Симметричные криптосистемы: AES

Лекция

Ревизия: 0.1

Последняя модификация: Vlad Kovtun

Дата последней модификации: 31.01.2011 21:22:00

История изменений

15.10.2009 – Версия 0.1. Первичный документ. Ковтун В.Ю.

Содержание

Лекция 3. Симметричные блочные криптосистемы: AES. Часть 1

Вопросы

1. Симметричные криптосистемы.
2. Описание симметричной блочной криптосистемы AES.
3. Особенности реализации.

Симметричные криптосистемы

Гамма-шифра - псевдослучайная последовательность, выработанная по заданному алгоритму для зашифровывания открытых данных и расшифровывания зашифрованных.

Гаммирование - процесс наложения по определенному закону гаммы шифра на открытые данные.

Перед зашифровыванием, открытые данные разбивают на блоки T_o^i одинаковой длины (обычно по 64, 128, 196 либо 256 бита). Гамма шифра вырабатывается в виде последовательности блоков G_c^i аналогичной длины: $T_c^i = G_c^i \oplus T_o^i$, $i = \overline{1, m}$, где m - количество блоков открытого текста.

Процесс расшифровывания сводится к повторной генерации гаммы и наложении этой гаммы на зашифрованные данные: $T_o^i = G_c^i \oplus T_c^i$, $i = \overline{1, m}$.

Стойкость шифра определяется длиной ключа.

Для генерации псевдослучайных последовательностей применяются криптографически стойкие генераторы. К ним предъявляются следующие требования:

- период гаммы обязан быть достаточно большим;
- гамма обязана быть достаточно непредсказуемой;
- генерирование гаммы не должно вызывать больших технических сложностей.

По мнению Шеннона в практических шифрах необходимо использовать два общих принципа - рассеивания и перемешивания.

Рассеивание - распространение влияния одного знака открытого текста на много знаков шифротекста, что позволяет скрыть статистические свойства открытого текста.

Перемешивание - использование таких шифрующих преобразований, которые усложняют восстановление взаимосвязи статистических свойств открытого и шифрованного текстов.

Современные симметричные криптосистемы - это составные шифры, реализованные в виде некоторой последовательности простых шифров, каждый из которых вносит свой вклад в суммарное рассеивание и перемешивание. В качестве простейших шифров чаще всего используют шифры замены и перестановки. В современном блочном шифре блоки представляют собой двоичные последовательности длиной 64,128,192 или 256 бит. Получают достаточно стойкие шифры. Типичными примерами симметричных криптосистем являются стандарты DES, ГОСТ 28-147-89, AES (RIJNDAEL), Camellia.

Все блочные шифры могут функционировать в нескольких режимах:

- ECB (Electronic Code Book)- электронная кодовая книга;
- CBS (Cipherblock Chainsng) - сцепление блоков шифра;
- CFB (Cipher Feedback)- обратная связь по криптотексту;
- OFB (Output Feedback) - обратная связь по выходу.

ECB

В режиме ECB последовательность двоичных символов открытого текста разбивается на блоки и затем осуществляется их независимое шифрование при помощи одного и того же ключа (длина ключа соответствует длине блока открытого текста).

Режим ECB обладает тем недостатком, что результаты шифрования одинаковых блоков, при помощи одного и того же ключа совпадают, С учетом того, что длина блока криптотекста (криптограммы) постоянна и относительно невелика, наличие в криптотексте идентичных блоков служит явным указанием на присутствие совпадающих блоков и в соответствующих фрагментах открытого текста, что в общем случае может существенно облегчить криптоанализ.

С другой стороны, наблюдение за информационным обменом между двумя абонентами применительно к ситуации, когда отправитель вторично посылает открытый текст, зашифрованный одним и тем же ключом, позволяет имитировать возврат той же криптограммы-ответа, который был передан получателем в первом случае.

Наконец, большинство алгоритмов в режиме ECB обладает свойством, так называемого размножения ошибок, которое состоит в том, что искажения одного бита криптограммы приводит к искажению нескольких (около половины) бит в открытом тексте, полученном в результате дешифрования.

CBC

Для нейтрализации этих недостатков разработан специальный режим использования, называемый CBC (режим сцепления блоков шифра).

В этом режиме сообщение M разбивается на блоки $M_1,\ldots,M_n$. К каждому блоку M_i открытого текста на текущем шаге шифрования, перед подачей его на вход алгоритма шифрования E, с помощью побитного суммирования по модулю 2 добавляется зашифрованный блок $„_{i-1}$, полученный на предыдущем шаге, так что $„_i = E_K\left(M_i \oplus C_{i-1}\right)$.

Если два разных сообщения будут начинаться одинаковыми блоками, то совпадающие первые блоки будут иметь и соответствующие криптотексты. Чтобы защититься и от этого недостатка, перед шифрованием каждому сообщению M приписывается начальное случайное число (длина числа соответствует длине блока шифрования) – так называемый *вектор инициализации* C_0.

Дешифрование выполняется по алгоритму $M_i \oplus C_{i-1} = E_K^{-1}\left(C_i\right)$ или, что то же самое $M_i = C_{i-1} \oplus E_K^{-1}\left(C_i\right)$.

Как следует из приведенных соотношений, каждый очередной блок криптограммы, является функцией предшествующих. Поэтому искажение одного бита в криптотексте искажает два смежных блока, полученных в результате дешифрования. Однако, поскольку искаженный блок криптограммы гаммируется со следующим блоком открытого текста, число искажений в следующем дешифруемом блоке равно числу искажений в криптограмме. Что касается криптостойкости, то режим CBC считается более стойким, нежели ECB по двум причинам. Во-первых, криптограмме является функцией не только открытого текста и ключа, но и в конечном итоге начального вектора C_0. Это, конечно, усложняет криптоанализ. С другой стороны, идентичным блокам открытого текста в общем случае соответствуют различные блоки криптограммы, что нейтрализует основной недостаток ECB.

CFB

В отличии от режимов ECB и CBC, оперирующих с блоками (идентичной длины, что и длина блока шифрования), режимы CFB и OFB оперируют с k-битными блоками (длина может варьироваться от 1 до длины блока шифрования без единицы). Это, в частности, позволяет шифровать текстовые данные посимвольно (для кода EBCDIC достаточно взять $k = 8$, а для кода ASCII – $k = 7$).

