DOI 10.37882/2223–2966.2020.04.17

ИССЛЕДОВАНИЕ HMAC НА ПРИМЕРЕ МЕТОДА АВТОРИЗАЦИИ ВЫЗОВ-ОТВЕТ

HMAC RESEARCH ON AN EXAMPLE OF AN AUTHORIZATION CALL-ANSWER METHOD

V. Zyuzin
D. Vdovenko
V. Bolshakov
A. Busenkov
A. Krivdin

Summary. This article explores the mechanism for ensuring and verifying message integrity, such as NMAC. All that is in the MAC is the ability to share a secret key to make sure that all messages that they pass through this algorithm can be changed.

This article provides a realistic example of the use of hash functions.

Keywords: message, hash function, hash, byte, key, size, subscriber, attacker, client, server, case, string, algorithm.

Зюзин Владислав Дмитриевич
Московский технический университет связи и информатики
vlados9495@gmail.com
Вдовенко Дмитрий Викторович
Московский технический университет связи и информатики
dmitriy575893@mail.ru
Большаков Василий Николаевич
Московский технический университет связи и информатики
vasyabvn@mail.ru
Бусенков Алексей Александрович
Московский технический университет связи и информатики
albus.rank@yandex.ru
Кривдин Александр Дмитриевич
Московский технический университет связи и информатики
krivdin.a@mail.ru

Аннотация. В данной статье исследуется механизм для обеспечения и проверки целостности сообщений, такой, HMAC. Цели и предназначения HMAC те же самые, что у MAC, дать возможность двум абонентам, у которых есть общий секретный ключ, средство убедиться в том, что те сообщения, которые они пропускали через этот алгоритм, никем в процессе передачи или хранения не изменялись.

В данной статье рассматривается реалистичный пример для использования хеш-функций на примере метода авторизации вызов-ответ.

Ключевые слова: сообщение, хеш-функция, хеш, байт, ключ, размер, абонент, злоумышленник, клиент, сервер, случай, строка, алгоритм.

Введение

MAC — это механизм для обеспечения и проверки целостности сообщений.

Если имеется два абонента A и B и они ведут между собой незашифрованную переписку, однако они хотят быть уверены в том, что те сообщения, которые они друг другу передают, никем в процессе передачи не изменяются.

Суть MAC заключается в том, если злоумышленник попытается изменить сообщения, которые абоненты передают друг другу, это сразу должно быть замечено.

Реализаций MAC бывает много, но в данном случае рассмотрим самую простую реализацию.

Для начала необходимо, чтобы у абонентов A и B был заранее оговорён общий секретный ключ. При отправке сообщения абонент A дописывает в конец сообщения общий секретный ключ и при заранее оговорённой хеш-функции находит хеш от новой строки. После этого он дописывает этот хеш в конец первоначального исходного сообщения и получает защищённое с помощью MAC, сообщение и передаёт абоненту B. По пути оно, конечно, перехвачено злоумышленником. Абонент B, получив сообщение, выбирает оттуда всё, кроме хеша, проделывает ту же опера-

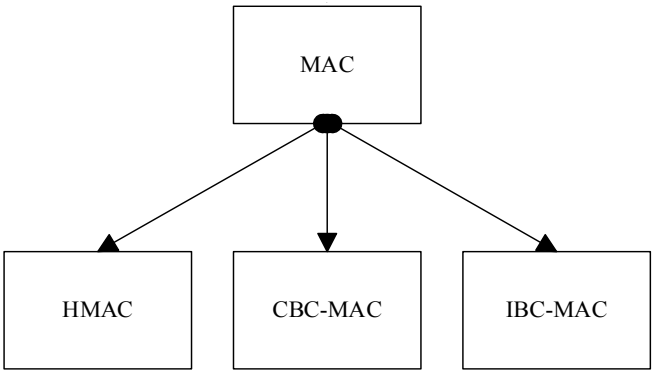

Рис. 1. Разновидности MAC

цию и сравнивает полученные хеши, если они равны, значит сообщение достоверно.

