Securing Ford Mobility System - GoodTurn

Katherine Snyder and Kevin Daimi
Computer Science and Software Engineering
University of Detroit Mercy
Detroit, USA
email: {snyderke, daimikj}@udmercy.edu

***Abstract*—Ford Mobility System, *GoodTurn*, is an application developed by the University of Detroit Mercy through a Ford Motor Company grant. In a manner similar to Uber, the application connects Ford employees interested in volunteering their time and vehicles with nonprofit organizations needing to transport goods and resources. Both drivers and requesters will use their iPhones to connect to the application and each other. The privacy of the data collected from drivers, requesters, and the nonprofit organizations is critical. The goal of this paper is to introduce the needed security protocols to protect the GoodTurn application. The proposed security protocols will rely on the advocated GoodTurn security architecture.**

Keywords—Security Architecture; Security Protocol, Symmetric Cryptography; Public Key Cryptography; Ford Mobility System(GoodTurn)

I. INTRODUCTION

Ford Motor Company offered a program to solicit ideas from their employees regarding the best ways to serve the society at large and chose three of these projects to fund. The program was carried out in both USA and UK. One of the ideas presented was to have Ford employees donate their spare time and vehicles to help nonprofit organizations in moving their goods and resources. Given the existing support for iPhones by Ford for its employees, the initial release of the application was specified to use iOS, with the intention to expand to other devices and systems in later versions. The idea for the application was reminiscent of the way the Uber application connects drivers with riders, but with no money exchanged. Ford Motor Company provides a grant to develop this system and the University of Detroit Mercy was selected to develop and implement the application, currently referred to as the Ford Mobility System, *GoodTurn*. Xcode [1] was used to develop the GoodTurn application, based on the Swift language [2]. Furthermore, Firebase 3.0 was employed for several components of the application [3].

As stated above, the idea for this application was modelled after Uber. However, the security approach followed by this paper has nothing to do with Uber. The security of Uber has not been made public to allow others to compare their own security approaches to Uber. There has been some controversy about the operation and use of Uber. In 2015, McCallion [4] stressed that Uber has accidentally leaked the private information of many of its drivers when the app was newly launched. This initial release of the Uber app apparently had a design defect that allowed drivers to access various sensitive scanned documents containing details such as, social security numbers, tax forms, insurance documents, and drivers' licenses. The bug emerged when an Uber driver tried to upload or edit such documents. The driver was directed to a screen containing details of Uber drivers within the United States.

Bernstein [5] and Kovacs [6] indicated that a Portuguese team has recently found 14 flaws in Uber apps which have enabled the team to obtain free rides and access details of passengers and drivers. Another flaw detected by the team was linked to Uber's promotion codes. The riders.uber.com website did not involve any countermeasures against brute-force attacks. This flaw enabled attackers to continue to create promo codes until valid codes were obtained. With the emergent attractiveness of developing Uber-style applications, the attempts to use Uber as a development platform for various applications accessible via the cloud, requires more vigilant attention to security issues.

Armerding [7] emphasized that scammers attacking Uber can get a free ride, while victims pay the the bill. This occurred when cyber attackers manage to obtain the login credentials of legitimate users and sell them to fraudsters. Popular apps like Uber are targets for online scammers and cybercriminals; therefore, these apps must employ rigorous security and privacy measures to deter and prevent these malicious activities. Taking into consideration the above-mentioned incidents regarding Uber security, Uber continues to introduce app improvements with the aim of further securing the application and safeguarding privacy of drivers and passengers. Cava [8] stated that Uber added a new feature requiring drivers to authenticate their identities via a selfie photo prior to each shift. The goal of such real-time ID proof is to thwart fraudulent utilization of a driver's account and provide passengers with a higher degree of confidence in using Uber vehicles.

With the constantly increasing sophistication of security threats and attacks on software applications, advances in security countermeasures should at least parallel this sophistication. Dong, Peng, and Zhao [9] suggested using security patterns to avoid security problems. They believed that security patterns provide professional solutions to common security problems and capture best practices on secure software design and development. Security risk analysis is definitely the first step to design a secure system.

