Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Криптографические методы обеспечения информационной безопасности»

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6

«Модель протокола защищенного соединения»

Выполнил:

Полевцов Артем Сергеевич, студент группы N34511

(подпись)

Проверил:

Волков Александр Григорьевич, инженер ФБИТ

(отметка о выполнении)

(подпись)

Санкт-Петербург
2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель: изучить подходы к применению криптопримитивов в рамках протоколов для защищенных соединений.

Задачи практической работы

1. Выполнить визуализацию 1 раунда алгоритма хэширования MD5;
2. Отразить в отчете алгоритм для генерации кодов аутентификации сообщений;
3. Протестировать возможности хэширования файлов и сообщений с помощью openssl dgst;
4. Генерация Hash based MAC.

1. МОДЕЛЬ ПРОТОКОЛА ЗАЩИЩЕННОГО СОЕДИНЕНИЯ

1.1 Ход работы

1.1.1 Визуализация 1 раунда алгоритма хэширования md5

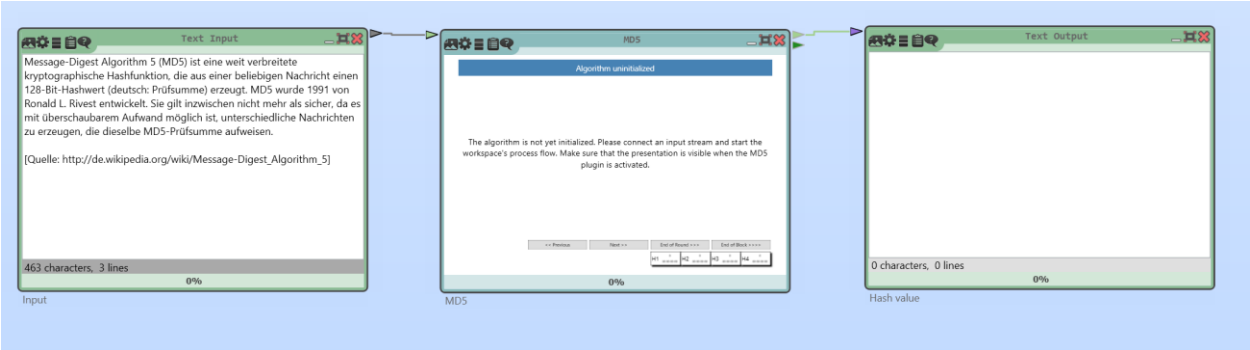

Рисунок 1 – Алгоритм MD5

MD5 (Message Digest Algorithm 5) - это криптографический алгоритм хеширования, который принимает на вход сообщение произвольной длины и выдаёт фиксированный 128-битный хеш-код, обычно представляемый в виде 32-символьной шестнадцатеричной строки.

H1	1732584193 01 23 45 67	H2	4023233417 89 AB CD EF	H3	2562383102 FE DC BA 98	H4	271733878 76 54 32 10

Рисунок 2 – Переменные-аккумуляторы

На первом этапе первого раунда происходит инициализация внутреннего состояния (часто обозначаемого как H1-H4 – это переменные-аккумуляторы, меняющиеся после выполнения функции сжатия) предварительными значениями, которые представляют из себя фиксированные константы. Эти значения образуют хеш-значение, которое будет обновляться по мере прохождения данных через алгоритм. И в конце выполнения хеш-функции данные переменные будут содержать ответ.

Далее осуществляется процесс считывания данных. Функция сжатия MD5 обрабатывает данные блоками размером 64 байта. Ее задача заключается в том, чтобы прочитать достаточное количество данных для последующего вызова функции сжатия. Таким образом, она продолжает считывать данные до тех пор, пока не наберется 64 байта.