Если по пути злоумышленник решит подменить сообщение и дописать тот же хеш, то при получении его абонентом B, тот получит другой хеш и сравнивая их поймет, что принятое сообщение недостоверно.

Плюс MAC — это простота в реализации.

Минус MAC — нужно договориться о общем секретном ключе так, чтобы злоумышленник о нём не узнал.

Из-за коллизий этот метод перестал быть популярным и ему на смену пришёл HMAC.

Методы. В данной статье, в качестве реалистичного примера для использования HMAC рассматривается метод авторизации вызов-ответ.

Результаты и обсуждение. Существуют HMAC, популярные из которых семейство MD, в частности MD5 и семейство SHA, в частности SHA-1 и SHA-256. Но стандарт SHA-1 на данный момент очень актуальный и его используют практически все протоколы и программы, которые предназначены для защиты информации.

Цели и предназначения HMAC те же самые, что у MAC, дать возможность двум абонентам, у которых есть общий секретный ключ, средство убедиться в том, что те сообщения, которые они пропускали через этот алгоритм, никем в процессе передачи или хранения не изменялись.

Формула HMAC, вне зависимости от того, какая криптографическая функция выбрана, будет иметь вид:

$$HMAC(text, K) =$$

$$= H((K0 + oblock) \,||\, H((K0 + iblock) \,||\, text)) \quad (1)$$

H — выбранная хеш-функция

$size$ — размер блока хеш-функции в байтах

K — ключ (массив байт)

$iblock$ — массив байт длинной $size$, состоящий из байта $0x36$

$K0$ — ключ K, подогнанный до размера $size$

R — размер возвращаемой строки H в байтах

$oblock$ — массив байт длинной $size$, состоящий из байта $0x5c$

$text$ — сообщение в виде массива байт

$+$ — операция XOR

$||$ — склеивание строк

MAC — это более обобщённое понятие, этой аббревиатурой обозначают вставку, которая получилась выше, для проверки целостности сообщений на основе общего секретного ключа, либо алгоритм, который позволяет получить эту вставку для проверки целостности сообщений на основе общего секретного ключа, потому что здесь мы это делаем с помощью хеш-функции, но бывают и другие реализации.

Эту вставку получают с помощью блочных шифров в режиме сцепления блоков.

Из вышесказанного следует, что MAC — это идея наличия такой вставки с общим секретным ключом.

А в данном случае мы разобрали одну из реализаций MAC — HMAC.

Хеш-функция — это особая математическая функция, которая принимает на вход произвольные данные любой длины и возвращает произвольные данные строго определённой длины (причём для любой входной строки существует только одна выходная строка).

Те свойства, которыми обладает хеш-функция, делают её очень полезной. В современном мире хеш-функция присутствует во многих криптографических протоколах и позволяет решать большой список задач (протокол для уста-

новления защищённого соединения, алгоритм проверки целостности сообщений, авторизация на сайтах, ЭЦП).

Значение, которое возвращает хеш-функция (произвольные данные) — называется хешем или хеш-суммой.

Например, имеем сообщение

$$M1 = fdfdjfsbdfndbnd43748HXmz,$$

для этого сообщения считаем хеш-функцию и получаем, например

$$H(M1) = r1 = 0F8457.$$

Имеем еще одно сообщение

$$M2 = aor93vDJS(39r954$$

и снова считаем хеш

$$H(M2) = r2 = HB8J0A.$$

То есть, несмотря на то, что сообщения $M1$ и $M2$ существенно разнятся по содержимому и по длине, хеш имеет статическую длину.

Свойства хеш-функции:

1) Фиксированный размер хеша

2) Непредсказуемость

Не существует алгоритма, который позволил бы получить какую-либо информацию о хеше, кроме как взять и посчитать этот хеш.

