квантувача. Ще одним методом збільшення швидкодії схеми адаптації є фіксація моментів адаптації. Проте в цьому випадку можлива початкова похибка перехідного режиму, яка після спрацьовування схеми адаптації швидко зменшується до значення мінімального кроку $\pm s_{min}^{(x)}$.

Поєднання обох методів збільшення швидкодії схеми адаптації призводить до максимально простої схеми, в якій повністю відсутній АЦП. Значення початкової похибки перехідного режиму може становити $\pm s_{max}^{(x)}$, яке в процесі адаптації зменшується до відповідно до $\pm s_{min}^{(x)}$.

Висновки

Застосування спрощених алгоритмів адаптації при кодуванні нестаціонарних сигналів із заданою точністю призводить до максимально простих реалізацій кодерів, які можуть з успіхом застосовуватись для обробки сигналів різної природи.

1. Тимченко О.В. Різницеві методи цифрової фільтрації. Львів, 1999.
2. Дурняк Б.В., Стрепко І.Т., Тимченко О.В. Алгоритми швидкодіючих систем реального часу, побудованих на основі різницевих підходів // Вісник ДУ "Львівська політехніка". 1999. №366. С.56-62.
3. Стрепко І.Т., Тимченко О.В., Дурняк Б.В. Проектування систем керування на однокристальних мікроЕОМ. К., 1998.
4. Дурняк Б., Стрепко І., Тимченко О. Розпаралелювання обчислень на основі вибору методів різницевого подання сигналів в САК реального часу // Комп'ютерні технології друкарства. Львів, 1998. С.120-123.

УДК 681.325

ТАБЛИЧНА РЕАЛІЗАЦІЯ ФУНКЦІЙ ПЕРЕТВОРЕННЯ В ПРОЦЕСОРІ ШИФРУВАННЯ ДАНИХ ЗА АЛГОРИТМОМ AES

© А. Мельник, С. Прудкий
Національний університет "Львівська політехніка"

Розглянуто алгоритм криптографічного захисту інформації за стандартом AES, основні проблеми, що виникають при його реалізації, а також запропоновано підходи до табличного обчислення основних перетворень алгоритму та скорочення обсягу таблиць.

The AES cryptographic algorithm of information protection and main problems that arise during its applications are considered. The approaches to look-up table calculation of the algorithm basic transformations and reduction of tables volume are offered.

Вступ

З появою потужних комп'ютерів кодування/декодування інформації [1] за алгоритмом *DES* (*Data Encryption Standard*) [2], що було надійне протягом останніх 20-ти років, стало неефективним. Тому уряд США оголосив конкурс на найкращий алгоритм для нового розширеного стандарту кодування даних *AES* (*Advanced Encryption Standard*), переможцем якого визнали алгоритм кодування/декодування *Rijndael*. Передбачається, що враховуючи стрімкий розвиток комп'ютерної техніки, алгоритм *Rijndael* буде надійно захищати інформацію понад двадцять наступних років [3].

В літературі можна знайти лише теоретичні основи нового алгоритму і невелику кількість його програмних реалізацій [4], [5], [6], а згадки щодо апаратної реалізації алгоритму, а отже й про архітектуру *AES* криптопроцесорів, взагалі відсутні.

У даній статті запропоновано підхід до табличного обчислення основних перетворень алгоритму *AES* при апаратній реалізації архітектури *AES* криптопроцесора.

Для більшої відповідності назви функціональних операторів алгоритму *Rijndael* подані англійською мовою згідно з оригіналом або українською, але разом з їх англійським аналогом, поданим у дужках.

1. Опис алгоритму

1.1 Входи/виходи алгоритму

Алгоритм *Rijndael* є ітеративним симетричним блоковим шифром, що має змінну довжину блоків даних і різні довжини ключів. Алгоритм взаємодіє із зовнішнім середовищем за допомогою вхідного, вихідного блоків і блоку ключа кодування (*Cipher Key*). Кожний з цих блоків складається з послідовності бітів, розмір яких може бути 128, 192 або 256 біт, з одним лише обмеженням – розмір вхідної і вихідної послідовностей повинні збігатися. Блок ключа і блоки даних можуть мати різну довжину незалежно один від одного.