Baca and Petersen [10] introduced the notion of countermeasure graphs — a risk analysis approach for software security. They added that countermeasure graphs grant decision support for prioritizing countermeasures, and support software developers in determining critical threats and implementing optimal solutions. A Case-Based Management System (CBMS) comprised of an artifact management system and a knowledge-based management system (KBMS) to handle cases for secure software development was introduced by Saito, et al. [11]. The goal was to manage the software artifacts created in the secure software life cycle, in addition to the software security knowledge using the two components of CBMS. Although useful in secure software development, nevertheless, none of these approaches addressed secure communication between the software itself and its external interface.

Software security vulnerabilities give rise to many security breaches and attacks. New security vulnerabilities are discovered daily. Vulnerabilities are behind many software failures. In any software development, coding is the critical issue because many security deficiencies are developed during the coding phase. Okun, Guthrie, Gaucher, and Black [12] investigate the use of static analyzers to identify defects in source code that could result in security breaches. Jain and Ingle [13] argued that to have secure software, a software security requirements process is essential. They designated a Software Security Requirements Gathering Instrument (SSRGI) and claimed it can help developers extract security requirements from various stakeholders, and indicated SSRGI can strengthen security during the consequent phases of software development. Software security testing plays an important role in detecting security flaws. According to Tian-Yang, Yin-Sheng, and You-Yuan [14], Software security testing is the process of identifying whether the security attributes of software implementation are consistent with the design. They stipulated that software security testing involves security functional testing and security vulnerability testing. Security functional testing analyzes whether the software security attributes are implemented appropriately and consistently with security requirements. While testing for vulnerabilities and security flaws are essential for secure development, they do not necessarily prevent security attacks where software applications are accessed via the internet.

This paper presents a security architecture for the Ford Mobility System, *GoodTurn*. A cryptographic protocol is used to implement the security architecture. A protocol is a multi-party technique represented as a sequence of steps that exactly identifies the actions required of two or more parties in order to accomplish a specified goal. Mainly, the goal is to secure the exchange of messages between the parties. If cryptography is used to secure messages, a cryptographic protocol will be involved. Protocols are probably the most difficult part of cryptography because neither the designer nor the implementer of the protocol has any control over other parties' behavior. Normally, it is very challenging to isolate the vulnerabilities of cryptographic protocols as they can be the outcome of subtle design flaws [15]-[17]. The remainder of the paper is organized as follows: Section II provides the FMS operation overview. Section III elaborates on the FMS security architecture. Section IV depicts the cryptographic protocols needed to secure the FMS. Section V concludes the paper.

II. FMS OPERATION OVERVIEW

The following use case scenario briefly illustrates the operation of the Ford Mobility System, *GoodTurn*. This is needed to understand the security architecture of GoodTurn and the associated cryptographic protocol. Volunteer drivers will be referred to as "driver". A representative of the nonprofit requesting a driver to move goods will be referred to as "requester".

1. The system starts with a splash screen to indicate the application is being launched.
2. New drivers/ requesters register with the system first.
3. The application requests the user name and the password of the user (driver/requester). Subsequent use is authenticated against this information.
4. Driver/requester can modify their information/profile.
5. If needed, the system can recover password, deactivate or reactivate user account.
6. Non-profit organization/Non-government organization (NPO/NGO) adds and removes requester users, and provides them with administrative rights
7. Drivers and requesters sign off on a privacy policy.
8. The system provides a list of current jobs to drivers provided by requesters to move goods.
9. The system calculates the estimated time needed to complete a job by a driver.
10. Requesters enter new jobs, include their organization information, add a job to job queue, modify a job request, or cancel a job.
11. If a requester/driver does not want to deal with a specific driver/requester, the driver/requester is added to that driver/requester's blacklist. At any time, driver/requester can be removed from a black list.
12. Driver/requester view the job history.
13. Drivers filter jobs, sort them in any way they prefer, accept jobs, reject jobs or cancel accepted jobs. As a result, the job list is updated.
14. The system notifies the requesters/drivers regarding any action listed in step #13.
15. If a requester's job reaches its pick-up time without being accepted, the system will allow the requester to reschedule it.
16. The driver/requester indicates that a job is completed.
17. FMS allows communication between requester and driver.