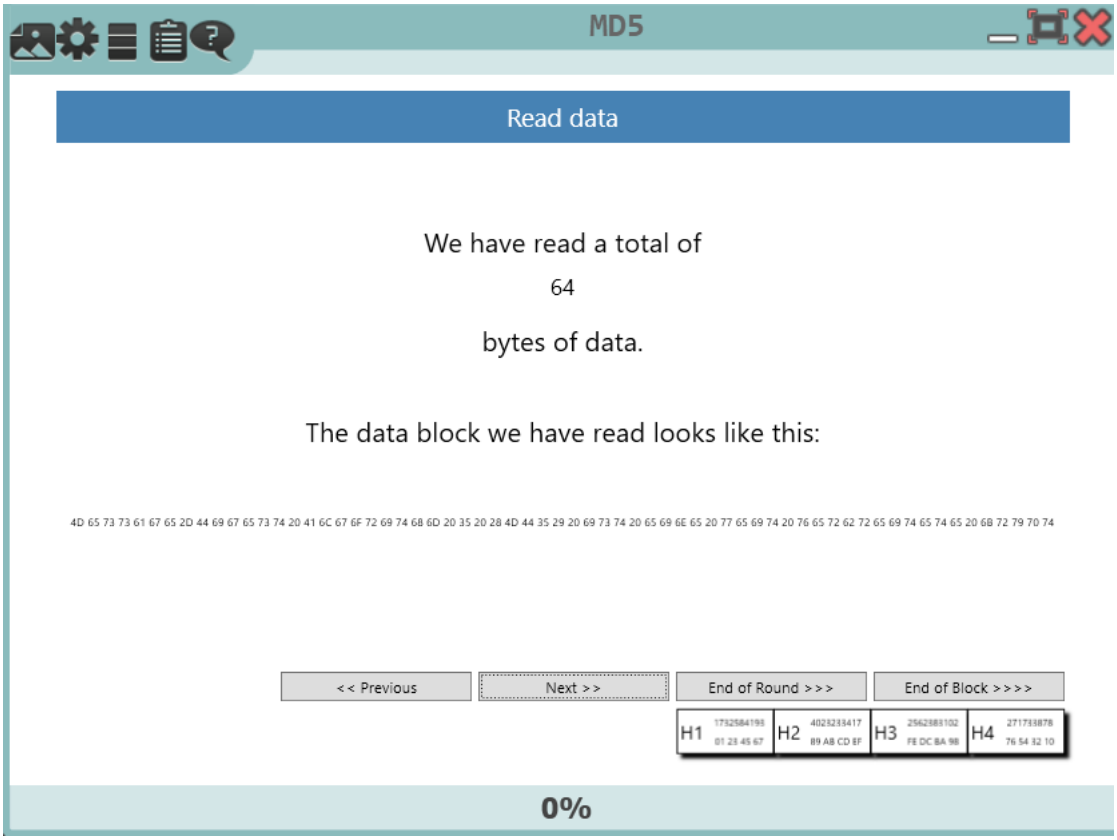

Рисунок 3 – Процесс чтения данных

Затем активируется функция сжатия для обработанного блока данных. В процессе сжатия используются четыре временные переменные: A, B, C и D, которые инициализируются значениями H1-H4.

Рисунок 4 – Инициализация переменных

Полученный на вход блок данных делится на 16 блоков, каждый по 32 бита.

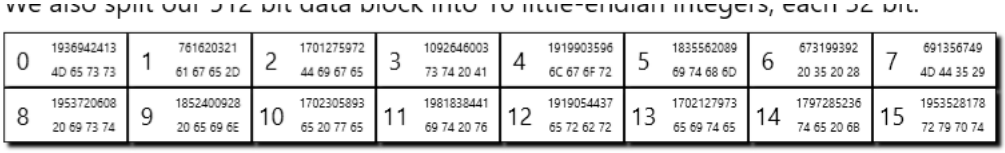

Рисунок 5 – Разделение входного потока данных на блоки

Затем начинается выполнение первого раунда сжатия, и общее количество таких раундов составляет 4. Каждый раунд включает в себя 16 шагов, которые влияют на временные переменные A-D. Для этих манипуляций используются нелинейные внутренние раундовые функции с обозначениями F, G, H и I. Эти функции являются ключевой частью вычислений, выполняемых на каждом этапе раунда. В данном случае в первом раунде используется функция F.

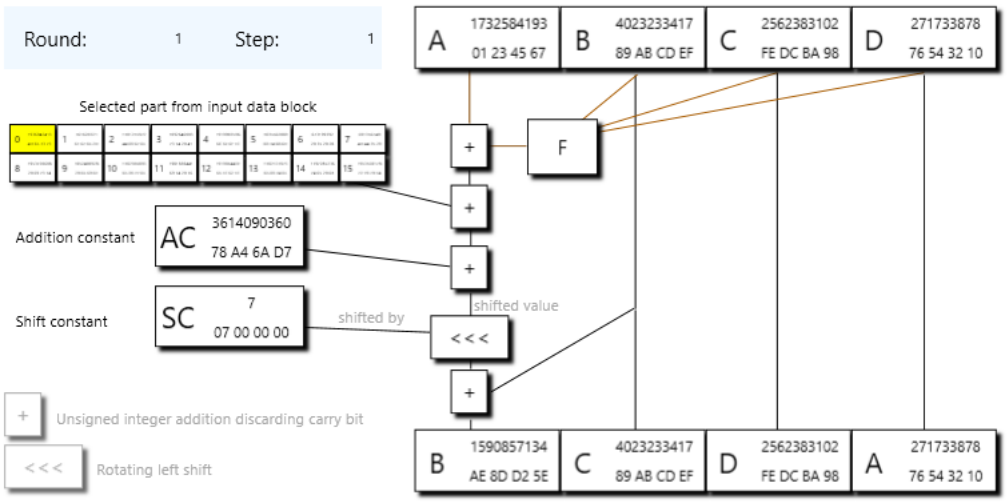

Рисунок 6 – Начало первого раунда

И в итоге выполнения 16 схожих шагов мы получаем результат.

