International Journal of Security and Its Applications
Vol. 10, No. 12 (2016), pp.261-282
http://dx.doi.org/10.14257/ijsia.2016.10.12.21

Java Implementation of a Cloud-based SIM Secure Element NFC Payment Protocol

Pardis Pourghomi[1*], Şadi Evren ŞEKER[2], Gheorghita Ghinea[3], Wassim Masri[4]

[1, 4]Technology Department, American University of the Middle East, Egaila Kuwait
[2]Department of Computer Science, Smith College, Massachusetts, United States
[3]Department of Computer Science, Brunel University London, Uxbridge United Kingdom
[1]pardis.pourghomi@aum.edu.kw[*], [2]sseker@smith.edu
[3]george.ghinea@brunel.ac.uk, [4]wassim.masri@aum.edu.kw

Abstract

A number of security protocols have been designed for mobile transactions using Near Field Communication technology in the last few years. However, the component architectures of these protocols are rarely implemented in Java for further evaluation. In this paper, we briefly discuss our previously proposed mobile transaction authentication protocol and extend our work by presenting its Java implementation. This implementation provides a detailed analysis based on a number of factors with respect to the security considerations of the protocol, particularly in its design stage. Thus, it provides a broad verification as well as step-by-step evaluation of the protocol specifications from its implementation point of view.

Keywords*: Near Field Communication; Security; Mobile transaction; authentication; Java cryptography.*

1. Introduction

A lack of interoperability and a paucity of accepted standards are major barriers towards the adoption and proliferation of NFC technologies on the service sector [1]. In terms of NFC-based payment protocols, the situation is exacerbated by the fact that current service applications provide multiple solutions for the associated ecosystem; as such, the service environment fails to fulfil minimum conditions for embracing NFC-based payments [2].

To address this issue, in this paper, we propose a technically feasible NFC-based payment ecosystem, acceptable to all its stakeholders. The main premise underlying our work is that the Mobile Network Operator (MNO) is the principal actor in the NFC ecosystem. The motivation behind this is twofold: firstly, the MNO owns the Secure Element (SE), i.e. SIM card, necessary to undertake all necessary security operations; secondly, and of particularly appeal to our work, is that the SIM card can then be configured and managed Over-the-Air (OTA) by the MNO.

Accordingly, the structure of this paper is as follows. Section 2 provides an overview of the NFC-based payment market, with its associated issues, challenges, and proposed solution. The principles behind our approach are then detailed in Section 3, whilst a proposed mechanism for Global System for Mobile Communications (GSM) authentication is explained in Section 4. Our proposed payment protocol is illustrated and briefly explained in Section 5, and a proof-of-concept Java implementation is then

[*]Corresponding Author

ISSN: 1738-9976 IJSIA

described in Section 6. Section 7 specifies Security Properties used for the implementation, and a performance evaluation of the protocol is described in Section 8. Finally, Section 9 draws conclusions and identifies opportunities for future work.

2. NFC Payment Market

Nowadays, the deployment of NFC payments are based on two different approaches: (1) Host Card Emulation (HCE) and (2) Subscriber Identity Module Secure Element (SIM SE). Major cards issuers such as MasterCard, Visa, and American Express operate their mobile payment services using the SE approach. These stakeholders, along with others, have established the Europay, MasterCard, and Visa Consortium (EMVCo), the body that defines the EMV standard for smart payment cards and payment terminals. EMVCo has also defined standards for contactless payments [3] specifically for managing contactless payment solutions such as PayPass, PayWave, and ExpressPay, all implementing the SE technology. The solution consists of having a secure and tamper-resistant smart card protected by strong encryption, which can be accessed only by certain elements in the system, typically called Trusted Service Manager (TSM).

On the other hand, HCE-based solutions use the cloud to store sensitive data rather than providing local storage options such as device memory, SIM or SE solutions. First introduced on the Blackberry platform, HCE technology was adopted by Google in 2013 and was released on Android 4.4 KitKat devices, both of which have contributed greatly to its widespread. In HCE, data received from the NFC reader is transmitted to an application that is located on the Host's Central Processing Unit (CPU). In fact, data is relayed through the NFC transmitter and the antenna without the involvement of SE. Data is then processed and the cloud will be contacted to settle the transaction. HCE will hence allow users to choose an app of their choice to make payments, transforming that app to a mobile wallet, which further helps them to avoid facing potential issues with provisioning and personalisation of the SE approach before and after the card issuance [4].

Since the arrival of HCE-based solutions to the market, there have been various debates and views [5, 6] about the scheme of processing NFC mobile wallets. The main focus of those debates is yet around selecting either the HCE or SE based transactions models.

The level of flexibility that HCE brings to stakeholders is undoubtedly more than what the traditional SE approach (where MNO and bank are disintegrated) has to offer. In order to process the transaction, the service provider does not need to rely on the SE or the TSM which itself arguably makes the route to market smoother and cheaper. However, the superiority of the SE approach, either SIM SE or embedded SE, is proven to MNOs and handset manufacturers. In fact, the operation mechanism of the two approaches is very different in terms of involved entities, security, usability, etc. nonetheless, we believe that looking at these two approaches as competitors won't assist the existing concerns but rather that these approaches should be incorporated as one payment processing mechanism.

Usually, the HCE-based application runs on the hosting CPU beside other applications, which makes it vulnerable to security threats such as theft of sensitive data or integrity violation, whereas SE solutions offer a highly secure storage place for the emulated card's sensitive information, hence protecting it from unauthorized access. Moreover, it is to be noted that even though the HCE-based service is protected by the system permissions where only the Operating System can exchange data with the service, it is still possible [5] to tamper the service itself or to provide it with falsified data from a malware that may corrupt the NFC transaction or eavesdrop on sensitive data.

On the other hand, HCE offers more flexibility by liberating service providers from many constraints imposed by the SE's owners and significantly reduces the payment process by using tokenisation. However, tokenisation is considered as a newly introduced paradigm which has not yet satisfied its surrounding security risk issues. In [7, 8, 9, 10], the SIM secure element approach recommends though an MNO-centred model, where the network operator is in charge of provisioning and managing the SE OTA. In contrast, embedded SE has always been an interest to handset manufacturers, where they can take a bigger role as the SE is embedded in the handset and not in the SIM; subsequently, there is not much expected from MNOs in this scenario. Generally, the SE solutions are commonly recognized to be more secure, but perhaps less granular in control which require a more complex and binding ecosystem for service delivery. Having said that, the growth of the society is dependent on the technology, and while HCE serves the NFC ecosystem, the technology remains with no standards and connected to the SE-based deployments.

The following section provides an overview of the leading NFC payment stakeholders in the market, where some rely on the HCE technology (e.g. Google Wallet) while others follow the more traditional SE solutions (e.g. Visa payWave).

2.1. Leading NFC Payment Providers

This section provides an overview of the current prominent mobile payment services deployed on a global scale.

(1) Visa payWave

Visa payWave [11] is an NFC-based technology that allows users to make payments by waving their cards in front of secured readers instead of inserting or swiping them. To minimize fraudulent transactions, Visa limits the purchase amount to $100, over which swiping or inserting the card will be necessary. Visa also made it possible to use the same technology with an NFC-enabled mobile device using the payWave application to make payments. Instead of using the card, the user will just wave his device near the reader (similar to any other NFC-based technology) and the transaction's encrypted information will be sent to the reader. The two main advantages of the payWave technology [12] are (1) reduced queuing time and (2) easy integration with existing cards. The Visa payWave ecosystem consists of several entities [13] that mainly hold root access to the secure element while managing the secure element's lifecycle; it provides OTA provisioning, and enables issuers to communicate OTA with the consumer's mobile device to manage the lifecycle of Visa payment accounts.

