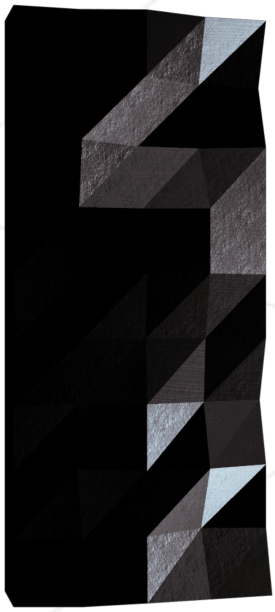


An introduction to Encryption

ft_ssl [base64] [des]

42 staff staff@42.fr

Summary: This project is the gateway to the encryption branch. You will recode part of the OpenSSL program, specifically BASE64, DES-ECB and DES-CBC.

Contents

Chapter I

Foreword

`Gaius Julius Caesar`, usually called Julius Caesar, was a Roman politician and general. He is well known for several things, the only one cryptographically significant being the `Caesar Cipher`, an encryption scheme he used to communicate with his generals and write in his diary.

The Caesar Cipher used the standard 26 letter alphabet and a numeric shift.
For example, encrypting with a left shift of 3:

- `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
- `XYZABCDEFGHIJKLMNOPQRSTUVW`

The encryption step is often incorporated as part of more complex schemes, and still has modern application in `ROT13`. As with all single-alphabet substitution ciphers, the cipher is easily broken and in modern practice offers no communication security. This is because it essentially has a single permutation, and only 26 keys (less than 6 bits of possible values!).

Ways to improve the security of the cipher include alphabet randomization and alphabet expansion:

- `Expansion:` `ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789`
- `Expansion:` `56789ABCDEFGHIJKLMNOPQRSTUVWXYZ01234`
- `Randomize:` `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
- `Randomize:` `QAZXSWEDCVFRTGBNHYUJMKIOLP`

It should be noted that neither of these protect against a modern liguistic frequency analysis. The cipher was most effective at the time of its use, primarily because most of his enemies were illiterate.

Julius Caesar died during an assassination during a session of the Roman Senate. Around 60 men participated in the assassination. He was stabbed 23 times.

As we can see from this example, `writing your own encryption algorithm is risky at best` and deadly at worst. For your safety during these excercises, you will rewrite existing algorithms rather than create new ones.

Chapter II

Introduction

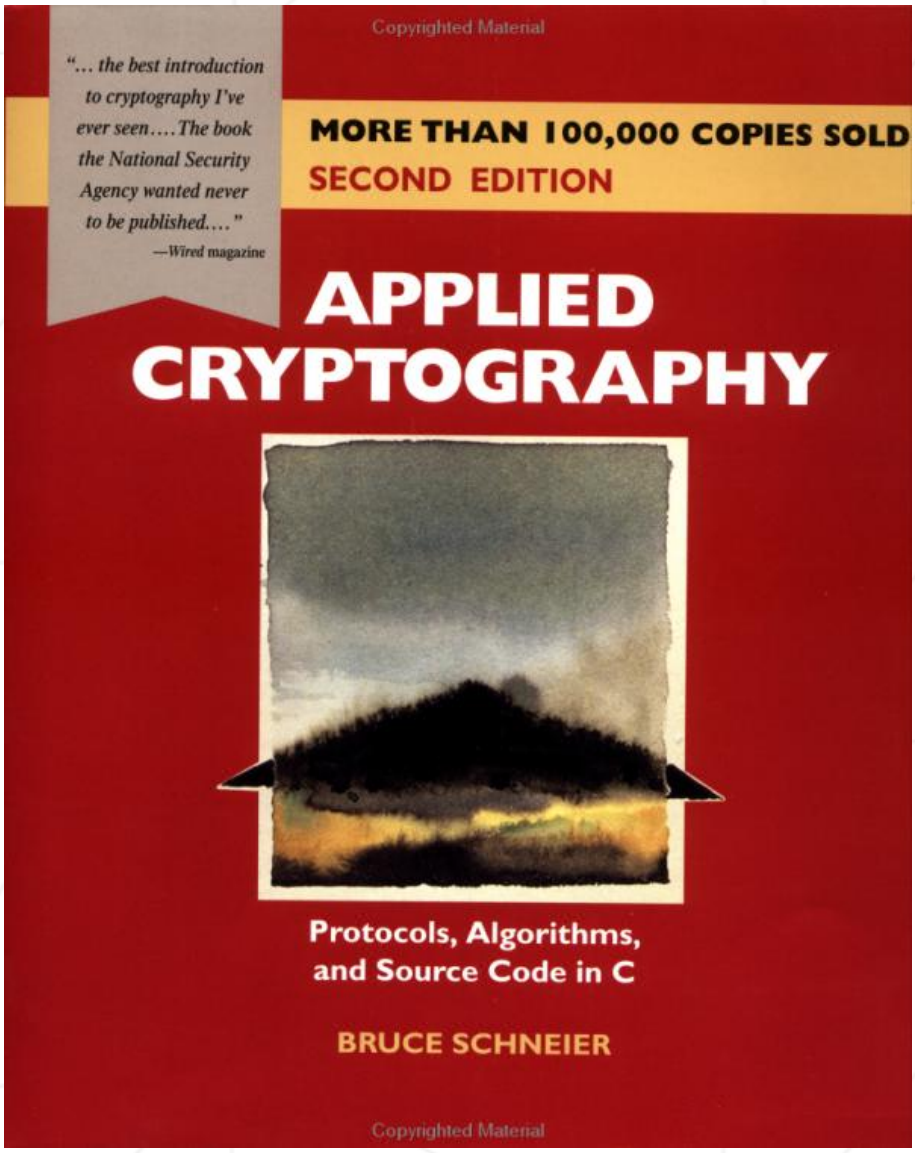
Modern cryptography is built using more complex techniques. Rather than substituting characters within an alphabet, `XOR` is used to manipulate data on the binary level.

