CS-E4300 Network security, 2021-2022
Project 1
Teachers responsible for the project: Tuomas Aura, Aleksi Peltonen

NFC ticket design

In this exercise, you will design a multi-ride amusement park ticket on an NFC memory card platform. This exercise has two purposes: (1) learning secure protocol and application design, (2) learning about NFC and smart card technology.

The ticket lifecycle typically starts with the smart card manufacturer printing a logo on the smart card, formatting the smart card memory with the necessary data structures, and setting the authentication keys. The card is then shipped to the ticket issuer, which sells tickets to users. The ticket issuer at the sales point sets the expiry date, number or rides, or other data related to individual tickets. The ticket is then validated by a reader device. The validation happens at an automated or supervised gate when the user enters the ride. However, in this exercise, you design a smart card ticket application for a small operator that uses cheap blank smartcards as the ticket medium, formats the cards when the first tickets are issued to the card, and uses NFC-enabled mobile phones or similar Android devices for the ticket validation.

When a ticket is validated (i.e., used), it is authenticated with a cryptographic challenge-response protocol, its validity is checked, and the rides count is incremented or decremented, depending on your design. After the validity period is over or the maximum number of uses has been reached, it should no longer be possible to use the ticket. The exact functionality depends on the customer, and there may be more than one correct way to design the ticket system.

Here are some features that are definitely desirable for the carousel tickets:

- Issue tickets with constant number of rides (5)
- Validate the ticket (check expiry time and remaining rides, decrement remaining rides)
- The tickets are valid for a certain time (normally one day, but you can use one minute for testing) from the time when they were issued
- Start the validity period only when the ticket is used for the first time (they can be given as gifts)
- If the tickets have expired or they have been fully used, reformat the card and issue a new ticket (savings on blank tickets and friendlier to the environment)
- Issue additional rides (+5) to a card without erasing any still valid ticket

In case you find the above problems easy and want more challenging ones:

- Implement logging of the ticket events to cloud.
- Implement blacklisting of tickets in the cloud, so that detected forgeries can be added to the blacklist, which is downloaded to the ticker reader. The reader device should be able to work without Internet connectivity, but it should make use of the cloud connection when available.
- Master key for the system is currently stored in an XML file on the reader device. Move the master-key to the *Android keystore.* (Note that the Android app has a key list feature in a menu for debugging purposes. You can ignore that feature and the keys stored by it. It would not be part of the actual application.)

We will use the NXP MIFARE Ultralight C smart card, which contains 192 bytes of memory divided into 48 pages of 4 bytes each. The memory is read and written one page at a time. The first two pages of the memory contain the UID, which is a 7-byte unique card identifier. Pages 2 and 40 each contain two lock bytes, which can be used for write-

protecting selected memory pages. Page 3 is a one-time programmable page (OTP) with 32 bits that can be set to one but not reset back to zero. It can be used as a kind of unary counter or for recording other irreversible state changes. Page 41 contains two bytes that form a 16-bit counter, which is a newer feature and easier to use than the OTP page. Pages 42-43 are used to configure authentication parameters, which can be set to require authentication before writing, or before reading and writing. The last four pages, i.e., pages 44-47, contain the authentication key, which is not readable. However, the key can be changed as needed. Pages 4-39 are normal memory that can be used for application data.

In order to design the ticket application, you have to understand what security features the Ultralight C smart card provides and which security goals an event tickets should fulfill. At minimum, *the user should not be able to create counterfeit tickets, increment the remaining ride count, copy the ticket, or restore and replay old tickets*. Ultralight C has "BREAKMEIFYOUCAN!" as the default 16-byte authentication key. You should use the default key to format the blank card and then change the key to a secret one as part of the process. The reader should recognize both black and formatted cards. In addition to being secure, *the ticket application should be reliable: any failures will upset the customers* or lead to disputes. Ultralight C is a very simple smart card, and it does not have many built-in features to protect against failures if, for example, the card is pulled away from the reader in the middle of the transaction. Instead, the reliability needs to be implemented with careful ticket and software design. The information shown by the ticket application should be short but easy to understand for both part staff and for the customers.

The recommended platform for implementing the ticket app is Android. You should be able to use any Android phone with NFC interface. You can use the provided Android application code as the starting point. In that case, you need to install Android Studio. All your code should probably go into the **Ticket.java** file, and you will need to submit this file together with your project reports. However, you are allowed to edit any of the provided code or even to write your own application from scratch.

You will be provided with some MIFARE Ultralight C smart cards. This simple card is not programmable; it is more like a secure memory card. It is easy to brick the smart card by locking pages or by setting a key that you do not remember. If this happens, you can ask for new cards. The number of cards available in November 2021 due to supply chain issues. For this reason, please avoid playing with the lock bits in the card; setting them is not necessary in the project. The cards do *not* need to be returned at the end of the course. If you do not have an Android phone with NFC capabilities, we have a few that can be borrowed. The location and time for picking up the cards will be in MyCourses. Please contact cs-e4300@aalto.fi if you need to borrow a phone. Include your student number and the time window when you expect to need the phone.

For the demo and submission instructions, see MyCourses.

Appendix A: Information sources

Ultralight C data sheet (*the most important information is in section 7.5*):
https://www.nxp.com/docs/en/data-sheet/MF0ICU2.pdf
NFC TagInfo by NXP in the Google Play store for reading smart card contents:
https://play.google.com/store/apps/details?id=com.nxp.taginfolite
(TagInfo is easier to use than the built-in debugging features in the provided app code.)

Appendix B: Additional information not required for the exercise

If you want to play additionally with a PC-based reader, SCL011 readers can be obtained from the course staff. We do not officially support the PC readers for the project because the PC/SC APIs are very archaic compared to Android smart card APIs. However, you can ask the course staff for an old codebase.

The Ultralight C card can also be formatted to function as an NDEF tag if you write the right bytes to the right pages. NDEF tags can be scanned with NFC smart phones and contain information such as URLs. Creating NDEF tags is not part of the course exercise, but you may want to play with the cards after completing the exercise.