В CFB режиме базовый алгоритм E служит для порождения блоков псевдослучайных битов B_i длиной равной длине блока шифрования: $B_i = E_K\left(I_i\right)$, $i = \overline{0, l}$, где I_0 – произвольный инициирующий вектор, а I_{i+1} получается из I_i отбрасыванием (методом сдвига) первых k битов и приписыванием (заполнением) справа к блоку первых (левых) k бит блока B_{i-1}. Блок зашифрованного сообщения передается в виде $C_i^{(k)} = m_i^{(k)} \oplus B_i^{(k)}$. Здесь $C_i^{(k)} - k$ битов текста на i-ом шаге шифрования; $m_i^{(k)} -$

соответствующие k битов открытого текста; $B_i^{(k)}$ – левые k бит результата шифрования на *i* - том шаге; $\oplus$ - символ побитного суммирования по модулю 2.

OFB

Режим OFB похож на предыдущий, только для обновления I_i используется C_{i-1} вместо B_{i-1}.

Как видно, оба рассмотренных режима использования алгоритма блочного шифровани с обратной связью, как и режим CBC, свободны от главного недостатка ECB: идентичным блокам открытого текста в общем случае соответствуют различные блоки криптотекста (что касается OFB, то данное утверждение верно по отношению к любым двум идентичным блокам, разделенных в открытом тексте блоками, количество которых не кратно периоду повторения, уменьшенному на единицу).

Существенным достоинством OFB является отсутствие явления размножения искажений, имевших место при передаче криптотекста в процессе дешифрования последнего. В то же время искажения одного бита k-битного блока криптотекста, полученного в режиме CFB, будет влиять на k-битные блоки открытого текста до тех пор, пока искаженный бит не будет вытолкнут за пределы входного блока в результате последовательности сдвигов его содержимого, выполняемых в процессе дешифрования.

Таковы основные особенности алгоритма блочного шифрования и различных режимов его использования. Кратко остановимся на вопросах применения указанных режимов для обеспечения безопасности информации, передаваемой и хранимой в информационных системах. Обычно рассматриваются три области применения алгоритма блочного шифрования:

- передача данных по каналам связи (ИВС);
- хранение;
- доступ к файлам (базам данных) ИВС;
- обмен коммерческой информацией.

Считается, что для обеспечения безопасности каналов связи могут применяться все описанные выше режимы, хотя в большинстве аппаратных реализаций (плат криптографического шифрования) используется режим CFB с $k=1$, обеспечивающий побитовое шифрование, Это объясняется его высокой криптостойкостью, по сравнению с OFB, особенно в плане криптографической поддержки бит-ориентированных протоколов, характерных для информационных систем, функционирующих на основе высокоскоростных широкополосных каналов связи.

В приложения, связанных с хранением информации во внешней памяти ЭВМ, различают три основных направления применения: криптографическая защита файлов прямого и последовательного доступа, а также отдельных полей записи таких файлов.

Описание симметричной блочной криптосистемы AES

Определения и вспомогательные процедуры

Таблица 1. Определения переменных, в алгоритме AES

Block	последовательность бит, из которых состоит input, output, State и Round Key. Также под Block можно понимать последовательность байт
Cipher Key	секретный, криптографический ключ, который используется Key Expansion процедурой, чтобы произвести набор ключей для раундов(Round Keys); может быть представлен как прямоугольный массив байтов, имеющий четыре строки и Nk колонок.
Ciphertext	выходные данные алгоритма шифрования
Key Expansion	Процедура, используемая для генерации Round Keys из Cipher Key

Round Key	Round Keys получаются из Cipher Key используя процедуру Key Expansion. Они применяются к State при шифровании и расшифровании
State	промежуточный результат шифрования, который может быть представлен как прямоугольный массив байтов, имеющий 4 строки и Nb колонок
S-box	нелинейная таблица замен, использующаяся в нескольких трансформациях замены байт и в процедуре Key Expansion для взаимно-однозначной замены значения байта
Nb	число столбцов (32-ух битных слов), составляющих *State*. Для, AES $Nb = 4$
Nk	число 32-ух битных слов, составляющих шифроключ. Для AES, $Nk = 4$, 6 или 8
Nr	число раундов, которое является функцией Nk и Nb. Для AES, $Nr = 10$, 12, 14
Rcon[]	массив, который состоит из битов 32-х разрядного слова и является постоянным для данного раунда

Таблица 2. Определения функций, в алгоритме AES

AddRoundKey()	трансформация при шифровании и обратном шифровании, при которой Round Key XOR'ится с State. Длина RoundKey равна размеру State (те, если $Nb = 4$, то длина RoundKey=128 бит или 16 байт)
InvMixColumns()	трансформация при расшифровании которая является обратной по отношению к MixColumns()
InvShiftRows()	трансформация при расшифровании которая является обратной по отношению к ShiftRows()
InvSubBytes()	трансформация при расшифровании которая является обратной по отношению к SubBytes()
MixColumns()	трансформация при шифровании которая берет все столбцы State и смешивает их данные (независимо друг от друга), чтобы получить новые столбцы
RotWord()	функция, использующаяся в процедуре Key Expansion, которая берет 4-х байтное слово и производит над ним циклическую перестановку
ShiftRows()	трансформации при шифровании, которые обрабатывают State, циклически смещая последние три строки State на разные величины
SubBytes()	трансформации при шифровании которые обрабатывают State используя нелинейную таблицу замещения байтов (S-box), применяя её независимо к каждому байту State
SubWord()	функция, используемая в процедуре Key Expansion, которая берет на входе четырёх-байтное слово, и применяя S-box к каждому из четырёх байтов выдаёт выходное слово

Шифрование

AES является стандартом, основанным на алгоритме Rijndael. Для AES длина `input` (блока входных данных) и State(состояния) постоянна и равна 128 бит, а длина шифроключа $\mathbf{K}$ составляет 128, 192, или 256 бит. При этом, исходный алгоритм Rijndael допускает длину ключа и размер блока от 128 до 256 бит с шагом в 32 бита. Для обозначения выбранных длин `input`, `State` и `Cipher Key` в байтах используется

нотация $Nb = 4$ для input и State, $Nk = 4, 6, 8$ для Cipher Key соответственно для разных длин ключей.

