Tracing Malicious Transactions in a Disconnected Micro-payment System

Trusit Shah
Computer Science Department
The University of Texas at Dallas
Richardson, USA
trusit.shah@utdallas.edu

Madison Pickering
Computer Science Department
The University of Texas at Dallas
Richardson, USA
madison.pickering@utdallas.edu

S. Venkatesan
Computer Science Department
The University of Texas at Dallas
Richardson, USA
venky@utdallas.edu

Max Irby
Computer Science Department
The University of Texas at Dallas
Richardson, USA
max.irby@utdallas.edu

Lan Vu
Computer Science Department
The University of Texas at Dallas
Richardson, USA
lan.vu1@utdallas.edu

Tung Vu
Computer Science Department
The University of Texas at Dallas
Richardson, USA
tung.vu@utdallas.edu

***Abstract*—It is critical that cashless payment systems feature an offline mode in remote areas due to unstable internet connections in those regions. Typically in offline mode, the vendor stores the transaction details on a smart card where these transactions are encrypted and, in some cases, signed by the vendor in order to prevent malicious activities. While these transactions are secure from outside attacks, they remain vulnerable to malicious acts on the part of the vendor. Many offline payment systems give vendors the power to change the user's card data, yet hinge on vendors not acting maliciously. This leaves the systems weak to malicious activity like double-spending (using the same transaction twice to charge double the correct amount), and the alteration of previous transactions. We have designed a partially offline cashless payment system to trace such malicious transactions. Our approach uses three principles to detect discrepancies in the transaction: non-repudiation, which helps to detect malicious vendors; token-based money spending, which prevents double and over-spending; and an out-of-order ledger system, which allows for the detection of altered transactions. We also provide an implementation and testing of our proposed payment system.**

***Index Terms*—micro-payment, offline payment**

I. Introduction

The common use of smart cards to make purchases in place of cash signifies a major paradigm shift in the way purchases are made. These systems are consequently referred to as cashless payment systems. A typical cashless payment system has four main components: a server, a vendor, a smart card and a buyer. The vendor processes the transaction and is responsible for purchasing product using the smart card. The buyer is the person, who is using the smart card to purchase the product and the owner of the card. The server performs a variety of important tasks, including communicating with the vendors, authorizing buyers, adding funds to the smart card and performing transactions [8]. As the server performs most of the basic functions, the current technologies for cashless payment systems require a continuous online connection to the server to perform each transaction.

Providing a stable internet connectivity in remote areas is a challenging task; many remote regions do not even have internet access. Even in urban areas, maintaining continuous internet connectivity throughout the whole day is a humongous task. As most cashless payment systems require a stable internet connection, the vendor is useless when it cannot connect to the server. Hence, buyers are forced to pay in cash or wait until the connectivity is back. Having a payment system that is resilient to intermittent connectivity - an offline payment system is highly desirable. [17].

In an offline payment system, the smart card stores the transactions. Most smart cards have a small amount of memory, and hence they store only the remaining balance in the card and the past few transactions. This system does not require internet connectivity to perform any operations and is therefore completely offline.

Offline payment systems use a shared key approach to encrypt the card data. Typically, a master key is shared among all vendors and the server. An example of such implementation is Octopus card, which uses a special type of smart card, a FeliCa card, to store data in an encrypted manner [3]. However, these systems do not provide non-repudiation due to using shared keys. If one of the vendors is compromised, it is impossible to detect which vendor has performed a transaction. A simple solution for this problem is to use a public key signature for each vendor [22]. Unfortunately, maintaining an up to date public key repository at the vendor is challenging as the vendor may not be connected to the server. If we assume an intermittent internet connectivity for vendors, the vendor can update its public key repository when connected to the server. While online, the vendor can additionally upload all transactions that took place. By uploading these transactions, the server can verify them, detecting any compromised transactions in the process. We use similar approach to achieve non-repudiation.

Other challenges with offline payment systems include

overspending and double spending. Overspending refers to a compromised or malicious vendor charging more money than the amount approved by the buyer [7]. Non-repudiation cannot prevent overspending, but it does help in detecting it: if the transactions are uploaded to the server and reported back to the buyer, the buyer can detect the discrepancy in the transaction. Just like overspending, preventing double spending is an equally challenging job. Double spending refers to the vendor charging multiple times for the same transaction. While it can't be prevented by non-repudiation, non-repudiation can help in detecting it. Hence, ensuring that an offline payment system supports non-repudiation is critical.

