


(https://profile.intra.42.fr)

SCALE FOR PROJECT FT_SSL_RSA (/PROJECTS/FT_SSL_RSA)

You should evaluate 1 student in this team



Git repository

vogsphere@vgs.42.us.or 

Introduction

In order to maintain high evaluation standards, you are expected to:

Stay polite, courteous, respectful and constructive at every moment of the discussion. Trust between you and our community depends on your behaviour.

Highlight the flaws and issues you uncover in the turned-in work to the evaluated student or team, and take the time to discuss every aspect extensively.

Please take into account that discrepancies regarding the expected work or fonctionnalités definitions might occur. Keep an open mind towards the opposite party (is he or she right or wrong?), and grade as honestly as possible. 42's pedagogy only makes sense if peer-evaluations are carried out seriously.

Guidelines

You must grade only what exists in the GiT repository of the student or team.

Be careful to check the GiT repository's ownership:: is it the student's or team's repository, and for the right project?

Check thoroughly that no wicked aliases have been used to trick you into grading something other than the genuine repository.


Any script supposed to ease the evaluation provided by one party must be thoroughly checked by the other party in order to avoid unpleasant

situations.

If the student in charge of the grading hasn't done the project yet, it is mandatory that he or she reads it before starting the evaluation.

Use the available flags on this scale to tag an empty work, a non functional work, a coding style ("norm") error if applicable, cheating, and so on. If a flag is set, the grade is 0 (or -42 in case of cheating). However, cheating case excluded, you are encouraged to carry on discussing what went wrong, why, and how to address it, even if the grading itself is over.

Attachments

 Ft SSL RSA (https://cdn.intra.42.fr/pdf/pdf/5253/ft_ssl_rsa.pdf)

Mandatory Part

The basics. Without mastering them, you are nothing.

Did they extend the executable from the previous project?

When you run `./ft_ssl invalid_param``, does it show an appropriate error message and display the list of available commands?

Are those commands the same as required in the previous project?

If you run one of the commands, does it behave as it should?

☒ Yes

☐ No

PRIMEP

The program should include a primality tester.

To the best of your ability, verify that it functions at a reasonable level (It shouldn't be slow).

Here are some numbers you can try testing:

``2018111111`` is prime

``215458247`` is prime

``21564653`` is not prime

``656452`` is not prime

``7577784281823102539`` is prime

☒ Yes

☐ No

RNG

The program should include a random number generator.

Verify that the standard C rand() function is not being called.

The number generated should be pulled from `/dev/urandom` or similar.

Justification is required if other methods are used to generate random numbers.

☒ Yes

☐ No

GENRSA

Generation of 64bit RSA keys.

Is the output format valid?

Compare the outputs of the following commands:

``./ft_ssl genrsa``

``openssl genrsa 64``

Are they formatted the same way?

(Disregard mismatching content as the keys are randomly generated. Just check that they are formatted the same way.)

☒ Yes

☐ No

Are subsequent keys random?

Run the command ``./ft_ssl genrsa`` several times (at least 5 times) in succession.

You should have gotten a different key each time the command was run.

☒ Yes

☐ No

Are the keys valid?

Run the command ``./ft_ssl genrsa > key.out; openssl rsa -check -in key.out`` several times (at least 5 times) in succession.

You should get the output ``RSA key ok`` or similar.

☒ Yes

☐ No

Are the keys the correct length?

Run the command ``./ft_ssl genrsa > key.out; openssl rsa -in key.out -text`` several times (at least 5 times) in succession.

You should get the output ``Private-Key: (64 bit)`` or similar. It must be 64 bit.

☒ Yes

☐ No

RSA

Generate a key using ``openssl genrsa 64 > test.key``. This key will be used throughout the rest of this correction. Feel free to repeat any of the following tests with multiple valid keys, this is encouraged! Be thorough!

Input

Verify that all the following commands give the same output:

```
`cat test.key | ./ft_ssl rsa`  
`cat test.key | openssl rsa`  
`./ft_ssl rsa -in test.key`  
`openssl rsa -in test.key`
```

☒ Yes

☐ No

Output

Run the following commands:

```
`./ft_ssl rsa -in test.key -out key0.out`  
`openssl rsa -in test.key -out key1.out`
```

Verify that the contents of ``key0.out`` and ``key1.out`` are the same:

```
`cat -e key0.out`  
`cat -e key1.out`
```

☒ Yes

☐ No

Text

Verify that all the following commands give the same or similar output:

```
`./ft_ssl rsa -in test.key -text`  
`openssl rsa -in test.key -text`
```

Regardless of formatting, the contents should be identical.

☒ Yes

☐ No

Validation

Run the following two commands. They should both inform you that the test key is valid.

```
`./ft_ssl rsa -in test.key -check`  
`openssl rsa -in test.key -check`
```

For the following keys, make sure the above commands agree on whether the keys are valid or not.
For the sake of the rest of the correction, do not modify `test.key`. Use another filename.