В начале шифрования input копируется в массив State по правилу $s[r, c] = in[r + 4c]$, для $0 \le r < 4$ и $0 \le c < Nb$. После этого к State применяется процедура AddRoundKey() и затем State проходит через процедуру трансформации (раунд) 10, 12, или 14 раз (в зависимости от длины ключа), при этом надо учесть, что последний раунд несколько отличается от предыдущих. В итоге, после завершения последнего раунда трансформации, State копируется в output по правилу $out[r + 4c] = s[r, c]$, для $0 \le r < 4$ и $0 \le c < Nb$.

Шифр описан в псевдокоде на рис.1. Отдельные трансформации SubBytes(), ShiftRows(), MixColumns(), и AddRoundKey() — обрабатывают State. Массив w[] — содержит key schedule.

Алгоритм процедуры шифрования:

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[0, Nb-1])
    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    out = state
end
```

SubBytes()

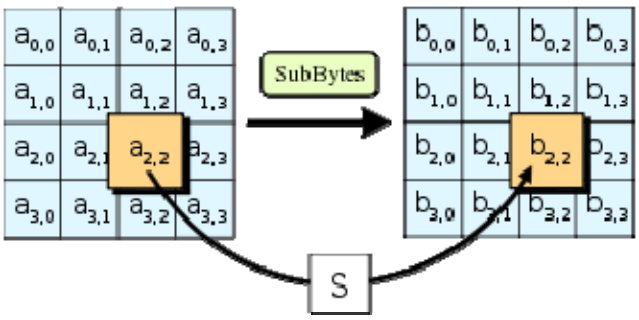

Рис. 1. процедуре SubBytes, каждый байт в state заменяется соответствующим элементом в фиксированной 8-битной таблице поиска, $\mathbf{S}$; $b_{ij} = \mathbf{S}(a_{ij})$.

Процедура SubBytes() обрабатывает каждый байт состояния, независимо производя нелинейную замену байтов используя таблицу замен (S-box). Такая операция обеспечивает нелинейность алгоритма шифрования. Построение S-box состоит из двух шагов. Во-первых, производится взятие обратного числа в $\mathbf{GF}(2^8)$. Во-вторых, к каждому байту b из которых состоит S-box применяется следующая операция: $b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus b_i$, где $0 \le i < 8$, и где b_i есть i-ый бит

b, а c_i — i-ый байт $c = \{63\}$ или $\{01100011\}$. Таким образом, обеспечивается защита от атак, основанных на простых алгебраических свойствах.

$$\begin{Vmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{Vmatrix} = \begin{Vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{Vmatrix} * \begin{Vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{Vmatrix} + \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{Vmatrix}$$

Рис. 2. Матричное представление процедуры `SubBytes()`

ShiftRows()

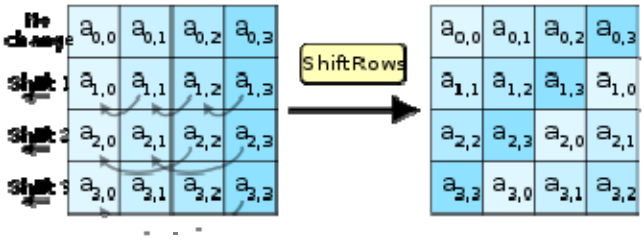

Рис. 3. В процедуре `ShiftRows`, байты в каждой строке `state` циклически сдвигаются влево. Размер смещения байтов каждой строки зависит от её номера

`ShiftRows` работает со строками `State`. При этой трансформации строки состояния циклически сдвигаются на r байт по горизонтали, в зависимости от номера строки. Для нулевой строки $r = 0$, для первой строки $r = 16$ и т.д. Таким образом, каждая колонка выходного состояния после применения процедуры `ShiftRows` состоит из байтов из каждой колонки начального состояния. Для алгоритма Rijndael паттерн смещения строк для 128 и 192-ух битных строк одинаков. Однако для блока размером 256 бит отличается от предыдущих тем, что 2, 3, и 4-е строки смещаются на 1, 3, и 4 байта соответственно.

MixColumns()

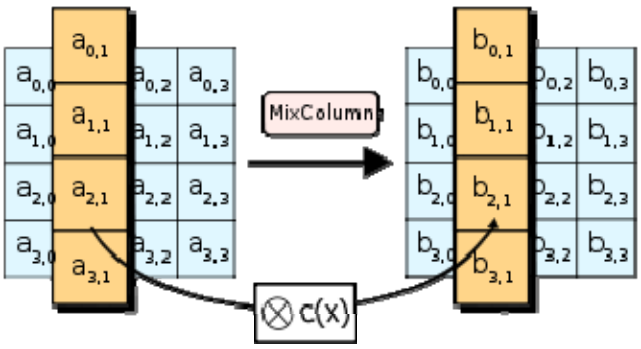

Рис. 4. В процедуре `MixColumns`, каждай колонка состояния перемножается с фиксированным многочленом $c(x)$

В процедуре `MixColumns`, четыре байта каждой колонки State смешиваются используя для этого обратимую линейную трансформацию. `MixColumns` обрабатывает состояния по колонкам, трактуя каждую из них как полином четвертой степени. Над этими

полиномами производится умножение в $\mathbf{GF}(2^8)$ по модулю x^4+1 на фиксированный многочлен $c(x)=3x^3+x^2+x+2$. Вместе с ShiftRows, MixColumns вносит диффузию в шифр.

AddRoundKey()

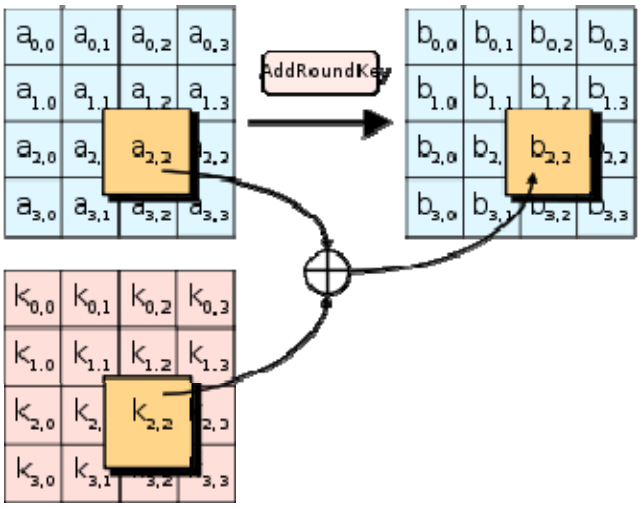

Рис. 5. В процедуре AddRoundKey, каждый байт состояния объединяется с RoundKey используя XOR operation ($\oplus$).

В процедуре AddRoundKey, RoundKey каждого раунда объединяется со State. Для каждого раунда Roundkey получается из CipherKey используя процедуру KeyExpansion; каждый RoundKey такого же размера, что и State. Процедура производит побитовый XOR каждого байта State с каждым байтом RoundKey .

KeyExpansion()

AES алгоритм, используя процедуру KeyExpansion() и подавая в неё Cipher Key, K, получает ключи для всех раундов. Всего она получает $Nb(Nr+1)$ слов: изначально для алгоритма требуется набор из Nb слов, и каждому из Nr раундов требуется Nb ключевых набора данных. Полученный массив ключей для раундов обозначается как w[i], $0 \le i < Nb(Nr+1)$. Алгоритм KeyExpansion() показан в псевдокоде на рис.6

Функция SubWord() берет четырёхбайтовое входное слово и применяет S-box к каждому из четырёх байтов то, что получилось, подается на выход. На вход RotWord() подается слово $[a_0, a_1, a_2, a_3]$, которое она циклически переставляет и возвращает $[a_1, a_2, a_3, a_0]$. Массив слов, слов постоянный для данного раунда, Rcon[i], содержит значения $[x^{i-1}, 00, 00, 00]$, где $x=\{02\}$, а x^{i-1} является степенью x в $\mathbf{GF}(2^2)$ (i начинается с 1).

Из рисунка можно увидеть, что первые Nk слов расширенного ключа заполнены Cipher Key. В каждое последующее слово, $w[i]$, кладется значение полученное при операции XOR $w[i-1]$ и $w[i-Nk]$, те XOR'а предыдущего и на Nk позиций раньше слов. Для слов, позиция которых кратна Nk, перед XOR'ом к $w[i-1]$ применяется трансформация, за которой следует XOR с константой раунда Rcon[i]. Указанная выше трансформация состоит из циклического сдвига байтов в слове RotWord(), за которой следует процедура SubWord() — то же самое, что и SubBytes(), только входные и входные данные будут размером в слово.

Важно заметить, что процедура KeyExpansion() для 256 битного Cipher Key немного отличается от тех, которые применяются для 128 и 192 битных шифроключей. Если Nk = 8 и $i-4$ кратно Nk, то SubWord() применяется к $w[i-1]$ до XOR'а.

Алгоритм процедуры расширения ключа:

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
```

```
begin
    word temp
    i = 0;
    while ( i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while
    i = Nk
    while ( i < Nb * (Nr+1))
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
```

