjętego, standardowego algorytmu AES, ze względu na szeroki zakres parametrów charakteryzujących rodzinę procesorów firmy Microchip wybór algorytmu szyfrującego nie jest oczywisty. Celem niniejszego przeglądu jest ułatwienie wyboru odpowiedniego algorytmu w zależności nie tylko od wymagań dotyczących bezpieczeństwa, ale również z uwzględnieniem ograniczeń wynikających z zastosowanego rozwiązania sprzętowego. Uzasadnieniem zastosowania implementacji w języku C jest jej łatwość i czytelność dla szeregu różnych algorytmów oraz dostępność referencyjnych implementacji. Tworzenie projektu opartego na mikrokontrolerze wymagającego zastosowania algorytmu kryptograficznego może rodzić pytanie, czy jedynym wyborem jest standardowy algorytm AES oraz jakie koszty trzeba ponieść (w sensie wymaganego rozmiaru pamięci i szybkości działania) przy wyborze innego, powszechnie uznanego algorytmu. Własna implementacja algorytmu kryptograficznego może być nietrywialna, dlatego pożądane jest porównanie ogólnodostępnych implementacji, bez odwoływania się do wysoko zoptymalizowanych, ale również kosztownych, bibliotek kryptograficznych.

W przeglądzie uwzględniono również algorytmy przeznaczone do implementacji sprzętowej ze względu na możliwość ich zwięzłej implementacji programowej, nie uwzględniając ewentualnego spadku wydajności związanego z pierwotnym przeznaczeniem.

Do przeglądu wybrano następujące algorytmy: AES, DES, TDES, TEA, XTEA, IDEA, SERPENT, SKIPJACK, PRESENT, HEIGHT, CLEFIA, SEA, mCrypton, RC6, BLOWFISH, TWOFISH, KLEIN, MARS. Pomimo że niektóre algorytmy nie są już obecnie uznawane za bezpieczne (głównie ze względu na stosowaną długość klucza), zawarto je w przeglądzie ze względu na brak skutecznych ataków na pełną wersję algorytmu (np. SKIPJACK), bądź ich prostotę (TEA). Do teoretycznego porównania algorytmów można wykorzystać pojęcie marginesu bezpieczeństwa wprowadzone przez Lenstrę i Verheula w [36]. Margines bezpieczeństwa określa rok, w którym można poziom bezpieczeństwa określonego algorytmu porównać do poziomu bezpieczeństwa algorytmu DES w roku 1982. Wartości marginesu

bezpieczeństwa dla typowych długości klucza przedstawia tabela 1. Należy zwrócić uwagę, że wartości te odnoszą się do algorytmów, dla których jedynym skutecznym atakiem jest atak polegający na przeszukaniu całej przestrzeni klucza. Głównymi parametrami porównania (biorąc pod uwagę budowę algorytmu) są: stosowana długość klucza, rozmiar bloku oraz liczba rund. Dla porównania szybkości algorytmów oraz pamięci potrzebnej do ich implementacji zastosowano referencyjne implementacje. Ewentualne optymalizacje lub zmiany w implementacji zostały opisane w tekście.

Tabela 1

Wartość marginesu bezpieczeństwa dla wybranych rozmiarów klucza

Rozmiar klucza w bitach	Margines bezpieczeństwa
80	2013
112	2055
128	2075
192	2159
256	2242

W artykule przedstawiono przegląd dużej grupy algorytmów, których zastosowanie na platformie 8-bitowej może być uzasadnione ze względu na ich rozpowszechnienie (AES, Serpent, IDEA) jak i na ich projektowy cel. Implementacje wybranej podgrupy algorytmów podlegały porównaniu, w rozdziale 2 zaprezentowano charakterystykę szerszej grupy algorytmów ze względu na częste ich występowanie w literaturze. Porównanie wybranych grup algorytmów można również odszukać w literaturze. Implementacje algorytmów DESL, HIGHT, SEA, TEA na 8-bitowym kontrolerze ATMEL zostały przedstawione w [10]. Algorytmy HIGHT, Present, IDEA, XTEA, SEA, AES zostały porównane w [11]. Implementacje AES, Serpent, Camelia, CAST5, MARS dla kontrolera Atmega128 zostały przedstawione w [14]. Wydajność i bezpieczeństwo algorytmów Skipjack, RC5, RC6, Rijndael, Twofish, MISTY1, KASUMI, Camelia z uwzględnieniem ich zastosowania w sięciach sensorowych zostały przeanalizowane w [33].

