Building a Secure Cross-Device Communication Channel for Smart Devices based on App Accounts

Lanlan Pan (潘蓝兰)

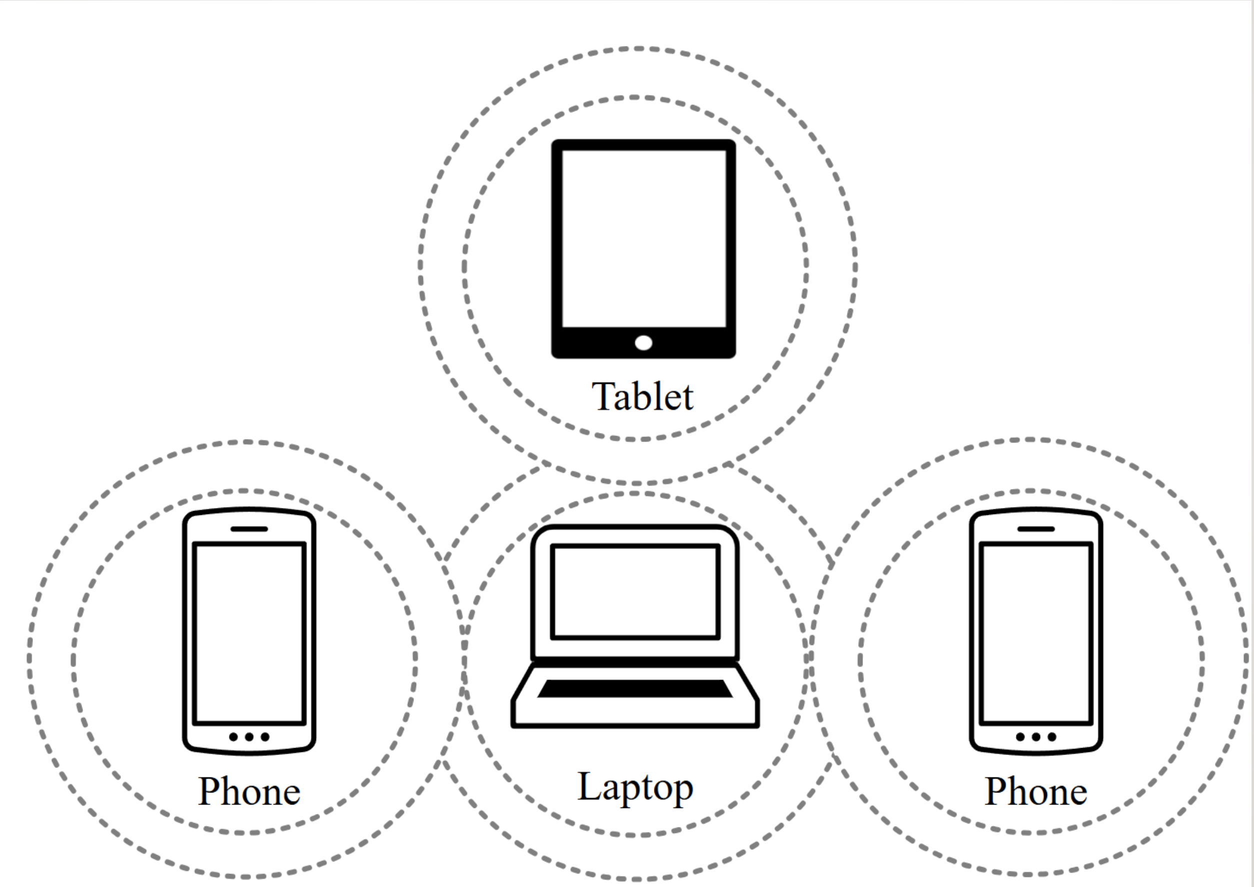
abbypan@gmail.com

Outline

- Introduction
- Challenges
- Our Scheme
- Conclusion

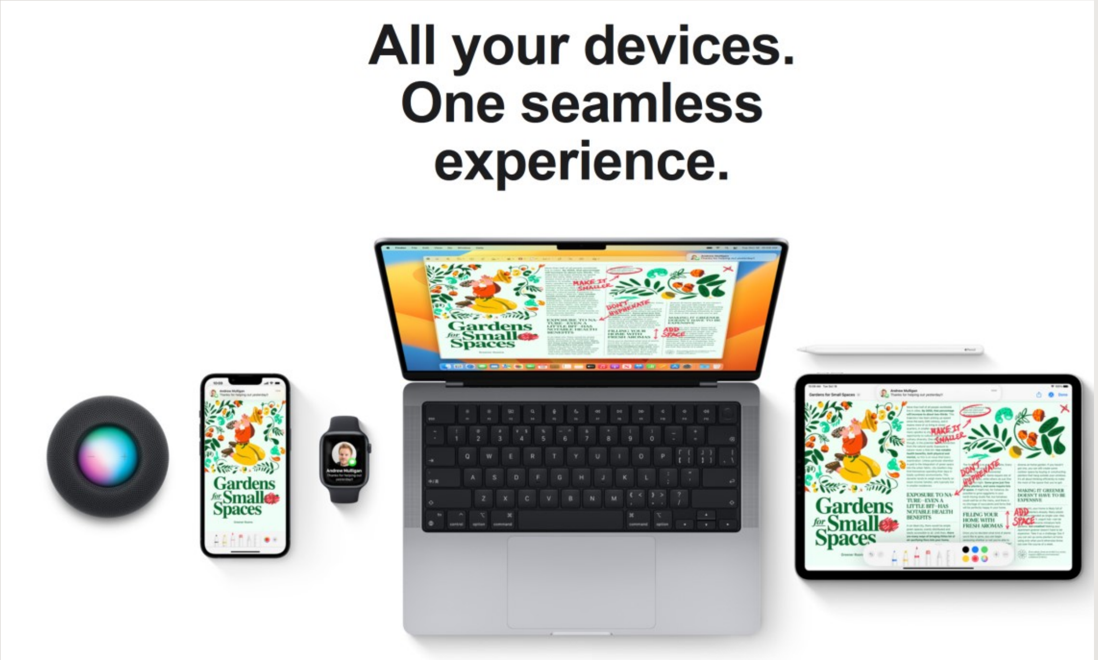
Introduction

- Communication is no longer only between device and cloud, but also exists between devices.