Всі перетворення, що визначені в алгоритмі, відбуваються над проміжним блоком *State*. Блок *State*, як і блоки вхідний, вихідний та ключа кодування є матрицями чотирибайтових слів розмірністю 4**Nb* (для даних) і 4**Nk* (для ключів), де *Nb* і *Nk* дорівнюють розмірам блоків даних і ключів, поділених на 32 (4 байти) відповідно.

Вхідні дані позначаються як байти *State* в порядку $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$. Після завершення дії перетворення *Round* вихідні дані беруться із блоку *State* у тому ж порядку.

Число *Nr* застосувань перетворень *Round* є функцією від розмірів ключа кодування *Nk* і блоку даних *Nb* (при $Nb = Nk = 4$ – $Nr = 10$).

1.2 Процес кодування

На початку кодування вхідні дані заносяться у блок *State* (рис.1). Пізніше відбувається проміжне додавання (функція *EXOR*) матриць *State* і матриці *Round Key* з записом результату в блок *State*. *Round Key* отримується з матриці *Key Expanded* (розширення матриці *Cipher Key* алгоритмом *Key Schedule*) і її розмір відповідає розміру блоку *State*. Далі над *State* здійснюють *Round* перетворення *Nr* – 1 разів. Завершується процес кодування перетворенням *FinalRound* – дещо видозмінене перетворення *Round*.

Функція *Round*, як і весь алгоритм кодування, містить чотири індивідуальні функції

перетворення – *SubBytes*, *ShiftRows*, *MixColumns*, та *AddRoundKey* (всі вони змінюють вміст *State*). Функція *FinalRound* – це функція *Round*, в якій відсутнє перетворення *MixColumns*.

Перетворення *SubBytes* передбачає нелінійну заміну байтів і виконується незалежно над кожним байтом *State*. Таблиці заміни (або *S*-блоки) побудовані на композиції двох перетворень:

1) Отримання оберненого елементу відносно операції множення в полі $GF(2^8)$. '00' переходить сам в себе.

Обернений елемент $b(x)$ до $a(x)$ шукається з рівняння $b(x) * a(x) \bmod m(x) = 1$, де $b(x)*a(x)$ – операція множення двох поліномів, $c(x) \bmod m(x)$ – операція знаходження остачі від ділення $c(x)$ на $m(x)$. Тут $m(x) = 11Bh$.

2) Застосування афінного перетвореня (в полі $GF(2)$):

$$b'_i = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i,$$

де b_i є i-тим бітом кожного байта і C_i є i-тим бітом байта, значення якого {63}, ($0 \le i < 8$).

Тут кожний біт окремого байта алгоритмічно причетний до бітової зміни цього ж байта.

Перетворення *ShiftRows* передбачає циклічний зсув вліво старших трьох рядків матриці *State* на різну кількість байтів. Рядок 1 зсувається на $C1$ байт, рядок 2 – на $C2$, рядок 3 – на $C3$ байти. Рядок 0 не змінюється.

Значення $C1$, $C2$ и $C3$ залежать від довжини блоку даних *Nb*. Для $Nb = 4$ – $C1 = 1$, $C2 = 2$, $C3 = 3$.

При перетворенні *MixColumns* стовпці *State* розглядаються як багаточлени $GF(28)$, що множаться за модулем $x^4 + 1$ на многочлен $a(x) = \{03\}\ x^3 + \{01\}\ x^2 + \{01\}\ x + \{02\}$. Це перетворення можна подати у вигляді множення двох матриць $s'(x) = a(x) \oplus s(x)$. Результатом перемноження буде заміна байтів кожного рядка відповідно такими значеннями:

$$s'_{0,c} = (\{02\} \cdot s_{0,c}) \oplus (\{03\} \cdot s_{1,c}) \oplus s_{2,c} \oplus s_{3,c};$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \cdot s_{1,c}) \oplus (\{03\} \cdot s_{2,c}) \oplus s_{3,c};$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \cdot s_{2,c}) \oplus (\{03\} \cdot s_{3,c});$$

$$s'_{3,c} = (\{03\} \cdot s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \cdot s_{3,c}),$$

де $0 \le c < Nb$.

Перетворення *AddRoundKey* передбачає додавання (використовується звичайна операція *EXOR*) матриці *Round Key* до матриці *State*. Кожна матриця *Round Key* містить *Nb* слів матриці *Key Expanded*, що утворена алгоритмом *Key Schedule*. Довжина *Round Key* дорівнює довжині *State*.

