1998. - № 1. - С. 82-87. ***10****. Семеренко В. П. Разработка универсального кодера-декодера циклических кодов // Электронное моделирование. - 1995. - №4. - С.26-31. 11. Семеренко В. П. Исправление пакетов ошибок в циклических кодах //Математические машины и системы. – 1999. - №1. - С.30-42.* ***12****. Кларк Дж., мл., Кейн Дж. Кодирование с исправлением ошибок в системах цифровой связи: Пер. с англ. - М.: Радио и связь, 1987. - 392 с.*

УДК 681.3.06.

ПРАКТИЧНА РЕАЛІЗАЦІЯ КРИПТОАЛГОРИТМУ AES З ВИКОРИСТАННЯМ МІКРОКОНТРОЛЕРІВ PIC16XXX

Всеволод Семенюк, Володимир Кишкан, Ігор Золотухін, Геннадій Леоненко*

ДСТСЗІ СБ України

** СФ СБ України ВІТІ НТУУ "КПІ"*

Анотація: **Розглянуто варіант реалізації криптоалгоритму AES з використанням мікроконтролерів PIC16XXX.**

Summary: **There is the report how the AES algorithm is implemented in a PIC16XXX microcontroller.**

Ключові слова: **Симетричні алгоритми шифрування, AES.**

I Вступ

Один з найбільш відомих у світі симетричних блочних алгоритмів шифрування DES було введено Американським національним інститутом стандартів (ANSI) у 1977 році. Після більше 20 років використання він не пройшов перезатвердження у 1998 році у звязку з малим розміром ключа (56 бітів). Національний інститут стандартів та технологій США (NIST) 12 вересня 1997 року оголосив конкурс на заміщення алгоритму шифрування DES.

NIST – це урядова організація, що діє під егідою Міністерства торгівлі США та розробляє стандарти і загальні вимоги, яких дотримуються всі федеральні та комерційні організації. Діяльність NIST у галузі комп'ютерної безпеки має безпосереднє відношення до технології HPCC (High – performance computing and communications – високоефективна комп'ютерна та комунікаційна технологія). Виконуючи свою місію, NIST намагається розробляти з малими витратами стандарти та загальні вимоги в галузі інформаційної безпеки для комерційного сектора та федеральної системи. Це стосується, насамперед, безпеки в Internet, криптографічних стандартів, генерації паролів, технології інтелектуальних карток, безпеки електронних комерційних операцій, керування ризиками та підготовки кваліфікованого персоналу. Програма HPCC NIST ставить за мету вирішення наступних завдань:

- прискорення розвитку і розповсюдження високоефективних комп'ютерних та мережених технологій відповідно до потреб Національної інформаційної інфраструктури;
- тестування та використання цих технологій у виробничій сфері;
- робота як координаційного агентства в процесі розробки компонентів федеральної програми.

Організації NIST конкурсу на заміщення стандарту DES є яскравою ілюстрацією щодо вирішення завдань, покладених на цю організацію [1].

На першому етапі конкурсу було розглянуто 21 алгоритм та відібрано 5 кращих з них. У жовтні 2000 року NIST оголосив переможця конкурсу на новий національний стандарт США. Переможцем було визнано алгоритм Rijndael, який розроблено добре відомими у відповідних колах бельгійськими фахівцями: Винсентом Ріджменом та Йоном Дайменом. У листопаді 2001 року NIST видав відповідний стандарт FIPS PUB 197, Advanced Encription Standart (AES).

Алгоритм AES побудовано на ітеративному використанні підстановок та лінійних перетворень, що виконуються над блоками даних. Кількість ітерацій залежить від розміру ключа. Усі операції цього алгоритму та обчислення таблиць підстановок основані на арифметиці скінчених полів. Саме цим пояснюється те, що алгоритм добре працює на усіх платформах та потребує мінімальних ресурсів пам'яті, а також забезпечує значну швидкодію.

II Основна частина

Арифметика скінчених полів.