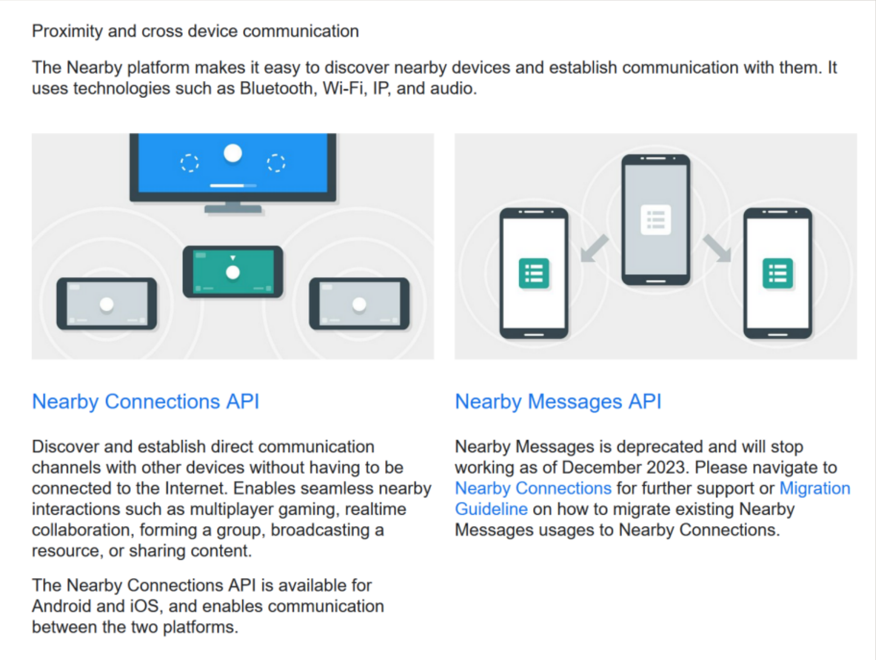


Cross-device Communication

- Device manufacturers offer their cross-device communication service in their own device ecosystem, bound with manufacturer accounts, such as Apple Continuity and Google Nearby.



https://www.apple.com/macos/continuity/


Proximity and cross device communication

The Nearby platform makes it easy to discover nearby devices and establish communication with them. It uses technologies such as Bluetooth, Wi-Fi, IP, and audio.

Nearby Connections API

Discover and establish direct communication channels with other devices without having to be connected to the Internet. Enables seamless nearby interactions such as multiplayer gaming, realtime collaboration, forming a group, broadcasting a resource, or sharing content.

The Nearby Connections API is available for Android and iOS, and enables communication between the two platforms.

Nearby Messages API

Nearby Messages is deprecated and will stop working as of December 2023. Please navigate to Nearby Connections for further support or Migration Guideline on how to migrate existing Nearby Messages usages to Nearby Connections.

https://developers.google.com/nearby

Cross-device Communication

- Apple Continuity
  - Manufacturer account, long-term certificate/public key.
  - Homekit Accessory Protocol (HAP): station-to-station (STS), based on paired long-term public key.
  - Larger data is established with mutual TLS based on user identity certificate, such as AirDrop.

- Google Nearby
  - Manufacturer account, long-term public key.
  - Device create a new pair of public and private keys bound with user identity.

Challenges

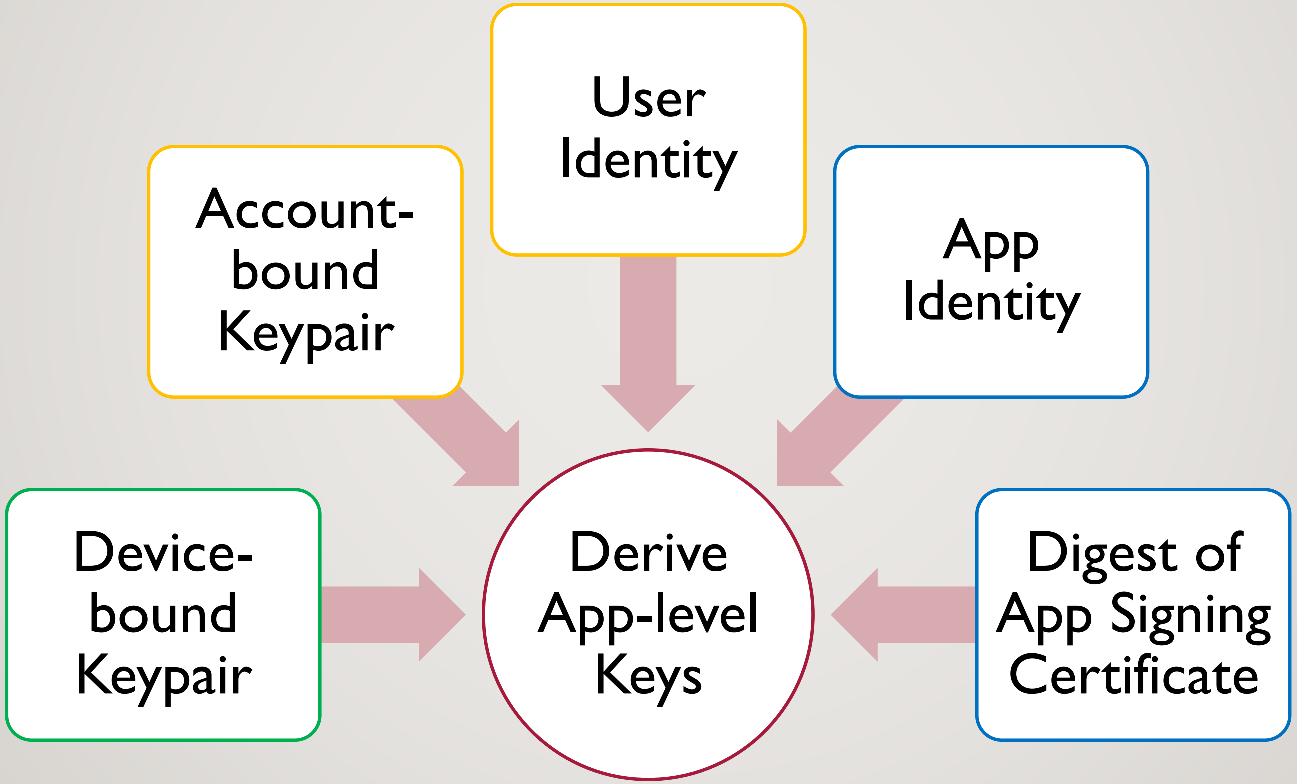
- Cross-manufacturer support.
  - Users often possess smart devices from various manufacturers and utilize apps across these devices.
  - Users should log in with an app account and sync data between the devices from different manufacturers.
  - Building a secure cross-device communication channel for smart devices based on app accounts is critical.
- App-level end-to-end security.
- Identity tracking.
  - The device may use a long-term manufacturer account certificate containing a unique identifier to setup TLS connection.
  - TLS versions below 1.3 transmit certificates in plain text.
  - The unique identifier in the certificate enables adversaries with access to network traffic to track users.
- App impersonation attack.
  - Attackers may create clone apps that mimic legitimate ones.
  - If the device cannot distinguish the legitimate app from fake apps, an attacker could use the fake apps to make cross-device communication for malicious use, leading to user data leakage.

