Key Management Based on Ownership of Multiple Authenticators in Public Key Authentication

Kodai Hatakeyama, Daisuke Kotani and Yasuo Okabe

Kyoto University, Japan

[Background] Public Key Authentication (PKA)

- Is an alternative or complement way of password authentication
- Before authentication, a user has registered her account with a service
  - The user manages a private key (also called secret key)
  - The service manages the corresponding public key bounded to her account

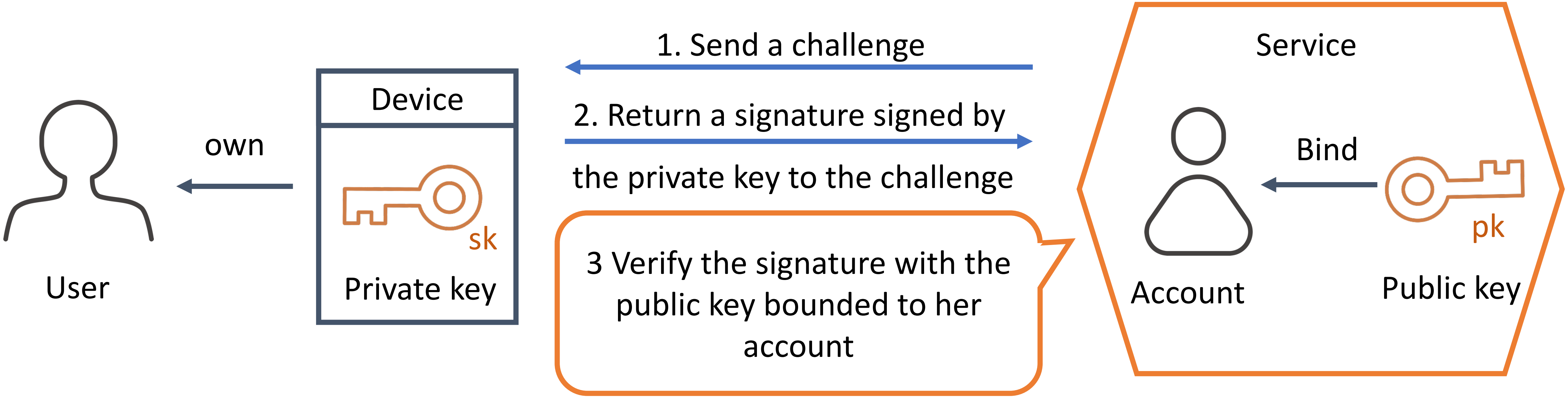
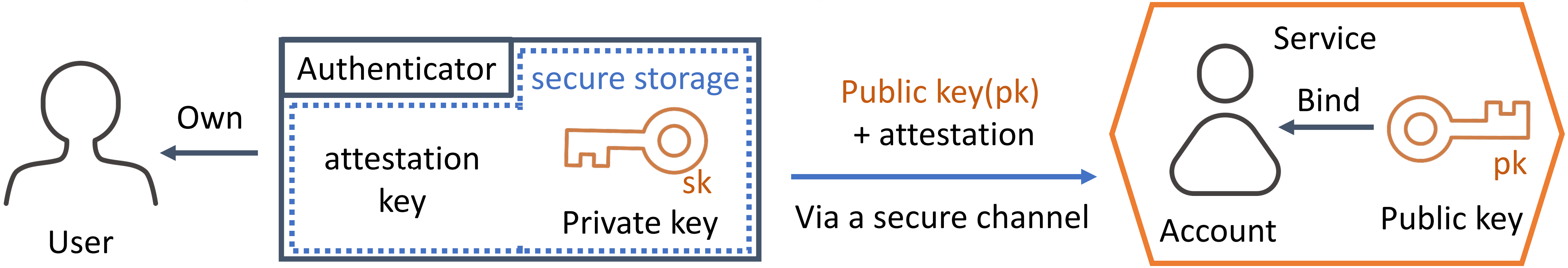


