AES Crypto

Due: Friday April 20th 23:59:59

Specs

- Java, C/C++, Python 3, Rust, Go, or Haskell
- Follow style guide: 10 points
  - Google Style Guide: Java, C/C++, Python
  - Rustfmt: Rust
  - gofmt: Go
  - Ganeti: Haskell
- Algorithm Implementation: 70 points
- README: 15 points

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 x n 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); shift R1 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 x 4 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 recursive process.

Assignment

Your assignment is to implement AES-128 and AES-256 in the same program. We will be following the 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 is the bytes of the file. To get the byte representation of a file use `xxd`.
    - For example, a file that says "Hello World" will actually have a byte representation of `4865 6c6c 6f20 576f 726c 640a`
- outpfile: what to save the result
- mode: encrypt or decrypt

The readme should explain how your code works inline with the algorithm.

Debuging

AES-128

```
key: 00000000000000000000000000000000

plaintext: 00000000000000000000000000000000

ciphertext: 66E94BD4EF8A2C3B884CFA59CA342B2E
```

AES-256

```
key: all 0s

plaintext: 00112233445566778899AABBCCDDEEFF

ciphertext: 1C060F4C9E7EA8D6CA961A2D64C05C18
```

Extra credit

- Implement CBC mode