Практическая работа по теме: "Фиксация контрольных сумм программного обеспечения."

Гришин Александр, Костромин Марк ККСО-01-21

03.06.2022

Содержание

1 Введение

В настоящее время очень остро стоит вопрос о защите информации в сети, а именно об защите ваших личных данных, которые шифруются с помощью контрольных сумм, именно поэтому изучение этого аспекта является важной частью обучения специалиста по защите информации.

В данной практической работе мы расскажем о основах криптографии, рассмотрим и сравним некоторые алгоритмы фиксации контрольных сумм, отечественные блочные алгоритмы шифрования и алгоритмы подсчета хэш-функций, а в конце разберем программный код, который мы написали для данной практической работы.

2 Основы криптографии

В переводе с греческого языка слово *криптография* означает *тайнопись*. Смысл этого термина выражает основное предназначение криптографии — защитить или сохранить в тайне необходимую информацию.

Криптография дает средства для защиты информации, и поэтому она *является частью деятельности по обеспечению безопасности информации.*

Существуют различные методы защиты информации, но криптография, в отличие от остальных, не «прячет» передаваемы сообщения, а преобразует их в форму, недоступную для понимания противником.

Помимо *сокрытия* существуют и другие проблемы защиты передаваемой информации. Например, при полностью открытом информационном обмене возникает *проблема достоверности полученной информации.* Для ее решения необходимо обеспечить:

- проверку и подтверждение подлинности содержания и источника сообщения;
- предотвращение и обнаружение обмана и других умышленных нарушений со стороны самих участников информационного обмена.

Для решения этой проблемы обычные средства, применяемые при построении систем передачи информации, подходят далеко не всегда. Именно криптография дает средства для обнаружения обмана в виде подлога или отказа от ранее совершенных действий, а также других неправомерных действий.

Поэтому можно сказать, что современная криптография является областью знаний, связанной с решением таких проблем безопасности информации, как конфиденциальность, целостность, аутентификация и невозможность отказа сторон от авторства. Достижение этих требований безопасности информационного взаимодействия и составляет основные цели криптографии. Они определяются следующим образом.

- Обеспечение конфиденциальности — решение проблемы защиты информации от ознакомления с ее содержанием со стороны лиц, не имеющих права доступа к ней. В зависимости от контекста вместо термина “конфиденциальная” информация могут выступать термины “секретная”, “частная”, “ограниченного доступа” информация.
- Обеспечение целостности — гарантирование невозможности несанкционированного изменения информации. Для гарантии целостности необходим простой и надежный критерий обнаружения любых манипуляций с данными. Манипуляции с данными включают вставку, удаление и замену.
- Обеспечение аутентификации — разработка методов подтверждения подлинности сторон (идентификация) и самой информации в процессе информационного взаимодействия. Информация, передаваемая по каналу связи, должна быть аутентифицирована по источнику, времени создания, содержанию данных, времени пересылки и т. д.
- Обеспечение невозможности отказа от авторства — предотвращение возможности отказа субъектов от некоторых из совершенных ими действий. Рассмотрим средства для достижения этих целей более подробно.

2.1 Конфидециальность

Для описания этой задачи введем следующие понятия:

- Отправитель – владелец информации;
- Противник – любой субъект, не имеющий права ознакомления с содержанием передаваемой информации. В качестве противника может выступать криптоаналитик, владеющий методами раскрытия шифров.
- Законный получатель – субъект, который должен получить информацию.

В самом начале отправитель, осуществляет преобразование исходной (открытой) информации (сам процесс преобразования называется шифрованием) в форму передаваемых получателю по открытому каналу связи шифрованных сообщений с целью ее защиты от противника.

Законный получатель информации осуществляет расшифрование полученных сообщений. Противник пытается овладеть защищаемой информацией (его действия обычно называют атаками). При этом он может совершать как пассивные, так и активные действия. Пассивные атаки связаны с прослушиванием, анализом трафика, перехватом, записью передаваемых шифрованных сообщений, дешифрованием, т. е. попытками "взломать" защиту с целью овладения информацией.

При проведении активных атак противник может прерывать процесс передачи сообщений, создавать поддельные (сфабрикованные) или модифицировать передаваемые шифрованные сообщения. Эти активные действия называют попытками имитации и подмены соответственно.

Определение Шифр - семейство обратимых преобразований, каждое из которых определяется некоторым параметром, называемым ключом, а также порядком применения данного преобразования, называемым режимом шифрования.

Определение Ключ — это важнейший компонент шифра, отвечающий за выбор преобразования, применяемого для зашифрования конкретного сообщения. Обычно ключ представляет собой некоторую буквенную или числовую последовательность. Эта последовательность как бы "настраивает" алгоритм шифрования.

Каждое преобразование однозначно определяется ключом и описывается некоторым криптографическим алгоритмом.

Определение Криптосистема (шифрсистема) – пара алгоритмов зашифрования и расшифрования.

Если обозначить через M открытое, а через C шифрованное сообщения, то процессы зашифрования и расшифрования можно записать в виде равенств:

$$E_{k_1}(M) = C$$
$$D_{k_2}(C) = M$$

в которых алгоритмы зашифрования E и расшифрования D должны удовлетворять равенству:

$$D_{k_2}(E_{k_2}(M)) = M$$

Различают симметричные и ассиметричные криптосистемы:

- симметричные. В таких системах знание ключа зашифрования k_1 позволяет легко найти ключ расшифрования k_2 (в большинстве случаев эти ключи просто совпадают);
- в асимметричных криптосистемах знание ключа k_1 не позволяет определить ключ k_2.