Fig: How to authenticate

[Background] Key Management in PKA

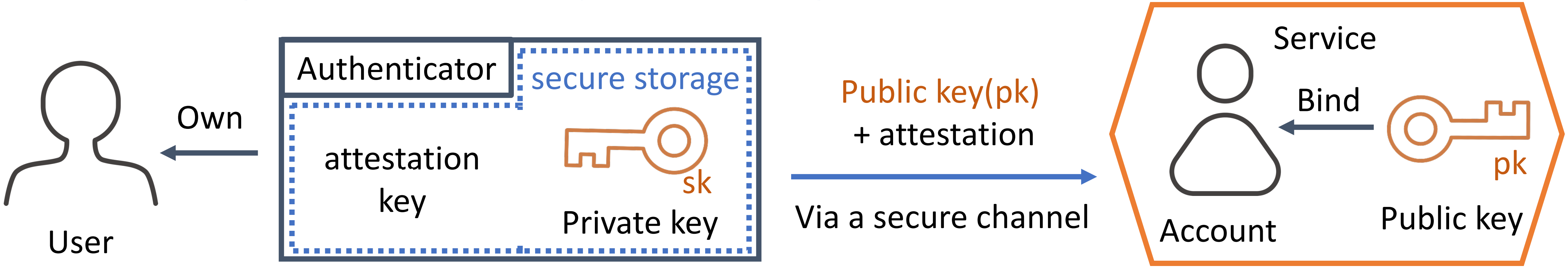


User manages private keys

Service manages public key

- On owned devices (called authenticators)
  - Smartphone, laptop, …
- Authenticators store private keys in secure storage where…
  - private keys cannot be exported
  - they require local authentication when using private keys
- Authenticators can send an attestation about public keys to be registered
  - Attestations are signed by the attestation key embedded by its manufacturer

[Background] Key Management in PKA



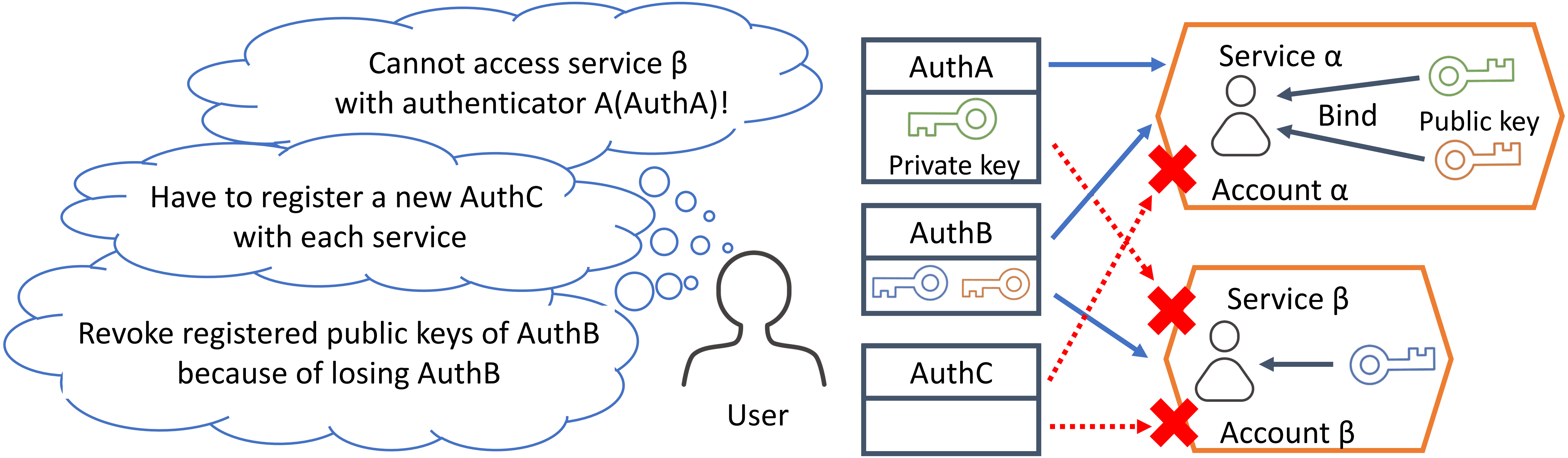
User manages private keys

Service manages public key

- With binding each public key to legitimate account
- Services bind a public key
  - When the user send it via trusted communication
    - e.g.) communication during account registration
  - When the attestation is successfully verified
    - They verify that the corresponding private key is securely stored

[Problem] Multiple Authenticators

- User can only access services with the registered authenticator
  - Because authenticators cannot export private keys to another authenticator



[The Big burden of user]
To manage public keys of each authenticator according to its lifecycle

Contribution

Proposal

The mechanism where users and services manage public keys based on the owner of authenticators storing the corresponding private keys.