2. Przegląd algorytmów kryptograficznych

Algorytm AES jest obecnym standardem szyfrowania w Stanach Zjednoczonych Ameryki (FIPS-197). Rekomendacja zastosowania algorytmu do ochrony danych przez ośrodki rządowe, jak i brak ograniczeń patentowych pociągnęły za sobą jego rozpowszechnienie w wielu produktach komercyjnych i ustanowienie algorytmu globalnym, standardowym algorytmem szyfrującym.

Algorytm AES jest algorytmem realizującym operacje na blokach 16-bajtowych z możliwością zastosowania klucza o długości 128, 192, 256 bitów. Twórcy algorytmu udostępnili

jego wydajną implementację, która stała się podstawą implementacji w szeroko wykorzystywanych bibliotekach, tj. OpenSSL. Istnieje wiele dostępnych implementacji algorytmu AES, również takich z uwzględnieniem ograniczeń wynikających z zastosowania mikrokontrolera 8-bitowego. Jeżeli istnieje konieczność stworzenia własnej implementacji, jest możliwe wykorzystanie jego bezpośredniej definicji zawartej w standardzie. Jest to przenośna wersja algorytmu realizująca operacje na elementach 8-bitowych, co zapewnia jej działanie na praktycznie każdej architekturze dysponującej kompilatorem języka C. Taka implementacja jest jednak mało wydajna i jej wykorzystanie w takiej postaci nie jest zalecane. Wydajna i odporna na ataki z kanałem bocznym implementacja została zaprezentowana w [1].

Algorytm DES jest jednym z najlepiej przebadanych algorytmów kryptograficznych. Założenia projektowe przyjęte podczas jego tworzenia ograniczyły czas życia tego algorytmu ze względu na praktyczną możliwość zrealizowania ataku z przeszukaniem całej przestrzeni klucza. Pomimo wycofania rekomendacji tego algorytmu przez NIST, może on być stosowany do ochrony danych o niskiej wartości lub takich, których okres ważności jest krótki. Poziom bezpieczeństwa oferowany przez rozwiązania bazujące na algorytmie DES może zostać podniesiony poprzez zastosowanie jego modyfikacji.

Potrójny DES to algorytm zdefiniowany w publikacji NIST 800-67. Jego działanie polega na trzykrotnym wykonaniu operacji z zastosowaniem podstawowej wersji algorytmu DES (szyfrowanie/deszyfrowanie/szyfrowanie). Różny może być dobór kluczy szyfrujących przy wykonywaniu tych operacji. Rozwiązanie z trzema identycznymi kluczami ma zapewnić wsteczną zgodność ze starymi produktami, rozwiązanie z trzema różnymi kluczami zapewnia najwyższy poziom bezpieczeństwa na poziomie 112 bitów (redukcja poziomu bezpieczeństwa wynika z możliwości zrealizowania ataku ze spotkaniem pośrodku – *ang. man-in-the-middle attack*).

Zastosowanie algorytmu DES może być uzasadnione w przypadku chęci wykorzystania algorytmu dobrze znanego z dostępnymi wieloma implementacjami. Należy jednak brać pod uwagę, że wybór ten będzie okupiony zarówno utratą wydajności, jak i zwiększonym zapotrzebowaniem na pamięć potrzebną na implementację algorytmu. Szacunkowy rozmiar kodu potrzebnego do implementacji algorytmu DES to 1.5 raza więcej, niż to co jest potrzebne do implementacji algorytmu AES. Pod względem wydajności należy oczekiwać uzyskania około 40% szybkości algorytmu AES [34].

W pracy [35] zaproponowano nowe warianty algorytmu DES: DESL oraz DESXL projektowane z myślą o urządzeniach wbudowanych. Chociaż zyski uzyskano dla implementacji sprzętowej, to już implementacja programowa wymaga zasobów zbliżonych do implementacji podstawowej wersji algorytmu DES i dlatego z punktu widzenia zastosowań dla mikrokontrolerów jednoukładowych nie są bardziej atrakcyjne.

Algorytm Serpent jest algorytmem zaprojektowanym przez Andersona, Bihama i Knudsena [31]. Został zgłoszony do konkursu na algorytm AES, spełnia więc wymagania dotyczące jego budowy: pracuje na bloku 128-bitowym i umożliwia wykorzystanie klucza 128-, 192- i 256-bitowego. Struktura algorytmu jest oparta na sieci podstawieniowo-przestawieniowej. Podczas projektowania algorytmu nacisk położono na bezpieczeństwo. Efektem tego podejścia jest nieco mniejsza wydajność niż algorytmu Rijndael – zwycięzcy konkursu na algorytm AES. Struktura algorytmu została tak opracowana, żeby była możliwa jego wydajna implementacja z wykorzystaniem techniki o nazwie *bit-slicing* [32]. Według twórców algorytm był drugim najszybszym spośród kandydatów zakwalifikowanych do 2 rundy konkursu na algorytm AES, jeżeli chodzi o implementację programową.