Поэтому для симметричных криптосистем оба ключа должны сохраняться в секрете, а для асимметричных — только один — ключ расшифрования k_2, а ключ k_1 можно сделать открытым (общедоступным). В связи с этим их называют еще шифрами с открытым ключом.

Симметричные криптосистемы принято подразделять на поточные и блочные системы. Поточные системы осуществляют зашифрование отдельных символов открытого сообщения. Блочные же системы производят зашифрование блоков фиксированной длины, составленных из подряд идущих символов сообщения.

Асимметричные криптосистемы, как правило, являются блочными.

Определение Криптоанализ – наука, которая разрабатывает методы вскрытия шифров.

2.2 Целостность

Наряду с конфиденциальностью не менее важной задачей является обеспечение целостности информации, другими словами, — неизменности ее в процессе передачи или хранении. Решение этой задачи предполагает разработку средств, позволяющих обнаруживать не столько случайные искажения, сколько целенаправленное навязывание противником ложной информации. Для этого в передаваемую информацию вносится избыточность. Как правило, это достигается добавлением к сообщению некоторой проверочной комбинации, вычисляемой с помощью специального алгоритма и играющей роль контрольной суммы для проверки целостности полученного сообщения. Главное отличие такого метода от методов теории кодирования состоит в том, что алгоритм выработки проверочной комбинации является "криптографическим", то есть зависящим от секретного ключа. Без знания секретного ключа вероятность успешного навязывания противником искаженной или ложной информации мала. Такая вероятность служит мерой имитостойкости шифра, то есть способности самого шифра противостоять активным атакам со стороны противника.

Итак, для проверки целостности к сообщению M добавляется проверочная комбинация S, называемая кодом аутентификации сообщения (сокращенно — КАС) или имитовставкой. В этом случае по каналу связи передается пара $C = (M, S)$. При получении сообщения M пользователь вычисляет значение проверочной комбинации и сравнивает его с полученным контрольным значением S. Несовпадение говорит о том, что данные были изменены.

Как правило, код аутентификации является значением некоторой (зависящей от секретного ключа) криптографической хэш-функции от данного сообщения: $h_k(\mathrm{M}) = S$. К кодам аутентификации предъявляются определенные требования. К ним относятся:

- невозможность вычисления значения $h_k(M) = S$ для заданного сообщения M без знания ключа k;
- невозможность подбора для заданного сообщения M с известным значением $h_k(M) = S$ другого сообщения M_1 с известным значением $h_k(M_1) = S_1$ без знания ключа k.

Первое требование направлено против создания поддельных (сфабрикованных) сообщений при атаках типа имитация; второе — против модификации передаваемых сообщений при атаках типа подмена.

2.3 Аутентификация

Определение Аутентификация — установление подлинности. Особенно остро эта проблема стоит в случае не доверяющих друг другу сторон, когда источником угроз может служить не только третья сторона (противник), но и сторона, с которой осуществляется взаимодействие.

Применительно к сторонам взаимодействия аутентификация означает проверку одной из сторон того, что взаимодействующая с ней сторона — именно та, за которую она себя выдает. Часто аутентификацию сторон называют также идентификацией.

Основным средством для проведения идентификации являются протоколы идентификации, позволяющие осуществлять идентификацию (и аутентификацию) каждой из участвующих во взаимодействии и не доверяющих друг другу сторон. Различают протоколы односторонней и взаимной идентификации.

Протокол — это распределенный алгоритм, определяющий последовательность действий каждой из сторон.

2.4 Формальные модели шифров

Для того чтобы иметь возможность доказывать в криптографии точные результаты, нужны математические модели основных исследуемых объектов, к которым относятся в первую очередь шифр и открытый текст. Введем сначала алгебраическую модель шифра (шифрсистемы), предложенную К. Шенноном.

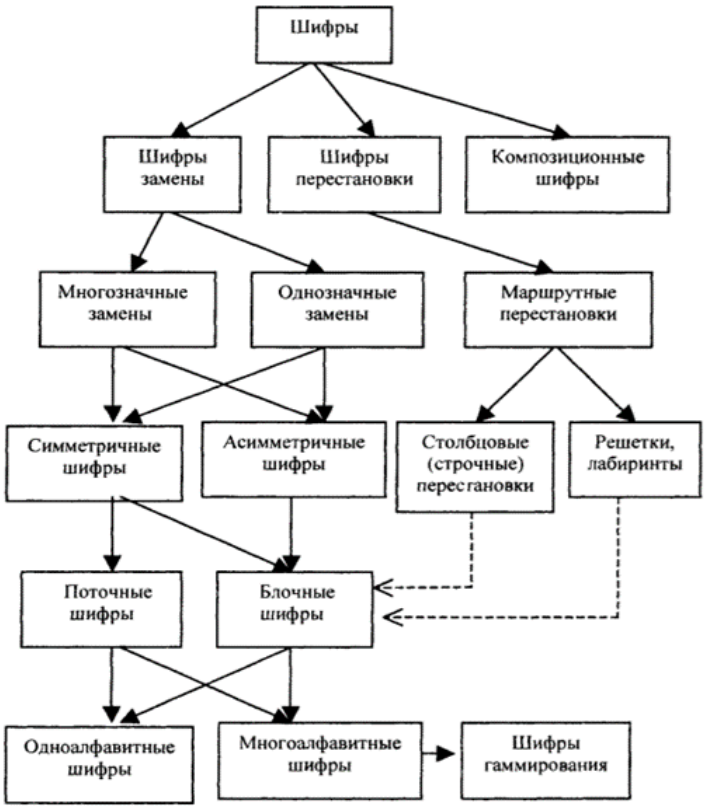
Пусть X, Y, K - конечные множества возможных открытых текстов, ключей и шифрованных текстов соответственно; $E_k : X \Rightarrow Y$ правило зашифрования на ключе $k \in K$. Множество $\{E_k : k \in K\}$ обозначим через E, а множество $\{E_k(x) : x \in X\}$ обозначим через $E_k(X)$. Пусть $D_k : E_k(X) \Rightarrow X$ - правило расшифрования на ключе $k \in K$, и D - множество $\{D_k : k \in K\}$