Скінчене поле (інакше кажучи поле Галуа) з **q** елементів будемо позначати **GF(q)**. Скінчені поля існують тільки у випадку, коли число елементів є простим числом або ступенем простого числа, тобто коли $\mathbf{q = p^m}$, де

p – просте число. Якщо **m = 1**, то поле **GF (p)** відповідно називається простим. Усім скінченим полям притаманні наступні властивості:

1) існують дві операції, які використовуються для комбінування елементів: множення **" · "** та додавання **"+" ;**

2) результатом множення чи додавання двох елементів поля є третій елемент, який належить тому ж полю;

3) поле завжди містить мультиплікативну одиницю 1 та адитивну одиницю 0, такі що **a + 0 = a** і **a · 1 = a** для будь – якого **a**;

4) для будь-якого елемента **a** існує зворотній елемент з додавання (**-a**) і зворотній елемент з множення $\mathbf{a^{-1}}$ (якщо $\mathbf{a \neq 0}$), такі що **a + (-a) = 0**, $\mathbf{a \cdot a^{-1} = 1}$; існування таких елементів дозволяє використовувати звичайні позначення для віднімання та ділення;

5) виконуються звичайні правила асоціативності [**a + (b + c) = (a + b) + c**, **(a · b) · c = a · (b · c)**], комутативності [**a + b = b + a**, **a · b = b · a**] та дистрибутивності [**a · (b + c) = a · b + a · c**].

Як приклад поля **GF** $\mathbf{(p^m)}$ розглянемо поле, елементами якого є усі поліноми ступеня **m-1** або меншого, коефіцієнти яких лежать у простому полі **GF (p)**. Множення та додавання таких поліномів здійснюються за модулем незвідного поліному **p(x)** над **GF (p)** ступеня **m**. Незвідним поліномам притаманна така особливість, що їх не можна розкласти на множники, використовуючи поліноми з коефіцієнтами із **GF (p)**. Як і прості числа їх знаходять методом перебору [2].

Наприклад, поліном $\mathbf{p(x) = 1 + x + x^3}$, незвідний над **GF (2)**, може бути використаний для побудови **GF (8)**. Нехай $\mathbf{b_1 = 1 + x + x^2}$ та $\mathbf{b_2 = 1 + x^2}$ – два елементи **GF (8)**.

Можна виконати їх додавання:

$$\mathbf{b_1 + b_2 = (1 + x + x^2) + (1 + x^2) = (1 + 1) \cdot 1 + x + (1 + 1) \cdot x^2 = x}$$

та множення:

$$\mathbf{b_1 \cdot b_2 = (1 + x + x^2) \cdot (1 + x^2) = 1 + x + x^2 + x^2 + x^3 + x^4 = 1 + x + x^3 + x^4}.$$

Для того, щоб привести ці вирази до модуля **p(x),** необхідно послідовно використати рівняння **p(x) = 0**. Так, $\mathbf{x^3 = 1 + x}$, $\mathbf{x^4 = x + x^2}$ відповідно:

$\mathbf{b_1 \cdot b_2 = 1 + x + (1 + x) + (x + x^2) = x + x^2}$.

Отже приведення поліному за модулем **p(x)** еквівалентно діленню на **p(x)** та визначенню лишку.

Перетворення алгоритму AES.

Основні операції алгоритму AES визначено на байтовому рівні, в якому байти представлено як елементи скінченого поля **GF** $\mathbf{(2^8)}$. Кілька операцій виконуються з 4 байтовими векторами. В класичному представленні байт, який містить біти $\mathbf{b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0}$, можна представити як поліном з **GF (2)** = {0,1}:

$$\mathbf{b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0}.$$

У алгоритмі AES незвідний поліном, який використовується в арифметичних операціях, позначається як **m (x)** та має вигляд: $\mathbf{m(x) = x^8 + x^4 + x^3 + x + 1}$ або **11Bh** як шістнадцятирічне число. Цей поліном взято з переліку незвідних поліномів ступеню 8, оприлюдненого в роботі [3].