- `plaintext XOR key = ciphertext`
- `ciphertext XOR key = plaintext`
- `plaintext XOR ciphertext = key`

However, while a simple `XOR` against a key will stop your kid sister from reading your files, it won't stop a cryptanalyst for more than a few minutes. Modern algorithms use `XOR` in combination with two additional techniques: `Confusion` and `Diffusion`.

- `Confusion` obscures the relationship between the plaintext and the ciphertext. A simple way to do this is through direct substitution with a key, as we saw before with the Caesar Cipher.
- `Diffusion` dissipates the redundancy of the plaintext by spreading it over the ciphertext. A simple way to do this is through transposition, also called permutation, or swapping the character positions of the plaintext.

An excellent resource if you would like to explore cryptographic theory and application in depth is `Applied Cryptography` written by Bruce Schneier.



Reading this book is not required to solve the following exercises.
(Kind of like how a jetpack isn't necessary to climb Mount Everest)

Chapter III

Objectives

The project `ft_ssl` opens the path to the `Encryption and Security` branch of the skill tree. You will have to recode from scratch some security technologies you may have already been using.

You will want to plan out the structure of your executable before you begin because you will build onto it in later security projects. It is of vital importance that your code is modular so it easy to re-use and add on to.

This series of projects will help you to understand the underlying structure and protocol of security technologies, and solidify through practice your knowledge of bitwise operations and data manipulation.

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.
- Your project must be written in accordance with the Norm.
- 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'll have to submit an author file at the root of your repository. You know the drill.
- You are allowed the following functions:
  - `open`
  - `close`
  - `read`
  - `write`
  - `getpass` (or `readpassphrase` or `getch`)
  - `malloc`
  - `free`
- 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 in the channel `#ft_ssl`

Chapter V

Mandatory Part

OpenSSL is a cryptographic toolkit library written C that is used to secure communications over computer networks. It implements the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) network protocols to protect against eavesdropping and verify identities during network communication.

You must create a program named `ft_ssl` that will recreate part of the OpenSSL functionality. Take care to do it well; the other projects in this series will build directly onto this program.

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

For this project, you will implement the options of the `base64` data encoding and the `Data Encryption Standard (DES)` encryption algorithm.

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

Standard commands:

Message Digest commands:

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

You will add Standard and Message Digest commands later.

You do not need to code any functionality into this program except what is explicitly asked for.

You will need to be able to input and output on STDIN/STDOUT and also files as necessary for the commands, but not for the ft_ssl itself.

For the Cipher commands, des should alias to des-ecb in ft_ssl.