A	171893504 00 E3 3E 0A	B	2541783476 B4 89 80 97	C	1993218222 AE 18 CE 76	D	120362304 40 95 2C 07

Рисунок 7 – Результат первого раунда

По завершении последнего раунда функции сжатия для получения окончательного результата выполняется операция сложения временных переменных A-D с соответствующими переменными-аккумуляторами.

H1	1732584193 01 23 45 67	H2	4023233417 89 AB CD EF	H3	2562383102 FE DC BA 98	H4	271733878 76 54 32 10
	+						
A	4247687520 60 91 2E FD	B	1833926389 F5 7E 4F 6D	C	2407530711 D7 00 80 8F	D	2080500507 1B EB 01 7C
	=						
H1	1685304417 61 B4 73 64	H2	1562192510 7E 2A 1D 5D	H3	674946517 D5 DD 3A 28	H4	2352234385 91 3F 34 8C

Рисунок 8 – Хеш

1.1.2 Визуализация алгоритма для генерации кодов аутентификации сообщений

HMAC (Hash-based Message Authentication Code) представляет собой криптографическую конструкцию, используемую для создания кода аутентификации сообщения на основе хеш-функции. Эта конструкция обеспечивает сочетание ключа и

данных сообщения для создания фиксированной длины кода, который может быть использован для проверки целостности и подлинности сообщения.

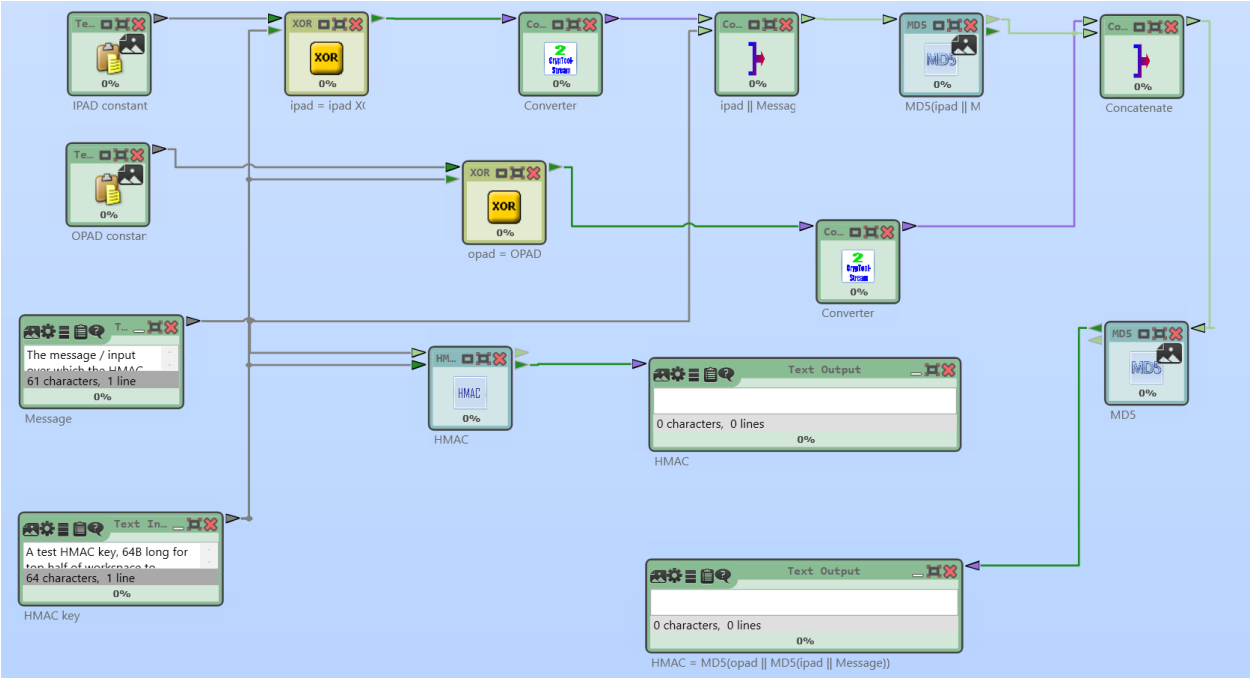

Рисунок 9 – HMAC в Cryptool 2

HMAC гарантирует участвующим в транзакции сторонам, что их сообщения не были перехвачены злоумышленником. Код (ключ) HMAC состоит из двух частей:

Общий набор криптографических ключей для отправителя (клиента) и получателя (сервера). Отправитель и получатель используют один и тот же ключ для создания и проверки HMAC;

Общая криптографическая хэш-функция, такая как SHA-1 или RIPEMD-128/60.