18. Both drivers and requesters provide feedback, submit problems if any, and ask for help.
19. Drivers and requesters rate each other.

III. GoodTurn security architecture

The Ford Mobility System (GoodTurn) security architecture introduced in Figure 1 illustrates all the components used. The participating parties are shown in Table 1 below. Furthermore, Table 2 provides a clarification of the symbols used.

A. Key Distribution Center

The Key Distribution Center (KDC) is the heart of the security architecture. It manages the symmetric keys distribution for each pair of the communicating parties, and providing the needed public keys for communicating parties. It further provides the keys needed for Message Authentication Code (MAC), which will be used for ensuring the integrity of various exchanged messages. The designated Security Service Agent (SSA) will act on behalf of host and servers it represents. Any of the servers or hosts of Fig. 1 can request communication with the components they are allowed to communicate with. Because some of the messages are relatively large and others are small, symmetric and public key cryptography will be used respectively. The request for keys should include the ID of the party to communicate with and the type of key. There are two types of keys, session key and MAC key. The MAC key will be used for message authentication. The SSA of the requesting party asks the Key Distribution Center for a session key, K_{XY}, to be shared between components X and Y to be sent to the component requesting it. Here X is the requesting component and Y is the component that X needs to communicate with. The KDC send the session key to party X together with the ID of the other party and type of key so that each party knows whom it will be communicating with and what will the key be used for. In what follows, ID_X and ID_Y are the IDs of component X and Y respectively, Key Type is 1 for session key and 2 for MAC key, and SSA_X is the SSA for component X. Note X and Y stand for Application server, Database Server, NPO/NGO, Driver or Requester. K_S is the symmetric key shared by KDC and SSA_X. Note that → indicates sending, and || stands for concatenate.

$$SSA_X \rightarrow KDC\text{: } E\ [K_S, \text{Request for Key} \ || \ ID_X \ || \ ID_Y \ || \ \text{Key Type}]$$

$$KDC \rightarrow X\text{: } E\ [K_S, K_{XY} \ || \ ID_Y \ || \ \text{Key Type}]$$

It is assumed that KDC and SSA shared public keys. Upon successful login of a component, the component receives the public key of KDC, PU_{KDC} via its SSA. This is needed to contact the KDC when requesting various keys. The component X who has received the session key and MAC key will then request the public key, PU_Y, of the component Y it wishes to communicate with and waits for Y to confirm the connection. The public key of Y is needed by X to share the session key and MAC key with Y. The protocol to achieve that is as follows:

1. X sends its ID and the ID of Y encrypted with the public key of KDC. A nonce, N_X is needed for assurance. A nonce is used by the sender to assure the receiver (party following →) the message is from sender.

$$X \rightarrow KDC\text{: } E\ [PU_{KDC}, ID_X \ || \ ID_Y \ || \ N_X]$$

2. KDC sends X the public key of Y together with ID of Y and its nonce, N_{KDC}, all encrypted with KDC's private key (signed) and then with the public key of X.

$$KDC \rightarrow X\text{: } E\ [PU_X, E\ (PR_{KDC}, ID_Y \ || \ PU_Y \ || \ N_X \ || \ N_{KDC})]$$

3. X contacts Y providing its ID, Y's ID, and a nonce N_X to show that the message is current. All these are encrypted with the public key of Y

$$X \rightarrow Y\text{: } E\ [PU_Y, ID_X \ || \ ID_Y \ || \ N_X]$$

4. Y verifies with KDC to see if it can communicate with X.

$$Y \rightarrow KDC\text{: } E\ [PU_{KDC}, ID_X \ || \ ID_Y \ || \ N_X]$$

5. If KDC confirms the message, it encrypts the public key of X, ID of X, and a time stamp T_{KDC}. Note that the message is first signed with PR_{KDC}, and then made confidential with PU_Y.

