ISSN 2409-546X

ЮНЫЙ УЧЁНЫЙ

МЕЖДУНАРОДНЫЙ НАУЧНЫЙ ЖУРНАЛ

2
Часть 1
2021

6+

Юный ученый

Международный научный журнал

№ 2 (43) / 2021

Издается с февраля 2015 г.

СОДЕРЖАНИЕ

Разработка алгоритма и программного обеспечения для шифрования данных

Оловянишников Александр Романович, учащийся 11-го класса

Научный руководитель: *Симаков Егор Евгеньевич, учитель математики, информатики и ИКТ*
МБОУ Лицей № 1 г. Южно-Сахалинска

Человечество за всю свою историю никогда не развивалось так быстро, как в новейшем времени. Если раньше компьютер могла купить только крупная компания, то сейчас его может позволить себе каждый. Вместе с компьютером в жизнь людей прочно вошел интернет, с помощью которого стало возможным передавать информацию по всему земному шару в считанные секунды. Однако такие возможности имеют свои минусы, и самым главным из них является проблема информационной безопасности.
Ключевые слова: *шифрование, информационная безопасность, защита информации, алгоритм AOCRYPT.*

I. Решение проблемы защиты данных. Алгоритмы шифрования

Данные в сети могут быть украдены злоумышленниками и использованы в неблагоприятных целях. Некоторые вредоносные программы могут просто занимать место на диске и замедлять ОС, а другие перезаписывать MBR (загрузочная область диска) своим кодом и при перезапуске компьютера стирать данные с дисков (например, VineMEMZ). Во избежание запуска трояна — вредоносной программы, маскирующейся под любую другую, используется криптографическая функция SHA-256, MD5 и др. Результат функции является уникальным для каждого файла, поэтому, зная проверочную сумму оригинала, можно сразу понять о принадлежности данного файла к вредоносному ПО. Кроме того, многие криптографические алгоритмы были реализованы на компьютере, что позволило зашифровывать информацию с помощью сгенерированного ключа, который невозможно подобрать, ведь на это могут уйти десятки лет (например, RSA-2048).

Все алгоритмы шифрования делятся на две группы: симметричные и ассиметричные. Для симметричных алгоритмов нужен один и тот же ключ для кодирования и декодирования информации, а для ассиметричных — разные. Если алгоритм не поддается взлому в течение пяти лет — значит он может использоваться для защиты секретной информации. Ниже приведены одни из надежных и распространенных криптографических алгоритмов:

— **AES** в настоящее время является федеральным стандартом шифрования США. Используется вариант шифра с размером блока 128 бит.
— **Blowfish** — сложная схема выработки ключа существенно затрудняет атаку на алгоритм методом перебора, однако делает его непригодным для использования в системах, где ключ часто меняется, и на каждом ключе шифруется небольшие по объему данные. Алгоритм подходит для систем, в которых на одном и том же ключе шифруются большие массивы данных.
— **RSA** — алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел. Стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Принцип работы всех криптографических алгоритмов одинаков: генерируется ключ, в соответствии с которым по установленному алгоритму изменяются данные и записываются в зашифрованный файл. Однако применение криптографических алгоритмов разное. Автором разработан собственный симметричный алгоритм (AOCRYPT) и программа-шифровальщик (AOCRYPT0R) на языке программирования C++.

Рассмотрим более подробно принципы работы двух наиболее популярных алгоритмов — BlowFish и RSA.

Алгоритм шифрования BLOWFISH. Blowfish — криптографический алгоритм, реализующий блочное симметричное шифрование с переменной длиной ключа. Разработан Брюсом Шнайером в 1993 году. Выполнен на простых и быстрых операциях: XOR, подстановка, сложение. Является незапатентованным и свободно распространяемым. По заявлению автора, критериями проектирования Blowfish были: скорость (шифрование на 32-битных процессорах происходит за 26 тактов); простота (за счёт использования простых операций, уменьшающих вероятность ошибки реализации алгоритма); компактность (возможность работать в менее, чем 5 Кбайт памяти); настраиваемая безопасность (изменяемая длина ключа).

Алгоритм разделён на этапы.