(2) American Express Expresspay

American Express introduced a contactless RFID-based solution [14] to offer their customers to make payments when their contactless cards or devices are in a close proximity to the contactless reader facilitating the "pay and go" service. When a transaction is run through a contactless terminal, information is encrypted and securely sent to the host with the purchase details. Before any contactless transaction takes place, the merchant will send all purchase details to the reader so the customer's card won't be charged accidentally if it gets too close to the terminal. The card or the device should be then held for more than half a second within 4cm of the contactless reader [15] in order for the transaction to be completed. Expressway follows EMVCo contactless specifications and hence it uses the SE-based solution where the chip on the card or the device will hold all the payment cards' information.

(3) Google Wallet

One of the major companies which operate the concept of Mobile Wallet is Google. In 2011, Google released the "Google Wallet" service [16, 17], a mobile payment system that collects payment cards, loyalty cards, and gift cards all in one place. At the beginning, Google Wallet 1.0 used the device-based SE system for card emulation, but soon almost all major network operators decided to support only their wallets' Secure Elements and blocked the access to all others. Google then had to change its design and to use HCE instead starting from Google Wallet 3.0, where the communication between the mobile phone and the Point-of-Sale (POS) is carried out when the phone is tapped against a contactless terminal through NFC technology that redirects communication from the terminal to the host operating system. Google Wallet, which is present on the device as an Android application, will then respond to the request by the corresponding virtual credit/debit card number created during the setup and will contact Google servers to map it to the real card number. In that scheme, the SE present on the device will only store the virtual card number, while the real card numbers will be stored on the secured cloud servers of Google. The transaction will then be finalized through the virtual card that transfers funds from Google Wallet into the merchant's POS.

On February 23, 2015, Google acquired the intellectual property of the card company SoftCard, and used its technology to roll out a new POS payment service called Android Pay [18]. As such, Google Wallet no longer supported point-of-sale payments as of late 2015, and Google introduced their new Google Wallet that supports peer-to-peer payments where users can cash out money received directly to their bank accounts, or even withdraw it from participating ATM using the Google Wallet Card [19].

(4) MasterPass

"MasterPass"[4, 20] is a service introduced in 2013 which has been developed by MasterCard as an extended version of PayPass Wallet Services [21] announced in 2012. MasterPass provides a digital wallet service for secure and convenient in-store and online shopping where delivery information and transaction data are stored in a central and secure location. MasterPass provides the following services [21]:

- *MasterPass checkout services:* This service enables the vendor's payment acceptance in a consistent way irrespective of the client's location. This means vendors have the ability to accept a payment without having to know where the client is. For instance, when the client is in store, he can use this service since it supports NFC, QR codes, tags, and mobile devices to pay for products at a vendor's POS. Similarly, a client can use the service to pay for a product he is purchasing online without having to enter the card and delivery details in each transaction conducted.
- *MasterPass-connected wallets:* Vendors, financial institutions, and partners are able to provide their own wallets using this service. The client's card information, address books, etc. can be saved in a secure cloud provided by a party they trust. Thus, clients can use other credit and debit cards in addition to their Mastercard cards.
- *MasterPass value added services:* the purpose of this service is to improve the client's shopping experience before, during and after checkout. Value added services include account balances, offers, loyalty programs, and real-time alerts.

In 2015, MasterCard launched an HCE-based project [22] in an attempt to keep up with the competition with other companies such as Visa, Apple and Google, all of which were heading toward using the same technology. MasterCard's new project

takes advantage of the new HCE technology as an alternative to the SE-based approach for digitizing card credentials into mobile devices, and storing critical information on the secure cloud instead. This approach is intended to simplify the deployment process of contactless mobile payment services for financial institutions and interested merchants by offering a more convenient and secure experience to their customers. Moreover, the company has recently released a Software Development Kit (SDK) for developers that includes a collection of libraries and tools to help them easily create applications compliant with the company's specifications, in a move intended to further extend the development of their mobile application on Android devices with KitKat 4.4 or above.

(5) Apple Pay

Apple Pay is an NFC-based technology that allows users to pay using their mobile devices (iPhone 6, iPhone 6s, Apple Watch) at any contactless POS, or even pay in iOS apps (for iPad Pro, iPad Air 2, iPad mini 4, iPad mini 3) [23]. Following the conventional two-factor authentication, the user will only need to hold his finger on Touch ID while keeping his device near the POS for a few seconds to make a payment.

The user first has to register his debit or credit card to Apple Pay. The information sent to Apple servers is encrypted [24] and only the part pertaining to the payment network is decrypted by Apple. The information is then encrypted again and sent along with some other information (e.g. phone number, name, location, etc.) to the bank who will decide whether or not to register the card.

Once the card is approved, the bank will create a device-specific Device Account Number, unique for that card, encrypts it, and sends it back to Apple along with the key used to generate dynamic security codes. Apple will not be able to decrypt this number but will add it to the SE physically present on the device. When the user decides to make a payment, s/he must authenticate using his Touch ID or a passcode. The SE will then send the card's Device Account Number and a transaction-specific dynamic security code to the store's POS which will contact the bank to finalize the transaction. The bank will be able to verify the payment information by checking the dynamic security code to make sure it is unique and that it belongs to the user's device.

3. Our Approach

As we discussed in [25], we believe that the SE provides a higher level of security compared to the HCE, thus our approach uses SE as the prominent component of the service. In our approach SE is partitioned into two sections; (1) stored in the SIM for authentication of a customer, and (2) stored in the cloud to store the credit/debit card details of the customer. This helps in managing multiple cards against a single customer. The authentication of the customer to the MNO is based on GSM authenticating mechanism with improved security features. The customer selects one of the already registered accounts to be used for transaction. Our protocol works on a similar pattern to "PayPal": the MNO acts as the PayPal and a user registers multiple banking cards for monetary transactions with the MNO. The user then selects a single card for monetary transactions at the time of the payment. The most important step in the mobile payment transaction is the SE, which holds all the authorization power. Whether it is a chip in the phone, or functions virtually in the cloud, the SE is tamper-proof and protected by a unique digital signature. As explained by Infineon's director, which manufactures secure element chips, the architecture of the SE is designed to be hardened against attacks on the phone. "*That includes software attacks but also hardware-based attacks*

where someone got your phone or SIM card, it would be extremely difficult to obtain info off of that because it's a chip that is designed to have security mechanisms that go well beyond a normal processor" [26].

4. GSM Authentication

When a mobile device signs into a network, the MNO first authenticates the device (specifically the SIM). The authentication stage verifies the identity and validity of the SIM and ensures that the subscriber has authorized access to the network. The Authentication Centre (AuC) of the MNO is responsible for authenticating each SIM that attempts to connect to the GSM core network through the Mobile Switching Centre (MSC). The AuC stores two encryption algorithms A3 and A8, as well as a list of all subscribers' identity along with the corresponding secret key K_i. This key is also stored in the SIM. The AuC first generates a random number known as R. This R is used to generate two responses, signed response S and key K_c as shown in Figure 1, where $S = E_{Ki}$ (R) using A3 algorithm and $K_c = E_{Ki}$ (R) using A8 algorithm [27].

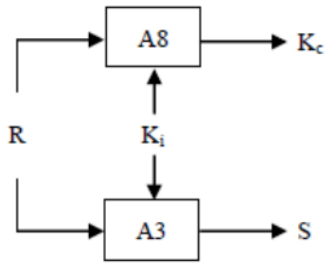

Figure 1. Generation of K_c and S from R

The triplet (R, S, K_c) is known as the Authentication triplet generated by the AuC. The AuC sends this triplet to the MSC. On receiving a triplet from the AuC, the MSC sends R (first part of the triplet) to the mobile device. The SIM of the mobile device computes the response S from R, as K_i is already stored in the SIM. The Mobile device transmits S to MSC. If this S matches the S in the triplet (which it should in the case of a valid SIM) then the mobile is authenticated. K_c is used for communication encryption between the mobile station and the MNO.