3) Необратимость

Любая хеш-функция является односторонней, то есть имея сообщения — легко находится хеш, а имея хеш — нет возможности найти сообщение, которое подавалось на вход.

4) Отсутствие коллиций

Коллизия — это эффект, когда у двух разных сообщений одинаковый хеш.

Например:

$$M1 = HDJdfkkd747345$$
$$M2 = 83h9s948395018dmbkJD$$
$$H(M1) = H(M2) = 96NBCA$$

Такое бывает и у всех существующих хеш-функций на сегодняшний день есть коллизии, и будет у всех хеш-функций, которые будут создаваться.

Это свойство в одном моменте противоречит самому определению хеш-функции.

Нахождение коллизий на сегодняшний день является основным вектором атаки на хеш-функцию.

5) Лавинный эффект

Суть заключается в том, что малейшее изменение в исходном сообщение должно приводить к заметным последствиям в хеше, например

$$M1 = fdfdjfsbfndbnd43748HXmz$$
$$H(M1) = r1 = 0F8457$$
$$M2 = fdfdjfsbfndbnd43748HXmy$$
$$H(M2) = r2 = YF74HA$$

Эти сообщения разнятся всего в одном последнем символе, но даже такое незначительное изменение должно приводить к полному изменению хеша.

Рассмотрим реалистичный пример для использования хеш-функций на примере метода авторизации вызов-ответ.

Имеется клиент, который хочет зарегистрироваться на каком-то почтовом домене (выбрал себе адрес почтового ящика и пароль) и передаёт его серверу. Сервер заносит эту информацию в базу данных. После регистрации клиент снова хочет зайти на свой почтовый ящик. В своей форме указывает логин и пароль. Но пароль в чистом виде отправлять не хочет, так как этот канал связи может быть прослушан. Отсюда возникает вопрос: как доказать серверу, что клиент знает пароль?

Именно здесь «в игру» вступает хеш-функция. Клиент отправляет серверу свой логин, которая перехватывает злоумышленник. После этого сервер генерирует случайную информацию, например, строку R и отправляет её клиенту, по пути её перехватывает злоумышленник. После этого клиент берёт эту одноразовую строку, добавляет её в конец своего пароля и по ранее договорённому алгоритму хеш-функции находит хеш от новой строки. Затем отправляет его серверу, который также по пути перехвачен злоумышленником. Сервер, у которого есть пароль клиента проделывает ту же процедуру и сравнивает полученные результаты. Если результаты равны, то сервер убеждается, что клиент знает пароль, значит ему можно давать доступ к этому почтовому ящику.

$$
\begin{aligned}
K0 = \{&(0x69);(0x6c);(0x6c);(0x75);(0x73);(0x69);(0x6f);(0x6e);\\
&(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);\\
&(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);\\
&(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);\\
&(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);\\
&(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);\\
&(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);\\
&(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00);(0x00)\}
\end{aligned}
$$

Рис. 2

$$
\begin{aligned}
iblock = \{&(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\
&(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\
&(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\
&(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\
&(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\
&(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\
&(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\
&(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36)\}
\end{aligned}
$$

Рис. 3

При этом злоумышленник никак не расшифрует хеш-функцию.

Разберём хеш функцию на конкретном примере.

Возьмём исходный текст и переводим его в байты по ASCII кодировке:

$$Text = "Crypto" = \{(0x43);(0x72);(0x79);$$

$$(0x70);(0x74);(0x6f)\}$$

Выбираем ключ и так же переводим его в байты по ASCII кодировке:

$$K = "illusion" = \{(0x69);(0x6c);(0x6c);(0x75);$$

$$(0x73);(0x69);(0x6f);(0x6e)\}$$

Выбираем хеш-функцию $H = MD5$, размер блока у которой $size = 64$ и размер возвращаемого значения $R = 16$.

