Данный материал является фрагментом электронного учебника
по информационной безопасности и может обновляться.
При цитировании рекомендуется использовать ссылку:

Амелин Р. В. Информационная безопасность. Конспект лекций // Амелин Р.В. Лаборатория преподавателя [Электронный ресурс]. URL: rv-lab.ru (2017).

Лекция 3. Обеспечение целостности данных на основе алгоритмов симметричного шифрования

Целостность информации означает, что изменять ее могут только лица, обладающие соответствующими полномочиями. Задача обеспечения целостности данных заключается в том, чтобы обнаружить любую несанкционированную модификацию данных при их передаче или хранении. Другими словами, *получатель сообщения может с уверенностью утверждать, что оно доставлено именно в том виде, в котором было отправлено, либо констатировать факт его искажения.*

Алгоритмы шифрования, рассмотренные в предыдущем разделе, не предназначены для обеспечения целостности. Предположим, Алиса отправляет Бобу (банковской программе) распоряжение перевести на счет Евы 1000$ и зашифровывает это сообщения с помощью самого надежного шифра в мире – одноразового блокнота с однократно используемым ключом. Ева перехватывает сообщение и хочет изменить его таким образом, чтобы на ее счет была переведена другая сумма, например, 9000$. На первый взгляд, она не может сделать этого, не зная ключа: ведь без ключа ей не удастся ни расшифровать исходное сообщение, ни зашифровать свое собственное сообщение, чтобы оно было принято банком. Однако на самом деле все, что нужно Еве – это определить, в каком именно месте сообщения содержится сумма перевода (если Боб – стандартизированная программа, придерживающаяся определенного протокола, включающего структуру сообщений). Пусть m_1 – участок зашифрованного сообщения m, в котором содержится зашифрованная сумма. Т.е. $m = m_0 \| m_1 \| m_2$. Чтобы изменить эту сумму и получить на выходе *корректное зашифрованное сообщение*, которое примет банк, Еве нужно проделать простую операцию:

$m_1` = m_1 \oplus 1000_2 \oplus 9000_2$

$m` = m_0 \| m_1` \| m_2$

Проблема в том, что большинство шифров разрабатывались, чтобы обеспечивать конфиденциальность информации, а не ее целостность. Классическим средством обеспечения целостности служат коды аутентификации сообщений (MAC).

3.1. Коды аутентификации сообщений (MAC)

Любые средства обеспечения целостности используют одну и ту же идею. На основе сообщения по некоторому алгоритму вычисляется величина (последовательность бит), которую в зависимости от алгоритма называют по-разному (контрольной суммой, электронной подписью, хэшем и т.д.), мы будем использовать общее название – *тег*. Этот тег отправляется вместе с сообщением. Получатель сообщения выполняет процедуру *проверки*. Процедура проверки сообщает, было ли сообщение изменено в процессе передачи.

Коды аутентификации сообщений (Message Authentication Code – MAC) для генерации тега и проверки его валидности используют секретный ключ, известный отправителю и получателю.

Таким образом, MAC I – это набор из двух функций, одна из которых называется *функцией генерации тега* S, а вторая – *функцией проверки* V. Первая функция принимает на входе сообщение m и секретный ключ k и возвращает тег t. Вторая функция принимает на входе сообщение, секретный ключ и тег и возвращает *true*, если тег прошел проверку на подлинность и *false* в противном случае. Если ввести следующие обозначения:

X – множество всевозможных сообщений,

K – множество всевозможных ключей,

T – множество всевозможных тегов,

то MAC можно задать следующими соотношениями:

$$I = (S, V);$$

$$S: K \times X \rightarrow T;$$

$$V: K \times X \times T \rightarrow \{true, false\};$$

$$V(k, m, S(k, m)) = true;$$

Последнее означает, что если функция получает для проверки сообщение m, ключ k и тег, сгенерированный для этой пары сообщение/ключ, то, очевидно, она должна вернуть значение true (см. рис. 3.1).