Algorytm Serpent jest algorytmem ogólnodostępnym, nie istnieją żadne ograniczenia dotyczące jego komercyjnego stosowania. Aktualnie nie są znane żadne metody ataku na algorytm Serpent, które w sposób praktyczny zagrażałyby jego bezpieczeństwu.

Algorytmy TEA oraz XTEA. Tiny Encryption Algorithm (TEA) jest algorytmem zaproponowanym przez D. Wheelera i R. Needhama [28]. Jest algorytmem wykorzystującym strukturę sieci Feistela, realizuje operacje na bloku 64-bitowym z zastosowaniem klucza o długości 128 bitów. Ze względu na możliwość zastosowania ataku z pokrewnym kluczem [24] zostały zaproponowane dwie kolejne wersje algorytmu: extended TEA (XTEA) oraz Corrected Block TEA (XXTEA) [29]. Kryptoanaliza różnicowa dla wersji algorytmu XXTEA została zaprezentowana w [30]. Algorytm TEA korzysta z operacji dodawania, operacji XOR i przesunięć. Do budowy algorytmu nie zastosowano S-bloków, co redukuje rozmiar pamięci wymaganej do implementacji algorytmu. Odpowiedni poziom ubezpieczeństwa uzyskuje się poprzez wykonanie odpowiednio dużej liczby rund, co ma wpływ na wydajność algorytmu.

Najskuteczniejszy atak na algorytm XTEA z wykorzystaniem spokrewnionego klucza jest ukierunkowany na wersję z 27 rundami. Rozsądne więc wydaje się przyjęcie wykonania co najmniej 32 rund. Algorytm nie podlega żadnym patentom i może być stosowany w komercyjnych rozwiązaniach bez ograniczeń.

Algorytm IDEA (*ang. International Data Encryption Algorithm*) został zaprojektowany jako zastępca algorytmu DES i opublikowany w 1991 roku [25]. Jest przeznaczony do implementacji programowej. Został opatentowany przez firmę MediaCrypt. W Europie patent wygasł w maju 2011 roku. Algorytm pracuje na bloku 64-bitowym z kluczem 128-bitowym. Został elementem pakietu Pretty Good Privacy, co powoduje, że jest algorytmem dobrze znanym i przebadanym. Kluczowym elementem algorytmu jest przekształcenie mnożenie-dodawanie realizujące pełną dyfuzję w pojedynczym kroku. Przekształcenie to jest również minimalne – do osiągnięcia takiego efektu wymagane są przynajmniej 4 operacje – ma to

miejsce właśnie w tym przekształceniu. Najskuteczniejszy atak na algorytm został opublikowany w 2007 roku [26], dotyczy on możliwego ataku na zredukowaną postać algorytmu (6 rund z 8.5 standardowo realizowanych w algorytmie). Wskazano również dużą grupę słabych kluczy [27].

Algorytm PRESENT jest nowym algorytmem szyfrującym zaprojektowanym specjalnie dla urządzeń o ograniczonych zasobach systemowych [21]. Jest oparty na sieci podstawieniowo-przestawieniowej, realizuje operacje w 31 rundach na bloku 64-bitowym, korzysta z klucza o długości 80 lub 128 bitów. Podczas projektowania algorytmu przyjęto założenia, że w docelowym środowisku wystarczy przyjęcie średniego poziomu bezpieczeństwa (stąd możliwość wykorzystania klucza o długości 80 bitów) oraz że nie będzie wymagane przetwarzanie dużej ilości danych (możliwa jest optymalizacja kodu pod kątem jego rozmiaru bez znaczącego narzutu na wydajność).

Dla algorytmu wskazano możliwość wykorzystania ataku ze słabym kluczem [22] oraz ataku z wykorzystaniem kryptoanalizy liniowej [23].

Algorytm HIGHT został zaproponowany w 2006 podczas konferencji CHES [19] jako algorytm do implementacji sprzętowej w urządzeniach o niewielkich zasobach systemowych stosowanych np. do budowy sieci sensorowych. Budowa algorytmu jest oparta na sieci Feistela. Operuje na blokach 64-bitowych z kluczem 128-bitowym. Podczas działania są realizowane operacje XOR, dodawania modulo 2^8 oraz rotacje bitowe.