И выбираем значения $K0$, исходя из условий:

$$if\,(K.Length == size)$$

$$K0 = K;$$

Если размер ключа K равен размеру блока хеш-функции $size$, тогда $K0$ равен ключу K.

В данном случае $K = 8$, то есть $K \neq size$.

Если размер ключа больше, чем $size$:

$$
\begin{aligned}
&if\,(K.Length \succ size)\\
&\{\\
&byte[]h = H(K);\\
&\}
\end{aligned}
$$

Тогда ключ пропускаем через выбранную хеш-функцию, которая возвращаем в данном случае 16 байт, а остальные недостающие 48 байт забиваются нулями. Но это условие тоже не подходит в данном случае.

В данном случае, $K < size$, остальные недостающие (в данном случае 56 байт) также забиваем нулями и получаем (см. рис.2).

Теперь представляем iblock (см. рис. 3).

Проводим первую операцию $K0 + iblock$ и получаем (см. рис. 4).

$$\begin{aligned} K0 + iblock = \{&(0x5f);(0x5a);(0x5a);(0x43);(0x45);(0x5f);(0x59);(0x58);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36)\} \end{aligned}$$

Рис. 4

$$\begin{aligned} (K0 + iblock) \,||\, text = \{&(0x5f);(0x5a);(0x5a);(0x43);(0x45);(0x5f);(0x59);(0x58);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);\\ &(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36);(0x36)\\ &(0x43);(0x72);(0x79);(0x70);(0x74);(0x6f)\} \end{aligned}$$

Рис. 5

$$\begin{aligned} H((K0 + iblock) \,||\, text) = \{&(0x1a);(0xf1);(0x75);(0x08);(0x39);(0x37);(0x4e);(0x1f);\\ &(0x27);(0xd4);(0xbb);(0x01);(0x12);(0x0c);(0x55);(0xcc)\} \end{aligned}$$

Рис. 6

$$\begin{aligned} oblock = \{&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\ &(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\ &(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\ &(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\ &(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\ &(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\ &(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\ &(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c)\} \end{aligned}$$

Рис. 7

$$
\begin{aligned}
K0 + oblock = \{&(0x35);(0x30);(0x30);(0x29);(0x2f);(0x35);(0x33);(0x32);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c)\}
\end{aligned}
$$

Рис. 8

$$
\begin{aligned}
(K0 + oblock) \,||\, H((K0 + iblock) \,||\, text) = \{&(0x35);(0x30);(0x30);(0x29);(0x2f);(0x35);(0x33);(0x32);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);(0x5c);\\
&(0x1a);(0xf1);(0x75);(0x08);(0x39);(0x37);(0x4e);(0x1f);\\
&(0x27);(0xd4);(0xbb);(0x01);(0x12);(0x0c);(0x55);(0xcc)\}
\end{aligned}
$$

Рис. 9

$$
\begin{aligned}
HMAC("Crypto","illusion") = \{&(0x75;(0x0c);(0xe0);(0x55);(0x6c);(0x22);(0x72);(0x44);\\
&(0xc9);(0x27);(0x2a);(0xe6);(0xad);(0xb7);(0x51);(0x2a)\}
\end{aligned}
$$

Рис. 10

Далее проводим операцию склеивания с текстом, то есть в конец этого множества дописываем текст (рис. 5).

Далее всю эту конструкцию пропускаем через выбранную хеш-функцию $H = MD5$ и получаем 16 байт (рис. 6).

Далее представляем $oblock$ (рис. 7).

Далее делаем операцию $K0 + oblock$ и получаем (рис. 8):

Далее в конец массива дописываем хеш

$$H((K0 + iblock) \,||\, text)$$

и получаем (рис. 9).

И последний раз пропускаем эту конструкцию через ту же хеш-функцию $H = MD5$ и получаем (рис. 10).

В протоколе MTProto используется хеш-функция $SHA\text{-}1$, но для упрощения мы рассмотрели $MD5$. Сравним оба алгоритма:

1) Безопасность.

