УДК 681.3.06:519.248.681

И. Д. ГОРБЕНКО, д-р техн. наук, Д. А. ЧЕКАЛИН

СВОЙСТВА И ВОЗМОЖНОСТИ ОПТИМИЗАЦИИ КРИПТОГРАФИЧЕСКИХ ПРЕОБРАЗОВАНИЙ В AES – RIJNDAEL

Введение

В конце 90-х годов в мире, в кругу специалистов-криптологов, сформулирована и решена задача создания стандарта блочного симметричного шифрования XXI века. В течение трех лет проводились три конгресса по прикладной криптологии, на которых из 15 алгоритмов к сегодняшнему дню выбрали Rijndael [1].

В соответствии с минимальными и общими требованиями к блочным симметричным алгоритмам проведены исследования, направленные на минимизацию вычислительной сложности уже существующих программных реализаций алгоритма Rijndael. Среди существующих реализаций нами была выбрана программа, написанная Брайаном Гладманом [1], обеспечивающая, по мнению авторов указанной реализации, максимальную скорость прямого и обратного криптографических преобразований, и, соответственно, имеющая минимальную вычислительную сложность.

Целью настоящей статьи является рассмотрение возможностей усовершенствования названной программной реализации с целью повышения скорости прямых и обратных преобразований.

1. Сущность цикловых табличных и криптографических преобразований

Rijndael является цикловым блочным симметричным криптоалгоритмом. Длины ключа и блока могут иметь независимо друг от друга значения 128, 192 и 256 бит. Количество циклов в алгоритме зависит от длин ключа и блока. Их зависимость отображается в табл. 1. Длины ключа и блока берутся деленными на 32.

Таблица 1

N_r	$N_b = 4$	$N_b = 6$	$N_b = 8$
$N_k = 4$	10	12	14
$N_k = 6$	12	12	14
$N_k = 8$	14	14	14

Обозначим промежуточный результат шифрования, согласно [1], как **State** (состояние). Состояние можно представить в виде прямоугольного массива байтов. Этот массив имеет 4 строки, а число столбцов обозначено как N_b и равно длине блока, деленной на 32. Цикловое преобразование состоит из четырех различных преобразований: трех табличных и одного криптографического (сложение с ключом). На псевдо-Си это выглядит следующим образом:

```
Round (State, RoundKey)
 {
ByteSub(State):                  // замена байт
ShiftRow(State);                 // сдвиг строк
MixColumn(State);                // замешивание столбцов
AddRoundKey(State, RoundKey);    // добавление циклового ключа
}
```

Программная реализация позволяет свести три табличных преобразования к одному. Обозначим *e* как результат циклового преобразования (State). Для преобразований MixColumn и сложения с ключом мы имеем:

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

Для преобразований ShiftRow и ByteSub мы имеем:

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

Подставив выражение (2) в (1) получим (3):

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

Перемножение матриц может быть представлено как линейная комбинация векторов:

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

Определим четыре таблицы $T_0 - T_3$

$$T_0[a] = \begin{bmatrix} S[a] \cdot 02 \\ S[a] \\ S[a] \\ S[a] \cdot 03 \end{bmatrix} \quad T_1[a] = \begin{bmatrix} S[a] \cdot 03 \\ S[a] \cdot 02 \\ S[a] \\ S[a] \end{bmatrix} \quad T_2[a] = \begin{bmatrix} S[a] \\ S[a] \cdot 03 \\ S[a] \cdot 02 \\ S[a] \end{bmatrix} \quad T_3[a] = \begin{bmatrix} S[a] \\ S[a] \\ S[a] \cdot 03 \\ S[a] \cdot 02 \end{bmatrix}$$

Таким образом, мы получили 4 таблицы, содержащие по 256 4-х байтных слов и занимающие около 4 Кб. Полностью таблицы $T_0 - T_3$ приведены ниже в С-транскрипции. Используя эти таблицы. представим цикловое преобразование как

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j-C1}] \oplus T_2[a_{2,j-C2}] \oplus T_3[a_{3,j-C3}] \oplus k_j$$