Do be aware that des is an alias for des-cbc in OpenSSL.

man openssl

V.0.1 All your base are 64

`Base64` is a binary-to-text encoding scheme that translates binary data in ASCII format into a base 64 character system. Yes, it really is that simple.

Create a command for your `ft_ssl` program that we can use to encode and decode in `base64` representation:

```
 > echo toto | ft_ssl base64 -e
dG90bwo=
 >
 > echo dG90bwo= | ft_ssl base64 -d
toto
```

Your program must be able to encode and decode to the existing `base64` character set, so it will cooperate with existing technologies.

```
 > echo foobar | ft_ssl base64 -e
Zm9vYmFyCg==
 >
 > echo Zm9vYmFyCg== | base64 -D
foobar
```

You must implement the following flags: `-e`, `-d`, `-i`, and `-o`.

`man base64`

`OpenSSL base64 puts a newline character after every 64 characters.`
`You must be able to parse the whitespace. Recreating it is optional.`

V.0.2 Doesn't Escape Surveillance

The Data Encryption Standard (DES) is a symmetric-key block cipher. It was developed in the early 1970's at IBM and was broken in early 1999. Even though it is now regarded as insecure, versions of it (including Triple DES) are still unbroken and considered secure.

You must also add a command to your `ft_ssl` program that will encrypt with the `DES` algorithm. It must be written to the standard for compatibility, regardless of the weak security (though you may, and should, improve upon that later).

You must include the following flags:

- `-e`, encrypt mode (default)
- `-d`, decrypt mode
- `-a`, base64 encode/decode
- `-k`, key in hex is the next arguement. Behave like `openssl des -K` not `openssl des -k`
- `-i`, input file for message
- `-o`, output file for message

If flags are not provided, be prepared to read/write from/to the console.

```
> ft_ssl des
enter des key in hex: 0123FF42dd1f0fcc
```

You may hide the key entry like a password would be, or not. Your choice.

Keys should be 64 bits long. If a key is too short, pad it with zeros until it is the necessary length. For example, hex key `FF12CD` becomes `FF12CD0000000000`. Hex key `FF1` becomes `FF10000000000000`. Longer keys are truncated with the remainder discarded. Secure key generation will be handled in a later project.

Messages that are not a multiple of the block size must be padded with the size difference byte, the same as `OpenSSL`.

Your DES must operate in ECB mode.

You may use the -nopad flag with OpenSSL while testing to make sure your algorithm is correct before checking the padding.

```
> echo "foo bar" | openssl des-ecb -K 6162636461626364 -a -nopad
YZF3QKaabXU=
>
```

V.0.3 Correcting Broken Ciphers

The previous exercise had your `DES` algorithm operate in `ECB` mode, meaning each encrypted block was concatenated to the end of the block before it. For this next part, you must implement `CBC` mode, or `Cipher Block Chaining`. Rather than simply concatenating the next block, each block is also XOR'd with the block before it.

You must add the extra flag `-v` to match the `openssl des -iv` option for the Initialization Vector.

```
 > echo "one deep secret" | ./ft_ssl des-cbc -a
enter des key in hex: 6162636461626364
enter initial vector: 0011223344556677
zqYWONX68rWNxl7msIdGC67Uh2HfVEBo
 >
 > echo "one deep secret" | ./ft_ssl des-cbc -a -k 6162636461626364 -v 0011223344556677
zqYWONX68rWNxl7msIdGC67Uh2HfVEBo
 >
 > echo "zqYWONX68rWNxl7msIdGCw==" | openssl des-cbc -d -a -K 6162636461626364 -iv 0011223344556677
one deep secret
 >
```

Your DES-CBC must operate in the real CBC mode, where the IV is modified after each block.

You must be able to encrypt and decrypt all modes with excecutables made by other students and the OpenSSL executable.

Chapter VI

Bonus part

You may expand upon the `ft_ssl` program even more for bonus. The suggested expansion at this level is to add `Triple DES` encryption.

`OpenSSL uses CBC mode for their triple DES.`

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

Standard commands:

Message Digest commands:

Cipher commands:
base64
des
des-ecb
des-cbc
des3
des3-ecb
des3-cbc
 >
```

As you should expect by now, the bonus will not be considered unless the mandatory part is complete and perfect.

Chapter VII

Turn-in and peer-evaluation

Submit your code to your `Git` repository as usual. Only the work in the repository will be considered for the evaluation. Any extraneous files will count against you unless justified.