Формула для HMAC:

HMAC = hashFunc (секретный ключ + сообщение)

В процессе обмена сообщениями между клиентом и сервером с использованием HMAC клиент создает уникальный HMAC (хэш), хэшируя данные запроса с закрытыми ключами и отправляя их как часть запроса. Сервер получает запрос и регенерирует свой собственный уникальный HMAC. Затем он сравнивает два HMAC. Если они равны, клиент считается доверенным, и запрос выполняется. Этот процесс часто называют секретным рукопожатием (secret handshake).

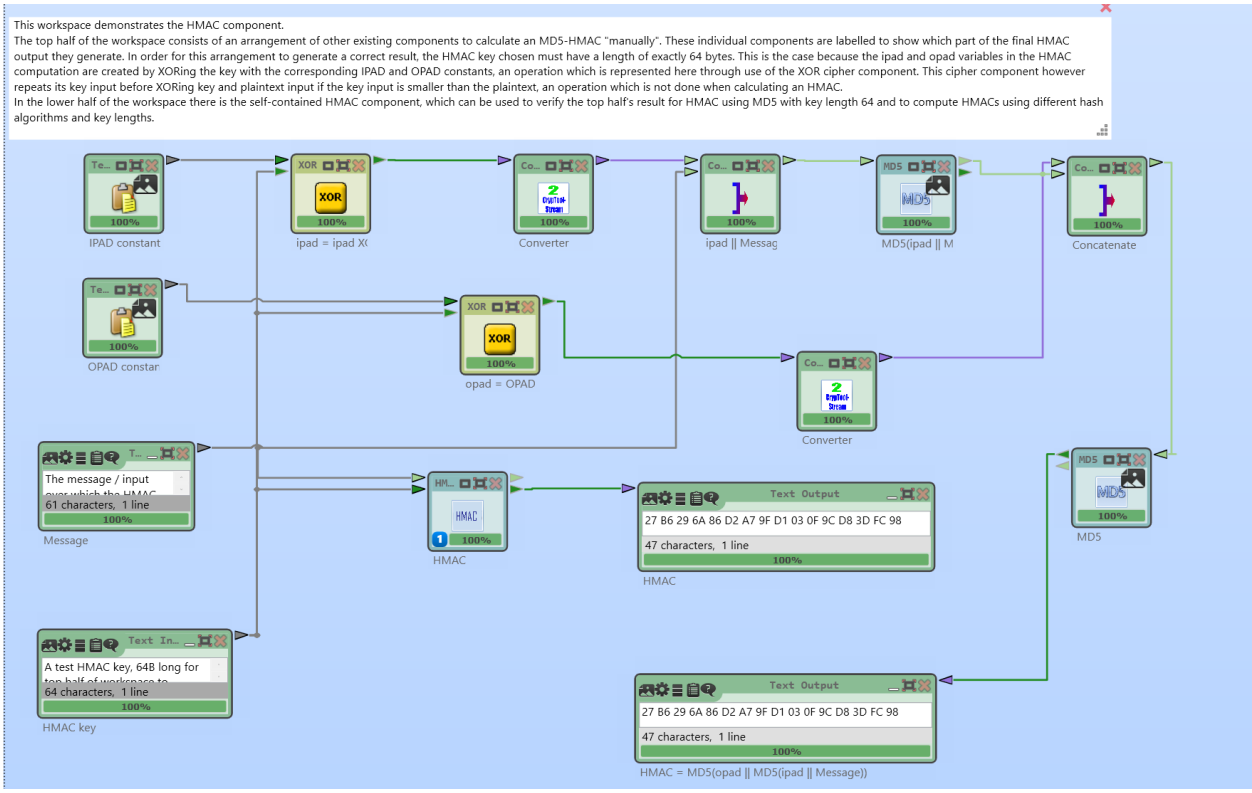

Рисунок 10 – HMAC в Cryptool 2

Представим алгоритм HMAC в виде формулы:

HMACK(K, text) = H ((K0 ⊕ opad) ‖ H((K0 ⊕ ipad) ‖ text)), где:

– «⊕» – исключающее «ИЛИ»;

– «‖» – склейка строк.

H - хэш-функция пароля (такая как MD5 или SHA-2) , который может выполнять сжатие группового цикла;

K — ключ (секретный ключ);

text — сообщение, которое должно быть аутентифицировано;

K0 — другой ключ, полученный из исходного ключа K (если K короче размера входного блока хэша). функция, затем правое дополнение нулями; решетка K, если она больше размера блока)

внутреннее заполнение ipad (0x5C5C5C...5C5C, шестнадцатеричная константа);

внешнее заполнение opad (0x363636...3636, шестнадцатеричная константа)

1.1.3 Обзор возможностей хэширования файлов и сообщений с помощью openssl dgst

Захешируем два произвольных сообщения, различающихся только регистром первого символа при помощи openssl dgst, применяя алгоритм md5