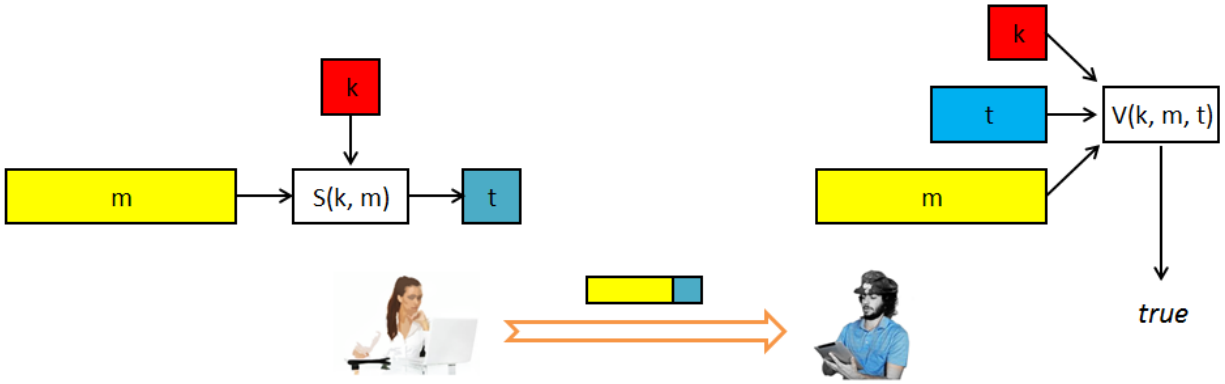

Рис. 3.1. Схема использования MAC

Для определения надежности MAC предполагается, что противник обладает следующими возможностями:

1) Противнику известны детали реализации функций S и V.

2) Противник может заставить отправителя сгенерировать теги для любого количества специально подобранных им сообщений (разновидность атаки с избранным открытым текстом). Другими словами, он имеет q пар (m, t), созданных с использованием одного и того же ключа k.

Целью противника является подделка. *Подделкой* называется такая пара (m', t'), что $V(k, m', t') = true$ (пара принимается как валидная), и она не входит в число q пар, наблюдаемых противником ранее – то есть, создана без знания ключа k (рис. 3.2.). MAC будет считаться *надежным*, если для любого вычислительно эффективного алгоритма вероятность генерации такой пары ничтожно мала.

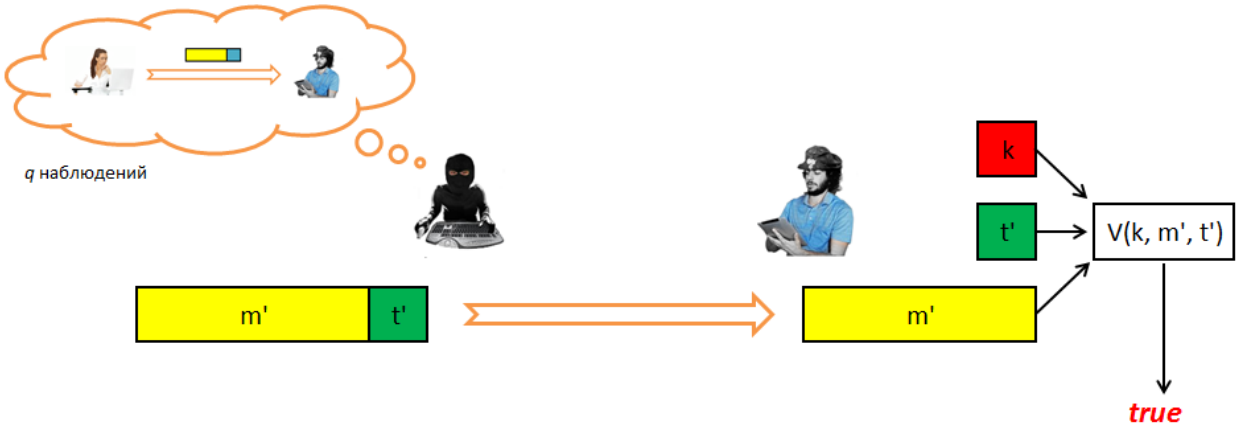

Рис. 3.2. Подделка MAC

Способы построения MAC

1. Если у нас есть надежный алгоритм шифрования (например, блочный шифр AES), то мы можем использовать его непосредственно в качестве функции генерации тега. Другими словами, в качестве тега будет выступать само сообщение, зашифрованное секретным ключом K с помощью надежного алгоритма шифрования E.

$$S(k, m) = E(k, m)$$

Функция проверки подлинности будет работать следующим образом:

$$V(k, m, t) = (t == E(k, m))$$

Если результат шифрования полученного сообщения m с ключом k совпадает с полученным тегом t, то пара $<m, t>$ принимается. С высокой долей вероятности сообщение подлинно, то есть, создано вторым владельцем секретного ключа. Никто не смог бы вычислить $E(m, k)$, не зная k, даже если бы он наблюдал перед этим множество других пар $<m, E(m, k)>$ – при условии, что шифр E надежный.

