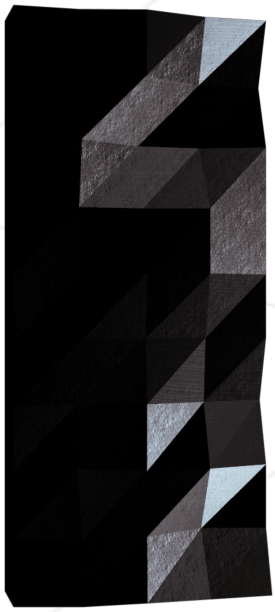


ft_ssl [base64] [des]

The real fun: symmetric encoding and encryption

Isaac Rhett irhett@student.42.us.org
42 staff staff@42.fr

Summary: This project is a continuation of the previous encryption project. 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

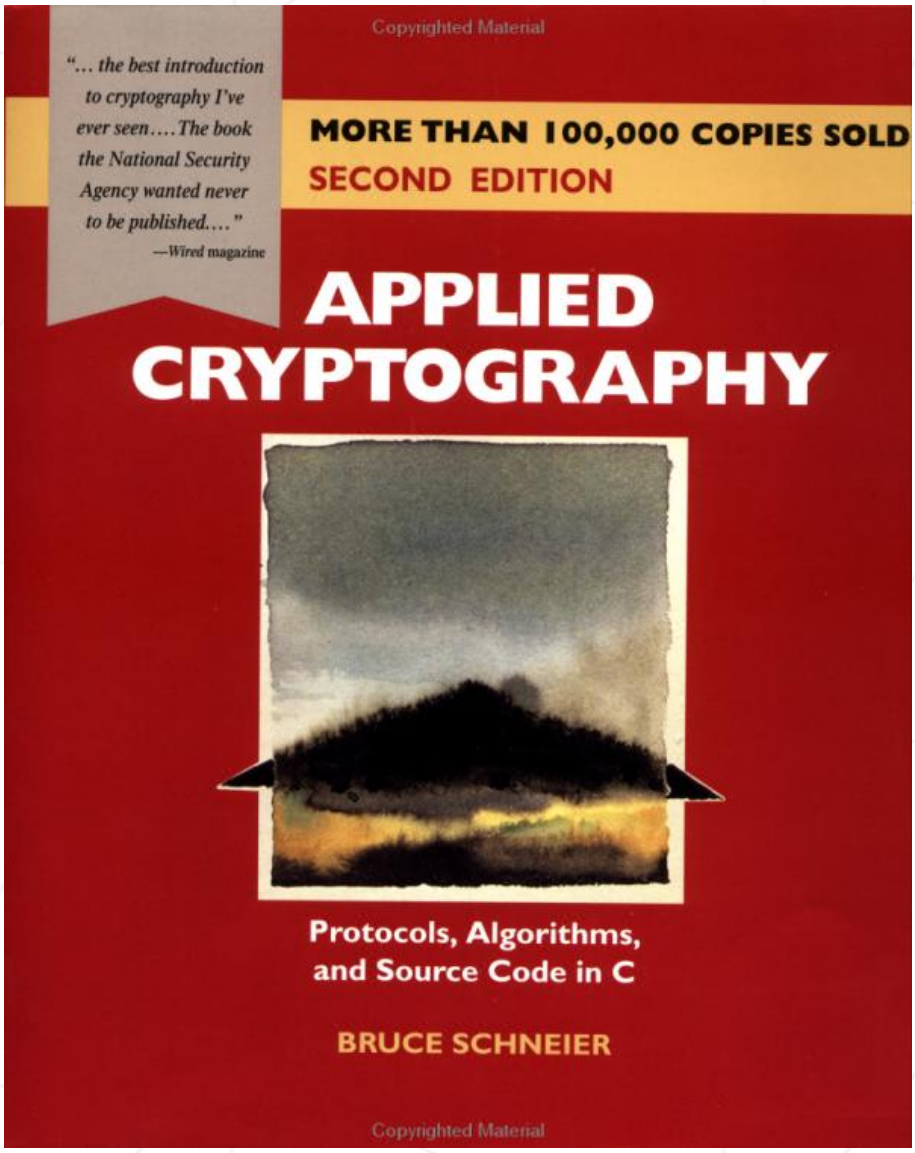
Modern symmetric (single-key) ciphers are built using more complex techniques. Rather than substituting characters within an alphabet, `XOR` is used to manipulate data on the binary level.

- `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

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

This series of projects will help you to understand the underlying structure and process of symmetric encryption algorithms, 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:
  - `getpass` (or `readpassphrase` or `getch`)
  - 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 in the slack channel `#ft_ssl`

Chapter V

Mandatory Part

You must build upon the `ft_ssl` executable that you created in the previous exercise. 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:
md5
sha256

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

You will add Standard commands later.

des is an alias for des-cbc in OpenSSL.

man openssl

V.0.1 All your base are belong to 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
 > echo "d G9 bwo  =" | ./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 for `base64`:

- `-d`, decode mode
- `-e`, encode mode (default)
- `-i`, input file
- `-o`, output file

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

`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 a modified `Triple DES`) will make the NSA have to get one of their cryptanalysts to actually look at it.