Our Scheme

- Building a Secure Cross-Device Communication Channel for Smart Devices based on App Accounts.
- Support cross-manufacturer scenarios.
- Achieve app-level end-to-end security.
  - Propose an app-level key derivation method associated with the app account extended from BIP-0032.
  - Separate individual cross-device channels for different app account communication scenarios with our three types of derived app-level keys.
- Use Noise as the communication protocol, and choose three Noise handshake patterns (NN psk0, XKpsk3, XX) to avoid the identity tracking.
- Combine the app signing certificate with a device-bound keypair to ensure robust device integrity and app integrity, mitigating app impersonation attacks.

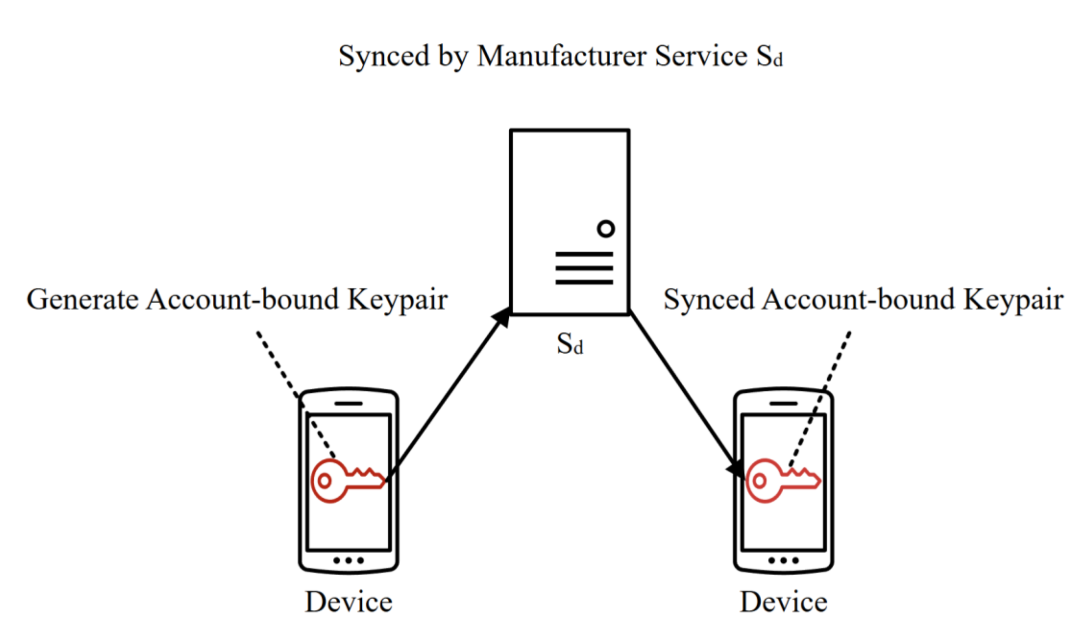
Derivate App-level Keys

The app can derive three app-level keys on the device.

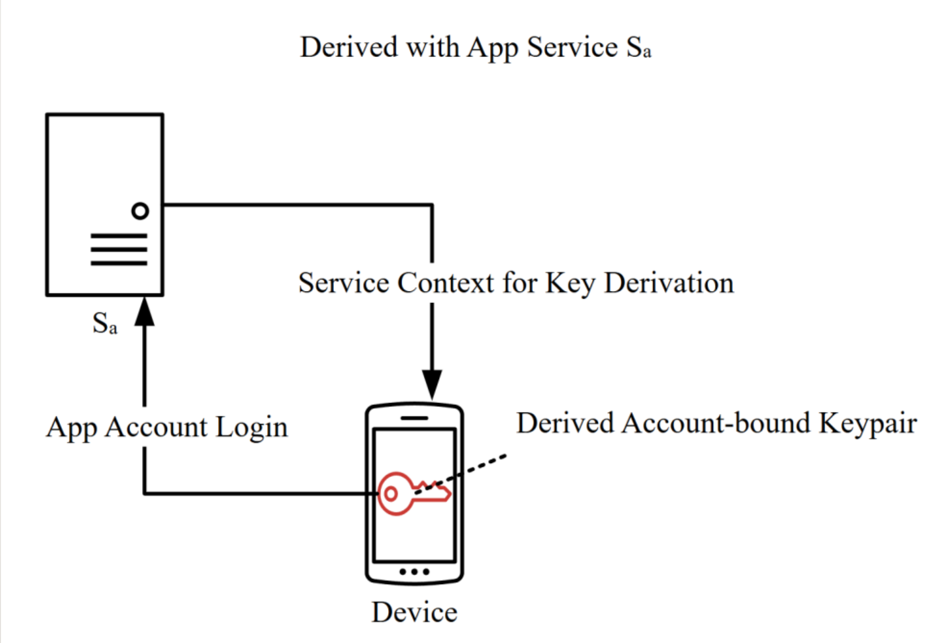


Account-bound Keypair

- App a generates the account-bound keypair $(x_u, Y_u = x_u \cdot P)$ as app public key credential.
- App a registers Y_u on S_a, associated with the user identity (UID).
- For simplicity, we assume that the account-bound keypair should be identical for the same app account on all devices.