In this paper, we present a methodology for a partially online payment system which can perform transactions while offline. Once the vendor is connected to the server, it then uploads the transactions to the server which the server validates, detecting any compromised transactions in the process. Our contributions in this paper include the following:

- We introduce an offline cashless payment system that is capable of analyzing discrepancies in transactions, including detecting the specific vendor(s) that caused them.
- We present techniques to detect and prevent double-spending.
- We provided implementation of our approach using MiFare cards [1].

There are multiple ways to perform a cashless transaction: card based payment system, payment using mobile phones and online web browser based payment system. Our approach is designed for card based payment system but it can be modified to support other two approaches also.

II. Previous Works

Researchers have developed a variety of mechanisms for implementing cashless payment systems. Rivest and Shamir [19] were one of the first one to develop a cashless payment system. They provided two methods, "PayWord" and "MicroMint", designed to make small purchases over the internet. They consequently dubbed these systems micro-payment systems. PayWord uses a public key based certificate to authorize the user and the payment. It also supports fully offline payments, but requires an internet connection for vendors in order to send user transactions to a centralized server. MicroMint is a coin-based approach: each user is provided a set of "coins" to perform transactions. These coins are are computationally hard to create but easy to verify, ensuring a lightweight system. The systems described by Rivest and Shamir. [19] are especially elegant, and inspired the works of Khan and Ahmed [13] and Dai and Grundy [9].

The work of Hinterwälder et. al [12] focuses on identifying areas of collusion and methods of detecting fraud in fast paced and low resource systems. Their proposed system was built for high traffic but required a near constant internet connection. Similarly, Fan et. al [10] suggest the use of a custom security element (SE) in order to provide ultra-secure payment services. Their proposed system is incredibly robust to an array of security attacks, including man in the middle, replay attacks, cloning of information, and brute force. However, their work does identify fraudulent users or vendors. Other works on secure payment protocols include that of Luo and Yang [16] and Pourghomi and Ghinea [18].

Khaled Baqer's dissertation [4] provides a thorough analysis of most major online payment systems and their vulnerabilities. The work of Baqer et al. on DigiTally [5] provides more information on implementation level issues in offline payment systems. DigiTally is optimized for phone-based environments but can be restructured for a vending machine based environment with only a few minor changes.

Lin and Chaiang [15] describe a scheme for distributing coupons in a micro-payment based system rather than a direct focus on computational feasibility of processing micro-payments, as much of the research on cashless payment systems does. They also discuss the usability of coupons in micro-payment schemes. The works of Green and Miers [11] and Spirovska et al. [21] provide insight on the use of ledger systems to keep track of transactions in micro-payment schemes. Their work in particular emphasizes the need for ensuring deterministic execution despite the parallel nature of transactions.

Rodrigues et. al [20] describe another mobile payment scheme which features an extensive pilot program. Their work was notable for preventing over-spending, as well as for identifying and implementing their micro-payment scheme in a variety of markets.

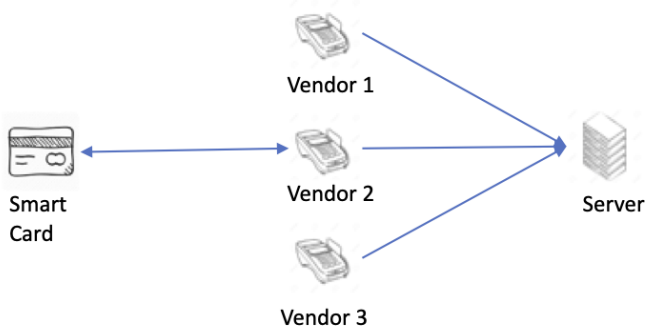

Fig. 1. System Architecture

III. System Architecture

Our system is predicated on three fundamental principles: non-repudiation, token-based spending, and out-of-order ledger verification. The first principle, non-repudiation, ensures that no vendor can deny after making the transaction. The token-based spending method prevents double-spending. Out-of-order ledger verification finds discrepancies in transactions and when used in conjunction with non-repudiation, the centralized server can detect compromised vendors.

Non-repudiation is achieved using public key cryptography. We use ECC based public key signature to provide non-repudiation. The centralized server assigns a public-private key

