Hi, we are group ten. This is the demo for the network security course project one.

Our demo is divided into five parts. I will introduce different parts of our project in the first four parts. In the final part, I will summarize the highlight that differentiates our project from others.

We implemented the backend with Deno and PostgreSQL based on the requirement that the reader can work without the Internet and update when the Internet connection is available. The server authentication is implemented with the help of HTTPS and TLS certificates. The client authentication is implemented as an API secret for the app and the login token for the admin. We store the card and HMAC master key in the cloud. We have a cache for the keys by encrypting using the Android Keystore and storing them in the Android shared preference. It is available locally for two minutes before expiry. Every time the cache expires, we fetch the keys from the cloud and compare them with the local ones. If any difference is detected, we send a log notifying the malicious issue to the cloud and stop the reader. We have also implemented blacklisting tickets in the cloud so stolen, hacked, or cloned tickets can get blocked manually. The reader also sends logs to the cloud recording the card UID, time, and remaining uses for notable events.

The backend renders the login page, ticket-blocking and logs viewing or clearing management pages. This is the login page for the admin. Admin can input their token string for login.

Admin can block and

unblock the card by UID

as well as viewing the logs. Because of privacy regulations, the admin can choose to manually wipe out the old logs by specifying the time to keep them.

Here is the ticket data structure. We make the first two pages the tag block, the following ten pages are for ticket data, and the last ten are for the logs. The ticket data block is calculated according to the counter parity, even for block two and odd for block one, so the previous ticket is always retained. A counter update commits the write operation to resolve the tearing issue. We use the card-featured counter for counting used rides. We write the initial counter value when issuing the ticket and the expected counter value when the ticket is used to prevent the man-in-the-middle attacker. We write the last check-in time on the card to avoid passing back. We set the expiration time to zero and check whether the card is the first use by matching the ticket's initial counter value with the card counter. We start the validity period only when the ticket is used for the first time. All the timestamps are stored in seconds. The message authentication code is calculated with UID and ticket data.

For issuing, we check the card UID, determine whether the card is blocked, and then calculate the card key. We first assume the card is blank and authenticate the card with the default key. If we succeed, we set the header, update the password, and issue tickets with five rides. If it fails, we authenticate with our key and check the header, read the ticket data from the block according to the counter parity, and then check if the ticket data is valid. To keep maximum rides under twenty, if ticket data is valid, but the remaining rides exceed fifteen, we reject top-ups more.

During issuing a new ticket, we set AUTH zero to three and AUTH1 to zero to protect the whole card memory against reading and writing without authentication.

If the flows get interrupted, like the user tearing the ticket, we will show "Communication error!" to the user. For using, the authentication and integrity checking parts are the same as the issuing. If all of those are successful, we check if the ticket's last check-in time is within five seconds, used up, or expired. If the initial counter value matches the current one, set the expiry time to two minutes later. Then write the ticket data to another new block, increase the counter, and log the event. The delay of the whole use flow shall be less than three hundred milliseconds and typically around two hundred and fifty milliseconds.

Then, the highlight. We realized all the features described in the project material. The API master secret is hidden in the source code using a plugin that stores the obfuscated encrypted secret into an NDK binary. All the keys used, including the API key and card key, hmac key, are diversely calculated by the master key combined with card UID using the slow hash function PBKDF two and safe hash function SHA five one two. We protect the whole memory instead of starting from the ticket block and ignoring the tag block. Otherwise, the attacker can modify the header to play a downgrade attack or just a Deny of Service attack by destroying the ticket. Of course, locking the tag block may be a good solution, but we shall make the ticket recyclable and upgradeable. Finally, as we fetch the hmac and card master key from the cloud, we introduce the distributed verification by expiry policy, so that we can ensure the integrity of the keys stored in the cloud.

Now it's the demo time. We will show you how the app performs.