1.3 Алгоритм вироблення ключів (Key Schedule)

Блоки *Round Key* утворюються з ключа кодування (*Cipher Key*) за допомогою алгоритму *Key Schedule*. Цей алгоритм передбачає виконання двох дій: розширення

ключа (*Key Expansion*) – утворюється матриця *Expanded Key*, та вибір з неї *Round Key* блоків (*Round Key Selection*). Алгоритм *Key Schedule* передбачає виконання наступних кроків:

- Загальний розмір блоку *Expanded Key* дорівнює довжині блоку даних, помноженій на число циклів (*Nr*) плюс 1.
- Блок *Cipher Key* розширюється в блок розширеного ключа (*Expanded Key*).
- *Round Key* блоки беруться з *Expanded Key* таким чином: перший блок *Round Key* містить перші *Nb* слів *Expanded Key*, другий – наступні *Nb* слів і т.д.

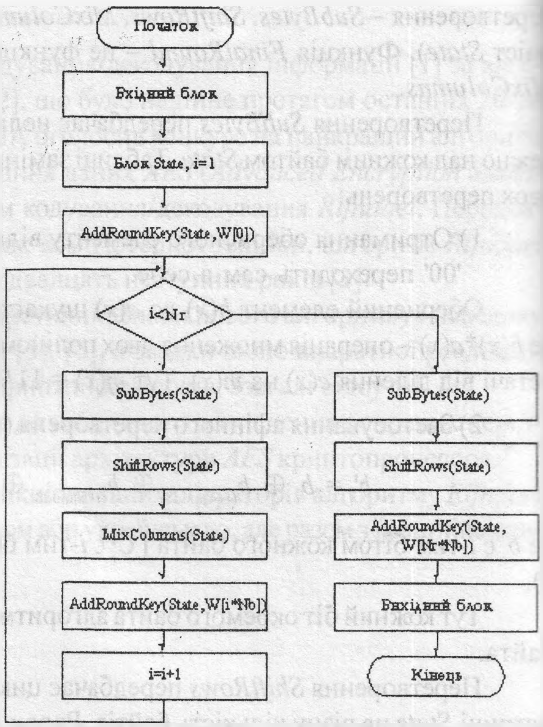

Рис.1. Блок-схема алгоритму кодування.

Блок *Expanded Key* є лінійним масивом чотирибайтових слів і позначається як $W[Nb * (Nr + 1)]$. Алгоритм вироблення ключів залежить від величини *Nk*. Перші *Nk* слів містять ключ шифрування (*Cipher Key*). Всі інші слова визначаються рекурсивно зі слів з меншими індексами. Кожне наступне слово $W[i]$ отримується за допомогою операції *EXOR* попереднього слова $W[i-1]$ і слова на *Nk* позицій раніше $W[i-Nk]$. Для слів, позиція яких кратна *Nk*, перед згадуваною операцією *EXOR* здійснюється перетворення над $W[i-1]$. При $Nk<=6$ це перетворення передбачає циклічний зсув на один байт вліво слова $W[i-1]$, над результатом якого здійснюється перетворення SubBytes і додається кодова константа. При $Nk > 6$ над кожним четвертим словом $W[i-1]$ здійснюється лише перетворення *SubBytes*.

Кодова константа залежить також від *Nk* і визначається так:

$Rcon[i] = (RC[i], '00', '00', '00')$, де $RC[0] = '01'$, а $RC[i] = \text{xtime}(Rcon[i-1])$, де функція xtime(*R*) здійснює зсув змінної *R* на один біт вліво.

Кожний *i*-ий *Round Key* є масивом з *Key Expanded* від $W[Nb * i]$ до $W[Nb(i+1)]$ слів.

1.4 Процес декодування

Алгоритм процесу декодування є зворотним до алгоритму процесу кодування. Тут використовуються наступні перетворення:

Перетворення *Inverse ShiftRows* є оберненим до перетворення *ShiftRows*. За цим перетворенням старші три рядки матриці State циклічно зсуваються вліво на різну кількість байтів. Рядок 1 зсувається на *C*1 байт, рядок 2 – на *C*2, рядок 3 – на *C*3 байти. Рядок 0 не змінюється.