Для кінцевого поля **GF** $\mathbf{(2^8)}$ 4 байтовий вектор може бути представлений як поліном ступеня, меншого за 4. Наприклад:

$\mathbf{a\ (x) = a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0}$, $\mathbf{b\ (x) = b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0}$.

Якщо **c (x) = a(x) b(x)** , то $\mathbf{c\ (x) = c_6 x^6 + c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x^1 + c_0}$, де:

$\mathbf{c_0 = a_0 \cdot b_0}$, $\mathbf{c_1 = a_1 \cdot b_0 + a_0 \cdot b_1}$, $\mathbf{c_2 = a_2 \cdot b_0 + a_1 \cdot b_1 + a_0 \cdot b_2}$, $\mathbf{c_3 = a_3 \cdot b_0 + a_2 \cdot b_1 + a_1 \cdot b_2 + a_0 \cdot b_3}$,

$\mathbf{c_4 = a_3 \cdot b_1 + a_2 \cdot b_2 + a_1 \cdot b_3}$, $\mathbf{c_5 = a_3 \cdot b_2 + a_2 \cdot b_3}$, $\mathbf{c_6 = a_3 \cdot b_3}$.

Поліном **c (x)** не може мати ступінь більшу, ніж 4. Для перетворення в алгоритмі AES обрано поліном $\mathbf{m\ (x) = x^4 + 1}$, який власно не є незвідним. Це викликано тим, що в операціях з фіксованими поліномами не потрібна інверсія.

Відповідно маємо:

$$\mathbf{d\ (x) = a\ (x)\ b\ (x) \bmod m\ (x),} \text{ тобто } \mathbf{d\ (x) = d_3 x^3 + d_2 x^2 + d_1 x^1 + d_0}$$

$$\mathbf{d_0 = a_0 \cdot b_0 + a_3 \cdot b_1 + a_2 \cdot b_2 + a_1 \cdot b_3,}$$

$$\mathbf{d_1 = a_1 \cdot b_0 + a_0 \cdot b_1 + a_3 \cdot b_2 + a_2 \cdot b_3,}$$

$$\mathbf{d_2 = a_2 \cdot b_0 + a_1 \cdot b_1 + a_0 \cdot b_2 + a_3 \cdot b_3,}$$

$$\mathbf{d_3 = a_3 \cdot b_0 + a_2 \cdot b_1 + a_1 \cdot b_2 + a_0 \cdot b_3.}$$

Така операція еквівалентна матричному множенню наступного виду:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Проміжні результати, ключ шифрування та кількість раундів обробки.

Проміжний результат, отриманий в ході різних трансформацій при роботі алгоритму, названо станом. Стан може бути представлений як прямокутний масив байтів. Цей масив завжди містить чотири рядки, число стовпців позначається як $\mathbf{N_b}$, що еквівалентно блоку розміром до 32 байтів. Ключ шифрування звичайно також представлено як прямокутний масив з чотирма рядками. Число стовпців позначається як $\mathbf{N_k}$, еквівалентний розмір блоку також досягає 32 байтів. Це представлено на рис. 1.

A_{00}	A_{01}	A_{02}	A_{03}	A_{04}	A_{05}
A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{30}	A_{31}	A_{32}	A_{33}	A_{34}	A_{35}

K_{00}	K_{01}	K_{02}	K_{03}
K_{10}	K_{11}	K_{12}	K_{13}
K_{20}	K_{21}	K_{22}	K_{23}
K_{30}	K_{31}	K_{23}	K_{33}

Рисунок 1– Приклад стану (для $N_b = 6$) та ключа шифрування (для $N_k = 6$)

У такому представленні ці блоки також можна розглядати як лінійні 4-байтові вектори, кожний вектор є відповідним стовпчиком прямокутного масиву. Отже розмір масивів є 4, 6 або 8 у представленні через 4-байтові вектори (індекси векторів у діапазонах 0..3, 0..5 або 0..7).