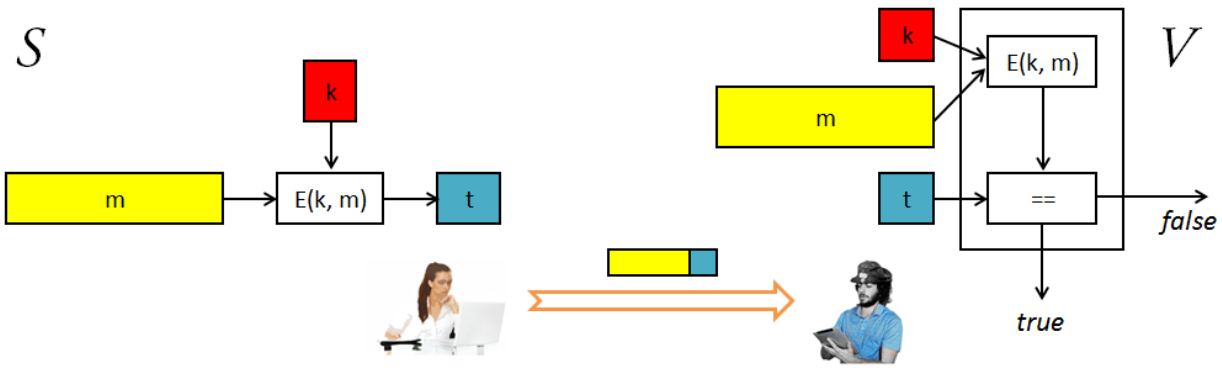

Рис. 3.3. MAC на основе симметричного алгоритма шифрования

Недостаток данного способа состоит в том, что блочный шифр оперирует блоками небольшой длины. Удобно таким образом вычислить MAC для короткого 128-битного со-

общения, но в реальной жизни приходится подписывать сообщения произвольной длины. Если же воспользоваться поточным шифром или одним из режимов шифрования, то мы получим очень длинный тег – такой же, как само сообщение. Очевидно, удваивать каждый раз размер пересылаемого текста не оптимально.

2. Для шифрования сообщений произвольной длины можно построить более сложную конструкцию, базовым элементом которой снова будет являться симметричный алгоритм шифрования E. Рассмотрим две наиболее популярные конструкции:

CBC-MAC. Похож на рассмотренный ранее режим сцепления шифрованных блоков. Основная разница заключается в том, что все блоки шифртекста кроме последнего (в создании которого приняли участие все остальные) отбрасываются, а этот последний блок проходит дополнительный этап шифрования с отдельным ключом k_1 (т.е. ключ CBC-MAC в два раза больше ключа алгоритма шифрования E и состоит из двух подключей k_0 и k_1). Схема вычисления MAC наглядно представлена на рис. 3.4.

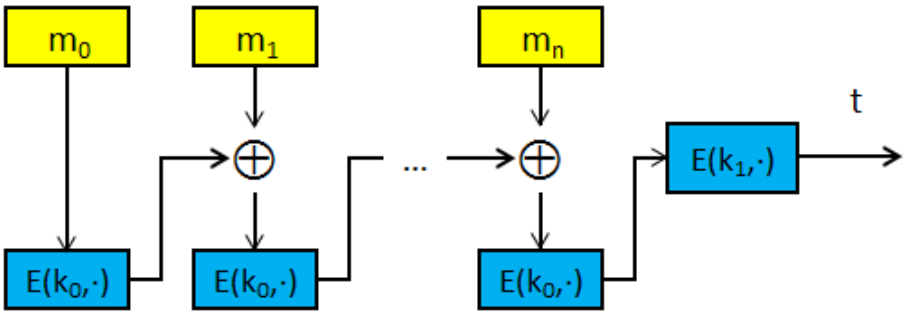

Рис. 3.4. CBC-MAC

NMAC. Основная идея та же самая: каждый предыдущий блок текста после шифрования воздействует на следующий. Только теперь они не складываются друг с другом, а связываются по ключу: результат каждого предыдущего шага шифрования служит ключом для следующего (см. рис. 3.5.). Очевидно, что для этого размер выходного блока алгоритма E должен совпадать с размером ключа этого алгоритма.