Расшифрование

Алгоритм процедуры расшифровывания:

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    for round = Nr-1 step -1 downto 1
        InvShiftRows(state)
        InvSubBytes(state)
        InvAddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        InvMixColumns(state)
    end for
    InvShiftRows(state)
    InvSubBytes(state)
    InvAddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    out = state
end
```

Варианты алгоритма

На базе алгоритма Rijndael, лежащего в основе AES, реализованы альтернативные криптоалгоритмы. Среди наиболее известных — участники конкурса Nessie: Anubis на нволюциях, автором которого является Винсент Рэймен и усиленный вариант шифра — Grand Cru Йохана Борста.

Особенности реализации

Введение

Стандарт шифрования AES является официальным стандартом правительства США для симметричного шифрования. Стандарт определяется публикацией FIPS #197 (2001) и используется в разнообразных приложениях, где предъявляются повышенные требования к производительности и безопасности.

Компания Intel представила новый набор команд вида "одна инструкция на несколько данных" (SIMD, Single Instruction Multiple Data), который будет реализован в следующем поколении процессоров этой фирмы. Эти инструкции обеспечивают быстрое и безопасное шифрование и дешифрование с помощью алгоритма AES.

Этот набор команд, называемый SSE (*Streaming SIMD Extensions*, потоковое SIMD-расширение) включает в себя шесть инструкций. Четыре из них, AESENC, AESENCLAST, AESDEC, и AESDELAST обеспечивают высокопроизводительное шифрование и дешифрование. Еще две, AESIMC и AESKEYGENASSIST, позволяют производить расширение ключа AES. Вместе они обеспечивают полную аппаратную поддержку стандарта AES с необходимым уровнем безопасности, производительности и гибкости.

Данная статья приводит обзор алгоритма AES и методы использования его инструкций для достижения высокой производительности и безопасности шифрования. Приводятся также некоторые примеры специального использования этого алгоритма.

Архитектура AES и Intel®

Стандарт шифрования AES является официальным стандартом правительства США для симметричного шифрования и описан в публикации FIPS №197 (FIPS197 hereafter).

AES представляет собой блочный шифр, кодирующий 128-битный текстовый блок в 128-битный шифрованный блок, или дешифрует 128-битный шифрованный блок в 128-битный текстовый блок.

AES представляет собой блочный шифр, кодирующий 128-битный текстовый блок в 128-битный шифрованный блок, или дешифрует 128-битный шифрованный блок в 128-битный текстовый блок.

AES-128, AES-192, AES-256 обрабатывают блоки данных за соответственно 10, 12 или 14 итераций. Каждая итерация представляет собой определенную последовательность трансформаций. Все итерации одинаковы за исключением последней, из которой исключено одно из преобразований. В дальнейшем будем именовать итерацию раундом.

Каждый раунд работает с двумя 128-битными блоками: "Текущий" и "ключ раунда". Все раунды используют разные "ключи раунда", которые получаются с пощью алгоритма расширения ключа. Этот алгоритм не зависит от шифруемых данных и может выполняться независимо от фазы шифрования/дешифрования.

Блок данных последовательно проходит через следующие стадии: над ним выполняется операция XOR первыми 128 битами ключа, на выходе получается "текущий" блок (эта стадия также называется нулевым раундом, с использованием нулевого ключа раунда – первых 128 битов ключа шифра). Затем текущий блок проходит через 10/12/14 раундов шифрования, после которых он превращается в шифрованный (или дешифрованный) блок.

Расположение байтов: Little Endian архитектура Intel и Big Endian спецификация FIPS197

В спецификации FIPS197 указано использование расположения байтов в словах от старшего к младшему (Big Endian), тогда как архитектура процессоров Intel подразумевает использование расположения байтов от младшего к старшему (Little Endian), так что при работе алгоритма шифрования необходимо применять соответсвующую процедуру изменения порядка байтов. Например, преобразование тестового вектора FIPS197 в стандартную нотацию Intel требует обращения порядка байтов. Если записать 128-битный вектор FIPS197 как [Byte0, byte1, …, Byte14, Byte15], в каждом байте последовательность битов идет как "самый левый – самый старший", т.е. запись в битах [7-0, 15-8, 23-16, 31-24, …, 127-120].

Чтобы преобразовать такой вектор в стандарт Intel, его нужно отобразить как [Byte15, byte14, …, Byte1, Byte0], или в битах [127-120, …, 31-24, 23-16, 15-8, 7-0].

В дальнейшем мы будем применять шестнадцатеричную систему записи, где каждый байт записывается парой шестнадцатеричных цифр.

Пример

Рассмотрим вектор d4bf5d30e0b452aeb84111f11e2798e5, записанный в нотации FIPS197. В байтовом отображении мы получим 16 2-х разрядных шестнадцатеричных чисел "d4 bf 5d 30 e0 b4 52 ae b8 41 11 f1 1e 27 98 e5". Здесь d4 – самый младший байт.

Запишем этот вектор в нотации Intel: e590271ef11141b8ae52b4e0305dbfd4, или побайтово "e5 90 27 1e f1 11 41 b8 ae 52 b4 e0 30 5d bf d4". Двоичное 128-битное отображение таково:

11100101100110000010011100011110111100010001000101000000110111000

10101110010100101011010011100000001100000101110110111111111010100

(самый старший бит содержит 1, самый младший – 0)

Сторонние каналы ПО

В этой главе приводится краткое описание сторонних каналов информации, которые могут быть использованы для атаки на шифр, и объясняется, почему структура доступа к памяти может быть использованы для атаки на коды программ AES, использующих таблицы подстановки.

