ft_ssl [genrsa] [rsa] [rsautl]

Blowing your mind with math.

Summary: This project is a continuation of the previous encryption project. You will recode part of the OpenSSL program, specifically GENRSA, RSA, and RSAUTL.

Version: 2.1

Contents

Chapter I

Foreword

The following is an excerpt from "The Alice and Bob After Dinner Speech"
given at the Zurich Seminar, April 1984.

Good eveni**n**g Ladies and Gen**t**lemen.

There comes a time when people at a technical conference like this need something more relaxing. A change of pace. A shift of style. To put aside all that work stuff and think of something refreshingly different.

So let**'s** talk about coding theory. There are perhaps some of you here tonight who are **not** experts in coding theory, but rather h**a**ve been dragged here kicking and s**cr**eaming. So I thought it would be a good idea if I gave you a sort of instant, five minute graduate course in coding theor**y**.

Coding theorists are concerned with two things. Firstly and most im**p**or**t**antly they are c**o**ncerned with the **pr**iv**a**te lives of two **pe**ople called Alice and Bob. In theo**r**y papers, **w**henever a coding theor**ist** wants to describe a transaction between two parties **h**e d**o**esn't call them A and B. No. For some longstanding traditional reason he calls them Alice and Bob.

Now there are h**u**ndreds of papers wri**t**ten about Alice and Bob. Over the year**s** Alice and B**o**b have tried to defraud insurance co**m**pani**e**s, they've played poker for high **st**ak**e**s by mail, and they've exchan**g**ed secret mess**a**ges over tapped telepho**n**es.

If we put t**og**ethe**r** **a**ll the little details from here and there, sni**p**pets from lots of papers, we get a fascinating picture of their lives. T**h**is ma**y** be the first time a definitive biography of Alice and Bob has been given.

In p**a**pers written by A**m**er**i**can authors Bob is f**r**equently sell**i**ng s**t**ock to sp**e**culators. From the number of stock market deals Bob is involved in we infer that he is probably a stockbroker. However from his concern about eavesdropping he is probably active in some subversive enterprise as well. And from the number of times Alice tries to buy stock from him we infer she is probably a speculator. Alice is also concerned that her financial dealings with Bob are not brought to the attention of her husband. So Bob is a subversive stockbroker and Alice is a two-timing speculator.

But Alice has a number of serious problems. She and Bob only get to talk by telephone or by electronic mail. In the country where they live the telephone service is very expensive. And Alice and Bob are cheapskates. So the first thing Alice must do is MINIMIZE THE COST OF THE PHONE CALL.

The telephone is also very noisy. Often the interference is so bad that Alice and Bob can hardly hear each other. On top of that Alice and Bob have very powerful enemies. One of their enemies is the Tax Authority. Another is the Secret Police. This is a pity, since their favorite topics of discussion are tax frauds and overthrowing the government.

These enemies have almost unlimited resources. They always listen in to telephone conversations between Alice and Bob. And these enemies are very sneaky. One of their favorite tricks is to telephone Alice and pretend to be Bob.

Well, you think, so all Alice has to do is listen very carefully to be sure she recognizes Bob's voice. But no. You see Alice has never met Bob. She has no idea what his voice sounds like.

So you see Alice has a whole bunch of problems to face. Oh yes, and there is one more thing I forgot so say - Alice doesn't trust Bob. We don't know why she doesn't trust him, but at some time in the past there has been an incident.

Now most people in Alice's position would give up. Not Alice. She has courage which can only be described as awesome. Against all odds, over a noisy telephone line, tapped by the tax authorities and the secret police, Alice will happily attempt, with someone she doesn't trust, whom she cannot hear clearly, and who is probably someone else, to fiddle her tax returns and to organize a coup d'etat, while at the same time minimizing the cost of the phone call.

A coding theorist is someone who doesn't think Alice is crazy.

`You may continue to read the rest of the speech here.`

Chapter II

Introduction

The `RSA` algorithm is named after Ron Rivest, Adi Shamir and Leonard Adleman, who invented it in 1977. The RSA cryptosystem is the most widely-used public key cryptography algorithm in the world. It can be used to encrypt a message without the need to exchange a secret key separately. It can be used for both public key encryption and digital signatures.

Its security is based on the difficulty of factoring large integers that are the product of two large prime numbers. (Multiplying these two numbers is easy, but determining the original prime numbers from the total is considered infeasable due to the time it would take even using today's supercomputers.)

The public and the private-key generation algorithm is the most complex part of the whole RSA cryptosystem. Two large prime numbers are generated using the `Rabin-Miller` primality test algorithm. A modulus is calculated by multiplying the two numbers, and is used by both the public and the private keys. Its length, usually expressed in bits, is called the key length. The public key consists of the modulus, a public exponent (normally 65537, as it's a prime number that's not too large). The exponent doesn't have to be secretly selected as the public key is shared with everyone. The private key consists of the modulus and a private exponent, which is calculated using the Extended Euclidian algorithm to find the multiplicative inverse with respect to the totient of the modulus.

`There is a lot of complex math terminology going on in here.`

Chapter III

Objectives

This is the third `ft_ssl` project on the path of `Encryption and Security`. You will recode some security technologies you may have already been using, from scratch.

You will also gain experience and knowledge in using public and private keys to encrypt using one-way trapdoor functions. If you are smart you will also research:

- Digital signing with a private key
- Verifying digital signatures with a public key
- Sending encrypted messages over untrusted networks without prior key distribution

By the end of this you should be intimately familiar with the workings of asymmetric key cryptosystems.