Значення *C*1, *C*2 и *C*3 залежать від довжини блоку даних *Nb*. Для $Nb = 4$ – $C1 = 3$, $C2 = 2$, $C3 = 1$.

Перетворення *Inverse SubBytes* є оберненим до перетворення *SubBytes*. Обернене афінне перетворення (в полі $GF(2)$) здійснюється за формулою:

$$b'_i = b_{(i+2)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus d_i,$$

де байт $d = \{05\}$.

Перетворення *Inverse AddRoundKey* є аналогічним перетворенню *AddRoundKey*.

Перетворення *Inverse MixColumns* є оберненим до перетворення *MixColumns* і його можна представити у вигляді множення двох матриць: $s'(x) = a^{-1}(x) \otimes s(x)$, де $a^{-1}(x)$ - обернена матриця до матриці $a(x)$ в полі $GF(2^8)$. Результатом такого множення буде заміна байтів кожного рядка відповідно такими значеннями:

$$s'_{0,c} = (\{0e\} \cdot s_{0,c}) \oplus (\{0b\} \cdot s_{1,c}) \oplus (\{0d\} \cdot s_{2,c}) \oplus (\{09\} \cdot s_{3,c});$$

$$s'_{1,c} = (\{09\} \cdot s_{0,c}) \oplus (\{0e\} \cdot s_{1,c}) \oplus (\{0b\} \cdot s_{2,c}) \oplus (\{0d\} \cdot s_{3,c});$$

$$s'_{2,c} = (\{0d\} \cdot s_{0,c}) \oplus (\{09\} \cdot s_{1,c}) \oplus (\{0e\} \cdot s_{2,c}) \oplus (\{0b\} \cdot s_{3,c});$$

$$s'_{3,c} = (\{0b\} \cdot s_{0,c}) \oplus (\{0d\} \cdot s_{1,c}) \oplus (\{09\} \cdot s_{2,c}) \oplus (\{0e\} \cdot s_{3,c}),$$

де $0 \le c < Nb$.

Для декодування кодованих даних використовується "інверсний" до кодового ключ. Він утворюється застосуванням перетворення *MixColumns* над кожним чотирибайтовим словом блоків *Round Key*. Винятком є перший і останній блоки *Round Key* з блоку *Key Expanded*, які використовуються без змін.

2. Таблична реалізація функцій перетворення

Подамо алгоритм кодування в матричній формі. Функції *AddRoundKey* та *MixColumns* виглядатимуть так:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}, \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \oplus \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}.$$

Функції *ShiftRows* та *SubBytes* виглядатимуть так:

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-C1} \\ b_{2,j-C2} \\ b_{3,j-C3} \end{bmatrix}, b_{i,j} = S\left[a_{i,j}\right].$$

В загальному процес кодування буде мати такий вигляд:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j-C1}] \\ S[a_{2,j-C2}] \\ S[a_{3,j-C3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}.$$

Множення матриць можемо подати як лінійну комбінацію векторів:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} S[a_{0,j}] \bullet 02 \\ S[a_{0,j}] \bullet 01 \\ S[a_{0,j}] \bullet 01 \\ S[a_{0,j}] \bullet 03 \end{bmatrix} \oplus \begin{bmatrix} S[a_{1,j-C1}] \bullet 03 \\ S[a_{1,j-C1}] \bullet 02 \\ S[a_{1,j-C1}] \bullet 01 \\ S[a_{1,j-C1}] \bullet 01 \end{bmatrix} \oplus \begin{bmatrix} S[a_{2,j-C2}] \bullet 01 \\ S[a_{2,j-C2}] \bullet 03 \\ S[a_{2,j-C2}] \bullet 02 \\ S[a_{2,j-C2}] \bullet 01 \end{bmatrix} \oplus \begin{bmatrix} S[a_{3,j-C3}] \bullet 01 \\ S[a_{3,j-C3}] \bullet 01 \\ S[a_{3,j-C3}] \bullet 03 \\ S[a_{3,j-C3}] \bullet 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}.$$

З аналізу отриманого можна прийти до висновку, що знаходження результату функції *SubBytes* для одного байту і множення цього результату на вектор-константу можна реалізувати за допомогою однієї табличної пам'яті розміром 256 32-розрядних слів – $T[a_{i,j}]$. Тоді весь процес кодування виглядатиме так:

$$[e_j] = T[a_{0,j}] \oplus (T[a_{1,j-C1}] \oplus Rotbyte(T[a_{2,j-C2}] \oplus Rotbyte(T[a_{3,j-C3}]))) \oplus k_j, \qquad (1)$$

де функція *Rotbyte*(*word*) – це циклічний зсув *word* (32-розрядний результат табличної пам'яті) вліво на один байт.