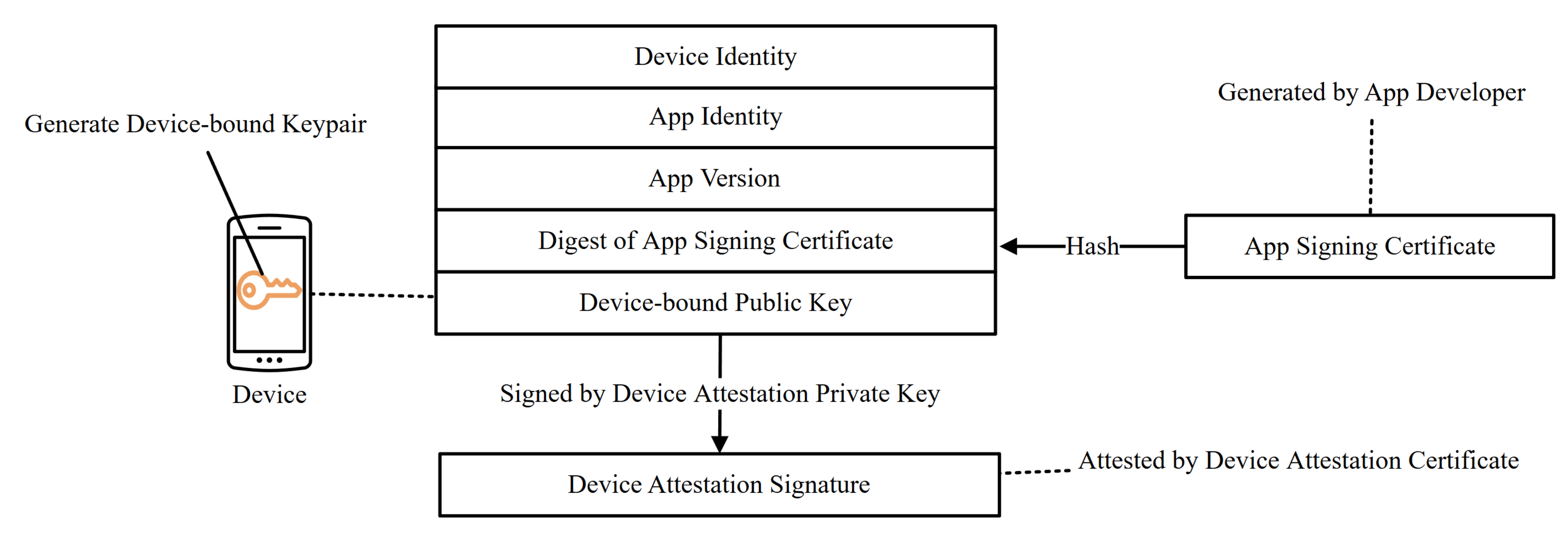
use WebAuthn, depend on passkey service to sync