Учитывая достаточную сложность получения таблиц и необходимость для разработчиков «эталона» таблиц преобразований, а также то, что алгоритм RIJNDAEL рекомендован в качестве стандарта XXI века, считаем необходимым опубликование их в настоящей статье.

```
u4byte ft_tab[4][256]={{
A56363C6, 847C7CF8, 997777EE, 8D7B7BF6,  DF2F2FF, BD6B6BD6, B16F6FDE, 54C5C591,
50303060,  3010102, A96767CE, 7D2B2B56, 19FEFEE7, 62D7D7B5, E6ABAB4D, 9A7676EC,
45CACA8F, 9D82821F, 40C9C989, 877D7DFA, 15FAFAEF, EB5959B2, C947478E,  BF0F0FB,
ECADAD41, 67D4D4B3, FDA2A25F, EAAFAF45, BF9C9C23, F7A4A453, 967272E4, 5BC0C09B,
C2B7B775, 1CFDFDE1, AE93933D, 6A26264C, 5A36366C, 413F3F7E,  2F7F7F5, 4FCCCC83,
5C343468, F4A5A551, 34E5E5D1,  8F1F1F9, 937171E2, 73D8D8AB, 53313162, 3F15152A,
 C040408, 52C7C795, 65232346, 5EC3C39D, 28181830, A1969637,  F05050A, B59A9A2F,
 907070E, 36121224, 9B80801B, 3DE2E2DF, 26EBEBCD, 6927274E, CDB2B27F, 9F7575EA,
1B090912, 9E83831D, 742C2C58, 2E1A1A34, 2D1B1B36, B26E6EDC, EE5A5AB4, FBA0A05B,
F65252A4, 4D3B3B76, 61D6D6B7, CEB3B37D, 7B292952, 3EE3E3DD, 712F2F5E, 97848413,
F55353A6, 68D1D1B9,        0, 2CEDEDC1, 60202040, 1FFCFCE3, C8B1B179, ED5B5BB6,
BE6A6AD4, 46CBCB8D, D9BEBE67, 4B393972, DE4A4A94, D44C4C98, E85858B0, 4ACFCF85,
6BD0D0BB, 2AEFEFC5, E5AAAA4F, 16FBFBED, C5434386, D74D4D9A, 55333366, 94858511,
CF45458A, 10F9F9E9,  6020204, 817F7FFE, F05050A0, 443C3C78, BA9F9F25, E3A8A84B,
F35151A2, FEA3A35D, C0404080, 8A8F8F05, AD92923F, BC9D9D21, 48383870,  4F5F5F1,
DFBCBC63, C1B6B677, 75DADAAF, 63212142, 30101020, 1AFFFFE5,  EF3F3FD, 6DD2D2BF,
4CCDCD81, 140C0C18, 35131326, 2FECECC3, E15F5FBE, A2979735, CC444488, 3917172E,
```

```
57C4C493, F2A7A755, 827E7EFC, 473D3D7A, AC6464C8, E75D5DBA, 2B191932, 957373E6,
A06060C0, 98818119, D14F4F9E, 7FDCDCA3, 66222244, 7E2A2A54, AB90903B, 8388880B,
CA46468C, 29EEEEC7, D3B8B86B, 3C141428, 79DEDEA7, E25E5EBC, 1D0B0B16, 76DBDBAD,
3BE0E0DB, 56323264, 4E3A3A74, 1E0A0A14, DB494992,  A06060C, 6C242448, E45C5CB8,
5DC2C29F, 6ED3D3BD, EFACAC43, A66262C4, A8919139, A4959531, 37E4E4D3, 8B7979F2,
32E7E7D5, 43C8C88B, 5937376E, B76D6DDA, 8C8D8D01, 64D5D5B1, D24E4E9C, E0A9A949,