4.1. Comp 128, A3/A8 Algorithm

The algorithm is widely used by all the GSM encryption and decryptions. The algorithm is confidential, thus the details of algorithm are not available for the implementation. Fortunately, the algorithm has a reverse engineered version first time implemented in C and we, for the first time, have converted the C code into the Java language.

The algorithm simply gets a random number R of 128 bits and a key K_i of 128 bits and generates a 96 bits output. The output of 96 bits is also a merged version of K_c and S values. The first 32 bits of output is K_c and the remaining 64 bits are S from the original implementation in C. We have also followed a similar approach.

Please note that the implementation is done only for simulation purposes; the real life implementation of the protocol will not require such an implementation since the algorithm is already embedded into the related GSM functions. Also detailed information about algorithm and cryptanalysis of the algorithm is discussed in [28]. Table 1 describes the abbreviations used in the proposed protocol.

Table 1. Abbreviations

AuC	Authentication Centre (subsystem of MNO)
MAC	Message Authentication Code
App_{ID}	Approval ID. Generated after credit approval
Acc_{ID}	Account ID of the customer
C_{r_req}	Credit Request Message
C_{r_app}	Credit Approved Message
IMSI	Internet Mobile Subscriber Identity
K_i	SIM specific key. Stored at a secure location in SIM and at AuC
K_c	E_{ki} (R) using A8 algorithm
K_1	Encryption key generated by shop
K_2	MAC key generated by shop
K_{pub}	Public key of MNO
K_{pr}	Private key of MNO
K_{sig}	Signing key of MNO
K_{ver}	Verification key of MNO
LAI	Local Area Identifier
MNO	Mobile Network Operator
R	Random Number (128 bits) generated by MNO
R_s	Random number generated by SIM (128 bits)
SE	Secure Element
TM_m	Transaction Message for mobile
TM_s	Transaction Message for shop
TMSI	Temporary Mobile Subscriber Identity
TP	Total Price
T_{SID}	Temporary Shop ID
TS_s	Shop Time Stamp
TS_t	Transaction Time Stamp

5. The Previously Proposed Protocol

The general overview of the cloud-based NFC payments is described in [7], where the NFC Cloud Wallet model is also proposed. We then proposed an extension to the previously proposed NFC Cloud Wallet model and designed an NFC payment protocol which was based on a GSM network [8]. This protocol was the improved version of Chen's protocol [29], in which user interaction with the system was improved, making it more user friendly. Since there were multiple options applicable to this model, we designed our protocol based on the following assumptions:

- The SE is part of SIM
- The cloud is part of the MNO
- The MNO manages the SE/SIM
- Banks, etc. are linked to the MNO

As illustrated in Figure 2, the proposed protocol is based on a cloud architecture, in which the cloud is managed by the MNO. The SE used in this protocol is divided into two domains: one, being a part of SIM, is used for authentication of a customer, whereas the other section, being a part of cloud, is used to store sensitive banking information of the customer. Interested readers are referred to study the full description and analysis of this protocol in [8].

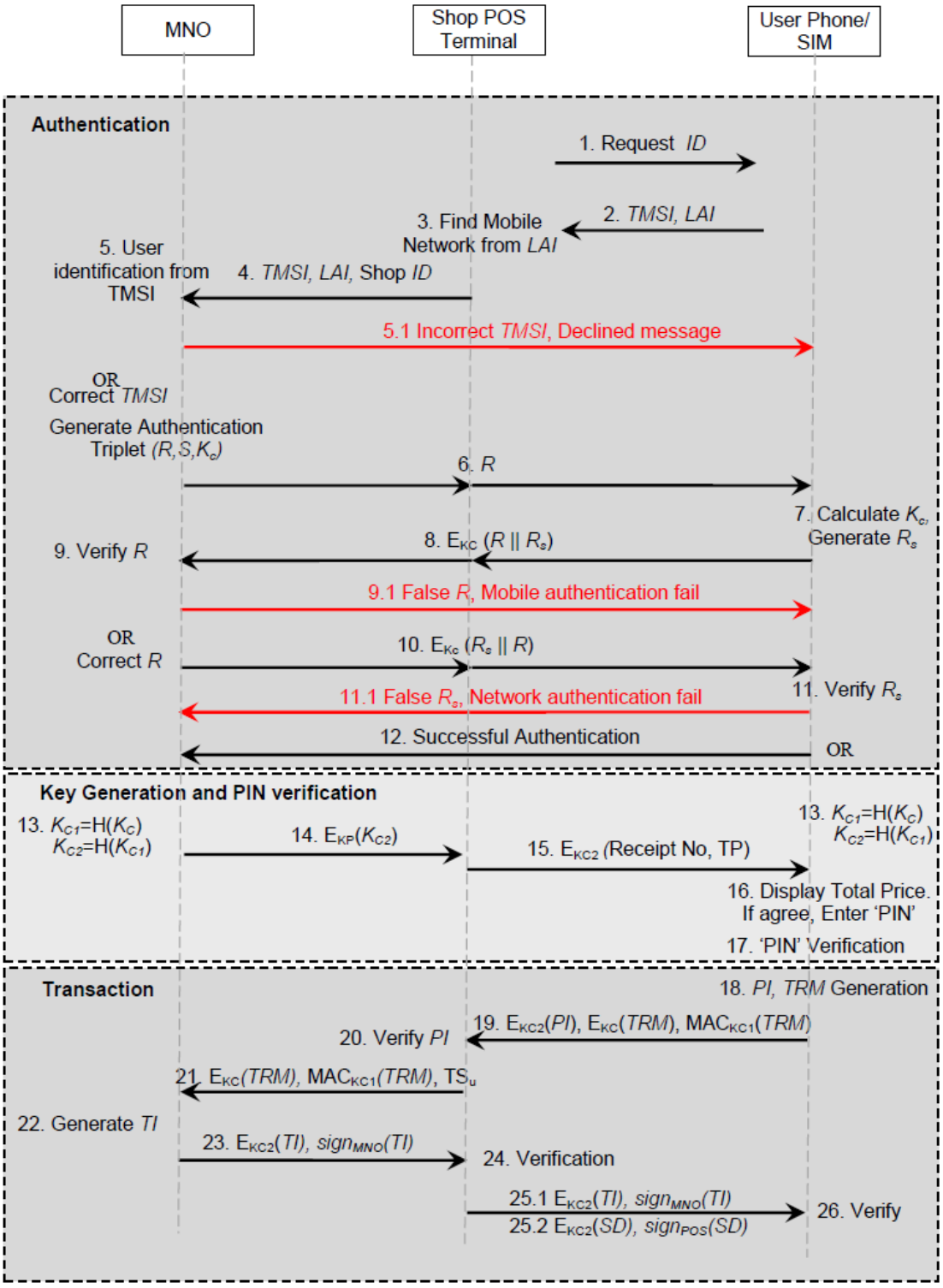

Figure 2. Overview of the proposed protocol [30]

6. Protocol Implementation

This section discusses the details about the protocol's simulation in Java. The protocol is simulated in a real environment with three peers and the simulation software of each peer is implemented in the Java language. The peers can be a Java 2 Platform, Enterprise Edition (J2EE) server, android cellphone and javaPOS or any other platform, which is supporting Java.

