Advances in Engineering Research, volume 118
2nd International Conference on Automation, Mechanical Control and Computational Engineering (AMCCE 2017)

Analysis on Apple Pay transaction

Xiaonan Meng[1, a], Tom Chothia[2, b] and Ye Du[3, c]

[1]School of computer and information technology, Beijing Jiaotong University, Beijing 100044, China;

[2]School of computer science, University of Birmingham, Birmingham B152TT, UK;

[3]School of computer and information technology, Beijing Jiaotong University, Beijing 100044, China

[a]xnm0722@hotmail.com, [b] T.Chothia@cs.bham.ac.uk, [c]ydu@bjtu.edu.com

Keywords: EMV, Apple Pay, NFC, tokenization.

Abstract. This paper is to illustrate how Apple Pay works and what the protocol follows during the transaction. This process verified EMV Visa payWave protocol, and then, Apple Pay bounded with Visa card was lead to be discovered by NFC Reader and talk to Java Applet via NFC Reader. The results would be analyzed to see how Apple Pay protocol goes through and the difference between Apple Pay and EMV contactless cards.

Introduction

Apple Pay has been popular among the modern society as it is a convenient and fast payment method. However, customers are wondering about the safety of using these contactless payment methods in the shop without typing PIN number or Signature, we implemented the evaluation process of the both EMV and Apple Pay transaction protocols and analyzed the difference between those two contactless payment methods. To achieve this goal, an NFC Reader was first implemented on Android which was used to communicate to the Visa payWave card and Apple Pay, and show the response on the screen. The second step of the project was to implement a Java Applet tool on a laptop which can send messages to the NFC Reader and wait for the contactless devices response which will be parsed to Java Applet. Having done this, the application then decoded the response.

Published Research

EMV book 4.3-book 3 defined the application specification. It specified the transaction protocol of visa paywave. This section will illustrate how Visa's contactless protocol works. The protocol is as shown below [1].

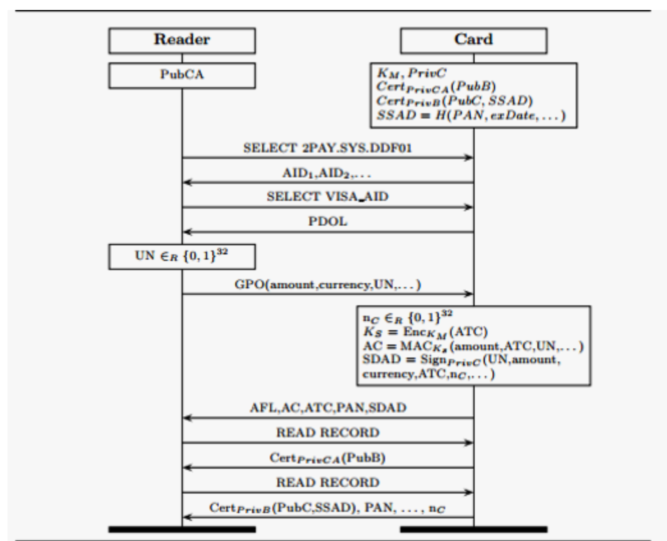

Figure 1. Visa's payWave protocol

Copyright © 2017, the Authors. Published by Atlantis Press.
This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

Design of the EMV-based software framework

This section demonstrates the design of developing an EMV-based software framework which includes a simple functional mobile phone application and a dedicated PC applet. PC applet is designed to be a message sender and decoder, which can send random messages to the mobile phone, and show well-decoded receiving messages on the applet screen. The NFC-enabled mobile phone is designed to be a simple shop reader which could talk to EMV bank card or Apple Pay.

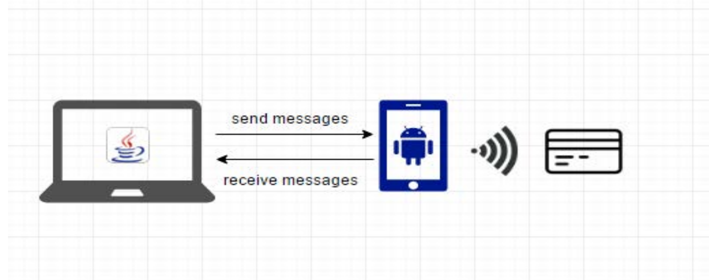

Figure 2. Design of EMV-based software framework

Implementation

A new java applet is implemented for the user to select and set transaction data which are parsed to the bank card, then the user can get the response command which then is decoded on the screen. This applet includes the following functionalities:

Selecting PSE: User can select PSE (Payment System environment) and choice will be uploaded by clicking the select button.

