Large-Scale Invisible Attack on AFC Systems with NFC-Equipped Smartphones

Fan Dang[1], Pengfei Zhou[1,2], Zhenhua Li[1*], Ennan Zhai[3], Aziz Mohaisen[4], Qingfu Wen[1], Mo Li[5]
[1] School of Software, TNLIST, and KLISS MoE, Tsinghua University, China
[2] Beijing Feifanshi Technology Co., Ltd., China
[3] Department of Computer Science, Yale University, USA
[4] Department of Computer Science and Engineering, State University of New York at Buffalo, USA
[5] School of Computer Science and Engineering, Nanyang Technological University, Singapore
{*dangf13, wenqf15*}*@mails.tsinghua.edu.cn*, {*zhoupf05, lizhenhua1983*}*@tsinghua.edu.cn*,
ennan.zhai@yale.edu, mohaisen@buffalo.edu, limo@ntu.edu.sg

***Abstract*—Automated Fare Collection (AFC) systems have been globally deployed for decades, particularly in public transportation. Although the transaction messages of AFC systems are mostly transferred in plaintext, which is obviously insecure, system operators do not need to pay much attention to this issue, since the AFC network is well isolated from public network (*e.g.*, the Internet). Nevertheless, in recent years, the advent of Near Field Communication (NFC)-equipped smartphones has bridged the gap between the AFC network and the Internet through Host-based Card Emulation (HCE). Motivated by this fact, we design and practice a novel paradigm of attack on modern distance-based pricing AFC systems, enabling users to pay much less than actually required. Our constructed attack has two important properties: 1) it is invisible to AFC system operators because the attack never causes any inconsistency in the backend database of the operators; and 2) it can be scalable to large number of users (*e.g.*, 10,000) by maintaining a moderate-sized AFC card pool (*e.g.*, containing 150 cards). Based upon this constructed attack, we developed an HCE app, named LessPay. Our real-world experiments on LessPay demonstrate not only the feasibility of our attack (with 97.6% success rate), but also its low-overhead in terms of bandwidth and computation.**

I. Introduction

Automated Fare Collection (AFC) systems have been globally deployed for decades to automate manual ticketing and charging systems, particularly in public transportation networks. As transit routes in modern cities are usually quite long, most of today's AFC systems adopt a distance-based pricing strategy, where the transit fee is calculated based on the length of the trip. To date, billions of AFC cards have been issued across the world.

A typical AFC system leverages a symmetric encryption method (*e.g.,* based on 3DES [1] or AES algorithm [2]) to authenticate both the entities and messages involved. When an AFC card is officially issued, an unchanged unique transaction key, *TK*, is written into the card, which will be used to generate a dynamic session key, *SK*, and a message authentication code (or *MAC*) [3] during the debit phase. Surprisingly, all the other data (*i.e.,* the data other than *TK*, *SK*, and *MAC*) exchanged between AFC cards and terminals (*i.e.,* faregates or fareboxes) are in the plaintext format, which is insecure [4]–[8]. The AFC system operators, nevertheless, need not to worry about such risky situation, because the AFC network is well isolated from public networks (*e.g.,* the Internet). Thus, it is quite difficult, if any attacker wants to hack into the infrastructure of AFC systems in practice.

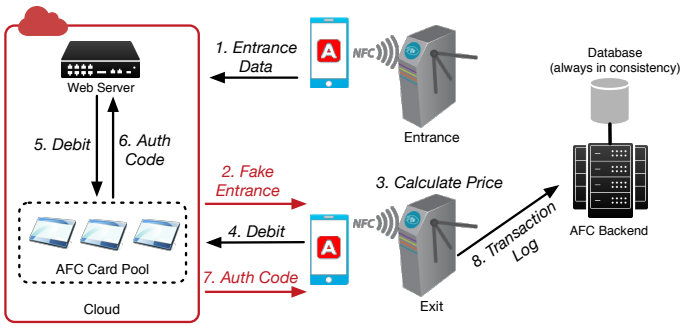


Fig. 1: Architectural overview of our designed attack on an AFC system. Red arrows denote the tampered messages, which however never cause inconsistency in the database of the AFC system.

Unfortunately, in recent years the advent of Near Field Communication (NFC)-equipped smartphones has bridged the gap between the AFC network and the Internet, thus putting AFC systems in a highly dangerous situation. Nowadays, the NFC module has become a typical component of mainstream smartphones such as iPhone 6 and 6s. It operates at the same frequency (13.56 MHz) and implements the same communication standard (ISO/IEC 7816 and ISO/IEC 14443) as those in AFC systems [9]. Moreover, it can work in a special *Host-based Card Emulation (HCE) mode* that allows any Android application to emulate an AFC card and talk directly to an AFC terminal.

Motivated by the above situation, we design, implement and test a novel paradigm of attack on modern distance-based pricing AFC systems. The goal of this study is to investigate the possibility of paying much less than actually required. As the basic requirements of launching such attacks, the users only need to have NFC-equipped smartphones and have installed

* Corresponding author.

LessPay – our developed HCE app based on our constructed attack – on their smartphones. Fig. 1 presents a step-by-step workflow of our constructed attack. In the attack, there are two important phases: *tampering entrance data* (Step 1-2 in Fig. 1) and *relay attack on AFC card* (Step 4-7 in Fig. 1), which we outline in the following.