B46C6CD8, FA5656AC,  7F4F4F3, 25EAEACF, AF6565CA, 8E7A7AF4, E9AEAE47, 18080810,
D5BABA6F, 887878F0, 6F25254A, 722E2E5C, 241C1C38, F1A6A657, C7B4B473, 51C6C697,
23E8E8CB, 7CDDDDA1, 9C7474E8, 211F1F3E, DD4B4B96, DCBDBD61, 868B8B0D, 858A8A0F,
907070E0, 423E3E7C, C4B5B571, AA6666CC, D8484890,  5030306,  1F6F6F7, 120E0E1C,
A36161C2, 5F35356A, F95757AE, D0B9B969, 91868617, 58C1C199, 271D1D3A, B99E9E27,
38E1E1D9, 13F8F8EB, B398982B, 33111122, BB6969D2, 70D9D9A9, 898E8E07, A7949433,
B69B9B2D, 221E1E3C, 92878715, 20E9E9C9, 49CECE87, FF5555AA, 78282850, 7ADFDFA5,
8F8C8C03, F8A1A159, 80898909, 170D0D1A, DABFBF65, 31E6E6D7, C6424284, B86868D0,
C3414182, B0999929, 772D2D5A, 110F0F1E, CBB0B07B, FC5454A8, D6BBBB6D, 3A16162C
}{
6363C6A5, 7C7CF884, 7777EE99, 7B7BF68D, F2F2FF0D, 6B6BD6BD, 6F6FDEB1, C5C59154,
30306050,  1010203, 6767CEA9, 2B2B567D, FEFEE719, D7D7B562, ABAB4DE6, 7676EC9A,
CACA8F45, 82821F9D, C9C98940, 7D7DFA87, FAFAEF15, 5959B2EB, 47478EC9, F0F0FB0B,
ADAD41EC, D4D4B367, A2A25FFD, AFAF45EA, 9C9C23BF, A4A453F7, 7272E496, C0C09B5B,
B7B775C2, FDFDE11C, 93933DAE, 26264C6A, 36366C5A, 3F3F7E41, F7F7F502, CCCC834F,
3434685C, A5A551F4, E5E5D134, F1F1F908, 7171E293, D8D8AB73, 31316253, 15152A3F,
 404080C, C7C79552, 23234665, C3C39D5E, 18183028, 969637A1,  5050A0F, 9A9A2FB5,
 7070E09, 12122436, 80801B9B, E2E2DF3D, EBEBCD26, 27274E69, B2B27FCD, 7575EA9F,
 909121B, 83831D9E, 2C2C5874, 1A1A342E, 1B1B362D, 6E6EDCB2, 5A5AB4EE, A0A05BFB,
5252A4F6, 3B3B764D, D6D6B761, B3B37DCE, 2929527B, E3E3DD3E, 2F2F5E71, 84841397,
5353A6F5, D1D1B968,        0, EDEDC12C, 20204060, FCFCE31F, B1B179C8, 5B5BB6ED,
6A6AD4BE, CBCB8D46, BEBE67D9, 3939724B, 4A4A94DE, 4C4C98D4, 5858B0E8, CFCF854A,
D0D0BB6B, EFEFC52A, AAAA4FE5, FBFBED16, 434386C5, 4D4D9AD7, 33336655, 85851194,
45458ACF, F9F9E910,  2020406, 7F7FFE81, 5050A0F0, 3C3C7844, 9F9F25BA, A8A84BE3,
5151A2F3, A3A35DFE, 404080C0, 8F8F058A, 92923FAD, 9D9D21BC, 38387048, F5F5F104,
BCBC63DF, B6B677C1, DADAAF75, 21214263, 10102030, FFFFE51A, F3F3FD0E, D2D2BF6D,
CDCD814C,  C0C1814, 13132635, ECECC32F, 5F5FBEE1, 979735A2, 444488CC, 17172E39,