Chapter IV

General Instructions

- This project will only be corrected by other human beings. You are therefore free to organize and name your files as you wish, although you need to respect some requirements below.
- The executable file must be named `ft_ssl`.
- You must submit a Makefile. The Makefile must contain the usual rules and compile the project as necessary.
- You have to handle errors carefully. In no way can your program quit unexpectedly (Segfault, bus error, double free, etc). If you are unsure, handle the errors like `OpenSSL`.
- You are allowed the following functions:
    - any functions allowed for the previous exercises
- You are allowed to use other functions as long as their use is justified. (Although they should not be necessary, if you find you need `strerror` or `exit`, that is okay, though `printf` because you are lazy is not)
- You can ask your questions on slack.

Chapter V

Mandatory part

You will be adding key generation using primality number tests, and one-way trapdoor encryption with the RSA algorithm.

```
> ft_ssl
usage: ft_ssl command [command opts] [command args]
```

For this project, you will be adding **genrsa**, **rsa**, and **rsautl**.

```
> ft_ssl foobar
ft_ssl: Error: 'foobar' is an invalid command.

Standard commands:
genrsa
rsa
rsautl

Message Digest commands:
md5
sha256

Cipher commands:
base64
des
des-ecb
des-cbc
>
```

`man openssl`

V.0.1 Optimus Prime

The first thing you must do with RSA keys before you use them is... have them! And to have them, you need to generate them. Not get them from someone else, because you can trust nobody, not even nobody. And certainly not me.

Creating keys is composed of multiple functions: The Diffie-Hellman

For this first part, you must create a function that takes an `unsigned 64 bit long` number and a probability between 0.0 and 1.0 (or 0 to 100, as you see fit) that the given number is prime.

If you don't know what prime numbers are, you should probably check on wikipedia.

`See Solovay-Strassen and/or Miller-Rabin algorithms.`

You will use this function later to test the primality of the random integers you use later for key generation, so be mindful of your work. If your code tests all cases (fully factors a given integer of that size) it will be too slow, so find a nice balance between accuracy and speed.

V.0.2 Ratchet Sideswipe Arcee

The first command that you must add to your `ft_ssl` is `genrsa`. It must generate a private key to the standard output or to a file as specified.

You must implement the following flag: `-out`.

```
> ft_ssl genrsa
Generating RSA private key, 64 bit long modulus
.............................++++++++++++
.....................++++++++++++
e is 65537 (0x10001)
-----BEGIN RSA PRIVATE KEY-----
MIIBOwIBAAJBAMLh8BxMEm/x+wDjpcMAeCANVFUfKdp9XR2H4VAnCK7b3x6SBD0v
q/e5iyp+zPDMiG2A263x6eQCRbUOXMpU1txEWgCk4w==
-----END RSA PRIVATE KEY-----
> ft_ssl genrsa -out key.pem
[..]
>
```

`man genrsa, urandom`

`You may not use rand() or time() for your RNG/PRNG!`

V.0.3 Rollout Smoothly, Autobots

You must also add `ft_ssl rsa` and `ft_ssl rsautl` that behave exactly like `openssl`.

`ft_ssl rsa` must include the following options:

```
ft_ssl rsa [-inform PEM] [-outform PEM] [-in file] [-passin arg] [-out file] [-passout arg] [-des] [-
    text] [-noout] [-modulus] [-check] [-pubin] [-pubout]
```

`man rsa`

`ft_ssl rsautl` must include specifically `encrypt` and `decrypt`, and we recommend that you implement all of these flags for a complete understanding.

```
ft_ssl rsautl [-in file] [-out file] [-inkey file] [-pubin] [-encrypt] [-decrypt] [-hexdump]
```

`man rsautl`

Chapter VI

Bonus part

VI.0.1 First bonus

You may also implement the `gendsa` key generation option, and also an option to generate `DES` keys (why don't we call it `gendes`?).

```
DSA keys behave like RSA keys in that they are asymmetric
public/private as well, however beyond that they are very
different.  DSA keys are faster at signing documents and generate
a smaller signature, but slower at verifying, and can't be used for
encryption/decryption.
```

```
man dsa, gendsa
```

VI.0.2 Second bonus

Up until now we have been working with a key size of 64 bits, which is generally considered insecure by today's standards. Now you get to learn how and why! Come up with any way to break a message encrypted by a 64 bit key.

- Lurk around the Mersenne Twister (oooh~!)
- Brute force break it (booo~!)
- ???
- Profit.

If you attempt this challenge, be nice to your corrector and provide a separate executable with its own Makefile, or include a reasonably named flag or command to your `ft_ssl`.

For example:

```
> ./ft_breakit
usage: ft_breakit [-k keysize] [-a algo] [-p plaintext] ciphertext_file
>
> ./ft_breakit sample.txt
No plaintext provided.
Begining analysis:...
Estimated keysize is 64 bits.
Attempting DES decryption.
<Saving DES log to "sample.txt.des">
...
```

```
> ./ft_ssl extractkey
usage: extractkey [-k keysize] [-a algo] [-p plaintext] ciphertext_file
```

The executable names, flags and output are suggestions for inspiration, they do not need to be exactly duplicated. You do still need to prove to your corrector that it works.

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter VII

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.