Margines bezpieczeństwa algorytmu szacowany pierwotnie na 13 rund, został zredukowany do 4 rund po opublikowaniu kryptoanalizy różnicowej dotyczącej wersji algorytmu HIGHT z 28 rundami [20].

Algorytm CLEFIA jest wydajnym algorytmem szyfrującym zaprezentowanym w 2007 roku [15]. Pracuje na bloku 128-bitowym z kluczem 128-, 192- lub 256-bitowym. Algorytm CLEFIA bazuje na zmodyfikowanej sieci Feistela. Modyfikacja polega na podwojeniu liczby gałęzi w sieci i równoległym zastosowaniu funkcji przekształcającej pracującej na bloku o rozmiarze 32 bitów, co umożliwia bardziej zwartą implementację. Algorytm został zgłoszony do IETF i jego specyfikacja została opublikowana jako dokument RFC 6114.

Nowy wariant kryptoanalizy liniowej zastosowano do ataku na algorytm CLEFIA z 13 rundami [16]. Zaproponowano również atak na implementację z wykorzystaniem śladów pamięci podręcznej [17]. Ataki te jednak nie mają praktycznego znaczenia.

Ograniczeniem stosowania algorytmu są warunki licencyjne. Algorytm CLEFIA bez licencji może być obecnie wykorzystywany jedynie w celach ewaluacyjnych, wykorzystanie algorytmu w implementacjach programowych i sprzętowych może być realizowane wyłącznie za zgodą firmy Sony.

Algorytm SEA [18] jest algorytmem blokowym ukierunkowanym na aplikacje wbudowane. Może być parametryzowany ze względu na rozmiar bloku/rozmiar klucza n i rozmiar słowa procesora b. Podczas pracy algorytmu są stosowane operacje XOR, rotacji słowa, dodawanie modulo, jest również stosowany blok podstawieniowy. Zgodnie z definicją algorytmu n musi być krotnością $6b$, co dla b=8 umożliwia utworzenie wersji algorytmu z kluczem, np. 96- czy 144-bitowym, które są oznaczane jako SEA_96_8, SEA_144_8. Twórcy algorytmu określają minimalną liczbę rund na $3*n/4 + 2(nb+\lfloor(b/2)\rfloor)$, gdzie $nb = n/2b$. Algorytm SEA jest odporny na kryptoanalizę różnicową i liniową. Nie są opublikowane żadne inne skuteczne metody ataku na algorytm.

Algorytm mCrypton [12] jest algorytmem zaprojektowanym dla małych urządzeń wbudowanych, takich jak identyfikatory RFID. Realizuje wykonanie 12 rund na bloku 64-bitowym i umożliwia wykorzystanie kluczy o długościach: 64, 96 i 128 bitów. W pracy [13] Park wykazał możliwość zrealizowania ataku na algorytm ze zredukowaną do 8 liczbą rund z wykorzystaniem wariantu ataku ze spokrewnionym kluczem.

Algorytm Camelia [2] jest szyfrem blokowym pracującym na bloku 128-bitowym. Został zaprojektowany przez NTT i Mitsubishi do zwartej implementacji zarówno programowej, jak i sprzętowej. W algorytmie korzysta się wyłącznie z operacji logicznych i odwołań do tablic. Może być efektywnie implementowany zarówno na procesorach 8-bitowych, jak i w systemach serwerowych z 64 bitowymi jednostkami obliczeniowymi.

Przeciwko algorytmowi nie są znane żadne ataki skuteczniejsze niż atak brutalny. Algorytm może być powszechnie wykorzystywany w postaci zdefiniowanej przez twórców (nie dopuszcza się modyfikacji algorytmu).

Algorytm RC5 [7] jest szybkim algorytmem wykorzystującym intensywnie przesunięcia zależne od danych. Algorytm RC5 ma zmienną długość bloku, liczbę rund i długość klucza. Pierwotnie proponowaną wersją algorytmu była wersja pracująca na bloku 32-bitowym, z kluczem 128-bitowym, wykonująca 12 rund. Ze względu na możliwość zastosowania kryptoanalizy różnicowej za podstawową wersję algorytmu obecnie uważa się wersję z wykonaniem 20 rund algorytmu [8].

Algorytm RC6 został zaproponowany w 1996 [6] roku jako rozszerzenie RC5 i spełnia wymagania zawarte w konkursie na algorytm AES. Pracuje na bloku 128-bitowym i wspiera możliwość wykorzystania klucza 128-, 192- i 256-bitowego. Zgodnie z zapewnieniami firmy RSA Security algorytm RC6 nie podlegałby ograniczeniom licencyjnym, gdyby został zwycięzcą konkursu na nowy algorytm szyfrujący. W związku z tym, że zwycięzcą nie został, jego sytuacja licencyjna nie jest do końca jasna.

