

Министерство образования Республики Беларусь

Учреждение образования
“Белорусский государственный университет
информатики и радиоэлектроники”

Кафедра интеллектуальных информационных технологий

Лабораторная работа №4
"Аспекты создания защищенных веб-приложений"

Выполнил студент
группы 421701

Бруцкий Д.С.

Проверил

Захаров В.В.

МИНСК
2016

1 ЗАДАНИЕ

1. Установить OpenSSL и ознакомиться с возможностями библиотеки.
2. Выполнить тестирование скорости выполнения различных алгоритмов шифрования.
3. Создать криптографические ключи. Выбрать несколько произвольных файлов и выполнить:
 - а) шифрование (зашифрование и расшифрование) посредством различных симметричных алгоритмов;
 - б) шифрование (зашифрование и расшифрование) посредством различных асимметричных алгоритмов;
 - в) хеширование различных файлов различными алгоритмами (обязательно md5 и sha1).
4. Создать самоподписанный сертификат X509. Изучить состав сертификата и назначение его компонентов.
5. Оформить отчёт.

2 ХОД РАБОТЫ

1. Тестирование скорости алгоритмов

После установки OpenSSL было проведено тестирование скорости работы различных алгоритмов шифрования посредством выполнения команды `speed`.

rizhi-kote@rizhikote-M5400: ~

The 'numbers' are in 1000s of bytes per second processed.

type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
md2	0.00	0.00	0.00	0.00	0.00
mdc2	0.00	0.00	0.00	0.00	0.00
md4	67426.02k	201708.39k	458428.33k	679378.75k	792750.76k
md5	45158.95k	132249.17k	290500.18k	414866.77k	474565.29k
hmac(md5)	38304.78k	117943.72k	272493.48k	402800.98k	472588.29k
sha1	54768.84k	149513.71k	315841.11k	453109.79k	551791.27k
rmd160	31467.08k	74772.57k	134572.80k	169152.51k	180460.78k
rc4	269657.00k	486903.70k	568499.88k	596011.01k	592857.77k
des cbc	49531.22k	52145.43k	52941.31k	53034.67k	53013.16k
des ede3	19420.32k	19800.43k	19925.33k	19956.39k	19903.02k
idea cbc	0.00	0.00	0.00	0.00	0.00
seed cbc	50413.38k	52578.88k	52436.74k	52952.06k	52771.72k
rc2 cbc	32013.29k	33014.66k	32904.25k	32997.72k	33338.71k
rc5-32/12 cbc	0.00	0.00	0.00	0.00	0.00
blowfish cbc	80359.42k	86167.50k	86858.58k	87949.65k	88337.07k
cast cbc	74669.78k	80512.53k	81291.35k	81109.76k	81652.39k
aes-128 cbc	95880.80k	105147.03k	107420.76k	108182.19k	106078.21k
aes-192 cbc	79644.95k	87354.84k	86583.89k	87750.00k	89380.18k
aes-256 cbc	69771.36k	75367.47k	76494.76k	74714.58k	74440.70k
camellia-128 cbc	68174.28k	109517.06k	126173.27k	129116.50k	131504.73k
camellia-192 cbc	61864.49k	86125.33k	95076.86k	97794.72k	98093.74k
camellia-256 cbc	61105.18k	84940.07k	95018.75k	96454.66k	95742.29k
sha256	37918.39k	83660.91k	145266.43k	178634.93k	188705.45k
sha512	30673.14k	121429.13k	186368.26k	266047.83k	292159.49k
whirlpool	19954.28k	42102.50k	69965.06k	83428.01k	88130.68k
aes-128 ige	94987.00k	101037.57k	102153.56k	102718.12k	99576.49k
aes-192 ige	81337.72k	84750.29k	85487.36k	85738.15k	85786.62k
aes-256 ige	71159.24k	73525.67k	73965.48k	73949.18k	73926.68k
ghash	937102.83k	2002938.05k	2213102.59k	2238688.94k	2236364.29k
sign verify sign/s verify/s					
rsa 512 bits	0.000070s	0.000006s	14323.2	181451.1	
rsa 1024 bits	0.000210s	0.000014s	4765.4	70703.0	
rsa 2048 bits	0.001617s	0.000048s	618.4	21048.2	
rsa 4096 bits	0.012340s	0.000181s	81.0	5530.7	
sign verify sign/s verify/s					
dsa 512 bits	0.000072s	0.000063s	13926.9	15872.2	
dsa 1024 bits	0.000178s	0.000173s	5608.0	5788.4	
dsa 2048 bits	0.000564s	0.000548s	1774.4	1825.5	
sign verify sign/s verify/s					
160 bit ecdsa (secp160r1)	0.0001s	0.0003s	12867.8	3576.1	
192 bit ecdsa (nistp192)	0.0001s	0.0003s	10508.0	2981.2	
224 bit ecdsa (nistp224)	0.0001s	0.0002s	9919.6	4517.0	
256 bit ecdsa (nistp256)	0.0002s	0.0004s	6456.0	2658.8	
384 bit ecdsa (nistp384)	0.0003s	0.0011s	3815.4	910.5	
521 bit ecdsa (nistp521)	0.0006s	0.0014s	1616.8	707.0	
163 bit ecdsa (nistk163)	0.0002s	0.0006s	4314.4	1777.9	

