Report NFC Ticket Design, Group 7
Soumya Lekkala, Anastasia Safargalieva

Memory management

Page Number		Contains
Decimal	Hex	
0-3	00h-03h	Serial number and OTP
4	04h	Application Tag (constant string specific to our application and ticket design)
5	05h	Version (constant string specific to our application and ticket design)
6	06h	Maximum counter value
7	07h	InitialCounterState
8	08h	Issue Date
9	09h	First Use(Timestamp when the ticket is used for the first time)
10	A	HMAC from issue (static values)
11	B	HMAC from use (static values + page 9)
41	29h	Counter used for rides
42	2Ah	Authentication configuration (auth0 set from 06h)
43	2Bh	Authentication configuration (auth1 set to 0 to restrict read and write access)
44-47	2Ch-2Fh	Authentication Key

Authentication of the card

Authentication of the card is the first step. In the beginning each new card has manufacturer default values for authentication key and hmac. We keep the default authentication key and default HMAC in secrets.xml.

We added one page of InitialCounterState to detect if the card is used for the first time. Also when tapping the card for the first time we can see if the card has already been in use or not by the authentication key with which the card was authenticated. If the card was blank, it was authenticated with the default key. If it was already in use, then the authentication happened with a diversified key. We are also storing the Issue Date to the card for any future purposes.

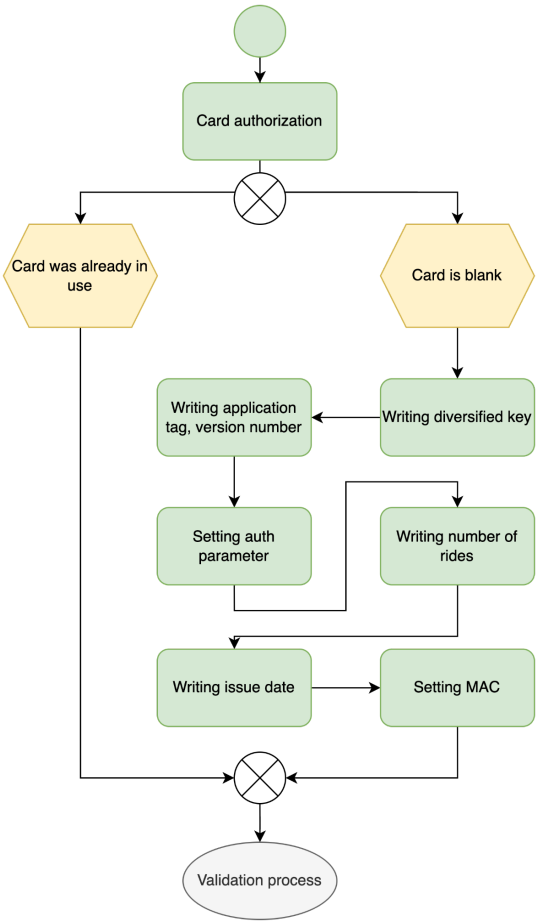
In case we have blank card, we want to change the key with diversified one, what can be computed by taking hash function of master secret and serial number of the card:
K = h(master secret | UID).

We also rely on the **application tag** and **version number.** If the correct application tag can be found, then we assume that the card was in use and that the authentication key has been changed to the application-specific (and diversified) key. If the application tag and version of the card is blank then we assume the card is new and we have to reset the default authentication key to application-specific one.

We are also setting up the authentication parameter.

AUTH1 (page 43, byte 0) – read and write operations are restricted.
AUTH0 (page 42, byte 0) – contains the address of the page from which access rights defined in AUTH1 are applied.

In the end, we calculate the HMAC value of the previously set data.