Algorytm Twofish [9] jest algorytmem opartym na strukturze sieci Feistela. Spełnia charakterystykę uczestników konkursu na algorytm szyfrujący: pracuje na bloku 128-bito-

wym, umożliwia stosowanie klucza o długości 128, 192 lub 256 bitów. Na pełną wersję algorytmu (16 rund) nie są znane żadne teoretyczne ataki. Algorytm nie jest objęty żadnymi ograniczeniami licencyjnymi.

Algorytm KLEIN [3] jest przeznaczony do stosowania w urządzeniach wbudowanych. Jest algorytmem nowym (pochodzi z 2010 roku) i nie doczekał się jeszcze obszernego opracowania dotyczącego jego kryptoanalizy. Algorytm ma strukturę sieci podstawieniowo-przestawieniowej. Pracuje na bloku 64-bitowym i umożliwia stosowanie klucza o długości 64, 80 lub 96 bitów. Zgodnie z zaleceniem autorów wersja z kluczem 64-bitowym powinna być wykorzystywana do tworzenia funkcji skrótu dla wiadomości o długości do dwóch bloków. Algorytm w wersji z kluczem o długości 80 i 96 bitów może być używany do szyfrowania danych.

Algorytm SKIPJACK [5] jest algorytmem blokowym stworzonym przez Agencję Bezpieczeństwa Narodowego Stanów Zjednoczonych. Algorytm pracuje na bloku 64-bitowym i umożliwia zastosowanie klucza o długości 80 bitów. Jest algorytmem dedykowanym dla urządzeń wbudowanych [4].

Tabela 2

Zestawienie podstawowych parametrów algorytmów

Algorytm	Rozmiar klucza [bitów]	Rozmiar bloku [bitów]	Liczba rund
AES	128, 192, 256	128	10,12,14
TEA	128	64	32
XTEA	128	64	32
IDEA	128	64	8,5
PRESENT	80,128	64	31
HIGHT	128	64	32
Clefia	128, 192, 256	128	18, 20, 22
Skipjack	80	64	32
RC5	0-2040	32, 64, 128	1-255
RC6	128, 192, 256	128	20

Zestawienie dotyczące budowy algorytmów, których implementacje zostały porównane, zamieszczono w tabeli 2. W celu porównania wybranych algorytmów wykorzystano ich implementacje w języku C. Pochodzenie źródeł implementacji:

- avrcryptlib (link): PRESENT, SKIPJACK,
- mcrypt (link): IDEA,
- implementacje referencyjne: TEA, RC5, RC6, AES, CLEFIA, KLEIN.

3. Środowisko uruchomieniowe

Kompilacja wszystkich kodów źródłowych była wykonywana za pomocą kompilatora MPLAB C for PIC18 v3.38 w środowisku MPLAB IDE. Testowanie i uruchamianie algorytmów zostało przeprowadzone za pomocą wbudowanego w środowisko programisty debuggera i symulatora mikrokontrolera PIC18F4550. Fizycznym urządzeniem wykorzystywanym podczas testów był ten sam mikrokontroler zainstalowany w systemie rozwojowym dla mikrokontrolerów PIC.

4. Wyniki

Każda z implementacji wybranych algorytmów była uruchamiana niezależnie. Do odczytu informacji o zajętości pamięci wykorzystano narzędzia wbudowane w środowisko MPLAB IDE. Liczba cykli była wyznaczana programowo, osobno dla operacji szyfrowania i deszyfrowania. Wyniki zajętości pamięci programu i danych zostały zamieszczone w tabeli 3.

Tabela 3

Zajętość pamięci

Algorytm	Pamięć programu [bajtów]	Pamięć danych [bajtów]
AES	6104	33
TEA	3419	348
XTEA	3329	354
IDEA	10612	552
PRESENT	3711	464
HIGHT	12756	858
Clefia	5959	805
Skipjack	3708	592
RC5	4429	548
RC6	4576	446

Dane dotyczące algorytmu AES pochodzą z noty aplikacyjnej AN953 opublikowanej przez firmę Microchip. Dane dotyczą implementacji algorytmu w języku C. Należy zauważyć, że implementacja w assemblerze, opublikowana w nocie aplikacyjnej, jest znacznie wydajniejsza, jednak ze względu na porównanie z pozostałymi implementacjami utworzonymi w języku C nie była ona brana pod uwagę.

