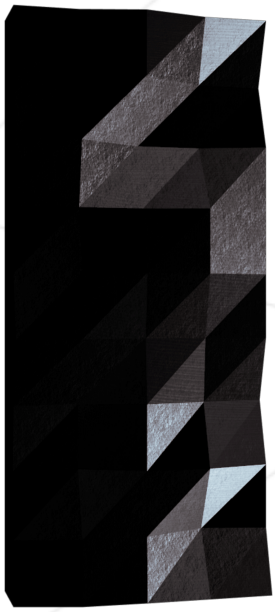

ft_ssl [md5] [sha256]

An introduction to cryptographic hashing algorithms

Summary: This project is the gateway to the encryption branch. You will recode part of the OpenSSL program, specifically the MD5 and SHA-256 hashing algorithms.

Contents

Chapter I

Foreword

Enjoy these quality mashups by good ol' Neil. *(images are links)*

Be careful, these songs have extremely dangerous memetic properties. A paper bag is your only line of defense. It will not help.

Oh whoops this was supposed to be about *Hashing* not *Mashing*.
My bad, 'H' and 'M' are so close I visually can't tell them apart.
Looks like I really need new glasses, or a good hashing algorithm.
Oh well... what's done is done.

"Hash up" and "Mash up" have a 5 character difference of 1 byte with only 2 bits changed, but when hashed with md5 their outputs differ by more than 37%

Chapter II

Introduction

This is what Wikipedia has to say on the subject of cryptographic hash functions:

A cryptographic hash function is a special class of hash function that has certain propertiesi which make it suitable for use in cryptography. It is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size. It is designed to also be a `one-way function`, that is, a function which is infeasible to invert. The only way to recreate the input data from an ideal cryptographic hash function's output is to attempt a brute-force search of possible inputs to see if they produce a match, or use a rainbow table of matched hashes. Bruce Schneier has called one-way hash functions "the workhorses of modern cryptography". The input data is often called the message, and the output (the hash value or hash) is often called the message digest or simply the digest.

The ideal cryptographic hash function has five main properties:

- it is deterministic, so the same message always results in the same hash
- it is quick to compute the hash value for any given message
- it is infeasible to generate a message from its digest except by trying all possible messages
- a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
- it is infeasible to find two different messages with the same hash value

Cryptographic hash functions have many information-security applications, notably in `digital signatures`, message authentication codes (MACs), and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as `checksums` to detect accidental data corruption. Indeed, in information-security contexts, cryptographic hash values are sometimes called (digital) fingerprints, checksums, or just hash values, even though all these terms stand for more general functions with rather different properties and purposes.

Chapter III

Objectives

This is the first `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 want to plan out the structure of your executable before you begin, because you will build directly onto it in later security projects. It is of vital importance that your functions are modular, your code is easy to re-use and adding onto the executable doesn't require rewriting the program.

`Re-read that last sentence again. I'm serious, do it.`

This project will focus specifically on cryptographic hashing algorithms to solidify your understanding of bitwise operations, integer overflow, and one-way functions. A subplot to this project is to emphasize writing clean code that can be efficiently extended.

`If you have ever downloaded a file of significance from the internet (an operating system installation image, for example), you have probably noticed a section with either SHA-1 or MD5 followed by gibberish. If you were smart, you have looked up what the string meant and verified your download with the given algorithm to verify the hash. If you haven't, I advise you do some research to learn more about digital fingerprinting so you can better protect yourself from malicious downloads.`

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:
  - `open`
  - `close`
  - `read`
  - `write`
  - `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 using a function or lib that do your work because you are lazy is not)
- You can ask your questions in the slack 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.

For this project, you will implement the `md5` and `sha256` hashing functions.

```
 > ./ft_ssl
usage: ft_ssl command [command opts] [command args]
 >
 > ./ft_ssl foobar
ft_ssl: Error: 'foobar' is an invalid command.

Standard commands:

Message Digest commands:
md5
sha256

Cipher commands:
```

You will add Standard and Cipher commands later.
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.

`man openssl`

*"**Use a dispatcher** buu-rp **function pointers, Morty**"*
-actual Rick Sanchez quote

V.0.1 Mangled Data 5

`MD5` is a cryptographic hash function. It is short for "Message Digest 5". It has found to contain several vulnerabilities, but can still be safely used as a checksum to verify data integrity against unintentional corruption.

You must create the `md5` command for your `ft_ssl` program. You must include the following flags:

- `-p`, echo `STDIN` to `STDOUT` and append the checksum to `STDOUT`
- `-q`, quiet mode
- `-r`, reverse the format of the output.
- `-s`, print the sum of the given string

Emulate the behavior of the `md5` executable (since it is more versatile than `OpenSSL`). If flags are not provided, be prepared to read/write from/to the console.

