Wh1t3h4t5

Singapore Management University

Cheah King Yeh, Won Ying Keat
Advised by: Mr Lee Yeow Leong
April 24, 2023

Design Overview

- Utilized XChaCha20-Poly1305 for authenticated encryption to ensure integrity and confidentiality
- Adopted a Trusted-Third-Party (TTP) scheme to sign packages and keys to reduce probability of MiTM and impersonation attacks
- Implemented an approach similar to the SIGMA protocol for Authenticated Key Exchange for secure fob pairing

Defensive Highlight

Authenticated Key Exchange Protocol

- XChaCha20-Poly1305 for Authenticated Encryption
- X25519 for shared secret key derivation, EdDSA for signing data and Blake2b for hashing operations
- Random number generation utilized entropy gathered from clock timing for initializing barriers in hardware (dsb, isb)
- Developed a protocol similar to SIGMA[1] for AKE used during fob pairing process
  1. Checking signature of paired fob's (PF) long term public key ensures the authenticity of key
  2. Generates ephemeral keys for DH key exchange
  3. Shared key derived using the formula:

Key1 || **Key2** = H(shared_secret || PF's temp pk || UPF's temp pk)

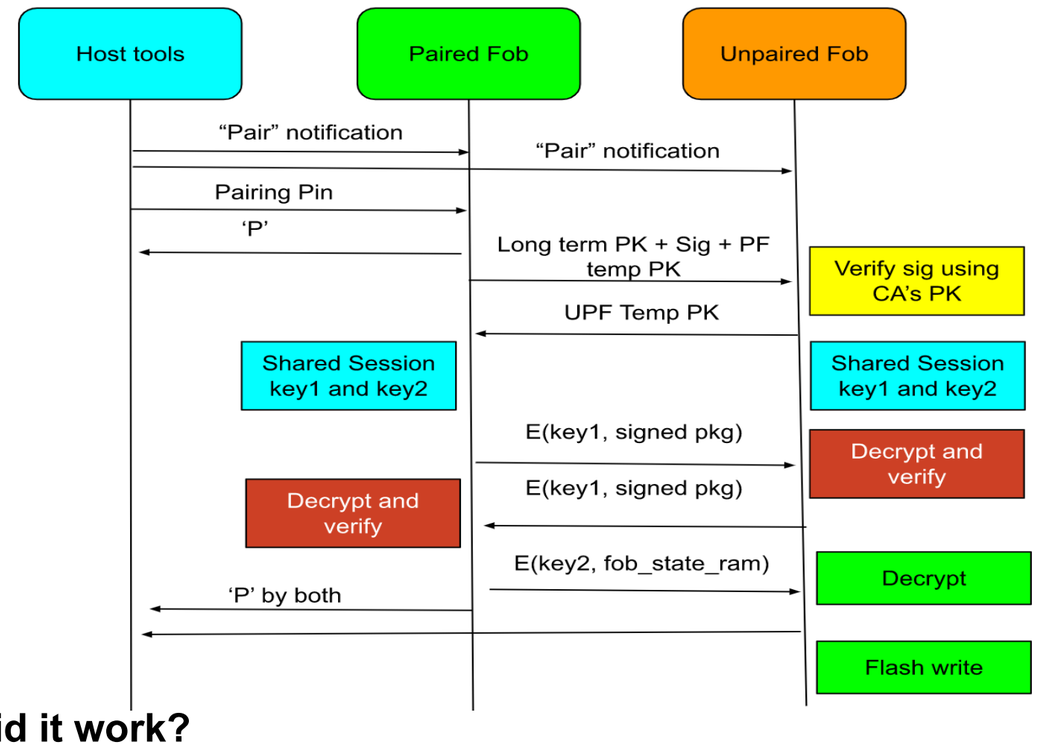
  4. PF will prove its identity by signing and encrypting

E_{key1}("PU" || $sign_{PF\_longterm\_SK}$(PF temp pk || UPF temp pk))

  5. Similarly, the unpaired fob performs the operation

E_{key1}("UP" || $sign_{UPF\_longterm\_SK}$(UPF temp pk || PF temp pk))

  6. Brings benefits like Perfect Forward Secrecy



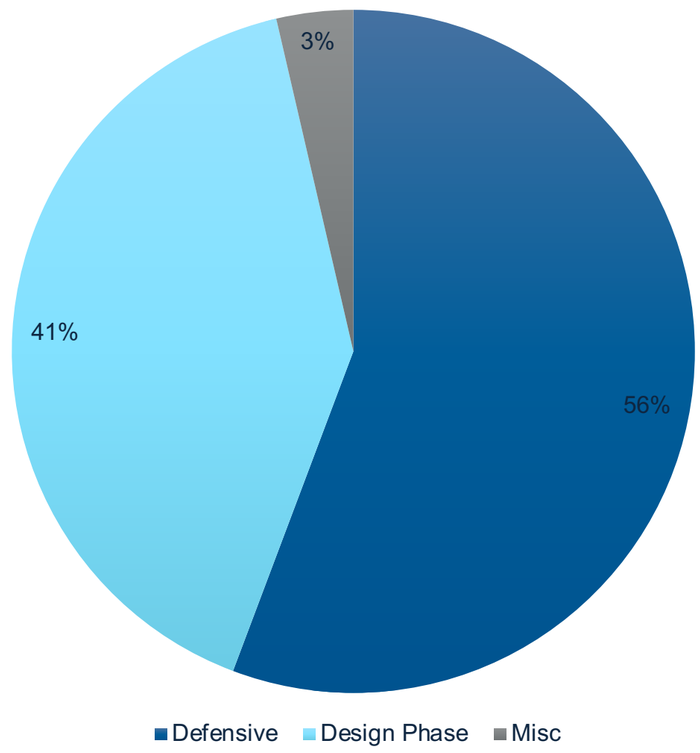
Did it work?

- Seems like it did. We managed to secure our flags for *passive unlock* and *leaked pairing pin*

Future Considerations

- Adopt other more widely used protocols for AKE
- Put in place brute force protection to prevent total protocol bypass!

Figure 1: Points Breakdown



Offensive Highlight

Brute Force Pairing Pin

- Many teams failed to include brute force protections, or had poor protections that can be bypassed
- Sending pin combinations across UART for all different 6-character pin combinations
- Use the fob as an oracle, since it will return early upon a wrong pin and would continue execution otherwise
- Continuously prompt to "pair" so long as the pin is wrong

Classic Buffer Overflows

- Many teams had overlooked the built in *uart_readline* function that potentially allow for reading in an arbitrary large number of bytes prior to termination upon a newline
- Lead to potential buffer overflows, changing local variables to potentially edit the state of the fob during operations like *pairing* or *enabling* of new features
- Multiple variables could be modified via this method

Attack PoC

- Connect to fob bridge using netcat or python
- Send a lot of bytes!

Did it work? ~~(Spoiler: Maybe!)~~

- All attacks described are theoretical and yet to be proven
- They are plausible attacks inferred from pure static code review and analysis. No successful attacks were performed on actual devices (yet).

Possible fixes

- Perform input length checking for all areas of user input
- Store number of wrong attempts in flash (or other permanent memory regions) to prevent brute force or reset attempts

References

1. https://www.iacr.org/cryptodb/archive/2003/CRYPTO/1495/1495.pdf