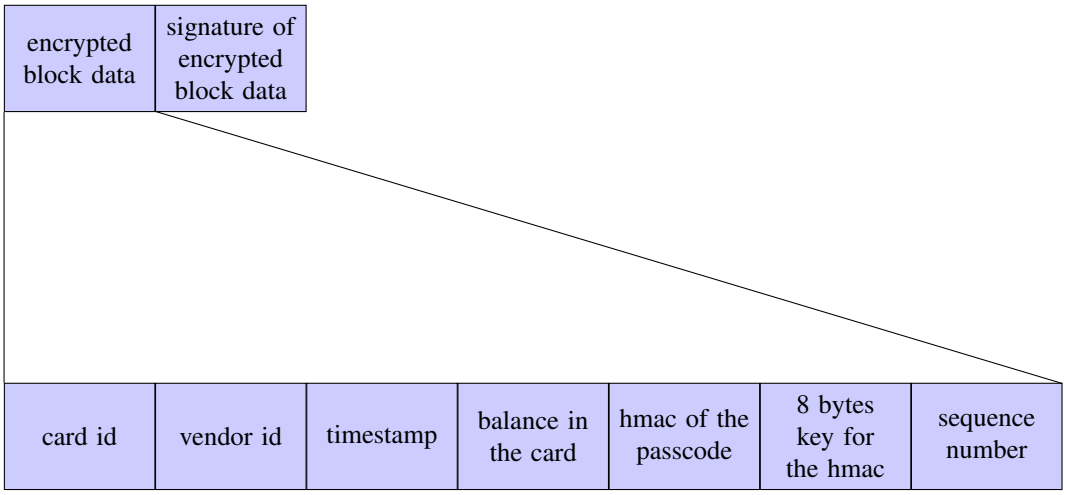

Fig. 2. Card block

pair to each vendor. Each vendor has the public keys of all vendors.

Each item that a vendor can sell has a token associated with it which can be used to deduct money from the card. This token is signed by the centralized server and all the vendors can verify it. Thus, a vendor can not deduct arbitrary amounts of money from the card. This method therefore prevents overspending and double spending problems.

All vendors connect to the server at different times and at different rates. Thus, it is not necessary that all transactions uploaded to the server may not be in the order that purchases were made. These out-of-order transactions are stored in the server the order transactions happened. We have created a data-structure called 'out-of-order ledger', which stores transactions in the order transaction happened. For brevity, we use the term 'ledger' for 'out-of-order ledger'. The ledger is maintained at the server and it verifies all transactions.

IV. Our Approach

Our approach to an offline payment system involves storing the balance in the card for offline mode and validating the transactions on the server for security. Vendors may not be online all the time, but when they are they upload their transactions to the server for validation. The server then validates the transactions and, if it finds any discrepancies, alerts the admin. Fig. 1 represents the architecture of our system.

A. Card Block

Each smart card stores the balance and vendor's signature in the data block. The data block consists of two parts: encrypted data and the signature of the encrypted data. The encrypted data is encrypted using a master AES key, which is shared among all the vendors and the server. The signature is created using a private key assigned to the vendor. Every vendor knows the public key of all the other vendors; hence, every vendor can verify the signature signed by any of the other vendors. The card maintains a sequence number to provide a chronological order to the ledger system. After each transaction, the vendor increments sequence number by one.

Algorithm 1 Algorithm to create a new transaction at vendor

```
1: procedure CreateTransaction(type, amount, passcode)
2:     prevTransaction ← card.ReadTransaction()
3:     if not Verify(prevTransaction, passcode) then
4:         return
5:     end if
6:     hashkey[1...8] ← GenerateRandom(8)
7:     passcode ← HMAC(password,hashkey)
8:     sequence ← prevTransaction.sequence + 1
9:     prevAmount ← prevTransaction.amount
10:    if type is AddMoney then
11:        newAmount ← prevAmount + amount
12:    else if type is SpendMoney then
13:        newAmount ← prevAmount − amount
14:    end if
15:    GenerateNewTransaction(
                    prevTransaction,
                    newAmount,
                    passcode,
                    sequence)
16: end procedure
```

Each card has a passcode to authorize the buyer. When a card is added to system, a random passcode is assigned to it and in future buyer can change it. The card does not store the passcode in raw format, instead stores HMAC (keyed hash) [14] of the passcode. The key for the HMAC is a eight bytes long random number.

The encrypted data is the encryption of the following properties:

1) Card ID
2) Vendor ID

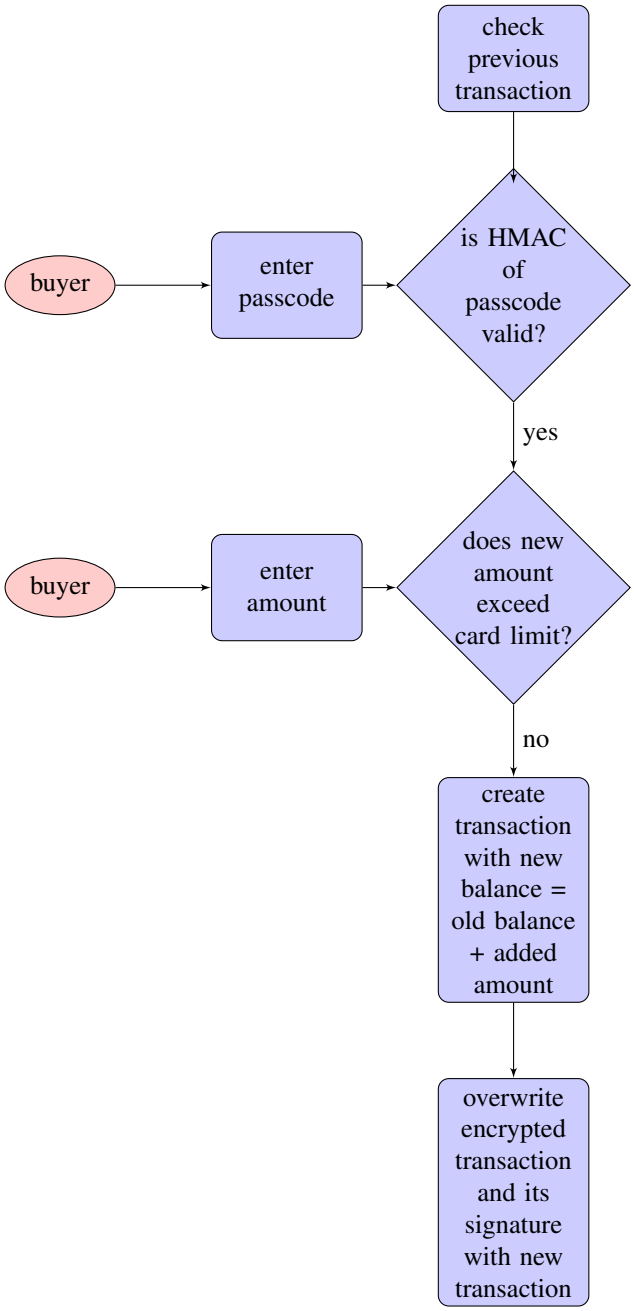

Fig. 3. Flow diagram for adding money

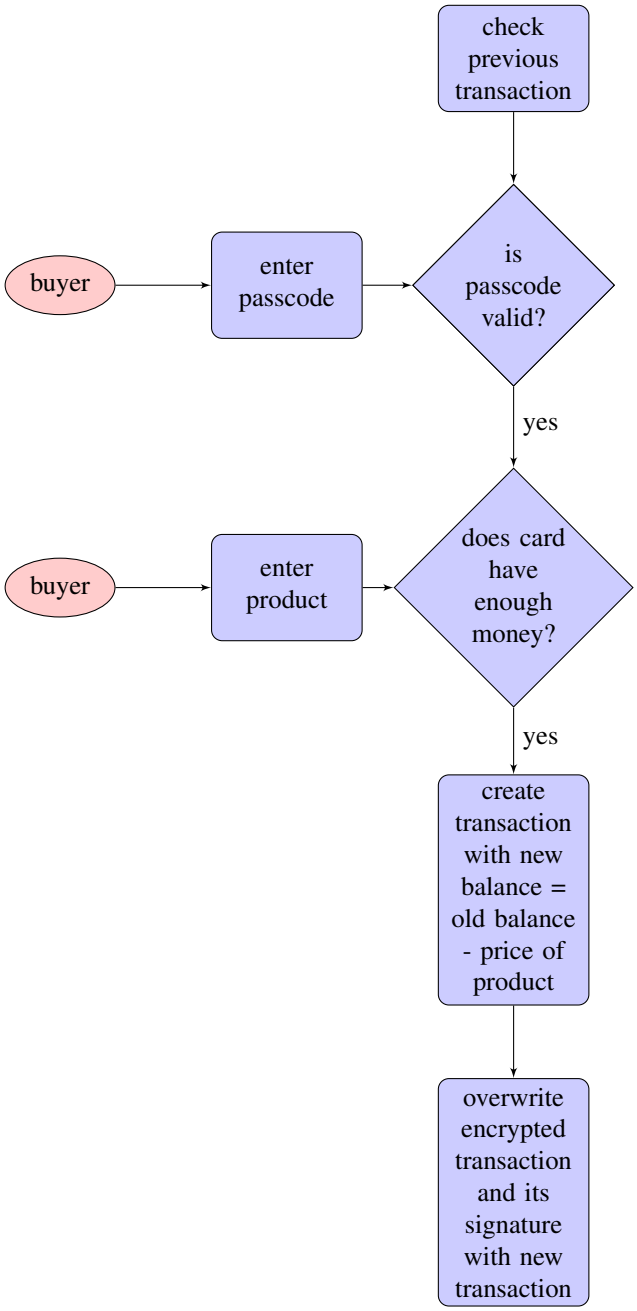

Fig. 4. Flow diagram for spending money

3) Timestamp
4) Balance of the card (Remaining amount of money in the card after the transaction)
5) HMAC of the Passcode
6) 8 bytes random key to generate HMAC
7) Sequence number

Fig. 2 shows a block diagram of the given properties.