Определение Шифром (шифрсистемой) назовем совокупность $\Sigma_A = (X, K, Y, E, D)$ введенных множеств, для которых выполняются следующие свойства:

1. Для любых $x \in X$ и $k \in K$ выполняется равенство: $D_k(E_k(x)) = x$;
2. $Y = \bigcup_{k \in K} E_k(X)$

2.5 Классификация шифров по различным признакам

В качестве первичного признака, по которому производится классификация шифров, используется тип преобразования, осуществляемого с открытым текстом при шифровании. Если фрагменты открытого текста (отдельные буквы или группы букв) заменяются некоторыми их эквивалентами в шифртексте, то соответствующий шифр относится к классу шифров замены. Если буквы открытого текста при шифровании лишь меняются местами друг с другом, то мы имеем дело с шифром перестановки. С целью повышения надежности шифрования шифрованный текст, полученный применением некоторого шифра, может быть еще раз зашифрован с помощью другого шифра. Все возможные такие композиции различных шифров приводят к третьему классу шифров, которые обычно называют композиционными шифрами. Заметим, что композиционный шифр может не входить ни в класс шифров замены, ни в класс шифров перестановки. В результате получаем первый уровень классификации шифров.



3 Алгоритмы фиксации контрольных сумм

3.1 Хэш-функция MD5

Рассмотрим алгоритм получения дайджеста сообщения MD5 (RFC 1321), разработанный Роном Ривестом из MIT.

Логика выполнения MD5

Алгоритм получает на входе сообщение произвольной длины и создает в качестве выхода дайджест сообщения длиной 128 бит. Алгоритм состоит из следующих шагов:

1. Добавление недостающих битов:

   Сообщение дополняется таким образом, чтобы длина $\equiv 448(mod512)$.

   Это означает, что длина добавленного сообщения на 64 бита меньше, чем число, кратное 512. Добавление производится всегда, даже если сообщение имеет нужную длину. Например, если длина сообщения 448 битов, оно дополняется 512 битами до 960 битов. Таким образом, число добавляемых битов находится в диапазоне от 1 до 512.

   Добавление состоит из единицы, за которой следует необходимое количество нулей.

2. Добавление длины:

   64-битное представление длины исходного (до добавления) сообщения в битах присоединяется к результату первого шага. Если первоначальная длина больше, чем 264, то используются только последние 64 бита. Таким образом, поле содержит длину исходного сообщения по модулю 264.

   В результате первых двух шагов создается сообщение, длина которого кратна 512 битам. Это расширенное сообщение представляется как последовательность 512-битных блоков Y0, Y1, . . ., YL-1, при этом общая длина расширенного сообщения равна L * 512 битам. Таким образом, длина полученного расширенного сообщения кратна шестнадцати 32-битным словам.

3. Инициализация MD-буфера:

   Используется 128-битный буфер для хранения промежуточных и окончательных результатов хэш-функции. Буфер может быть представлен как четыре 32-битных регистра (A, B, C, D). Эти регистры инициализируются следующими шестнадцатеричными числами:

   $A = 01234567$
   $B = 89ABCDEF$
   $C = FEDCBA98$
   $D = 76543210$

4. Обработка последовательности 512-битных (16-словных) блоков:

   Основой алгоритма является модуль, состоящий из четырех циклических обработок, обозначенный как HMD5. Четыре цикла имеют похожую структуру, но каждый цикл использует свою элементарную логическую функцию, обозначаемую $f_F, f_G, f_H f_I$ соответственно.

   Каждый цикл принимает в качестве входа текущий 512-битный блок Y_q, обрабатывающийся в данный момент, и 128-битное значение буфера ABCD, которое является промежуточным значением дайджеста, и изменяет содержимое этого буфера. Каждый цикл также использует четвертую часть 64-элементной таблицы $T[1...64]$, построенной на основе функции sin. i-ый элемент T, обозначаемый $T[i]$, имеет значение, равное целой части от $232 * abs(sin(i))$, i задано в радианах. Так как $abs(sin(i))$ является числом между 0 и 1, каждый элемент T является целым, которое может быть представлено 32 битами. Таблица обеспечивает "случайный"набор 32-битных значений, которые должны ликвидировать любую регулярность во входных данных.

Для получения MD_{q+1} выход четырех циклов складывается по модулю 2^{32} с MD_q. Сложение выполняется независимо для каждого из четырех слов в буфере.

5. Выход:

После обработки всех L 512-битных блоков выходом L-ой стадии является 128-битный дайджест сообщения.

Рассмотрим более детально логику каждого из четырех циклов выполнения одного 512-битного блока. Каждый цикл состоит из 16 шагов, оперирующих с буфером ABCD. Каждый шаг можно представить в виде:

$$A \longleftarrow B + CLSs(A + f(B, C, D) + X[k] + T[i])$$

где

A, B, C, D - четыре слова буфера; после выполнения каждого отдельного шага происходит циклический сдвиг влево на одно слово.
f - одна из элементарных функций f_F, f_G, f_H, f_I.
$CLSs$ - циклический сдвиг влево на s битов 32-битного аргумента.
$X[k] - M[q^{16} + k]$ - k-ое 32-битное слово в q-ом 512 блоке сообщения.
$T[i]$ - i-ое 32-битное слово в матрице T.
$+$ - сложение по модулю 232.