$$KDC \rightarrow Y\text{: } E\ [PU_Y, E\ (PR_{KDC}, ID_X \ || \ ID_Y \ || \ PU_X \ || \ T_{KDC})]$$

6. Y carries out the required decryptions and obtains PU_X. It informs X it is ready to communicate by encrypting a message containing the ID of X, ID of Y, and a nonce, N_Y encrypted with the public key of X.

$$Y \rightarrow X\text{: } E\ [PU_X, ID_X \ || \ ID_Y \ || \ N_Y]$$

7. At this point X shares the session key and the MAC key, KM_{XY}, with Y. Here the message is also signed by PR_X first and then confidentiality is enforced through encryption by PU_Y.

$$X \rightarrow Y\text{: } E\ [PU_Y, E\ (PR_X, ID_X \ || \ ID_Y \ || \ N_X \ || \ K_{XY} \ || \ KM_{XY})]$$

The session and MAC keys are valid for a single communication only. Fresh session and MAC keys are requested for subsequent communications. Note that in what follows, the communication with the Key Distribution Center will not be mentioned because it has already been taken care of in this section. For example, the Application Server communicates with five components including the KDC. The link with the KDC will be subtracted from the total number of links resulting in four links only.

B. Application Server

The Application Server (AS) runs the FMS, and therefore, controls all the functions of the system. It communicates with Database Server, NPO/NGO, Driver, and Requester components. To achieve all these communications securely, four session keys and four MAC keys are needed. Certainly, the Application Server could have also played the role of KDC in addition to its original role. However, it is safer to have independent server taking care of key distribution.

C. Database Server

The Database Server (DS) stores information about drivers, requesters, and NPO/NGO. In addition to profiles of the requesters and drivers, it keeps the blacklist of drivers and requesters, job history, active jobs, deactivated/reactivated user accounts, rejected jobs, and accepted jobs. The DS aids the Application Server in carrying out its job. From Fig. 1, it is clear that DS exchanges messages with the Application Server only. No other component is allowed to access the Database Server. Hence, one session key and one MAC key are needed.

D. Non-Profit/Non-Government Organization

Non-Profit Organization (NPO) / Non-Government Organization (NGO) component communicates with the Requester and with the Application Server. It adds and removes users and requests some reports and displays from the Application Server (AS). Two session keys and two MAC keys are needed for such interaction.

E. Requester

The Requester (R) should be associated with an NPO/NGO. It interacts with both the Driver and the Application server. The requester exchanges a number of messages with the Driver and Application Server. Some of these messages include registration, profile change, list of current job, privacy policy, job addition, job cancelling, feedback, blacklist addition, driver rating, and completed jobs. Because there are two connections, two session keys and MAC keys are needed.

F. Driver

The Driver (D) communicates with both the Application Server and the Requester to exchange various messages, such as registration, profile change, list of current jobs, privacy policy, accepted jobs, blacklist insertion, requester rating, and completed jobs. The driver needs two sessions keys and two MAC keys.

TABLE I. PARTICIPATING PARTIES

Symbol	Meaning
KDC	Key Distribution Center
SSA	Security Service Agent
NPO	Non-Profit Organization
NGO	Non-Government Organization
AS	Application server
DS	Database Server
R	Requester
D	Driver

IV. SECURING THE SYSTEM

The security of the FMS system relies on both symmetric and asymmetric cryptography. In addition, MAC keys are shared between the communicating parties. As noted above, the KDC will provide symmetric and MAC keys to the party initialing the communication. Then that party will request the public key of the receiver to forward the session and MAC keys. Several messages shared by the Driver and Requester components with the Application Server are similar. Those messages will not be repeated.