During the code development, the protocol is implemented in Java using an object-oriented approach. There are three objects representing each peer of the protocol, which are MNO, POS and NFC-enabled Cell Phone (CP), where all objects can be moved into a real environment or kept as simulation objects. The implementation also has a library of functions for encrypt/decrypt functions, MAC, Hash, functions or time stamp calculations.

For example the A3/A8 encryption algorithm of all GSM mobile devices is implemented for the first time in Java and, since it is a confidential algorithm, we have also implemented (again, in Java) for the first time the reverse engineered version of algorithm by Marc Briceno, Ian Goldberg, and David Wagner in 1998.

The section will start with the initial definitions of simulation environment setup and the details of each class and its role in the simulation will then be introduced. The protocol will also be handled in a step-by-step approach. Protocol phases, related objects, message passing between objects, decisions and alternatives of each step will be discussed.

The protocol is designed in 26 steps, yet due to some implementation limits and atomicity, the implementation is handled in 15 steps in addition to the initial environment setup phase.

A. Object oriented design of simulation environment

Each protocol object is represented in an individual class. The basic three classes, which are MNO, POS and CP, represent the MNO, shop Point-of-Sales and NFC-enabled Cell Phone peers of the protocol. Message passing between objects is also handled by message passing interfaces. Each message has a unique gate of data transfer; also each class has definitions from the factory settings. For example, the cell phone's SIM card has a built in factory key for encryption and decryption algorithms. That type of information is initially loaded into the object variables during the first time creation from the class.

Besides the three objects for three peers, there is an observer object to track the steps and signal related object calls. The observer is mainly implemented to track the step of the algorithm. It also stops the further execution of the protocol while the user clicks on the "next step" button from the simulation interface. Another group of classes are implemented as a library to the rest of the classes. For example, encryption or hashing algorithms are kept in these libraries. Finally, a user interface of the simulation is implemented as both an applet and an application. The applet interface is loaded into the web interface for simulations from the web. The application is the downloadable version of the simulation, where users can download and run it offline.

The Unified Modelling Language (UML) class diagram in Figure 3, demonstrates the details of each class and the message passing at every step of the protocol. Each class has a name indicating its peer such as MNO for the MNO from the protocol, Shop for the "Shop POS Terminal" in the protocol or CellPhone for the "User Phone/SIM" from the protocol. The SESeker_Pardis class is the observer class, which is responsible for tracking the steps of the simulation, gathering information and reporting protocol related details for the simulation. Finally, an Applet class is implemented for the graphical user interface.

The rest of the classes, like A3/A8, SHA-1, CipherUtils, or MAC are responsible for the library functions of the protocol. Implementation details of the classes will be given in

the flow of the protocol. So the protocol will be handled in a step-by-step manner and the related method invocations and related libraries and objects will be explained in detail on the rest of the subsections. Before starting the step-by-step details of the protocol the required algorithms will be presented in subsections.

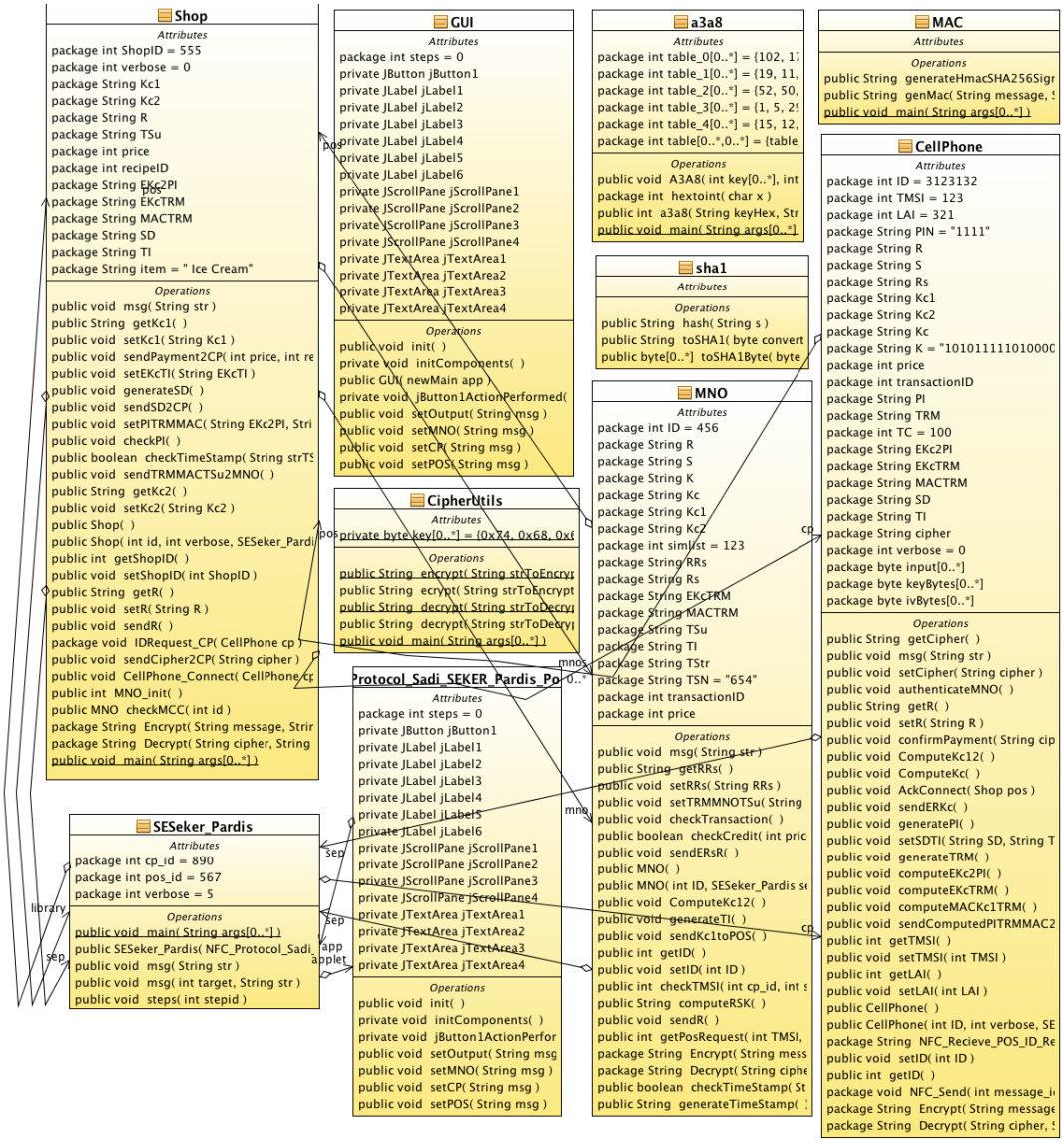

Figure 3. UML Class Diagram of Simulation Software

6.1. Implementation Stages

Initial Step

In this step, the objects are created from the classes and the variables are initiated. For example the secret keys from factory setup or cell phone ID, or Shop ID or the MNO values of TMSI and LAI are predefined in the real world and we have implemented those values into the related variables for each peer. During the simulation, one object of cell phone, one object of shop POS and three objects of MNO are created, as depicted in Figure 4.

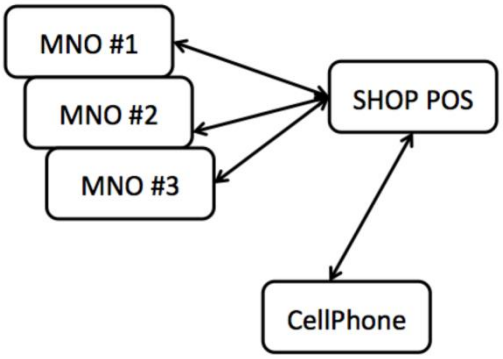

Figure 4. Initial Setup of Protocol Objects

The reason for having multiple MNOs is the simulation of multiple telecom operators in the environment. Thus, the POS of shop searches the intended MNO for the cell phone to communicate with and verify, as detailed in the further steps below.