1. Инициализация массивов P и S при помощи секретного ключа K
— Инициализация фиксированной строкой, состоящей из шестнадцатеричных цифр мантиссы числа пи.
— Производится операция XOR над P_1 с первыми 32 битами ключа K, над P_2 со вторыми 32-битами и так далее. Если ключ K короче, то он накладывается циклически.
2. Шифрование ключей и таблиц замен
— Алгоритм шифрования 64-битного блока, используя инициализированные ключи P_1 — P_18

и таблицу замен S_1 — S_4, шифрует 64 битную нулевую (0x0000000000000000) строку. Результат записывается в P_1, P_2.
— P_1 и P_2 шифруются изменёнными значениями ключей и таблиц замен. Результат записывается в P_3 и P_4.
— Шифрование продолжается до изменения всех ключей P_1 — P_18 и таблиц замен S_1 — S_4.

3. Шифрование текста полученными ключами и F(x), с предварительным разбиением на блоки по 64 бита. Если невозможно разбить начальный текст точно на блоки по 64 бита, используются различные режимы шифрования для построения сообщения, состоящего из целого числа блоков. Суммарная требуемая память 4168 байт.

Дешифрование происходит аналогично, только P_1 — P_18 применяются в обратном порядке.

Алгоритм шифрования RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом. Принцип работы алгоритма представлен на схеме ниже.

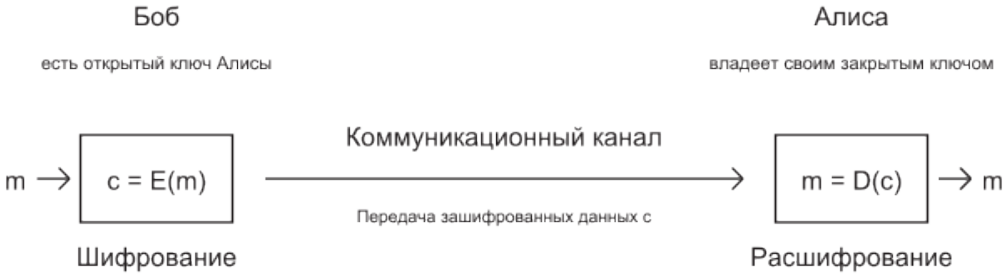

Рис. 1. Схема работы RSA (шифрование и дешифрование)

Предположим, Боб хочет послать Алисе сообщение m. Сообщениями являются целые числа в интервале от 0 до n-1, т. е. m принадлежит Zn.

1. Алгоритм шифрования:
— Взять открытый ключ (e, n) Алисы
— Взять открытый текст m
— Зашифровать сообщение с использованием открытого ключа Алисы:
— C = E(m) = me mod n
2. Алгоритм дешифрования:
— Принять зашифрованное сообщение c
— Взять свой закрытый ключ (d, n)
— Применить закрытый ключ для расшифрования сообщения:
— m = D(c) = cd mod n

Данная схема на практике не используется по причине того, что она не является надёжной. Функция E(m) является детерминированной — при одних и тех же значениях входных параметров выдаёт одинаковый результат. В настоящее время используется смешанный алгоритм шифрования, в котором сначала шифруется сеансовый ключ, а потом уже с его помощью участники шифруют свои сообщения симметричными системами. После завершения сеанса сеансовый ключ, как правило, уничтожается.

Рис. 2. Алгоритм шифрования сеансового ключа

II. Разработка алгоритма AOCRYPT

Создание программного обеспечения на основе разрабатываемого алгоритма AOCRYPT произведено на языке C++ в среде Code::Blocks — это свободная кроссплатформенная среда разработки. Code::Blocks использует библиотеку wxWidgets. Имея открытую архитектуру, может масштабироваться за счёт подключаемых модулей. Code::Blocks разрабатывается для Windows, Linux и Mac OS X. Среду можно собрать из исходников практически под любую Unix-подобную систему, например FreeBSD, PC-BSD.

Принцип работы AOCRYPT

Самый простой способ зашифровать данные — это просто инвертировать каждый байт файла, то есть: ***новое значение = 255 — оригинальное значение.*** Данный алгоритм является самым простым и небезопасным способом шифрования. Для надежного шифрования нужно сгенерировать ключ, но создавать его с помощью функции srand() крайне небезопасно в криптографии, потому что данная функция создает предсказуемые значения ключа. Безопаснее будет использовать специальные библиотеки, но самым безопасным вариантом является генерация ключа, связанная с человеческим фактором. Человек может вводить данные с помощью клавиатуры и мыши. Создавать ключ с помощью ввода с клавиатуры займет много времени, а с мышью проще и «неповторимей», т. к. сдвиг на 1–2 пикселя повлияет на результат. Ниже приведен алгоритм генерации ключа (max = 512 байт — размер ключа):

```
for (int i = 0; i < max; i++)
{
    again:
    if (GetCursorPos(&p))
    {
        if ((p.x == oldx) && (p.y == oldy)) goto again;
        oldx = p.x;
        oldy = p.y;
        key[i] = (p.x*p.y) / 256;
        if (key[i] == 0) goto again;
    }
    delay(10);
}
```