Jeżeli podstawowym kryterium doboru algorytmu jest rozmiar pamięci potrzebnej do jego zapisania, to zdecydowanymi faworytami są algorytmy XTEA oraz PRESENT. W grupie algorytmów wymagających mniej niż 4kB pamięci kodu znalazły się również algorytmy Skipjack oraz TEA. Jeżeli długość klucza stosowanego w algorytmie Skipjack nie stanowi

przeszkody, to warto również brać pod uwagę ten algorytm. Ze względu na słabości podstawowej wersji algorytmu TEA zalecane jest wykorzystanie poprawionej wersji – algorytmu XTEA, tym bardziej że nie pociąga to za sobą zwiększenia rozmiaru pamięci potrzebnego do zapisania algorytmu.

Wyniki dotyczące wydajności algorytmów zostały zamieszczone w tabeli 4. Kolumny zawierające dane dotyczące wydajności wyrażane w cyklach/bajt zostały wyznaczone po uwzględnieniu rozmiaru bloku stosowanego w wybranej implementacji.

Tabela 4

Wydajność algorytmów

Algorytm	Szyfrowanie [cykli]	Deszyfrowanie [cykli]	Szyfrowanie [cykli/bajt]	Deszyfrowanie [cykli/bajt]
AES	34448	47040	2153	2940
TEA	13553	13629	1694	1704
XTEA	17432	17508	2179	2188
IDEA	6742	6741	843	843
PRESENT	8044	37511	1005	4689
HIGHT	42656	43680	5332	5460
Clefia	23811	49318	1488	3082
Skipjack	22982	23144	2873	2893
RC5	17890	19352	4472	4838
RC6	29217	29798	1826	1862

Jeżeli kluczowym elementem wyboru algorytmu jest szybkość działania, to rozważyć należy przypadki szyfrowania małych i dużych porcji danych oraz czy będą realizowane zarówno operacje szyfrowania i deszyfrowania czy tylko operacja szyfrowania. W przypadku szyfrowania małych porcji danych (o rozmiarach mniejszych niż 64 bity) należy uwzględnić konieczność zrealizowania operacji na całym bloku. W tym przypadku w czołówce plasują się algorytmy IDEA i PRESENT, przy czym ten pierwszy zachowuje znacznie korzystniejszą proporcję dotyczącą szybkości operacji szyfrowania i deszyfrowania. W drugiej grupie rozwiązań pod względem wydajności można umieścić algorytmy TEA, XTEA oraz RC5 – w przypadku tego ostatniego należy brać pod uwagę ograniczenie rozmiaru danych do 32 bitów. W przypadku konieczności szyfrowania większych porcji danych najbardziej wydajne nadal są algorytmy IDEA oraz PRESENT. Na kolejnych miejscach pod względem wydajności pojawiają się algorytmy Clefia, TEA oraz RC6. Warto zwrócić uwagę, że od tej grupy niewiele wolniejszy jest również algorytm AES.

W przypadku urządzeń mobilnych lub sieci sensorowych często zachodzi potrzeba jedynie wysyłania zgromadzonych danych. W takim przypadku można zrezygnować z implementacji operacji deszyfrowania. W przypadku gdy obie operacje muszą być zastosowane, to dużą symetrycznością czasu działania operacji szyfrowania i deszyfrowania cechują się algorytmy IDEA, XTEA oraz RC6. Należy również zwrócić uwagę, że algorytm

PRESENT cechuje się znacznym spadkiem wydajności podczas realizacji operacji deszyfrowania. Algorytm ten został stworzony dla urządzeń wbudowanych, których częstym przypadkiem użycia jest jedynie wysyłanie zaszyfrowanych danych i to może być uzasadnienie tak dużego spadku wydajności przy realizacji operacji deszyfrowania.

5. Podsumowanie

Dobór algorytmu kryptograficznego nie jest zadaniem oczywistym. Konieczne jest uwzględnienie wielu czynników wynikających z ograniczeń platformy, na której tworzone jest finalne rozwiązanie, jak i również jego ostateczna funkcjonalność. Pomimo że ogólnie przyjętym, szeroko stosowanym rozwiązaniem jest algorytm AES, warto również rozważyć zastosowanie innych algorytmów. Algorytmy, które również warto brać pod uwagę to algorytmy XTEA, IDEA i PRESENT. Implementacje pozostałych algorytmów nie różnią się znacząco od implementacji algorytmu AES, co powoduje, że ich wybór staje się nieuzasadniony. Wybór każdego z wyróżnionych algorytmów niesie ze sobą pewne konsekwencje: algorytm XTEA wydaje się być kryptograficznie najsłabszy, algorytm IDEA wymaga największego obszaru pamięci, algorytm PRESENT zdecydowanie najgorzej wypada w przypadku konieczności realizacji zarówno operacji szyfrowania, jak i deszyfrowania. W przypadku braku dostępności biblioteki kryptograficznej okazuje się jednak, że wskazane algorytmy (a konkretnie ich referencyjne implementacje w języku C) mogą skutecznie konkurować pod względem wydajności i zajętości pamięci z rozwiązaniami komercyjnymi oferowanymi przez producenta mikrokontrolerów.