Step #1

In this step, the cell phone gets into the range of the POS and the initial data transfer starts between the POS and cell phone.

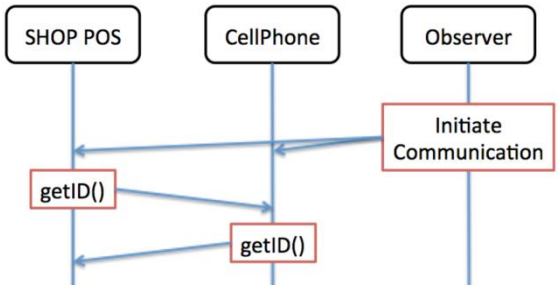

Figure 5. UML Sequence diagram of Step #1

The UML sequence diagram of initial communication between shop and cell phone is demonstrated in Figure 5. The observer object initiates the communication between cell phone and shop POS. The initialization is the simulation of a cell phone getting into the range of shop POS and just after the communication initialized, both of the peers, the POS and cell phone reads each other's IDs by method calls to the getter/setter methods of ID variables.

Step #2

On the second step of simulation, the observer object starts a connection from POS to the cell phone. The connection protocol indicates the reading of LAI and TMSI values from the cell phone as indicated in steps #2 and #3 of the protocol. The values are read back to the shop object and stored into the object variables.

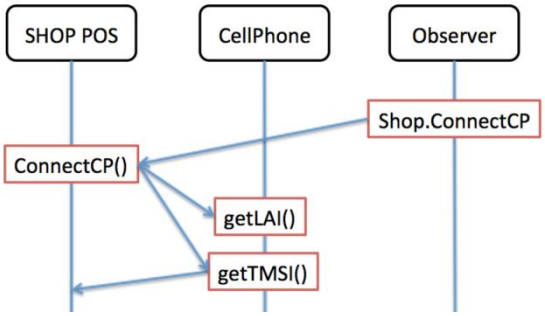

Figure 6. UML Sequence Diagram of Step #2

During the initialization, the function call to POS initializes the reading of two variables from the cell phone object which are responsible for returning the LAI and TMSI values. The whole inter-object messaging is demonstrated in Figure 6.

Step #3

After receiving the LAI and TMSI from the cell phone, the shop initiates a search of related MNO, which the cell phone is using as a telecom operator. The search is done on a database with all MNO information stored as in Figure 7.

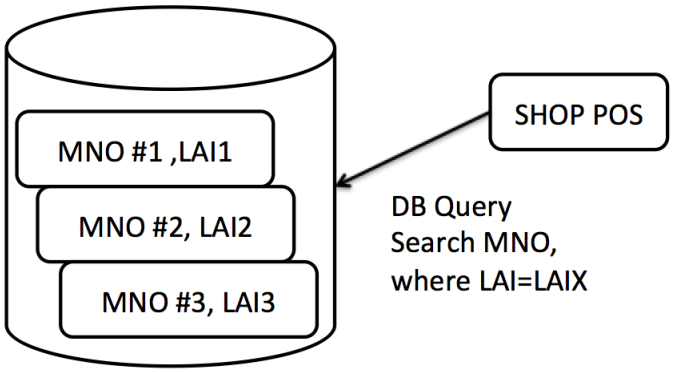

Figure 7. Demonstration of finding MNO from the Database

During the query, the shop POS makes an SQL select query to the database. The result may be an empty set (Null) or one MNO information with supplied LAI parameter. The algorithm has two alternatives at this step, if the MNO is not recognized from the database, the communication can be terminated from the POS side, or an object referrer to the returned MNO information is created in the POS object.

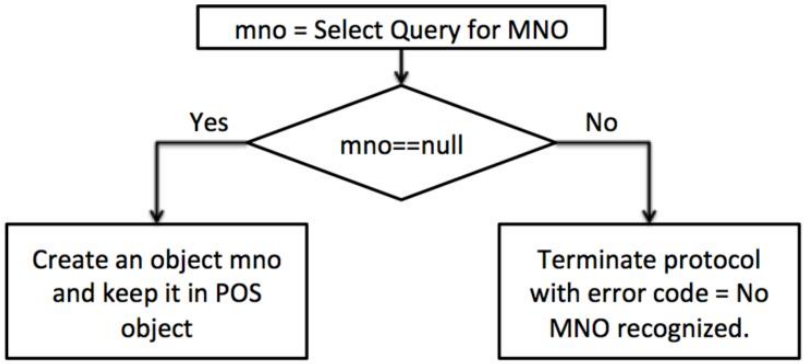

Figure 8. Flow Chart of MNO Selection Algorithm

In parallel to the flow chart in Figure 8 and depending on the return value of the database query, the protocol is either terminated by the POS object or a new object of MNO is created and stored into a variable "MNO" in POS object.

Step #4

In this step, the MNO calculates a random number R and encrypts this random number with a key using the A3/A8 algorithm. The details of the algorithm is already given in section 4.1. The random number generation is done through the Random class of Java Standard Edition. Java Standard Edition has a built in random generator from a space for 2^{48} and the random generator uses linear congruential generator as given in equation (1) [30].

$$n_{i+1} = (a \times n_i + c) mod\ m \qquad (1)$$

Here n_i is the i^{th} element of the random generation series and (a, c) is a couple of linear equation. Additionally, the initial number is seeded as usual by the time of computer. The problem of random generator is its output limit, which is only 32 bits. In the protocol, the R value is 128 bits and in order to complete the 128 bits, the random generator is executed four times with different seed values. Also, the R value is stored in a String variable type. After the generation of the random value R, the value is transferred from the MNO to the shop's POS and the shop's POS to the cell phone.

Steps #5, #6 and #7

The cell phone calculates K_c in the same manner as the MNO, with the secret key stored in the SIM card via A3/A8 algorithm. Again the key is stored in a string variable.
In this step, another random number R_s is generated by cell phone with the random generator, similar to the previous step. The random number R generated by MNO and transferred to the cell phone from previous step and random number R_s generated by cell phone in this step are concatenated and encrypted using the Advanced Encryption Standard (AES). This step represents the first instance an encryption algorithm is called. The encryption function is stored in CipherUtils class and uses javax.crypto.Cipher class. The java code for object creation is quoted below:

```
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
```

The algorithm uses AES with the electronic codebook option. Moreover, the missing blocks are handled by the Public Key Cryptography Standards (PKCS#5) padding

algorithm, where the missing blocks are filled with the number of missing blocks[1]. In addition to the AES output with Electronic Codebook (ECB) mode encryption as well as PKCS#5 padding, we also put the output of encryption to BASE64 decoding. The reason of implementing the BASE64 algorithm is to convert any input string to a byte array and thus handle the differences of character sets. For example, an output in a Chinese character set would contain undefined characters in US-English character table and this would be a problem in the AES encryption/decryption steps. In order to avoid such problems, we simply convert the output of encryption algorithm via the Base64 algorithm.

```
final String encryptedString =
Base64.encodeBase64String(cipher.doFinal(strToEncrypt.getBytes()));
```

The code of the final conversion in Java is given as above.

Step #6 is very similar to step #5, with the only difference being that the same code is implemented from the MNO side, way back to the cell phone peer. Finally at step #7, the two random numbers R and R_s are compared on the cell phone side and, if they are equal to each other, the MNO is authenticated. The implementation is nothing more than a simple string comparison.