Что такое атака по сторонним признакам?

Сторонние каналы программного обеспечения – это некий набор уязвимостей, возникающих при работе современных компьютеров. Уязвимости могут быть использованы для атаки на криптографические приложения, работающие в многозадачном окружении.

Эти атаки используют то обстоятельство, что практически все современные коммерческие компьютеры запускают несколько задач на одном наборе оборудования. Некоторые современные научные публикации наглядно показывают, что многозадачные операционные системы, работающие на одном процессоре, могут иметь утечки сторонней информации, если в системе параллельно с работающей криптосистемой запущен непривилегированный сторонний процесс-шпион, который может собирать информацию о структуре доступа к памяти или особенностях выполнения процессов.

Кэш процессора и основы атаки на кэш

Применение кэш-памяти – широко используемая технология оптимизации работы современных процессоров. Для кэша используется быстрая (и дорогая) память, которая обеспечивает скоростной доступ к данным, гораздо быстрее, чем обычная память. Процессор использует кэш-память для хранения данных из областей памяти, к которым было обращение. Таким образом, при каждом обращении к памяти процессор сначала проверяет наличие требуемых данных в кэше, и если они там оказываются (cache hit), доступ к ним оказывается очень быстрым.. Если требумых данных там не оказывается (cache miss), то они прочитываются (довольно медленно) из основной памяти и сохраняются в кэше для возможного использования в будущем. Естественно, чтобы сохранить новые данные, процессор должен очистить место и выгрузить из кэша некие предыдущие данные. Как результат использования кэша, среднее время доступа к некоторым областям памяти существенно сокращается, и зловредный код может это определить и использовать, в случае если криптографическое приложение имеет уязвимую структуру доступа к памяти.

Таблицы подстановки и возможные уязвимости

В настоящее время наиболее производительные реализации алгоритма AES используют большие таблицы подстановки (например, Gladmans или OpenSSL). Обычно программы используют пять таблиц подстановки. Доступ к этим таблицам зависит от ключа и шифруемых данных, открывается сторонний канал информации.

Для проведения атаки необходим сторонний шпионский процесс, запущенный в системе параллельно с программой шифрования. Этот процесс заполняет кэш своими данными и затем обращается к ним. Измеряя время доступа, шпион определяет, какие области памяти были исключены из кэша, и по этой информации можно определить, к каким областям обращался шифрующий процесс. Анализ этих областей может привести к раскрытию секретного ключа (на самом деле, первый и последний раунды дают самую большую утечку информации о ключе). Для получения детальной информации об атаках по сторонним каналам можно обратиться к следующим (среди многих) источникам: D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES", Lecture Notes in Computer Science series, Springer-Verlag, 3860: 1-20, (2006), а также D. J. Bernstein, "Cache-timing attacks on AES" (2005).

Ослабление возможностей атаки на AES уменьшает производительность алгоритма

Есть способы написать реализующую AES программу таким способом, который исключит существенную зависимость структуры доступа к памяти от ключа и шифруемых данных. Один из таких способов – периодически менять таблицы подстановки местами. Однако такие защитные меры не даются даром, в случае алгоритма AES они существенно снижают его производительность. За подробностями можно обратиться к работе E. Brickell, E., G. Graunke, M. Neve, J. P. Seifert, "Software mitigations to hedge AES against cache based software side channel vulnerabilities". Другой способ -- вообще не пользоваться таблицами подстановки, но опять же существенно страдает производительность. Подробности в работе M. Matsui and S. Fukuda. "How to Maximize Software Performance of Symmetric Primitives on Pentium III and 4 Processors". LNCS, Springer Verlag, 3557: 398–412 (2005).

Новые AES инструкции процессоров Intel увеличивают защищенность алгоритмов AES от атак с помощью сторонних утечек информации

Новый AES набор инструкций процессоров Intel позволяет производить быстрое шифрование/дешифрование существенно снижая при этом риск атаки с помощью сторонних каналов информации. Все операции имеют фиксированное и независимое от данных время исполнения. Более того, все вычисления выполняются исключительно аппаратно. Таким образом, операции шифрования и дешифрования, а также расширения ключа имеют фиксированное время исполнения и не требуют доступа к памяти. Как результат, исчезает опасность утечки информации с помощью наблюдения за доступом процессов к памяти.

AES архитектура Intel

Набор инструкций AES состоит из шести команд.

Четыре инструкции AESENC (AES Encrypt Round), AESENCLAST (AES Encrypt Last Round), AESDEC (AES Decrypt Round), AESDECLAST (AES Decrypt Last Round) реализуют шифрование и дешифрование данных. Инструкции имеют формат как регистр-регистр, так и регистр-память.

Две другие инструкции, AESIMC (AES Inverse Mix Columns) и AESKEYGENASSIST (AES Key Generation Assist), работают с расширением ключа.

Четыре инструкции раундов

Инструкции AESENC, AESENCLAST, AESDEC, AESDECLAST можно описать приведенным ниже псевдокодом. Эти инструкции выполняют набор групповых преобразований, который соответствует потокам шифрования и дешифрования AES.

