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

Fan Dang\textsuperscript{1}, Pengfei Zhou\textsuperscript{1,2}, Zhenhua Li\textsuperscript{1}, Ennai Zhai\textsuperscript{3}, Aziz Mohaisen\textsuperscript{4}, Qingfu Wen\textsuperscript{1}, Mo Li\textsuperscript{5}

\textsuperscript{1} School of Software, Tsinghua University, China
\textsuperscript{2} Beijing Feifanshi Technology Co., Ltd., China
\textsuperscript{3} Department of Computer Science, Yale University, USA
\textsuperscript{4} Department of Computer Science and Engineering, State University of New York at Buffalo, USA
\textsuperscript{5} School of Computer Science and Engineering, Nanyang Technological University, Singapore
Introduction

Automated Fare Collection (AFC) system
Introduction

MIFARE Classic

Processor Cards
External Authentication: a card verifies a terminal
Internal Authentication: a terminal verifies a card
Introduction

Random Number

Data with MAC

Message authentication code: MAC = Digest(data, rnd, key)
Introduction

What is a possible flaw?
Flaw

City Traffic Card

ISO/IEC 14443-4 based
Millions issued
Flaw

1. Entrance Data

2. Entrance Data

3. Calculate Price

4. Debit

5. Auth Code

6. Transaction Log

Database

AFC Backend
In the attack, there are two important phases:

1. **Phase 1: Tampering entrance data.**
   - The user triggers the web server to tamper with the entrance data.
   - The entrance and exit protocols provide important insight for the feasibility and scalability of our designed attack.
   - In summary, this paper makes the following contributions:
     - We construct a large-scale invisible attack on AFC systems.
     - We provide a detailed description of our attack design.
     - We implement our attack on a real-world AFC system.

2. **Phase 2: Relay attack on AFC card.**
   - The entrance and exit protocols provide important insight for the feasibility and scalability of our designed attack.
   - In summary, this paper makes the following contributions:
     - We construct a large-scale invisible attack on AFC systems.
     - We provide a detailed description of our attack design.
     - We implement our attack on a real-world AFC system.

---

**Flaw**

- **Root**
  - **Card Info**
  - **Purse**
    - **Bus Data**
    - **Metro Data**
    - **Transaction History**
with the same key (the right-hand operation in Fig. 4), the AFC card first encrypts writing the entrance data, the AFC card needs to perform a mutual authentication with the terminal. After the mutual authentication, the terminal uploads the entrance data to the AFC backend, the card and terminal need to perform a one-way authentication to the terminal. As shown in Fig. 3, the terminal gets a random number from the AFC card, and then calculates a MAC of the random number with a pre-installed key. If the terminal is fake, the card can check whether the terminal's authentication passes through comparing the two ciphertext. If the terminal is fake, the authentication fails.

Because the AFC card has received the terminal's random number, the AFC card can calculate a MAC of the terminal's random number with its own private transaction key. The terminal authenticating the card is almost the same as the authentication step in the entrance protocol. On the contrary, the verification process is the same as the first step in the entrance protocol.

When the trip is finished, the passenger taps her AFC card on the exit terminal. Fig. 5 details the exit protocol. The verification process is the same as the first step in the entrance protocol, which is shown in Fig. 3.

Entrance protocol.

Exit protocol.