Дані, що надходять та виходять на зовнішні інтерфейси, представлені як лінійні масиви з 8-бітових байтів, нумерованих від 0 до $\mathbf{4*N_{b-1}}$. Ці блоки мають довжину 16, 24 або 32 байти (індекси масивів лежать у діапазонах 0..15, 0..23 або 0..31). Ключі шифрування представлені лінійними масивами з 8-бітових байтів, нумерованих від 0 до $\mathbf{4*N_{k-1}}$. Ці блоки також мають довжину 16, 24 або 32 байти (індекси масивів лежать у діапазонах 0..15, 0..23 або 0..31).

Байти на вході шифрування („вхідний текст”) можуть бути представлені в байтах стану в наступному порядку: $\mathbf{A_{00}}$, $\mathbf{A_{10}}$, $\mathbf{A_{20}}$, $\mathbf{A_{30}}$, $\mathbf{A_{01}}$, $\mathbf{A_{11}}$, $\mathbf{A_{21}}$, $\mathbf{A_{31}}$; байти ключа відбиваються на масив у порядку: $\mathbf{K_{00}}$, $\mathbf{K_{10}}$, $\mathbf{K_{20}}$, $\mathbf{K_{30}}$, $\mathbf{K_{01}}$, $\mathbf{K_{11}}$, $\mathbf{K_{21}}$, $\mathbf{K_{31}}$, **…** . По закінченні роботи алгоритму шифрування байти надходять на вихід у відповідному порядку.

Зауваження: якщо лінійний індекс байту у блоку є **n** та індекс двомірного масиву є (**i, j**), то маємо:

i = n mod 4 ; j = [n / 4] ; n = i + 4 * j.

Крім того, індекс **i** також є номером 4-байтового вектора або слова.

Між кількістю раундів обробки $\mathbf{N_r}$ та параметрами $\mathbf{N_b}$ та $\mathbf{N_к}$ алгоритм встановлює співвідношення, яке наведено в табл.1.

Таблиця 1 – Кількість раундів обробки $\mathbf{N_r}$ в залежності від розмірів блоку та ключа

$\mathbf{N_r}$	$\mathbf{N_b = 4}$	$\mathbf{N_b = 6}$	$\mathbf{N_b = 8}$
$\mathbf{N_k = 4}$	10	12	14
$\mathbf{N_k = 6}$	12	12	14
$\mathbf{N_k = 8}$	14	14	14

Алгоритм складається з початкового раунду (AddRoundKey) та **r** так званих стандартних раундів (Standard Round). Перші **r-1** раундів містять у собі чотири типи обробки: заміна байтів (Substitution Bytes); зсув рядків (Shift rows); змішування стовпців (Multiplay columns) та додавання ключа (Xored by key). Останній раунд шифрування складається з трьох типів трансформацій (виключено змішування стовпців).

Розклад ключів.

Виконання раундових обробок з масивом A потребує (крім першої операції AddRoundKey (A, Kr)) генерації списків раундових ключів (sub-keys) K[r][j i] перед початком кожного раунду. Це важлива складова алгоритму. Генеровані раундові ключі K[r] складають лінійний масив W. Довжина масиву дорівнює $\mathbf{[N_b*(Nr+1)]}$. Наприклад, Nr = 10, Nb = 4 тобто масив W містить 44 слова по 32 біти. Ключі обробляються за наступними угодами:

для **(i = 0; i < N_k; i ++), $N_k \le 6$** маємо:

W [i] = (Key[4 * i], Key[4 * i +1], Key[4 * i +2], Key[4 * i +3]);

для **(i = N_k; i < N_b* (N_r + 1); i ++)** маємо:

W [i] = W [i - N_k] ^ temp,

temp = W [i – 1], або якщо **i mod N_k = 0**, то

temp = SubByte (RotByte (temp)) ^ Rcon [i / N_k].