- Share a secret among owned authenticators only once in advance
- Avoid collation of accounts by collusion with multiple services
- Support for updating authenticators in response to their lifecycle

Evaluation

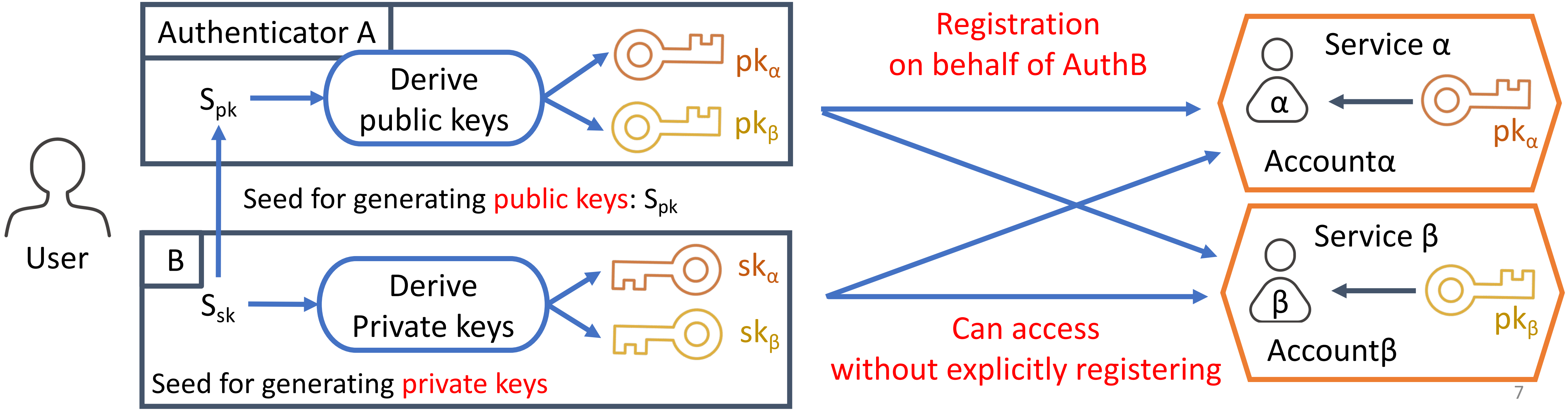
- analyze the proposal mechanism with threat modeling
- evaluate what measures our proposal takes against the found threats

[Related Work] Proxy registration of public keys

The registered authenticator generates and registers a public key of another one on behalf of it.