use OPAQUE to derive account-bound keypair

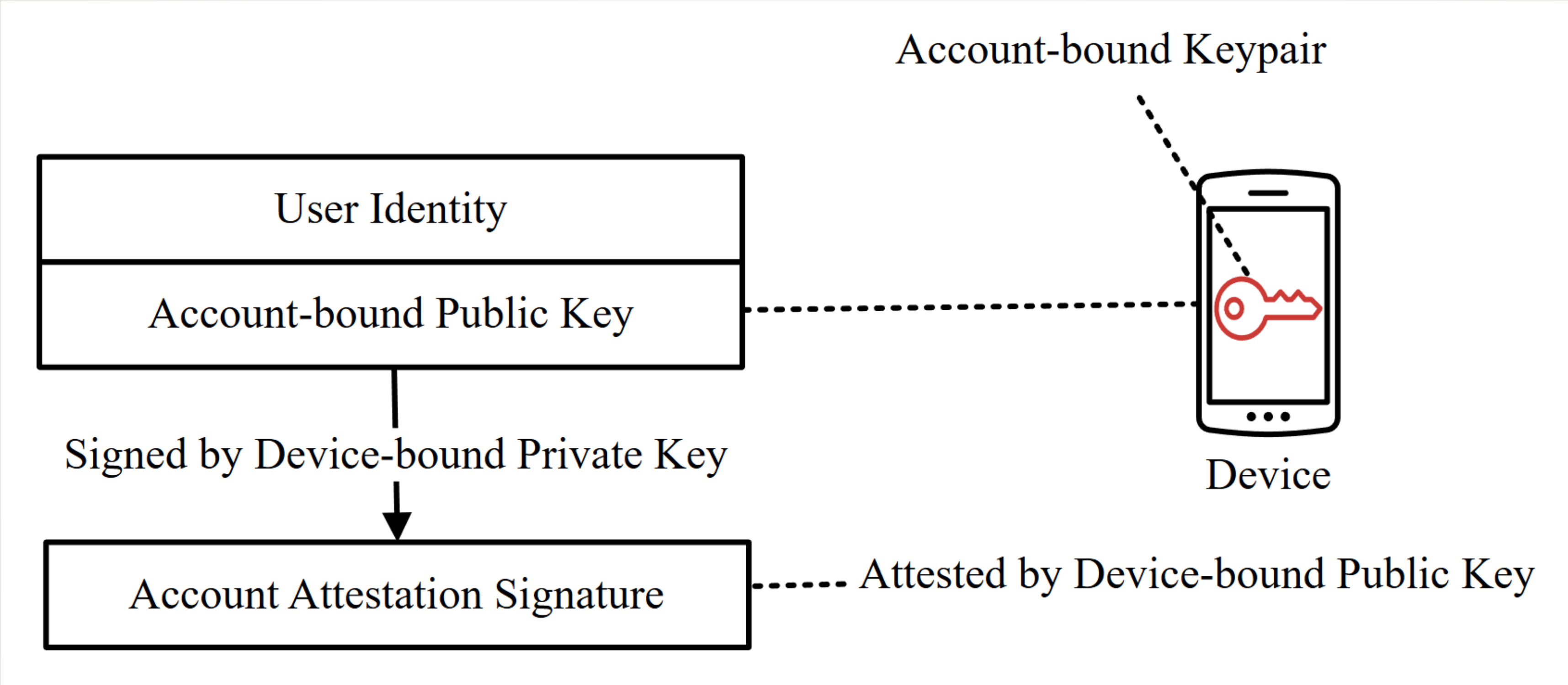
Device-bound Keypair

- Device-bound keypair (x_d, $Y_d = x_d \cdot P$) is generated for the verification of device integrity and app integrity.
- App a calls the device attestation API provided by the manufacturer to generate the device-bound keypair.



Account Attestation

- App a prepares the account attestation content (UATTct): UID and Yu.
- App a uses xd to sign the digest of UATTct, gets the account attestation signature (UATTsig).



Account Attestation Registration

Device Integrity:

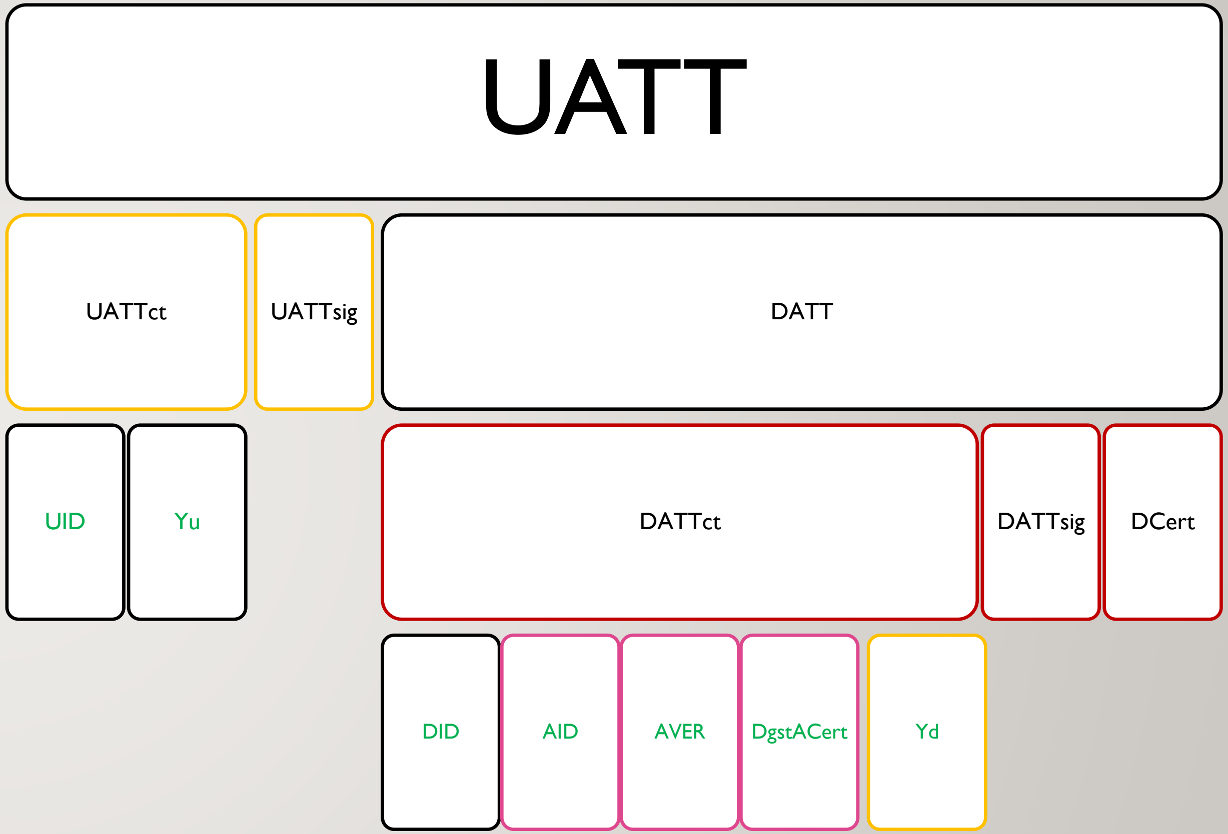
- Sa validates device attestation certificate (DCert) following the trusted certificate chain.
- Sa extracts the device attestation public key (Ydatt) from DCert.

App Integrity:

- Sa uses Ydatt to verify DATTsig on DATTct.
- Sa extracts (AID, AVER) from DATTct, gets the corresponding app signing certificate (ACert) of (AID, AVER) from a trusted-party, and calculates its digest (DgstACert2).
- Sa will end the process if DgstACert2 is not identical to DgstACert.

Account Attestation Verification:

- Sa extracts Yd from DATTct, and uses Yd to verify UATTsig on UATTct.
- Sa will end the process if the UID is not identical to the UID of current logined app account or Yu is not identical to the registered account-bound public key of UID.
- Sa securely stores (UID, DID, AID, AVER, Yd, Yu, DgstACert).



App-level Key Derivation: Self-account pre-shared key

- The self-account pre-shared key (kupsk) is used for cross-device communication between all devices of the same app account.

