

# Matrix Cryptographic Key Infrastructure

---

Sumner Evans

21 September 2024

Beeper (Automattic)

# Why Cryptography?

Matrix uses cryptography for two main purposes:

1. **Message Security** — only the people who are part of the conversation should be allowed to view messages of the conversation.
2. **Identity** — verifying that a user or device is who they say they are.

# Why Cryptography?

Matrix uses cryptography for two main purposes:

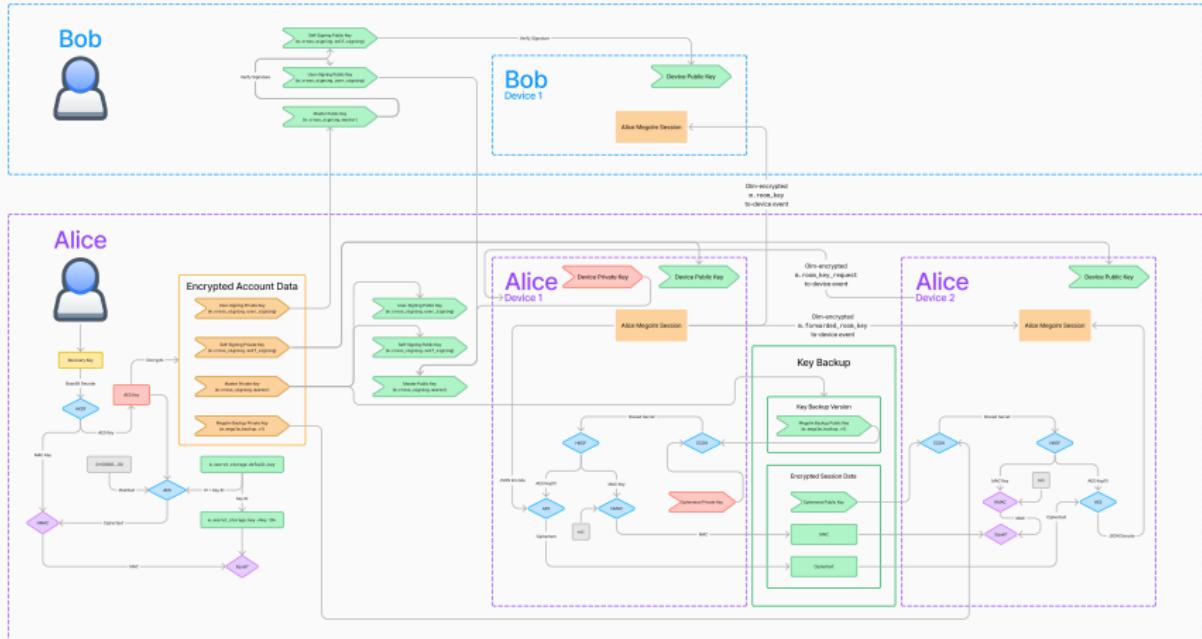
1. **Message Security** — only the people who are part of the conversation should be allowed to view messages of the conversation.
2. **Identity** — verifying that a user or device is who they say they are.

# Why Cryptography?

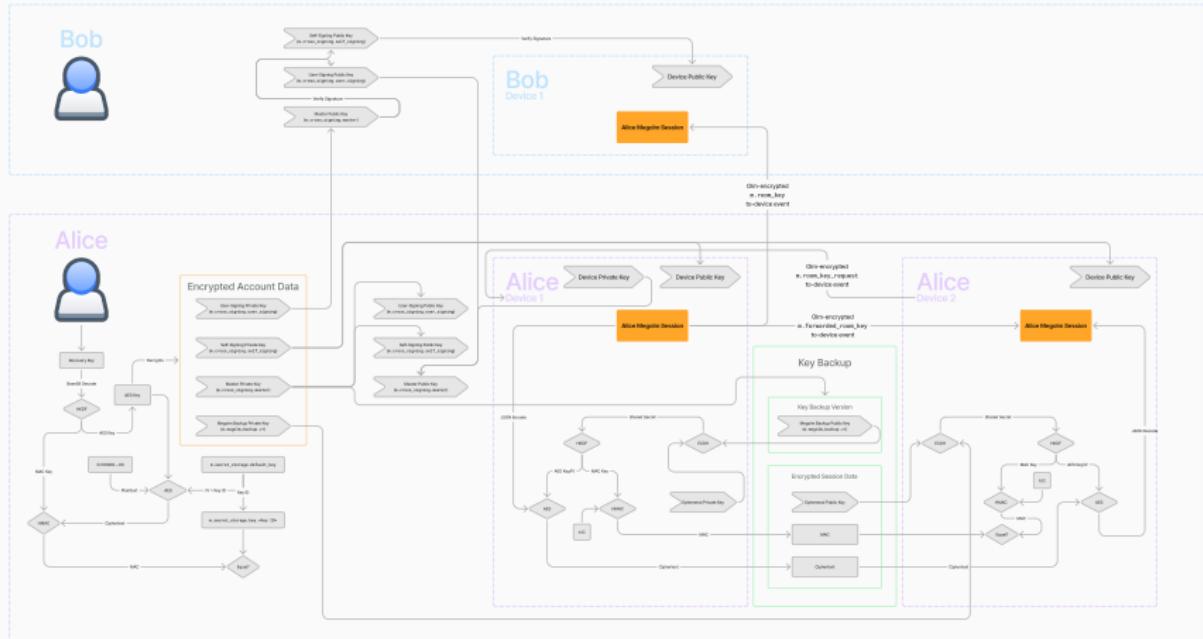
Matrix uses cryptography for two main purposes:

1. **Message Security** — only the people who are part of the conversation should be allowed to view messages of the conversation.
2. **Identity** — verifying that a user or device is who they say they are.