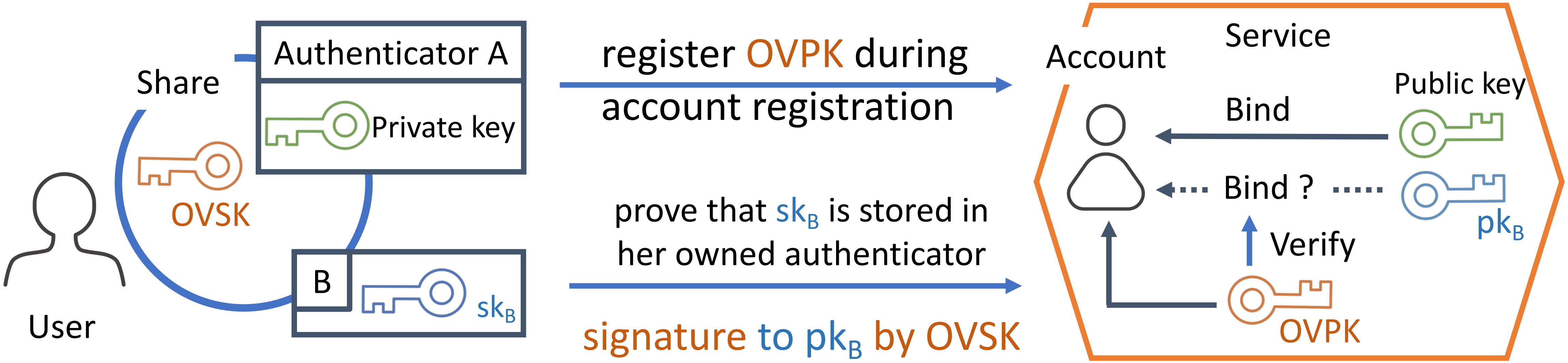
Problems

1. When the user has three authenticators
2. Support for updating authenticators in response to their lifecycle



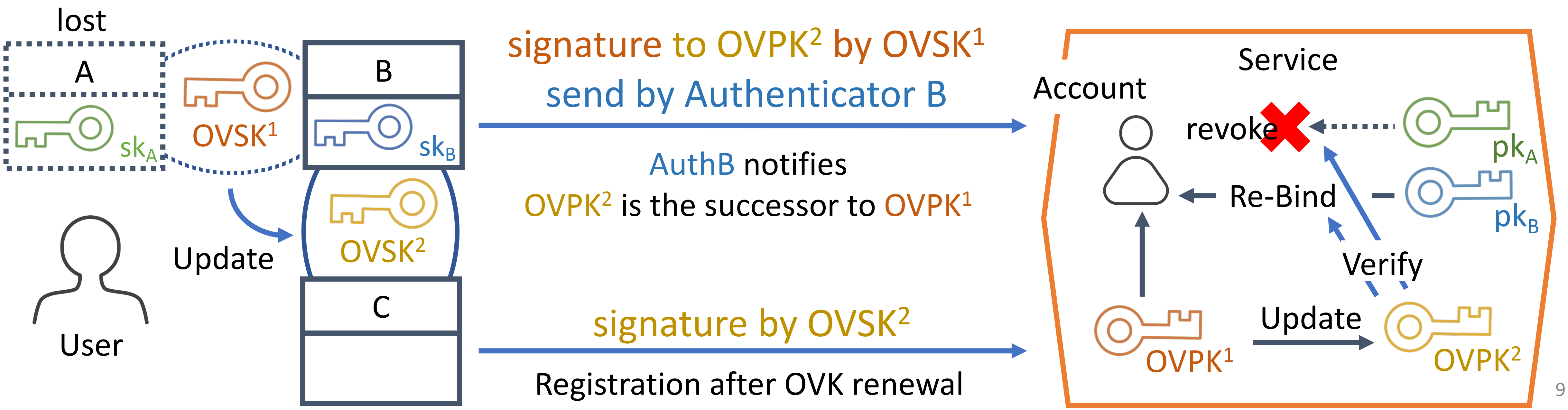
[Proposal] Overview(1/2)

- Introduce a cryptographic key pair called an Ownership Verification Key (OVK)
  - The private key of the OVK (OVSK): shared among all authenticators owned by a user
  - The public key of the OVK (OVPK): bounded to her account and managed by a service
- By using OVSK/OVPK
  - a user can prove the ownership of authenticators
  - a service can verify whether the public key to be registered is generated on her authenticators



[Proposal] Overview(2/2)

- Updates OVK when a user changes a set of her authenticators
- A user shares a new $OVSK^2$ among all owned authenticators including a new one
- Registered authenticators notify services of updating the $OVPK^2$.
  - the message contains the new $OVPK^2$ signed by the previous $OVSK^1$.
- Services bind a new OVPK to her account based on the most trustworthy message



