Secure Payment via Mobile Phone

Adarsh Kirnelli Rangaiah (520), Shamya Karumbaiah (620)

April 29, 2016

Abstract

With the boom in the online marketplace, a lot of consumers are choosing shopping at the convenience of their personal devices. Mobile payment method is both timesaving and effortless. With this convenience comes the greater risk of compromising the safety of the customers' confidential information. This project proposes a framework for secure mobile payment. The goal is to address security risks at various stages of the customer activity and provides a prototype implementation for any vendor who would like a secure mobile payment system in his mobile application. Towards this, different algorithms are studied and compared for their effectiveness and efficiency in the system.

1 Introduction

Mobile payment can be formally defined as the payment for products or services between two parties for which a mobile device, such as a mobile phone, plays a key role in the realization of the payment. A lot of issues persist around insecure mobile payment. Customers are at the risk of identity theft with their personal information at disclosure for these malicious attacks. In addition to this, are the sensitive information like credit card or online banking details. Attackers try novel techniques to hack weak systems using attacks like the rainbow table attack or dictionary attack to recover plaintext from hashed passwords. A successful attack could lead to greater loss like illegal transfer of funds. These attacks may also try to interfere with the normal functioning of the online system by denial of service. A physical theft of the mobile device would also contribute to these directly.

Currently, there are many mechanisms to overcome the above mentioned security risks. Unfortunately, there are still some attacks that succeed hacking these. Consider the example of Amazon[1]. All the transactions are secured using Secure Sockets Layer (SSL). To protect the clients' critical financial information while it is transmitted over the Internet, all critical information such as credit card information, password, and personal data are encrypted using 128-bit SSL. For further protection, sensitive information remains encrypted while stored on our servers.

There exists proved techniques to hack these protocols[12]. SSL and TLS are both secure protocols for Internet communication and work by encrypting traffic between two computers. Most TLS clients will downgrade the protocol they use to SSL 3.0 if they have to work with legacy servers. The vulnerability lies in the fact that an attacker can potentially interfere with the handshake process which verifies which protocol the server can use and force it to use SSL 3.0 even if a newer protocol is supported. The vulnerability was disclosed by Google[3], which said that a successful exploit could allow an attacker to carry out a man-in-the-middle (MITM) attack to decrypt secure HTTP cookies, which could let them steal information or

take control of the victim's online accounts. The attack can be executed both on the server side and client side. In this context, our project's aim is to provide an end to end solution for any mobile payment requirement giving utmost priority to security.

2 Algorithms Research

Below are the summary of different mechanisms chosen after researching on the attacks and the locations in an application we need to tighten the security.

1. Double hashing password using SHA256 to prevent identity theft - Identity theft could occur when an attacker gets hold of the confidential information of a user. To login to our system, the user needs to authenticate with his username and password. An attack might get hold of this information and make purchases unknown to the user. SHA stands for Secure Hash Algorithm. Cryptographic hash functions are mathematical operations run on digital data; by comparing the computed 'hash' (the output from execution of the algorithm) to a known and expected hash value, a person can determine the data's integrity.[5]

2. Salted password to prevent dictionary attack - In addition to the hashing in the last point, a salt is random data that is used as an additional input to a one-way function that "hashes" a password or passphrase.[9]

3. Encrypt card Information using AES to prevent information disclosure - Advanced Encryption Standard(AES) is a symmetric encryption algorithm.[13] We used it in our system to encrypt the card details entered by the user. AES is based on a design principle known as a substitution-permutation network and is fast in both software and hardware. [14]