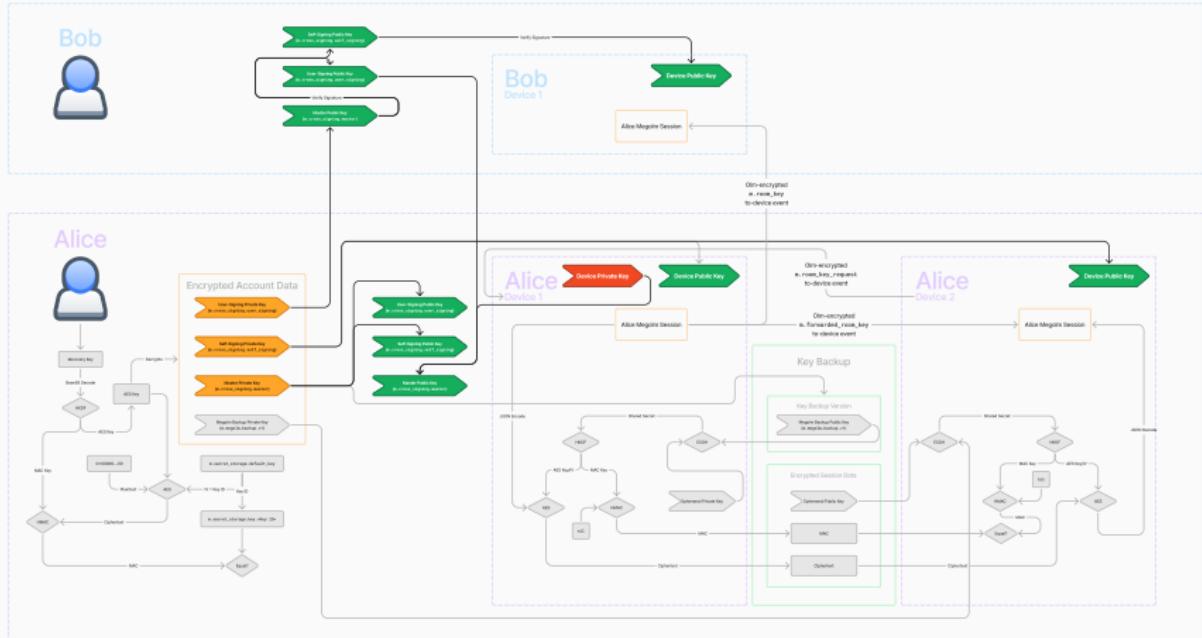
# Big Picture



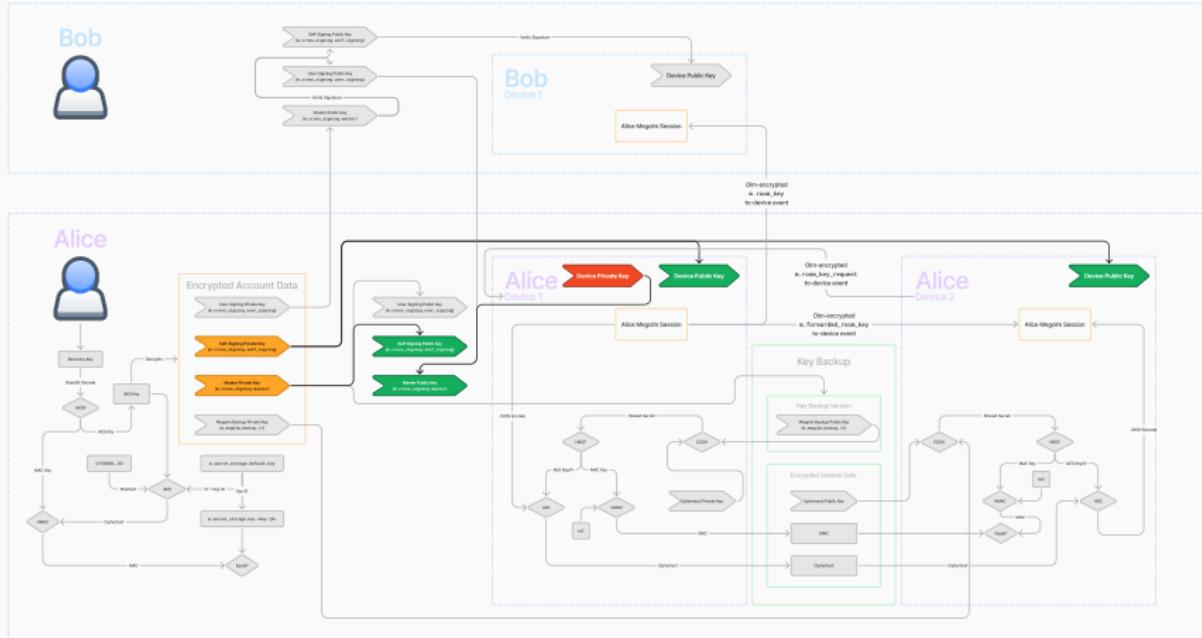
# Big Picture: Message Security



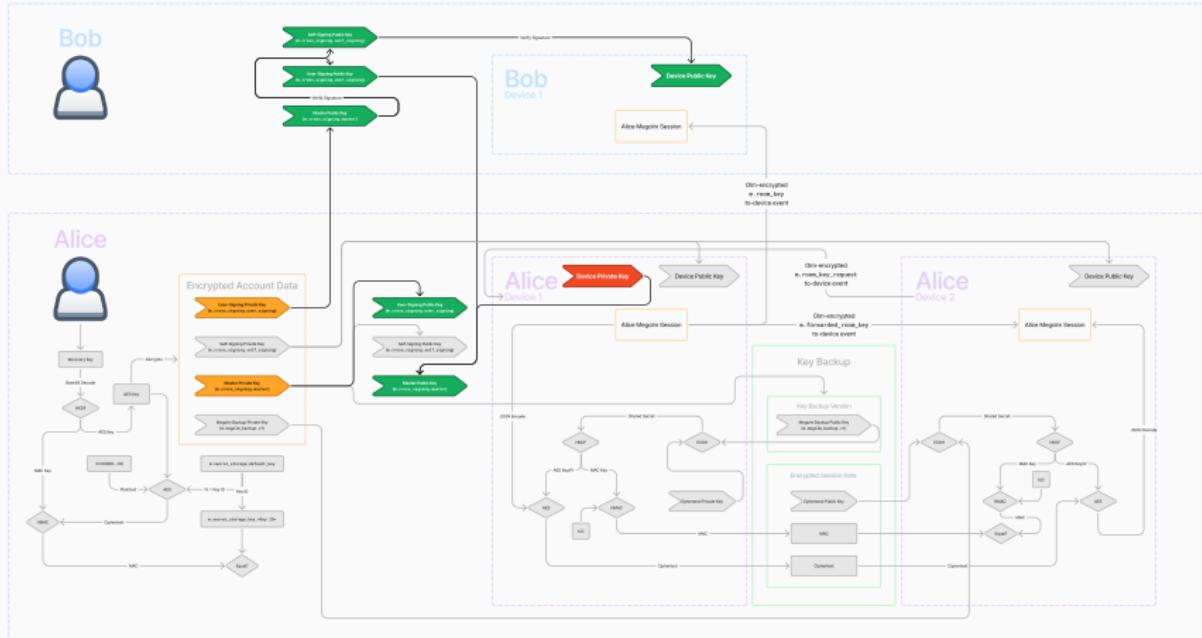
## Big Picture: Identity



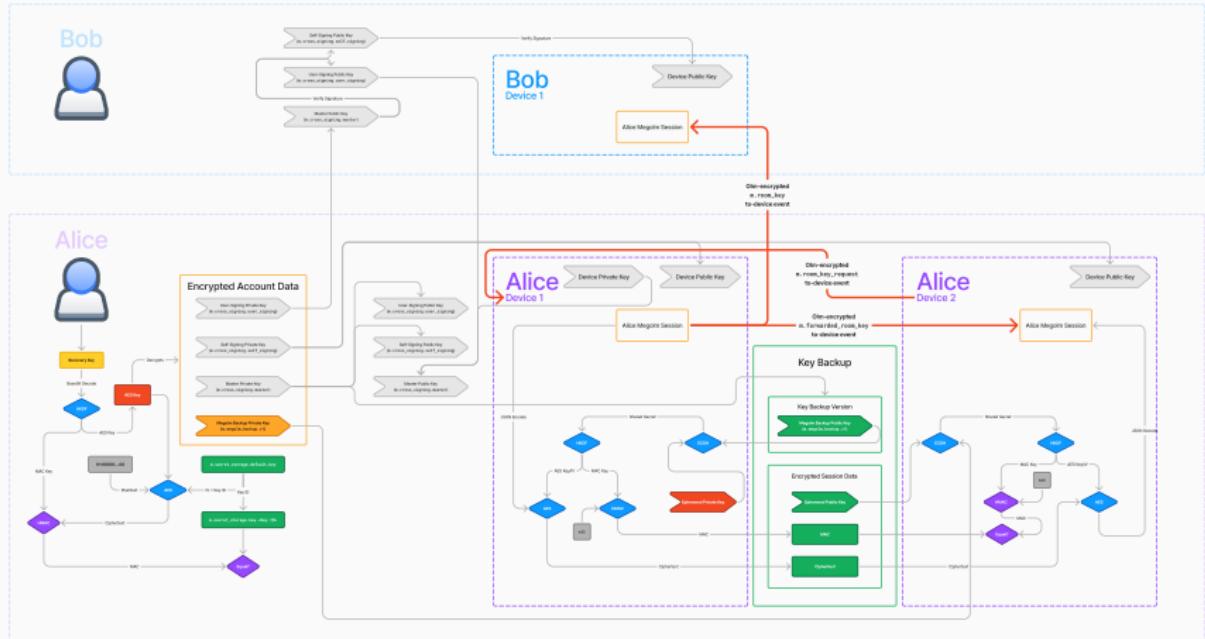
# Big Picture: Identity: Device Verification



