

# CS 3110 Group Project Design

**Names:** Alex Jiang, Karol Styrzula, Jaryung (Jane) Kim

**NetIDs:** alj55, kss238, jk2498

**Meeting Location:** Carpenter Hall group rooms

**Meeting Frequency/Schedule:** MWF 4:30-6:00ish

**Other methods of communication:** Git, Email, GroupMe, Facebook Messenger

## Core vision for our system

We plan to create an OCaml implementation of the Advanced Encryption Standard (AES), a NIST specified symmetric encryption standard based on the Rijndael algorithm and is approved for protection of classified information. We plan to create a GUI displaying the input, output, and user preferences that will be easy to navigate.

## Key Features

- **Encryption algorithm**
- **Decryption algorithm**
- **GUI**
- **Key length choice**
- **Benchmark feature**

## Functionality

We plan to create an OCaml implementation of the Advanced Encryption Standard using the NIST Standard. The program will have a GUI where the user can select a key length for encryption, generate a random key or input a key, input text to be encrypted or decrypted, and

view or select (copy) the encrypted/decrypted output. The program will also have a feature to benchmark (check computation time) the implementation using different key lengths and compare against other software and hardware implementations.

Detailed spec here: <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>

## **System design** (modules):

Our system will be built around four different modules -- one module will contain information about the current state of the encryption process and the encryption/decryption sub functions, two modules that contain the cipher and decipher functions. There will also be a GUI module to display the input and its resulting output. If time permits, we will also create an animated tutorial of the matrix operations that are being performed as the program runs for users to learn how the AES encryption algorithm works.

Below are the names of each module and their function in the overall system:

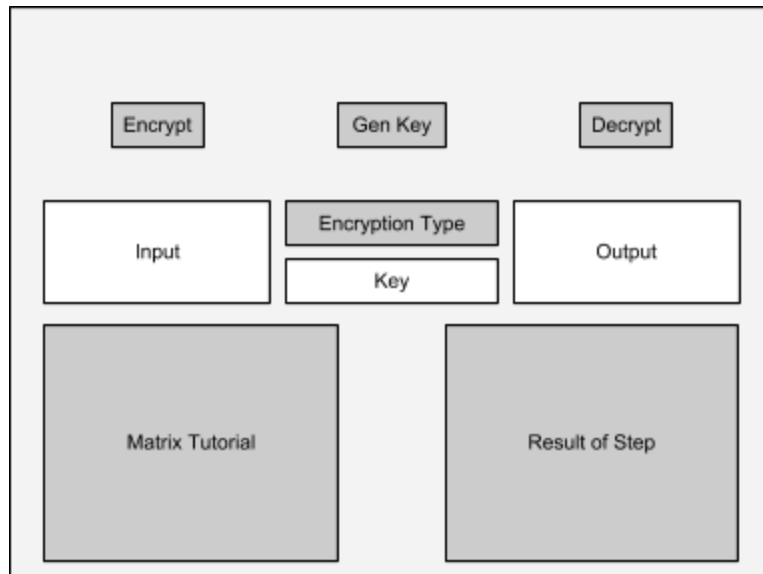
- **Main:** contains the main function called at the start of execution, does NOT take input or display output (that is the GUI's job)
- **Cipher\_AES:** a module containing a series of functions that must be executed in sequence to transform plaintext to ciphertext (both ASCII) using Cipher Key
- **Decipher\_AES:** contains functions that are inverse of those in Cipher\_AES and transforms ciphertext to plaintext using Cipher Key
- **GUI:** the user interface that collects input and displays output
  - Drop-down list for key length choice
  - Option to generate random key
- **State:** data structure that stores data and keys
  - Functions implementing matrix operations
  - Functions implementing key expansion

## Module design (signatures):

Cipher and Decipher should only have the high-level cipher and decipher functions exposed in their signatures. The only functions exposed in State should be the encryption\_type function that indicates the type of encryption (currently always AES, but we may implement other algorithms) and those necessary to update the data structure (including operations on data matrices and on keys). The only things the GUI should have exposed to the outside are functions to initialize and update it.