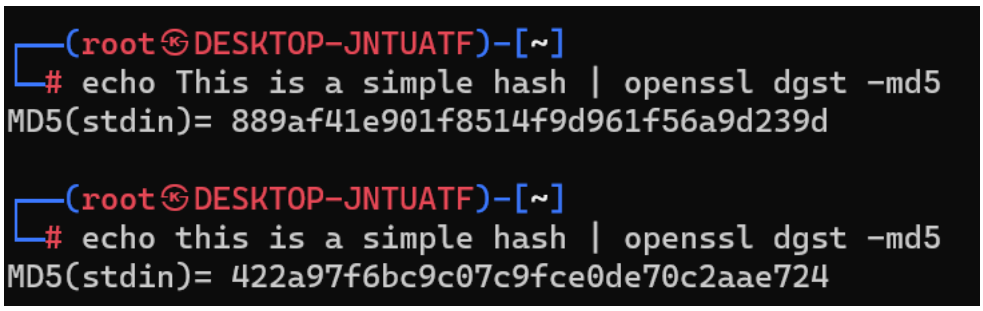

Рисунок 11 – Хеширование сообщения при помощи Openssl

Видим два хеша длиной 128 бит совершенно не похожих друг на друга. Что говорит о том, что минимальные изменения в исходном сообщении кардинально меняют внешний вид хеша.

Теперь протестируем другие алгоритмы хеширования – SHA-256 и SHA-512.

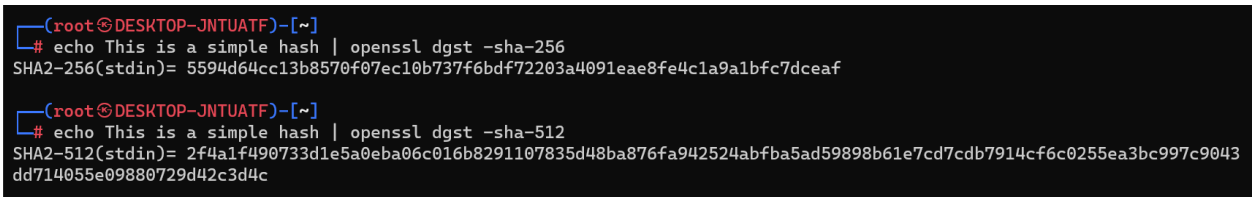

Рисунок 12 - Хеширование сообщения при помощи Openssl

Видим, что выходные значения не похожи и имеют разную длину.

1.1.4 Генерация hash based mac

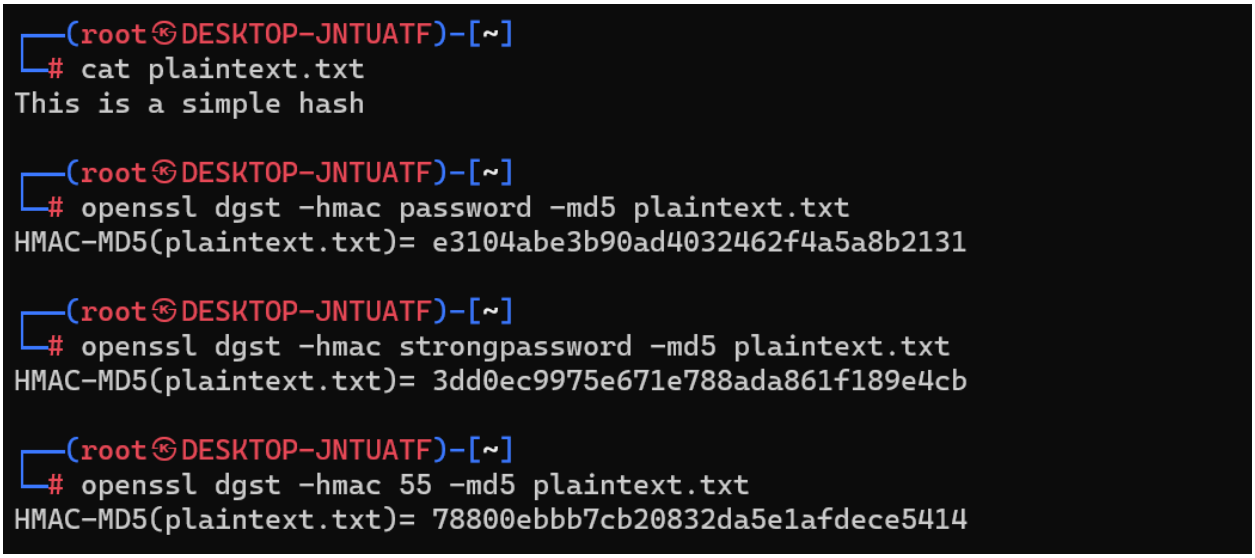

Рисунок 13 – Вычисление HMAC-MD5

Из рисунка выше мы видим, что вычисления HMAC-MD5 прошли успешно и все выходные значения отличаются друг от друга, поскольку каждый раз был использован другой пароль.