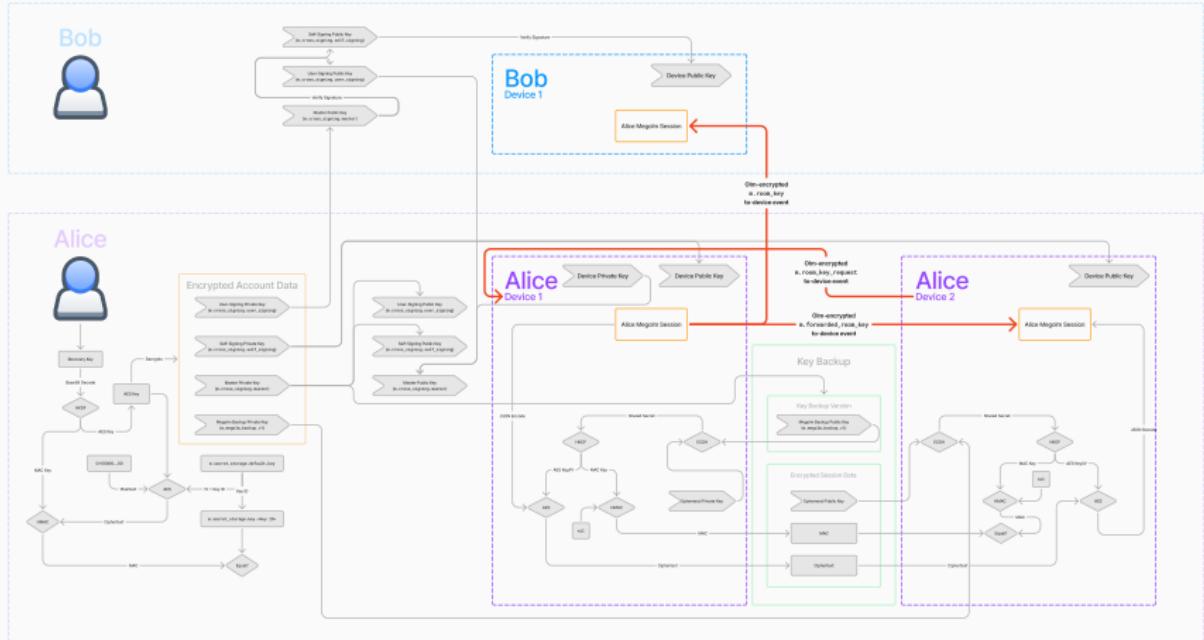
# Big Picture: Identity: User Verification



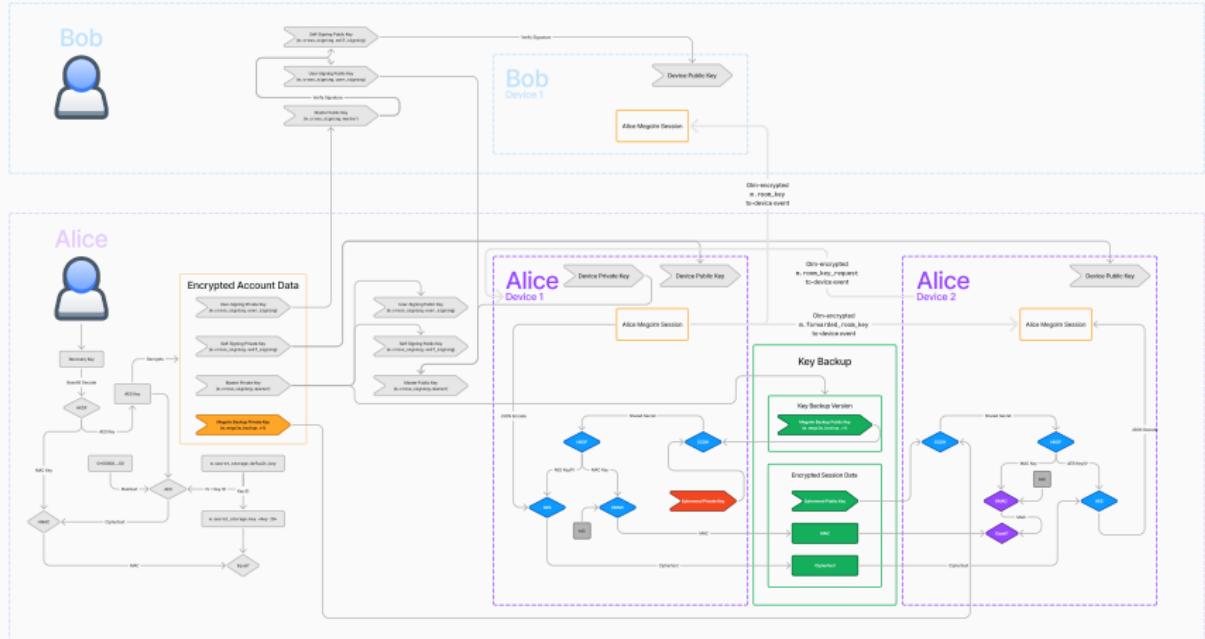
## Big Picture: The Other Stuff



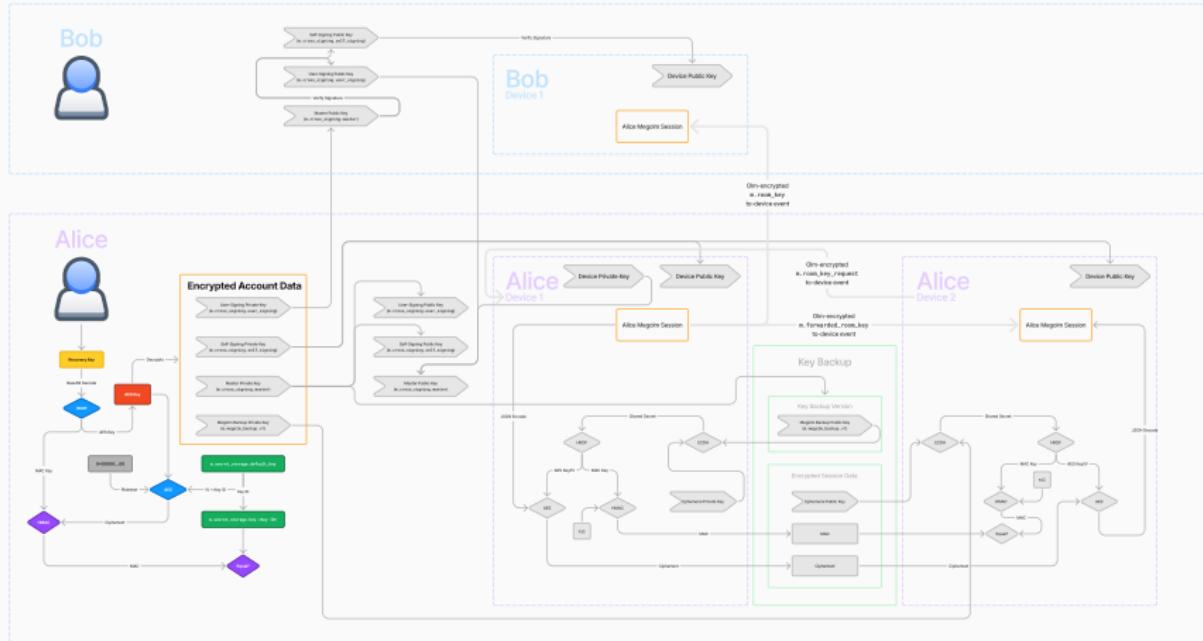
# Big Picture: The Other Stuff: To-Device



