Network Security Project
Implementation Details

Nicholas Capo - Nicholas.Capo@Gmail.com

March 15, 2012

Contents

1 Requirements Analysis

1.1 Language and Binary

1. The program shall be referred to herein as *Viper*
2. The overall design shall follow the description in [2]
3. One version of the program shall be produced using the C language
4. One version of the program shall be produced using the Python language

1.2 Binaries

1. Each version shall be compiled into two binaries (`viper` and `viperBlockTest`) with the following usage:
   (a) `viper [ -h | --help ] [ -e | -d | --encrypt | --decrypt ] [ -t | --threads NUM ] [ -k | --key KEY ]`
   (b) `viperBlockTest [ -h | --help ] [ -e | -d | --encrypt | --decrypt ] [ -k | --key KEY ] input_block`

1.3 Modules

1. Each implementation shall be broken into at least three modules: (See 1.6 for details of the threading requirements)
   (a) a single–threaded `main()`
   (b) a multi–threaded `main()`
   (c) a `viperCrypt` module, containing the implementation of the cipher specification itself.

1.4 Input/Output

1. `viper` shall expect input on `stdin`, and generate output on `stdout`
2. `viperBlockTest` shall expect a single block of 32 hexadecimal values as the last argument on the command line
3. `viper` shall be the general case of `viperBlockTest` and shall encrypt or decrypt until reaching end–of–input
4. All errors and help texts shall be written to `stderr`

1.5 Compatibility

1. Each version of `viper` shall be ciphertext compatible with the reference implementation of `Serpent` [1, 3]

1.6 Threading

1. Each version of `viper` shall implement a single–threaded mode

2. Each version of `viper` shall implement a multi–threaded mode, using 32 threads

2 Design

2.1 Overview

The software shall be designed using a primarily functional approach. The core of the algorithm shall be created in a module named `viperCrypt`. Normal interaction with the module shall occur by calling the `crypt()` function (See Dataflow Diagram) and passing the user key, the plain– or cipher–text, and a flag, which indicates whether to encrypt or decrypt. The `crypt()` function shall return the result of the encryption or decryption.

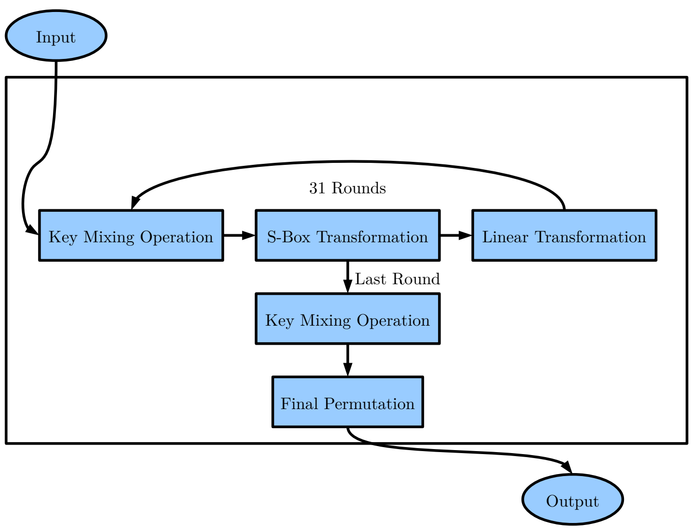

Figure 1: Dataflow Diagram

2.2 Multi–Threading

The multi–threaded version of `Viper` shall be implemented as 32 threads (See Threaded Dataflow Diagram), where each thread consists of `viperCrypt.crypt()` operating on a separate block of input data.