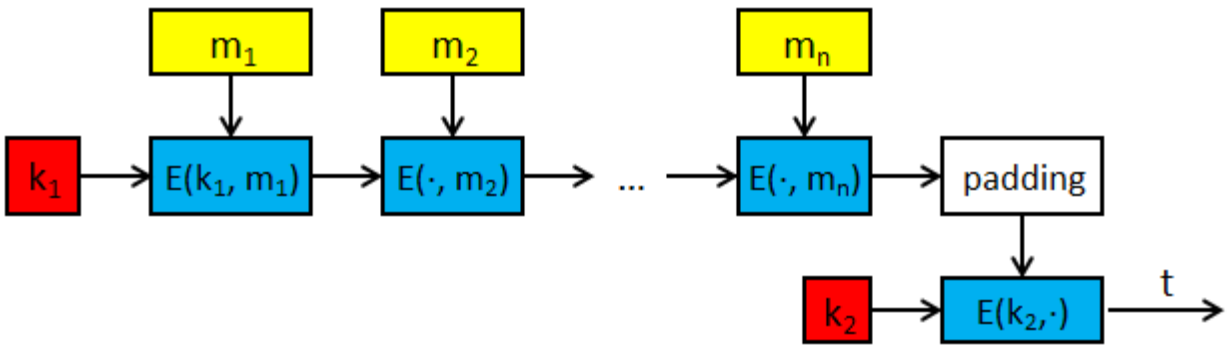

Рис. 3.5. NMAC

Заметим, что во всех приведенных схемах вычисление MAC производится только в прямом порядке. Когда получатель проверяет валидность пары <сообщение, тег>, он вычисляет MAC для сообщения и сравнивает его с пришедшим тегом. Если они совпадают, пара принимается. Функция дешифрования *D* вообще не используется. Следовательно, в общем случае мы можем заменить алгоритм *E* на *псевдослучайную функцию F* (главное отличие заключается в том, что *F* не обязана быть обратимой, и размер выхода может быть меньше размера входа). Именно поэтому в схеме NMAC появляется этап, называемый *дополнением* (padding) – если выход функции F меньше, чем ключ, то он дополняется до нужного размера определенной последовательностью бит, после чего применяется последний раунд шифрования с отдельным ключом k_2.

Заметим, что последний раунд шифрования с другим ключом k_2 является обязательным для защиты от подделки. Предположим, этого этапа бы не было. Тогда, перехватив пару <m, t>, противник смог бы сгенерировать MAC для любого своего сообщения, начинающегося с m, поскольку будет наблюдаться следующее соотношение:

$$S(k, m \,||\, m_1) = S(S(k, m), m_1).$$

Приведенные схемы наглядно демонстрируют процедуру построения MAC для сообщения произвольной длины в случае, когда длина сообщения кратна размеру блока. Если это не так, то последний блок сообщения будет короче, чем принимает на входе алгоритм шифрования E (или псевдослучайная функция F). В этом случае сообщение *дополняют* определенной последовательностью бит до требуемого размера. Главное при этом, чтобы два разных сообщения после дополнения не стали одинаковыми. В этом случае их MAC будут равны, чем не замедлит воспользоваться злоумышленник.

Предположим, мы решили дополнить сообщение до нужной длины нулевыми битами. Тогда, перехватив MAC сообщения «001», противник будет знать MAC для сообщений «0010», «00100», «001000» и т.д. – ведь они будут одинаковыми. А наличие такой возможности делает MAC небезопасным.

В большинстве современных стандартов используется дополнение вида «1000…0». Т.е. в конец сообщения дописывается бит «1», а затем – нужное количество нулевых битов до кратной длины. Если же длина сообщения изначально кратна длине блока, то в конец сообщения добавляется целый блок вида «1000…0». Тогда любые два различных сообщения после дополнения не будут совпадать.