Дайджест *SHA-1* на 32 бита длиннее, чем у *MD5*, поэтому *SHA-1* — более надёжный алгоритм. Применяя фронтальную атаку, труднее создать случайное сообщение, которое обладает тем же дайджестом. Если необходимо порядок операций, как в SHA-1, а не в MD5. Если использовать фронтальную атаку, сложнее сделать два сообщений, который имеют одинаковый дайджест, если необходим порядок операций в случае SHA-1, а не MD5.

2) Темп.

Оба метода осуществляют суммирование по модулю 2^{32}, так как предназначены для 32-битной архитектуры. *SHA-1* содержит более шагов.

SHA-1–80 шагов, MD5–64 шага

Также реализовывается на буфере размерностью 160 бит, если сравнивать с MD5, который реализовывается на буфере размерностью 128 бит. Следовательно, *MD5* должен работать примерно на 25% быстрее, чем *SHA-1*, если анализировать на одинаковой аппаратуре.

3) Сжатость и проста.

SHA-1 и MD5 элементарны в описании и в осуществлении. Им не нужны большие программы или подстановочные таблицы. Но всё равно, *SHA-1* использует одношаговую структуру, если сравнивать с четырьмя структурами, которые используются в *MD5*. Более того, обрабатывание слов в буфере одна и та же абсолютно для всех шагов *SHA-1*, а в *MD5* структура слов уникальна для каждого из шагов.

4) Архитектуры LE (Little Endian) и BE (Big Endian)

MD5 применяет LE схему с целью интерпретации информации как очерёдности 32-битных слов, *SHA-1* применяет схему BE. Каких-либо положительных сторон в данных подходах не имеется.

Стандарт хеширования SHA-1 на данный момент очень актуальный и его используют практически все протоколы и программы, которые предназначены для защиты информации.

ЛИТЕРАТУРА

1. Блэк У. Интернет протоколы безопасности. Москва: издательство «Питер». 2001. ISBN5–318–00002–9 (ISBN оригинала на английском языке: ISBN0–13–014249–2).
2. RFC2104. Krawczyk H., Bellare M., Canetti R. «HMAC: Keyed-hashing for message authentication». Февраль 1997.
3. Stallings W. Cryptography and network security principles and practices. 2005. ISBN0–13–187316–4.
4. Словарь криптографических терминов / Под ред. Б. А. Погорелова и В. Н. Сачкова. — М.: МЦНМО, 2006. — С. 94. — ISBN5–94057–257-X.
5. J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, «UMAC: Fast and provably secure message authentication», Advances in Cryptology — CRYPTO '99, LNCS vol. 1666, pp. 216–233, Springer-Verlag, 1999.
6. Брюс Шнайер. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. — М.: Триумф, 2002. — ISBN5–89392–055–4.
7. Дональд Кнут. Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming, vol.3. Sorting and Searching. — 2-е издание. — М.: «Вильямс», 2007. — С. 824. — ISBN0–201–89685–0.
8. Никлаус Вирт. Алгоритмы и структуры данных. — М.: «Мир», 1989. — ISBN5–03–001045–9.
9. Никлаус Вирт. Алгоритмы и структуры данных. Новая версия для Оберона. — М.: «ДМК Пресс», 2010. — ISBN978–5–94074–584–6.
10. RFC6151 — Updated Security Considerations for the MD5 Message-Digest and the HMAC–MD5 Algorithms". Internet Engineering Task Force. March 2011. Retrieved 15 June 2015.

© Зюзин Владислав Дмитриевич (vlados9495@gmail.com), Вдовенко Дмитрий Викторович (dmitriy575893@mail.ru),
Большаков Василий Николаевич (vasyabvn@mail.ru),
Бусенков Алексей Александрович (albus.rank@yandex.ru), Кривдин Александр Дмитриевич (krivdin.a@mail.ru).
Журнал «Современная наука: актуальные проблемы теории и практики»