C4C49357, A7A755F2, 7E7EFC82, 3D3D7A47, 6464C8AC, 5D5DBAE7, 1919322B, 7373E695,
6060C0A0, 81811998, 4F4F9ED1, DCDCA37F, 22224466, 2A2A547E, 90903BAB, 88880B83,
46468CCA, EEEEC729, B8B86BD3, 1414283C, DEDEA779, 5E5EBCE2,  B0B161D, DBDBAD76,
E0E0DB3B, 32326456, 3A3A744E,  A0A141E, 494992DB,  6060C0A, 2424486C, 5C5CB8E4,
C2C29F5D, D3D3BD6E, ACAC43EF, 6262C4A6, 919139A8, 959531A4, E4E4D337, 7979F28B,
E7E7D532, C8C88B43, 37376E59, 6D6DDAB7, 8D8D018C, D5D5B164, 4E4E9CD2, A9A949E0,
6C6CD8B4, 5656ACFA, F4F4F307, EAEACF25, 6565CAAF, 7A7AF48E, AEAE47E9,  8081018,
BABA6FD5, 7878F088, 25254A6F, 2E2E5C72, 1C1C3824, A6A657F1, B4B473C7, C6C69751,
E8E8CB23, DDDDA17C, 7474E89C, 1F1F3E21, 4B4B96DD, BDBD61DC, 8B8B0D86, 8A8A0F85,
7070E090, 3E3E7C42, B5B571C4, 6666CCAA, 484890D8,  3030605, F6F6F701,  E0E1C12,
6161C2A3, 35356A5F, 5757AEF9, B9B969D0, 86861791, C1C19958, 1D1D3A27, 9E9E27B9,
E1E1D938, F8F8EB13, 98982BB3, 11112233, 6969D2BB, D9D9A970, 8E8E0789, 949433A7,
9B9B2DB6, 1E1E3C22, 87871592, E9E9C920, CECE8749, 5555AAFF, 28285078, DFDFA57A,
8C8C038F, A1A159F8, 89890980,  D0D1A17, BFBF65DA, E6E6D731, 424284C6, 6868D0B8,
414182C3, 999929B0, 2D2D5A77,  F0F1E11, B0B07BCB, 5454A8FC, BBBB6DD6, 16162C3A
}
{
63C6A563, 7CF8847C, 77EE9977, 7BF68D7B, F2FF0DF2, 6BD6BD6B, 6FDEB16F, C59154C5,
30605030,  1020301, 67CEA967, 2B567D2B, FEE719FE, D7B562D7, AB4DE6AB, 76EC9A76,
CA8F45CA, 821F9D82, C98940C9, 7DFA877D, FAEF15FA, 59B2EB59, 478EC947, F0FB0BF0,
AD41ECAD, D4B367D4, A25FFDA2, AF45EAAF, 9C23BF9C, A453F7A4, 72E49672, C09B5BC0,
B775C2B7, FDE11CFD, 933DAE93, 264C6A26, 366C5A36, 3F7E413F, F7F502F7, CC834FCC,
34685C34, A551F4A5, E5D134E5, F1F908F1, 71E29371, D8AB73D8, 31625331, 152A3F15,
 4080C04, C79552C7, 23466523, C39D5EC3, 18302818, 9637A196,  50A0F05, 9A2FB59A,
 70E0907, 12243612, 801B9B80, E2DF3DE2, EBCD26EB, 274E6927, B27FCDB2, 75EA9F75,
 9121B09, 831D9E83, 2C58742C, 1A342E1A, 1B362D1B, 6EDCB26E, 5AB4EE5A, A05BFBA0,
52A4F652, 3B764D3B, D6B761D6, B37DCEB3, 29527B29, E3DD3EE3, 2F5E712F, 84139784,
53A6F553, D1B968D1,        0, EDC12CED, 20406020, FCE31FFC, B179C8B1, 5BB6ED5B,