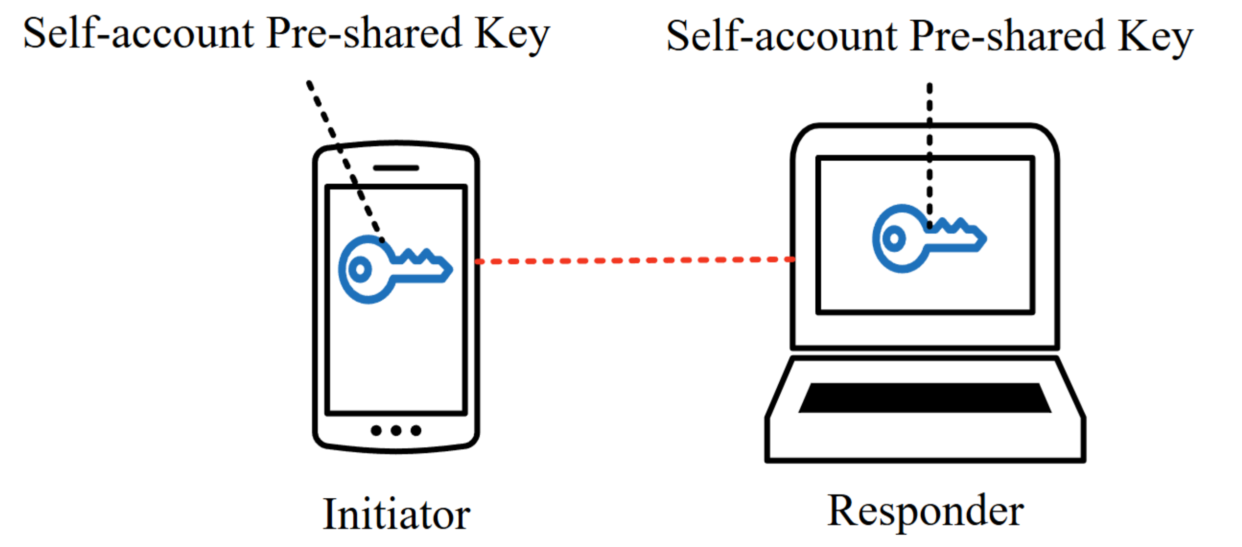


Table I
CKDUPSK

$$
\begin{array}{l}
CKDupsk(x_u,\ UID,\ AID,\ DgstACert) \\
\hline
\{ \\
\quad i \leftarrow HASH(UID \,||\, AID) \\
\quad salt \leftarrow 0x00 \,||\, ser(x_u) \,||\, ser(i) \\
\quad info \leftarrow \text{`}CKDupsk\text{'} \\
\quad k_{upsk} \leftarrow HKDF(DgstACert,\ salt,\ info,\ n) \\
\quad return\ k_{upsk} \\
\}
\end{array}
$$

App-level Key Derivation: Self-account device keypair.

- The self-account device keypair ($x_{ud}, Y_{ud} = x_{ud} \cdot P$) is used for the cross-device communication with a specific device (DID) of the same app account.

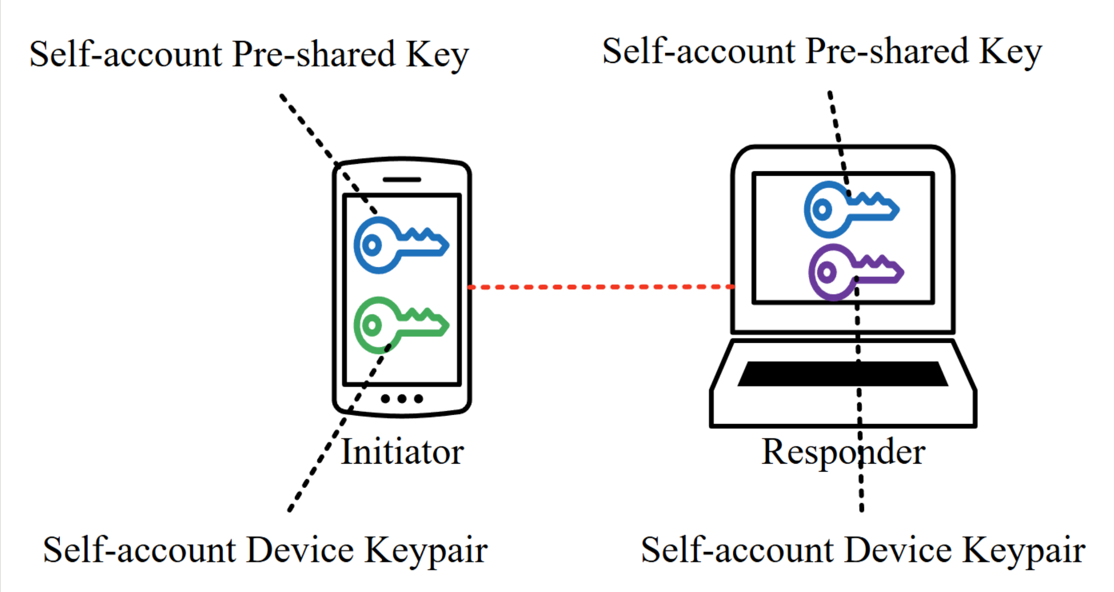


Table II
CKDUD

$CKDud(x_u,\ UID,\ AID,\ DgstACert,\ x_d,\ Y_d)$

$\{$
$\quad i \leftarrow HASH(UID \mathbin{||} AID)$
$StartUD:$
$\quad I \leftarrow HMAC(DgstACert,\ 0x00 \mathbin{||} ser(x_u) \mathbin{||} ser(Y_d) \mathbin{||} ser(i))$
$\quad x_i \leftarrow (parse(I) + x_u + x_d)\ mod\ q$
$\quad If((parse(I)\ \geq\ q)\ or\ (x_i = 0)), then$
$\qquad i \leftarrow i + 1$
$\qquad goto\ StartUD$
$\quad else$
$\qquad x_{ud} \leftarrow x_i$
$\qquad Y_{ud} \leftarrow x_{ud} \cdot P$
$\qquad return\ (x_{ud}, Y_{ud})$
$\}$