[Proposal] Technical details

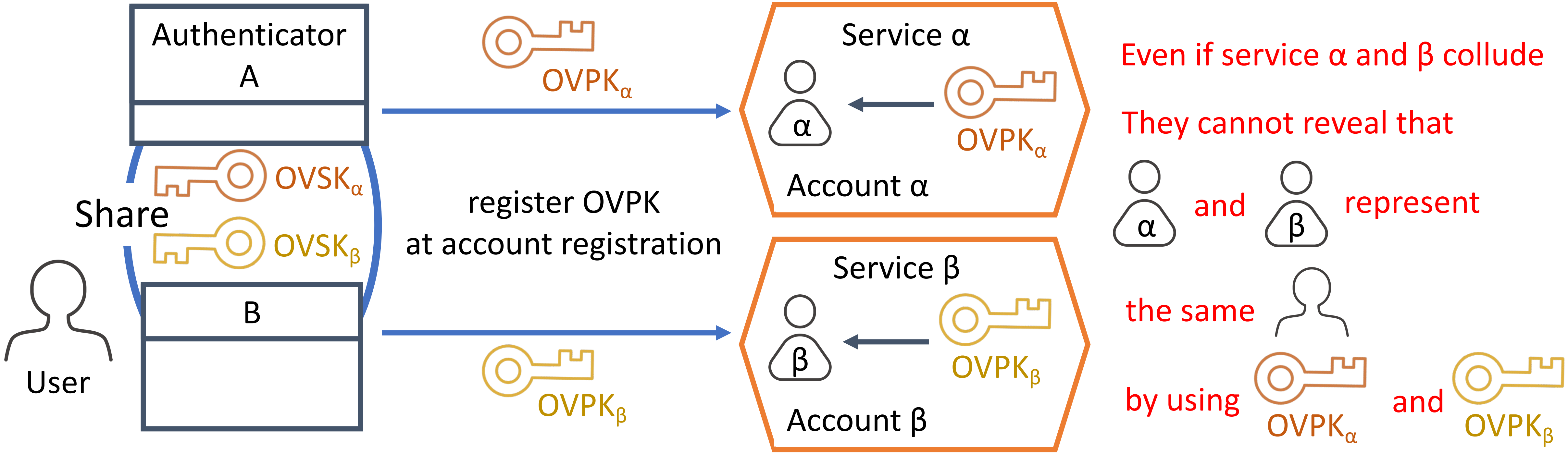
- How to derive the same OVK among authenticators for each service.
  - [P1] Sharing a seed among authenticators
  - [P2] Deriving the same OVSK among all authenticators from a pre-shared secret
- How to update a new OVK in authenticators and services
  - [P1] Re-sharing a new seed among authenticators
  - [P3] Updating an OVPK registered with services

Explain the proposals marked in red (P2 and P3)

[P2] Deriving an OVSK from a pre-shared secret

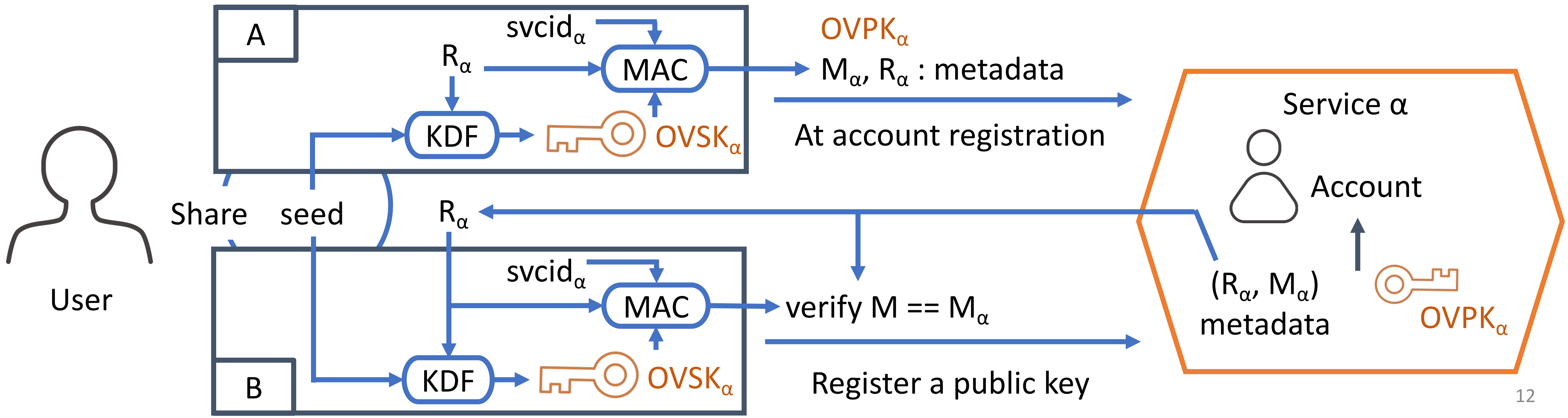
Requirement Register different OVPKs with each service