Example usage is on the next page.

```
man md5
```

```
 > echo "pickle rick" | openssl md5
c5e433c1dbd7ba01e3763a9483e74b04
 >
 > echo "pickle rick" | md5
c5e433c1dbd7ba01e3763a9483e74b04
 >
 > echo "pickle rick" | ./ft_ssl md5
c5e433c1dbd7ba01e3763a9483e74b04
 >
 > echo "Do not pity the dead, Harry." | ./ft_ssl md5 -p
Do not pity the dead, Harry.
2d95365bc44bf0a298e09a3ab7b34d2f
 >
 > echo "Pity the living." | ./ft_ssl md5 -q -r
e20c3b973f63482a778f3fd1869b7f25
 >
 > echo "And above all," > file
 > ./ft_ssl md5 file
MD5 (file) = 53d53ea94217b259c11a5a2d104ec58a
 > ./ft_ssl md5 -r file
53d53ea94217b259c11a5a2d104ec58a file
 >
 > ./ft_ssl md5 -s "pity those that aren't following baerista on spotify."
MD5 ("pity those that aren't following baerista on spotify.") = a3c990a1964705d9bf0e602f44572f5f
 >
 > echo "be sure to handle edge cases carefully" | ./ft_ssl md5 -p file
be sure to handle edge cases carefully
3553dc7dc5963b583c056d1b9fa3349c
MD5 (file) = 53d53ea94217b259c11a5a2d104ec58a
 >
 > echo "some of this will not make sense at first" | ./ft_ssl md5 file
MD5 (file) = 53d53ea94217b259c11a5a2d104ec58a
 >
 > echo "but eventually you will understand" | ./ft_ssl md5 -p -r file
but eventually you will understand
dcdd84e0f635694d2a943fa8d3905281
53d53ea94217b259c11a5a2d104ec58a file
 >
 > echo "GL HF let's go" | ./ft_ssl md5 -p -s "foo" file
GL HF let's go
d1e3cc342b6da09480b27ec57ff243e2
MD5 ("foo") = acbd18db4cc2f85cedef654fccc4a4d8
MD5 (file) = 53d53ea94217b259c11a5a2d104ec58a
 >
 > echo "one more thing" | ./ft_ssl md5 -r -p -s "foo" file -s "bar"
one more thing
a0bd1876c6f011dd50fae52827f445f5
acbd18db4cc2f85cedef654fccc4a4d8 "foo"
53d53ea94217b259c11a5a2d104ec58a file
ft_ssl: md5: -s: No such file or directory
ft_ssl: md5: bar: No such file or directory
 >
 > echo "just to be extra clear" | ./ft_ssl md5 -r -q -p -s "foo" file
just to be extra clear
3ba35f1ea0d170cb3b9a752e3360286c
acbd18db4cc2f85cedef654fccc4a4d8
53d53ea94217b259c11a5a2d104ec58a
```

The error message syntax does not have to match perfectly so long as the error message itself is sensible. Use your brain!

V.0.2 Space Human Adventures 256

SHA-256 is one of the functions in the SHA (Secure Hash Algorithms) family. The SHA algorithms were designed by the NSA. SHA-512 is currently used in part of the process to authenticate Debian software packages.

You must create the sha256 command for your ft_ssl program. You must include the following flags:

- -p, echo STDIN to STDOUT and append the checksum to STDOUT
- -q, quiet mode
- -r, reverse the format of the output.
- -s, print the sum of the given string

The behavior should be exactly the same as the command above, with the only difference being the hash algorithm in question.

```
 > echo "https://www.youtube.com/watch?v=w-5yAtMtrSM" > big_smoke_order_remix
 > ./ft_ssl sha256 -q big_smoke_order_remix
a8dc621c3dcf18a8a2eddae1845e8e5f6498970a867056ac5f7121ac3d66cfd9
 >
 > openssl sha -sha256 big_smoke_order_remix
SHA256 (big_smoke_order_remix) = a8dc621c3dcf18a8a2eddae1845e8e5f6498970a867056ac5f7121ac3d66cfd9
 >
 > ./ft_ssl sha256 -s "wubba lubba dub dub"
SHA256 ("wubba lubba dub dub") = 23a0944d11b5a54f1970492b5265c732044ae824b7d5656acb193e7f0e51e5fa
```

man sha256 will not work on the macs.

You can use shasum for testing.

Google will have the BSD man pages for sha256.

Chapter VI

Bonus part

There are several bonuses that can be earned for this project.
An easy one is to parse commands from `STDIN` the same way that `OpenSSL` does.
You could also include additional hashing algorithms in your `ft_ssl` executable. The bonus algorithms are expected to have the same flag functionality as above.

`SHA-512 is a good option; Whirlpool is a better one.`

`A hash function that is weaker than MD5 will not grant a bonus.`

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.

I almost got arrested at the doctors when I ran a checksum on my prescription, but I was let off because the hash was for medicinal purposes.