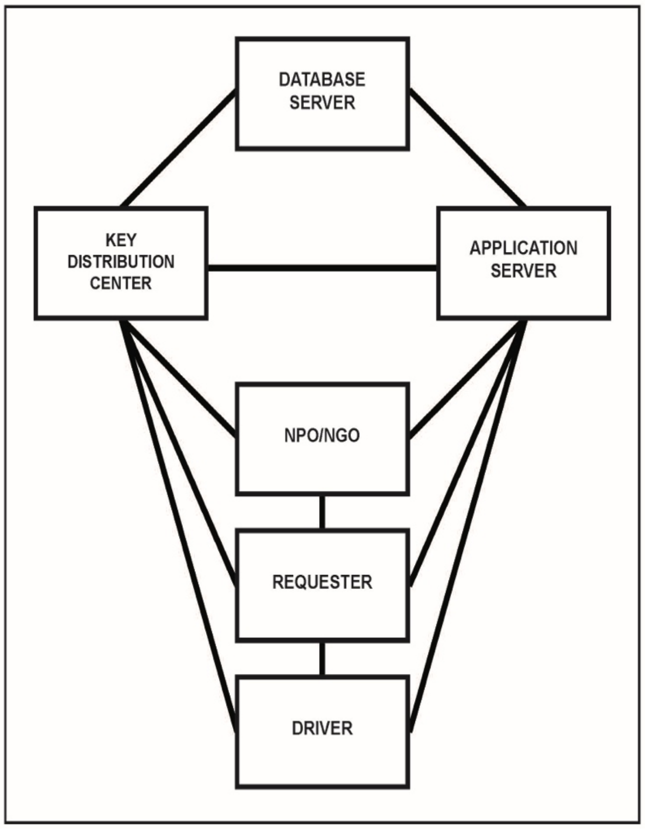

Figure 1. GoodTurn Security Architecture

A. *Driver-Server Communication*

The Driver component needs to send the following short messages to the Application Server (AS): email, password, security question/answer, request to reactivate account, request to register, registration information (name, email, driver/requester, address, company name, phone #, organization code, accepting/rejecting privacy policy, request to rate, rating, and job completed. All these messages are the same for the Requester (R). The symbol M will refer to any of these messages because the security procedure handling them is the same. D signs a message including its ID, ID of AS, M, the MAC of M (E ($KM_{D\text{-}AS}$, M)), and time stamp T_D, and then encrypt them all with the public key of AS, PU_{AS}

$$D \rightarrow AS\text{: } E\ [PU_{AS}, E\ (PR_D, ID_D \,\|\, ID_{AS} \,\|\, E\ (KM_{D\text{-}AS}, M) \,\|\, M \,\|\, T_D)]$$

On receiving such messages, AS performs the needed decryptions to obtain M. It then calculates the MAC of M and compares with the received MAC, and checks the currency of the message using T_D. Once they are equal, it accepts these messages and informs the Database Server (DS) to store the information or acts on them.

The Driver component also sends other messages that are specific to the Driver. They include vehicle make, model, year, color, license plate, type, maximum mileage, and messages to indicate job is accepted, job is rejected, and job is cancelled. These messages are treated as above.

The SA sends D messages that are somehow long, such as privacy policy, available jobs, accepted jobs list, cancelled jobs list, and completed jobs list. For this purpose, symmetric key will be adopted because public key tends to be slow with long messages. To this end, AS encrypts the ID of D, its ID, the MAC of message M, message M, and the time stamp T_{AS} with the symmetric key, $K_{D\text{-}AS}$, shared with D. T_{AS} is inserted to assure D the message is current

$$AS \rightarrow D\text{: } E\ [K_{D\text{-}AS}, ID_D \,\|\, ID_{AS} \,\|\, E\ (KM_{D\text{-}AS}, M) \,\|\, M \,\|\, T_{AS}]$$

D will decrypt this message using the key, $K_{D\text{-}AS}$, and verify the MAC and the message is current.