Рисунок 1 – Тестирование скорости выполнения алгоритмов шифрования

Результаты говорят о том, что самая высокая скорость работы была у алгоритмов rc4(при размере блока от 16 байт до 256 байт) и hmac(md5)(при

размере блока от 1024 до 8192), эти алгоритмы обработали больше всего информации за секунду; самая низкая скорость была у алгоритма md2. Полученные данные не противоречат сведениям о данных алгоритмах. md2 действительно является относительно медленным алгоритмом хеширования и практически не используется в настоящее время, т.к. считается устаревшим. Об алгоритме rc4, напротив, известно, что он выполняется достаточно быстро и используется широко. Алгоритм HMAC — алгоритм хеширования с ключом для аутентификации сообщения — алгоритм позволяет хешировать входное сообщение L с некоторым ключом K, такое хеширование позволяет аутентифицировать подпись. На рассматриваемом интервале заметно увеличение производительности алгоритма с увеличением размера блока.

3. Симметричное шифрование(алгоритм blowfish)

При шифровании указывается пароль.

```
rizhi-kote@rizhikote-M5400:~$ openssl bf -a -in 1.jpg -out 1_cr.jpg
enter bf-cbc encryption password:
Verifying - enter bf-cbc encryption password:
rizhi-kote@rizhikote-M5400:~$ openssl bf -a -d -in 1_cr.jpg -out 1_out.jpg
enter bf-cbc decryption password:
```

Рисунок 3 – Шифрование алгоритмом blowfish

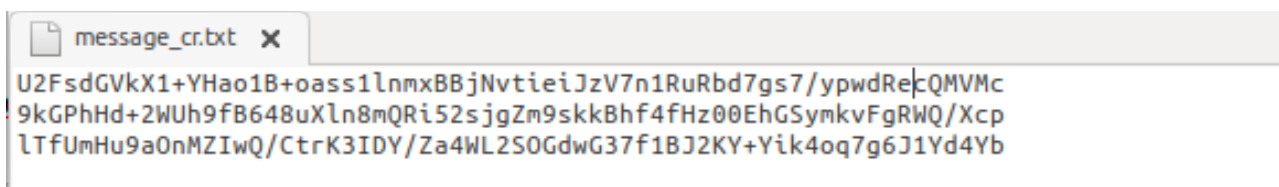


Рисунок 4 – Зашифрованный файл

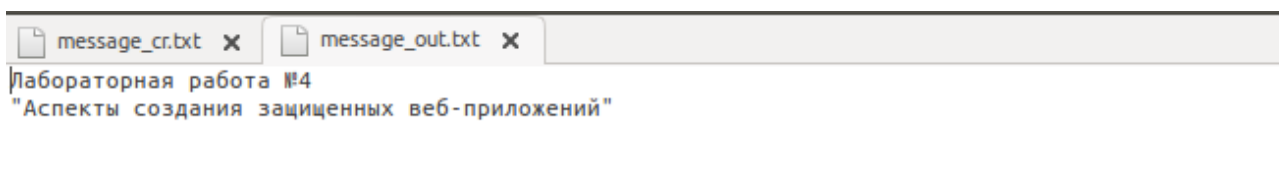


Рисунок 5 – Расшифрованный файл

3. Асимметричное шифрование(алгоритм rsa)

```

rizhi-kote@rizhikote-M5400:~$ openssl genrsa -out privateKey.txt 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
rizhi-kote@rizhikote-M5400:~$ openssl rsa -in privateKey.txt -pubout -out public
Key.txt
writing RSA key
rizhi-kote@rizhikote-M5400:~$