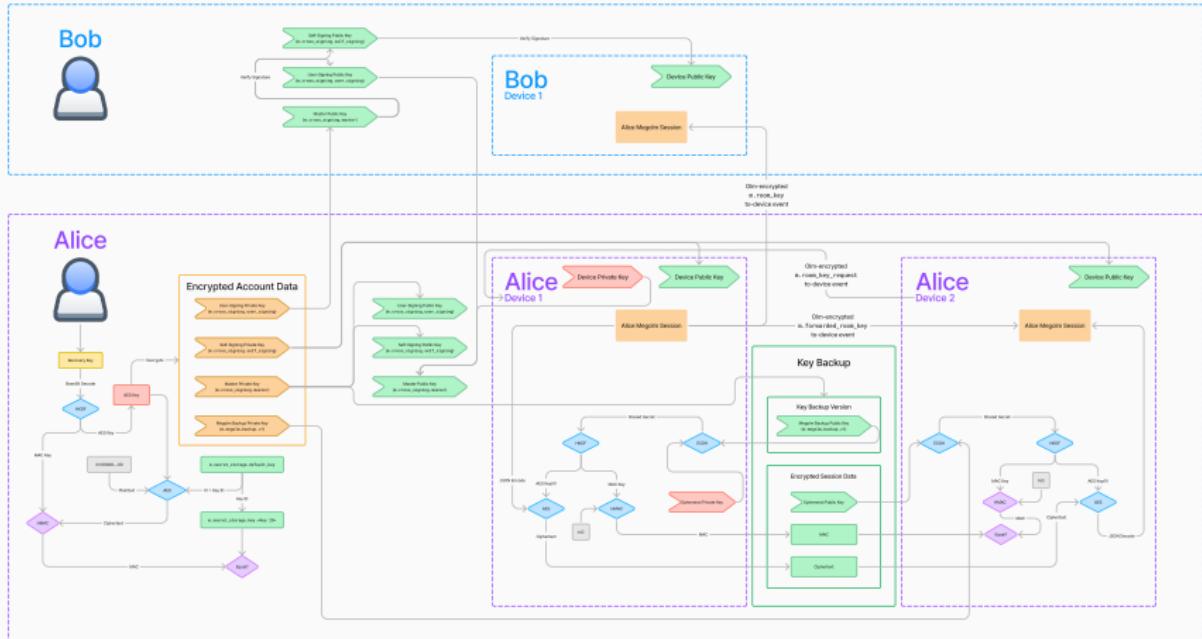
# Big Picture: The Other Stuff: Key Backup



## Big Picture: The Other Stuff: Secure Secret Storage and Sharing



# Big Picture



# Cryptography Crash Course

---

# Encryption: Symmetric vs Asymmetric

There are two main categories of encryption schemes:

- **Symmetric** — both **the encryptor and the decryptor share the same key** and that key is used in both the encryption and decryption of the message
- **Asymmetric** — the encryptor needs the public key, and the decryptor needs the private key and the encryptor encrypts the message with the public key, and the private key is required to decrypt the message

# Encryption: Symmetric vs Asymmetric

There are two main categories of encryption schemes:

- **Symmetric** — both **the encryptor and the decryptor share the same key** and that key is used in both the encryption and decryption of the message
- **Asymmetric** — **the encryptor needs the public key, and the decryptor needs the private key** and the encryptor encrypts the message with the public key, and the private key is required to decrypt the message

## Asymmetric Signatures

In addition to providing encryption, asymmetric encryption schemes also provide **signatures**.

Signing uses the *private* key, and anyone who possesses the *public* key can verify the signature.

## Asymmetric Signatures

In addition to providing encryption, asymmetric encryption schemes also provide **signatures**.

Signing uses the *private* key, and anyone who possesses the *public* key can verify the signature.

# Hashes and HMAC

A **cryptographic hash function** is a one-directional function which takes an arbitrarily large set of data and produces a unique fixed-size output (called the hash).

*Given the same data, a hash function will always return the same output.*

This allows us to verify that the data did not change in transit (for example, by a malicious actor).

Hashes are vulnerable to **metadata attacks**. To prevent these, we use HMAC which adds a secret key to the hash.

# Hashes and HMAC

A **cryptographic hash function** is a one-directional function which takes an arbitrarily large set of data and produces a unique fixed-size output (called the hash).

**Given the same data, a hash function will always return the same output.**

This allows us to verify that the data did not change in transit (for example, by a malicious actor).

Hashes are vulnerable to **metadata attacks**. To prevent these, we use HMAC which adds a secret key to the hash.

# Hashes and HMAC

A **cryptographic hash function** is a one-directional function which takes an arbitrarily large set of data and produces a unique fixed-size output (called the hash).

**Given the same data, a hash function will always return the same output.**

This allows us to verify that the data did not change in transit (for example, by a malicious actor).

Hashes are vulnerable to **metadata attacks**. To prevent these, we use HMAC which adds a secret key to the hash.

## Hashes and HMAC

A **cryptographic hash function** is a one-directional function which takes an arbitrarily large set of data and produces a unique fixed-size output (called the hash).

**Given the same data, a hash function will always return the same output.**

This allows us to verify that the data did not change in transit (for example, by a malicious actor).

Hashes are vulnerable to **metadata attacks**. To prevent these, we use HMAC which adds a secret key to the hash.

## Key-Derivation Functions (HKDF)

Sometimes, we want to turn a small key into a larger key (or set of larger keys).

**Key-Derivation Functions (KDFs)** are used to do this.

Matrix uses HKDF which uses HMAC for the key derivation process.

## Key-Derivation Functions (HKDF)

Sometimes, we want to turn a small key into a larger key (or set of larger keys).

**Key-Derivation Functions (KDFs)** are used to do this.

Matrix uses HKDF which uses HMAC for the key derivation process.

# Diffie-Hellman Key Exchanges

Often, we need a way to share keys with both the sending and receiving parties across an unsecured channel.

**Diffie-Hellman** is a method for using public-key cryptography to facilitate keysharing.

$$\mathbf{ECDH}(A_{private}, B_{public}) = \mathbf{ECDH}(B_{private}, A_{public}) = K_{shared}.$$