B. *Requester-Server Communication*

Most of the messages sent by the Requester, R, are the same as those sent by D. These are mentioned in the first paragraph of the Driver-Server Communication above. Here, a message is first signed with the private key of R, PR_R, and $KM_{R\text{-}AS}$ is the MAC key shared between R and AS.

$$R \rightarrow AS\text{: } E\ [PU_{AS}, E\ (PR_R, ID_R \,\|\, ID_{AS} \,\|\, E\ (KM_{R\text{-}AS}, M) \,\|\, M \,\|\, T_R)]$$

The requester transmits more messages that are specific to it. Some of these messages are: new job request, items to be moved, quantity, size of vehicle (truck, Sedan, SUV), load weight estimate (heavy, medium, light), pickup location, drop off location, date and time, ASAP, modify job, and reschedule job if not selected by driver. These are treated as above using the R → AS message. However, the requester has a long message to report a problem. This is treated using symmetric key, $K_{R\text{-}AS}$, which shared between R and AS, as follows:

$$R \rightarrow AS\text{: } E\ [K_{R\text{-}AS}, ID_R \,\|\, ID_{AS} \,\|\, E\ (KM_{R\text{-}AS}, M) \,\|\, M \,\|\, T_R]$$

The AS server disseminates the following messages to R: new password, credentials accepted, and privacy policy. New password and credentials accepted are communicated using public key. However, because the policy is long, symmetric key is used.

M_1 = New password | Credentials accepted

M_2 = Privacy policy

$$AS \rightarrow R\text{: } E\ [PU_R, E\ (PR_{AS}, ID_R \,\|\, ID_{AS} \,\|\, E\ (KM_{R\text{-}AS}, M_1) \,\|\, M_1 \,\|\, T_{AS})]$$

$$AS \rightarrow R\text{: } E\ [K_{R\text{-}AS}, ID_R \,\|\, ID_{AS} \,\|\, E\ (KM_{R\text{-}AS}, M_2) \,\|\, M_2 \,\|\, T_{AS}]$$

C. *Server-Database Communication*

In this communication, there are many frequent messages. In addition, both the AS and DS perform a lot of processing. Using public key will further slow the system. Therefore, symmetric key cryptology will be used.

The Application Server transmits the following messages to DS: request to verify password, user name, security Q/A, and registration information, deactivated/reactivated accounts, rating of both drivers and requesters, feedback, blacklist update, job history update, completed jobs list, and problems (lost item, complaint, vehicle feedback, broken link). Using M to refer to any of these messages, the message sent to DS can be represented as:

$$AS \rightarrow DS\text{: } E\ [K_{DS\text{-}AS}, ID_{DS} \,\|\, ID_{AS} \,\|\, E\ (KM_{DS\text{-}AS}, M) \,\|\, M \,\|\, T_{AS}]$$

On the other hand, DS transfers the following messages to AS: password verified, name verified, security Q/A, activation/deactivation info completed, rating stored, feedback stored, blacklist updated, driver job selection updates, requester requests for service, alerts on driver/requester of blacklisted requesters/drivers, removing from blacklist completed, privacy policy, list of completed jobs, and job history. The transferred message, M, is protected as follows:

$$DS \rightarrow AS\text{: } E\ [K_{DS\text{-}AS}, ID_{DS} \,\|\, ID_{AS} \,\|\, E\ (KM_{DS\text{-}AS}, M) \,\|\, M \,\|\, T_{DS})]$$

D. AS-NPO/NGO Communication

The NPO/NGO exchanges few messages with the Application Server. Three of which, request to add user, request to remove a user, and request to join, are short. Hence, public key is used. The third message, information about the organization, is large, and therefore, symmetric key is used.

M_1 = Request to add user | Request to remove user | Request to join
M_2 = Information about NPO/NGO

$$NPO/NGO \rightarrow AS: E\ [PU_{AS}, E\ (PR_{NPO/NGO}, ID_{NPO/NGO} \,||\, ID_{AS} \,||\, E\ (KM_{NPO/NGO\text{-}AS}, M_1) \,||\, M_1 \,||\, T_{NPO/NGO})]$$

$$NPO/NGO \rightarrow AS: E\ [K_{NPO/NGO\text{-}AS}, ID_{NPO/NGO} \,||\, ID_{AS} \,||\, E\ (KM_{NPO/NGO\text{-}AS}, M_2) \,||\, M_2 \,||\, T_{NPO/NGO}]$$

The AS server will forward these messages to the Database Server after carrying out the needed decryptions and verifying the MAC. The server will send acknowledgement messages to the NPO/NGO using public key cryptography.