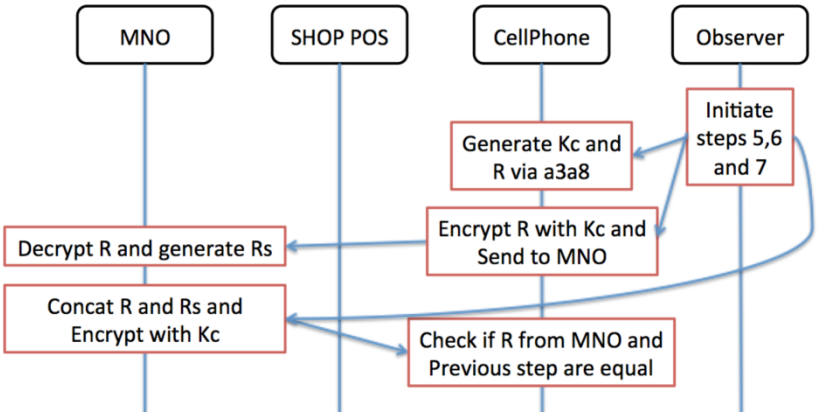

Figure 9. UML Sequence Diagram of Step #5,#6 and #7

The UML sequence diagram of steps #5, #6 and #7 is depicted in Figure 9. Apart from the message passing parts in Figure 9, there is also a validation of R values by the cell phone.

Step #8

In this step, the key values are generated from the factory implemented key values via hashing algorithms. In the simulation, we have implemented the SHA-1 hashing algorithm which outputs 160 bits; however, we have implemented the protocol over 128 bits only. The problem has been solved by removing the last 32 bits of the SHA-1 output. The reduction of bits is applied after both hashing and double hashing has taken into action, so the implementation of algorithm can be summarized as below.

[1] The password-based encryption algorithm as defined in RSA Laboratories, "PKCS #5: Password-Based Encryption Standard," version 1.5, Nov 1993. Note that this algorithm implies *ECB* as the cipher mode and *PKCS5Padding* as the padding scheme and cannot be used with any other cipher modes or padding schemes.

$$K_{c1}\_temp = SHA\text{-}1\ (K_c)$$
$$K_{c2}\_temp = SHA\text{-}1\ (K_{c1})$$
$$K_{c1}=ReduceBits\ (K_{c1}\_temp)$$
$$K_{c2}=ReduceBits\ (K_{c2}\_temp)$$

After the bit reduction, both K_{c1} and K_{c2} are 128 bits long. Also, the sharing of K_{c1} to POS is done in this step.

Step #9

In this step, the payment is requested by the shop's POS and the request is sent to the cell phone through an encrypted channel with key K_{c2}. After receiving, the cell phone side can confirm the payment with the Personal Identification Number (PIN) verification from user. This verification is simulated by a user input from the keyboard. Again, a pre-stored integer value is checked for verification for simulation purposes. If the PIN entered by user, during the simulation, does not match the pre-stored PIN, then the protocol is terminated.

Steps #10, #11 and #12

In this step, a time stamp is generated for the first time. The time stamp value is kept in objects created from java.sql.TimeStamp classes. The PI variable is created from the price, transaction ID and the time stamp. The time stamp is in the form of year, month, day, hour, minute, second, millisecond format. A sample time stamp generated is given as below.

2014-11-21 19:54:52.272

The time stamp is converted to a string and then concatenated to the price and transaction ID from the POS. The data is transferred to the cell phone. After the PI is generated, the variable holding PI string is attached to the TRM with the transaction counter, which is an integer counter with 5 digits during the simulation, and the random value R_s from the previous steps.

The strings thus concatenated at this step should be able to separate in the next steps. For this reason, delimiter symbols are added during the concatenation. For the simulation, the ";" symbol is added besides all the strings merged.

In step #11, the PI and TRM values are encrypted using the AES algorithm from step #5 and also the message authentication code is generated from TRM. It is the first time that the MAC is generated, and in order to generate the MAC, again Java cryptography expansion is utilized. An object created from javax.crypto.spec.SecretKeySpec class with key value is used for the key encryption. Also, the class SecretKeySpec has a property of creating an object with "HmacSHA256" parameter[2].

After the computation of encrypted versions of PI and TRM and MAC value of TRM, the data is transferred from the cell phone to the MNO through the POS.

Also in step #12, the MNO first decrypts and then tokenizes the string into pieces and checks the time stamp using a new function implemented with time stamp and time window. The function simply adds the given time window as seconds into the given time stamp parameter and checks if the current time is after the parameter timestamp and before the created timestamp with the window.

[2] The HmacSHA* algorithms as defined in RFC 2104 "HMAC: Keyed-Hashing for Message Authentication" (February 1997) with SHA-* as the message digest algorithm

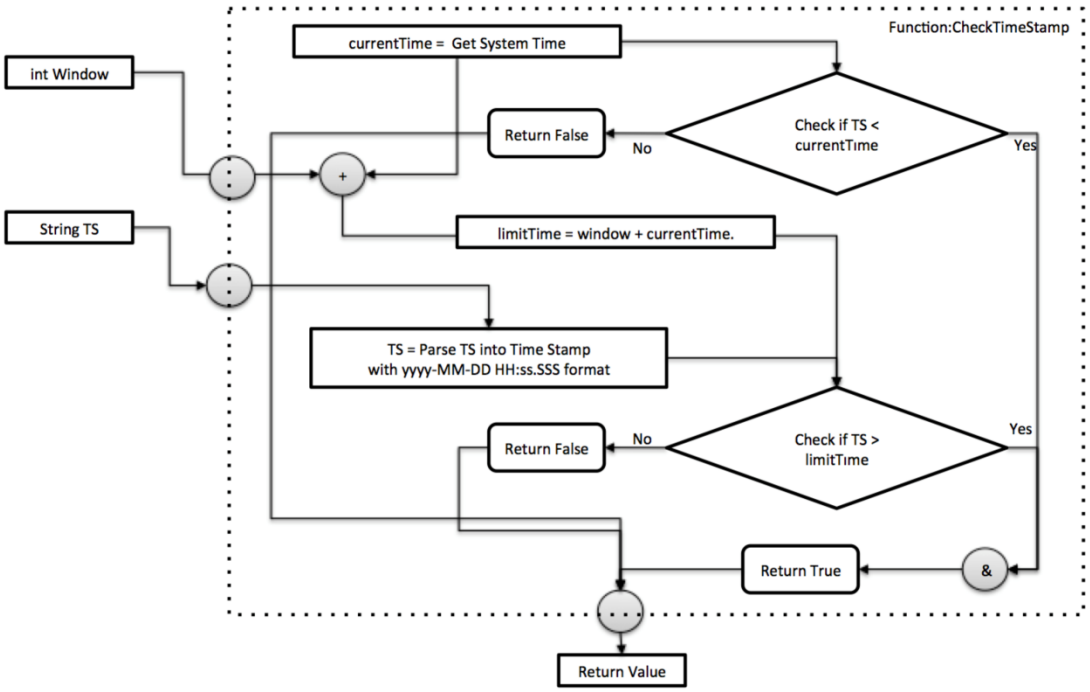

Figure 10. Data Flow Diagram of "Check Time Stamp" function

The data flow of function “CheckTimeStamp” with two input and one output points is illustrated in Figure 10.

Steps #13, #14 and #15

In these final steps, the shop generates TI and SD values, encrypts them and sends them to the cell phone. During the implementation of simulation the previously implemented functions are used. The confirmation and format of recipe is not taken into account during the simulation.