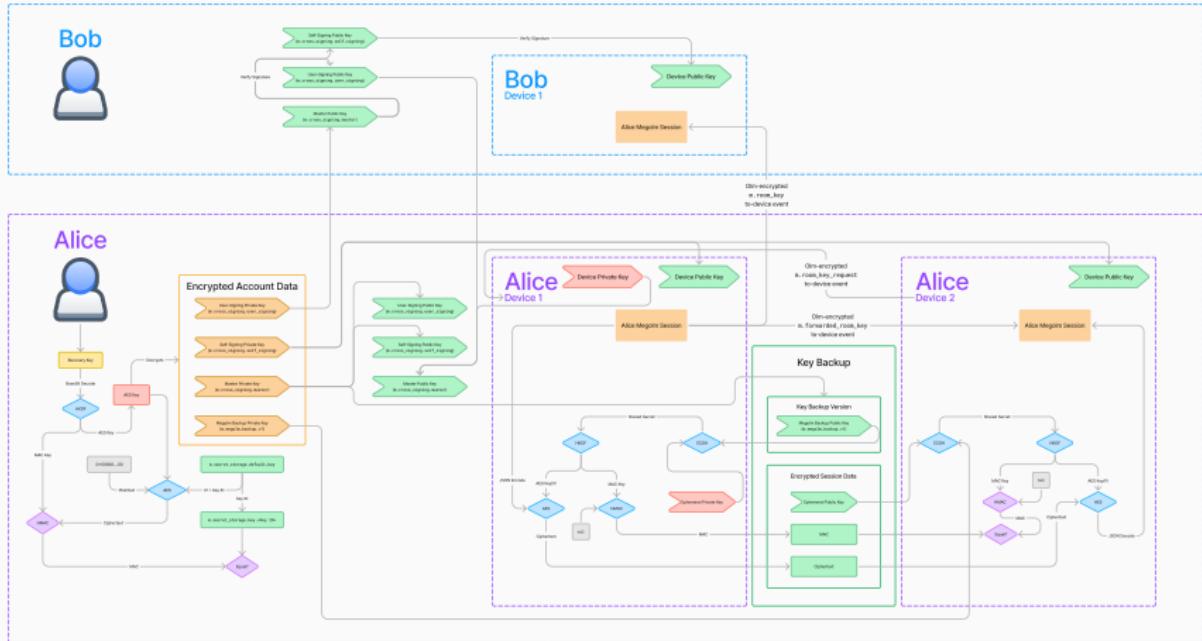
# Diffie-Hellman Key Exchanges

Often, we need a way to share keys with both the sending and receiving parties across an unsecured channel.

**Diffie-Hellman** is a method for using public-key cryptography to facilitate keysharing.

$$\mathbf{ECDH}(A_{private}, B_{public}) = \mathbf{ECDH}(B_{private}, A_{public}) = K_{shared}.$$