Теперь вычислим HMAC-SHA-256 и HMAC-SHA-512.

```
┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# openssl dgst -hmac passworpassword -sha-512 plaintext.txt
HMAC-SHA2-512(plaintext.txt)= 8182535954a1a02221f1cf154e4897f930ed1185dd442251602ba97351221916e6986aae216bf8b28c45c0127e1ca6e9d
9764490a2160fec84fc11d2857c82ad

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# openssl dgst -hmac passworpassword -sha-256 plaintext.txt
HMAC-SHA2-256(plaintext.txt)= 4d13027b0195068b022bf89ba5eda2ac08b592c40996aa0e9f9a787dd75faa24
```

Рисунок 14 – Вычисление HMAC-SHA-256 и HMAC-SHA-512

Как можем видеть, вычисления прошли успешно.

1.1.5 Формирование ключевой пары rsa, шифрованиесимметричного ключа, генерация и проверка подписи

```
┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# openssl genrsa -out rsa_key.txt

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# cat rsa_key.txt
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQC8ZoGecMsCiQ0W
dZvqGBLh2okXDR/2E8culYMerSI/3fxho2XtGBttJQpLtfd7f072QIgAuttLnHtT
7f5mEshV1237VizH6nW15CYAj0lPe+xRkatKqD7gf3FjOGGV+IcTtCZUUr6wW6BC
cdwEjnO2TNycHgPky+AnmFbM1Z1n6DRjtThFIjoXIPRzUbXHQcrjVcDfpPYER5LZ
hfRYNYuxkwqQbvqP5A0vPC1eJYvWfM0I6tLRpDHpD/r80Ipjg1fPtQYzMle8RObR
ktw1sHOQntwm/jtM9KcwNAGwCaYrsJPyr+fp8uwd+JzbvV7UBT7R2URUTbDjXKwu
W5QyV+IpAgMBAAECggEAFC4XVeJ3Xvr8IRA8QPZkK2IDoNRMkfNWDforfyEvaSTn
et6MNTm0UlwPKW3r0i7K7/CYlhoYZuseK1lzkE5pvAbP0TCj/6PwXiklHZaQ1TKN
F8L2fzrcmqJmrFM74gWqvk8Oc1DLpnX3Xragc7J92CSNKtcXjAMp1u6Sju1Ap4JA
s4nTwX8orDL7959w1OMcQTHZlpC1o/gNsR6IVB6Rw+DzZoly3ecZBv6wHCsK0b+g
xx9DuDxnpwuB4IYZ2+Mzl572JSJWx7yA6VnkSKwOr690zcXjOlfMKUv9Rbd72Qw3
bvsShod6yGhSLYOIoUOp75fyCHvVtKRMeIx8UTeFYwKBgQDzLP213nadW1LvR21h
djx1iCyqBnNN8iiMbaXQ/0zSDRRtprIyjChTkUQqIZmB3OxxiDlRCKMO+2xclZ0E
etfjVmaBYnPBYfUjbBulZW7SIQZLg0ILQInZtz8G9bp+JuQTqNoQUKKdYEA1lwHZ
Tm2Usbes1Q+B4tAeKwuoPx1y1wKBgQDGVgSARJy9PnhnvzVKS0Xs6An+9wOW/bp7
lL31C6YoJ6smBwhrbv5M1d3WPXiuUeXLj2BXQE4nsf1enolM4qTr/gx0BkGEWl/J
HW+ql2DEctet9SlRhh3M0ZZRXlPKn27y79ssINQ4NTqEuoRKnXAdGZ0kHuF5izZ3
uW3tev+y/wKBgHScN5G1+0VcHLoXk3EwJK/rvBdQfox7OkyU6/X75xe3I/us+zjY
EyJ3v6tAb39htJSA0ZN1Q9QUZxNCv4f9bHdc4EzdLEsosUfqOJuot2rWYVFmwQ3q
RfB8OiIZRfPmrGptBxzZvhEfrcrE/iL5w/yQ+FZN2WDcphZBaBanJXUDAoGBAInb
pjtF9bBwqJ5sQG1Iju0RiEkphtw/toxJohOeqjqFAChpgQez2ZyslwpgJe78y/5G
xU9ASlK7HALVrVe8vaevKRbIdme5rZSMGl7343II5TNsEfExiKsiDxq1twZC9Yp0
/+HXW8tJ7EfvoR5/+jqv4BMIR+8EvbJQOEF2Wh8fAoGAapCXSH6ZLvfNDf//DYOb
R1xcYxmRtQNjPicsNWtxzGGNxNWIyZ9hDHu8tcnf+T8081DZGIn9cuxLW353df/R
3owBOF+ptKYd0JPsWEngdnmhtGg/UCyZUMB+xZLCmzbL6SZ4qyiYDRj2PlWj1Cbe
B4PEj3xYG6Pw0TkSEwrZBq0=
-----END PRIVATE KEY-----
```