BIBLIOGRAFIA

1. Konighofer R.: A fast and cache-timing resistant implementation of the AES. Lecture Notes in Computer Science, Vol. 4964, Springer, 2008, s. 187÷202.
2. Aoki K., Ichikawa T., Kanda M., Matsui M., Moriai S., Nakajima J., Tokita T.: Specification of Camellia – a 128-Bit Block Cipher. Specification Version 2.0, Nippon Telegraph and Telephone Corporation and Mitsubishi Electric Corporation. 2001.
3. Gong Z., Nikova S., Law Y. W.: KLEIN: A New Family of Lightweight Block Ciphers, Technical Report, Centre for Telematics and Information Technology, University of Twente, 2010.
4. Law Y.W., Doumen J.,Hartel P. H.: Survey and benchmark of block ciphers for wireless sensor networks. ACM Transactions Sensor Networks, 2(1), 2006, s. 65÷93.

5. Skipjack and kea algorithm specifications (version 2.0). National Institute of Standards and Technology online document. http://csrc.nist.gov/groups/ST/toolkit/documents-/skipjack/skipjack.pdf, 1998.
6. Rivest R., Robshaw, M., Sidney, R., And Yin, Y.: The RC6™ Block Cipher. Specification version 1.1.
7. Rivest R. L.: The RC5 encryption algorithm. Lecture Notes in Computer Science, Vol.1008, 1995, s. 86÷96
8. Biryukov A. Kushilevitz E.: Improved Cryptanalysis of RC5. EUROCRYPT 1998.
9. Schneier B., Kelsey J., Whiting D., Wagner D., Hall Ch., Ferguson N.: The Twofish encryption algorithm: a 128-bit block cipher. John Wiley & Sons, Inc., New York, 1999.
10. Rinne S., Eisenbarth T., Paar C.: Performance Analysis of Contemporary Light-Weight Block Ciphers on 8-bit Microcontrollers. Ecrypt workshop SPEED – Software Performance Enhancement for Encryption and Decryption, 2007.
11. Eisenbarth T., Kumar S., Paar Ch., Poschmann A. Uhsadel L.: A Survey of Lightweight-Cryptography Implementations. IEEE Computer Society Press, Los Alamitos 2007.
12. Lim C.H., Korkishko T.: mCrypton – a lightweight block cipher for security of low-cost RFID tags and sensors. Lecture Notes in Computer Science, Vol. 3786, Springer, Berlin 2005, s. 243÷258.
13. Park J. H.: Security analysis of mCrypton proper to low-cost ubiquitous computing devices and applications. International Journal of Communication Systems, John Wiley and Sons Ltd. ,Chichester, UK, 2009, s. 959÷969.
14. Murat Ç.: Software implementation and performance comparison of popular block ciphers on 8-bit low-cost microcontroller. International Journal of the Physical Sciences Vol. 5(9), 2010, s. 1338÷1343.
15. Shirai T., Shibutani K., Akishita T., Moriai S., Iwata T.: The 128-bit blockcipher CLEFIA (extended abstract). Lecture Notes in Computer Science, Vol. 4593, Springer-Verlag, 2007, s. 181÷195.
16. Bogdanov A., Rijmen V.: Zero-Correlation Linear Cryptanalysis of Block Ciphers. IACR ePrint archive: Report 2011/123.
17. Rebeiro Ch., Mukhopadhyay .: Cryptanalysis of CLEFIA using Differential Methods with Cache Trace Patterns. Lecture Notes in Computer Science Vol. 6558, Springer-Verlag, 2011, s. 89÷103.
18. Standaert F., Piret F., Gershenfeld N.:, Quisquater J: SEA: A Scalable Encryption Algorithm for Small Embedded Applications," Lecture Notes in Computer Science Vol. 3928, Spain, 2006, s. 222÷236.