3.2. Хэш-функции

В общем случае *хэш-функция* H: X $\rightarrow$ T преобразует сообщение (битовую последовательность) произвольной длины в выходную строку фиксированной длины, называемую хэшем. Как правило, размер выхода намного меньше размера входа (результат хэш-функции порядка нескольких тысяч бит, в то время как исходное сообщение может занимать мегабайты). Ее еще называют *функцией сжатия*. Хэш-функция не зависит от ключа и для одного и того же сообщения всегда генерирует один и тот же хэш.

Ситуация, когда для двух разных сообщений $m_1 \neq m_2$ их хэши совпадают, т.е. $H(m_1) = H(m_2)$, называется *коллизией*.

Криптографию интересуют особые хэш-функции, обладающие рядом специальных свойств:

1. *Односторонность (необратимость)*. Для любого h должно быть практически невозможно вычислить или подобрать такое x, что $H(x) = h$.
2. *Стойкость к коллизиям первого рода*. Для любого сообщения x должно быть практически невозможно вычислить или подобрать другое сообщение y, такое что $H(x) = H(y)$.
3. *Стойкость к коллизиям второго рода*. Должно быть практически невозможно вычислить или подобрать любую пару различных сообщений x и y для которых $H(x) = H(y)$.

Такие хэш-функции называются *криптостойкими*. Они играют важную роль при построении различных криптографических конструкций.

Структура Меркла-Дамгарда

Большинство современных хэш-функций строятся по одной общей схеме, которая называется *структура Меркла-Дамгарда*. Идея заключается в том, чтобы, имея функцию, генерирующую хэш для блока фиксированной длины, расширить ее на сообщение произвольной длины.

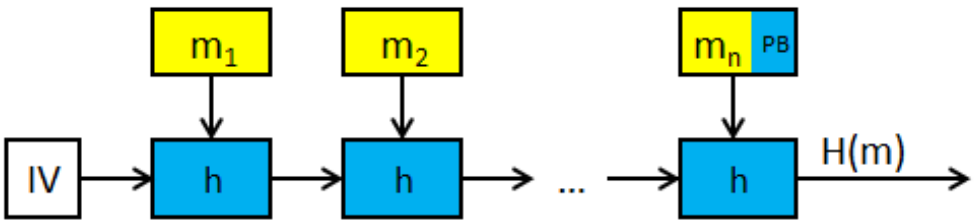

Рис. 3.6. Структура Меркла-Дамгарда

Как видно из рисунка 3.6, идея та же самая, что и в алгоритме NMAC. Главное отличие заключается в том, что вместо секретного ключа на вход конструкции подается общеизвестный и постоянный для конкретного алгоритма инициализационный вектор IV, поскольку вычислить хэш для заданного сообщения может кто угодно.

Функция h, на основе которой строится хэш-функция, называется *функцией сжатия*. Она строит хэш для блока фиксированной длины. Доказана теорема, что если функция сжатия h обладает стойкостью к коллизиям, то тем же свойством обладает функция H, построенная по схеме Меркла-Дамгарда.

Чтобы сообщение разбивалось на целое число блоков, его дополняют. Дополнение PB обычно имеет вид 10000…0 || длина сообщения (последние 64 бита).

Структура Меркла-Дамгарда дает хороший *лавинный эффект*: изменение малого количества бит в сообщении ведет к изменению большого количества бит в хэше.

Очевидно, что функция сжатия может быть построена на основе блочного шифра, хотя использовать в качестве *h* просто алгоритм шифрования *E* не рекомендуется – алгоритмы шифрования разрабатывались для обеспечения конфиденциальности, а не сопротивляемости коллизиям. Криптографически стойкой является функция сжатия ***Дэвиса-Мейера***, имеющая следующую простую формулу:

$$h(H, m) = E(m, H) \oplus H$$

Обратите внимание, в качестве ключа используется блок хэшируемого сообщения, а шифруется хэш, полученный на предыдущем шаге (соответственно IV на шаге 1).

Наглядно функция Дэвиса-Мейера отражена на рис. 3.7.

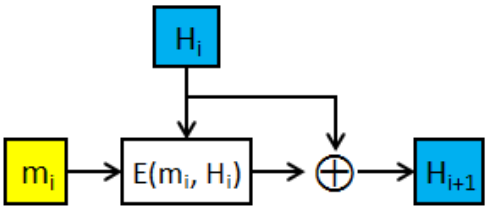

Рис. 3.7. Функция Дэвиса-Мейера

Существует 12 вариантов подобных функций, например, $h(H, m) = E(H \oplus m, m) \oplus m$ и т.д. Существует также множество небезопасных вариантов, таких как $h(H, m) = E(m, H) \oplus m$. Почему первый вариант безопасный, а второй нет – можете исследовать самостоятельно.