Рисунок 15 – Генерация ключа RSA

В этом файле помимо закрытого ключа также содержится и открытый.

```
┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# openssl rsa -in rsa_key.txt -pubout -out public_key.txt
writing RSA key

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# cat public_key.txt
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvGaBnnDLAokNFnWb6hgS
4dqJFw0f9hPHLpWDHq0iP938YaNl7RgbbSUKS7X3e39O9kCIALrbS5x7U+3+ZhLI
Vddt+1Ysx+p1teQmAI9JT3vsUZGrSqg+4H9xYzhhlfiHE7QmVFK+sFugQnHcBI5z
tkzcnB4D5MvgJ5hWzNWdZ+g0Y7U4RSI6FyD0c1G1x0HK41XA36T2BEeS2YX0WDWL
sZMKkG76j+QNLzwtXiWL1nzNCOrS0aQx6Q/6/NCKY4NXz7UGMzJXvETm0ZLcNbBz
kJ7cJv47TPSnMDQBsAmmK7CT8q/n6fLsHfic271e1AU+0dlEVE2w41ysLluUMlfi
KQIDAQAB
-----END PUBLIC KEY-----
```

Рисунок 16 – Публичный ключ

Выше мы наблюдаем публичный ключ в отельном файле.

Затем нам нужно сгенерировать симметричный ключ системы AES и поместить его в файл aes_key.

```
┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# openssl enc -aes-128-ctr -P
enter AES-128-CTR encryption password:
Verifying - enter AES-128-CTR encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=AA767CB6CA2066D8
key=A2BF1125B68D323F9FADC509DFC7E493
iv =D4CB5C95638C1F661B8FD41EE23AD407
```

Рисунок 17 – Генерация симметричного ключа

Далее выполним шифрование симметричного ключа при помощи открытого.

```
┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# openssl pkeyutl -encrypt -pubin -inkey public_key.txt -in aes_key -out secure.txt

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# cat secure.txt
[◆◆◆◆]◆}◆ED◆1◆
-◆◆◆,◆k◆zB;n◆b◆◆◆p◆◆.◆Ý~[%◆"◆◆◆◆H◆H◆◆◆zS◆9=◆,,◆羽◆◆"13◆c6◆„◆◆◆◆◆`~T◆o_◆C◆@◆◆\◆◆<4h◆◆A◆0◆◆◆◆◆&◆W3◆~◆?◆RSU◆◆U◆Y5{◆3d◆"ñ◆◆◆◆◆B
◆0O◆Q◆F◆~◆◆◆ ◆r◆c[◆◆4◆5◆◆◆◆◆0◆
                        ◆AR◆◆◆◆◆j>MK5◆ýTB◆t3i◆W◆
                                                t◆◆:)◆◆◆ ◆◆4
◆y◆R.◆}y◆◆◆ro
```

Рисунок 18 – Симметричный ключ

Расшифруем при помощи закрытого ключа симметричный ключ.

```
┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# openssl pkeyutl -decrypt -inkey rsa_key.txt -in secure.txt
A2BF1125B68D323F9FADC509DFC7E493
```

Рисунок 19 – Расшифрованный симметричный ключ

Генерируем хеш для зашифрованного ключа.

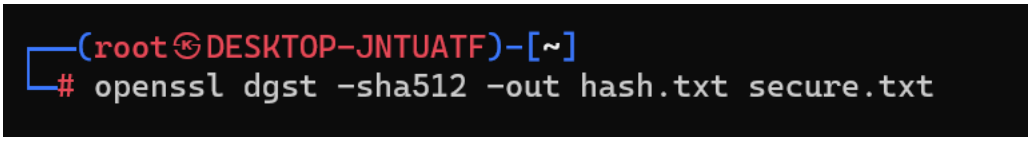

Рисунок 20 – Генерация хеша зашифрованного симметричного ключа

Подписываем получившийся хеш при помощи сгенерированного ключа RSA.

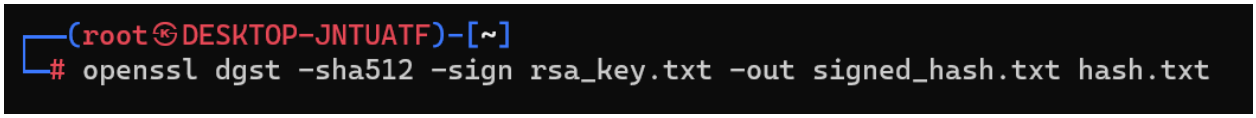

Рисунок 21 – Подпись хеша

Проверяем подписанный хеш.