E. Requester-NPO/NGO Communication

Normally, the Requester should join an NPO/NGO to be able to request moving goods and resources. Obviously, a message to get information about the organization before joining, and if the user is convinced, a message to request to add the user is issued. Obviously, "information about NPO/NGO" is large.

$$R \rightarrow NPO/NGO: E\ [PU_{NPO/NGO}, E\ (PR_R, ID_{NPO/NGO} \,||\, ID_R \,||\, E\ (KM_{NPO/NGO\text{-}R}, \text{add-user-info}) \,||\, \text{add-user-info} \,||\, T_R)]$$

$$R \rightarrow NPO/NGO: E\ [K_{NPO/NGO\text{-}R}, ID_{NPO/NGO} \,||\, ID_R \,||\, E\ (KM_{NPO/NGO\text{-}R}, \text{NPO-info}) \,||\, \text{NPO-info} \,||\, T_R]$$

F. Requester-Driver Communication

Communication between Driver and Requester is needed for last minute changes to the job, check list for delivered items, and delivered item status list (good condition, damaged). The secured messages forwarded by D to R are given below. Note that Check-out list can be small or large. To be safe, symmetric key is used.

M_1 = Check out list of delivered items

M_2 = Last minute changes (driver-side) | Approved last minute changes

$$D \rightarrow R: E\ [K_{D\text{-}R}, ID_D \,||\, ID_R \,||\, E\ (KM_{D\text{-}R}, M_1) \,||\, M_1 \,||\, T_D]$$

$$D \rightarrow R: E\ [PU_R, E\ (PR_D, ID_D \,||\, ID_R \,||\, E\ (KM_{D\text{-}R}, M_2) \,||\, M_2 \,||\, T_D)]$$

For the Requester to Driver communication, we have the following relations:

M_1 = Delivered item status list | signed check out list
M_2= Last minute changes (Requester-side) | Approved last minute changes

$$R \rightarrow D: E\ [PU_D, E\ (PR_R, ID_D \,||\, ID_R \,||\, E\ (KM_{D\text{-}R}, M_2) \,||\, M_2 \,||\, T_R)]$$

$$R \rightarrow D: E\ [K_{D\text{-}R}, ID_D \,||\, ID_R \,||\, E\ (KM_{D\text{-}R}, M_1) \,||\, M_1 \,||\, T_R]$$

TABLE II. SYMBOLS USED

Symbol	Meaning
PU_{AS}, PR_{AS}	Public and Private key of AS
PU_{DS}, PR_{DS}	Public and Private key of DS
$PU_{NPO/NGO}$	Public key of NPO/NGO
$PU_{NPO/NGO}$	Private key of NPO/NGO
PU_D, PR_D	Public and Private key of D
PU_R, PR_R	Public and Private key of R
$K_{D\text{-}R}$	Symmetric key shared by D, R
$K_{D\text{-}AS}$	Symmetric key shared by D, AS
$K_{R\text{-}AS}$	Symmetric key shared by R, AS
$K_{DS\text{-}AS}$	Symmetric key shared by DS, AS
$K_{NPO/NGO\text{-}AS}$	Symmetric key shared by NPO/NGO, AS
$K_{NPO/NGO\text{-}R}$	Symmetric key shared by NPO/NGO, R
MAC	Message Authentication Code
$KM_{D\text{-}R}$	MAC key shared by D, R
$KM_{D\text{-}AS}$	MAC key shared by D, AS
$KM_{R\text{-}AS}$	MAC key shared by R, AS
$KM_{DS\text{-}AS}$	MAC key shared by DS, AS
$KM_{NPO/NGO\text{-}AS}$	MAC key shared by NPO/NGO, AS
$KM_{NPO/NGO\text{-}R}$	MAC key shared by NPO/NGO, R
HDS	Historical data store
$\rightarrow$	Then in Section III, Sends in section IV
$\leftarrow\rightarrow$	Both parties apply security requirements
T_D	Time stamp issued by D
T_R	Time stamp issued by R
T_{AS}	Time stamp issued by AS
T_{DS}	Time stamp issued by DS
$T_{NPO/NGO}$	Time stamp issued by NPO/NGO

V. CONCLUSION AND FUTURE WORK

This paper presented a security architecture for the Ford Mobility System, GoodTurn. To secure the communication between various components of this architecture, a cryptography protocol was adopted. Both symmetric key and public key cryptography were employed. Furthermore, Message Authentication Codes were relied upon. The suggested approach satisfied the security requirements; integrity, confidentiality, and authentication. The architecture will be tested and implemented when the Ford Mobility System, *GoodTurn*, is completed.