B. Transaction Flow

When a smart card is added to the system, an initial transaction with a random passcode and initial balance greater than or equal to zero is added into the card. This transaction is then signed with the server's private key.

C. Adding money to the card

Some vendors allow users to add money to the card. If a user wants to add money using a credit or debit card, internet connectivity is required to access the card credentials. Fig. 3 shows the flow to add money into the card.

The vendor follows these steps to add money to the card:

1) It checks the previous transaction and validates the signature. If the transaction is missing or the signature is invalid, the vendor raises an error.

2) It asks user to enter the passcode and validates its HMAC with the previous transaction's HMAC.
3) If the passcode is correct, the vendor receives money from the user either through a credit/debit card or cash.
4) It creates a new transaction with new balance where the new balance = old balance + money deposited. The vendor then encrypts the new transaction with the master AES key and signs it with its own private key. The new transaction block is written on the card, overwriting the previous transaction. Algorithm 1 describes the method to create a transaction.
5) The new transaction is also logged into the local memory. When internet connectivity is available, it is uploaded to the centralized server. We will discuss more about this step in the following sections.
6) It provides the receipt of the transaction to the user.

D. Spending money from the card

The vendor follows the following steps to deduct money from the card:

1) It checks the previous transaction and validates the signature. If the transaction is missing or the signature is invalid, the vendor raises an error.
2) It asks the user to enter his or her passcode and validates its HMAC with the previous transaction's HMAC.
3) If the passcode is correct, the vendor checks if the card has sufficient funds to purchase the product.
4) It creates a new transaction with new balance where the new balance = old balance – the price of the product. The vendor then encrypts the new transaction with the master AES key and signs it with its own private key. The new transaction block is written on the card, overwriting the previous transaction.
5) The new transaction is also logged into the local memory. When internet connectivity is available, it is uploaded to the centralized server. We will discuss more about this step in the following sections.
6) It provides the receipt of the transaction to the user.

Fig. 4 shows the flow to deduct money from the card.

E. Validation of the transaction

Each vendor uploads the transaction log of each transaction to the server. This process is required for the following reasons:

- To have a permanent record in the system of each transaction
- To detect any discrepancies in the transactions

Since these vendors may connect to the server at different times, the transactions uploaded to the server may not be consecutive in nature. The ledger system, maintained at the server, processes these out of order transactions and store them in the correct order.

For each transaction, the vendor uploads the following details to the server:

- Card ID
- Timestamp of the transaction
- Vendor ID
- Remaining amount after the transaction
- Timestamp of the previous transaction
- Vendor ID of the previous transaction
- Remaining amount after the previous transaction
- Current sequence number

We denote each transaction as Tr and previous transaction as Tr_{prev}. The card ID, timestamp, vendor ID and remaining amount after the transaction are denoted as $Tr(card\_id)$, $Tr(t)$, $Tr(v\_id)$ and $Tr(amount)$ respectively.

At the centralized server, all the transactions are separated using the card ID and then stored in a ledger. Each ledger has transactions sorted with respect to the sequence number.

Definition 1: Adjacent transaction: A transaction Tr is adjacent to another transaction Tr‘ if both of them appear adjacent to each other on a ledger. If Tr appears before Tr‘, then Tr is left adjacent to Tr‘. If Tr appears after Tr‘, then Tr is right adjacent to Tr‘.

Definition 2: Consecutive transactions: Two transactions Tr_1 and Tr_2 are called consecutive if,

1) Tr_1 is left adjacent to Tr_2.
2) They satisfy the following conditions,
   - $Tr_1(t) = Tr_2\_prev(t)$
   - $Tr_1(v\_id) = Tr_2\_prev(v\_id)$
   - $Tr_1(amount) = Tr_2\_prev(amount)$

Definition 3: Complete transaction: A transaction is called complete if all the transaction happened before this transaction is uploaded to the ledger.

The first transaction of any card is always a complete transaction and after that the transaction is complete if,

- The transaction is consecutive with its left adjacent.
- The left adjacent transaction is complete.

