# Course Projects

In these projects you are supposed to practice some of the security concepts you learn in this course. The projects are done in groups of **two**.

The deadline of each phase and the delivery method will be posted on BeachBoard.

You get to work in a group to experience teamwork, however, your work is evaluated individually. At each phase, each individual contribution has to be clearly stated and separated from the other's. Always make clear who has done what part of the project. Otherwise each individual in the group will receive half of the total points. Do NOT forget to leave LOTS of COMMENTS in your code. Make sure that the relationship between various components, classes and entities are clearly described. If your project is hard to follow, you certainly will lose all the points. Remember, no grade if no proper documentation. That is, if anyone on this planet stumbles upon your repository, they should be able to compile your work and run all the modules without any difficulty.

All of your documentation materials HAVE TO be TYPED and submitted through your github repo. Documentation is required. You either can use girhub wiki (recommended) or Doxygen.

***Github:*** All of your work has to be backed up on a github repo. You need to share your repo with everyone at BeachBoard. Under Discussion, there is a forum called github links. Each group with create a new thread with their group name as the title. In the body, include all of the members' names (along with their github handler) as well as the link to your repo. Include each member's github handler along with their name. so we can recognize you on github. Make sure that you keep pushing any change you make to your repo. It has to be updated very frequently. Do not wait!

The assumption is that you begin working on your projects right after reading this document.

## End-to-end secure chat application

The projects are, in fact, multiple pieces of a bigger project, namely a secure end-to-end encrypted text messaging application (limited-size text messages only, one-to-one chat, group chat functionality and ephemeral key exchange is a **big** bonus but not required).

In this project, you are assigned a task of designing, implementing and testing a secure messaging app that provides end-to-end encryption for one-to-one messaging (at minimum). The idea is that you'll have a server that distributes the messages among client applications. You'll need to develop a secure, authenticated way of communicating between your client and server. Then you'll need to design and implement an end-to-end encrypted (that proves confidentiality and integrity) of message exchange between two clients thru the server.

To this end, you will develop a RESTful web server (https only) to receive a message from one client and pushes to the recipients when they become available (i.e., store messages in a database). You need to make sure that only the client applications developed by you can connect to the server (tip: Use an application authentication mechanism such as an app key). In addition, you'd need to authenticate the user (username/password or something similar. If your platform is mobile you can read the phone number w/out any user interaction and set it as username). Each message will contain info about the sender, the receiver, the encryption meta data, the encrypted message and anything else necessary. Apps periodically ask the server if there is a new message for them (e.g., you can use a GET request). The receiving end, will use the encryption meta data (if any exists) to decrypt the message. Then the message along with the identity of the sender is displayed. A very basic GUI is enough, there is absolutely no emphasis on how fancy your GUI looks.

Note: If you are after the bonus points of adding group chat (multiple people) functionality or ephemeral key exchange , make sure you run your ideas with the instructor.

If you are not familiar with web servers, API requests, JSON web tokens (JWT) then get started on reading up on those ASAP. There are some resources on BeachBoard (under Discussion-> Useful links). If you intend to use a framework, start reading documents as early as you can.

## ***TASKS***

**Phase I:** Setup your aws LAMP Ubuntu server

For this task, follow Setting up an "AWS EC2 Instance with LAMP and GIT" tutorial on BeachBoard. DigitalOcean is OK too but AWS is recommended. Make sure you can SSH to your LAMP server (MEAN stack at your own risk. You are free to use well-established frameworks such as Django or web2py for Python, Laravel for PHP, Spring for Java, Qt for C++,Node and ....). Make sure you can pull changes from your github repository on your server.

Get a free domain (register with namecheap using your .edu email). Point your domain to your aws instance. Make sure all is good and "It works".

Your server should have a simple database to keep record of messages (sender, receiver, encrypted text, security data, timestamp, …). Incorporate SQL injection defense mechanisms if applicable (more on that in class).

**Phase II**: A simple HTTPS Server

Use Let's Encrypt to setup a free certificate (all instructions on Let's Encrypt). Test your TLS configuration with SSL Labs.

You will be graded based on your SSL Labs performance and the robustness of your API routes along with your authentication methods.

**Phase III**: Encapsulation/Decapsulation

Use OpenSSL command line prompts to generate 2048-bit RSA public/private key pairs (ECC is bonus).

Develop an application with two modules:

*Encrypter*:
In this module, the input is a message string and an RSA public key file path (.pem file).
Initialize an RSA object (OAEP only). Your RSA object will load the public key. You then initialize an AES object (be careful of modes of encryption and padding) and generate a 256-bit AES key.
You encrypt the message with your AES (do not forget to prepend the IV). In addition, you'll generate an HMAC 256-bit key. Run HMAC (SHA 256 is good enough) on your ciphertext to compute the integrity tag. Finally concatenate the keys (AES and HMAC keys) and encrypt the keys with the RSA object. Output the RSA ciphertext, AES ciphertext and HMAC tag (JSON is a good choice to represent the final output. Pay attention to size of each key, value pair).