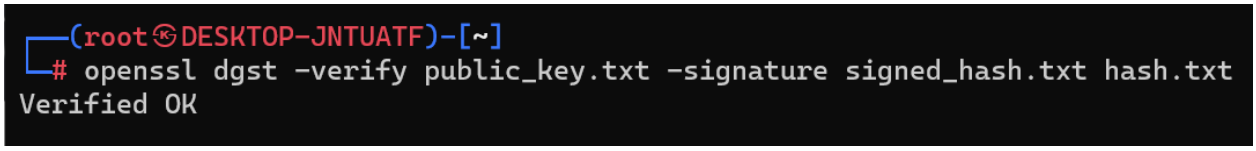

Рисунок 22 – Проверка подписи хеша

Продемонстрируем процесс шифрования и дешифрования открытого сообщения прим помощи симметричной криптосистемы на примере AES, при помощи команды openssl enc.

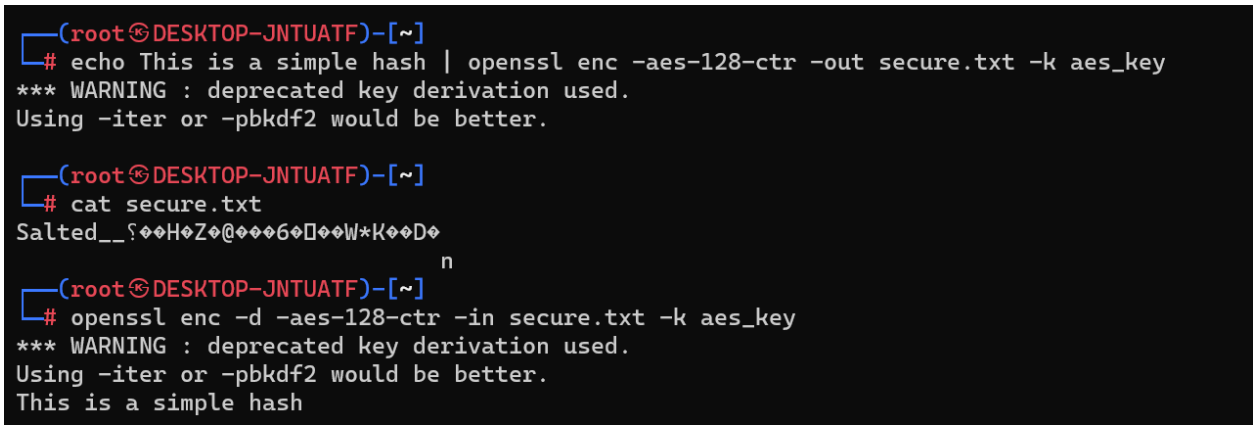

Рисунок 23 – Шифрование и дешифрование исходного сообщения

1.1.6 Сравнение двух хешей based mac

```
┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# echo This is a simple hash | openssl dgst -hmac passwordpassword -md5 -out hmac

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# echo Another hash | opennsl dgst -hmac passwordpassword -md5 -out hmac1
-bash: opennsl: command not found

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# echo Another hash | opennssl dgst -hmac passwordpassword -md5 -out hmac1
-bash: opennssl: command not found

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# echo Another hash | openssl dgst -hmac passwordpassword -md5 -out hmac1

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# cat hmac
MD5(stdin)= e3b119e319ddc4955d8c926c7cc5100c

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# cat hmac1
MD5(stdin)= 5bc8a9bf36f5539fe7e53638ca644927

┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# diff hmac hmac1
1c1
< MD5(stdin)= e3b119e319ddc4955d8c926c7cc5100c
---
> MD5(stdin)= 5bc8a9bf36f5539fe7e53638ca644927
```

Рисунок 24 – Сравнение двух HMAC

Для того, чтобы окончательно убедиться в корректности. Создадим еще один HMAC для первого открытого сообщения.

```
┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# diff hmac1 hmac2
```

Рисунок 25 – Сравнение идентичных HMAC

ВЫВОД

В ходе данной лабораторной работы были рассмотрены алгоритмы хеширования MD5, SHA-1, SHA256, SHA512. Были сгенерированы пары открытый-закрытый ключи для криптосистемы RSA. Был зашифрован случайный симметричный ключ алгоритма AES. Далее было проведено сравнение двух идентичных HMAC с помощью команды diff для операционных систем linux.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бабенко, Л. К. Современные алгоритмы блочного шифрования и методы их анализа / Л.К. Бабенко, Е.А. Ищукова. - М.: Гелиос АРВ, 2015. - 376 с.

2. Бабенко, Л.К. Современные интеллектуальные пластиковые карты / Л.К. Бабенко. - М.: Гелиос АРВ, 2015. - 921 с.

3. Болотов, А. А. Элементарное введение в эллиптическую криптографию. Протоколы криптографии на эллиптических кривых / А.А. Болотов, С.Б. Гашков, А.Б. Фролов. - М.: КомКнига, 2012. - 306 с.

4. Бузов, Геннадий Алексеевич Защита информации ограниченного доступа от утечки по техническим каналам / Бузов Геннадий Алексеевич. - М.: Горячая линия - Телеком, 2016. - 186 с.