```

Рисунок 6 – Генерация приватного и публичного ключей rsa

```

privateKey.txt x
|-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQC1b/kkB3ZmpKlbymmjYkrBjpy6aA+lUSDCvkmUtWbCQ3BIJfAy
dNQGlleM0cu8WVVSvW/1wt+y0iPYn9ViYpuukDtBVtry54BTcevwL/mQz//8cRdp
OleL5CWRnc180gBk47+GPnxtz050SdI3qB6IGQorFfJ81B2/XG+6fkKwJQIDAQAB
AoGBALSVr0aR+uGssDgBHeWVdkxyMkJCQphLSC0skjXl21ItJztVL1s49rNbveV+
v05Jh+fjTuUwQdBM4YrBgvp6q3GFGMT1etM6PGjCyA8UvzVxC3I5WpxKg1WUyxsT
i2yZyqzXKrckxH/tjZPQU60SpLqd01eW1y5+yQG7nPB1B+cdAkeA4DQSnJlfYlEU
UGUDId33Zekq80mW/FWzHdfsqjZ7MzHqh8mI0ZBQ2CKLxcIjQ18fNMH4QafJ8DyD
xR5hPK8NDwJBAM8rPOkD3c82YrH8WGi0Dnw6rzvr+sN6SB1QWnSaN7wyuvOUQemR
8yT3LBXCT7zGv9+hqkxg8Dl3Xh4aUELRL14sCQQC3lCvBKheBoGZqnw8G0HgAmH2
v0Tv7MQx00DqJp40N0sQUEbFQWPQ9ch47x0rt6XXrx01vsus9//1QYDhDXylAkEA
yafR3WF/GoF05o2cvQFyRiiVhD161aaE5cvekNvA3vd+ltU+R45i4BudFP0Uwsuv
344tB7Xgf/+aZLD2aWR7awJAdfppFkohAvALmk8NUoImMlyZeHrFEqPdmPbyhCb
nlhtb/vvi82kEK+L+l2PoS0lxopNOVjL/3wxcx5rIjN3Lg==
-----END RSA PRIVATE KEY-----

```

Рисунок 7 – Приватный ключ

```

publicKey.txt x
|-----BEGIN PUBLIC KEY-----
MIGfMA0GCsGqGSIB3DQEBAQUAA4GNADCBiQKBgQC1b/kkB3ZmpKlbymmjYkrBjpy6
aA+lUSDCvkmUtWbCQ3BIJfAydnNQGlleM0cu8WVVSvW/1wt+y0iPYn9ViYpuukDtB
Vtry54BTcevwL/mQz//8cRdpOleL5CWRnc180gBk47+GPnxtz050SdI3qB6IGQor
FfJ81B2/XG+6fkKwJQIDAQAB
-----END PUBLIC KEY-----

```

Рисунок 8 – Публичный ключ

```

rizhi-kote@rizhikote-M5400:~$ openssl rsautl -encrypt -inkey publicKey.txt -pubi
n -in message.txt -out des.txt
rizhi-kote@rizhikote-M5400:~$ openssl rsautl -decrypt -inkey privateKey.txt -in
des.txt -out desd.txt
rizhi-kote@rizhikote-M5400:~$

```

Рисунок 9 – Зашифровка файла и последующая расшифровка

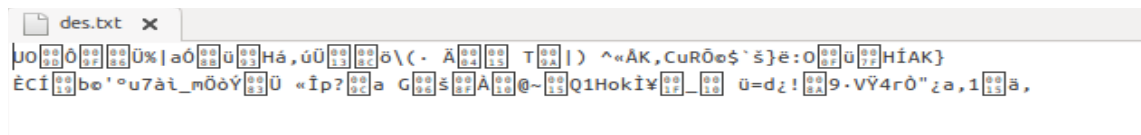


Рисунок 10 – Зашифрованный файл

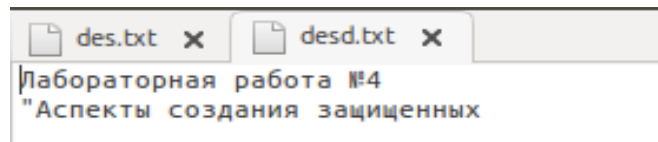


Рисунок 11 – Расшифрованный файл

4. Хеширование файлов (md5, sha1, rmd160)

```
rizhi-kote@rizhikote-M5400:~$ openssl md5 message.txt
MD5(message.txt)= 7e68ee5a2649a927169b0e2dbcee2856
rizhi-kote@rizhikote-M5400:~$ openssl md5 message_out.txt
MD5(message_out.txt)= 7e68ee5a2649a927169b0e2dbcee2856
rizhi-kote@rizhikote-M5400:~$ openssl sha1 message.txt
SHA1(message.txt)= f97c172a7c91f4627006e44acf7a9df9aa5fce2e
rizhi-kote@rizhikote-M5400:~$ openssl sha1 message_out.txt
SHA1(message_out.txt)= f97c172a7c91f4627006e44acf7a9df9aa5fce2e
rizhi-kote@rizhikote-M5400:~$ openssl rmd160 message.txt
RIPEMD160(message.txt)= 884d0c9fa1b62a2d166a081f05a3d358a11d7c31
rizhi-kote@rizhikote-M5400:~$ openssl rmd160 message_out.txt
RIPEMD160(message_out.txt)= 884d0c9fa1b62a2d166a081f05a3d358a11d7c31
```

Рисунок 12 – Вычисление хеш-значений различными алгоритмами

5. Создание самоподписанного сертификата X509

Генерируем запрос на получение публичного ключа, публичный ключ и сертификат. Далее необходимо добавить сгенерированный ключ в хранилище ключей.