Процес декодування даних в матричній формі виглядатиме так:

$$\begin{bmatrix} ie_{0,j} \\ ie_{1,j} \\ ie_{2,j} \\ ie_{3,j} \end{bmatrix} = \begin{bmatrix} iS[a_{0,j}] \bullet 0E \\ iS[a_{0,j}] \bullet 09 \\ iS[a_{0,j}] \bullet 0D \\ iS[a_{0,j}] \bullet 0B \end{bmatrix} \oplus \begin{bmatrix} iS[a_{1,j+C1}] \bullet 0B \\ iS[a_{1,j+C1}] \bullet 0E \\ iS[a_{1,j+C1}] \bullet 09 \\ iS[a_{1,j+C1}] \bullet 0D \end{bmatrix} \oplus \begin{bmatrix} iS[a_{2,j+C2}] \bullet 0D \\ iS[a_{2,j+C2}] \bullet 0B \\ iS[a_{2,j+C2}] \bullet 0E \\ iS[a_{2,j+C2}] \bullet 09 \end{bmatrix} \oplus \begin{bmatrix} iS[a_{3,j+C3}] \bullet 09 \\ iS[a_{3,j+C3}] \bullet 0D \\ iS[a_{3,j+C3}] \bullet 0B \\ iS[a_{3,j+C3}] \bullet 0E \end{bmatrix} \oplus \begin{bmatrix} ik_{0,j} \\ ik_{1,j} \\ ik_{2,j} \\ ik_{3,j} \end{bmatrix}.$$

Знаходження результату функції *Invers SubBytes* для одного байту і множення цього результату на вектор-константу можна реалізувати за допомогою однієї табличної пам'яті розміром 256 32-розрядних слів – $iT[a_{i,j}]$. Тоді весь процес декодування виглядатиме так:

$$[ie_j] = iT[a_{0,j}] \oplus (iT[a_{1,j+C1}] \oplus Rotbyte(iT[a_{2,j+C2}] \oplus Rotbyte(iT[a_{3,j+C3}]))) \oplus ik_j, \qquad (2)$$

Проаналізувавши алгоритм кодування (1) і алгоритм відповідного декодування (2), можна побачити певну відповідність – порядок слідування змістових перетворень однаковий. Це дає змогу реалізувати алгоритм кодування і декодування на одному тракті даних.

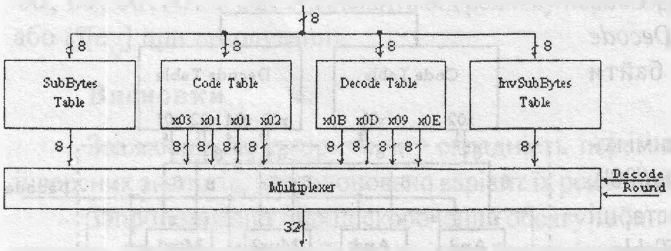

Рис.2. Оновлена пара таблиць з двома додатковими таблицями

Отже, для того щоби реалізувати алгоритм кодування/декодування в одному тракті даних, необхідно включити в цей тракт дві таблиці $T[a_{i,j}]$ (*Code Table*) та $iT[a_{i,j}]$ (*Decode Table*), які будем називати парою таблиць так, щоби в режимі кодування байт рядка, що перетворюється, проходив через таблицю кодування, а в режимі декодування – через таблицю декодування. Через те, що рядок матриці *State* містить чотири байти, зазначених таблиць буде чотири пари (окрема пара на окремий байт рядка). Виконання перетворення *FinalRound* відмінне від виконання перетворення *Round* в різних режимах відсутністю функцій *MixColumns* та *Invers MixColumns* відповідно. Це означає, що при одному тракті даних в пару таблиць треба додати ще дві таблиці розміром 256 слів по одному байту *SubBytes Table* та *InvSubBytes Table*. У такому випадку вибирати результат табличного перетворення даних в парі відповідно до режиму (кодування чи декодування) та перетворення (*FinalRound* чи *Round*) буде апаратно складний мультиплексор (рис.2). Сигнали на лініях *Round* та *Decode* керують мультиплексором так, щоби він видавав результат згідно з типом перетворення *Round* чи *FinalRound* для різних режимів роботи алгоритму (кодування чи декодування).