3.2 Алгоритм MD4

Алгоритм MD4 является более ранней разработкой того же автора Рона Ривеста. Первоначально данный алгоритм был опубликован в октябре 1990 г., незначительно измененная версия была опубликована в RFC 1320 в апреле 1992 г. Кратко рассмотрим основные цели MD4:

1. Безопасность: это обычное требование к хэш-коду, состоящее в том, чтобы было вычислительно невозможно найти два сообщения, имеющие один и тот же дайджест.

2. Скорость: программная реализация алгоритма должна выполняться достаточно быстро. В частности, алгоритм должен быть достаточно быстрым на 32-битной архитектуре. Поэтому алгоритм основан на простом множестве элементарных операций над 32-битными словами.

3. Простота и компактность: алгоритм должен быть простым в описании и простым в программировании, без больших программ или подстановочных таблиц. Эти характеристики не только имеют очевидные программные преимущества, но и желательны с точки зрения безопасности, потому что для анализа возможных слабых мест лучше иметь простой алгоритм.

4. Желательна little-endian архитектура: некоторые архитектуры процессоров (такие как линия Intel 80xxx) хранят левые байты слова в позиции младших адресов байта (little-endian). Другие (такие как SUN Sparcstation) хранят правые байты слова в позиции младших адресов байта (big endian). Это различие важно, когда сообщение трактуется как последовательность 32-битовых слов, потому что эти архитектуры имеют инверсное представление байтов в каждом слове. Ривест выбрал использование схемы little-endian для интерпретации сообщения в качестве последовательности 32-битных слов. Этот выбор сделан потому, что big-endian процессоры обычно являются более быстрыми.

Эти цели преследовались и при разработке MD5. MD5 является более сложным и, следовательно, более медленным при выполнении, чем MD4. Считается, что добавление сложности оправдывается возрастанием уровня безопасности. Главные различия между этими двумя алгоритмами состоят в следующем:

1. MD4 использует три цикла из 16 шагов каждый, в то время как MD5 использует четыре цикла из 16 шагов каждый.

2. В MD4 дополнительная константа в первом цикле не применяется. Аналогичная дополнительная константа используется для каждого из шагов во втором цикле. Другая дополнительная константа используется для каждого из шагов в третьем цикле. В MD5 различные дополнительные константы, $[i]$, применяются для каждого из 64 шагов.

3. MD5 использует четыре элементарные логические функции, по одной на каждом цикле, по сравнению с тремя в MD4, по одной на каждом цикле.

4. В MD5 на каждом шаге текущий результат складывается с результатом предыдущего шага. Например, результатом первого шага является измененное слово . Результат второго шага хранится в D и образуется добавлением к циклически сдвинутому влево на определенное число бит результату элементарной функции. Аналогично, результат третьего шага хранится в и образуется добавлением D к циклически сдвинутому влево результату элементарной функции. MD4 это последнее сложение не включает.

3.3 Хэш-функция SHA-1

Безопасный хэш-алгоритм (Secure Hash Algorithm) был разработан национальным институтом стандартов и технологии (NIST) и опубликован в качестве федерального информационного стандарта (FIPS PUB 180) в 1993 году. SHA-1, как и MD5, основан на алгоритме MD4.

Логика выполнения SHA-1

Алгоритм получает на входе сообщение максимальной длины 264 бит и создает в качестве выхода дайджест сообщения длиной 160 бит.

Алгоритм состоит из следующих шагов:

1. Добавление недостающих битов.

   Сообщение добавляется таким образом, чтобы его длина$\equiv 448 mod 512$. Добавление осуществляется всегда, даже если сообщение уже имеет нужную длину. Таким образом, число добавляемых битов находится в диапазоне от 1 до 512.

   Добавление состоит из единицы, за которой следует необходимое количество нулей.

2. Добавление длины.

   К сообщению добавляется блок из 64 битов. Этот блок трактуется как беззнаковое 64-битное целое и содержит длину исходного сообщения до добавления.

   Результатом первых двух шагов является сообщение, длина которого кратна 512 битам. Расширенное сообщение может быть представлено как последовательность 512-битных блоков $Y_0, Y_1, ..., Y_{L-1}$, так что общая длина расширенного сообщения есть L * 512 бит. Таким образом, результат кратен шестнадцати 32-битным словам.

3. Инициализация SHA-1 буфера.

   Используется 160-битный буфер для хранения промежуточных и окончательных результатов хэш-функции. Буфер может быть представлен как пять 32-битных регистров A, B, C, D и E. Эти регистры инициализируются следующими шестнадцатеричными числами:

   $A = 67452301$
   $B = EFCDAB89$
   $C = 98BADCFE$
   $D = 10325476$
   $E = C3D2E1F0$

4. Обработка сообщения в 512-битных (16-словных) блоках.

   Основой алгоритма является модуль, состоящий из 80 циклических обработок, обозначенный как H_{SHA}. Все 80 циклических обработок имеют одинаковую структуру.

Каждый цикл получает на входе текущий 512-битный обрабатываемый блок Y_q и 160-битное значение буфера $ABCDE$, и изменяет содержимое этого буфера.