Table IV
CKDUDPUB

$CKDudPub(x_u,\ UID,\ AID,\ DgstACert,\ Y_u,\ Y_{dj})$

$\{$
$\quad i \leftarrow HASH(UID \mathbin{||} AID)$
$StartUDPub:$
$\quad I \leftarrow HMAC(DgstACert,\ 0x00 \mathbin{||} ser(x_u) \mathbin{||} ser(Y_{dj}) \mathbin{||} ser(i))$
$\quad Y_i \leftarrow point(parse(I)) + Y_u + Y_{dj}$
$\quad If((parse(I)\ \geq\ q)\ or\ (Y_i\ is\ the\ point\ at\ infinity)), then$
$\qquad i \leftarrow i + 1$
$\qquad goto\ StartUDPub$
$\quad else$
$\qquad Y_{udj} \leftarrow Y_i$
$\qquad return\ Y_{udj}$
$\}$

App-level Key Derivation: Cross-account keypair.

- The cross-account keypair (x_{uc}, $Y_{uc} = x_{uc} \cdot P$) is used for the cross-device communication with the device of another app account.

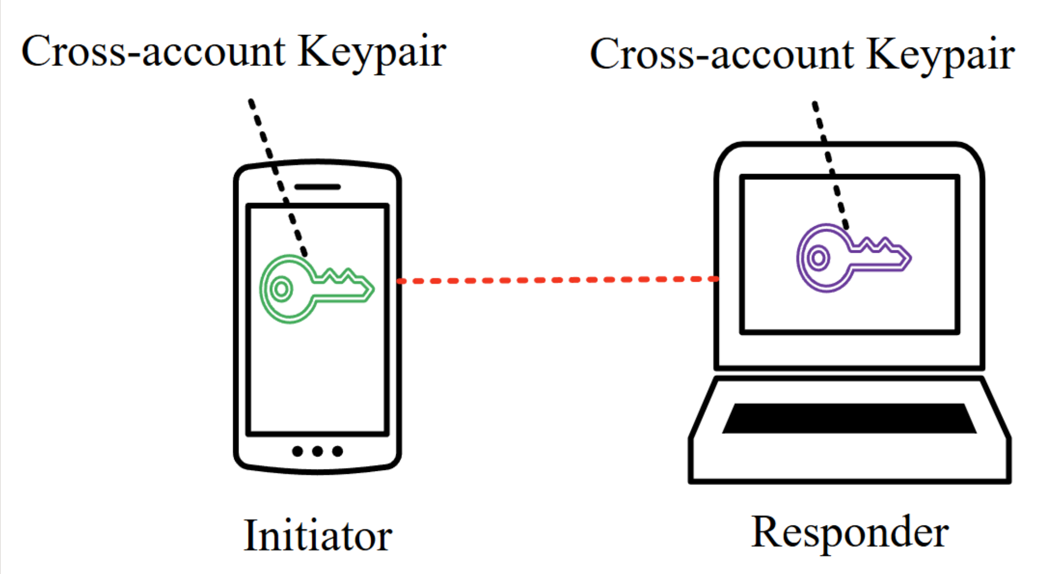


Table III
CKDuc

$CKDuc(x_u,\ UID,\ AID,\ DgstACert,\ Y_u)$

$\{$
$\quad i \leftarrow HASH(UID \parallel AID)$
$StartUC:$
$\quad I \leftarrow HMAC(DgstACert,\ 0x00 \parallel ser(Y_u) \parallel ser(i))$
$\quad x_i \leftarrow (parse(I) + x_u)\ mod\ q$
$\quad If((parse(I)\ \geq\ q)\ or\ (x_i = 0)), then$
$\qquad i \leftarrow i + 1$
$\qquad goto\ StartUC$
$\quad else$
$\qquad x_{uc} \leftarrow x_i$
$\qquad Y_{uc} \leftarrow x_{uc} \cdot P$
$\qquad return\ (x_{uc}, Y_{uc})$
$\}$

Table V
CKDucPub

$CKDucPub(UID,\ AID,\ DgstACert,\ Y_{uj})$

$\{$
$\quad i \leftarrow HASH(UID \parallel AID)$
$StartUCPub:$
$\quad I \leftarrow HMAC(DgstACert,\ 0x00 \parallel ser(Y_{uj}) \parallel ser(i))$
$\quad Y_i \leftarrow point(parse(I)) + Y_{uj}$
$\quad If((parse(I)\ \geq\ q)\ or\ (Y_i\ is\ the\ point\ at\ infinity)), then$
$\qquad i \leftarrow i + 1$
$\qquad goto\ StartUCPub$
$\quad else$
$\qquad Y_{ucj} \leftarrow Y_i$
$\qquad return\ Y_{ucj}$
$\}$

Communication

Communication Channel Isolation

https://noiseexplorer.com/patterns/

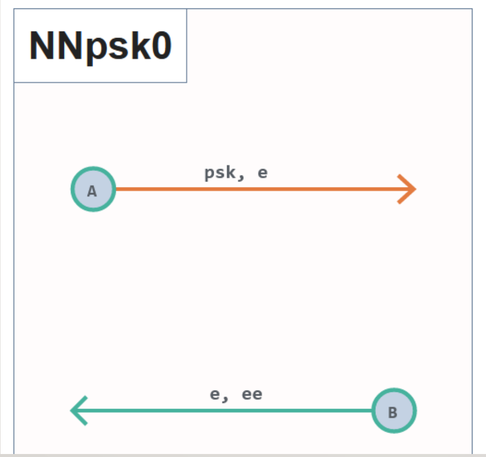
- Self-account cross-device communication.
  - use Kupsk.
  - Noise NNpsk0, similar to TLS 1.3 psk with (EC)DHE key establishment.