Рис. 3. Алгоритм генерации ключа

Итак, ключ готов! Пусть файл шифруется по данному алгоритму:

новое значение = ключ [i] + оригинальное значение

Такой способ обеспечивает более высокую безопасность, но зная тип зашифрованного файла, можно сразу определить первые значения ключа. Например, каждое выполняемое приложение ОС Windows и некоторые для MSDOS начинаются с MZ — инициалов Марка Збиковски. Подставив значения в формулу выше и решив простое уравнение, можно определить два значения ключа, а так как каждые 512 байт будут шифроваться одними и теми же числами, то можно расшифровать часть значений после инициалов.

```
4D 5A 80 00 01 00 00 00 04 00 10 00 FF FF 00 00  MZЂ.........яя..
40 01 00 00 00 00 00 00 40 00 00 00 00 00 00 00  @.......@.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00  ............Ђ...
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..є..ґ.H!ё.LH!Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
6D 6F 64 65 2E 0D 0A 24 00 00 00 00 00 00 00 00  mode...$........
```

Рис. 4. Заголовок.exe файла (обведен в синий прямоугольник)

Было решено разбить ключ на две части по 256 байт. Пусть первая часть будет отвечать за числа, с которыми производят действия, а вторая — за сами действия, причем первые 4 бита — номер действия, а вторые — номер дополнительного числа в ряду. Теперь с каждым байтом файла будут производиться действия, в которых будут принимать участие 2 байта ключа.

Первое число = ключ [i]

Второе число = ключ [номер доп. числа в строке из 16 байт]

новое значение = оригинальное значение ± результат действия

Данный алгоритм является еще более надежным, чем предыдущие версии AOCRYPT, но если файл имеет данные, которые повторяются каждые 256 байт, то злоумышленник может сразу догадаться, что файл имеет повторяющееся фрагменты. Заключительный этап разработки AOCRYPT — добавление сдвига действий на 4 бита каждые 256 байт данных. Конечный алгоритм шифрования данных приведен ниже. В программе использованы следующие переменные:

— ac — массив номеров доп. чисел и действий над ними

— f — номер элемента ключа; d — номер действия или доп. числа
— oddnum — доп. число
— sec — номер действия; shft — сдвиг (в 4 бит)
— fdata — данные файла; fsize — размер файла

```
int ac[max];

int f = 0;

for (int i = 256; i<max; i++)
{
    ac[f] = key[i] / 16;
    ac[f + 1] = key[i] - ac[f] * 16;
    f += 2;
}

f = 0;
int d = 0, n = 0, sec = 0, oddnum = 0, chk = 0, shft = 0;

for (long i = 0; i<fsize; i++)
{
    n = 0;
    sec = ac[d + shft];
    repeat:
    d++;
    if (d >= max - shft) d = 0;
    if (n == 0)
    {
        n = 1;
        chk = fdata[i];
        oddnum = key[ac[d + shft] + 16 * (int)(f / 16)];
        if (sec == 0) fdata[i] += key[f] + oddnum;
        if (sec == 1) fdata[i] += key[f] - oddnum;
        if (sec == 2) fdata[i] += key[f] * oddnum;
        if (sec == 3) fdata[i] -= key[f] * oddnum;
        if (sec == 4) fdata[i] -= key[f] + oddnum;
        if (sec == 5) fdata[i] -= key[f] - oddnum;
        if (sec == 6) fdata[i] += key[f] * key[f] + oddnum;
        if (sec == 7) fdata[i] += key[f] * key[f] - oddnum;
        if (sec == 8) fdata[i] += key[f] + oddnum * oddnum;
        if (sec == 9) fdata[i] += key[f] - oddnum * oddnum;
        if (sec == 10) fdata[i] += key[f] * key[f] + oddnum * oddnum;
        if (sec == 11) fdata[i] += key[f] * key[f] - oddnum * oddnum;
        if (sec == 12) fdata[i] += key[f] * oddnum + key[f];
        if (sec == 13) fdata[i] += key[f] * oddnum - key[f];
        if (sec == 14) fdata[i] += oddnum * (key[f] + oddnum);
        if (sec == 15) fdata[i] += oddnum * (key[f] - oddnum);
        if (fdata[i] == chk) fdata[i] += 32;
        goto repeat;
    }
    f++;
    if (f >= max / 2) f = 0, shft++;
    if (shft >= max) shft = 0;
}
```

Рис. 5. Конечный вариант алгоритма шифрования

III, Разработка программного обеспечения AOCRYPT0R

Для создания графического интерфейса использован Windows API — набор базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows, который предоставляет прямой способ взаимодействия приложений пользователя с операционной системой Windows. Windows API представляет собой множество функций, структур данных и числовых констант, следующих соглашениям языка Си. В то же время конвенция вызова функций отличается от cdecl, принятой для языка C: Windows API использует stdcall (winapi). Для облегчения переноса на другие платформы программ, написанных с опорой на Windows API, разработана библиотека Wine.