В каждом цикле используется дополнительная константа $_t$, которая принимает только четыре различных значения:

$0 \leq t \leq 19$ $K_t = 5A827999$
(целая часть числа $[2^{30} \times 2^{\frac{1}{2}}]$)

$20 \leq t \leq 39$ $K_t = 6ED9EBA1$
(целая часть числа $[2^{30} \times 3^{\frac{1}{2}}]$)

$40 \leq t \leq 59$ $K_t = 8F1BBCDC$
(целая часть числа $[2^{30} \times 5^{\frac{1}{2}}]$)

$60 \leq t \leq 79$ $K_t = CA62C1D6$
(целая часть числа $[2^{30} \times 10^{\frac{1}{2}}]$)

Для получения SHA_{q+1} выход 80-го цикла складывается со значением SHA_q. Сложение по модулю 232 выполняется независимо для каждого из пяти слов в буфере с каждым из соответствующих слов в SHA_q.

5. Выход.

После обработки всех 512-битных блоков выходом L-ой стадии является 160-битный дайджест сообщения.

Рассмотрим более детально логику в каждом из 80 циклов обработки одного 512-битного блока. Каждый цикл можно представить в виде:

$$A, B, C, D, E(CLS_5(A) + f_t(B, C, D) + E + W_t + K_t), A, CLS_{30}(B), C, D$$

где

A, B, C, D, E - пять слов из буфера.
t - номер цикла, $0 \leq t \leq 79$.
f_t - элементарная логическая функция.
CLS_s - циклический левый сдвиг 32-битного аргумента на s битов.
W_t - 32-битное слово, полученное из текущего входного 512-битного блока.
K_t - дополнительная константа.
$+$ - сложение по модулю 232.

3.4 Сравнение SHA-1 и MD5

Оба алгоритма, SHA-1 и MD5, произошли от MD4, поэтому имеют много общего. Можно суммировать ключевые различия между алгоритмами.

	MD5	***SHA*-1**
Длина *дайджеста*	128 бит	160 бит
Размер блока обработки	512 бит	512 бит
Число итераций	64 (4 цикла по 16 итераций в каждом)	80
Число элементарных *логических функций*	4	3
Число дополнительных констант	64	4

Сравним оба алгоритма в соответствии с теми целями, которые были определены для алгоритма MD4:

1. Безопасность: наиболее очевидное и наиболее важное различие состоит в том, что дайджест SHA-1 на 32 бита длиннее, чем дайджест MD5. Если предположить, что оба алгоритма не содержат каких-либо структурированных данных, которые уязвимы для криптоаналитических атак, то SHA-1 является более стойким алгоритмом. Используя лобовую атаку, труднее создать произвольное сообщение, имеющее данный дайджест, если требуется порядка 2^{160} операций, как в случае алгоритма SHA-1, чем порядка 2^{128} операций, как в случае алгоритма MD5. Используя лобовую атаку, труднее создать два сообщения, имеющие одинаковый дайджест, если требуется порядка 2^{80} как в случае алгоритма SHA-1, чем порядка 2^{64} операций как в случае алгоритма MD5.

2. Скорость: так как оба алгоритма выполняют сложение по модулю 2^{32}, они рассчитаны на 32-битную архитектуру. SHA-1 содержит больше шагов (80 вместо 6) и выполняется на 160-битном буфере по сравнению со 128-битным буфером MD5. Таким образом, SHA-1 должен выполняться приблизительно на 25% медленнее, чем MD5 на той же аппаратуре.

3. Простота и компактность: оба алгоритма просты и в описании, и в реализации, не требуют больших программ или подстановочных таблиц. Тем не менее, SHA-1 применяет одношаговую структуру по сравнению с четырьмя структурами, используемыми в MD5. Более того, обработка слов в буфере одинаковая для всех шагов SHA-1, в то время как в MD5 структура слов специфична для каждого шага.

4. Архитектуры little-endian и big-endian: MD5 использует little-endian схему для интерпретации сообщения как последовательности 32-битных слов, в то время как SHA-1 задействует схему big-endian. Каких-либо преимуществ в этих подходах не существует.

3.5 Хэш-функции SHA-2

В 2001 году NIST принял в качестве стандарта три хэш-функции с существенно большей длиной хэш-кода. Часто эти хэш-функции называют SHA-2 или SHA-256, SHA-384 и SHA-512 (соответственно, в названии указывается длина создаваемого ими хэш-кода). Эти алгоритмы отличаются не только длиной создаваемого хэш-кода, но и длиной обрабатываемого блока, длиной слова и используемыми внутренними функциями. Сравним характеристики этих хэш-функций.

Алгоритм	Длина сообщения (в битах)	*Длина блока* (в битах)	Длина слова (в битах)	Длина *дайджеста сообщения* (в битах)	Безопасность (в битах)
SHA-1	$<2^{64}$	512	32	160	80
SHA-256	$<2^{64}$	512	32	256	128
SHA-384	$<2^{128}$	1024	64	384	192
SHA-512	$<2^{128}$	1024	64	512	256

3.6 Хэш-функция ГОСТ 34.11

Алгоритм ГОСТ 34.11 является отечественным стандартом для хэш-функций. Его структура довольно сильно отличается от структуры алгоритмов SHA-1, 2 или MD5, в основе которых лежит алгоритм MD4.

Длина хэш-кода, создаваемого алгоритмом ГОСТ 34.11, равна 256 битам. Алгоритм разбивает сообщение на блоки, длина которых также равна 256 битам. Кроме того, параметром алгоритма является стартовый вектор хэширования H - произвольное фиксированное значение длиной также 256 бит.