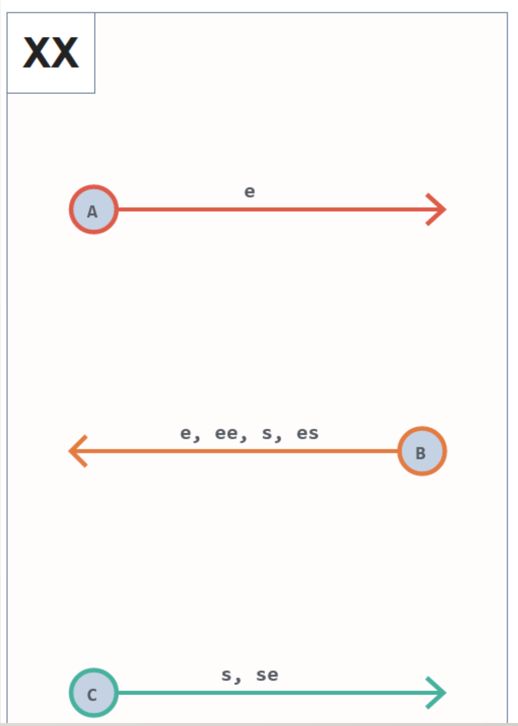
- Self-account cross-device communication with specific device.
  - Use Kupsk and self-account device keypair.
  - Noise XKpsk3, similar to TLS 1.3 extension for certificate-based authentication with an external pre-shared key.

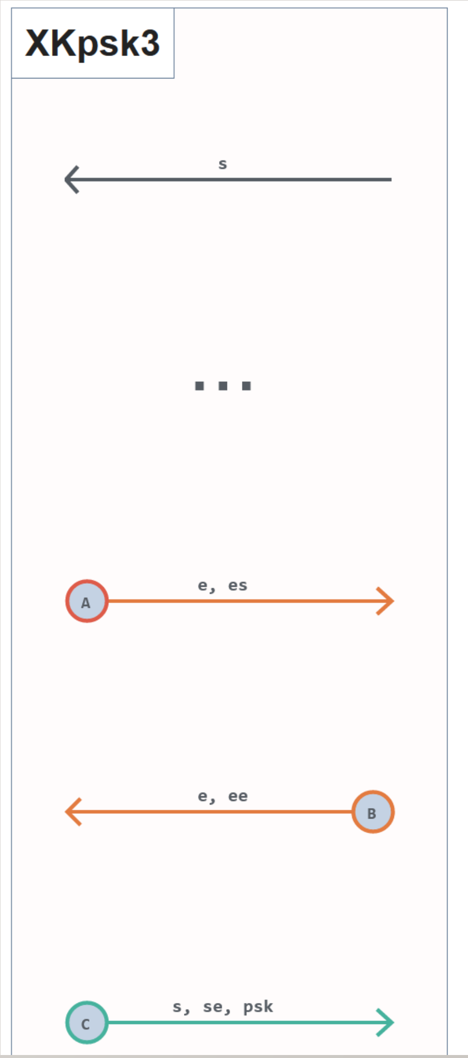
- Cross-account cross-device communication.
  - Use cross-account keypair.
  - Noise XX, similar to TLS 1.3 mutual authentication.

- Forward Secrecy:
  - DH calculation (ee)

- Identity Hiding:
  - long-term public key (s) transmits after DH calculation (ee), encrypted by the handshake key (k).







Evaluation

- We performed the evaluation for the key derivation and communication on a 64-bit Allwinner H616 ARM Cortex-A53 microprocessor (4-Core, 4-Thread, 1.5 GHz).
- Since Noise handshake messages are shorter than TLS and UKEY2, our scheme has the minimum communication payload size and CPU time of the three schemes.

Table VI
PARAMETERS SELECTED FOR EVALUATION

Scheme	Curve	HASH	KDF	Cipher
Apple Continuity (TLS)	P-256	SHA256	HKDF	AES-128-GCM
Google Nearby (UKEY2)	P-256	SHA256	HKDF	AES-256-CBC
Our Scheme	P-256	SHA256	HKDF	AES-256-GCM

Table VII
KEY DERIVATION: CPU TIME (MILLISECONDS)

CKDupsk	CKDud	CKDudPub	CKDuc	CKDucPub
277.1	277.0	280.0	277.0	278.7

Table VIII
COMMUNICATION: PAYLOAD SIZE(KILOBYTES)

	Plaintext Message Length ($\|m\|$)			
Scheme	**1KB**	**10KB**	**100KB**	**500KB**
Apple Continuity (TLS)	7.06	25.06	206.84	1011.02
Google Nearby (UKEY2)	2.49	20.49	200.50	1000.50
Our Scheme	2.18	20.18	200.18	1000.18

Table IX
COMMUNICATION: CPU TIME (SECONDS)

	Plaintext Message Length ($\|m\|$)			
Scheme	**1KB**	**10KB**	**100KB**	**500KB**
Apple Continuity (TLS)	4.26	4.27	4.36	4.85
Google Nearby (UKEY2)	1.74	1.79	1.97	2.58
Our Scheme	1.73	1.76	1.94	2.34

Conclusion

- Our Work
  - We propose an app-level secure cross-device communication scheme, supporting cross-manufacturer scenarios and defending against app impersonation attacks and cross-device eavesdropping.
  - Regarding account authentication, our scheme is based on the app account, Apple Continuity and Google Nearby are based on the manufacturer account.
  - We introduce a derivation method for app-level keys and separate communication channels through the three types of keys for cross-account and self-account usage.
  - We carefully choose Noise handshake patterns to avoid identity tracking.
- Limitation
  - We don't mention the app account management, permission access control on the app, and device pairing.
  - We don't discuss the wireless communication channels below the data layer, such as BLE and Wi-Fi.
  - We don't divide into the cryptographic improvement on cryptographic functions and protocols, using standard BIP-0032 and Noise.
  - We don't make the formal verification on our scheme, following the security analysis presented by Noise Explorer.
- Future Work
  - Do more evaluation on our scheme, and deploy it on smart devices in the future.

THANK YOU

Q&A