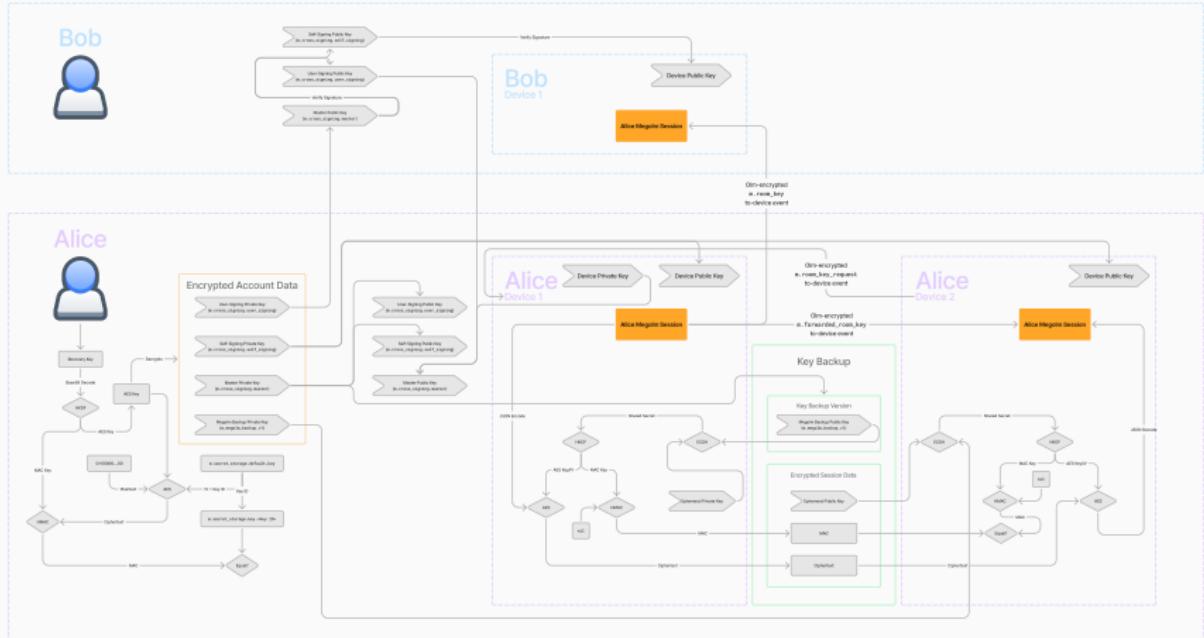
# Big Picture



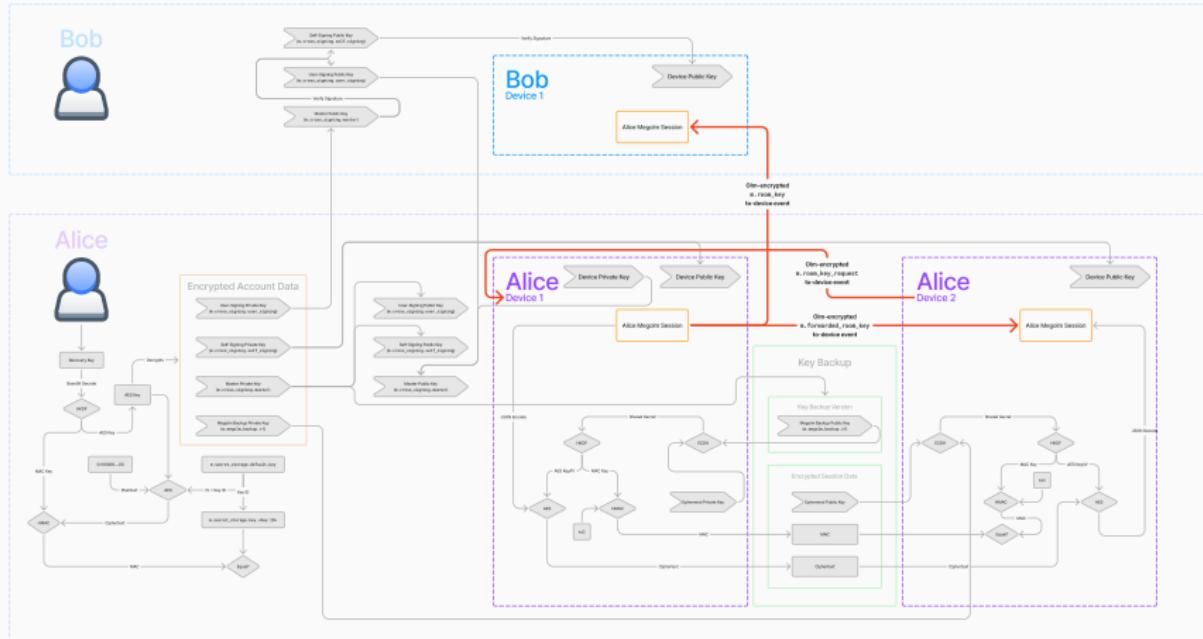
## Sharing Keys

---

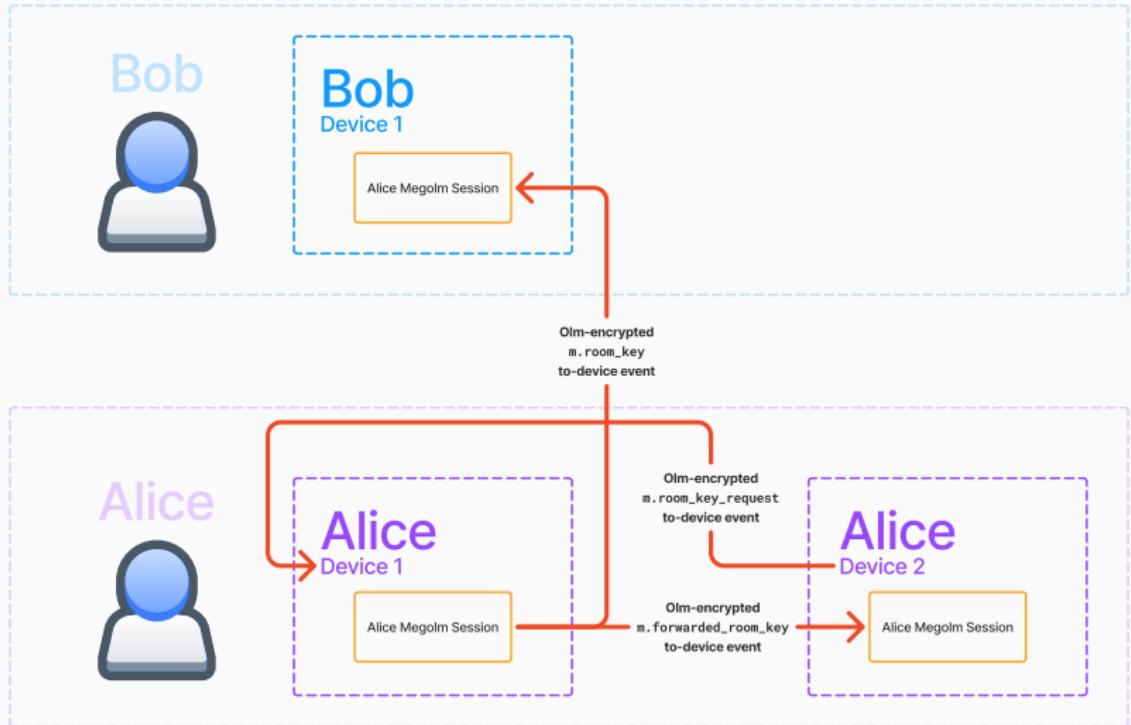
# Big Picture: Message Security



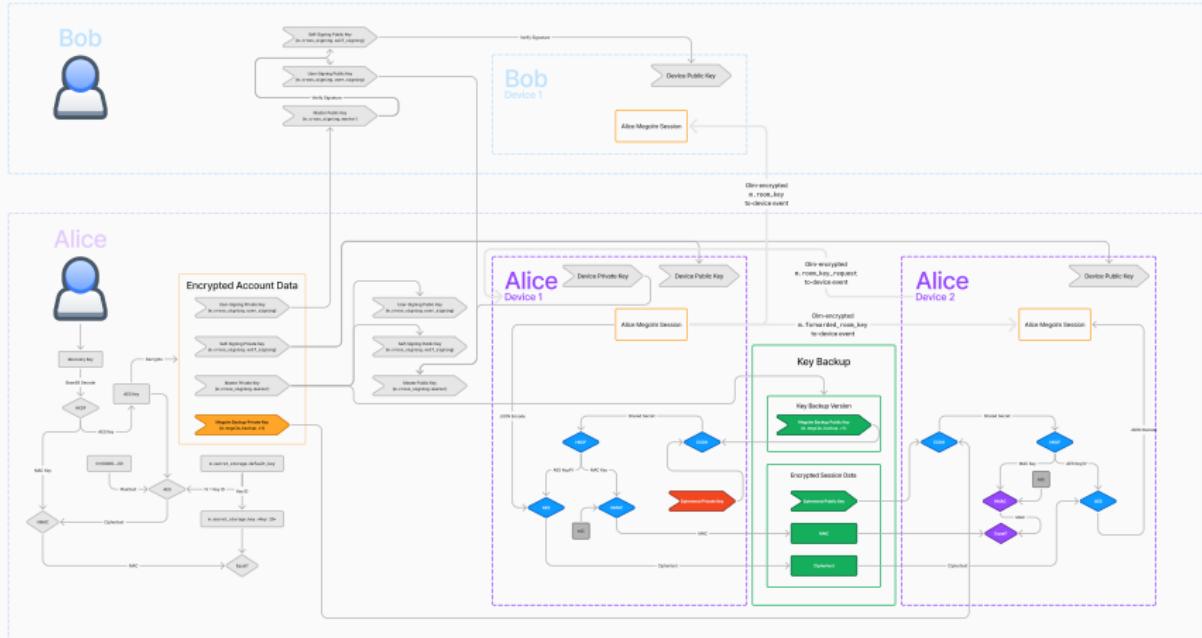
## Encrypted Olm Events



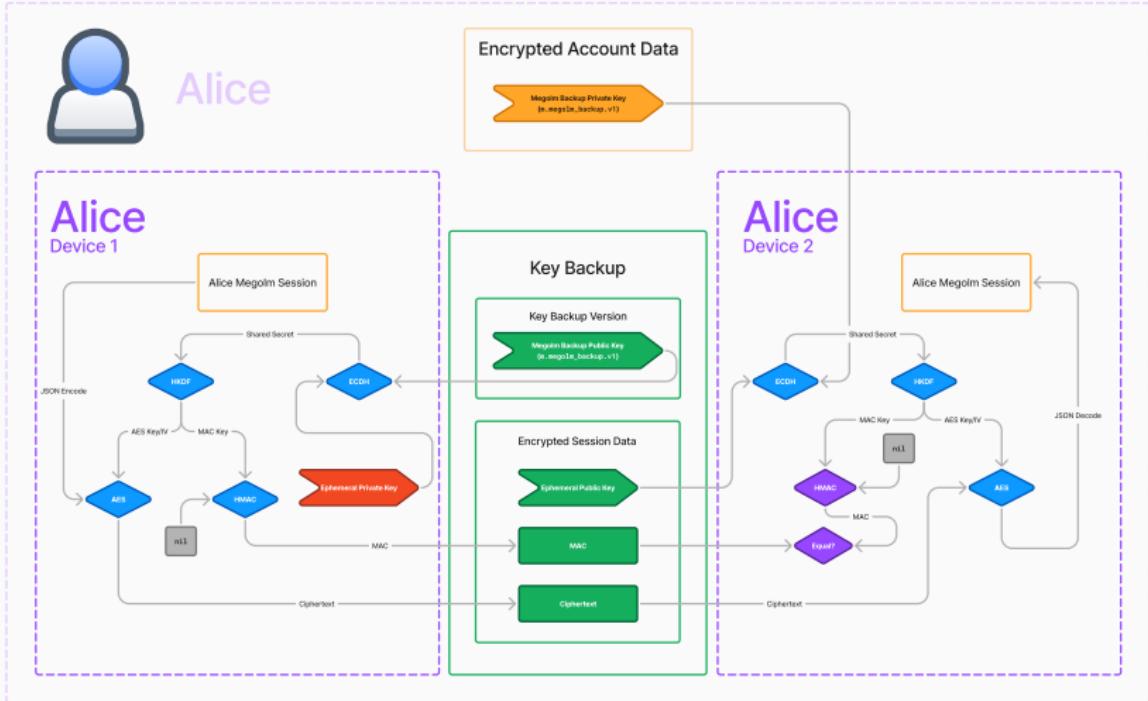