3.6.1 Алгоритм обработки одного блока сообщения

Сообщение обрабатывается блоками по 256 бит справа налево.

Каждый блок сообщения обрабатывается по следующему алгоритму.

1. Генерация четырех ключей длиной 256 бит каждый.
2. Шифрование 64-битных значений промежуточного хэш-кода H на ключах $K_i (i = 1, 2, 3, 4)$ с использованием алгоритма ГОСТ 28147 в режиме простой замены.
3. Перемешивание результата шифрования.

Для генерации ключей используются следующие данные:

- промежуточное значение хэш-кода H длиной 256 бит;
- текущий обрабатываемый блок сообщения M длиной 256 бит;
- параметры - три значения C_2, C_3 и C_4 длиной 256 бит следующего вида: C_2 и C_4 состоят из одних нулей, а C_3 равно $1^8 0^8 1^{16} 0^{24} 1^{16} 0^8 (0^8 1^8)^2 1^8 0^8 (0^8 1^8)^4 (1^8 0^8)^4$
  где степень обозначает количество повторений 0 или 1.

4 Отечественные блочные алгоритмы шифрования и алгоритм подсчета хэш-функций Стрибог

4.1 Стрибог

«Стрибог» (англ. STREEBOG) — криптографический алгоритм вычисления хеш-функции с размером блока входных данных 512 бит и размером хеш-кода 256 или 512 бит.

Описывается в ГОСТ 34.11-2018 «Информационная технология. Криптографическая защита информации. Функция хэширования» — действующем межгосударственном криптографическом стандарте.

Разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО «ИнфоТеКС» на основе национального стандарта Российской Федерации ГОСТ Р 34.11-2012 и введен в действие с 1 июня 2019 года приказом Росстандарта № 1060-ст от 4 декабря 2018 года.

4.1.1 Концепции построения хэш-функции «Стрибог»

В соответствии с требованиями, высказанными на конференции РусКрипто-2010, в работе, посвящённой новой хеш-функции:

- у новой хеш-функции не должно быть свойств, которые позволяли бы применить известные атаки;
- в хеш-функции должны использоваться изученные конструкции и преобразования;
- вычисление хеш-функции должно быть эффективным, занимать мало времени;
- не должно быть лишних преобразований, усложняющих конструкцию хеш-функции. Причем каждое используемое в хеш-функции преобразование должно отвечать за определённые криптографические свойства.

В той же работе вводятся «универсальные» требования, касающиеся трудоемкости атак на хеш-функцию:

Задача	Сложность
построение прообраза	2^n
построение коллизии	$2^{n/2}$
построение второго прообраза	2^n/(длина сообщения)
удлинение прообраза	2^n

4.1.2 Функция сжатия

В хеш-функции важным элементом является функция сжатия. В ГОСТ Р 34.11-2012 функция сжатия основана на конструкции Миагути — Пренеля. Схема конструкции Миагути — Пренеля: h, m — вектора, поступающие на вход функции сжатия; $g(h, m)$ — результат функции сжатия; E — блочный шифр с длиной блока и ключа 512 бит. В качестве блочного шифра в хеш-функции ГОСТ Р 34.11-2012 взят XSPL-шифр. Этот шифр состоит из следующих преобразований:

- сложение по модулю 2;
- преобразование замены или подстановки. Обозначается S-преобразование;
- преобразование перестановки. Обозначается P-преобразование;
- линейное преобразование. Обозначается L-преобразование.

Преобразования, используемые в новой хеш-функции, должны быть хорошо изучены. Поэтому в блочном шифре E используются преобразования X, S, P, L, которые хорошо изучены.

Важным параметром блочного шифра является то, как выбирается ключ, который будет использовать на каждом раунде. В блочном шифре, используемом в ГОСТ Р 34.11-2012, ключи $K1, K2, \ldots, K13$ для каждого из 13 раундов генерируются с помощью самой функции шифрования.

$C1, C2, \ldots, C12$ — итерационные константы, которые являются 512 битовыми векторам. Их значения указаны в соответствующем разделе стандарта.

4.1.3 Описание

В основу хеш-функции положена итерационная конструкция Меркла — Дамгора с использованием MD-усиления. Под MD-усилением понимается дополнение неполного блока при вычислении хеш-функции до полного путём добавления вектора $(0 \ldots 01)$ такой длины, чтобы получился полный блок. Из дополнительных элементов нужно отметить следующие:

- завершающее преобразование, которое заключается в том, что функция сжатия применяется к контрольной сумме всех блоков сообщения по модулю 2^{512};
- при вычислении хеш-кода на каждой итерации применяются разные функции сжатия. Можно сказать, что функция сжатия зависит от номера итерации.

Описанные выше решения позволяют противостоять многим известным атакам.

Кратко описание хеш-функции ГОСТ Р 34.11-2012 можно представить следующим образом. На вход хеш-функции подается сообщение произвольного размера. Далее сообщение разбивается на блоки по 512 бит, если размер сообщения не кратен 512, то оно дополняется необходимым количеством бит. Потом итерационно используется функция сжатия, в результате действия которой обновляется внутреннее состояние хеш-функции. Также вычисляется контрольная сумма блоков и число обработанных бит. Когда обработаны все блоки исходного сообщения, производятся ещё два вычисления, которые завершают вычисление хеш-функции:

- обработка функцией сжатия блока с общей длиной сообщения.
- обработка функцией сжатия блока с контрольной суммой.