- *Phase 1: Tampering entrance data.* As shown in Fig. 1, when a LessPay user wants to have a trip by metro, she first taps her smartphone on an entrance terminal. Then, the entrance terminal writes the entrance data into the AFC card emulated by LessPay, indicating the user's entrance station and timestamp. Subsequently, the entrance data is reported to the cloud of LessPay via a cellular connection (Step 1 in Fig. 1). After receiving the entrance data, the cloud periodically sends fake entrance data to the user (Step 2), in order to minimize the *expected* fare paid by her (note that the cloud does not know the user's destination). In practice, the period is typically configured as two minutes and the cellular traffic cost is within tens of KBs.
- *Phase 2: Relay attack on AFC card.* When the user reaches her destination, she taps her smartphone on an exit terminal, and the exit terminal calculates how much the user should pay for the trip according to the fake entrance data (Step 3). Afterwards, the exit terminal sends a debit message to the emulated AFC card, which is instantly forwarded to the cloud by LessPay (Step 4). On the cloud side, this debit message is first relayed to the physical AFC card corresponding to the emulated AFC card (Step 5), and then the message authentication code (*MAC*) is relayed to the web server (Step 6). Finally, the web server returns the debit message together with *MAC* to LessPay (Step 7) and a transaction log is reported to the AFC backend by the exit terminal (Step 8). So far, we finish a typical workflow of our attack. According to our measurement results, the round trip time from Step 4 to Step 7 is generally within 100 ms, which is totally acceptable to user's real-world experience.

At the heart of our attack architecture is an AFC card pool that maintains a number of physical AFC cards for conducting relay attacks (*i.e.,* Step 5 and Step 6 in Fig. 1). The success of relay attacks guarantees two important properties. First, AFC backend cannot detect any data inconsistent during the process of our attack, which means our attack is invisible to AFC system operators. In other words, for an AFC system operator, the debit & *MAC* provided by LessPay is *indistinguishable* from the ones offered by a normal AFC card. Second, as the web server (at the cloud side in Fig. 1) tampers both the station and timestamp information in the entrance data to forge a very short trip, we only need to maintain a relatively small number of cards in the pool to serve for a large number of users, *e.g.,* 150 cards serving for 10,000 users. This is because our users' very short fake trips can be easily scheduled by the cloud to totally avoid conflicts.

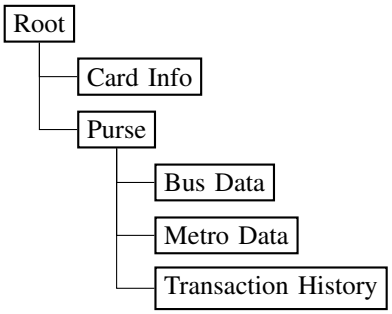


Fig. 2: Example: File structure of CTC.

As a representative case study, we conducted real-world attacks to the City Traffic Card (CTC) system in City X, one of the major cities in China, with tens of millions population. Specifically, 100 users were recruited and each user randomly used LessPay to take a subway 40 times a month. During three-month experiments (from Jan. 10th to Apr. 10th, 2016) with a total of 12,000 tests, 97.6% tests passed (the failed tests are owing to the poor quality of cellular connections). After the experiments, all AFC cards in our card pool still work well. This demonstrates the feasibility and scalability of our designed attack. We have reported the attack to several popular AFC systems including CTC. Nevertheless, there does not seem to be a good solution to prevent the attack in current AFC systems.

In summary, this paper makes the following contributions:

- We construct a large-scale invisible attack on AFC systems with NFC-equipped smartphones, thus enabling users to pay much less than actually required.
- We develop an HCE app, named LessPay, based on our constructed attack (detailed in Section III).
- We evaluate LessPay with real-world large-scale experiments, which not only demonstrate the feasibility of our attack (with 97.6% success rate), but also shows its low-overhead in terms of bandwidth and computation (detailed in Section IV).

The rest of this paper is organized as follows. Section II describes the overview of a typical AFC transaction. Section III presents how we construct the attack and how we implement an HCE app, named LessPay, to enable the attack in practice. Section IV evaluates LessPay through both real-world case study and overhead measurement. Section V reviews the related works, and Section VI draws some conclusions.

II. Overview of an AFC Transaction

Before we describe our constructed attack and the developed app, LessPay, in Section III, this section shows an overview of the working principle of current AFC transactions, including stored file structure, entrance protocol, and exit protocol. Note that entrance and exit protocols provide important insight for our attack design.

File structure. Among today's AFC systems, the majority of AFC cards follow the ISO/IEC 14443 standard. In this standard,

data in a smart card is stored in a very simple file system, organized in a hierarchical tree structure. Each file is identified by its unique file identifier. As an example, Fig. 2 shows the file structure of CTC. The basic card information including card number, card type, and expiration is stored under the root directory. The data involved in the transactions of bus and metro is stored in the purse directory.

Entrance protocol. When a passenger (with an AFC card) wants to enter a station, the AFC system needs to execute the entrance protocol, as shown in Fig. 3, based on the following three steps.

- First, the station's terminal requests and reads the basic information of this passenger's AFC card, including the card number, the expiration, and the balance. The terminal verifies this information, including checking the expiration and whether the balance is sufficient.
- Second, if the above verification succeeds, the terminal would try to write the entrance data to the `Metro Data` file (just using the metro as an example). However, before writing the entrance data, the AFC card needs to perform a one-way authentication to the terminal. As shown in Fig. 3, the terminal gets a random number R from the AFC card, and then calculates a *MAC* by encrypting R with a pre-installed key[1] shared with this AFC card (right-hand operations in Fig. 5).
- Finally, after generating *MAC*, the terminal sends the entrance data with the calculated *MAC* to the AFC card. The card performs an external authentication (shown in Fig. 5): if passed, the entrance data would be written on the card. On the other hand, the external authentication works as follows. As shown in Fig. 5 (left-hand), the AFC card first encrypts the random number R with the key shared with the terminal. Because the AFC card has received the terminal's *MAC*, which has been computed by encrypting the same random number R with the same key (the right-hand operation in Fig. 5), the AFC card can check whether the terminal's authentication passes through comparing the two ciphertext. If the terminal is fake, the authentication fails.