3. Скорочення об'єму таблиць, що реалізують функції перетворення

Недоліком схеми на рис.2 є великий об'єм пам'яті для реалізації функцій перетворення, що становить: $A = 2 * 4 + 2 * 1 = 10$ таблиць об'ємом 256 слів по одному байту і складна організація мультиплексора. Розглянемо підхід, що дозволяє зменшити загальний обсяг табличних пам'ятей.

На рис.3 подано структуру, що дозволяє зменшити обсяг пам'яті, що необхідний на реалізацію оновленої пари на рис.2.

Як бачимо, в структурі на рис.3 використовується всього дві таблиці взамін чотирьох (рис.1) загальним обсягом 6 таблиць розміром 256 слів по одному байту, тобто на 4-ри таблиці менше від попереднього варіанту. Отже, загальний обсяг табличної пам'яті скорочується в $10 / 6 \approx 1{,}7$ рази.

Структура на рис. 3 має такі особливості:

1. Введено стиснену (відкинута надлишковість даних) за вмістом таблиця *Code Table* (256 16-розрядних слів замість 32-розрядних).
2. Введено змінену за вмістом таблицю *Decode Table*.
3. Введено логічно ускладнений мультиплексор *Logic Block*, що містить три рівні логіки (верхній – ВРЛ, середній – СРЛ, нижній – НРЛ).

ВРЛ з результатів табличних перетворень формує 4-байтове слово і видає його на проміжну шину (шина між ВРЛ та СРЛ і складається з чотирьох індивідуальних байтів a_3, a_2, a_1, a_0).

Елементи *MUX* ВРЛ видають на проміжну шину в режимі кодування (*Decode* =

0) 2-х байтовий вихід з таблиці *Code Table*, а в режимі декодування (*Decode* = 1) видають два молодші байти виходу таблиці *Decode Table*.

Елементи *AND* ВРЛ в режимі кодування видають два нуль-байти ('00'), а в режимі декодування – два старші байти виходу таблиці *Decode Table*.

Елементи *EXOR* СРЛ здійснюють побайтову операцію *EXOR* ("виключне АБО") над байтами проміжної шини a_3, a_2, a_1, a_0 і формують 4- байтове результуюче слово, що складається з чотирьох індивідуальних байтів b_3, b_2, b_1, b_0 за таким алгоритмом:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_3 \oplus a_2 \oplus a_1 \oplus 00 \\ a_3 \oplus 00 \oplus 00 \oplus a_0 \\ a_3 \oplus a_2 \oplus 00 \oplus a_0 \\ a_3 \oplus 00 \oplus a_1 \oplus a_0 \end{bmatrix}.$$

