# CHEKS 2.0 Architecture guidelines

# Version 1 (May 10, 2015)

The following diagram shows the key components (classes).



## Main Classes

### Engine:

This is the center piece of the architecture. It will glue the different modules together.

### Abstract classes:

Three classes are shown abstract with implementation examples. The main reason why these classes are abstract is to allow for easy testing by removing dependencies between modules.

* AbstractChaoticSystem: allows different implementations of chaotic systems
* AbstractEncrypter: allows different encryption algorithm
* AbstactCommunicator

The actual implementation of these abstract concept to be used by the Engine will be injected when instantiating the engine (increasing testability).

### User Interface:

The user interface could be built to support a one-to-many combination (1 user chats with many friends, each in individual chat rooms). To support this, a whole MVC approach is used, but limited to the user interface.

### Chaotic System:

When asked to generate a key of length N, the chaotic system forks and uses the next X states (generated by evolving the forked system with a pre-specified factor[[1]](#footnote-1)) to generate the key (X depends of the size of a state and the value of N). If asked to generate a second key, the chaotic system continues on the fork with as many evolutions as needed.

When asked to reset, the chaotic system will simply delete the fork (if any) and come back to its main branch.

When asked to evolve with factor F, the chaotic system will delete the fork and evolve its main branch with factor F.

## Workflow:

Below is an overview of the three main use cases of the application.

### Case 1: Adding a new contact to chat with.

When creating a new contact to chat with, the following information must be provided by the user through the GUI:

* The contact name
* Which encrypter to use (Alice could use Rinjdael to talk to Bob, OneTimePad to talk to Carolyn and DES to talk to Dave, as long as the recipient uses the same encrypter)
* The chaotic system used to talk to this contact (as specified by an ID or a FileName)
* The chaotic system used to listen to this contact
* The communicator used to talk to this contact (as specified by the destIP + destPort)

The GUI will fire an event and the controller will react by adding a new Contact to the Model’s contact “list”. The model will then inform its observers (GUI and Engine) that a new Contact has been added. GUI will simply add a new “TabPane” to its view.

The engine will register itself to the communicator associated with this contact (to be notified when a communication is received from the contact in an asynchronous way). Note that the receiver part of the communicator should be a singleton as we use the same (port) regardless of who is trying to connect with us.

### Case 2: user is sending a message (Alice sends to Bob)

Alice writes the message and press submit. The controller catches this event and relays it to the Model to be added to the message collection as a sent message to Bob. The model then notifies its observers (GUI and Engine) that a message is sent. The GUI simply adds this message to Bob’s chat room with the appropriate color (message sent).

The engine will:

1. Create a Communication object with that message.
2. Retrieve Bob Contact from the contact list and obtain the encrypter, the talking chaotic system and the communicator.
3. Request a keyLength from the encrypter for the specific message
4. Request a key of specific length from the chaotic system
5. Use the encrypter to encrypt the message with the key
6. Add cipher to Communication object
7. Generate message hash
8. Repeat steps 3-6 for hash
9. Ask communicator to send the message (waiting for acknowledge)
   * If ack is received: trigger evolution from chaotic system (with factor) and notify Model that message was transmitted.
   * If ack is not received: reset chaotic system and notify model of failure.

### Case 3: user is receiving a message (Bob sends to Alice)

Alice’s computer receives a communication. The receiver is the same for each of Alice’s contact, but its enclosing communicator is unique to Bob. So receiver asks enclosing communicator to notify of a communication reception. The communicator will notify the engine, identifying Bob as the sender, passing along an “hexadecimal” string.

The Engine will create a communication from the hexadecimal string. A failure results in stopping to process the communication and alerting the user of an anomaly through the UI.

In case of success, the engine will retrieve Bob’s contact from contact list and extract the listening chaotic system and the encrypter. An identity check is performed, testing if the chaotic system ID in the communication matches the ID of the listening chaotic system for Bob. A failure results in stopping to process the communication and alerting the user of an anomaly through the UI.

In case of success the engine will:

1. Request a keyLength from the encrypter for the ciphertext
2. Request a key of specific length from the chaotic system
3. Use the encrypter[[2]](#footnote-2) to decrypt the cipher message with the key
4. Compute hash of decrypted cipher
5. Obtain cipher check from communication
6. Repeat steps 1-3 for cipher check
   * If decrypted cipher check does not match the result obtained at step 4, stop and notify + reset chaotic system
   * If decrypted cipher check matches the result obtained at step 4, evolve chaotic system (with factor) and tell controller that a new message was received from Bob.

The controller will tell model to add a received message from Bob. The model will then notify its two observers:

* UI just add message with appropriate color (message received).
* Engine does noting with this type of notification.

## Notes:

* The cipher check generation behavior (hash + encrypt or other) should be injected to the engine for the prototype so we can play with different implementations.
* We still have to think of a more secure way for Alice to verify that the message she sent to Bob was indeed received by Bob, but also successfully decrypted by Bob. This mechanic should be included in the architecture. In particular, sent messages should have a different status indicating whether they have been properly acknowledged by the receiver. As long as a message has not been acknowledged, Alice cannot send another message to Bob (this would risk desynchronization).

1. The pre-specified factor used to evolve the forked chaotic system to generate key material could be an arbitrary value chosen randomly when creating the chaotic system. [↑](#footnote-ref-1)
2. Ok, a better name maybe. [↑](#footnote-ref-2)