After the whole protocol is executed, the passenger will be allowed to enter the station, and her AFC card has been written her entrance information.

Exit protocol. When the trip is finished, the passenger taps her card on the exit terminal. The terminal preforms exit protocol, which is shown in Fig. 4, based on the following two steps.

- First, the terminal reads the same basic information as the entrance stage, including the card number and the expiration, as well as the entrance data from the card. Then, the terminal verifies the above information. If the verification succeeds, the terminal calculates the fare that the passenger needs to pay. The verification process is the same as the first step in the entrance protocol.
- Second, in order to upload the transaction log information to the AFC backend, the card and terminal need to perform a mutual authentication with each other. In other words, besides the authentication to the terminal, in this step (called *debit checking* step), the terminal also needs to check whether the AFC card is emulated or fake. The process that the card authenticates the terminal is almost the same as the authentication step in the entrance protocol. On the contrary, *i.e.,* the terminal authenticating the card, the AFC card needs to use its private transaction key *TK* to generate a session key *SK* and a *MAC'* (generated using the *SK*), and then sends them to the terminal for the authentication. The most important property in this step is: *a fake or emulated AFC card cannot have a transaction key to pass the authentication.*

After the mutual authentication, the terminal uploads the transaction information to its backend.

[1]The key of each card is unique in practice. Instead of storing all keys (which is obviously impossible), the key of each card is generated using a root key and its card number. The root key is stored in a so-called SAM module attached on the terminal. The terminal uses SAM to generate the each-card key.

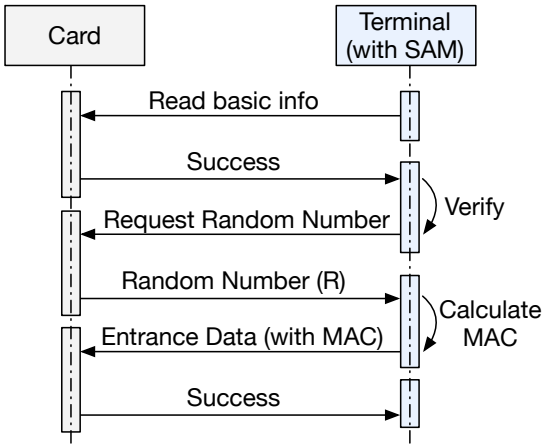


Fig. 3: The entrance protocol.

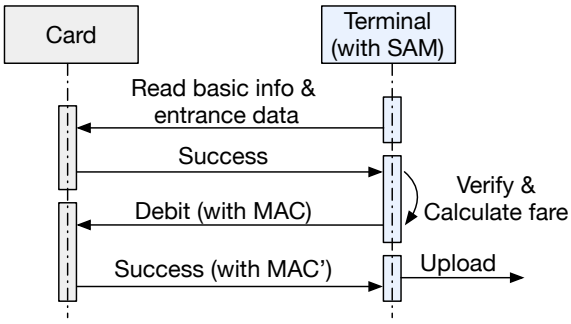


Fig. 4: The exit protocol.

III. Attack Design And LessPay Implementation

In this section, we first present how we design our attack (in Section III-A and Section III-B) and the implementation of the LessPay app (in Section III-C).

As shown in Fig. 1, our attack has six steps (*i.e.,* Step 1-2 and Step 4-7). Step 3 and 8 do not belong to our attack, since

TABLE I: Metro Entrance Data

#	Entrance Data	Enter Time	Metro Line	Station	Balance When Entering
1	1512051417043D014C1D	2015-12-05 14:17	4	Station A	75.00
2	1511301135020801B009	2015-11-30 11:35	2	Station B	24.80
3	15112215225E1D01AC0D	2015-11-22 15:22	X	Station C	35.00
4	15112009560A11016612	2015-11-20 09:56	10	Station D	47.10
5	15111220090401015203	2015-11-12 20:09	1	Station E	8.50

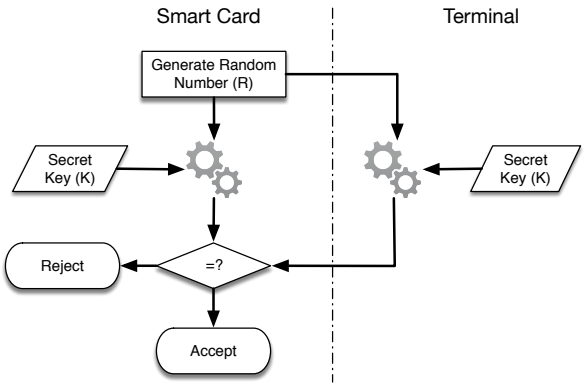


Fig. 5: External authentication, used by the card to validate the terminal.

they occur on the terminal side and are not controlled by us. Step 1-2 and Step 4-7 formulate two important phases in our attack: tampering entrance data (Step 1-2) and relay attack on AFC card (Step 4-7). We now detail each of the phases.

A. Tampering Entrance Data

In order to tamper the entrance data, we need to know two important things: 1) the data structure of entrance data, and 2) the station data, *e.g.,* GPS latitude and longitude coordinates. In this section, we describe a collection of approaches to infer the above information.

Collecting entrance data. Instead of collecting entrance data by physically accessing metro stations, we developed a lightweight app (different from LessPay app) to specifically collect data listed in Fig. 2. To attract users to download the app, the app itself provides useful features including parsing the balance and transaction histories (which metro line and when the user rode, as well as the fare) when the user taps the card on her NFC smartphone. We distributed this app in *Google play*. With the agreement of our users, we collected these anonymous data (the card is innominate) from 97 different cards.