You must also add commands to your `ft_ssl` program that will encrypt with the `DES` algorithm in different block cipher modes of operation. It must mimic the behavior of `OpenSSL` for compatibility, regardless of the weak security and glaring flaws (which you will soon learn about).

You must include the following flags for `DES`:

- `-a`, decode/encode the input/output in base64, depending on the encrypt mode
- `-d`, decrypt mode
- `-e`, encrypt mode (default)
- `-i`, input file for message
- `-k`, key in hex is the next arguement.
  (Behave like `openssl des -K` not `openssl des -k`)
- `-o`, output file for message
- `-p`, password in ascii is the next argument.
  (Behave like a modifiedd `openssl des -pass` not like `openssl des -p` or `-P`)
  (A verbose explanation is given in the next section)
- `-s`, the salt in hex is the next argument.
  (Behave like `openssl des -S`)
- `-v`, initialization vector in hex is the next argument.
  (Behave like `openssl des -iv` not `openssl des -v`)

If flags are not provided, be prepared to read/write from/to the console for the missing parameters similar to the behavior of `OpenSSL`.

`The flags differ from OpenSSL partly for aesthetic reasons, but mostly because OpenSSL doesn't follow either the Unix or GNU Norms for single- and multi-character flags. Really, it's quite gross.`

V.0.3 Pretty Awful Security, Seriously

When the user does not have a cryptographically secure key, a new one must be created. This is why when a key is not provided, OpenSSL asks the user for a password. The key is generated using a Password-Based Key Derivation Function, or PBKDF.

```
 > openssl des-ecb
enter des-ecb encryption password:
```

To see the key that is generated from the password, you can use the -P flag.

```
 > openssl des-ecb -P
enter des-ecb encryption password:
Verifying - enter des-ecb encryption password:
salt=EDCFEEFCA1850351
key=914A103B0CE0A235
 >
 > openssl des-ecb -P
enter des-ecb encryption password:
Verifying - enter des-ecb encryption password:
salt=3EB317A13C39A7D8
key=DD334A3DE9C4C449
```

The password given for both of the above examples was password, but did you notice the keys are different? This is because the salt is purely random data that changes every time. (man 4 random)

To make your own keys from passwords, you will have to implement your own PBKDF. You must read data from STDIN (using your choice of the allowed functions above to prevent it showing up on the terminal) or read it as a flag. Your flag will of course be much cleaner than that of OpenSSL.

```
> openssl des-ecb -pass "pass:MySuperSecurePassword"
```

```
> ./ft_ssl des-ecb -p "MySuperSecurePassword"
```

RFC 2898 sets PBKDF2 as the standard and RFC 8018 still endorses it.

Another hint: re-creating OpenSSL's PBKDF will not be very hard: it's not really secure. Really.

V.0.4 Everything Can Break

The DES algorithm works by splitting the plaintext into blocks of 64 bits (8 chars) in length, and performing a set of operations on those blocks with a 64-bit key. The process it performs on each block is the same, and by itself this is simply called ECB mode.

The requirements of ECB mode are:

- A 64-bit long key. If a key is too short, pad it with zeros.

  For example, hex key FF12CD becomes FF12CD0000000000.

  Hex key FF1 becomes FF10000000000000.

  Longer keys are truncated with the remainder discarded.

- A 64-bit long block. If a block is too short, pad it with the size difference byte padding scheme, the same as OpenSSL.

To avoid all that password nonsense, you can input a key directly with the -k flag.

```
> ./ft_ssl des-ecb -k 01020304FEFDFCFB
```

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=
```

Initialization Vector is not used for ECB mode, but can still be taken and not used.

V.0.5 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 `BC` mode, or `Cipher Block Chaining`. Rather than simply concatenating the next block, each block is also XOR'd with the block before it.

This is where the -v flag comes in for the Initialization Vector.

```
 > echo "one deep secret" | ./ft_ssl des-cbc -a -k 6162636461626364 -v 0011223344556677
zqYWONX68rWNxl7msIdGC67Uh2HfVEBo
 >
 > echo "zqYWONX68rWNxl7msIdGC67Uh2HfVEBo" | 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

There are several bonuses that can be earned for this project. One easy expansion is to include three-key `Triple DES` (in `E-D-E`, of course). Be careful, your `PBKDF` will be a little trickier than it is above because you need to generate a longer keystream.

`OpenSSL uses CBC mode for their Triple DES by default`

You may also include other block cipher modes of operation, such as:

- `CFB`
- `CTR`
- `OFB`
- `PCBC`
- And many more...

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

Standard commands:

Message Digest commands:
md5
sha256

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

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.