4. Initialization vector (IV) for a stronger encryption - IV is analogous to salt in case of hashing.

5. SSL to prevent transaction repudiation - SSL (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral.[12]

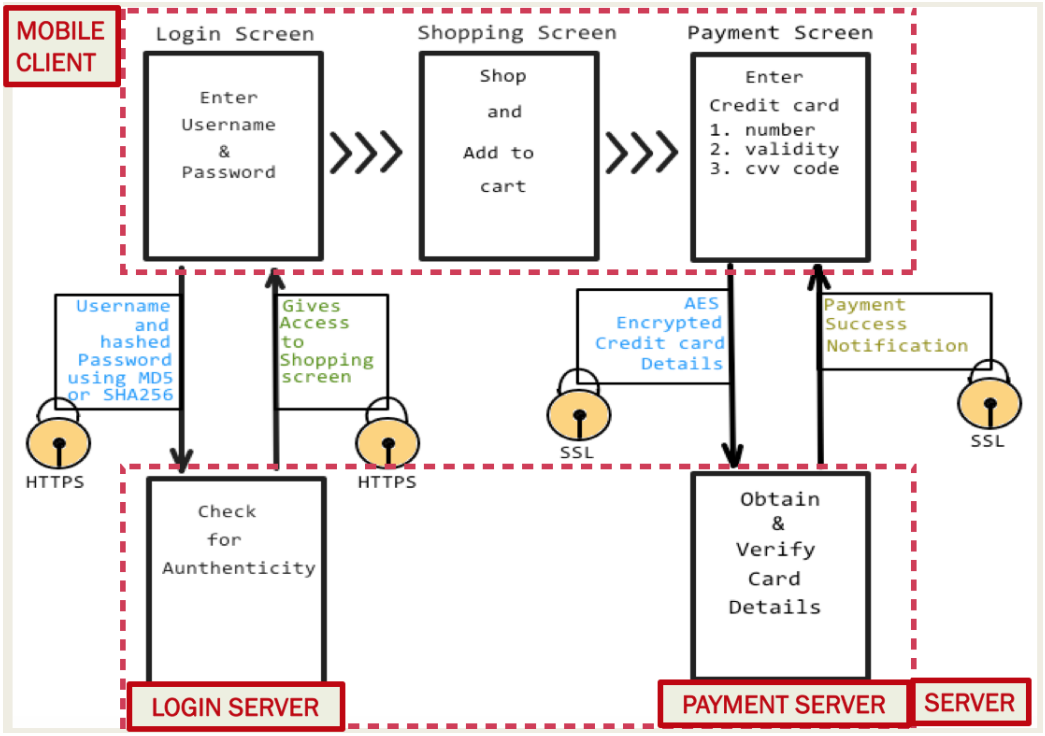
6. Login screen to secure the app during physical theft

7. Using time stamp to overcome replay attack - A replay attack (also known as playback attack) is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed.[10] Using time stamping, synchronization should be achieved with a secure protocol. The receiver only accepts messages for which the timestamp is within a reasonable tolerance.

3 System Architecture

Our goal in this project, is to address each security risk one by one and provide a complete solution for any vendor who would like a secure mobile payment system in his mobile application. Below is the system architecture diagram. We follow a client server model. On the server end, we have 2 servers - a payment server that verifies card details and another login server that authenticates the user at login. The client is a

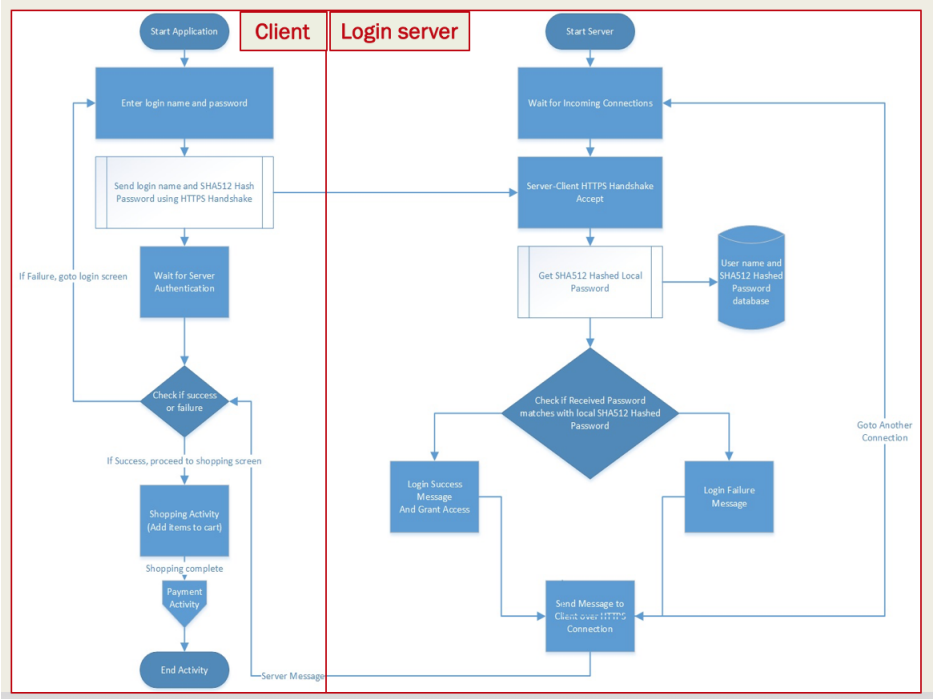
mobile client. It could be any mobile application.



