AES Crypto

Date: 9/28

Requirements

Languages:

Use latest version

- Python 3, C/C++ (Preferred)
- Rust, Haskell, Go, Swift, Javascript, C# (Net Core) (Allowed)
- Java (Caution) - char and bytes get confused and therefore most Java implementations come out wrong.
- Choose your language - please ask

Grading:

- Follow Style Guide: 10%I will **harshly** grade based off the style guide listedIf any of these style guides have tools, use them.
  - Rust: rustfmt
  - Go: gofmt
  - Javascript: Airbnb
  - Swift: Google
  - Haskell: Ganeti
  - Python 3: pep8
  - Java: Google
  - C++: Google
  - C#: Microsoft
- Algorithm: 75%
  - Encrypt text file - 50%
  - Encrypt any file (image, song, etc..) - 25%
  - No advanced math APIs or encryption libraries allowed (e.g numpy, crypto, etc...)
- README: 15% (**Must** be .md, or .pdf)

How to turn in

You have two options

1. Github repo (if you want to make it private, add @michaelasper as contributor)
2. Zip file with the `.git` dir intact

This algorithm is fairly popular, so I will require you to use version control so I can see the progress of your work and to ensure you are not cheating. Meaningful and often commit messages would be extremely helpful. It also helps me debug your progress during office hours.

Background

AES uses repeat cycles or "rounds". There are 10, 12, or 14 rounds for keys of 128, 192, and 256 bits,respectively. The input text is represented as a 4 x 4 array of bytes. The key is represented as a 4 xn array of bytes, where n depends on the key size.

Each round of the algorithm consists of four steps:

subBytes: for each byte in the array, use its value as an index into a fixed 256-element lookup table,and replace its value in the state by the byte value stored at that location in the table. You can find the table and the inverse table on the web.

shiftRows: Let Ri denote the ith row in state. Shift R0 in the state left 0 bytes (i.e., no change); shiftR1 left 1 byte; shift R2 left 2 bytes; shift R3 left 3 bytes. These are circular shifts. They do not affect the individual byte values themselves.

mixColumns: for each column of the state, replace the column by its value multiplied by a fixed 4 x4 matrix of integers (in a particular Galois Field). This is the most complex step.

addRoundkey: XOR the state with a 128-bit round key derived from the original key K by a recursiveprocess.

Assignment

Your assignment is to implement AES-128 and AES-256 in the same program. We will be following the (Click here ->>>>) AES standard. Use Electronic Code Book (ECB) mode. The program should run like this (or similar depending on the language e.g. python):

```
./program --keysize $KEYSIZE --keyfile $KEYFILE --inputfile $INPUTFILE
--outputfile$OUTFILENAME --mode $MODE
```

keysize: Either 128 or 256 bits

keyfile: Should take in a keyfile that fits one of the specified sizes. To generate a key, use: `head -c 16 < /dev/urandom > key` (this generates 16 random bytes ~ 128 bits)

inputfile: This should be able to handle any file and any size (padding might be required). All we care about are the bytes of the file. To get the byte representation of a file use `xxd`

outpfile: what to save the result

mode: encrypt or decrypt

How do files work?

This is usually the most confusing part of the assignment.

When you have a file that is filled with "Hello World" saved in `inputfile`

What you're actually encrypting is not the string inside the textfile, but the bytes of the textfile (this is important)

To see the bytes of a file, you can run `xxd inputfile` on any POSIX compatible shell

During grading, I will running `xxd outputfile` to see if the encryption worked.

For example, if your inputfile is just a bunch of null bytes (i.e. when I run `xxd inputfile` I should see all 0s), then rounding to the first 16 bytes, the outputfile should be `66E94BD4EF8A2C3B884CFA59CA342B2E` i.e. this is what you see when running `xxd outputfile`

README Grading

AES is pretty popular algorithm, therefore to ensure that you wrote the code, I will need you to explain every step of the algorithm and how your code performs this. I will also need detailed instructions on how to run your program if it differs from above. If you have any worries about your README, either of the TAs will look over it ahead of time for you.*

* do not wait til the last second

Debugging

AES-128

```
(xxd keyfile) key: 00000000000000000000000000000000

(xxd inputfile) plaintext: 00000000000000000000000000000000

(xxd outputfile) ciphertext: 66E94BD4EF8A2C3B884CFA59CA342B2E
```

AES-256

```
(xxd keyfile) key: all 0s

(xxd inputfile) plaintext: 00112233445566778899AABBCCDDEEFF

(xxd outputfile) ciphertext: 1C060F4C9E7EA8D6CA961A2D64C05C18
```

Extra Credit (max 7%)

- Implement CBC Mode (+3.5%)
- Extra creative implementation (+3.5%)