-----BEGIN RSA PRIVATE KEY-----

MCwCAQACBQC4Gd5vAgMBAAECBFqVEIECAwDcyQIDANV3AgIkEQICNaUCAwDGDA==

-----END RSA PRIVATE KEY-----

-----BEGIN RSA PRIVATE KEY-----

MDoCAQACASoCAwEAAQIIIVWdCrvQXjn0CBHhJ7ocCCGkpsqfOmsZLAgMkuWUCCFVn
Qq70F459AgRjiS+d

-----END RSA PRIVATE KEY-----

Run the following commands as many times as you'd like.

`./ft_ssl genrsa > key.out`

The following two commands should always agree that the key is valid.

`./ft_ssl rsa -in key.out -check`

`openssl rsa -in key.out -check`

☒ Yes

☐ No

Generating Public Keys

Verify that all the following commands give the same output:

`./ft_ssl rsa -in test.key -pubout`

`openssl rsa -in test.key -pubout`

☒ Yes

☐ No

Reading Public Keys

Run the following command:

`./ft_ssl rsa -in test.key -pubout -out test.pub`

Verify that all the following commands give the same output:

`./ft_ssl rsa -in test.pub -pubin`

`openssl rsa -in test.pub -pubin`

☒ Yes

☐ No

Option Parser

Run the following sets of commands. For each set that matches, add a point.

`./ft_ssl rsa -in test.pub -pubin -noout -modulus`

`openssl rsa -in test.pub -pubin -noout -modulus`

```
`./ft_ssl rsa -modulus -text -in test.key`  
`openssl rsa -modulus -text -in test.key`  
  
`./ft_ssl rsa -modulus -text -in test.key -pubout`  
`openssl rsa -modulus -text -in test.key -pubout`  
  
`./ft_ssl rsa -text -pubin -in test.pub`  
`openssl rsa -text -pubin -in test.pub`
```

If every set matched, add another point.



Rate it from 0 (failed) through 5 (excellent)

Error Management

Run the following commands. Each command should deal with their errors appropriately (not ignore them!). Add a point for each command that properly handles its errors.

```
`./ft_ssl rsa -in test.key foo`  
`./ft_ssl rsa -in test.key -foo`  
`./ft_ssl rsa -in test.key -pubin`  
`./ft_ssl rsa -in test.pub`  
`./ft_ssl rsa -in not.real.file`
```



Rate it from 0 (failed) through 5 (excellent)

RSAUTL

RSA in application.

Encrypt

Verify that all the following commands give the same or similar output:

```
`echo -n 01234567 | ./ft_ssl rsautl -inkey test.key -encrypt -hexdump`  
`echo -n 01234567 | openssl rsautl -raw -inkey test.key -encrypt -hexdump`
```

Formatting may vary, but the output bytes must not differ *at all*.

Run the command ``diff <(echo -n 01234567 | ./ft_ssl rsautl -inkey test.key -encrypt) <(echo -n 01234567 | openssl rsautl -raw -inkey test.key -encrypt)``.

There should be no output.

Feel free to test with 64bit messages other than `01234567`. Some examples are `HiThere.` and `EfT64bFg`. Testing with multiple keys is encouraged.

☒ Yes

☐ No

Decrypt

Verify that all the following commands give the exact output `01234567`:

```
`echo -n 01234567 | ./ft_ssl rsautl -inkey test.key -encrypt | openssl rsautl -raw -inkey test.key -decrypt`
```

```
`echo -n 01234567 | ./ft_ssl rsautl -inkey test.key -encrypt | ./ft_ssl rsautl -inkey test.key -decrypt`
```

```
`echo -n 01234567 | openssl rsautl -raw -inkey test.key -encrypt | ./ft_ssl rsautl -inkey test.key -decrypt`
```

Again, feel free to test with other keys and 64bit messages.

☒ Yes

☐ No

Bonus

DES-epticons

Has `gendsa` been implemented? Does it work?

How about `dsa`?

Have they implemented a method to generate DES keys as well?

Be sure to test against `openssl gensa` and `openssl dsa`.

Rate it from 0 (failed) through 5 (excellent)



C-C-C-Combo breaker

Is there any implementation in the submission to break 64bit keyed encryption?

When we say break, we mean deriving the private key and original message with only the ciphertext and the public key.

Grade this section based on the complexity of the algorithm used, and make sure to ask lots of questions.

Brute force algorithms should not receive 5 points!

Rate it from 0 (failed) through 5 (excellent)



Ratings

Don't forget to check the flag corresponding to the defense



Ok



Outstanding project



Empty work



Incomplete work



No author file



Invalid compilation



Norme



Cheat



Crash



Leaks



Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation

General term of use of the site (<https://signin.intra.42.fr/legal/terms/6>)

Privacy policy (<https://signin.intra.42.fr/legal/terms/5>)

Legal notices (<https://signin.intra.42.fr/legal/terms/3>)

Declaration on the use of cookies (<https://signin.intra.42.fr/legal/terms/2>)

Terms of use for video surveillance (<https://signin.intra.42.fr/legal/terms/1>)

Rules of procedure (<https://signin.intra.42.fr/legal/terms/4>)