TABLE I
TRANSACTIONS OF CARD c_1 AT DIFFERENT VENDORS. THE BALANCE IN THE CARD BEFORE THE FIRST TRANSACTION IS \$100.

vendor	timestamp	amount
v_1	8^{th} September 12:03 PM	\$5
v_2	8^{th} September 2:03 PM	\$3
v_1	8^{th} September 4:07 PM	\$2
v_3	9^{th} September 8:23 AM	\$1
v_4	9^{th} September 10:41 AM	\$4

Steps to add a new transaction to the ledger:

1) Vendor uploads the transaction to the server.
2) Server finds the place in the ledger where the new transaction can be added. Note that all the transactions are sorted with respect to sequence number.
3) If the left adjacent transaction of the newly added transaction is complete and consecutive with the newly added transaction, mark the newly added transaction complete.
4) If the newly added transaction is marked complete and if the right adjacent transaction to the newly added transaction is consecutive to newly added transaction,

TABLE II
LEDGER AFTER v_1 UPLOADS THE TRANSACTIONS

vendor	balance	timestamp	prev vendor	prev balance	prev timestamp	sequence number	complete
v_1	\$95	8^{th} September 12:03 PM	-	100	-	1	yes
v_1	\$90	8^{th} September 4:07 PM	v_2	\$92	8^{th} September 2:03 PM	3	no

TABLE III
LEDGER AFTER v_2 UPLOADS THE TRANSACTIONS

vendor	balance	timestamp	prev vendor	prev balance	prev timestamp	sequence number	complete
v_1	\$95	8^{th} September 12:03 PM	-	100	-	1	yes
v_2	\$92	8^{th} September 2:03 PM	v_1	\$95	8^{th} September 12:03 PM	2	yes
v_1	\$90	8^{th} September 4:07 PM	v_2	\$92	8^{th} September 2:03 PM	3	yes

TABLE IV
LEDGER AFTER v_3 AND v_4 UPLOADS THE TRANSACTIONS AND v_4 HAS ALTERED THE BALANCE FROM v_3'S TRANSACTION

vendor	balance	timestamp	prev vendor	prev balance	prev timestamp	sequence number	complete
v_1	\$95	8^{th} September 12:03 PM	-	\$100	-	1	yes
v_2	\$92	8^{th} September 2:03 PM	v_1	\$95	8^{th} September 12:03 PM	2	yes
v_1	\$90	8^{th} September 4:07 PM	v_2	\$92	8^{th} September 2:03 PM	3	yes
v_3	\$89	9^{th} September 8:23 AM	v_1	\$90	8^{th} September 4:07 PM	4	yes
v_4	\$81	9^{th} September 10:41 PM	v_3	\$85	9^{th} September 8:23 AM	5	no

mark the right adjacent transaction as complete. Repeat this procedure for the transactions on the right side of the newly added transaction.

Table I describes a set of transactions of a card c_1. The balance inside the card before the first transaction is \$100. The amount column in this table represents the money spent at the vendor. All these transactions happened while the vendors are not connected to the server, hence, the vendors store the transactions in their local memory and upload them to the server once they receive internet connectivity. The first vendor to get internet connectivity is v_1. The transactions uploaded to the server by v_1 are mentioned in Table II. From Table II, it is clear that there is one transaction performed by v_2 between the two transactions performed by v_1. Therefore, the second transaction is incomplete. Once v_2 gets internet connectivity, it uploads the transaction to the server and Table III shows the ledger after that. At this point all the transactions are consecutive to each other and are therefore complete.

Let's consider a situation in which one of the vendors is malicious and tries to change the previous transaction's data. This alteration has two key transactions: the transaction being altered and the transaction that altered the value. Once both transactions are uploaded to the server, the server can detect the anomaly in the transactions and report the malicious vendor to the admin. From Table I, assume that v_4 is malicious, and it uploaded the transaction before v_3 and altered the balance. The actual balance after the fourth transaction is \$89 and v_4 reports the server \$85 instead. In Table IV, the 4^{th} and 5^{th} transactions have a mismatch in balance and previous balance. Thus, the server can trace anomalies in transactions.

F. Token-based transactions

In the above approach, the vendor can deduct money multiple times from the card. As the transactions are offline, the buyer doesn't receive notification from the server regarding the purchase immediately. In the future, when the transaction is posted to the server and the buyer does receive a notification, the buyer may have forgotten the details of the purchase. Consequently, the buyer may not detect the extra money charged by a malicious vendor. We have designed a token-based approach that restricts the vendors from deducting amounts multiple times from the card to solve this problem.

Every product sold at the vendor has a token associated with it. This token is a signature of the price of the product. To add randomness to the token, the signature also contains timestamp of product's manufacturing date and a random nonce. This way signature of each product is different. The vendor uploads this token with the transaction to the server where it is signed using the server's private key. Each item has a separate token associated with it and can be used only once. If the vendor tries to re-use the token, the server can detect it. Hence, double spending is prevented through detection.

V. IMPLEMENTATION

We have implemented our work using a MiFare card [1] as a smart card and a Raspberry Pi as a vendor. We have used Node.js [2] as our server-side back end in which the Raspberry Pi communicates with the server using web APIs.