- See interfaces.zip
- State functions:
  - get\_data() : returns the current data encoded as a string, needed for GUI output
- Basic specifications for matrix and key cipher operations:
  - sub\_bytes() : performs substitution with SBox (substitution table storing a certain byte transformation encoded in hexadecimal)
  - shift\_rows(): performs permutation of the rows by shifting the last three rows of state by differing offsets
  - mix\_columns() : mixes the data in columns in order to make new columns
  - add\_round\_key() : performs XOR operation with the round key
- Decipher operations
  - Inverse of each of the 4 above cipher functions
- GUI operations
  - init() : initializes and opens the GUI
  - update() : takes in a state and updates the GUI accordingly
  - run\_tutorial() : runs the “matrix tutorial” animation showing the steps of encryption

## GUI Diagram



### Input

- Text field for user to type in plain text

### Output

- Cipher text output

### Encrypt

- Button that encrypts the input

### Decrypt

- Button that decrypts the input

### Gen Key

- Button to generate cryptographically secure random key for encryption

### Encryption Type

- Dropdown list for possible encryption types to select from

### Key

- Text field to input key if user supplies one or output field for Gen Key

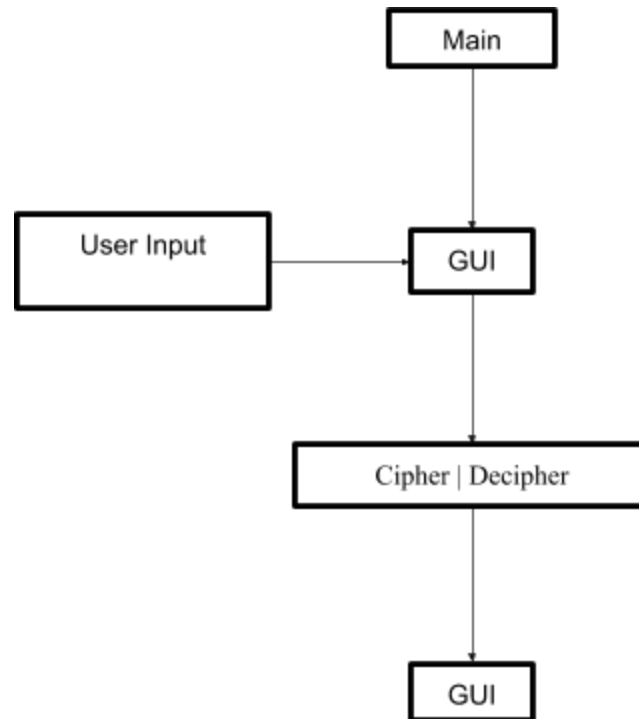
### Matrix Tutorial

- Displays a step-by-step animation of how the AES algorithm works (e.g. matrix operations), possibly include “run/pause/step” like debugger

### Results of Step

- Stores history of the results of previous encrypt / decrypt computations

**Control Flow** (simplified, technically Main is intermediate in every step)



## Data Management

What data will your system maintain?

- The system will maintain a copy of the inputted data as it goes through the encryption process
- The system will also maintain a structure containing the keys created from the inputted/generated key using key expansion.

What formats will be used for storage or communication?

- All data will be stored in the state structure accessed through the State module
- The GUI will collect user input and display encrypted/decrypted output
- Users can communicate their inputs (as strings) to the system through the GUI

What data structures do you expect to use as part of your implementation?

- An array of 2D arrays for data (represented as strings or bytes)
- A 2D array for the byte substitution table
- An array of keys (from key expansion)

**External dependencies:** What third-party libraries (if any) will you use?

- **LabGtk2** - (GUI library) is the OCaml interface for Gtk2, a toolkit for graphical applications. Will be used to create a user interface (GWindow.window) displays an input and output, a drop-down list for key length choice (GMenu.menu\_bar), and an option to generate a random key (GButton.button).
- **CryptoKit** - existing library in OCaml that contains a wide range of cryptographic primitives (e.g. random number generation); will be used to test against (both correctness and benchmarks)

## Testing Plan

- **utop** - quick tests while implementing, immediately after finishing each function
- **OUnit**
  - Black box testing:
    - Each member will create test cases for the functions that he or she is responsible for, and
    - Other group members will write additional test cases based on the specifications (preconditions) of his or her functions.
  - Glass box testing
    - Each member will be responsible for extensively testing his or her functions before moving onto a different part of the program.
  - Randomized testing

- Check that we get original strings back when we encrypt and decrypt some large number of random strings
  - AES Known Answer Test (KAT) Vectors: The NIST documentation of the AES algorithm provides a set of known ciphers with their keys and inputs that we will use to test our program.
- **Code review**
  - Each piece of code will be checked by at least 2 members and we will help each other on writing difficult functions.