Reason To prevent services from correlating their accounts by using OVPKs

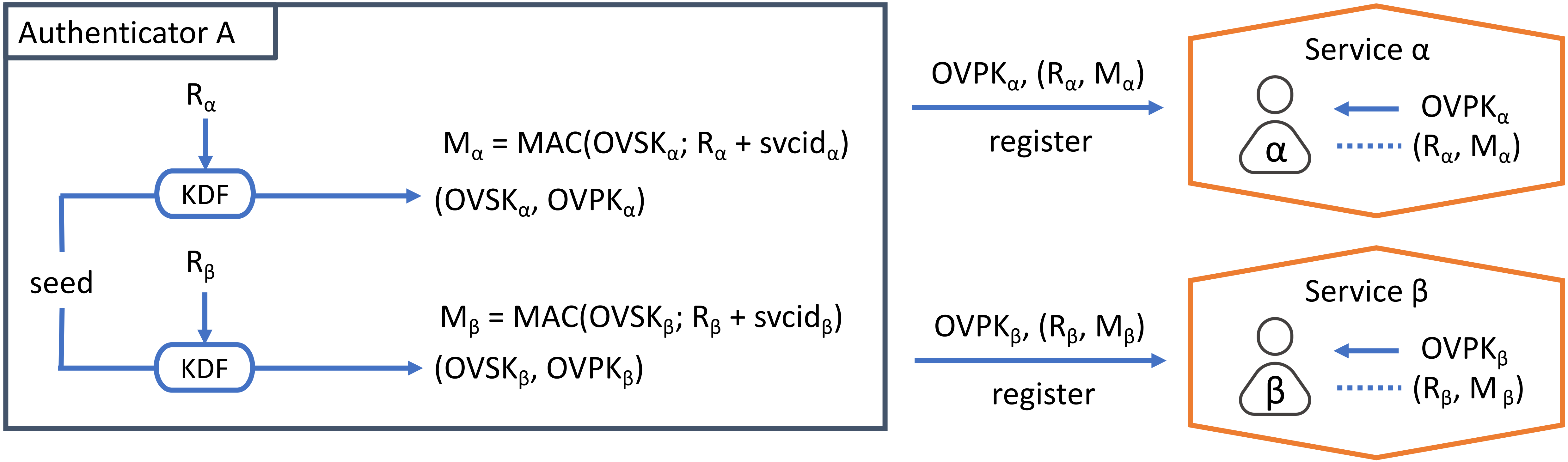


[P2] Deriving an OVSK from a pre-shared secret

- Authenticators agree in advance on the following parameters
  - seed: the secret shared among authenticators, $svcid_\alpha$: identifier of Service α
  - KDF: the key derivation function: input = (seed, R_α) and output = $OVSK_\alpha$
  - MAC: the message authentication code function: key = $OVSK_\alpha$
- Authenticator A registers OVPK and metadata (R_α, M_α) with Service α
  - R_α : service α stores the random value and provides it to other authenticators
  - M_α : other authenticators verify the received mac value is for R_α and service α