У цьому записі функція **SubByte** відповідає заміщенню кожного байту з 4-байтового слова на певний байт з таблиці заміщення (**S – box**). Функція **RotByte** відповідає зсуву слова ліворуч. Наприклад, вхідне слово (a,b,c,d) перетворюється на слово (b,c,d,a)

Можна побачити, що перші N_k слів складаються з початкового ключа. Кожне наступне слово **W [i]** є результатом виконання операції **EXOR** у парі з попереднім словом для випадків, коли **i** не кратне N_k. Для кратних позицій замість **W [i – 1]** використовується перетворення з раундовою константою. Зв'язок між константою та N_k такий:

Rcon [i] = (RC[i], '00', '00', '00'),

RC[i] представляє собою елемент поля Галуа **GF (2^8)** з коефіцієнтом $\mathbf{x}^{(i-1)}$ таким що:

RC[1] = 1 (така що '01')

RC[i] = x (така що '02') · (RC[i - 1]) = $\mathbf{x}^{(i-1)}$.

Перетворення типу заміна байтів (Substitution Bytes).

Заміна байтів (Substitution Bytes) – перетворення, де кожний байт блоку замінюється на певне число із таблиці (так званий S-box). Це перетворення є нелінійним, виконується з байтами результатів кожного стану незалежно. У специфікації алгоритма таблиці (S-box) для шифрування та дешифрування наведені, тому не має потреби їх обчислювати в процесі роботи програмного забезпечення. Перетворення є зворотнім та складається з композиції двох трансформацій:

1. обчислення зворотнього елемента відносно множення;
2. афінна трансформація (над полем **GF (2)**.

Система команд мікроконтролерів PIC16 дозволяє проводити заміну байтів за допомогою команди **RETLW** (повернення з підпрограми із завантаженням константи до акумулятора W)

Перетворення типу зсув рядків (Shift rows).

Зсув рядків (Shift rows) – трансформація, яка виконується шляхом зсуву рядків прямокутної матриці станів в залежності від N_b (див. табл. 2). Рядок 0 не зсувається, рядок 1 зсувається на C1 байт, рядок 2 відповідно – на C1 байт та рядок 3 – на C1 байт.

Таблиця 2 – Зсув рядків в залежності від N_b

N_b	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Система команд мікроконтролерів PIC16 дозволяє проводити трансформацію блоків за допомогою команди **RLF** (зсув ліворуч через перенесення).

Перетворення типу змішування стовпців (Multiplay columns).

Змішування стовпців (Multiplay columns) – трансформація шляхом множення кожного стовпця матриці стану на фіксовану матрицю за правилами арифметичних операцій у полі $GF(2^8)$:

b(x) = c(x) ·a(x) mod m(x), де **c(x) = '03' x^3 + '01' x^2 + '01' x + '02' , m(x) = x^4 + 1**,

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}.$$

Інверсна до змішування стовпців **(Multiplay columns)** операція розраховується з рівняння

c(x) ·d(x) mod m(x) = 1, звідки:

d(x) = '0B' x^3 + '0D' x^2 + '09' x + '0E'.

Наприклад, B0 = (2 * 5A) xor (3 * 11) xor (1 * FD) xor (1 * 89), де
2 * 5A = xtime (5A) = B4, 3 * 11 = 11 xor (xtime(11)) = 11 xor 22 = 33, 1 * FD = FD, 1 * 89 = 89
B0 = B4 xor 33 xor FD xor 89 = F3.

Операція xtime означає зсув ліворуч, якщо отриманий результат менший, ніж 80h. У випадку, коли результат більше або дорівнює 80h, додатково виконується операція „+" результату з константою 1Bh.

Виконання змішування стовпців за допомогою системи команд мікроконтролерів PIC16 не викликає труднощів, але потребує відносно багато циклів.

Перетворення типу додавання ключа (Xored by key).

Додавання ключа (Xored by key) – трансформація за рахунок логічного XOR з ключем.