6AD4BE6A, CB8D46CB, BE67D9BE, 39724B39, 4A94DE4A, 4C98D44C, 58B0E858, CF854ACF,
D0BB6BD0, EFC52AEF, AA4FE5AA, FBED16FB, 4386C543, 4D9AD74D, 33665533, 85119485,
```

```
458ACF45, F9E910F9,  2040602, 7FFE817F, 50A0F050, 3C78443C, 9F25BA9F, A84BE3A8,
51A2F351, A35DFEA3, 4080C040, 8F058A8F, 923FAD92, 9D21BC9D, 38704838, F5F104F5,
BC63DFBC, B677C1B6, DAAF75DA, 21426321, 10203010, FFE51AFF, F3FD0EF3, D2BF6DD2,
CD814CCD,  C18140C, 13263513, ECC32FEC, 5FBEE15F, 9735A297, 4488CC44, 172E3917,
C49357C4, A755F2A7, 7EFC827E, 3D7A473D, 64C8AC64, 5DBAE75D, 19322B19, 73E69573,
60C0A060, 81199881, 4F9ED14F, DCA37FDC, 22446622, 2A547E2A, 903BAB90, 880B8388,
468CCA46, EEC729EE, B86BD3B8, 14283C14, DEA779DE, 5EBCE25E,  B161D0B, DBAD76DB,
E0DB3BE0, 32645632, 3A744E3A,  A141E0A, 4992DB49,  60C0A06, 24486C24, 5CB8E45C,
C29F5DC2, D3BD6ED3, AC43EFAC, 62C4A662, 9139A891, 9531A495, E4D337E4, 79F28B79,
E7D532E7, C88B43C8, 376E5937, 6DDAB76D, 8D018C8D, D5B164D5, 4E9CD24E, A949E0A9,
6CD8B46C, 56ACFA56, F4F307F4, EACF25EA, 65CAAF65, 7AF48E7A, AE47E9AE,  8101808,
BA6FD5BA, 78F08878, 254A6F25, 2E5C722E, 1C38241C, A657F1A6, B473C7B4, C69751C6,
E8CB23E8, DDA17CDD, 74E89C74, 1F3E211F, 4B96DD4B, BD61DCBD, 8B0D868B, 8A0F858A,
70E09070, 3E7C423E, B571C4B5, 66CCAA66, 4890D848,  3060503, F6F701F6,  E1C120E,
61C2A361, 356A5F35, 57AEF957, B969D0B9, 86179186, C19958C1, 1D3A271D, 9E27B99E,
E1D938E1, F8EB13F8, 982BB398, 11223311, 69D2BB69, D9A970D9, 8E07898E, 9433A794,
9B2DB69B, 1E3C221E, 87159287, E9C920E9, CE8749CE, 55AAFF55, 28507828, DFA57ADF,
8C038F8C, A159F8A1, 89098089,  D1A170D, BF65DABF, E6D731E6, 4284C642, 68D0B868,
4182C341, 9929B099, 2D5A772D,  F1E110F, B07BCBB0, 54A8FC54, BB6DD6BB, 162C3A16
}
{
C6A56363, F8847C7C, EE997777, F68D7B7B, FF0DF2F2, D6BD6B6B, DEB16F6F, 9154C5C5,
60503030,  2030101, CEA96767, 567D2B2B, E719FEFE, B562D7D7, 4DE6ABAB, EC9A7676,
8F45CACA, 1F9D8282, 8940C9C9, FA877D7D, EF15FAFA, B2EB5959, 8EC94747, FB0BF0F0,
41ECADAD, B367D4D4, 5FFDA2A2, 45EAAFAF, 23BF9C9C, 53F7A4A4, E4967272, 9B5BC0C0,
75C2B7B7, E11CFDFD, 3DAE9393, 4C6A2626, 6C5A3636, 7E413F3F, F502F7F7, 834FCCCC,
685C3434, 51F4A5A5, D134E5E5, F908F1F1, E2937171, AB73D8D8, 62533131, 2A3F1515,