Obtaining data structure of entrance data. By collecting the entrance data, we analyze it and try to learn its structure. For example, Table I lists five items of our collected data. By observing and cross-checking the data, we find that the metro entrance data contains the following elements:

- The entrance time (`yyMMddhhmm` format, 5 bytes[2])
- The entrance metro line number (1 byte)
- The entrance station identifier (1 byte)
- The balance when entering the station (little endian in 2 bytes, *e.g.,* `4C1D` represents `0x1D4C` (`7500`) cents)

Thus, we obtain the data structure of entrance data, as shown in Fig. 6.

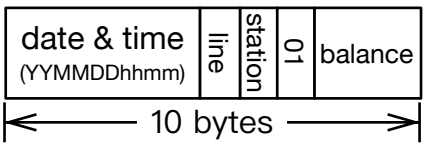


Fig. 6: Data structure of entrance data.

Obtaining station information. Rather than collecting station data by visiting each station (seems impossible), we found a third-party application called E-Card Tapper [10], which is able to parse the transaction histories as well as the trip records and details. Driven by this finding, we reversed this application using Apktool [11] and dumped the station data from the inner SQLite database of E-Card Tapper in order to extract its stored station information, such as the station identifier.

Besides this basic information on stations, we also need to infer the GPS latitude and longitude coordinates of each station. Thus, we get the location coordinates of stations using Google Maps.

Tampering the entrance data. We now already have enough information (*i.e.,* entrance data structure and station information) to tamper the entrance data. In our LessPay implementation, as shown in Fig. 1, the web server in the cloud is responsible for generating the fake entrance data based on our above collected data. To falsify a piece of valid entrance data, we simply prepare the legitimate entrance time, station information, and the balance. In order to *minimize* the fare, our cloud will generate the proper entrance data according to the destination. More details about the implementation of tampering the entrance data is described in Section III-C.

B. Relay Attack on AFC Card

This phase covers Step 4-7 shown in Fig. 1. During this phase, our purpose is to try to pass the mutual authentication in the exit protocol (mentioned in Section II). This is because our emulated card receives a debit from the terminal, and

[2]Noted using patterns for formatting and parsing in JDK 1.8.

the debit is protected by transaction key *TK* via the generated session key *SK* and *MAC'* (mentioned in Section II). In practice, because a contactless smart card is a combination of MCU (microcontroller unit, like the most popular Intel 8051) and an RF (radio frequency) module, under the protection of the firmware in the MCU, the *TK* is not readable. Therefore, it is impossible to emulate an AFC card with debit support. In other words, the challenge in this phase is how we can get a transaction key *TK* for our emulated card to make it pass the mutual authentication.

To address this challenge, we use the physical card equipped with *TK* to bypass this obstacle. This physical card is put in the cloud's AFC card pool (see Fig. 1), and it corresponds to the emulated card that receives the debit from the terminal. In other words, in LessPay, the emulated card should have a corresponding physical AFC card in the cloud-side pool. Our intuition here is to make our emulated card act as a "proxy"-card and make the cloud server together with the physical card act as a "proxy"-reader. Such a design enables emulated card to forward the debit to real card (*i.e.,* the physical card) to generate *MAC'*, because only that physical card has the needed transaction key *TK*.

During Step 4-7 in Fig. 1, the debit message transmitted by the terminal is first received by the "proxy"-card (*i.e.,* the emulated card) and relayed the debit to the cloud server. The cloud server will transmit the debit to a physical AFC card. Since the message is authenticated by *MAC'*, the physical card will assume that it is communicating with a legitimate terminal and respond normally. Then, the response is forwarded to LessPay, which will respond to the terminal with the debit response. Still, the intact message is authenticated by *MAC'*, which is identical as a real card, so the terminal can not distinguish between the physical card and our emulated card.

Using such a relay attack, we are able to overcome the fact that our emulated card lacks *TK*. Moreover, the valid *MAC'* will not lead any inconsistency.

C. System Implementation

Based on the above two important design phases, we are now able to present the system implementation, which consists of a front-end mobile app LessPay and a cloud-side service (*i.e.,* the cloud in Fig. 1). The LessPay app requires an NFC-equipped smartphone with Android 4.4 or above. While in the cloud server side, any regular server or workstation is enough to meet the requirement.

1) LessPay Implementation: Before HCE techniques are proposed, a secure element is required to perform the communication with the NFC terminal, and no Android application is involved in the transaction at all. Nevertheless, from Android 4.4, it is possible to emulate a card using the HCE technology to emulate an ISO/IEC 14443 smart card without a secure element. Emulating an AFC card requires the following three features:

An Application ID (AID). When tapping the phone on a terminal, the HCE service is triggered by a `SELECT` command. This is identified by an AID. The AID of CTC is `1PAY.SYS.DDF01`, which we use to register our app.

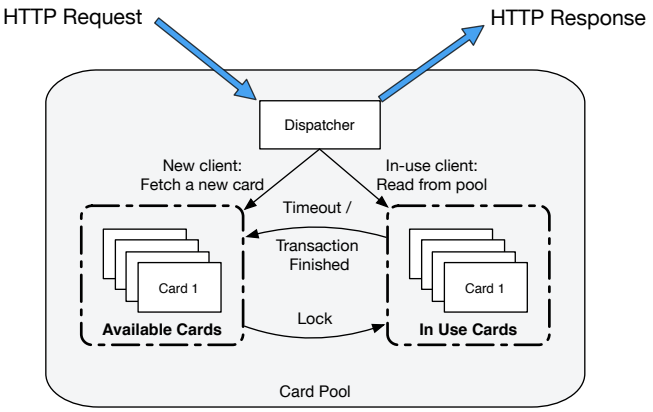


Fig. 7: Card pool scheduler.