19. Hong D., Sung J., Hong S., Lim J., Lee S., Koo B., Lee C., Chang D., Lee J., Jeong T., Kim H., Kim J., Chee J.: HIGHT: A new block cipher suitable for low-resource device. Lecture Notes in Computer Science Vol. 4249, Springer-Verlag, 2006, s. 46÷59.
20. Lu J.: Cryptanalysis of reduced versions of the HIGHT block cipher from CHES 2006. Lecture Notes in Computer Science, Vol. 4817, Springer-Verlag, 2007, s. 11÷26.
21. Bogdanov A., Knudsen L.R., Le G., Paar C., Poschmann A., Robshaw M.J.B., Seurin Y., Vikkelsoe M.: PRESENT: An Ultra-Lightweight Block Cipher, Springer, 2007
22. Ohkuma K.: Weak keys of reduced-round present for linear cryptanalysis. Springer-Verlag, 2009.
23. Cho J.Y.: Linear cryptanalysis of reduced-round present. Lecture Notes in Computer Science Vol. 5985, 2010, s. 302÷317.
24. Kelsey J., Schneier B., Wagner D.: Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. Springer-Verlag, 1997.
25. Lai X., Massey J., Murphy S.: Markov Ciphers and Differential Cryptanalysis. Lecture Notes in Computer Science Vol. 547, Springer-Verlag, 1992, s. 17÷38.
26. Biham E., Dunkelman O., and Keller N.: A New Attack on 6-Round IDEA. Lecture Notes in Computer Science Vol. 4593, Springer-Verlag, 2007, s. 211÷224.
27. Biryukov A., Nakahara J., Preneel B., Vandewalle J.: New Weak-Key Classes of IDEA. Lecture Notes in Computer Science Vol. 2513, Springer, 2002, s. 315÷326.
28. Wheeler D.J., Needham R.M.: TEA, a tiny encryption algorithm. Lecture Notes in Computer Science Vol. 1008, Springer, 1994, s. 363÷366.
29. Needham R.M., Wheeler D.J.: TEA extensions. Technical report, Computer Laboratory, University of Cambridge, 1997.
30. Yarrkov E.: Cryptanalysis of XXTEA. Cryptology ePrint Archive: Report 2010/254.
31. Anderson R., Biham E., Knudsen L.: The Case for Serpent. AES Candidate Conference, http://www.cl.cam.ac.uk/~rja14/Papers/serpentcase.pdf , 2000.
32. Biham E.: A Fast New DES Implementation in Software. Fast Software Encryption, Springer-Verlag, 1997.
33. Law Y.W., Doumen J., Hartel P: Survey and benchmark of block ciphers for wireless sensor networks. ACM Transactions on Sensor Networks Vol. 2, ACM, 2006, s. 65÷93.
34. Eisenbarth T., Kumar S., Paar Ch., Poschmann A., Uhsadel L.: A Survey of Lightweight-Cryptography Implementations. IEEE Design & Test of Computers Vol. 24(6), 2007, s. 522÷533.
35. Leander G., Paar Ch., Poschmann A., Schramm K.: New Lightweight DES Variants. Lecture Notes in Computer Science Vol. 4593, Springer, 2007, s. 196÷210.

36. Lenstra A.K., Verheul E.R.: Selecting cryptographic key sizes. Journal of Cryptology 14(4), 2001, s. 255÷293.

Recenzent: Dr inż. Bożena Wieczorek

Wpłynęło do Redakcji 7 lipca 2011 r.

Abstract

This article surveys cryptography algorithms that can be used for software implementation in resource limited environments. The list of surveyed algorithms contains AES, DES, TDES, TEA, XTEA, IDEA, SERPENT, SKIPJACK, PRESENT, HEIGHT, CLEFIA, SEA, mCrypton, RC6, BLOWFISH, TWOFISH, KLEIN, MARS. Subgroup of this list was selected for comparison. Selected algorithms are characterized in Table 2. The comparison was conducted using standard implementation of the algorithms in C language. The reason for such a implementation comparison was to answer the question: is it reasonable to consider other algorithms then AES in a project built on microcontroller platform and is it possible to use some well-known, secure algorithm reference implementation and still enjoy acceptable performance/code size. The results of memory size comparison are presented in Table 3, while speed comparison containing both speed of operating on whole block and single byte are demonstrated in Table 4. XTEA, IDEA, PRESENT algorithms appeared to be the most interesting in terms of this article. However their choice comes with certain cost. XTEA is assumed to be cryptographically less secure then the others, IDEA is the algorithm with biggest code size and PRESENT performs well during encryption but decryption is significantly slower. Nevertheless it seems that those algorithms could be an interesting alternative for AES algorithm when software implementation on a platform with limited resources is taken into consideration.

Adres

Jacek LACH: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, jacek.lach@polsl.pl.