A. MiFare card

A MiFare card is a smart card that has 1kB of memory. It also comes with a unique id to uniquely identify the card. MiFare card has an inbuilt encryption system. Each memory block (eight bytes) is encrypted using shared key encryption. The whole transaction block (data block and its signature) is additionally encrypted using this mechanism to provide an extra layer of security.

TABLE V
COMPARISON

	[12]	[19]	[5]	[10]	[16]	[18]	[11]	our work
offline	✓	✓	✓	✓				✓
double spending	✓	✓	✓					✓
man in the middle	✓			✓		✓	✓	✓
required resources	low	low	low	low	high	high	low	medium
replay attack	✓			✓	✓	✓	✓	✓
card cloning		✓	✓	✓				✓
non-repudiation					✓	✓	✓	✓
card based	✓			✓	✓			✓

We have used AES-128 to encrypt the transaction data. The data block has an eight byte long card id, four byte long vendor id, two byte long sequence number, eight byte long timestamp, two byte long balance, 32 bytes long passcode, and eight bytes long random key for hmac [14]. The total size of the data block is 64 bytes and, after encrypting 64 bytes, AES-128 will generate 64 bytes. We have used ECC [6] for signing the encrypted data to ensure non-repudiation. The size of signature is 72 bytes. Thus, the total size of a transaction is 136 bytes.

B. Vendor - Raspberry Pi

The Raspberry Pi runs three concurrent threads: the first one to communicate with the MiFare card, the second one to upload transactions to the server, and the third one to read the buyer's input.

The first thread continuously listens for a Mifare card connection. Once it detects a Mifare card, it initiates communication with it. Fig. 3 and 4 describe the operation flow of the communication. After each transaction this thread sends the transaction to the second thread.

The second thread stores all the transactions provided by the first thread into a queue and continuously checks for internet connectivity. Once internet is available, it uploads all transactions in the queue to the server.

C. Server

The server is divided into two parts: a web API and database. The web API acts as a bridge between the Raspberry Pi and the database. The database stores ledgers for the cards. A separate thread is running on the Node.js server to detect discrepancies in the transactions. We host our server on a macbook pro with intel i7 processor and 16 GB RAM.

D.

VI. COMPARISON

Table V gives a brief comparison of our work with other related works. We consider the following parameters as comparison parameters: offline mode, double spending, man in the middle, required resources, replay attack, card cloning and modification, and non-repudiation.

Offline mode refers to systems that do not require a continuous internet connection to function. Many systems, such as [12] and [10] can perform transactions in offline mode, but at some point the proposed systems will require an internet connection to upload transactions to the server. For the purposes of the above table, systems are considered to be offline systems as long as they do not require a constant internet connection.

Man in the middle is a broad class of attacks which refers to attacks in which an adversary acts as a middleman between the server and the client. The attacker typically acts as the server to the client and as the client to the server. One of the most common ways of preventing this attack is through the use of authentication and changing session keys on a periodic basis.

One major concerns of micro-payment schemes is the resource allocation for each transaction. That is, since the price of the items being purchased are low, the transaction must be suitably efficient such that the cost of processing the transaction is very low. For each transaction, a computational resource is required to process, store and validate the transaction. The payment systems that use RSA, like public key encryption, require a large amount of resources to store and process a transaction. Shared key encryption is lightweight in both processing and storing a transaction. Schemes like ECC fall in between, providing lightweight storage but requiring more resources than shared key encryption and less resources than RSA for processing and validation. Table V shows the resource requirements in three categories: high, medium and low, where high is comparable to RSA, medium is comparable to ECC, and low is comparable to shared key encryption.

Card cloning refers to the duplication (cloning) of data that should be unique to the card that made the purchase, such as the card's balance. As many of the above schemes are not card-based, in the table V, they are considered to prevent card cloning as long as they provide a method for preventing the duplication of unique information. "DigiTally: Piloting Offline Payments for Phones" [5] and "PayWord and MicroMint: Two simple micropayment schemes" [19] are two examples of such systems.

Malicious transaction detection refers to the ability of a system to trace discrepancies in transaction information. This trait allows the server to detect vulnerable or malicious vendors. Non-repudiation is required to detect such malicious transactions. Table V provides a list of systems that support non-repudiation and detect discrepancies in the transactions. Note that the majority of protocols that support non-repudiation are online-only protocols. This is obviously undesirable as non-repudiation is critical to detecting overspending and double spending, two common areas of concern. Our proposed

method however is one of few that is both offline and supports non-repudiation.