Wine — свободное программное обеспечение, позволяющее пользователям UNIX-подобных операционных систем исполнять 16-, 32- и 64- битные приложения без наличия при этом установленной Microsoft Windows. Wine также предоставляет программистам библиотеку программ Winelib, при помощи которой они могут компилировать Windows-приложения для портирования их в UNIX-подобные операционные системы. Название W. I. N.E. — рекурсивный акроним и расшифровывается «Wine Is Not an Emulator» — «W. I. N.E. — это не эмулятор». Имеется в виду, что Wine это не эмулятор компьютера или виртуальная машина, как, например, qemu, VirtualBox и другие подобные им. С помощью Wine данная программа может свободно работать на ОС Linux и MacOS.

В интерфейсе программы использованы следующие элементы: кнопки, поля для отображения локации файла на диске. Соответственно, при нажатии кнопки *Browse* создается диалог выбора файла, кнопки *Process* файл шифруется или расшифровывается, кнопки *Quit* программа завершает работу. Также в интерфейс добавлен логотип разработчика, рамки, музыкальное сопровождение. Все это было сжато в 50.0Кб, что обеспечивает быстрое скачивание и эксплуатацию данного ПО.

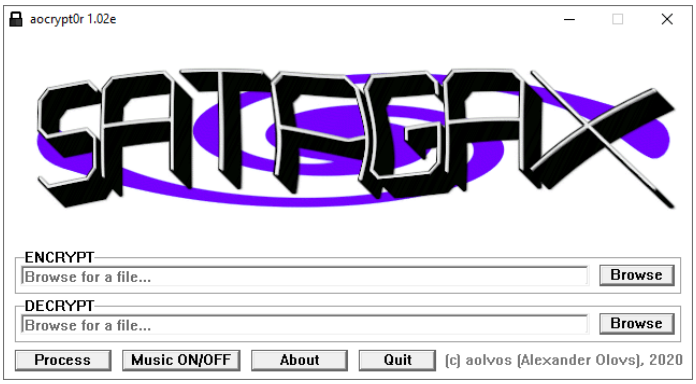

Рис. 6. Графический интерфейс программы Aocrypt0r

При разработке программного обеспечения были решены следующие проблемы. Во-первых, во время работы программы было замечено сильное мерцание всех элементов окна. Это было обусловлено тем, что видеокарта отрисовывала большее число кадров в секунду, чем мог отрисовать монитор. Окно программы вырисовается следующим образом: сначала происходит полное заполнение всего белым цветом, а потом начинают отображаться элементы. Так, мерцание происходило из-за того, что на экран выводились моменты, когда все окно было закрашено белым цветом. Для решения проблемы использован алгоритм вертикальной синхронизации — двойная буферизация. Для этого изображение, отрисованное видеокартой, подаётся не сразу на экран, а в специальный буфер (область памяти). Как только монитор отрисует следующий кадр, содержимое этого буфера передаётся на монитор, а в буфер пишется следующий кадр. С технической точки зрения это реализованно как смена указателя на первичный буфер (тот, из которого изображение поступает на монитор): два буфера меняются местами, и, пока изображение из нового буфера выводится на экран, следующий кадр пишется в старый буфер. Затем эта операция повторяется.

Логотип SATAGAX был сжат с помощью **собственного алгоритма imagecom-256.** Формат поддерживает до 255 цветов. Первые два слова — размеры изображения в пикселях, затем идет байт, отвечающий за количество цветов в палитре, затем данные самой палитры в формате RGB, потом номера цветов в соответствии с палитрой. Если есть повторяющееся подряд цвета, то ставится сигнальный байт 0, затем количество повторений цвета и номер цвета в палитре. Это очень эффективно сжимает простые изображения, в отличие от BMP. Так в формате BMP логотип весит 126.0Кб, PNG: 25.2Кб, imagecom-256: 53.0Кб. Но, если PNG фактически больше наподдаётся сжатию, то изображения imagcom-256 очень хорошо сжимаются, и после сжатия, например, в zip архив, размер становится равным 25.3Кб. Программа распаковывает данные в память в формате BGRA, а затем создает в памяти 24-х битное BMP изображение и выводит его на экран с помощью команды: **StretchDIBits(hdcmem, 0, 0, xres, yres, 0, 0, xres, yres, logodata, &info, DIB_RGB_COLORS, SRCCOPY);** xres, yres — размеры изображения, logodata — данные, info — заголовок BMP изображения в памяти.