REFERENCES

[1] MacUpdate, “Xcode: Integrated Development Environment (IDE) for OS X,” https://www.macupdate.com/app/mac/13621/xcode, 2016, [retrieved: March, 2017].

[2] Swift Documentation, “The Swift Programming Language,” https://swift.org/documentation, 2006, [retrieved: March, 2017].

[3] Firebase, “Apple Success Made Simple,” https://firebase.google.com, [retrieved: March, 2017].

[4] J. McCallion, “Uber suffers massive security breach,” http://www.itpro.co.uk/data-leakage/25435/uber-suffers-massive-security-breach, 2015, ITPRO, 2015, [retrieved: March, 2017].

[5] P. Bernstein, “Bounty Hunters find Security Flaws in Uber Apps,” 2016, http://www.cloudsecurityresource.com/topics/cloud-security/articles/422447-bounty-hunters-find-security-flaws-uber-apps.htm, [retrieved: March, 2017].

[6] E. Kovacs, “Flaws Allowed Hackers to Access Uber Driver, Passenger Details,” http://www.securityweek.com/flaws-allowed-hackers-access-uber-driver-passenger-details, Security Week, 2016, [retrieved: March, 2017].

[7] T. Armerding, “Uber fraud: Scammer takes the ride, victim gets the bill,” http://www.csoonline.com/article/3059461/data-breach/uber-fraud-scammer-takes-the-ride-victim-gets-the-bill.html, CSO Online, 2016, [retrieved: November, 2017].

[8] M. Cava, “Uber to use driver selfies to enhance security,” http://www.usatoday.com/story/tech/news/2016/09/23/uber-use-driver-selfies-enhance-security/90859082/, USA Today, 2016, [retrieved: March, 2017].

[9] J. Dong, T. Peng, and Y. Zhao, “Automated Verification of Security Pattern Compositions,” Information and Software Technology, vol. 52, 2010, pp. 274-295.

[10] D. Baca and K. Petersen, “Countermeasure Graphs for Software Security Risk Assessment: An Action Research,” The Journal of Systems and Software, vol. 86, 2013, pp. 2411-2428.

[11] M. Saito, A. Hazeyama, N. Yoshioka, T. Kobashi, H. Wahizaki, H. Kaiya, and T. Ohkubo, “A Case-based Management System for Secure Software Development Using Software Security Knowledge,” Procedia Computer Science, vol. 60, 2015, pp. 1092-1100.

[12] V. Okun W. F. Guthrie, R. Gaucher, and P. E. Black, “Effect of Static Analysis Tools on Software Security: Preliminary Investigation,” in Proc. the 2007 ACM Workshop on Quality of Protection (QoP’07), Alexandria, Virginia, USA, 2007, pp. 1-5.

[13] S. Jain and M. Ingle, “Software Security Requirements Gathering Instrument,” International Journal of Advanced Computer Science and Applications (IJACSA), vol. 2, no. 7, 2011, pp. 116-121.

[14] G. Tian-Yang, S. Yin-Sheng, and F. You-Yuan, “Research on Software Security Testing,” International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 4, No. 9, 2010, pp. 1466-1450.

[15] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptology, CERC Press, 1997.

[16] N. Ferguson and B. Schneier, Practical Cryptology, John Wiley, 2003.

[17] T. Coffey and R. Dojen, “Analysis of a Mobile Communication Security Protocol,” in Proc. the first International Symposium on Information and Communication Technologies, Dublin, Ireland, 2003, pp. 322 – 328.