An emulated card. An emulated ISO/IEC 14443 card needs to be implemented for communicating with the terminal. As we mentioned in Section II, the data in a card is organized in files. The file structure of this emulated card is the same as the structure shown in Fig. 2. The messages transmitted and received between the card and the terminal are called application protocol data unit (APDU) [4]. The application-level protocol is half-duplex, by implementing a `processCommandApdu` method: the input is the command APDU that the reader sends and the output is the response APDU. The following commands in the standard are implemented in LessPay:

- `SELECT`: Select a different directory.
- `READ BINARY`: Read data from a specific file.
- `UPDATE BINARY`: Update data in a specific file. As we mentioned in Section II, updating a file requires authentication. According to the standard, it is a one-way authentication that the card validates the terminal. In our attack model, we have to trust the terminal and ignore the *MAC* unconditionally. As a result, when the terminal gets a random number, we simply return a fixed one (see next item) and accept the *MAC* without any calculation and comparison.
- `GET RANDOM NUMBER`: We use a fixed number `00000000` instead of random numbers.
- `GET BALANCE`: Return the balance of the card. Note that since the card is reused by many users, therefore the balance is fetched from the cloud when the app starts and updated periodically together with the fake entrance data.

The relayed part. The debit command is protected by *TK* and requires mutual authentication (as we mentioned in Section III-B). Therefore the debit command is relayed to the cloud server. We do not implement this command in an emulated card. We respond to the terminal whatever the cloud server returns.

In order to *minimize* the expected fare, we need to falsify the entrance data of the closest station. To achieve a better user experience, we will not ask the user her destination. Instead,

we use the Android API to locate the user via the Cell-ID and Wi-Fi. We upload the user's location every two minutes. In each HTTP request, we send the user's coordinate to the cloud and get the balance, the card number (see Fig. 4: card number is required to generate the *TK*, *SK*, and *MAC*), as well as the fake entrance data accordingly.

2) Cloud-Side Implementation: The configuration of the deployed server is: 2 × 4-core Xeon CPU E5-2609 @2.50GHz, 8GiB memory, 500GiB 10K-RPM SAS disk, and a 100Mbps network. The system on the cloud side is implemented in Akka 2.4, which is a JVM-based concurrent system.

Fake entrance generator. PostGIS [12], which is a spatial extender for PostgreSQL object-relational database, is used to find the nearest station. Since we are targeting at a relatively small area and City X is not located in high latitude, we choose to use Cartesian distance to measure the distance rather than the spherical distance for a better performance.

"Proxy"-card. We use ACR122u (NXP PN532 based) contactless smart card readers to communicate with the AFC cards. In the 100-user test, we prepared 5 readers and 5 cards. The server itself maintains the usage of different cards. We use an LRU dispatching algorithm to select a card from the cards that were not used in the past two minutes when receiving a request. Each card is set to the state `IN USE` for 2 minutes once we send the card number to the app. After a successful transaction or timing out, the state is set to `AVAILABLE` again. The scheduler is shown in Fig. 7.

IV. Performance Evaluation

This section evaluates LessPay through attacking real AFC systems in City X. In this evaluation, we aim to answer the following three questions:

- How much money users can "save" through using LessPay (in Section IV-B)?
- What is the overhead of using LessPay (in Section IV-C)?
- Whether LessPay can support to large number of users (in Section IV-D)?

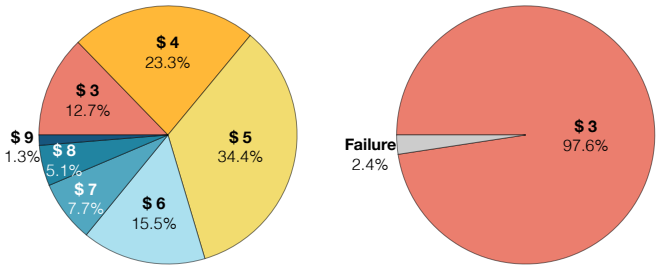
A. Experimental Setup

We recruited 100 volunteers to use LessPay. These users are equipped with HCE Android smartphones. The typical models are Samsung Galaxy S5, Huawei Mate 7, Moto XT1095, and LGE Nexus 5X. 62 users use LTE-TDD network, and the others use LTE-FDD network.

The experiment lasted for three months (from Jan. 10th to Apr. 10th, 2016). Each user was asked to use LessPay 40 times per month, with a total of 12,000 tests performed.

B. How Much We Can Save?

We now answer the first evaluation question: how much money users can "save". The metro fares in City X vary from \$3 to \$9 (in local currency) according to the distance. During the 12,000 tests, the "legitimate" fares are presented in Fig. 8(a). The average fare that users should pay is \$5.03. After using the LessPay app, all users only need to pay \$2.03 instead of the original fare of \$3 (*i.e.,* without using LessPay). This is



(a) Users should pay the fares from \$3 to \$9.

(b) Except for 2.4% failures, users actually paid only \$3.

Fig. 8: The fares that users should pay and actually paid.

clear using LessPay enables users to pay less than the users should pay. \$25,181 in total is "saved" (see Fig. 8(b)).

As shown in Fig. 8(b), we also noticed that among these tests, there are 2.4% cases that do not succeed, which means these 2.4% attacks fail to "save" the money of our users. According to the log, we found that the reason is the poor network connection – the `DEBIT` command requires relatively good quality Internet connection.

C. System Overhead

We evaluate the overhead of LessPay based on two aspects: client-side overhead and cloud-side overhead. The former one means the overhead on smartphones, while the latter one means the overhead on the cloud server side.