# Encrypted Olm Events



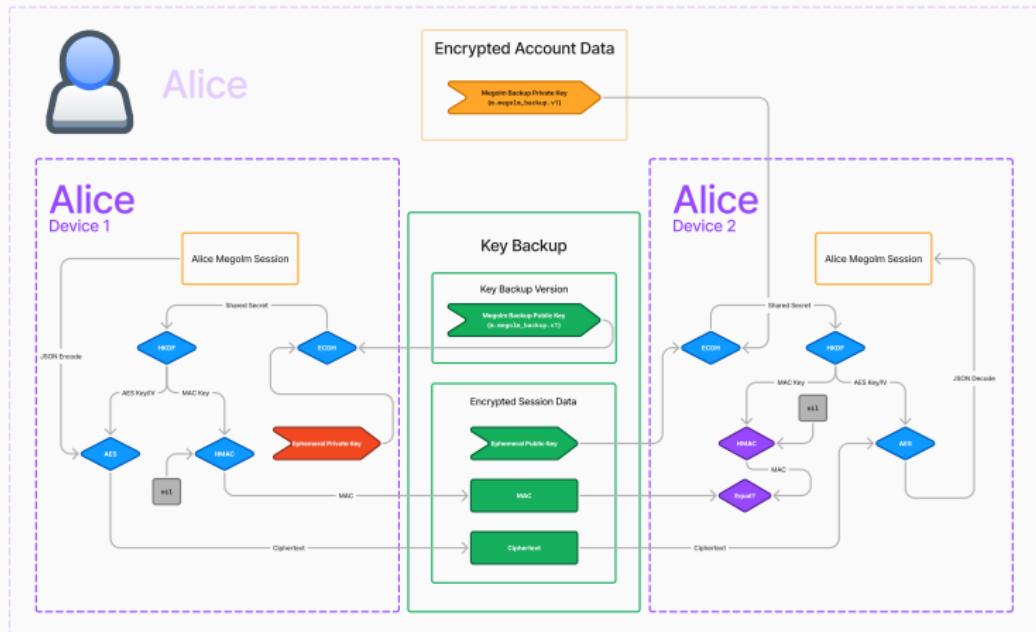
# Key Backup



# Key Backup

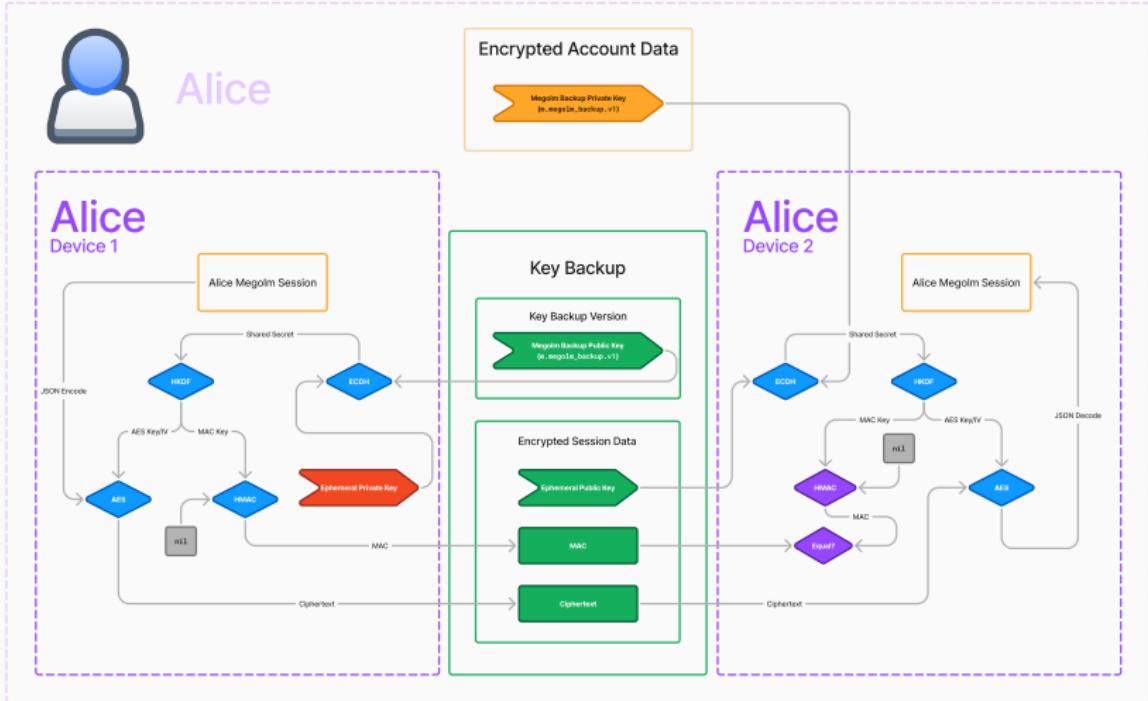


# Key Backup



$$\mathbf{ECDH}(A_{private}, B_{public}) = \mathbf{ECDH}(B_{private}, A_{public}) = K_{shared}$$