```
AESENC xmm1, xmm2/m128
Tmp := xmm1;
Round Key := xmm2/m128;
Tmp := ShiftRows (Tmp);
Tmp := SubBytes (Tmp);
Tmp := MixColumns (Tmp);
```

```
AESENCLAST xmm1, xmm2/m128
Tmp := xmm1;
Round Key := xmm2/m128;
Tmp := Shift Rows (Tmp);
Tmp := SubBytes (Tmp);
xmm1 := Tmp xor Round Key
```

xmm1 := Tmp xor Round Key

Рис. 6. Инструкции AESENC и AESENCLAST

AESDEC xmm1, xmm2/m128
Tmp := xmm1;
Round Key := xmm2/m128;
Tmp := InvShift Rows (Tmp);
Tmp := InvSubBytes (Tmp);
Tmp := InvMix Columns (Tmp);
xmm1 := Tmp xor Round Key

AESDECLAST xmm1, xmm2/m128
State := xmm1;
Round Key := xmm2/m128
Tmp := InvShift Rows (State);
Tmp := InvSubBytes (Tmp);
xmm1 := Tmp xor Round Key

Рис. 7. Инструкции AESDEC и AESDECLAST

Пример использования AESENC

Пример AESENCLAST

Пример AESEDEC

Пример AESDECLAST

Потоки шифрования в терминах инструкций раундов AES

Потоки шифрования и дешифрования стандарта AES (см. примеры выше) реализуются с помощью четырех инструкций раундов AES. Будем использовать цветовой код для различения разных потоков с разными длинами ключа (128, 192, 256 бит). В приведенном псевдокоде будем считать, что 10, 12 или 14 ключей раундов уже получены (расширены) из ключа шифра и сохранены в должном порядке в массив данных (Round_Key_Encrypt или Round_Key_Decrypt). Также примем, что элемент Round_Key_Encrypt[0] содержит первые 128 бит основного ключа (которые используются в нулевом раунде, для побитовой операцией исключающего ИЛИ).

Реализация AES шифрования с помощью инструкций AESENC/AESENCLAST

1. Tmp = AddRoundKey (Data, Round_Key_Encrypt [0])
2. For round = 1-9 or 1-11 or 1-13:
3. Tmp = AESENC (Tmp, Round_Key_Encrypt [round])
4. end loop
5. Tmp = AESENCLAST (Tmp, Round_Key_Encrypt [10 or 12 or 14])
6. Result = Tmp

Реализация AES дешифрования (эквивалентное инверсное шифрование) с помощью инструкций AESDEC/AESDECLAST

1. Tmp = AddRoundKey (Data, Round_Key_Decrypt [0])
2. For round = 1-9 or 1-11 or 1-13:
3. Tmp = AESDEC (Tmp, Round_Key_Decrypt [round])
4. end loop
5. Tmp = AESDECLAST (Tmp, Round_Key_Decrypt [10 or 12 or 14])
6. Result = Tmp

Пример реализации шифрования AES-128

Приведенный ниже код осуществляет последовательность алгоритма шифрования AES-128.

1. ;; Последовательность шифрования
2. ;; данные в xmm1. регистры xmm2 – xmm12 содежат ключи раундов.

3. ;; По завершении xmm1 содержит результат шифрования
4. pxor xmm1, xmm2 ; Round 0 (Round 0)
5. aesenc xmm1, xmm3 ; Round 1
6. aesenc xmm1, xmm4 ; Round 2
7. aesenc xmm1, xmm5 ; Round 3
8. aesenc xmm1, xmm6 ; Round 4
9. aesenc xmm1, xmm7 ; Round 5
10. aesenc xmm1, xmm8 ; Round 6
11. aesenc xmm1, xmm9 ; Round 7
12. aesenc xmm1, xmm10 ; Round 8
13. aesenc xmm1, xmm11 ; Round 9
14. aesenclast xmm1, xmm12 ; Round 10

Пример дешифрования AES-192

Использование инструкции AESIMC в AES-192 для дешифрования.

1. ;; Последовательность дешифрования
2. ;; данные в xmm1. Регистры xmm14 – xmm2 содержат ключи раундов.
3. ;; По завершении - xmm1 содерит дешифрованный результат
4. pxor xmm1, xmm14 ; Round 0 (Round 0)
5. aesdec xmm1, xmm13 ; Round 1 (ключи употребляются в обратном по рядке)
6. aesdec xmm1, xmm12 ; Round 2
7. aesdec xmm1, xmm11 ; Round 3
8. aesdec xmm1, xmm10 ; Round 4
9. aesdec xmm1, xmm9 ; Round 5
10. aesdec xmm1, xmm8 ; Round 6
11. aesdec xmm1, xmm7 ; Round 7
12. aesdec xmm1, xmm6 ; Round 8
13. aesdec xmm1, xmm5 ; Round 9
14. aesdec xmm1, xmm4 ; Round 10
15. aesdec xmm1, xmm3 ; Round 11
16. aesdeclast xmm1, xmm2 ; Round 12

Расширение ключа AES

Генерация ключей происходит с помощью двух инструкций: AESKEYGENASSIST генерирует ключи раундов для шифрования, AESIMC конвертирует полученные ключи в форму, пригодную для дешифрования.

Использование инструкции AESKEYGENASSIST для расширения ключа.

1. AESKEYGENASSIST xmm1, xmm2/m128, imm8
2. Tmp := xmm2/LOAD(m128)
3. X3[31-0] := Tmp[127-96];
4. X2[31-0] := Tmp[95-64];
5. X1[31-0] := Tmp[63-32];
6. X0[31-0] := Tmp[31-0];
7. RCON[7-0] := imm8;

8. xmm1 := [Rot (SubWord (X3)) ⊕ RCON, SubWord (X3), Rot (SubWord (X1)) ⊕ RCO N, SubWord (X1)]

Примечание: "Doubleword ⊕ RCON" в реализации расширения ключа, означает операцию [Byte3, Byte2, Byte1, Byte0] XOR [0, 0, 0, RCON], где Byte3, Byte2, Byte1, Byte0 -- байты соответствующего двойного слова.

Пример AESKEYGENASSIST

Генерация ключей таблицы ключей с помощью инструкции AESKEYGENASSIST.

1. ;; xmm2 содержит 128-битный входной блок; imm8 содержит значение RCON
2. ;; результат сохраняется в xmm1
3. xmm2 = 3c4fcf098815f7aba6d2ae2816157e2b
4. imm8 = 1
5. результат AESKEYGENASSIST (в xmm1): 01eb848beb848a013424b5e524b5e434

Генерация таблицы ключей с помощью AESKEYGENASSIST

Следующий код демонстрирует пример подготовки таблицы ключей AES-128 и последующей дешифровки. В следующих главах рассмотрены примеры с ключами большей длины.

Расширение ключей AES-128

1. $data->data(<<'DATA');
2. key do 03c4fcf098815f7aba6d2ae2816157e2bh ; 128-битный ключ (FIPS doc)
3. keyex_addr: keyex
4. do 00000000000000000000000000000000h;
5. do 00000000000000000000000000000000h;
6. do 00000000000000000000000000000000h;
7. do 00000000000000000000000000000000h;
8. do 00000000000000000000000000000000h;
9. do 00000000000000000000000000000000h;
10. do 00000000000000000000000000000000h;
11. do 00000000000000000000000000000000h;
12. do 00000000000000000000000000000000h;
13. do 00000000000000000000000000000000h;
14. do 00000000000000000000000000000000h;
15. do 00000000000000000000000000000000h;
16. ;; Подготовка таблицы ключей
17. movaps xmm1, OWORD PTR key ; загрузка ключа
18. movaps OWORD PTR keyex, xmm1 ; Сохранение ключа в памяти с остальными расширенными ключами
19. mov rcx, OFFSET keyex_addr+16 ; установка адреса сохранения расширенного ключа
20. aeskeygenassist xmm2, xmm1, 0x1 ; Генерация ключа 1 раунда
21. call key_expansion_128
22. aeskeygenassist xmm2, xmm1, 0x2 ; Генерация ключа 2 раунда
23. call key_expansion_128
24. aeskeygenassist xmm2, xmm1, 0x4 ; Генерация ключа 3 раунда
25. call key_expansion_128
26. aeskeygenassist xmm2, xmm1, 0x8 ; Генерация ключа 4 раунда