Алгоритм работает следующим образом. Пусть существует некоторое изображение. В начале оно конвертируется в изображение с палитрой на 255 цветов, затем imagecom-256 создает и записывает в файл палитру изображения. Далее imagecom-256 считывает цвет каждого пикселя изображения и сравнивает его с цветом в палитре. Если цвета совпадают, то в файл записывается номер соответствующего цвета в палитре.

Одним из критериев при создании программного обеспечения был минимальный размер (до 50 кБ). На свободное после разработки основной части программы интерфейса место была добавлена фоновая трекерная музыка. Размер аудио файла — 12.0Кб. Для сравнения: WAV файл всего в 2 секунды весит целых 1.5Мб! Трекерная (модульная) музыка — музыка, созданная на компьютере при помощи программы-»трекера» (в данном случае Fasttracker II), представляющей собой специализированный музыкальный редактор. Музыка формата XM занимает очень маленькие объемы из-за принципа хранения данных в виде номеров нот, команд и 8-ми битных WAV семплов. Для проигрывания XM формата использована библиотека UFMOD, разработанную на ассемблере.

Заключение

Таким образом, и алгоритм, и программное обеспечение для шифрования были успешно созданы, но это далеко не предел возможностей криптографии. Программа-шифровальщик имеет общий размер, равный 50.0Кб, что делает ее портативной и удобной в использовании. Кроме того, с помощью ПО Wine стал возможен запуск данной программы не только на ОС Windows, но и на Linux и MacOS! Ключ алгоритма имеет размер в 512 байт, а также зависимость выполняемых операций от ключа, что делает взлом перебором практически невозможным в настоящее время. В целом, любая программа-шифровальщик представляет собой несложное, но мощное приложение, превращающее файлы в массив данных, который можно декодировать только при наличии ключа. Данный алгоритм может применятся как для электронных подписей, так и для шифрования файлов или архивов. Способен работать с большими объемами информации, для чего и был предназначен.

Сравнение кодировок приведено ниже (1-исходный файл, 2-инвертирование, 3-aocrypt + 4-ключ)

В стадии разработки находится отдельная операционная система, загружающееся с внешнего устройства и работающая в 16-ти битном незащищенном режиме, что позволяет использовать драйвера BIOS и неограниченно изменять любые данные на жестких дисках. Данная ОС позволит зашифровывать целые диски с помощью алгоритма AOCRYPT, который будет совершенствоваться, если будут обнаружены серьезные уязвимости. Шифрование дисков целиком не только не даст загрузить систему и просмотреть файлы, но и просмотреть файлы на диске, подключив его к другому устройству. Так, работа над проектом еще не завершена, и в будущем есть множество перспектив его развития.

Разработанное программное обеспечение Aocrypt0r 1.02e находится в свободном доступе по ссылке: http://satagax.com/data/aocrypt0r1.02e.zip Альтернативная ссылка:

https://drive.google.com/uc?export=download&id=146XmbIRZwHj6gF7mGPg-xa8nSt1aIvjl).

Исходный код алгоритма сжатия изображений imagecom-256 доступен по ссылке: https://github.com/aolvos/imagecom-256

Рис. 7. Сравнение кодировок

ЛИТЕРАТУРА:

1. Баричев, С. Г. Основы современной криптографии. — М.: СИНТЕГ, 2011.
2. Фергюсон, Н.. Практическая криптография. — М.: Диалектика, 2004.
3. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы. — М.: Триумф, 2002.
4. Интернет-энциклопедия «Википедия». RSA. MZ. Blowfish [Электронный ресурс]. URL: https://ru.Wikipedia.org/wiki/RSA; https://ru.wikipedia.org/wiki/MZ; https://ru.wikipedia.org/wiki/Blowfish
5. Официальный сайт UFMOD [Электронный ресурс]. URL: https://ufmod.sourceforge.io/ind_ru.htm
6. Портал stackoverflow.com [Электронный ресурс]. URL: https://ru.stackoverflow.com/
7. Портал rohos.ru [Электронный ресурс]. URL: http://www.rohos.ru/help/crypto_algorithms