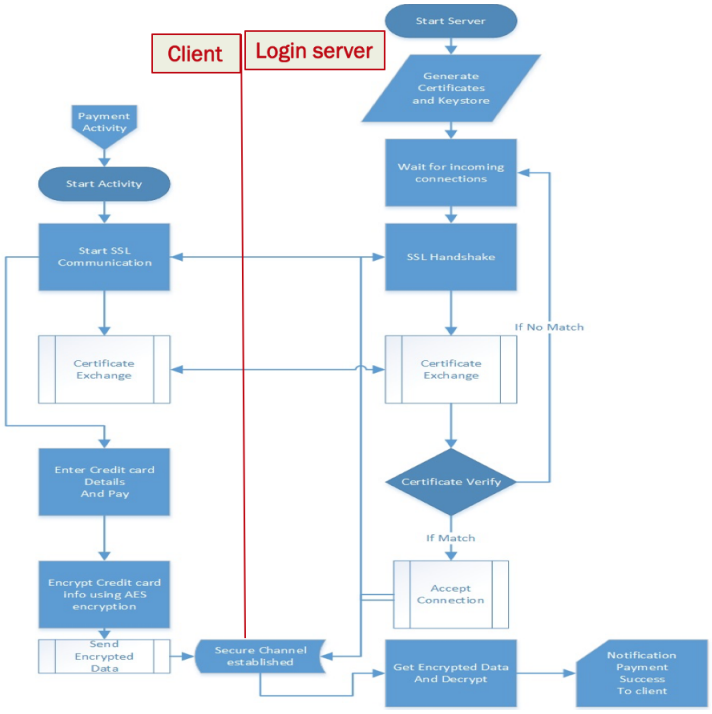
4 High Level Design

This section gives the visualization to understand the high level design if the system. To get a glimpse of the system, it is convenient to follow these application flow diagrams for the two major activities that happen in the system. A more detailed design is represented in the next section with a UML diagram.

The diagram below gives the details of the **login activity** -



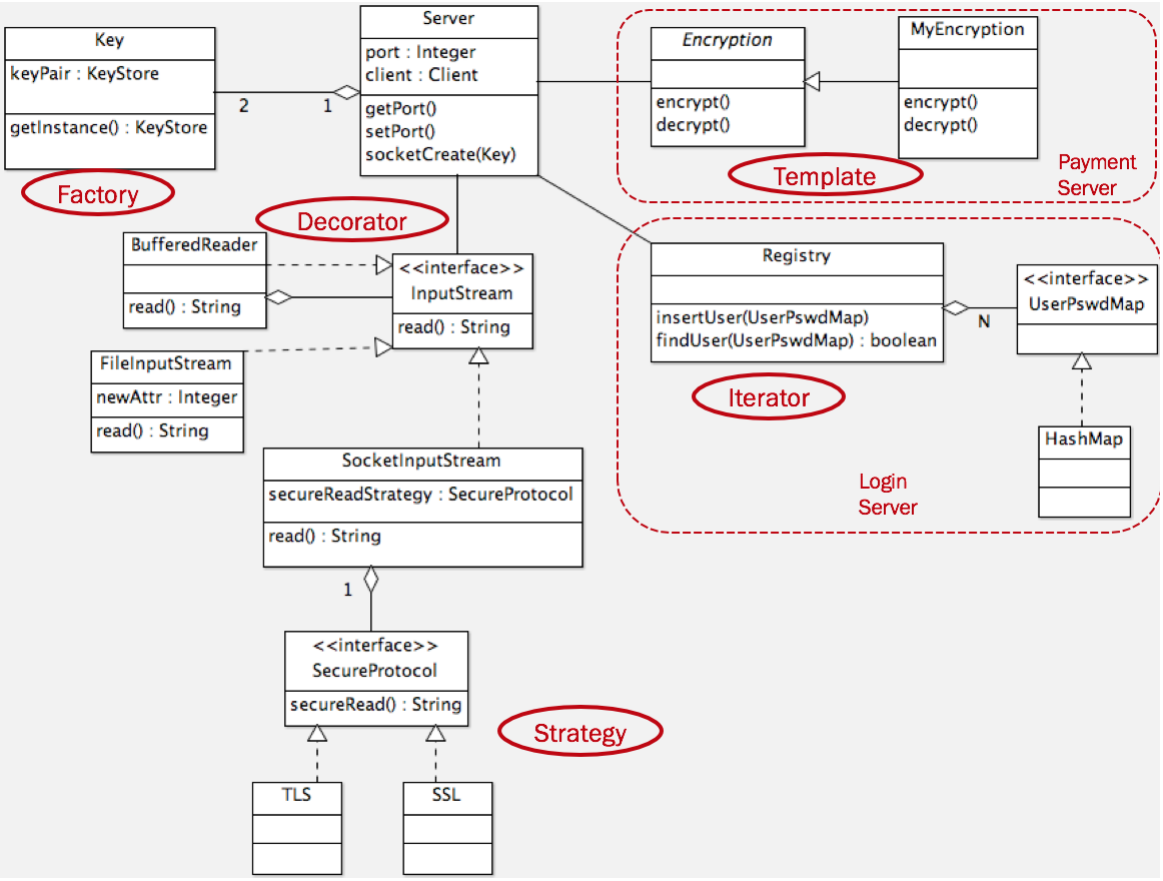
The diagram below gives the details of the **payment activity** -