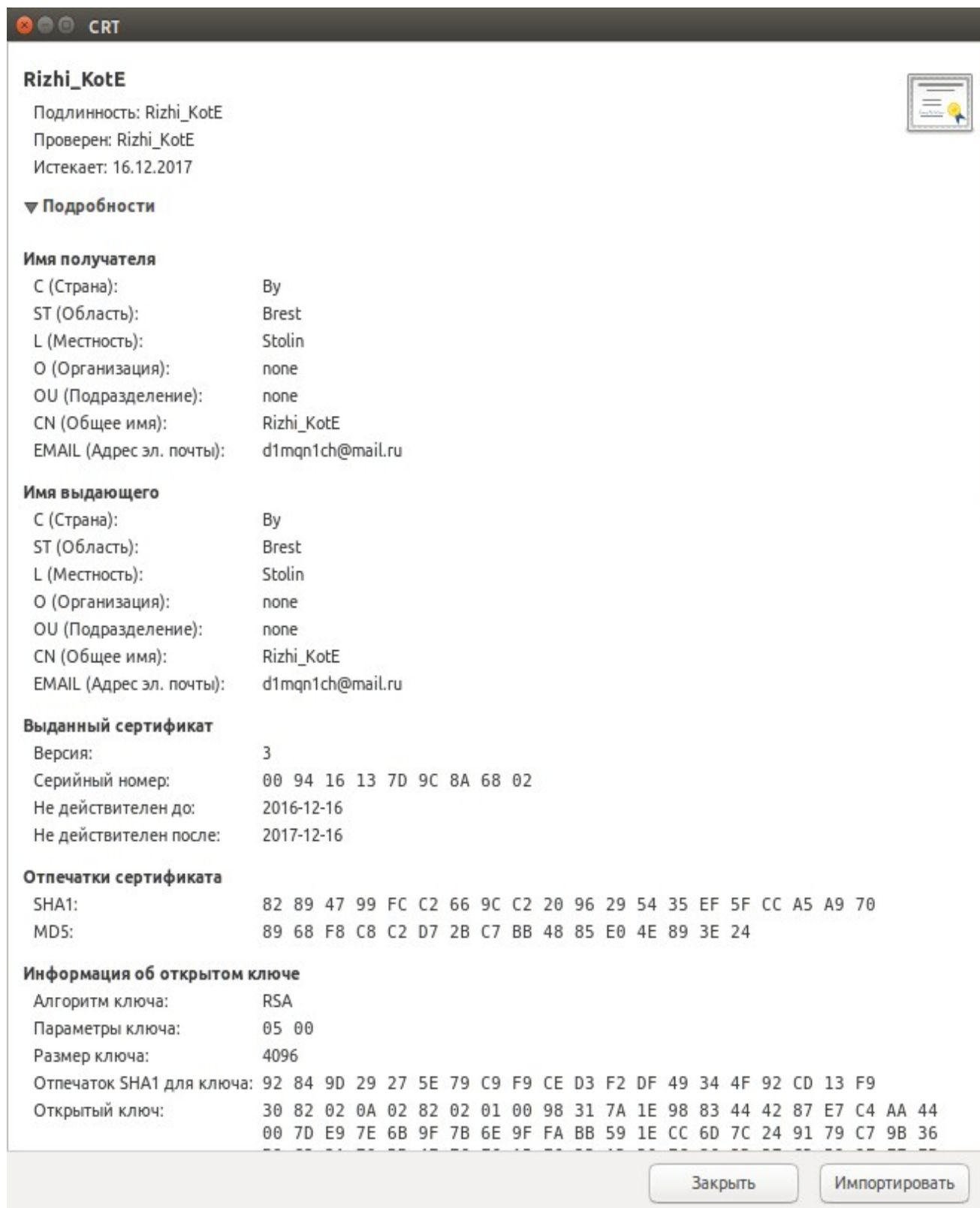


Рисунок 13. Сертификат

6 ВЫВОД

В результате выполнения работы был проанализирован http и https трафик. В результате анализа было выявлено, что защищенные данные сложнее интерпретировать, как следствие обеспечивает безопасность https протокол более высокую.

Также были проанализированы различные алгоритмы шифрования, при этом было выявлено, что алгоритмы rc4 и hmac(md5) обработали наибольшее количество байт за секунду.

Был создан и проанализирован сертификат безопасности. Сертификат включает в себя: версию, срок действия, алгоритм шифрования, имя компании, выдавшей сертификат, а также другие атрибуты.