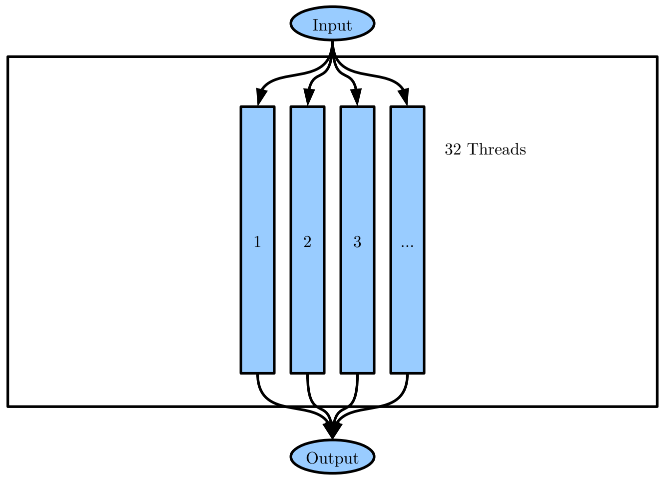

Figure 2: Threaded Dataflow Diagram

3 Implementation Results

3.1 Environment

The implementation was constructed and tested using the following environment on an x86 architecture:

- Ubuntu Linux (version 10.04.4 LTS)
- Debian Linux (version Testing/“Weezy”)
- Python (version 2.6.5)
- GCC (version 4.4.3)
- GNU Make (version 3.81)

It is expected that the implementation will be compatible with any platform that runs Python and/or C.

3.2 Source Files

3.2.1 C

- `sbox.h`
- `viperBlockTest.c`

- viperCrypt.c
- viper.c

3.2.2 Python

- sbox.py
- viperBlockTest.py
- viperCrypt.py
- viper.py

3.3 Internal Dependencies

Each version of viper depends on the viperCrypt module with in turn depends on the sbox module.

3.4 External Dependencies

3.4.1 C

Only the standard C libraries were used.

3.4.2 Python

Each of the following Python modules were imported into one or more source files:

- argparse
- sys
- print_function [1]

3.5 Build Instructions

3.5.1 C

Using the provided Makefile should be sufficient. However the following commands may be used as well:

- gcc -Wall viper.c -o viper.exe
- gcc -Wall viperBlockTest.c -o viperBlockTest.exe

[1] This function was imported from the future module to provide Python 3.x printing features

3.5.2 Python

No building is necessary, all required compilation will occur as a result of running `python viper.py`.

4 Test Methodology

4.1 Unit Tests

1. Unit tests, ad-hoc tests, and other small tests shall be used to confirm the basic operation of functions etc.

4.2 Single Block Acceptance Tests

1. `viperBlockTest` shall be used in conjunction with the *Known Answer Test*, *and Monte Carlo Test* in [1] to confirm the correctness of the simple cipher implementation.

4.3 Multi–Block Acceptance Tests

1. The single– and multi–threaded versions of `viper` shall be used in conjunction with the reference Implementations in [1, 3] to confirm the correctness of the complete cipher implementation, and that no errors have been introduced in the multi–threaded implementation.

4.4 Speed Tests

1. The single– and multi–threaded versions of `viper` shall be used to encrypt and decrypt files of various sizes and the encryption and decryption times recorded for comparison.

2. The following Speed Tests shall be used:

   (a) A zero–filled file in the following sizes

      i. 1B
      ii. 32B
      iii. 100B
      iv. 500B
      v. 1KB
      vi. 32KB
      vii. 100KB
      viii. 500KB
      ix. 1MB
      x. 32MB
      xi. 100MB

xii. 500MB
xiii. 1GB

(b) Randomly generated files in the following sizes

i. 1B
ii. 32B
iii. 100B
iv. 500B
v. 1KB
vi. 32KB
vii. 100KB
viii. 500KB
ix. 1MB
x. 32MB

3. Each test shall be run no less than three times and the results averaged.

References

[1] Ross Anderson, Eli Biham, and Lars Knudsen. *Full submission package, which contains the algorithm specification, a reference implementation in C, an optimised implementation in C and an optimised implementation in Java*. [Online; accessed 18-February-2012]. URL: http://www.cl.cam.ac.uk/~rja14/Papers/serpent.tar.gz (cit. on pp. 2, 6).

[2] Ross Anderson, Eli Biham, and Lars Knudsen. *Serpent: A proposal for the Advanced Encryption Standard*. [Online; accessed 18-February-2012]. URL: http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf (cit. on p. 2).

[3] Frank Stajano. *Serpent reference implementation*. [Online; accessed 26-January-2012]. URL: https://www.cl.cam.ac.uk/~fms27/serpent/ (cit. on pp. 2, 6).