 80C0404, 9552C7C7, 46652323, 9D5EC3C3, 30281818, 37A19696,  A0F0505, 2FB59A9A,
 E090707, 24361212, 1B9B8080, DF3DE2E2, CD26EBEB, 4E692727, 7FCDB2B2, EA9F7575,
121B0909, 1D9E8383, 58742C2C, 342E1A1A, 362D1B1B, DCB26E6E, B4EE5A5A, 5BFBA0A0,
A4F65252, 764D3B3B, B761D6D6, 7DCEB3B3, 527B2929, DD3EE3E3, 5E712F2F, 13978484,
A6F55353, B968D1D1,        0, C12CEDED, 40602020, E31FFCFC, 79C8B1B1, B6ED5B5B,
D4BE6A6A, 8D46CBCB, 67D9BEBE, 724B3939, 94DE4A4A, 98D44C4C, B0E85858, 854ACFCF,
BB6BD0D0, C52AEFEF, 4FE5AAAA, ED16FBFB, 86C54343, 9AD74D4D, 66553333, 11948585,
8ACF4545, E910F9F9,  4060202, FE817F7F, A0F05050, 78443C3C, 25BA9F9F, 4BE3A8A8,
A2F35151, 5DFEA3A3, 80C04040,  58A8F8F, 3FAD9292, 21BC9D9D, 70483838, F104F5F5,
63DFBCBC, 77C1B6B6, AF75DADA, 42632121, 20301010, E51AFFFF, FD0EF3F3, BF6DD2D2,
814CCDCD, 18140C0C, 26351313, C32FECEC, BEE15F5F, 35A29797, 88CC4444, 2E391717,
9357C4C4, 55F2A7A7, FC827E7E, 7A473D3D, C8AC6464, BAE75D5D, 322B1919, E6957373,
C0A06060, 19988181, 9ED14F4F, A37FDCDC, 44662222, 547E2A2A, 3BAB9090,  B838888,
8CCA4646, C729EEEE, 6BD3B8B8, 283C1414, A779DEDE, BCE25E5E, 161D0B0B, AD76DBDB,
DB3BE0E0, 64563232, 744E3A3A, 141E0A0A, 92DB4949,  C0A0606, 486C2424, B8E45C5C,
9F5DC2C2, BD6ED3D3, 43EFACAC, C4A66262, 39A89191, 31A49595, D337E4E4, F28B7979,
D532E7E7, 8B43C8C8, 6E593737, DAB76D6D,  18C8D8D, B164D5D5, 9CD24E4E, 49E0A9A9,
D8B46C6C, ACFA5656, F307F4F4, CF25EAEA, CAAF6565, F48E7A7A, 47E9AEAE, 10180808,
6FD5BABA, F0887878, 4A6F2525, 5C722E2E, 38241C1C, 57F1A6A6, 73C7B4B4, 9751C6C6,
CB23E8E8, A17CDDDD, E89C7474, 3E211F1F, 96DD4B4B, 61DCBDBD,  D868B8B,  F858A8A,
E0907070, 7C423E3E, 71C4B5B5, CCAA6666, 90D84848,  6050303, F701F6F6, 1C120E0E,
C2A36161, 6A5F3535, AEF95757, 69D0B9B9, 17918686, 9958C1C1, 3A271D1D, 27B99E9E,
D938E1E1, EB13F8F8, 2BB39898, 22331111, D2BB6969, A970D9D9,  7898E8E, 33A79494,
2DB69B9B, 3C221E1E, 15928787, C920E9E9, 8749CECE, AAFF5555, 50782828, A57ADFDF,
 38F8C8C, 59F8A1A1,  9808989, 1A170D0D, 65DABFBF, D731E6E6, 84C64242, D0B86868,
82C34141, 29B09999, 5A772D2D, 1E110F0F, 7BCBB0B0, A8FC5454, 6DD6BBBB, 2C3A1616
}};
```