[P2] Unique OVKs per Service



$R_\alpha \neq R_\beta$ ⇒ $OVPK_\alpha \neq OVPK_\beta$

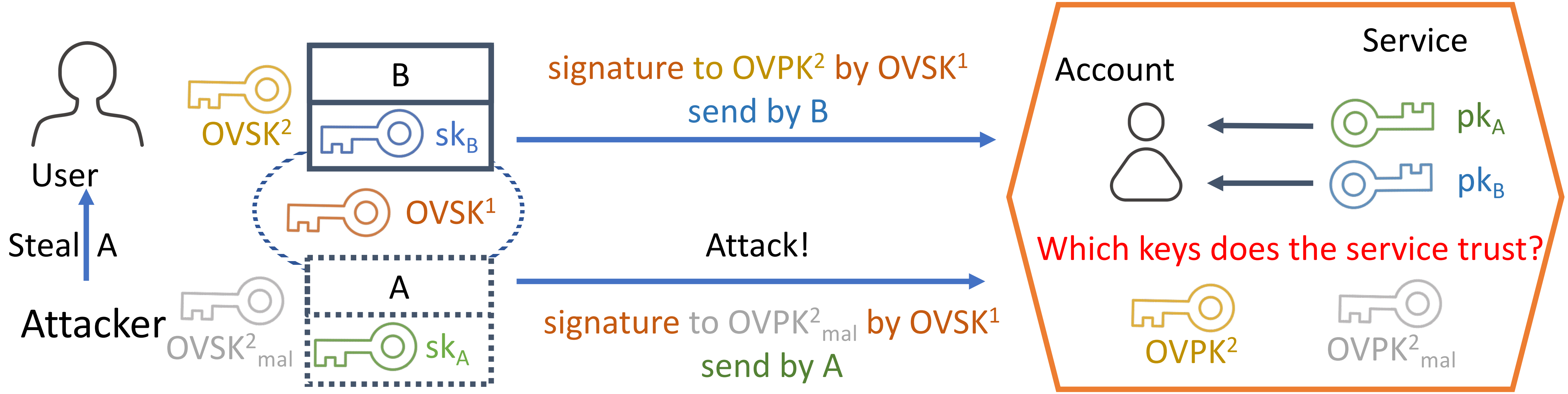
services cannot correlate their accounts by using registered OVPKs

[P3] Updating an OVK

Goal — $OVPK^{n+1}$ inherits as much trustworthiness of $OVPK^{n}$ as possible

Method — Registered authenticators send the updating message containing the legitimate $OVPK^{n+1}$ signed by the previous $OVSK^{n}$.

Problem — Attackers try to update a malicious $OVPK^{n+1}_{mal}$



[P3] Evaluating the Trustworthiness of an Updating Message

contains a candidate OVPK signed by the registered OVSK

Assumption

- The trustworthiness of all registered authenticators is equal.
  - It is difficult for a service to determine whether an authenticator is stolen or not.
- It takes time for an attacker to gain control of a stolen authenticator

Method

- If the same updating message comes from more than half of authenticators
  - the service trusts the message
- Otherwise, the service trusts
  - the updating message sent from the most authenticators
  - the earliest received message if more than one the most trustworthiness messages

[Evaluation] Threat modeling

- We evaluated our proposal mechanism by using threat modeling
- We confirmed that our proposal achieves some security goals such as
  - [SG-2] preventing correlation of accounts and
  - [SG-3] correctly binding public keys to accounts.
- We discussed how our proposal mitigates threats for which measures are not sufficient.

[Evaluation] Threat Analysis Example

Threat	Homograph Mis-Registration
Scenario	A malicious service • pretends legitimate services and sends metadata stolen from the services. • prompts the user to register a new public key The malicious service correlates OVPKs by whether the user requests a public key registration or not
Violation	SG-2: Services cannot correlate their accounts
Address	Authenticators verify the MAC value of the received metadata including the identifier of the service that they communicate

Conclusion

Proposal

Introduce a key pair called Ownership Verification Key (OVK)

The mechanism where users and services manage public keys based on the owner of authenticators storing the corresponding private keys.

- A user derives OVSK on her authenticators from the pre-shared seed
- A service binds OVPK and public keys signed by an OVSK to her account.
- They update OVK in response to authenticators' lifecycle

Evaluation

- analyze the proposal mechanism with threat modeling
- evaluate what measures our proposal takes against found threats

Future Work

- formal verification of cryptographic operations
- improvement of calculating trustworthiness of update messages.