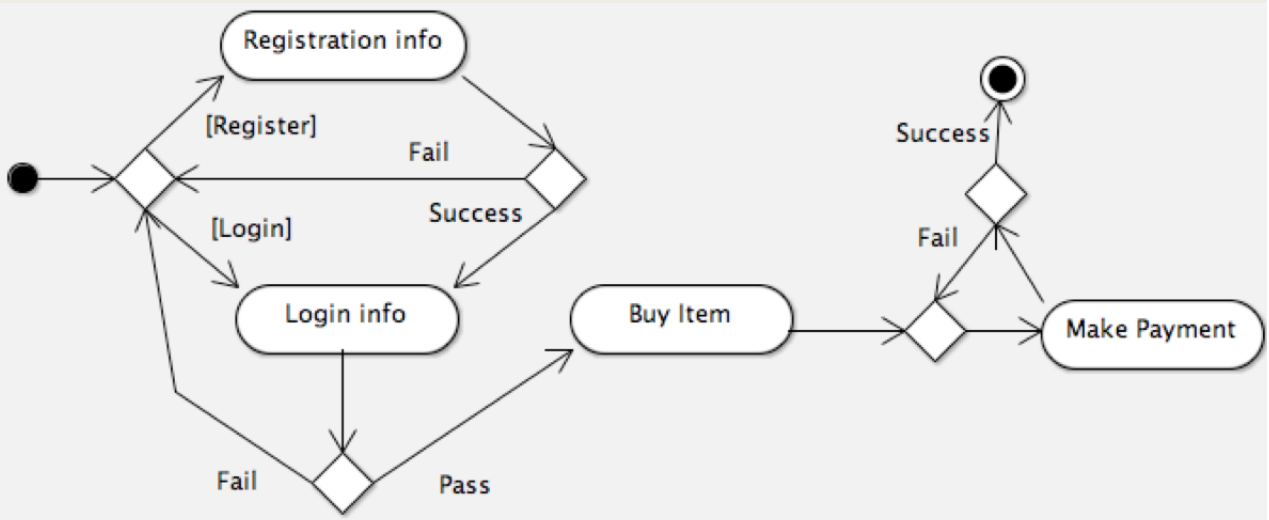
5 Implementation

This section explains the system implementation and revisits the design to make concrete comments on the implementation specific details. The easy and dirty way to implement this system would be with three files - two for the servers and one for the client. But such a monolithic implementation would have many shortcomings. A clear separation of concerns would help. Firstly, the server side code has common logic for both payment and login servers. Hence, a clear separation of concern aids in code reuse. This is depicted in the UML diagram below. Secondly, this helps in testing. The different pieces of the system could be tested extensively.