Client-side overhead. The client-side overhead of LessPay comes from three sources: memory, network traffic, and battery usage. The typical memory usage is 20MiB, which is modest.

In terms of bandwidth overhead, our measured results show that the size of a single request is 48 bytes (16-byte location and 32-byte user token). The size of a single response is 20 bytes (6-byte card number, 4-byte balance, and 10-byte entrance data). Including TCP handshakes, and TCP / HTTP headers, the total network traffic cost is less than 1KB. The cumulative distribution function (CDF) of network traffic consumed in these 12,000 tests are shown in Fig. 10. The average network traffic in all tests is 21.8KB, which costs only cents. For 80% users, the network traffic cost is less than 36KB. The average total traffic cost in a month (calculated over 40 trips) is less than 1MB.

To understand the overhead of LessPay on battery life, we record the battery power consumption in these tests. As shown in Fig. 11, the average power consumption per trip is 3.4 mWh, which is extremely low given that the battery capacity of popular smartphones lies between 5 - 20 Wh [13].

Cloud-side overhead. Fig. 9 illustrates the CPU utilization of the server on a typical day. The web service is not a CPU-bound application. In most time, the CPU usage is as low as 1 $\sim$ 2%. Even in rush hours (*e.g.,* 7 - 9 A.M.), the CPU usage is below 15%.

The inbound/outbound bandwidth for cloud-side server is quite low. There is no network traffic when no users turn the

app on. As we pointed out, the traffic in each round-trip is less than 1KB. As a result, network with 100Mbps bandwidth is able to serve hundreds of thousands of users.

D. Scalability

We now explore whether LessPay can scale to large number of users. The scalability of LessPay depends on the number of physical cards in the cloud-side pool. In other words, more physical cards can make LessPay support more users. In order to evaluate the scalability of LessPay, we conducted a simulation study. The simulation assumes: 1) users use LessPay in rush hours, 2) all the users use LessPay within two hours, and 3) users' arrivals follow the Poisson distribution. The user can be denied service, if she has to wait for longer than 15 seconds. We present the simulation results – the relationship between the amount of users LessPay can support and the number of physical cards in the card pool – in Fig. 12. We also choose different service denial rates (0.1 and 0.2) to evaluate the scalability of LessPay under different environments. As shown in Fig. 12, even during rush hours, maintaining a card pool size of 150 will satisfy 10,000 users' need, which means LessPay can serve much more users by simply adding a few more cards to the pool. Thus, we conclude that LessPay scales well to large number of users by only maintaining a moderate-sized AFC card pool at the cloud-side.

V. Related Work

This section reviews previous studies on relay attack and attacks on contactless payment, smart card and AFC system.

Relay attack. Attackers have been trying to implement a relay attack using various approaches. Initially, researchers built specific hardware to relay the communication between a smart card and a terminal. Hancke *et al.* [14] used a self-built hardware to increase the distance up to $50m$. They also deeply reviewed relay attacks in [15], discussing relay resistant mechanisms.

With the development of NFC, recent works have focused on relay attacks using mobile phones. Nokia 6131 was the first phone ever produced with NFC capability. Francis *et al.* [16] revealed the possibility to perform a relay attack using COTS devices. In [16]–[18], researchers performed relay attacks using Nokia mobile phones and discussed the feasibility of some countermeasures, such as timing, distance bounding, and GPS-based or network cell-based location.

More recently, researchers focused on relay attacks with Android mobile phones. Roland *et al.* [19], [20] described relay attack equipment and procedures on Android phones. Lee [21] demonstrated an open-source software NFCProxy that is able to proxy transactions using Android phones. Korak and Hutter [22] compared timing on relay attacks using different communication channels. Still some other work relates to privacy or human interaction issues [23], [24].

Contactless payment. Extracting information from the transaction communication between a credit card and a POS terminal using eavesdropping is possible. Haselsteiner and Beitfuß [25] showed a possible way to eavesdrop NFC. They suggested that, while normal communication distances for NFC are up to 10cm, eavesdropping is possible even if there is a distance of several meters between the attacker and the attacked devices. However, this information (mainly credit card numbers, and expiration) can be obtained directly via NFC or even through social engineering. Paget [26] showed the process and later encode this information and wrote to magnetic stripe cards. This attack is also known as downgrade attack, which may not apply nowadays, due to banks refusing magnetic stripe cards and migrating to *Chip and PIN*.

Smart card and AFC system. Originally, the MIFARE chip, which is a memory card chip, was developed as a solution for AFC. In 1994, an AFC system based on MIFARE was first deployed in Oslo, Norway. Ten years after its introduction, the MIFARE Classic was seen as the major candidate for AFC systems. In 2008, however, researchers discovered a serious security flaw in MIFARE Classic cards [27]–[29]. In particular, the cipher algorithm used in MIFARE Classic, known as CRYPTO1, has been reversed and reconstructed in detail, and a relatively easy method to retrieve cryptographic keys was revealed. Since then, the AFC cards have been gradually replaced by processor cards globally.

According to a public report, in Dec. 2010, two engineers from Qihoo used the flaw of MIFARE Classic chip to crack four Beijing Municipal Administration Traffic Cards and modified the balances. [30] Beijing had stopped issuing the MIFARE Classic card since then. The newly issuing cards are processor cards, which are the cards we used in our attack model.

VI. Conclusions and Future Work

A. Conclusions

Today's AFC systems have been globally adopted and billions of AFC cards have been issued all over the world. Among these systems, ISO/IEC 14443 is the main protocol used worldwide, being near universal in East Asia and Europe, and in its early adoption in the rest of the world. Under this background, this paper proposes a new attack on AFC systems, which is scalable and invisible to AFC system operators.