6.2. User Interface of Simulation

During the simulation, the user interface is designed to hold 4 text areas and a button. The button is designed to be clicked from each step. A step counter tracks the actions and starts or stops the step. The step numbers of the protocol and its simulation are not identical; the step numbers of the simulation are already given above and a counter keeps track of these steps.

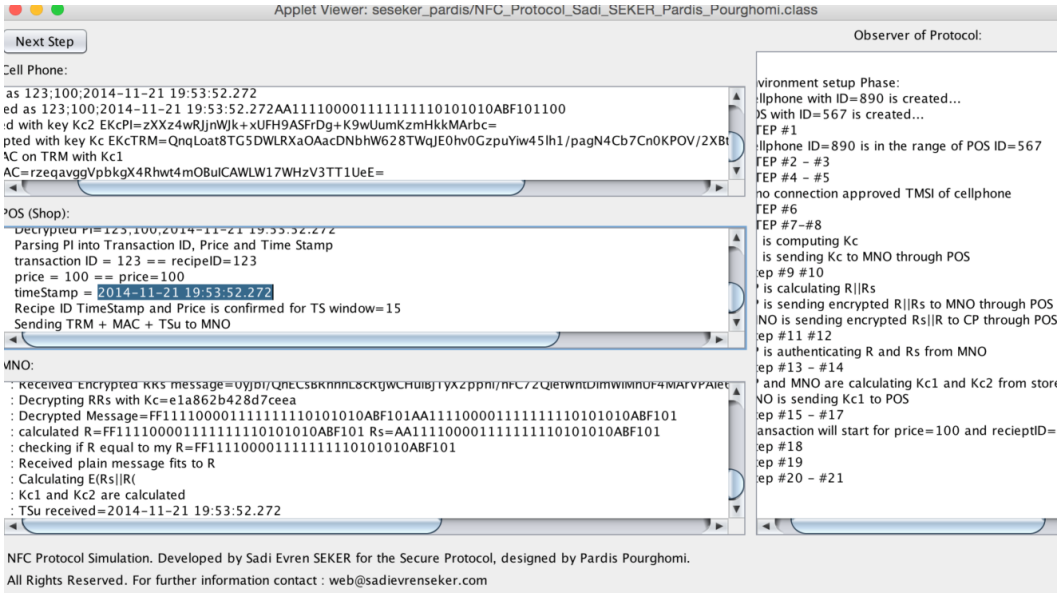

Figure 11. Screenshot of Simulation GUI

All the peers are represented with a separate text area. The interface, demonstrated in Figure 11, holds three output text areas for cell phone, POS and MNO and a final text area for the observer. The observer mainly displays the steps and some critical messages such as the reason of termination being displayed at the observer when the protocol is terminated.

A "next step" button is also deployed for the user to jump from step to step. Each time the "next step" button is clicked, the step counter on simulation is increased and the related part from the observer object triggers the related code block.

7. Security Properties used for the Implementation

During the implementation of the protocol, we have used the following algorithms:

1. AES Algorithm for all the encryption and decryption operations
2. ECB option for the AES algorithm
3. PKCS5 Padding option for the AES algorithms
4. BASE64 for arranging the input size
5. SHA-1 for hashing
6. MAC for related parts of the protocol

Alternatively, AES would be replaced with any symmetric encryption algorithm and parallel to this change, ECB and PKCS5 would be updated. SHA-1 can also be replaced with any hashing algorithm.

The sizes of the key and random variables also defines the level of security. They are originally defined as 128 bits but can be updated to 256 bits for a higher level of security. The SHA-1 and AES and MAC algorithms also support this update.

The performance of execution with 128 bits or 256 bits does not make a significant difference. The amount to be processed doubles and also the communication times increases, but the simulation tests show the whole processing can be done in a few seconds and communication is less than 1 KB of information for each step.

8. Performance Evaluation of Simulation

The performance of simulation can be crucial information for the performance of the protocol. The performance can be evaluated from different aspects and can be listed as below:

1. Time Performance
2. Processor Performance
3. Memory Performance
4. Networking Performance

Although all the above performance evaluation criteria are related to each other, they can be monitored separately. Time performance is less than a second and depending on the current load of the computer it takes between 100 milliseconds to 150 milliseconds.
Since the algorithmic complexity does not change, the processor and networking performance is much more related to the memory performance. The total size of memory required for execution of the whole simulation, including the three peers and observer object is 55592 bytes. The top 6 object and variables in simulation can be listed as in Table 2.

Table 2. Memory Performance of Objects and Variables in Simulation

Class Name - Live Allocated Objects	Live Bytes	Live Objects	Allocated Objects
char[]	12120	168	2069
int[]	4552	62	323
java.util.HashMap$Node	4320	135	417
java.lang.String	4104	171	1461
java.util.LinkedHashMap$Entry	4040	101	101
java.lang.Object[]	3976	18	260

Since most of the data is kept in string types, char arrays are widely used during the string operations like parsing or splitting or merging etc. In the development of simulation, the random numbers, keys or hash outputs are all kept in integer arrays or strings. On the other hand the LinkedHashMap entry is for the hashing and object array is for the objects in simulation.

Increasing the key size or random numbers generated during the protocol can increase the memory size and this increase would affect the processor and networking time because of the increased volume of data.

Even considering the worst case, when the amount of data generated is 10 times bigger, which would be about 500 KB, neither the CPU, nor the Memory of POS or Cell phone would have a problem. For example, currently all the NFC supporting phones are smart phones and they have a CPU with processing power higher than 1GHZ with multiple cores in most of the cases, iphone or Samsung galaxy phones can be examples of that. Also, the CPU in most POS devices with NFC support are higher than 1.5 GHz. Thus the memory in both cell phone and POS devices is more than 1GB. According to the simulation results, we do not expect any problem related to the CPU, memory or network load.

8. Conclusion

NFC-based payment ecosystems still suffer from a lack of interoperability and accepted standards. To address these twin challenges, an NFC-based payment protocol centred around the MNO has been put forward in this paper. Additionally, we have also

provided its implementation details in Java. In so doing, not only have we demonstrated its feasibility and potential acceptability to all ecosystem stakeholders, but it has also enabled us to undertake a detailed evaluation of the proposed protocol's performance.

Specifically, in terms of time, memory, processor, and network requirements, the novel protocol detailed in this paper was shown to have entirely feasible and reasonable demands, easily met by current hardware specification of contemporary cell phones and POS terminals alike. This bodes well for its acceptance by stakeholders.

Future work will explore alternative potential ecosystem scenarios for secure mobile payment services with the aim of design and implementation of secure protocols for such potentials. Moreover, two other protocols that have initially been proposed by us in [9][10] will be implemented in Java for future comparison with the one we discussed in this paper in order to reach a comprehensive conclusion as to which one of them would suit the current market demands the most.