Pic. 1. Authentication of the card

Ticket validation

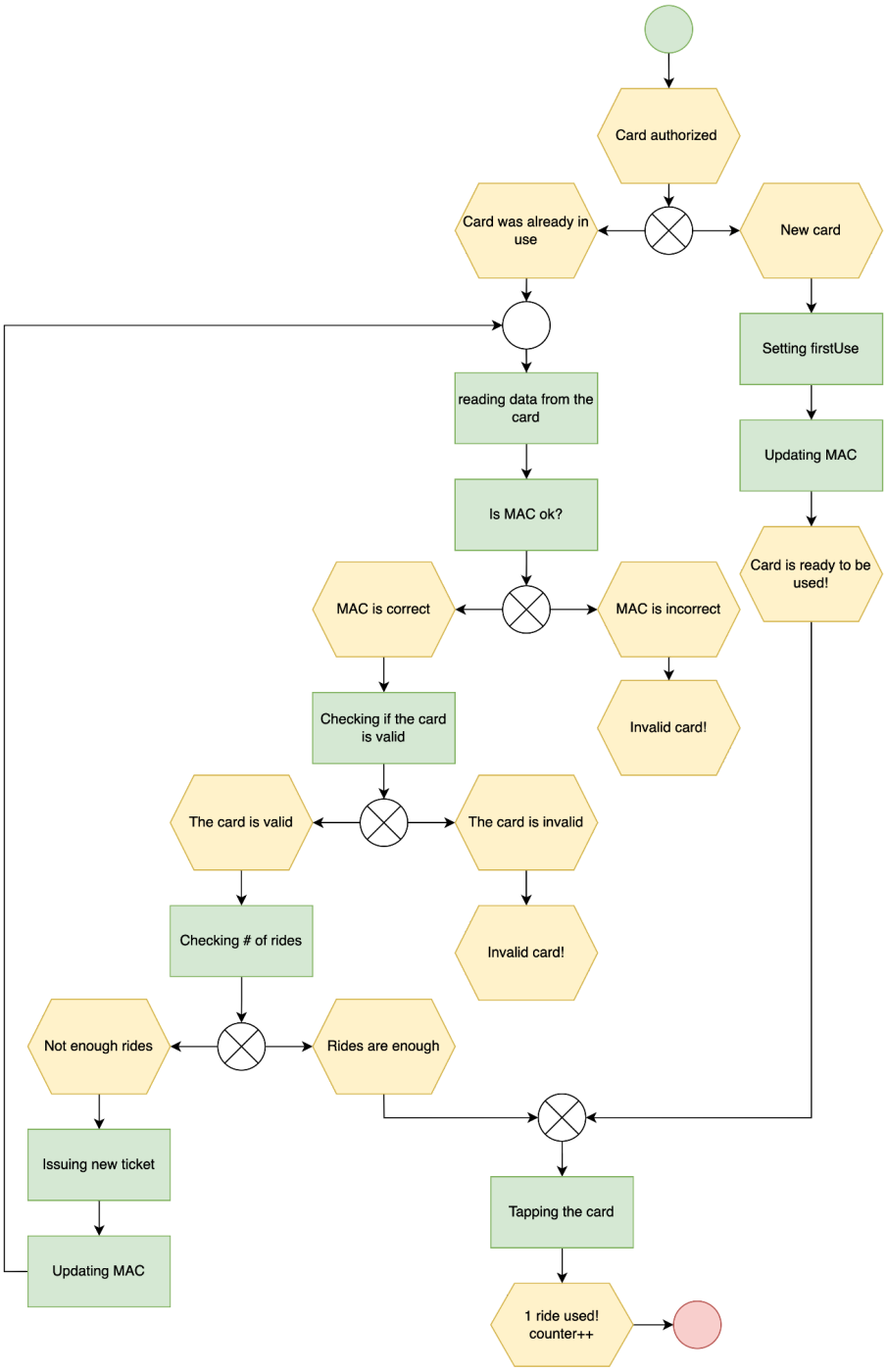
Validation of the ticket differs if it is validated for the first time and if the validation is subsequent. After successful authentication, the card is being checked if it is valid. If the ticket is validated for the first time, then after this we write the issue date of the ticket. We can issue a ticket on one day and use it for the first time several days later. For these purposes we planned 2 pages of the memory. issueDate is on the 8th page and firstUse is on the 9th page. Timestamp will be set to the current time when a ticket is first used. We also recalculate MAC, including firstUse information.

If there is subsequent validation, to validate the ticket, we read all information from the card. We firstly check that HMAC is correct. If it's not, then we deal with invalid ticket. If HMAC is correct, we check for the number of available rides. If there are enough rides, the next step is to increment the counter on the ticket and use 1 ride.

If tearing happens at any of the above steps, the user can simply retry.

We convert byte array of counterState to integers to increment the number of rides. Then, to write this information to the memory we convert counterState to bytes again.