4.1.4 Криптостойкость

Криптоанализ старого стандарта выявил некоторые его слабые стороны с теоретической точки зрения. Так в одной из работ, посвящённых криптоанализу ГОСТ Р 34.11-94, было выявлено, что сложность алгоритма построения прообраза оценивается в 2^{192} вычислений функций сжатия, коллизии 2^{105}, что меньше «универсальных» оценок, которые для ГОСТ Р 34.11-94 равны 2^{256} и 2^{128}. Хотя по состоянию на 2013 год нет большого числа работ, посвящённых криптостойкости новой хеш-функции, исходя из конструкции новой хеш-функции, можно сделать некоторые выводы о её криптостойкости и предположить, что её криптостойкость будет выше, чем у ГОСТ Р 34.11-94:

1. в разделе «Описание» из схемы видно, что все блоки сообщения суммируются по модулю 2^{512} и уже результат суммирования всех блоков подается на вход завершающего этапа (stage3). Благодаря тому, что здесь суммирование — это не побитовое сложение, получается защита от следующих атак:
   - построение мультиколлизий;
   - удлинение прообраза;
   - дифференциальный криптоанализ;
2. в функции сжатия используется конструкция Миагути — Пренели, это обеспечивает защиту от атаки, основанную на фиксированных точках, так как для конструкции Миагути — Пренели не найдено лёгких способов для поиска фиксированных точек;
3. на каждой итерации при вычислении хеш-кода используются различные константы. Это затрудняет атаки на основе связанных и разностных связанных ключей, атаки скольжения и отражения.

4.1.5 Быстродействие

На сайте, посвящённом VI Международной конференции «Параллельные вычисления и задачи управления» (PACO'2012), представлена статья П. А. Лебедева «Сравнение старого и нового стандартов РФ на криптографическую хеш-функцию на ЦП и графических процессорах NVIDIA», в которой проводится сравнение быстродействия семейства криптографических хеш-функций ГОСТ Р 34.11-94 и ГОСТ Р 34.11-2012 на процессорах архитектуры $x86\_64$ и видеокартах NVIDIA с поддержкой технологии CUDA.

Для сравнения быстродействия на процессоре архитектуры $x86\_64$ были взяты 4 разных реализации хеш-функций:

1. реализация ГОСТ Р 34.11-1994 из криптографического пакета OpenSSL (версия 1.0.1c) с открытым исходным кодом. В этой реализации нет алгоритмических и программных оптимизаций;
2. реализация ГОСТ Р 34.11-1994 в программе RHash (версия 1.2.9). В этой реализации есть алгоритмические и программные оптимизации, в том числе ассемблерные оптимизации;
3. реализация ГОСТ Р 34.11-2012, написанная А. Казимировым;
4. реализации ГОСТ Р 34.11-1994 и ГОСТ Р 34.11-2012, написанные П. А. Лебедевым.

Использовался процессор Intel Core i7-920 CPU на базовой частоте 2,67 ГГц. Результаты производительности:

	ГОСТ Р 34.11-1994		ГОСТ Р 34.11-2012	
Реализация №	МБ/с	Тактов/байт	МБ/с	Тактов/байт
1	18	143	-	-
2	49	52	-	-
3	-	-	38	67
4	64	40	94	27

Сравнение быстродействия старого и нового стандартов хеш-функций на GPU проводилось между реализациями П. А. Лебедева. Использовалась видеокарта NVIDIA GTX 580. Результаты производительности (8192 потока данных по 16 КБ):

ГОСТ Р 34.11-1994		ГОСТ Р 34.11-2012	
МБ/с	Тактов/байт	МБ/с	Тактов/байт
1697	-	608	-

На основании этих результатов сделан вывод, что хеш-функция ГОСТ Р 34.11-2012 может быть в два раза быстрее хеш-функции ГОСТ Р 34.11-94 на современных процессорах, но медленнее на графических картах и системах с ограниченными ресурсами.

4.2 ГОСТ (блочный шифр)

ГОСТ блочный шифр (Магма), определенный в стандарте ГОСТ 28147-89 (RFC 5830), представляет собой советский и российский государственный стандартный симметричный ключевой блочный шифр с размером блока 64 бита. Оригинальный стандарт, опубликованный в 1989 году, не давал шифру никакого названия, но самая последняя редакция стандарта, ГОСТ Р 34.12-2015 (RFC 7801, RFC 8891), указывает, что он может называться Магмой. Хеш-функция ГОСТ основана на этом шифре.

4.2.1 Алгоритм

ГОСТ имеет 64-битный размер блока и длину ключа 256 бит. Его S-боксы могут быть секретными, и они содержат около 354 ($log_2(16!^8)$) бит секретной информации, поэтому эффективный размер ключа может быть увеличен до 610 бит; однако атака с выбранным ключом может восстановить содержимое S-боксов примерно за 2^{32} шифрования.

ГОСТ - это сеть Фейстеля из 32 раундов. Его функция округления очень проста: добавьте 32-битный подраздел по модулю 2^{32}, поместите результат через слой S-ящиков и поверните этот результат влево на 11 бит. Результатом этого является вывод функции round. На соседней схеме одна строка представляет 32 бита.

Подразделы выбираются в заранее заданном порядке. Расписание ключей очень простое: разбейте 256-битный ключ на восемь 32-битных подразделов, и каждый подраздел используется в алгоритме четыре раза; первые 24 раунда используют ключевые слова по порядку, последние 8 раундов используют их в обратном порядке.