In this paper, we have developed an HCE app, named LessPay, based on our proposed and reported attack, and evaluated the app through real-world experiments. The evaluational results demonstrate the feasibility, practicality and scalability of our approach.

B. Future Work

As this work has shown, the new attack challenges the current thinking about the security of near field payments. Therefore it is time for the industry to take an interest, which leaves the possible contermeasures as future work. From the brute-force to smart solutions, the following ideas could be applied or tested:

1) Replace the system entirely with online transactions. Currently, most systems work offline due to historical reasons, which leaves this flaw that attackers may fool the systems and terminals without being detected. If the

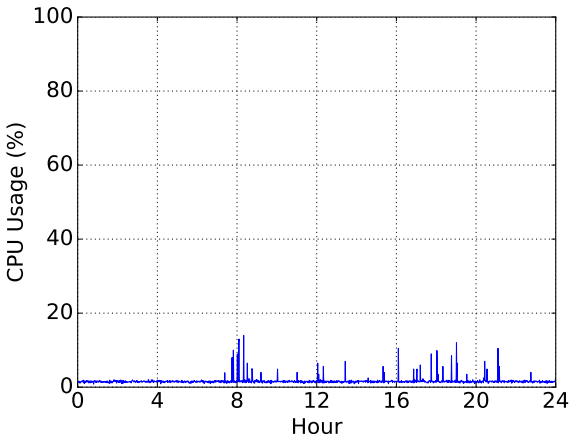


Fig. 9: CPU overhead of the cloud-side server.

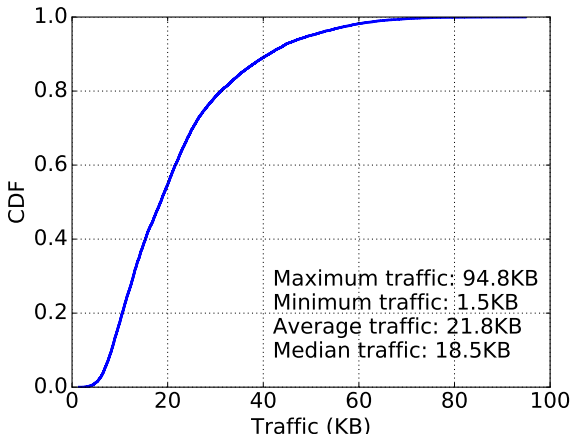


Fig. 10: The network traffic consumed in LessPay.

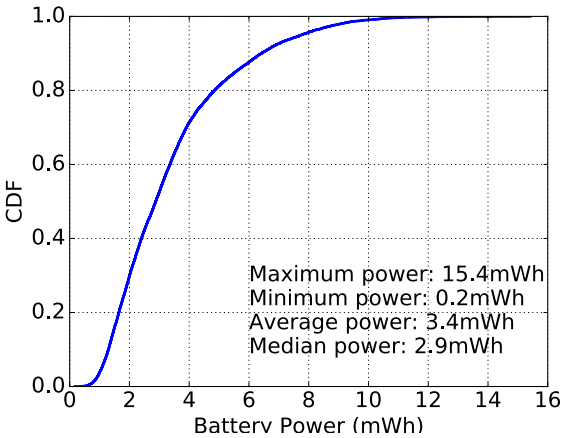


Fig. 11: The battery power consumed in LessPay.

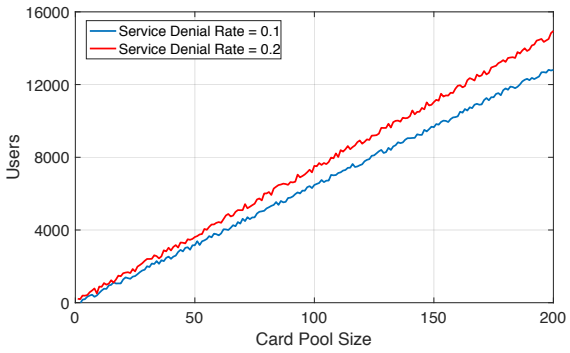


Fig. 12: The amount of users that the card pool can support.

systems are built online as how the banking payments are made (authorized and debited online), it would be much more secure, agile and adaptive.

2) Though online systems are widely used in banking payments and proved to be secure, replacing the whole systems costs too much. Alternatively, adding an authentication code or digital signature to the entrance data may be an easy but limited way to defend this attack. The major limitation is that the authentication code or digital signature must be stored along with the entrance data **statically**, which is a proof that indicates no modification is made since the whole entrance data is written. As a consequence, reading partial data using `READ BINARY` command is possible but not protected by the statically generated proof.
3) ISO/IEC 7816-4 provides a kind of mechanism for secure messaging. It allows encrypted data transmitting between the card and the terminal. However, this requires upgrading the system as well as the cards, which will cost huge amount of money and time.
4) It might be useful to reject the attacker's transaction if we are able to detect the attack. Timing constraints [15], [22], [31] are mainly used protocols to detect possible relay attacks. However, commands in the entrance phase are fully implemented using HCE, which have no significant difference with a physical card. Therefore, these proposed countermeasures would fail if we simply apply timing contraints. New mechanism is required to detect the attack.

ACKNOWLEDGMENT

This work is supported in part by the High-Tech Research and Development Program of China ("863 China Cloud" Major Program) under grant 2015AA01A201, the National Natural Science Foundation of China (NSFC) under grants 61471217, 61432002 and 61632020, the Natural Science Fund of Anhui Province under grant KJ2014ZD31, the CCF-Tencent Open Fund under grant RAGR20160105, and NSFC/RGC Joint Research Scheme under grant 61361166009. Prof. Aziz Mohaisen is supported in part by the NSF under grant CNS-

1643207 and the Global Research Lab (GRL) Program of the National Research Foundation (NRF) funded by Ministry of Science, ICT (Information and Communication Technologies) and Future Planning (NRF-2016K1A1A2912757).