A card based system refers to systems where the system uses a card to make transactions. Many NFC based systems require the SE (Secure Element) in a phone to work, and are not considered to be card based as they consequently require a phone. Digitally [5] is an example of a phone-based system. Other systems, such as "NetPay: An Off-Line, Decentralized Micro-Payment System for Thin-Client Applications" [9] require the use of a web browser to function.

VII. Conclusion

In this paper, we discuss methods to trace anomalies in offline payment systems. We have developed an approach to prevent three major security threats in offline payment systems: double-spending, overspending and alteration of previous transactions. Our work is scalable, easy to integrate, and lightweight in order to support small-scale devices. We have created a demo of our system at our lab to verify all the above characteristics.

Recovering the system to a safe state is critical after detecting an anomaly. The future work for this paper will be to design approaches for card and vendor recovery. Card recovery refers to recovering the correct balance of the card while vendor recovery resets the vendor's program and security parameter.

References

[1] Mifare. https://www.mifare.net/en/. (Accessed on 01/10/2020).

[2] Node.js. https://nodejs.org/en/. (Accessed on 01/10/2020).

[3] Sony global - felica web site. https://www.sony.net/Products/felica/. (Accessed on 01/07/2020).

[4] Khaled Baqer. *Resilient payment systems*. PhD thesis, University of Cambridge, 2018.

[5] Khaled Baqer, Ross Anderson, Lorna Mutegi, Jeunese Adrienne Payne, and Joseph Sevilla. Digitally: piloting offline payments for phones. In *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)*, pages 131–143, 2017.

[6] Simon Blake-Wilson, Bodo Moeller, Vipul Gupta, Chris Hawk, and Nelson Bolyard. Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls). 2006.

[7] Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible e-cash made practical. *IET Information Security*, 10(6):332–347, 2016.

[8] David Chen, Zhiyue Zhang, Ambrish Krishnan, and Bhaskar Krishnamachari. Payflow: Micropayments for bandwidth reservations in software defined networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 26–31. IEEE, 2019.

[9] Xiaoling Dai and John Grundy. Netpay: An off-line, decentralized micro-payment system for thin-client applications. *Electronic Commerce Research and Applications*, 6(1):91–101, 2007.

[10] Kai Fan, Panfei Song, Zhao Du, Haojin Zhu, Hui Li, Yintang Yang, Xinghua Li, and Chao Yang. Nfc secure payment and verification scheme for mobile payment. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 116–125. Springer, 2016.

[11] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 473–489. ACM, 2017.

[12] Gesine Hinterwälder, Christian T Zenger, Foteini Baldimtsi, Anna Lysyanskaya, Christof Paar, and Wayne P Burleson. Efficient e-cash in practice: Nfc-based payments for public transportation systems. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 40–59. Springer, 2013.

[13] Bazeem Ismaeil Khan and Shaikh Zubair Ahmed. Secure device for offline micro payment.

[14] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. Hmac: Keyed-hashing for message authentication. 1997.

[15] Iuon-Chang Lin and Hsiao-Chi Chiang. A novel sharing m-coupons with lightweight computations via cloud computing. In *Workshops of the International Conference on Advanced Information Networking and Applications*, pages 459–470. Springer, 2019.

[16] Jia Ning Luo and Ming Hour Yang. An anonymous nfc-based payment protocol. In *Applied Mechanics and Materials*, volume 764, pages 812–816. Trans Tech Publ, 2015.

[17] Jonas Muleravicius, Inga Timofejeva, Aleksejus Mihalkovich, and Eligijus Sakalauskas. Security, trustworthiness and effectivity analysis of an offline e-cash system with observers. *Informatica*, 30(2):327–348, 2019.

[18] Pardis Pourghomi, Gheorghita Ghinea, et al. A proposed nfc payment application. *arXiv preprint arXiv:1312.2828*, 2013.

[19] Ronald L Rivest and Adi Shamir. Payword and micromint: Two simple micropayment schemes. In *International workshop on security protocols*, pages 69–87. Springer, 1996.

[20] Helena Rodrigues, Rui José, André Coelho, Ana Melro, Marta Ferreira, João Cunha, Miguel Monteiro, Carlos Ribeiro, et al. Mobipag: Integrated mobile payment, ticketing and couponing solution based on nfc. *Sensors*, 14(8):13389–13415, 2014.

[21] Kristina Spirovska, Diego Didona, and Willy Zwaenepoel. Paris: Causally consistent transactions with non-blocking reads and partial replication. *arXiv preprint arXiv:1902.09327*, 2019.

[22] Jen-Ho Yang, Ya-Fen Chang, and Yi-Hui Chen. An efficient authenticated encryption scheme based on ecc and its application for electronic payment. *Information Technology and Control*, 42(4):315–324, 2013.