Расшифрование в алгоритме Rijndael осуществляется с использованием таблицы расшифрования, согласованной с таблицей зашифрования.

2. Оптимизация программной реализации

Проведенный анализ наилучшей программной реализации Rijndael показал, что существует возможность уменьшения вычислительной сложности даже в этой реализации. В известной наиболее быстрой реализации значительной сложностью обладает операция вычисления адреса. Этих затрат

можно избежать, учитывая малую длину цикла. Сущность оптимизации заключается в «разворачивании» цикла и прямого указания адресов (вместо их вычисления) в функциях зашифрования-расшифрования.

Рассмотрим известную [1] функцию зашифрования с точки зрения ее программной реализации:

```
void encrypt(const u4byte in_blk[4], u4byte out_blk[4])
{   u4byte  i, b0[4], b1[4];

    b0[0] = in_blk[0] ^ e_key[0];
    b0[1] = in_blk[1] ^ e_key[1];
    b0[2] = in_blk[2] ^ e_key[2];
    b0[3] = in_blk[3] ^ e_key[3];

    for(i = 1; i < k_len + 6; ++i)
    {
        f_rn(b1, b0, 0); f_rn(b1, b0, 1);
        f_rn(b1, b0, 2); f_rn(b1, b0, 3);

        b0[0] = b1[0] ^ e_key[4 * i];
        b0[1] = b1[1] ^ e_key[4 * i + 1];
        b0[2] = b1[2] ^ e_key[4 * i + 2];
        b0[3] = b1[3] ^ e_key[4 * i + 3];
    }

    f_rl(b1, b0, 0); f_rl(b1, b0, 1);
    f_rl(b1, b0, 2); f_rl(b1, b0, 3);

    out_blk[0] = b1[0] ^ e_key[4 * k_len + 24];
    out_blk[1] = b1[1] ^ e_key[4 * k_len + 25];
    out_blk[2] = b1[2] ^ e_key[4 * k_len + 26];
    out_blk[3] = b1[3] ^ e_key[4 * k_len + 27];
};
```

Видно, что операция вычисления адреса производится достаточно часто. При вычислении используются операции умножения и сложения. Цикл в оригинальной функции зашифрования используется для реализации циклового преобразования. Оптимизация производится путем записи функции зашифрования в виде макросов, так как макросы подставляются в программный код, и все необходимые предвычисления выполняются компилятором. Разобьем программу на три части: начального, циклового преобразования и завершающую согласно тексту программы, приведенному выше.

Обозначим начальную часть (операцию сложения с ключом) в виде макроса **e_rn**:

```
#define e_rn(bo, bi, n)           \
  bo[0] = bi[0] ^ e_key[n];       \
  bo[1] = bi[1] ^ e_key[n + 1];   \
  bo[2] = bi[2] ^ e_key[n + 2];   \
  bo[3] = bi[3] ^ e_key[n + 3]
```

Обозначим цикловое преобразование в виде макроса **e_round**:

```
#define e_round(bo, bi, n)        \
  f_rn(bo, bi, 0);                \
  f_rn(bo, bi, 1);                \
  f_rn(bo, bi, 2);                \
  f_rn(bo, bi, 3);                \
  e_rn(bi, bo, n)
```

Обозначим завершающую часть в виде макроса **e_final**:

```
#define e_final(bo, bi, n)    \
  f_rl(bo, bi, 0);            \
  f_rl(bo, bi, 1);            \
  f_rl(bo, bi, 2);            \
  f_rl(bo, bi, 3);            \
  e_rn(out_blk, bo, n)
```

Полностью предлагаемая альтернативная функция зашифрования имеет вид:

```
void encrypt(const u4byte in_blk[4], u4byte out_blk[4])
{   u4byte  b0[4], b1[4];

  e_rn(b0, in_blk, 0);     e_round(b1, b0,  4);
  e_round(b1, b0,  8);  e_round(b1, b0, 12);
  e_round(b1, b0, 16);  e_round(b1, b0, 20);
  e_round(b1, b0, 24);  e_round(b1, b0, 28);
  e_round(b1, b0, 32);  e_round(b1, b0, 36);

    switch(k_len)
    {
  case 4:       e_final(b1, b0, 40);        break;

  case 6:       e_round(b1, b0, 40);        e_round(b1, b0, 44);
                e_final(b1, b0, 48);        break;

  case 8:       e_round(b1, b0, 40);        e_round(b1, b0, 44);
                e_round(b1, b0, 48);        e_round(b1, b0, 52);
                 e_final(b1, b0, 56);
    }
};
```

Таким образом, оптимизированная функция зашифрования имеет явные преимущества перед ригинальной: отсутствует цикл, не выполняются операция вычисления адреса (указывается явно).