REFERENCES

[1] W. C. Barker and E. Barker, *Recommendation for the triple data encryption algorithm (TDEA) block cipher*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2004.

[2] N.-F. Standard, "Announcing the advanced encryption standard (aes)," *Federal Information Processing Standards Publication*, vol. 197, pp. 1–51, 2001.

[3] "Information technology Security techniques Message Authentication Codes (MACs) Part 1: Mechanisms using a block cipher," International Organization for Standardization, Geneva, Switzerland, ISO/IEC 9797-1:2011.

[4] "Identification cards Integrated circuit cards Part 4: Organization, security and commands for interchange," International Organization for Standardization, Geneva, Switzerland, ISO/IEC 7816-4:2005(E).

[5] "City union card of digital city General technology requirements," Standardization Administration of the People's Republic of China, Beijing, China, GB/T 31778-2015.

[6] "Specification for Contactless ePurse Application (CEPAS)," Singapore Standards Council, Singapore, SS 518:2014.

[7] "Contactless pre-paid/post pay IC card User card," Korean Standards Association, Seoul, South Korea, KS X 6924:2009.

[8] "CIPURSE V2 - Operation and Interface Specification," OSPT Alliance, Munich, Germany, CIPURSE 2.0.

[9] W. Rankl and W. Effing, *Smart Card Handbook*. Wiley, 2010.

[10] "E-card tapper," http://www.wandoujia.com/apps/com.siodata.uplink, [Online; accessed on July 20, 2016].

[11] "Apktool - a tool for reverse engineering android apk files," https://ibotpeaches.github.io/Apktool/, [Online; accessed on July 21, 2016].

[12] "PostGIS Spatial and Geographic Objects for PostgreSQL," http://postgis.net/, [Online; accessed on July 21, 2016].

[13] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855861

[14] G. P. Hancke, "A practical relay attack on iso 14443 proximity cards," *Technical report, University of Cambridge Computer Laboratory*, vol. 59, pp. 382–385, 2005.

[15] G. P. Hancke, K. E. Mayes, and K. Markantonakis, "Confidence in smart token proximity: Relay attacks revisited," *Computers and Security*, vol. 28, no. 7, pp. 615–627, Oct. 2009.

[16] L. Francis, G. Hancke, K. Mayes, and K. Markantonakis, "Practical nfc peer-to-peer relay attack using mobile phones," in *Proceedings of the 6th International Conference on Radio Frequency Identification: Security and Privacy Issues (RFIDSec)*. Springer-Verlag, 2010, pp. 35–49.

[17] K. Markantonakis, "Practical relay attack on contactless transactions by using nfc mobile phones," *Radio Frequency Identification System Security*, vol. 12, p. 21, 2012.

[18] R. Verdult and F. Kooman, "Practical attacks on nfc enabled cell phones," in *2011 3rd International Workshop on Near Field Communication (NFC)*, Feb 2011, pp. 77–82.

[19] M. Roland, J. Langer, and J. Scharinger, *Information Security and Privacy Research: 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. Relay Attacks on Secure Element-Enabled Mobile Devices, pp. 1–12.

[20] M. Roland, J. Langer, and J. Scharinger, "Applying relay attacks to google wallet," in *2013 5th International Workshop on Near Field Communication (NFC)*, Feb 2013, pp. 1–6.

[21] E. Lee, "Nfc hacking: The easy way," *Defcon hacking conference*, vol. 20, 2012.

[22] T. Korak and M. Hutter, "On the power of active relay attacks using custom-made proxies," in *2014 IEEE International Conference on RFID (IEEE RFID)*, April 2014, pp. 126–133.

[23] W. Gu, L. Shangguan, Z. Yang, and Y. Liu, "Sleep hunter: Towards fine grained sleep stage tracking with smartphones," *IEEE Transactions on Mobile Computing*, vol. 15, no. 6, pp. 1514–1527, June 2016.

[24] X. Chen, X. Wu, X. Y. Li, X. Ji, Y. He, and Y. Liu, "Privacy-aware high-quality map generation with participatory sensing," *IEEE Transactions on Mobile Computing*, vol. 15, no. 3, pp. 719–732, March 2016.

[25] E. Haselsteiner and K. Breitfuß, "Security in near field communication (nfc). strengths and weaknesses," in *In Workshop on RFID security*, 2006, pp. 12–14.

[26] K. Paget, "Credit card fraud – the contactless generation," in *ShmooCon*, 2012. [Online]. Available: http://www.tombom.co.uk/Paget-shmoocon-credit-cards.pdf

[27] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs, *Computer Security - ESORICS 2008: 13th European Symposium on Research in Computer Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ch. Dismantling MIFARE Classic, pp. 97–114.

[28] G. de Koning Gans, J.-H. Hoepman, and F. D. Garcia, *Smart Card Research and Advanced Applications: 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ch. A Practical Attack on the MIFARE Classic, pp. 267–282.

[29] N. Courtois, K. Nohl, and S. O'Neil, "Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards." *IACR Cryptology ePrint Archive*, vol. 2008, p. 166, 2008.

[30] "The engineers in qihoo 360 cracked bmac," http://tech.sina.com.cn/i/2011-09-28/17166123872.shtml, [Online; accessed on July 20, 2016].

[31] J. Reid, J. M. G. Nieto, T. Tang, and B. Senadji, "Detecting relay attacks with timing-based protocols," in *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '07. New York, NY, USA: ACM, 2007, pp. 204–213. [Online]. Available: http://doi.acm.org/10.1145/1229285.1229314