The verification process is the same as the first step in the entrance protocol. When a passenger (with an AFC card) wants to enter a station, the AFC system needs to execute the entrance protocol. After the whole protocol is executed, the passenger will be allowed to enter the station, and her AFC card has been written with the entrance data to the terminal. The terminal uses SAM to generate the each-card key. The verification process is the same as the first step in the entrance protocol. If the verification succeeds, the terminal calculates the fare that the passenger needs to pay. The terminal transfers this information, including checking the expiration and whether the balance is sufficient.
With the same key (the right-hand operation in Fig. 4), the AFC card first encrypts writing the entrance data, the AFC card needs to perform a verification process. The verification process is the same as the first step in the entrance protocol. Because the AFC card has received the terminal's calculated MAC, the terminal gets a random number from the AFC card, and then the terminal authenticates the card. The process that the terminal authenticating the card is almost the same as the authentication step in the entrance protocol. On the contrary, the terminal also needs to check whether the terminal's authentication passes through comparing the two ciphertext. If the terminal is fake or emulated, the key of each card is generated using a root key and obviously impossible, the key of each card is generated using a root key and shared with this AFC card (right-hand operations in Fig. 4). When a passenger (with an AFC card) wants to enter a station, the AFC system needs to execute the entrance protocol, which is shown in Fig. 3. The entrance stage, including the card number and the expiration, is allowed to enter the station, and her AFC card has been written her entrance information. When a passenger (with an AFC card) wants to enter a station, the AFC system needs to execute the entrance protocol, which is shown in Fig. 3. The entrance stage, including the card number and the expiration, is allowed to enter the station, and her AFC card has been written her entrance information. The verification process is the same as the first step in the entrance protocol. Because the AFC card has received the terminal's calculated MAC, the terminal gets a random number from the AFC card, and then the terminal authenticates the card. The process that the terminal authenticating the card is almost the same as the authentication step in the entrance protocol. On the contrary, the terminal also needs to check whether the terminal's authentication passes through comparing the two ciphertext. If the terminal is fake or emulated, the key of each card is generated using a root key and obviously impossible, the key of each card is generated using a root key and shared with this AFC card (right-hand operations in Fig. 4). When a passenger (with an AFC card) wants to enter a station, the AFC system needs to execute the entrance protocol, which is shown in Fig. 3. The entrance stage, including the card number and the expiration, is allowed to enter the station, and her AFC card has been written her entrance information.
Attack model

1. Entrance Data
2. Fake Entrance
3. Calculate Price
4. Debit
5. Debit
6. Auth Code
7. Auth Code
8. Transaction Log

Web Server
Cloud
AFC Card Pool

Entrance
Exit
AFC Backend
Database (always in consistency)

Fig. 1: Architectural overview of our designed attack on an AFC system. Red arrows denote the tampered messages, which allows any Android application to emulate an AFC card and talk directly to an AFC terminal.

Abstract
Automated Fare Collection (AFC) systems have been globally deployed for decades, particularly in public transportation networks. The goal of this study is to investigate the possibility of paying much less than actually required. As 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 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.

System architecture

- **Web Server**: Manages the web interface and communicates with the AFC Backend.
- **Cloud**: Stores the cloud-based data and manages the authentication process.
- **AFC Card Pool**: Stores the AFC cards.
- **Entrance**: The point where the user enters the transportation network.
- **Exit**: The point where the user exits the transportation network.
- **AFC Backend**: Manages the backend database and transactional processes.
- **Database**: Stores the transactional data and authentication related data.

**Attack Model**

1. **Entrance Data**: The user enters the transportation network with their NFC-equipped smartphone.
2. **Fake Entrance**: The attacker tampers with the entrance data to create a fake entrance record.
3. **Calculate Price**: The transportation network calculates the fare based on the fake entrance data.
4. **Debit**: The transportation network debits the user's account based on the calculated fare.
5. **Debit**: The transportation network debits the user's account based on the calculated fare.
6. **Auth Code**: The transportation network sends an authentication code to the user.
7. **Auth Code**: The user verifies the authenticity of the transaction.
8. **Transaction Log**: The transportation network logs the transaction for record keeping.

**Conclusion**
Our real-world experiments demonstrate not only the feasibility of our attack (with 97.6% success rate), but also its low-overhead 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. For future work, we suggest exploring the feasibility of such attacks on different AFC systems and different types of transportation networks.
Tampering Entrance Data

1. Collecting entrance data
   We developed a lightweight app (different from LessPay app) to specifically collect data.
2. Obtaining data structure of entrance data

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

3. Obtaining station information
   Reverse an app E-Card Tapper (e卡贴)
4. Tampering the entrance data
   Location based
System Implementation

Server with 100Mbps network

5 ACR 122u readers with 5 CTC cards

Cellphones:
- Samsung Galaxy S5
- Huawei Mate 7
- Moto XT1095
- LGE Nexus 5X

MNOs:
- LTE-TDD
- LTE-FDD
System Implementation

![Diagram of card pool scheduler with HTTP Request and Response flows](image)

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 a 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 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 [21].

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

Except for 2.4% failures, users actually paid only $3.
Performance

Card Pool Size

Users

Service Denial Rate = 0.1
Service Denial Rate = 0.2
Performance

- Maximum traffic: 94.8KB
- Minimum traffic: 1.5KB
- Average traffic: 21.8KB
- Median traffic: 18.5KB
Countermeasures

1. Switch to online transactions
2. Encrypt/sign data
3. Use secure messaging in ISO/IEC 7816-4
4. Detect relay attack
Conclusions

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