Доказано, что для идеального шифра E нахождение коллизии функции Дэвиса-Мейера потребует $O(2^{n/2})$ вычислений, где n – длина хэша. Это наилучший из возможных результатов, поскольку именно таких усилий требует подбор коллизии методом грубой силы – на основе *парадокса задачи о днях рожденья***.**

Алгоритмы хэширования

На основе структуры Меркла-Дамгарда были построены известные алгоритмы хэширования семейства MD, последний из которых, **MD5** широко использовался до конца 2000 годов. В настоящий момент считается устаревшим и криптографически взломанным.

Алгоритм шифрования **SHA-256** построен на основе уже рассмотренных составляющих:

- структура Меркла-Дамгарда,
- функция Дэвиса-Мейера,
- в качестве блочного шифра алгоритм SHACAL-2 (интересно, что этот алгоритм шифрования был разработан на основе SHA-256, а не наоборот).

Использование хэш-функций для защиты паролей

Системы, использующие авторизацию на основе пароля, должны хранить эти пароли где-то у себя, чтобы было с чем сравнивать. Рассмотрим три способа организации базы паролей.

1) Пароли хранятся как есть, в открытом виде. Если злоумышленник получает доступ к базе паролей, система становится полностью скомпрометирована.

2) Хранятся только хэши паролей. База данных почти бесполезна для злоумышленника. Свойства хэш-функции не позволяет получить по хэшу подходящий пароль. Но можно использовать *атаку по словарю*: злоумышленник вычисляет хэши для нескольких тысяч популярных паролей, после чего ищет в базе совпавшие значения. Если пользователи этой системы сами придумывают себе пароли, совпадения непременно найдутся.

Злоумышленники могут использовать даже специальные таблицы с хэшами известных паролей (с учетом того, что популярных алгоритмов хеширования немного, эти таблицы можно построить заранее и использовать при взломе множества систем). Их называют *радужными таблицами*. В таблицу войдут хэши для коротких паролей (6–7 символов и меньше), для слов естественных языков и популярных символьных комбинаций. Таблицу можно отсортировать по хэшам для максимальной быстроты поиска – с помощью бинарного алгоритма, к примеру.

3) Хранятся хэши паролей с солью. К хэшу пароля добавляется несколько случайных символов (называемых *солью*) и результат хэшируется еще раз, т.е.

$$h = H(H(password) \,||\, salt)$$

Это уже более надежный вариант. Подобрать пароль с помощью таблиц таким методом не получится. Для каждого пароля используется своя соль, поэтому, по сути, радужную таблицу необходимо генерировать для каждого пользователя (т.е. задача становится вычислительно сложной)[1]. Особенно если использовать при этом *медленную функцию хэширования* – да, некоторые алгоритмы хэширования специально делают очень медленными, чтобы максимально затруднить злоумышленнику поиск строки, дающей нужный хэш. Например, алгоритм Bcrypt имеет настраиваемый параметр, влияющий на скорость.

3.3. MAC на основе хэш-функции

Если скомбинировать сообщение с секретным ключом и применить к этой комбинации функцию хэширования, то мы получим MAC, генерирующий тэг фиксированной длины для сообщения произвольной длины. Самый распространенный алгоритм построения MAC на основе хэш-функции называется HMAC. В нем используется следующая формула:

$$S(k, m) = H(k \oplus opad \;||\; H(k \oplus ipad \,||\, m))$$

Здесь opad и ipad – два фиксированных битовых блока, которые накладываются на ключ для обеспечения лучшего *лавинного эффекта*. Напомним, что лавинный эффект яв-

[1] Это не спасет от взлома пароля администратора (если этот пароль короткий или встречается в словаре), но спасет от одновременной атаки на пароли всех пользователей.

ляется признаком хорошего криптографического алгоритма и применительно к MAC означает, что изменение хотя бы одного бита в секретном ключе или сообщении приведет к изменению многих битов тэга.

HMAC – наиболее часто используемый алгоритм в интернет-протоколах.