Графічне представлення усіх операцій алгоритму шифрування представлено на рис. 2.

Оцінка програмної складності алгоритму на асемблері мікроконтролера PIC16XXX.

Для якісного порівняння розглянуто витрати на програмну реалізацію алгоритму DES. Відповідні результати наведено в табл. 3 та 4.

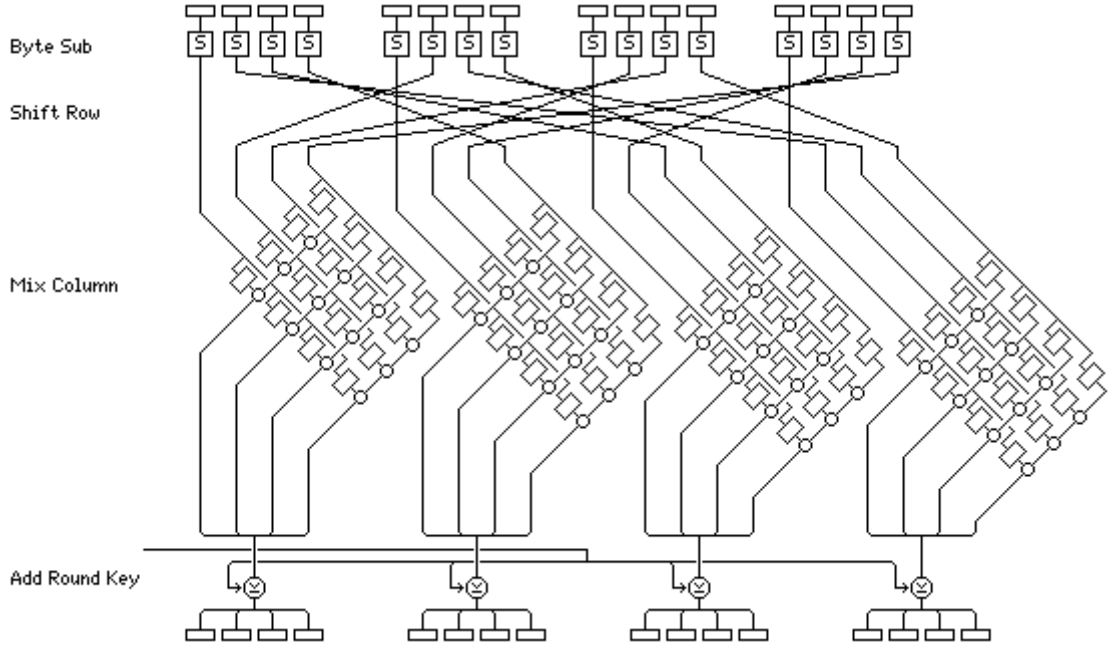

Рисунок 2 – Графічне представлення операцій алгоритму шифрування

Таблиця 3 – Витрати програмних ресурсів для алгоритму DES [4].

Функція	Програмна пам'ять	Кількість машинних циклів	Час, мс (160 нс/цикл)
Генерація ключів	382	2729	0,436
Шифрування	789	7714	1,234

Таблиця 4 – Витрати програмних ресурсів для алгоритму AES [5].

Функція	Програмна пам'ять	Кількість машинних циклів	Час, мс (160 нс/цикл)
Генерація ключів	79	652	0,104
Шифрування	659	4623	0,739

Максимальна швидкість для алгоритму DES становить 51 кбіт/с, а для алгоритму AES при рівних умовах – близько 100 кбіт/с. Підрахунок швидкості зроблено за умови, що час, витрачений на шифрування, не може перевищувати час на передачу одного інформаційного блоку.

III Висновки

1. Використання у алгоритмі шифрування AES вузлів заміни на основі арифметичних операцій в скінчених полях забезпечує суттєве спрощення програмно-апаратної реалізації. Існує практична можливість розробки криптопроцесору на базі дешевих мікроконтролерів з відносно малими вимогами до програмних ресурсів. При цьому гарантована швидкість обробки інформації становить 64 кбіт/с.