Selecting AID: User can select AID (Application Identifier) and the choice will be uploaded by clicking the select button.

Setting Transaction information: User set amount, currency, transaction data, country, and finally, choose one of the payment methods (Visa payWave or Apple Pay), then the whole information will be translated to a byte string and sent to the contactless payment system.

Reading record: User can type the specific commands to read card records or send random messages, then the card response will be shown on the screen.

Decoding byte string: The card response is plain byte string which can be decoded. The decoded message is shown as the raw string with the dedicated meaning on the screen.

Algorithm of decoding. We received hex byte string from the card response. We proposed an algorithm to decode the APDU string. The algorithm is shown as below.

First, we created a two-dimension tag dictionary array. Each entry has four elements: tag, tag description, tag type and tag attribute. Second, we wrote *containTag* function to check if we can find that response message contains the specific tag in the dictionary. The algorithm is shown as right. Then the function returns an array to record the results. Third, after calling the *containTag* function, which returned *tagData*. When we receive the card response, the string first is parsed to *DecodeApdu* function. Then the response is parsed to *containTag* to get a set of *tagData* values.

- first, we check if the tag is found, if true, we then check tag type if it is constructed, if true, we get the tag length and the rest of string, which will be used for iteration as *DecodeApdu(rest)*.
- second, if tag is not constructed, then we go to check if it is primitive. If it is true, we get the tag length and tag data elements. Then the rest of string will be iterated to *DecodeApdu* function.
- then, if tag type is not primitive, if it satisfies “DOL” type. If it does, we call *getPdol(rest)* function to handle decoding.
- if it doesn't match “DOL” but matches “FORMAT”. If true, we call *getFormat(rest)* to handle.
- finally, if the tag can't be found in dictionary, we output the unknown tag string and declare it is the unknown value.

Fourth, after card response string has been illustrated, we called function *shownOnScreen* to show the decoding of the response in a well-looking format.

```
Algorithm 1: Check if the string contains the defined tag
Result: Data[0], Data[1], Data[2], Data[3], Data[4],
        Data[5]
arr ← tagDictionary;
str ← response message;
tag ← tagDictionday[i][0],tagDictionary[i][1] ,
tagDictionary[i][2] tagDictionary[i][3];
while tag in tagDictionary do
    if tag[0] equals first two digits of str then
        Data[0] ← true;
        Data[1] ← tagDictionary[0];
        Data[2] ← tagDictionary[1];
        Data[3] ← tagDictionary[2];
        Data[4] ← tagDictionary[3];
        Data[5] ← str - tag;
    end
    if tag[0] equals first four digits of str) then
        Data[0] ← true;
        Data[1] ← tagDictionary[0];
        Data[2] ← tagDictionary[1];
        Data[3] ← tagDictionary[2];
        Data[4] ← tagDictionary[3];
        Data[5] ← rest;
    end
end
while tag not in tagDictionary do
    Data[0] ← false;
    Data[1] ← unknowntag;
    Data[2] ← "unknown tag";
    Data[3] ← unknown tag type;
    Data[4] ← unknowntag length;
    Data[5] ← rest;
end
```

Figure 3. Algorithm of decoding

Testing

We tested Apple Pay bound with Lloyds Visa Debit card and HSBC Visa credit card. we selected PSE (1PAY.SYS.DDF01) or PPSE (2PAY.SYS.DDF01) to challenge Apple Pay AID. Then we challenged the card with GET LOG FORMAT command, GET PIN TRY COUNTER command and GET ATC command. We tried to read the record with SFI and record number. After that, we set the amount, transaction date, currency, location country, payment method and submitted to the background to process. These Get Processing Options information is sent to Apple Pay and will be immediately responded to. We want to test the card response in following situations: 1) set amount as 50 pounds, 2) set invalid transaction date, 3) set expired transaction date, and 4) select pse1.

Conclusion

In "SELECT" part, after AID selected, Apple Pay needed Merchant Name and Location for transaction. Apple Pay also contains IPB which can filter the data in response to GPO. Apple Pay doesn't support 1PAY.SYS.DDF01. 2PAYP.SYS.DDF01 (PPSE) is only for contactless. Apple Pay has Log Entry but the number of the log record is 0, so we can't read transaction log by sending "READ RECORD" command.