References

[1] G. Antoniou and L. Batten, "E-commerce: protecting purchaser privacy to enforce trust", Electronic commerce research, vol. 11, no. 4, **(2011)**, pp. 421-456.
[2] R. J. Kauffman, J. Liu and D. Ma, "Technology investment decision-making under uncertainty: the case of mobile payment systems", Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS), Hawaii, **(2013)** January 7-10.
[3] EMVco. "EMV Contactless Communication Protocol Specification". Version 2.5. March, 2015.https://www.emvco.com/specifications.aspx?id=21. Accessed 24 March 2016.
[4] A. Bodhani, "New ways to pay [communications near field]", Engineering & Technology, vol. 8, no. 7, **(2013)**, pp. 32-35.
[5] M. Alattar and M. Achemlal, "Host-Based Card Emulation: Development, Security, and Ecosystem Impact Analysis", Proceedings of High Performance Computing and Communications, Paris, France, **(2014)** August 20-22.
[6] A. Armando, A. Merlo and L. Verderame, "Trusted host-based card emulation", Proceedings of the International Conference on High Performance Computing & Simulation (HPCS), Amsterdam, Netherlands, **(2015)** July 20-24.
[7] P. Pourghomi and G. Ghinea, "Managing NFC payments applications through cloud computing", Proceedings of the 7th International Conference for Internet Technology and Secured Transactions, London, UK, **(2012)** December 10-12.
[8] P. Pourghomi, M. Q. Saeed and G. Ghinea, "A proposed NFC payment application" International Journal of Advanced Computer Science and Applications, vol. 4, no. 8, **(2013)**, pp. 173-181.
[9] P. Pourghomi, M. Q. Saeed and G. Ghinea, "A Secure Cloud-based NFC Mobile Payment Protocol", International Journal of Advanced Computer Science and Applications. vol. 5, no. 10, **(2014)**, pp. 24-31.
[10] M. Q. Saeed, C. Walter, P. Pourghomi and G. Ghinea, "Mobile Transactions over NFC and GSM" Proceedings of the 8th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM), Rome, Italy, **(2014)** August 24-28.
[11] "Visa payWave" [online]. Visa Company. http://www.visa.ca/en/personal/visa-paywave/index.jsp. Accessed 25 November 2015.
[12] R. Garg and S. Jain, "Requirements Analysis on Pay Wave", Proceedings of the Second International Conference on Advances in Computing and Communication Engineering (ICACCE), Dehradun, India, **(2015)** May 1–2.
[13] "The Visa payWave for Mobile Ecosystem" [online]. Visa Company. https://arch.developer.visa.com/paywavemobile. Accessed 24 March 2016.
[14] "Expresspay Merchants"[online]. American Express Company. https://web.archive.org/web/20110707132353/https://www295.americanexpress.com/cards/loyalty.do?page=expresspay.merchantslist. Accessed 24 March 2016.
[15] "Contactless Payments FAQs (Global)" [online]. American Express Company. https://network.americanexpress.com/en/globalnetwork/Images/Contactless%20FAQ.pdf. Accessed 24 March 2016.
[16] G. Goth, "Mobile security issues come to the forefront", IEEE Internet Computing, vol. 16, no. 3, **(2012)**, pp. 7-9.
[17] O. Ghag and S. Hegde, "A comprehensive study of Google wallet as an NFC application", International Journal of Computer Applications, vol. 58, no. 16, **(2012)**.
[18] "Google Wallet, Softcard partner to take on Apple Pay" [online]. CNET. http://www.cnet.com/news/google-wallet-softcard-partner-on-mobile-payments/. Accessed 30 November 2016.

[19] "Google Wallet" [online]. Google. https://www.google.com/wallet/faq/#google-wallet-android-pay. Accessed 30 November 2016.
[20] Bajkowski, Julian. AMEX corporate bills go to BPAY [online]. Government News, Vol. 35, No. 3, Jun 2015: 26. http://search.informit.com.au/documentSummary;dn=352372249879551;res=IELHSS> ISSN: 1447-0500. Accessed 24 March 2016.
[21] "MasterCard unveils MasterPass digital wallet and mobile payments platform" [online]. NFC World. http://www.nfcworld.com/2013/02/25/322610/mastercard-unveils-masterpass-digital-wallet-and-mobile-payments-platform/. Accessed 24 March 2016.
[22] "Cloud-Based Projects in Key Markets around the Globe Enable Consumers to Make Contactless Payments from their NFC-enabled Phones" [online]. MasterCard Company. http://newsroom.mastercard.com/press-releases/mastercard-drives-host-card-emulation-hce-momentum-with-mobile-payment-deployments-in-more-than-15-countries/. Accessed 24 March 2016.
[23] "Apple Pay" [online]. Apple Company. http://www.apple.com/apple-pay/. Accessed 25 November 2015.
[24] "Apple Pay security and privacy overview" [online]. Apple Company. https://support.apple.com/en-us/HT203027. Accessed 25 November 2015.
[25] P. Pourghomi, P. E. Abi-Char and G. Ghinea, "Towards a mobile payment market: A Comparative Analysis of Host Card Emulation and Secure Element", International Journal of Computer Science and Information Security, vol. 13, no. 12, **(2015)**, pp. 156-164.
[26] "Everything you need to know about NFC and mobile payments" [online]. CNET. http://www.cnet.com/how-to/how-nfc-works-and-mobile-payments/ . Accessed 25 November 2015.
[27] ETSI Specification of the Subscriber Identity Module (1996). Mobile Equipment (SIM - ME) interface (GSM 11.11), European Telecommunications Standards Institute (ETSI Std. Version 5.3.0) http://www.etsi.org/deliver/etsi_gts/11/1111/05.03.00_60/gsmts_1111v050300p.pdf Accessed 8 January 2016.
[28] E. Barkan, E. Biham and N. Keller, "Instant ciphertext-only cryptanalysis of GSM encrypted communication", Proceedings of Annual International Cryptology Conference Santa Barbara, California, USA, **(2003)** August 17-21.
[29] W. Chen, G. Hancke, K. Mayes, Y. Lien and J.H. Chiu, "NFC mobile transactions and authentication based on GSM network", Proceedings of the International Workshop on Near Field Communication, Monaco, France, **(2010)** April 20-22.
[30] D. E. Knuth, "Computer programming as an art" Proceedings of ACM Turing award lectures (p. 1974). 2007.

Author

Pardis Pourghomi is an Assistant Professor of Computer Science at the American University of the Middle East (AUM) in Kuwait. Prior to joining AUM, he worked as an Information Security Consultant in Coventry Building Society, UK. Before that, he worked as a part-time lecturer at Brunel University London. He received his BSc (Hons), MSc and PhD, all in Information Security, from University of East London (2009), Royal Holloway, University of London (2010) and Brunel University London (2014) respectively. His current research interests revolve around applied security and privacy architectures especially in the field of smart cards and communication architectures.

Sadi E. Seker grew up in Turkey. He is currently a faculty member of Smith College, department of computer science. He earned a bachelor's, a master's and a doctorate in computer science. He has lectured in six different countries and completed more than 20 research projects. Previously, Seker was a postdoctoral researcher at the Cyber Security Center at The University of Texas at Dallas, and he is actively researching in the areas of data science, machine learning and big data.

Gheorghita Ghinea is a Reader in the Computer Science Department at Brunel University, United Kingdom. He received the B.Sc. and B.Sc. (Hons) degrees in Computer Science and Mathematics, in 1993 and 1994, respectively, and the M.Sc. degree in Computer Science, in 1996, from the University of the Witwatersrand, Johannesburg, South Africa; he then received the Ph.D. degree in Computer Science from the University of Reading, United Kingdom, in 2000. His research activities lie at the confluence of Computer Science, Media and Psychology. In particular, his work focuses on the area of building secure end-to-end communication systems incorporating user perceptual requirements. He has over 250 publications in leading international conferences and journals, and consults regularly for both public and private organisations in his field of expertise.

Wassim Masri received his PhD in Computer Science from Paul Sabatier University (Toulouse, France) in 2009, M.Sc. in Computer Science from ENSEEIHT – INPT (France) in 2005, and B.Sc. in Computer Science from the Lebanese University in 2004. He has been an Assistant Professor at the Faculty of Engineering and Technology in the American University of the Middle East in Kuwait since 2012. He is the Department Chair of the Technology Department since 2013. His primary research interest is in Wireless Sensor Networks, Internet of Things, Cloud Systems, and Social Networks. He is an IEEE member, member of the Computing Society and Co-founder of the IEEE Kuwait Chapter, and an IET member in the Kuwait Network.