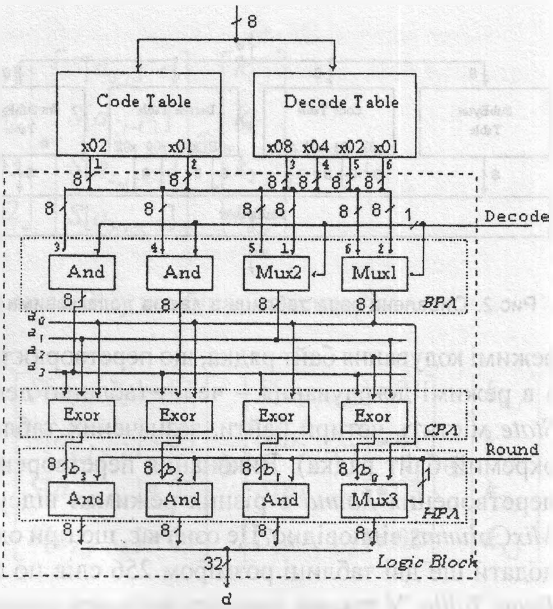

Рис.3. Структура, що зменшує вимоги щодо реалізації табличної пам'яті

У режимі кодування результат роботи СРЛ буде таким:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 00 \oplus 00 \oplus 02 \bullet S[a_{i,j}] \oplus 00 \\ 00 \oplus 00 \oplus 00 \oplus 01 \bullet S[a_{i,j}] \\ 00 \oplus 00 \oplus 00 \oplus 01 \bullet S[a_{i,j}] \\ 00 \oplus 00 \oplus 02 \bullet S[a_{i,j}] \oplus 01 \bullet S[a_{i,j}] \end{bmatrix} = \begin{bmatrix} 02 \bullet S[a_{i,j}] \\ 01 \bullet S[a_{i,j}] \\ 01 \bullet S[a_{i,j}] \\ 03 \bullet S[a_{i,j}] \end{bmatrix}.$$

У режимі декодування результат роботи СРЛ буде таким:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 08 \bullet iS[a_{i,j}] \oplus 04 \bullet iS[a_{i,j}] \oplus 02 \bullet iS[a_{i,j}] \oplus 00 \\ 08 \bullet iS[a_{i,j}] \oplus 00 \oplus 00 \oplus 01 \bullet iS[a_{i,j}] \\ 08 \bullet iS[a_{i,j}] \oplus 04 \bullet iS[a_{i,j}] \oplus 00 \oplus 01 \bullet iS[a_{i,j}] \\ 08 \bullet iS[a_{i,j}] \oplus 00 \oplus 02 \bullet iS[a_{i,j}] \oplus 01 \bullet iS[a_{i,j}] \end{bmatrix} = \begin{bmatrix} 0E \bullet iS[a_{i,j}] \\ 09 \bullet iS[a_{i,j}] \\ 0D \bullet iS[a_{i,j}] \\ 0B \bullet iS[a_{i,j}] \end{bmatrix}.$$

Робота елементів НРЛ залежить від рівня сигналу на лінії *Round*. При перетворенні *Round* сигнал *Round* = 1, а при *FinalRound* – *Round* = 0.

При перетворенні *Round* чотирибайтове слово *d* формується з чотирьох індивідуальних байтів b_3, b_2, b_1, b_0, а при перетворенні *FinalRound d* формується з байтів

'00', '00', '00', ('01'·*P*), де *P* залежить від режиму перетворення і рівне $S[a_{ij}]$ при кодуванні, або $iS[a_{ij}]$ при декодуванні.

Висновки

Зважаючи на алгоритмічну складність перетворень *SubBytes*, *MixColums* і їх інверсних аналогів, запропоновано варіант їх реалізації табличним способом.

Запропоновано підхід скорочення обсягу табличної пам'яті через використання стисненої таблиці *Code Table*, зміненої за змістом таблиці *Decode Table* та логічно ускладненого керованого відповідно до режиму і типу перетворення комутатора, що загалом дозволило в 1,7 разів зменшити загальний обсяг таблиць перетворення даних.

1. A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC Press, New York, 1997, p. 81-83.
2. FIPS 46, "Data Encryption Standard", Federal Information Processing Standard (FIPS), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C.
3. Philip Bulman. Advanced encryption standard (AES). Questions and Answers. March 5, 2001.
4. J. Daemen and V. Rijmen, AES Proposal: Rijndael, AES Algorithm Submission, September 3, 1999.
5. Dr. Brian Gladman. A Specification for Rijndael, the AES Algorithm. v3.1, 3rd March 2001, pp 1-29.
6. Federal Information Processing Standards Publication ZZZ. Announcing the Advanced Encryption Standard (AES). 2001 MONTH DAY.

УДК 681.142.2

ОРГАНІЗАЦІЯ WEB-ОРІЄНТОВАНИХ НАВЧАЛЬНИХ СИСТЕМ

© Ю. Рашкевич, Д. Пелешко, М. Пасєка, А. Стецюк
Національний університет "Львівська політехніка"

Пропонується підхід до побудови Web-орієнтованих віддалених навчальних систем в рамках української системи освіти.

There is proposed the method for design of Web-oriented remote learning systems for the Urkainian study sytem

Вступ

Протягом останніх десяти років відбувається бурхливий розвиток інформаційних та телекомунікаційних технологій (англійське скорочення - *ICT*) в навчальному процесі.