```
call key_expansion_128
aeskeygenassist xmm2, xmm1, 0x10    ; Генерация ключа 5 раунда
call key_expansion_128
aeskeygenassist xmm2, xmm1, 0x20    ; Генерация ключа 6 раунда
call key_expansion_128
aeskeygenassist xmm2, xmm1, 0x40    ; Генерация ключа 7 раунда
call key_expansion_128
aeskeygenassist xmm2, xmm1, 0x80    ; Генерация ключа 8 раунда
call key_expansion_128
aeskeygenassist xmm2, xmm1, 0x1b    ; Генерация ключа 9 раунда
call key_expansion_128
aeskeygenassist xmm2, xmm1, 0x36    ; Генерация ключа 10 раунда
call key_expansion_128
key_expansion_128:
mov rdx, rcx
pshufd xmm2, xmm2, 0b11111111
pxor xmm2, xmm1
movd eax, xmm2
mov DWORD PTR [rcx], eax
add rcx, 4
pshufd xmm1, xmm1, 011100101b
movd ebx, xmm1
xor eax, ebx
mov DWORD PTR [rcx], eax
add rcx, 4
pshufd xmm1, xmm1, 011100110b
movd ebx, xmm1
xor eax, ebx
mov DWORD PTR [rcx], eax
add rcx, 4
pshufd xmm1, xmm1, 011100111b
movd ebx, xmm1
xor eax, ebx
mov DWORD PTR [rcx], eax
add rcx, 4
movaps xmm1, DWORD PTR [rdx]
ret
```

Приготовление ключей раундов для дешифрования с использованием AESIMC

Команда AESIMC используется для создания ключей раундов для дешифровки с помощью метода "эквивалентного инверсного шифрования". Приведенный псевдокод иллюстрирует работу этой инструкции.

Подготовка раундовых ключей с помощью инструкции AESIMC.

1. AESIMC xmm1, xmm2/m128
2. RoundKey := xmm2/m128;

3. xmm1 := InvMixColumns (RoundKey)

Команды AESDEC и AESDECLAST выполняют дешифрование данных с помощью метода "эквивалентного инверсного шифрования". Для получения в процессе корректных ключей раундов, должна быть правильным образом подготовлена таблица ключей дешифровки. Если считать, что таблица ключей шифровки уже существует в памяти в виде массива, таблица ключей для дешифровки легко создается с помощью инструкции AESIMC. Для этого каждый из 10/12/14 ключей раундов шифровки прогоняется через команду AESIMC и на выходе получается соответствующий ключ раунда дешифровки. Алгоритм дешифрования использует ключи в обратном порядке, так что ключи дешифровки должны сохраняться в массиве в обратном порядке (по отношению к массиву ключей шифрования).

Например, возьмем метод AES-128 и будем считать, что регистр xmm2 содержит ключ раунда 3. Тогда после выполнения команды AESIMC xmm1, xmm2, регистр xmm1 будет содержать ключ раунда 8 дешифрования.

Формирование таблицы ключей для дешифрования производится с помощью инструкции AESIMC, на основе ранее подготовленного массива, который содержит таблицу в виде массива.

1. ;; xmm2 содержит один 128-блок входных данных (xmm2 = ключ раунда)
2. ;; результат сохраняется в xmm1
3. xmm2 = 48692853686179295b477565726f6e5d
4. результат AESIMC (в xmm1): 627a6f6644b109c82b18330a81c3b3e5

Генерация ключей раундов дешифрования AES-192

Следующий код показывает вариант подготовки таблицы ключей AES-192 и последующего дешифрования.

Использование инструкции AESIMC в AES-192 для генерации раундовых ключей для дешифрования.

1. kjd
2. ; Ключи раундов шифрования в регистрах xmm2 -- xmm14
3. ; Ключи раундов обрабатываются командой aesimc
4. ; Дешифрование использует преобразованные ключи раундов в обратной по следовательности
5. ;; DECRYPTION
6. aesimc xmm3, xmm3 ; Генерация ключей дешифрования
7. aesimc xmm4, xmm4 ; (с помощью команды Inverse Mix Columns)
8. aesimc xmm5, xmm5 ;
9. aesimc xmm6, xmm6 ;
10. aesimc xmm7, xmm7 ;
11. aesimc xmm8, xmm8 ;
12. aesimc xmm9, xmm9 ;
13. aesimc xmm10, xmm10 ;
14. aesimc xmm11, xmm11 ;
15. aesimc xmm12, xmm12 ;
16. aesimc xmm13, xmm13 ;
17. pxor xmm1, xmm14 ; Round 0 (операция XOR)
18. aesdec xmm1, xmm13 ; Round 1 (ключи берутся в обратном порядке)
19. aesdec xmm1, xmm12 ; Round 2
20. aesdec xmm1, xmm11 ; Round 3
21. aesdec xmm1, xmm10 ; Round 4
22. aesdec xmm1, xmm9 ; Round 5