In "GPO" part, if we set amount equals 50 pounds, the transaction goes and we can get a response from Apple Pay. Then, Apple Pay disconnects with the terminal immediately.

From the message in response to "GPO", AIP of Apple Pay is 20 40. This means Apple Pay supports DDA but no AFL value, so we can't read SSDA (Signed Static Data Authentication) from the response. The most important thing is "Track 2 Equivalent Data", this value shows a "PAN and Expired Date"-like string but not the real card number and expired date. This means Apple Pay takes a tokenization technology which can replace the real card number and expired date. This technology can provide security for card information. That can be alternative to "encryption" technology which provides data authentication to the card. Therefore, we can see Apple Pay doesn't contain SSDA data.

According to the response, we can see Apple Pay doesn't return CID value which means the card can't take action to generate cryptogram for the transaction. This means Apple Pay doesn't support the offline transaction and terminates the transaction immediately. Another difference between Apple Pay and EMV contactless bank card is Apple Pay's tag is dynamic. The tag is different for each time it is discovered. When tag discovered, the connection between the Apple Pay and terminal is kept less than one minute. This means the if the transaction is exceeded the card timeout, Apple Pay will terminate the transaction. According to the analysis of Apple Pay above, we can draw the protocol how Apple Pay work.

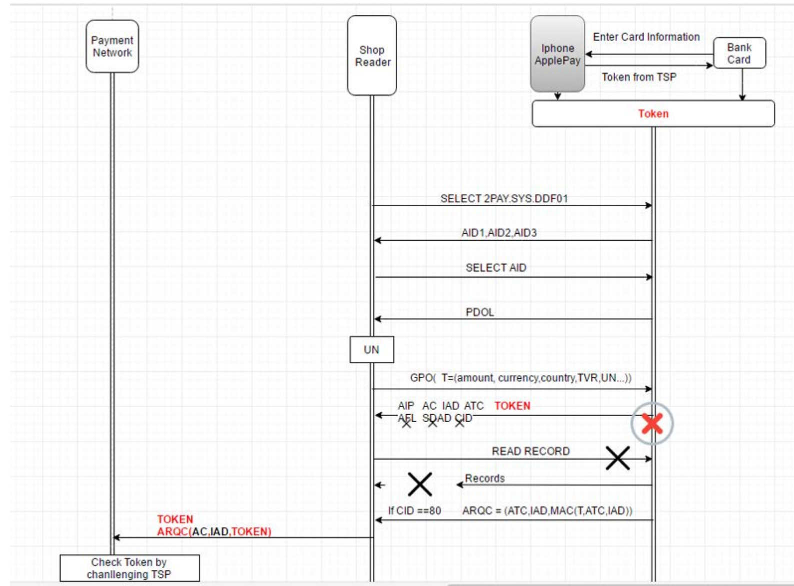

Figure 4. Apple Pay's Protocol

Appendices

Response APDU Format. The response is a byte hex-string which is encoding as the following format. It can be divided into two parts, one is the body part, the other is the trailer. The data in body field are objects structured as the BER-TLV format. As defined in ISO/IEC 8825. A BER-TLV data object consists of 2-3 consecutive fields [2].

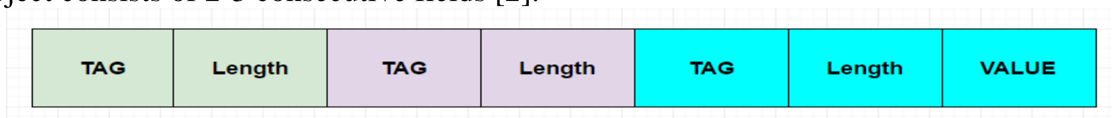

Figure 5. BER-TLV structure

If the tag is constructed, the value field consists of consecutive tag-length-value objects. As the figure shown above, the first and second tags are constructive. If the tag is primitive, the value field of the tag contains the data elements. For example, the third tag is primitive.

Acknowledgements

The author would thank to Tom Chothia and Ye Du to help and support author with this project. Author also would like to thank Kenny Cao who provided cards for author on testing purpose.

References

[1] Tom Chothia et al. Relay Cost Bounding for Contactless EMV Payments, 2015, p. 5.

[2] EMVCo. EMV Integrated Circuit Card Specifications for Payment Systems Book 3: Application Specification, 2011, p. 155.