S-боксы принимают четырехразрядный вход и выдают четырехразрядный выход. Подстановка S-box в функции round состоит из восьми S-ящиков 4×4. S-боксы зависят от реализации, поэтому стороны, которые хотят защитить свои коммуникации с помощью ГОСТа, должны использовать одни и те же S-боксы. Для дополнительной безопасности S-боксы могут храниться в секрете. В исходном стандарте, где был указан ГОСТ, никаких S-боксов не давалось, но их надо было как-то поставлять. Это привело к предположению, что организациям, за которыми правительство хотело шпионить, давали слабые S-боксы. Один из производителей ГОСТ-чипов сообщил, что сам генерировал S-боксы с помощью генератора псевдослучайных чисел.

4.2.2 Криптоанализ ГОСТ

Новейший криптоанализ ГОСТ показывает, что он безопасен в теоретическом смысле. На практике сложность данных и памяти лучших опубликованных атак достигла уровня практической, в то время как временная сложность даже самой лучшей атаки по-прежнему составляет 2^{192} при наличии данных 2^{64}.

С 2007 года было разработано несколько атак против сокращенных реализаций ГОСТ и/или слабых ключей.

В 2011 году несколько авторов обнаружили более существенные недостатки в ГОСТЕ, впервые сумев атаковать полный 32-раундовый ГОСТ произвольными ключами. Николя Куртуа даже назвал его "глубоко ошибочным шифром". Первоначальные атаки смогли снизить временную сложность с 2^{256} до 2^{228} за счет огромных требований к памяти, и вскоре они были улучшены до 2^{178} временной сложности (за счет 2^{70} памяти и 2^{64} данных).

В декабре 2012 года Куртуа, Гавинеки и Сонг улучшили атаки на ГОСТ, вычислив только 2^{101} ГОСТ-раундов. Isobe уже опубликовала одну ключевую атаку на полный ГОСТ-шифр, которую Динур, Дункельман и Шамир улучшили, достигнув 2^{224} временной сложности для 2^{32} данные и память 2^{36}, и временная сложность 2^{192} для данных 2^{64}.

Поскольку атаки уменьшают ожидаемую силу с 2^{256} (длина ключа) примерно до 2^{178}, шифр можно считать взломанным. Однако для любого блочного шифра с размером блока n бит максимальный объем открытого текста, который может быть зашифрован до того, как произойдет повторное шифрование, составляет $2^{n/2}$ блока из-за парадокса дня рождения, и ни одна из вышеупомянутых атак не требует менее 2^{32} данных.

4.3 Кузнечик (шифр)

«Кузнечик» (англ. Kuznyechik или англ. Kuznechik) — симметричный алгоритм блочного шифрования с размером блока 128 бит и длиной ключа 256 бит, использующий для генерации раундовых ключей SP-сеть.

4.3.1 Общие сведения

Данный шифр утверждён (наряду с блочным шифром «Магма») в качестве стандарта в ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры» приказом от 19 июня 2015 года № 749-ст. Стандарт вступил в действие с 1 января 2016 года. Шифр разработан Центром защиты информации и специальной связи ФСБ России с участием АО «Информационные технологии и коммуникационные системы» (АО «ИнфоТеКС»). Внесён Техническим комитетом по стандартизации ТК 26 «Криптографическая защита информации».

Протоколом № 54 от 29 ноября 2018 года, на основе ГОСТ Р 34.12-2015, Межгосударственным советом по метрологии, стандартизации и сертификации был принят межгосударственный стандарт ГОСТ 34.12-2018. Приказом Федерального агентства по техническому регулированию и метрологии от 4 декабря 2018 года № 1061-ст стандарт ГОСТ 34.12-2018 введен в действие в качестве национального стандарта Российской Федерации с 1 июня 2019 года.

4.3.2 Криптостойкость

Ожидается, что новый блочный шифр «Кузнечик» будет устойчив ко всем видам атак на блочные шифры.

На конференции «CRYPTO 2015» Алекс Бирюков, Лео Перрин и Алексей Удовенко представили доклад, в котором говорится о том, что несмотря на утверждения разработчиков, значения S-блока шифра Кузнечик и хеш-функции Стрибог не являются (псевдо)случайными числами, а сгенерированы на основе скрытого алгоритма, который им удалось восстановить методами обратного проектирования. Позднее Лео Перрин и Алексей Удовенко опубликовали два альтернативных алгоритма генерации S-блока и доказали его связь с S-блоком белорусского шифра BelT. В этом исследовании авторы также утверждают, что, хотя причины использования такой структуры остаются неясны, использование скрытых алгоритмов для генерации S-блоков противоречит принципу отсутствия козыря в рукаве, который мог бы служить доказательством отсутствия специально заложенных уязвимостей в дизайне алгоритма.

Riham AlTawy и Amr M. Youssef описали атаку «встречи посередине» на 5 раундов шифра Кузнечик, имеющую вычислительную сложность 2^{140} и требующую 2^{153} памяти и 2^{113} данных.

5 Код

Для своего программного обеспечения мы выбрали отечественный алгоритм подсчета хэш-функции «Стрибог». За основу своего кода мы взяли готовую реализацию на языке Python с github. Наш код может вычислять контрольные суммы с введенного сообщения или с файла. Полученный результат он будет сохранять в созданный вами файл.

6 Список литературы

1. Книга "Основы криптографии"(А. П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А. В. Черемушкин)
2. https://github.com/Desa0197/cripto/tree/4b8b6a529f4734cda34c5bd85b45a5770789cca9/lab6
3. https://intuit.ru/studies/curriculums/18282/courses/28/lecture/20424?page=3
4. https://ru.wikipedia.org/wiki/Стрибог
5. https://en.wikipedia.org/wiki/GOST