*Decrypter*:
In this module, the inputs are:
a JSON object with keys as: RSA Ciphertext, AES ciphertext, HMAC tag

A file path to an RSA private key

First, initialize an RSA object (OAEP only with 2048 bits key size). Load the private key into your RSA object. Decrypt the RSA Ciphertext (from JSON) and recover a pair of 256-bit keys (one is AES and the other is HMAC).
Run an HMAC (SHA 256 with the HMAC key recovered from above) to re-generate a tag. Compare this tag with the input tag (from JSON). If no match, return failure. Otherwise, continue by initializing an AES key (same as the Encrypter mdoule) and decrypt the AES Ciphertext (from JSON input). Pay attention to IV and padding. You return the recovered plaintext (or failure at any step).

Our approach here is very similar to the principles of PGP (Pretty Good Privacy).

For more info, Google OpenPGP implementation of PGP protocol. Also, you can check out the source of the Open Whisper Systems at github.

We will check your encryption/decryption method calls as well as your integrity check.

**Phase Design**: Project requirements and design documentation

In your repo as well as BeachBoard (under Dropbox, folder titled "Phase Design") you'll include all your design documentation of your solution. That is, you explain any additional security requirements your system will satisfy. You'll explain how you plan to achieve your goals. Your document should include a high-level overview of your system components (e.g., authentication methods, message integrity checks, etc). Your documentation needs to justify why your solution works (meeting the security requirements thru proper analysis). Be creative yet rigorous in your solutions.

You need to include system diagrams (depicting what your system components are and how they tie together). Also, include use case diagrams or tables. Most likely you will not cover all possibilities but try to be as comprehensive as possible. Include an attack tree in your design documentation.

Even though this phase includes no programming, this is the most important task since here you actually design your solution to your assignment. Make sure that you think of all possible venues of attack and how things could go wrong. For any attack surface that you can think of, incorporate a defense method in your solution. By the end of this task, you ought to have a very clear understanding of what you'll be doing and how you'll achieve the desired outcomes.

Ask your friends to go over your documents and criticize you before the instructor finds a mistake in your work. Remind your fellow students of: "I scratch your back your scratch mine".

**Phase IV:** JWT and RESTful

Work on GET, POST and other necessary API requests. You may want to focus on JSON or XML structures for your responses.

In class you'll learn about JWT. Implement your JWT (use the resources under Useful links).

**Phase Client**: Client side

Now you can put all the pieces together.

You can develop your application for any platform (desktop or mobile). Little attention is given to GUI. Your client has to connect to your sever over TLS protocol using your http API calls. The message delivery should work even when the receiver is offline at the time of sending (i.e., store them to the DB, clients have to do GET pooling to check for any new message periodically. The alternative method of Websockets is bonus but this is a risky area if you have not used Websockets before).

Implement your public key exchange method.

We will check whether you can post a message or get a message from the server at this stage.

----------------------------More info will be posted in future----------------------------

For those who want to do even more and are looking at other ideas, some simple examples of project could be:

-Security of Campus Payment System: Throughout the campus there are digital systems (like vending machines and such) that use your campus card to make payments on your behalf. These machines are connected to the campus network and they charge each student’s card. A project could be a complete security analysis of

such systems. It should include identifying security flaws in the system along with solutions to address such flaws. It does not have to include implementations though it’d be up for extra credit.
-Electronic cash (e.g., Bitcoin)
-Electronic voting systems (E-Voting)
-Secure card playing over the network
-Secure auction
-Return-to-lib attack: Consider new techniques of exploiting return-to-lib attacks and implement such an attack. Show clearly the level of success of your attack in your target system. If your project includes attacks then there should be an implementation of such an attack. There are numerous other attacks (look at chapters 10-13).

## Cryptographic libraries

You need to find a library that is most suitable for you. Below is a very incomplete list of cryptographic libraries should you need to use them:

The GNU Multiple Precision (GMP) library; http://gmplib.org.

OpenSSL crypto library; there are also the SSL library and the openssl command line tool. Recently there have been new forks of openssl project. http://www.openssl.org/.

Crypto++ library, available from http://www.cryptopp.com/.

Open Whisper repos at: https://github.com/whispersystems/

Libgcrypt, available from http://www.gnu.org/software/libgcrypt/.

MIRACL library, available from http://www.certivox.com/miracl/. Good for efficient elliptic curve functions.