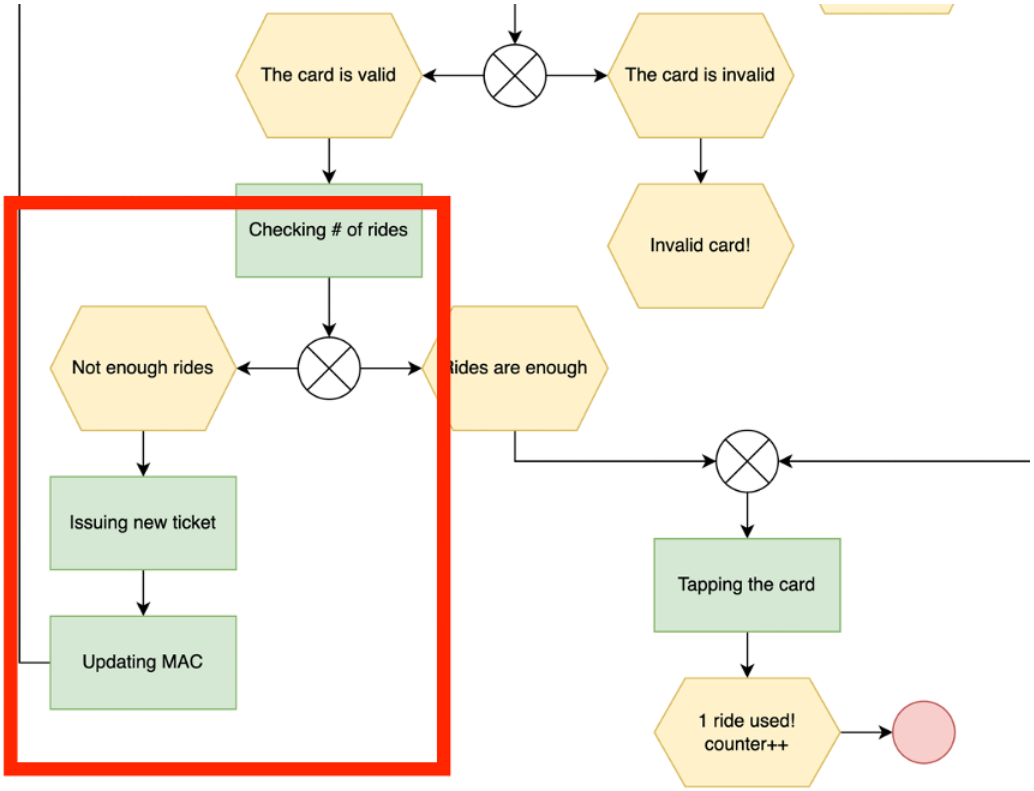
The current number of tickets can be retrieved as the difference between counterState(= max.number of rides) and initialCounter, which is written on page 41. initialCounter is used to count the rides, we can only increment the number of rides. It is done to prevent rollback attacks.



Pic. 2. Ticket validation

Adding new rides

If the ticket has not expired yet and there are no more rides, we can issue additional rides. We update maxNumber of rides and then update MAC.



Pic. 3. Adding new rides

Attacks and what was done to prevent them

Counterfeit tickets.

Mutual secret key authentication (pages 44-47) allows us to check that the ticket is authentic. We also implemented authentication parameters.

AUTH1 (page 43, byte 0) – write operations are restricted
AUTH0 (page 42, byte 0) – contains the address of the page from which access rights defined in AUTH1 are applied.

Setting AUTH1 parameter also prevents replay attack, since it doesn't allow the attacker to read the data from the card.

Each card has its own key specific key, which is calculated as following:

Auth Key = h(Master Secret|Card UID)

This prevents affecting the tickets in the system if the key was leaked. Even if the key was leaked, the attacker would still need to get a master secret, which is stored secretly and the rest of the cards won't be affected.

Man in the middle attack.

In order to prevent the attacker from modifying the content of the card, we compute MAC to protect the integrity of the card. Message authentication code (MAC) is computed over the static information of the card. When reading the ticket data verify that static MAC and backup MAC match.

With MAC, even if the card was tampered, the tampering will be discovered.

Auth Key = h(Master Secret|Card UID)
MAC = h(max number of rides | issue date | first use)

Mac contains only static data. It is written in one page on the card since it is only 4 bytes. One page is enough because even in this case if the attacker wants to tamper the card, he will need to tap the card at least 2^31 times in order to make one malicious change to the card contents

Replay attack.

It is possible to relay the earlier card state. That is, the attacker reads the card memory after the ticket was issued. To protect memory against reading, we set the AUTH1 parameter.

Roll back attack.

A kind of man in the middle attack where the card is authenticated with a reader and the attacker rewrite old data into the card including the HMAC. To prevent it, we use an OTP counter to count used rides, which is done by incrementing the number of rides.

it is more secure to use the counter for counting used rides than to write the number of remaining rides to the normal memory pages. The MitM attacker cannot revert the counter value to an earlier state.

Tearing.

There is the possibility that the user will pull the card away from the reader too early, in the middle of writing. This easily happens when "tapping" the reader in a hurry. If the card is removed from the reader in the middle of writing the MAC, the result may be a card with a partly written MAC. After that, the reader will reject the ticket as invalid.

Tapping operations do not update MAC or other multi-page content on the card. Card authentication and MAC is used to protect static content of the original, issued ticket, such

as expiry date and value loaded to a wallet. Then, we use the 16-bit counter or OTP to keep track of how much of the issued ticket has been used.

In order to mitigate tearing, we need to minimize writing operations, however, if it is the first use of the ticket, following writing operation happen:

1. Setting firstUse time
2. Updating MAC with firstUse page
3. Incrementing the counter