```
23. aesdec xmm1, xmm8          ; Round 6
24. aesdec xmm1, xmm7          ; Round 7
25. aesdec xmm1, xmm6          ; Round 8
26. aesdec xmm1, xmm5          ; Round 9
27. aesdec xmm1, xmm4          ; Round 10
28. aesdec xmm1, xmm3          ; Round 11
29. aesdeclast xmm1, xmm2      ; Round 12
```

Модель программирования приложений

Расширения AES следуют тем же программным моделям, что и Intel SSE, Intel SSE2, Intel SSE3, Intel SSSE3 и Intel SSE4 (см. IA-32 Intel Architecture Software Developer's Manual, Пособие для разработчиков программ для 32-разрядной архитектуры Intel, том 1). Операционные системы, поддерживающие расширения SSE, будут поддерживать и работу приложений с командами AES. То же относится и к последующим версиям SSE.

Определение поддержки процессором инструкций AES

Перед тем как приложение попытается использовать инструкции AES (AESDEC, AESDECLAST, AESENC, AESENCLAST, AESIMC, AESKEYGENASSIST), оно должно убедиться, что центральный процессор эти инструкции поддерживает. Расширение AES поддерживается, если CPUID.01H:ECX.AES[бит 25] = 1.

Использование AES-NI в режиме параллельной обработки

В этой главе объясняется, как производительность алгоритма AES может быть существенно увеличена с помощью использования приемов параллельного программирования. Посмотрите на код реализации AES-128 шифрование в режиме ECB. В нем обрабатываются 8 блоков данных в регистрах xmm2-xmm9, и ключ раунда содержится в xmm1. В каждом раунде выполняется 8 инструкций над 8 разными блоками с одним и тем же ключом раунда. Затем загружается следующий ключ раунда. 8 блоков с кодированной информацией сохраняется в память, в готовности загрузить следующие 8 блоков. Таким образом, получается, что программа параллельно обрабатывает 8 блоков данных, но порядок обработки отличается от порядка в примере, рассмотренном в предыдущей главе. Вместо того чтобы полностью шифровать один блок и потом переходить к следующему, код выполняет один раунд на всех восьми блоках с использование одного ключа раунда, затем переходит к следующему раунду с другим ключом. Такая циклично-реверсивная ("loop-reversal") технология подходит для любых режимов параллельного выполнения, таких как CTR и расшифровка CBC (но не CBC шифрование).

Конвейерный процессор подразумевает, что за каждый такт может обрабатываться одна инструкция AES, при условии готовности данных. В режиме параллельной обработки, с использованием инструкций AES и технологии loop-reversing, данные определенно будут готовы для обработки в (почти) каждом цикле.

Следующий довольно грубый тест производительности показывает выигрыш в скорости (пренебрегая временем на обращение к памяти). Примем, что время выполнения инструкций AES составляет L тактов, и $L \leq 8$ (фактическая латентность инструкций AES составляет 6 тактов, и в этом примере они оперируют с 8 регистрами xmm). Тогда шифрование 8 блоков данных будет выполнено за (приблизительно) 88+L тактов (операция pxor выполняется за 1 такт). Следовательно, достигнутая производительность достигает (88+6)/8=12 тактов на блок (16B), что приближается к теоретическому пределу производительности. Этот упрощенный расчет не учитывает некоторые факторы (такие как время на загрузку и сохранение), но, тем не менее, эффект близок к полученной нами величине.

Заключение

В этом документе описаны новые инструкции шифрования AES фирмы Intel, которые будут поддерживаться всеми процессорами этой фирмы, начиная с 2009 года. AES является ведущим стандартом симметричного шифрования, который используется в широком диапазоне приложений, и постоянно растущий спрос на безопасность и приватность данных обеспечил его повсеместное применение. Отсюда следует и

важность технологии, обеспечивающей высокую производительность и безопасность алгоритмов шифрования в бытовых процессорах. Представленная архитектура AES располагает шестью командами для шифрования, дешифрования и расширения ключа, в данном документе приведены детальные подробности их использования в программах. Вместе это дает всестороннее аппаратное решение для технологии AES, предоставляя высокую производительность, гибкость и безопасность в отношении атак по сторонним каналам.

Литература

1. Криптографическая защита информации в АСУ СН. Курс лекций. В.И. Долгов. ХВУ. 1998.

2. Криптографическая защита информации в информационных системах. Курс лекций. И.Д. Горбенко. ХНУРЭ. 2002.

3. Теория информации. Курс лекций. В.И. Долгов. ХВУ. 1998.

4. Брюс Шнайер. Прикладная криптография. 2-ое издание. Протоколы, алгоритмы и исходные тексты на языке С. Доступно: http://nrjetix.com/r-and-d/lectures

5. Национальный институт стандартов и технологий. Web-сайт: National Institute of Standards and Technology

6. Описание федерального стандарта AES. Federal Information Processing Standards Publication 197 November 26, 2001 Specification for the ADVANCED ENCRYPTION STANDARD (AES). Доступно: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

7. Шэй Герон (Shay Gueron). Описание стандарта шифрования AES и новые инструкции процессоров Intel. Доступно: http://software.intel.com/ru-ru/articles/advanced-encryption-standard-aes-instructions-set/