While designing the system, there were some design patterns that naturally fit into this setting. For example, using a strategy pattern for secure protocol selection. We tried Transport Layer Security (TLS) and Secure Sockets Layer (SSL) and changing these were easy with a strategy pattern. Likewise, a factory pattern for KeyStore feels necessary. The client side uses Bouncy Castle (BKS) format and the server side uses Java Keystore (JKS). An iterator is an obvious choice too. A template to choose the encryption method helped. An AES encryption is used as yet. There is potential for other algorithms in future. But there were other design pattern decorator which wasn't quite intuitive but proved useful like decorator. On the whole, design patterns turned out to be very helpful in designing the system better. We had concerns about the incurred complexity by these patterns. This trade off and our strategy towards it is explained in the latter section on lessons learnt.



For the client side implementation, android studio was used to develop an android application. There were standard activities that we leveraged. There are 5 activities at the client. This is better depicted with an activity diagram below. Since, our focus is mainly on security, we haven't paid close attention to the design of the user interface. It is a simple marketplace with one product to buy and has 3 major screens - login screen, shopping screen and the payment screen. Also, to allow better testing, the restrictions on the username and password was not imposed. This helps us do the dictionary attack testing (explained in testing section).



6 Testing Research

Testing is the most crucial element of this system. We claim to provide a more secure platform and this implies that we must test the solution extensively. It is hard to prove in theory that it is unbreakable. Hence, we take an approach of extensive testing using multiple platforms and concepts to check the effectiveness of our methods. Though we have tried to cover the well known attacks and tools, we cannot call it complete.

6.1 Ettercap and Man-in-the-middle Attack

Man-in-the-middle attack is a type of attack where the attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other.[6]

To simulate this, we have used Ettercap [4] which is an open source tool. To simplify, we have tested it in our LAN. It works by putting the network interface into promiscuous mode and by ARP poisoning the target machines - thereby acting as a 'man in the middle' and unleashing various attacks on the victims. We have used a ARP-based mode which uses ARP poisoning to sniff on a switched LAN between two hosts. ARP poisoning is a technique by which an attacker sends Address Resolution Protocol (ARP) messages onto a local area network.

In our testing scenario, we used Ettercap to intercept the data sent from the mobile client to the login and payment server. During the login activity, data obtained like this will be a hashed password. On this, we induce a dictionary attack explained in the next subsection. During the payment activity, data obtained like this will be the encrypted card details. On this, we induce a brute force attack explained in the next subsection as well.

6.2 Dictionary Attack

A dictionary attack is based on trying all the strings in a pre-arranged listing, typically derived from a list of words such as in a dictionary[8]. In contrast to a brute force attack (next subsection), where a large proportion of the key space is searched systematically, a dictionary attack tries only those possibilities which are deemed most likely to succeed. Dictionary attacks often succeed because many people have a tendency to choose short passwords that are ordinary words or common passwords, or simple variants obtained, for example, by appending a digit or punctuation character.

Assuming the attacker figured out the length of the password, he could now use this information to form a dictionary of passwords. To simulate, weak passwords were entered with short length, least variation in the string, common names. Our security mechanism of salting the password and double hashing it, makes it very hard for the attacker to hack the password even when the users have chosen weak passwords.

6.3 Brute Force Attack

Brute force attack is the most naive form of attack and is usually done when all other attacks fail. It is done by systematically checking all possible keys until the correct one is found. In the worst case, this would involve traversing the entire search space.[7]

To simulate this, we have the encrypted data obtained by the man-in-the-middle attack in the previous subsection. We assume that the attacker knows that AES encryption is used - this is not an unreasonable assumption provided the widespread use of this algorithm. There are many possible bit sizes for the key. For simplicity, we assume that the attacker is aware of the key size as 256 bit. Now, there are 2^{256} possible values of this key. The goal is to find the key value used for encryption, so the same could be used to decrypt. It is impractical to test every possible combination. Hence, we wrote a simulation with two approaches. One that uses combinations of smaller string sequence. And the other that randomly picks one of these keys in each of the 10,000 iterations.

Though one can say that this is driven by the randomness, the same is applicable in the real world. There is a clear trade off between the computing power one would like to invest on such attacks. So, we believe this simulation captures such a behavior. With a key size of 256, we are providing a higher security. But we must note that with the increase in the key size, we are also increasing the decryption time in the application. Hence, we have found 256 bit key size as widely popular with 512 being close to it.

7 Lessons Learnt

This project was a good experience to get a feel of a systematic software development process. The following are some lessons learnt -

- Appropriate effort estimation for black box testing based on the complexity of the project - This piece was overly underestimated by us at the first stage of the effort estimation. We had given an equal amount share of efforts for development and testing. But as the project evolved we realized the importance of testing in the nature of the project it was. It was very time consuming to find the right tools and methodology to conduct this test.