3. Оценка вычислительной сложности

Произведем оценку вычислительной сложности. В качестве показателя вычислительной сложности функций зашифрования-расшифрования используется количество команд (тактов), необходимых для выполнения функций с учетом вероятности выполнения каждой операции. Сразу заметим, что асчет будет учитывать элементарные операции и не будет учитывать затраты на реализацию цикла, ледовательно, будет приблизительным. В данном случае такое упрощение допустимо, так как при ычете оптимизированной версии алгоритма такое упрощение также будет иметь место. Вычислительная сложность для функций зашифрования-расшифрования в обеих реализациях одинакова.

Проведем непосредственный анализ предлагаемой программной реализации вычислительной ожности цикловых преобразований. При этом вычислительная сложность известных функций зашифрования-расшифрования составляет:

$$I_1 = 41 + 35 \bullet N_r \text{ (тактов)}, \tag{4}$$

N_r – число циклов, зависящее от значений длины блока N_b и длины ключа N_k. Их зависимость показывается в таблице 1.

Вычислительная сложность оптимизированных функций зашифрования-расшифрования состав-т:

$$I_2 = 40 + 28 \bullet N_r \text{ (тактов)} \tag{5}$$

Произведем теоретическую оценку сложности известного и оптимизированного алгоритмов. Результаты оценки приведены в табл. 2 при длине блока (N_b), равной 128 бит, в соответствии с формулами (1) и (2).

Таблица 2

Длина ключа/алгоритм	Известный	Оптимизированный
128 бит	391 такт	320 тактов
192 бит	461 такт	376 тактов
256 бит	531 такт	432 тактов

4. Экспериментальная оценка скорости преобразований

Полученные результаты свидетельствуют о наличии 18% повышения скорости работы алгоритма. Для проверки теоретических данных были проведены экспериментальные исследования. Сущность исследований заключается в осуществлении большого количества операций зашифрования-расшифрования (в нашем случае 100 000 000) при разной длине ключа и оценке времени, за которое эти операции были осуществлены. Исследования проводились в среде Visual C 6.0 с включенной оптимизацией по скорости. Результаты тестирования при количестве итераций 1000000 приведены в табл.3. В табл.4 приведены результаты оценки выигрыша в скорости оптимизированного алгоритма по сравнению с оригинальным. Эти результаты были получены на компьютере с процессором Celeron 600 Мгц, 128 Мб в операционной системе Windows 98 с минимально возможным числом запущенных процессов.

Таблица 3

Длина ключа	Время зашифр.(сек)		Время расшифр.(сек)	
	Известный алг.	Оптимизир. алг.	Известный алг.	Оптимизир. алг.
128	75,47	**66,63**	73,54	**64,37**
192	87,12	**77,77**	86,01	**75,85**
256	99,19	**89,26**	99,14	**86,89**
Длина ключа	Скорость зашифр.(Мбит/с)		Скорость расшифр.(Мбит/с)	
	Известный алг.	Оптимизир. алг.	Известный алг.	Оптимизир. алг.
128	169,6	**192,1**	174,05	**198,85**
192	146,92	**164,58**	148,81	**168,75**
256	129,04	**143,4**	129,11	**147,31**

Таблица 4

Длина ключа	Процент выигрыша (зашифр.)	Процент выигрыша (расшифр.)
128	**11,71%**	**12,47%**
192	**10,73%**	**11,81%**
256	**10,01%**	**12,36%**

Заключение

Таким образом, полученные теоретические и экспериментальные сложности (скорости) прямых и обратных криптографических преобразований в алгоритме Rijndael с использованием предложенных усовершенствований позволяют сделать вывод о его предпочтительности по сравнению с известным. Причем выигрыш достигается не менее, чем на 11%. Очевидно, что дальнейшее повышение скорости преобразований возможно за счет применения процессоро-зависимой реализации преобразований на ассемблере.

Список литературы: 1. *J. Daemen, V. Rijemen*, The Rijndael block cipher. AES Proposal. Http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf

Харьковский государственный технический университет радиоэлектроники

Поступила в редколлегию 16.03.2001