UMazda

UMass Amherst

Ryan Lee, Ibrahima Keita, Dung Nguyen, Gilbert Hoermann, Sergio Ly
Advised by: Prof. Wayne Burleson
April 24, 2023

Design Overview

Our design used a modified version of SCRAM-SHA256 to perform mutual authentication and p256 ECDSA signatures to authenticate features. We generated the shared secrets used in SCRAM-SHA256 by using HMAC to combine a base secret and the car ID and PIN. We achieved channel binding for subsequent post-authentication messages by deriving a protocol key from the SCRAM exchange and using that as an HMAC key. Besides of these cryptographic features, we also used the MPU to completely disable code execution from RAM.

Defensive Highlight

Our design uses the following security features:

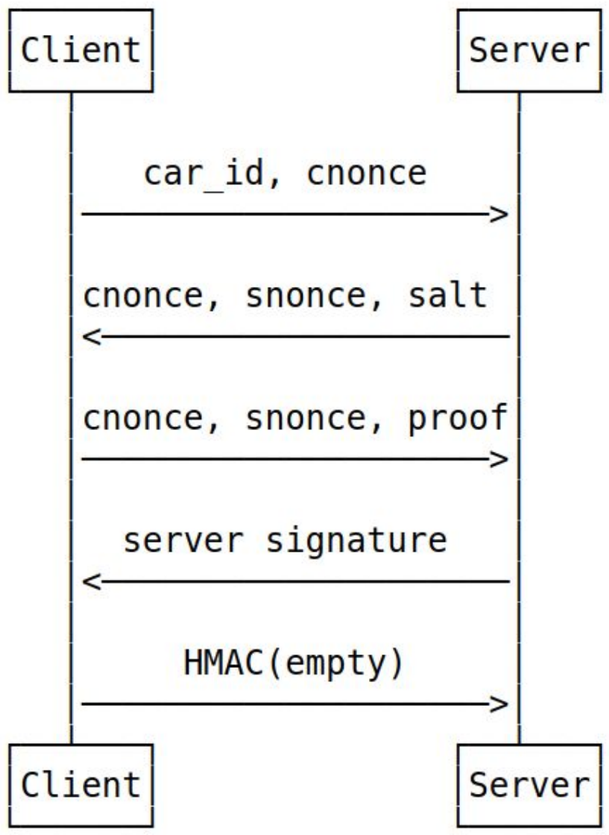
- Modification of SCRAM-SHA256 (“SCRAMish”) for mutual authentication, used in car unlocking and fob pairing.
- p256 ECDSA signatures for features.
- RNG for SCRAMish, initially seeded from a random seed in the EEPROM and reseeded based on human timings.
- MPU enabling to disable all code execution from SRAM.

These are the changes we made to create SCRAMish:

- We used argon2id for deriving a salted hashed password instead of PBKDF2 and used hardcoded parameters instead of exchanging them during authentication.
- We omit negotiation of channel binding.
- All the messages and fields are fixed-size.
- We extract a protocol_key for channel binding HMAC-SHA256(SHA256(client_key), "PARED session key" || client_key || auth), used as an HMAC key for future messages in that session.
- The client confirms authentication success by sending an empty MAC-protected success message to the server instead of signaling failure closing the connection as would be done with TCP in a non-embedded system. (The server would have confirmed authentication by sending the ServerSignature, so the other direction is unnecessary.)

The shared secrets to be authenticated were derived with HMAC, mixing global secrets with the car ID and pairing PIN. We also ensured that the car and paired fob did not have global secrets that they did not need.

The protocol was cryptographically secure but did not provide complete protection due to insufficiently secure RNG. We used a ChaCha based RNG but did not reseed it until after the first complete attempt at a protocol. Unfortunately, this meant that reflashing the board images would completely reset the RNG, allowing for state reset based attacks. To fix this, we would need to add more entropy sources and mix them in immediately before using the RNG.



Offensive Highlight

One team used the following authentication flow for unlocking the car:

1. The fob sends an AES-encrypted random value to the car.
2. The car sends an AES-encrypted random value to the fob.
3. The devices XOR the random values to get a session key.
4. The fob uses the session key to encrypt an unlock packet and sends it to the car.
5. The car decrypts the packet, checks its contents, and unlocks the car if the contents are correct.

If we send the car the exact same message that it would send, it would derive a session key of all 0’s. We could then send an unlock packet encrypted with this known key, thereby breaking the system’s security. Predicting the car’s message is possible because of the use of a static seed for the RNG, which could be reset by re-flashing the car firmware onto the device.

To exploit this flaw, we captured the value the car would send to the fob, re-flashed the car’s firmware, and replayed this value, followed by a properly encrypted unlock packet.

There are several immediate fixes for this flaw, all of which should be applied:

- Combine the random values with a proper KDF
- Use other sources of entropy to prevent RNG state reset.

However, a better fix would be to adopt an authentication protocol that provides proper integrity and authenticity guarantees for transmitted messages.

References

1.