- Striking the right balance between using design patterns and code complexity. As mentioned in the earlier section, other than the most obvious design patterns which fit in perfectly, there was always a tradeoff between the convenience a design pattern could offer and the induced code complexity. In some cases, we decided to go against a prescribed design pattern for that scenario as it was unnecessarily complicating the program. It was also yet times difficult to debug without referring the UML diagram. Thus, the rule of thumb we followed is to go with a design pattern only for those parts of the code which had a higher priority than others in the big picture. These were mostly the ones that directly interacted with the choice of algorithms.

- Potential of activity diagram for better communication in UI intense projects - The notations are intuitive and easy to represent the overall UI design.

- Time complexity suffered while trying to increase security (increased by 47% as measured by execution times) - In our earlier proposal, we didn't have computational complexity as one of the metrics. The only goal was to find the most robust system in terms of security concerns. As we added multiple layers of protection and used duplication like double hashing, it was clear that the time complexity suffered for the overall system. This is a big concern when we look at the practical application of the system. Considering a vendor uses this for his login and payment activities, it will not be user friendly to keep the user waiting for so long even though it is for his own security purposes.
  To illustrate, we captured the time taken to decrypt with several runs of the application as this was the most time consuming action in the application. On average, it was approximately 0.208 s for the 256 bit encryption and stood out in the tradeoff between time complexity and security offered among the different key sizes available. Time to encrypt will be lesser than the decryption time. This overhead is usually lower due to the high computing ability of the devices that host these server or app.

- Separation of concerns at the early stage of development towards a better code reusability - Monolithic code is easy and dirty. For a project of this scale, it is necessary to separate the concerns early on easy development, design and testing. This also helps in better communication in the team and division of labor.

8 Demo

Kindly follow the instructions in the README file. You could also watch a demo here - https://drive.google.com/open?id=0B3SpqX1jb1RhWkxKbUF5ekNrakU

9 Conclusions

The proliferation of mobile devices such as smartphones and tablets has given consumers a varied choice of platforms to make payment for their online purchases. The methods being experimented in this project would be a good value add to any current mobile payment system. A comparative study of the different security mechanisms would help in the selection of the appropriate algorithm as per their need. There could be a possible performance impact due to double encryption methods used. Future work towards this should look at improving the performance. There are many enhancements possible to make this framework user friendly and secure. A verification process while registration either with email or mobile could be used. Camera scan of the credit card could be used to extract and store the details for payment after encryption. The user's account could also have a provision to save the credit card details.

References

[1] *https://payments.amazon.com/help/5969*. Amazon.

[2] Y. Sheffer et all. *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*. Internet Engineering Task Force (IETF), 2070-1721.

[3] Bodo Moller, Thai Duong, Krzysztof Kotowicz. *This POODLE Bites: Exploiting The SSL 3.0 Fallback*. Google.

[4] *Ettercap*. http://ettercap.github.io/ettercap.

[5] *SHA256*. Wikipedia

[6] *Man in the middle attack*. Wikipedia

[7] *Brute force attack*. Wikipedia.

[8] *Dictionary attack*. Wikipedia.

[9] *Salt*. Wikipedia.

[10] *Replay attack*. Wikipedia.

[11] *Ettercap*. http://www.larc.usp.br/ pbarreto/hflounge.html .

[12] T. Dierks, E. Rescorla. *The Transport Layer Security (TLS) Protocol*. Version 1.2.

[13] Joan Daemen, Vincent Rijmen *AES*.

[14] Bruce Schneier et all *The Twofish Team's Final Comments on AES Selection*

A Project tasks and timeline

- **Task1** : Develop an android application and establish socket communication with the server hosted on a laptop. (Owner - Adarsh) 03/15

- **Task2** : Identify the library functions for SHA512 hashing, AES Encryption and SSL socket. Understand the method of obtaining Certificates from Certificate Authority. (Owner - Shamya) 03/25
- **Task3** : Integrate the security mechanisms leant in Task2 with the android application developed in Task1. (Owner - Adarsh) 04/01
- **Task4** : Research and write a detailed report about the security mechanisms used in existing mobile payment apps in comparison to our approach. (Owner - Shamya) 04/15
- **Task5** : Test and evaluate the system on the evaluation metrics mentioned in the previous section (Owner - Adarsh/Shamya) 04/15