2. Наявність безкоштовно розповсюджуваного налагоджувального програмного забезпечення (ПЗ) фірми *Microchip* типу MPLAB дозволяє організувати його використання в учбовому процесі для підготовки студентів за фахом „Інформаційні технології та захист інформації" для практичного вивчення особливостей функціонування та побудови алгоритму AES на основі мікроконтролерів. Важливо підкреслити, що забезпечується легальне використання ПЗ з оглядом на Розпорядження КМ України № 247 – р від 15 травня 2002 р. „Про затвердження Концепції легалізації програмного забезпечення та боротьби з нелегальним його використанням". З іншого боку, налагоджувальне ПЗ розробки надвеликих інтегральних схем (НВІС) фірм ALTERA, XILINX має вартість від сотень до тисяч доларів, тобто практично недоступне для використання у вищих навчальних закладах. Недоступною (з фінансових міркувань) є також закупівля готових проектів НВІС (core – технологія).

3. Відносна простота програмно-апаратної реалізації алгоритму AES може привести до зростання продажу засобів КЗІ на його основі, що відповідно загострить проблему використання сертифікованих засобів захисту від несанкціонованого доступу до інформації.

Література: ***1.*** *Вакка Дж. Секреты безопасности в Internet: Пер. с англ. // К.: Діалектика Киев, 1997. - 505 с.* ***2.*** *Кларк Дж., Кейн Дж. Кодирование с исправлением ошибок в системах цифровой связи: Пер. с англ. // М.: Радио и связь, 1987. -391 с.* ***3.*** *R. Lidl and H. Niederreiter, Introduction to finite fields and their application, Cambridge University Press, 1986.* ***4.*** *Microchip: Implementation of the Data Encription Standard Using PIC17C42. http://www.microchip.com.* ***5****. Microchip: Application Note AN821 Advanced Standard Using the PIC16XXX. http://www.microchip.com.*

УДК 681.3.34.

ГЕНЕРАТОР ТЕСТОВОЇ ПОСЛІДОВНОСТІ ІКМ 30 (Е1) ІНТЕРФЕЙСУ

Всеволод Семенюк, Володимир Кишкан, Геннадій Леоненко*

ДСТСЗІ СБ України

** СФ СБ України ВІТІ НТУУ "КПІ"*

Анотація: **Розглянуто генератор тестової послідовності для забезпечення досліджень ПЕМВН цифрового комутаційного обладнання, що має інтерфейс ІКМ-30 (Е1) згідно з рекомендаціями CCIT G.704 для ІКМ 30.**

Summary: **In the article designing test generator for CEPT trunk digital link interface conforming to CCIT Recommendation G.704 for PCM 30.**

Ключові слова: **Тестове обладнання, ІКМ 30.**

I Вступ

Проведення досліджень побічних електромагнітних випромінювань та наведень (ПЕВМН) певною мірою залежить від пошуку інформативних складових сигналу у цифровій телекомунікаційній техніці, що перевіряється. Для спрощення цього завдання формують відповідні тестові сигнали (ТС), що відповідають наступним вимогам: ідентичність робочому сигналу; максимальне спектральне навантаження; помітність на фоні завад.

Як генератор ТС можна використовувати, по-перше, стандартну апаратуру. Але вона часто є багатофункціональною, має значні габаритні розміри, складається із багатьох блоків, що не дозволяє використовувати її як мобільний комплекс і викликає певні труднощі при проведенні дослідження ПЕВМН. По-друге, можна використати обладнання, яке саме проходить дослідження завдяки введенням у тестовий режим, завантаживши відповідне програмне забезпечення (якщо це передбачено). По-третє, можна розробити спеціальне додаткове оснащення порівняно малої вартості, яке б виконувало необхідні функції у мобільному варіанті.

Останній шлях розглянемо більш детально.