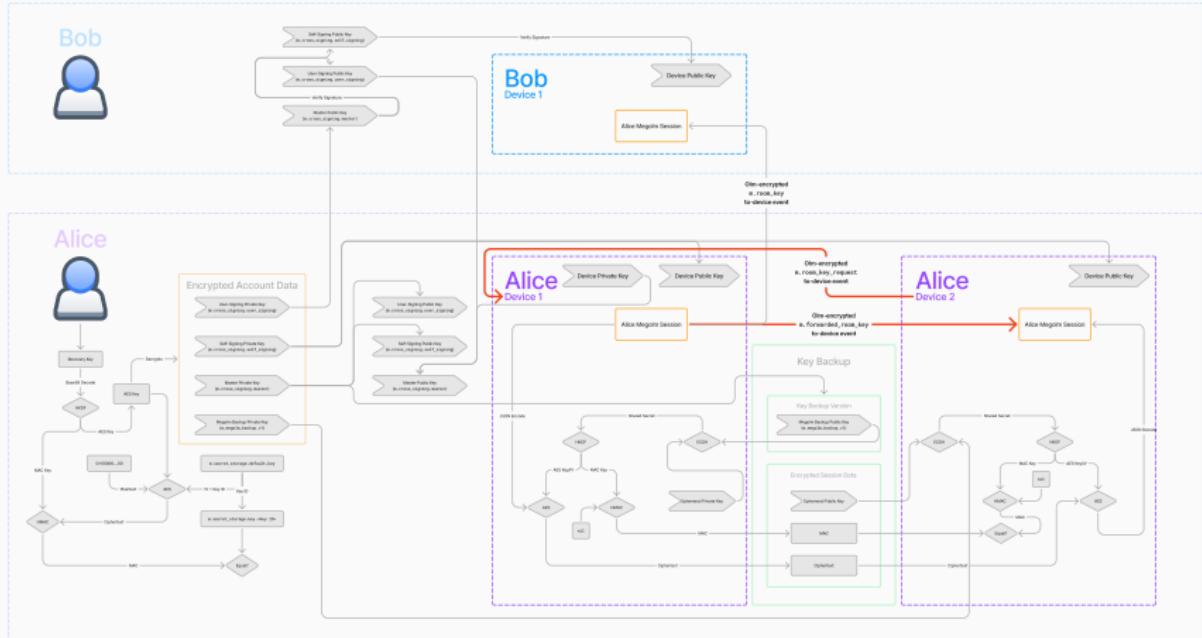
# Key Backup



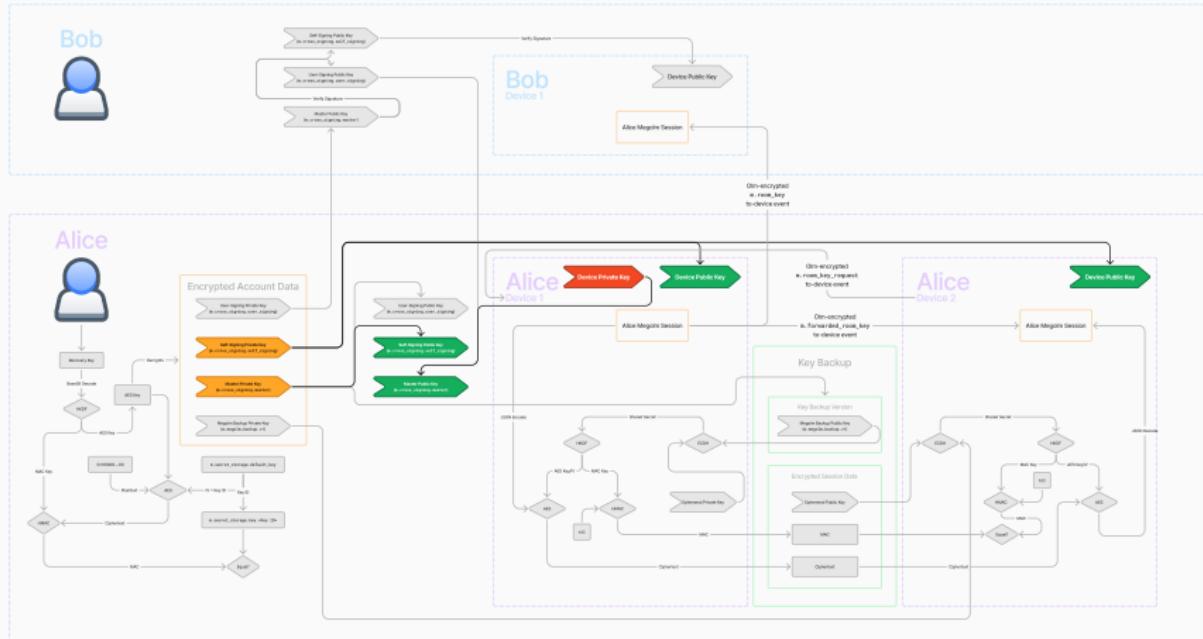
## Device Verification

---

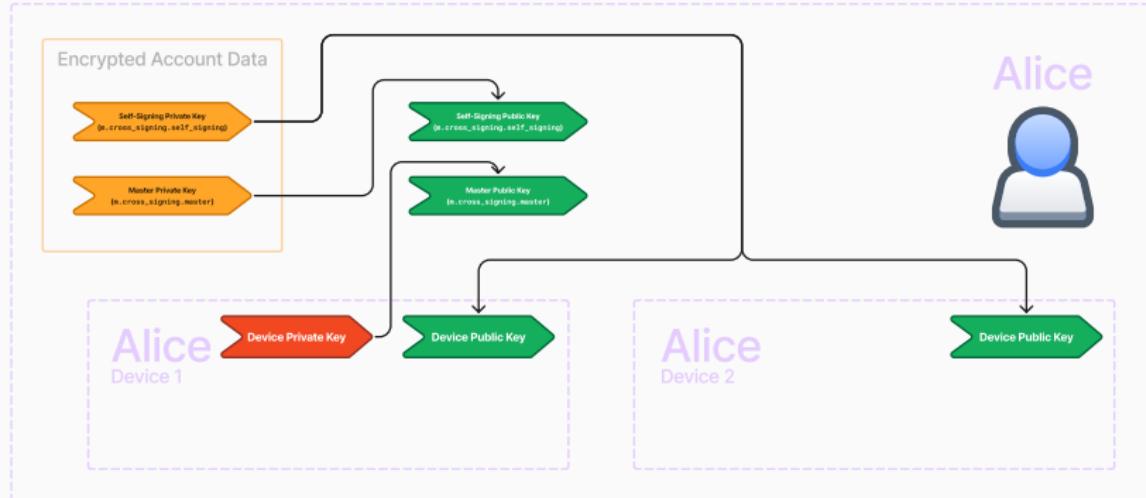
# Who Can We Send Keys To?



## Signatures



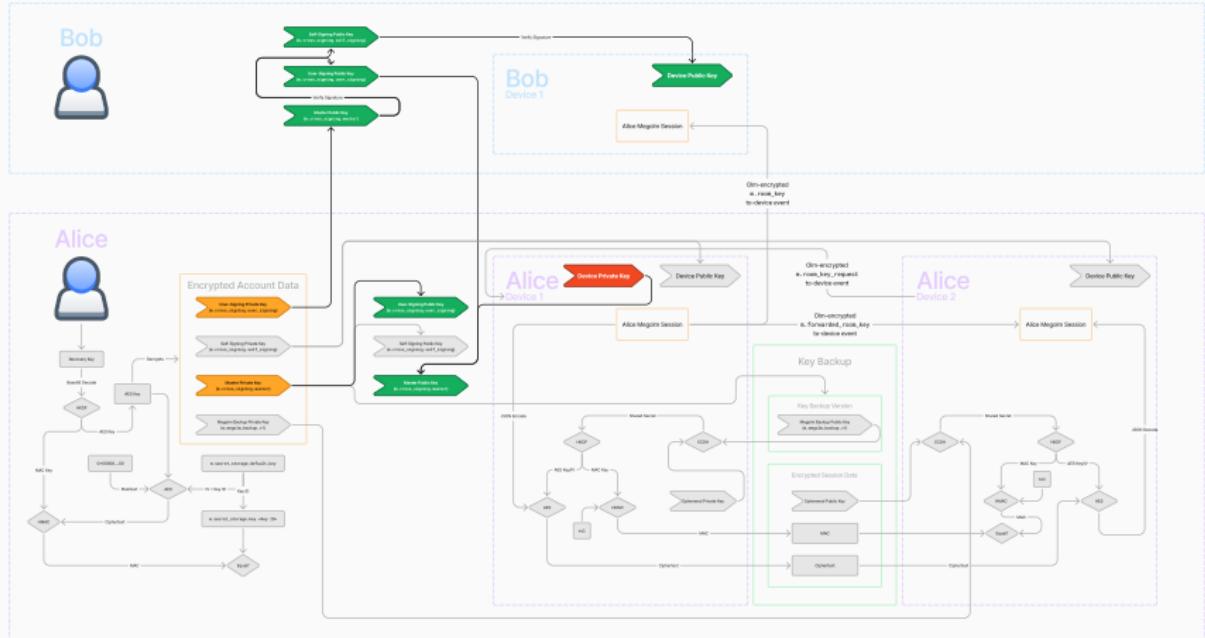
# Signatures



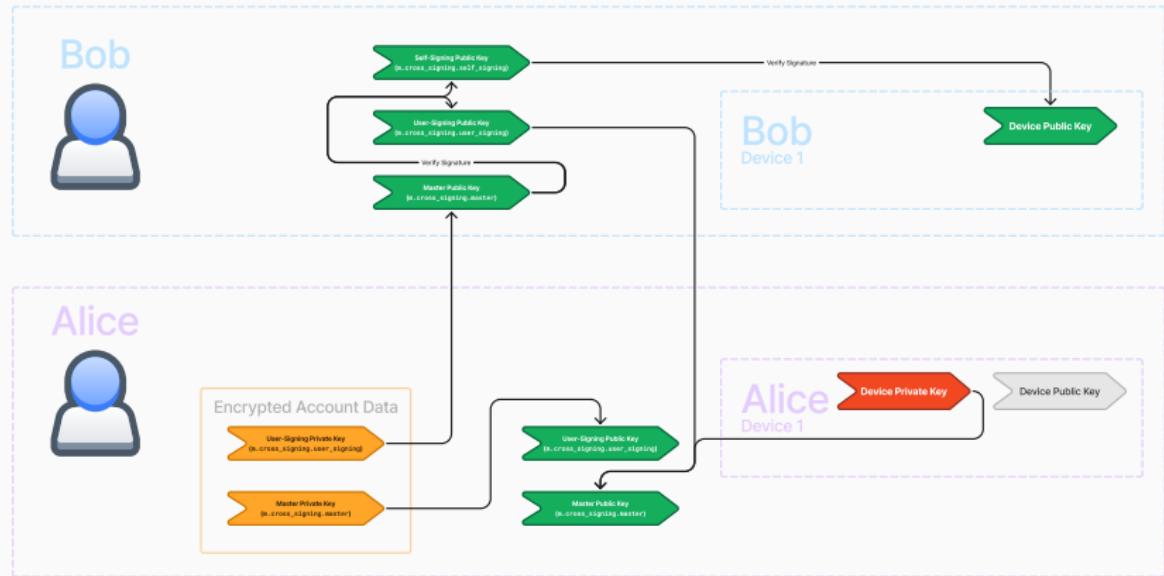
## User Verification

---

# Additional Identity Verification



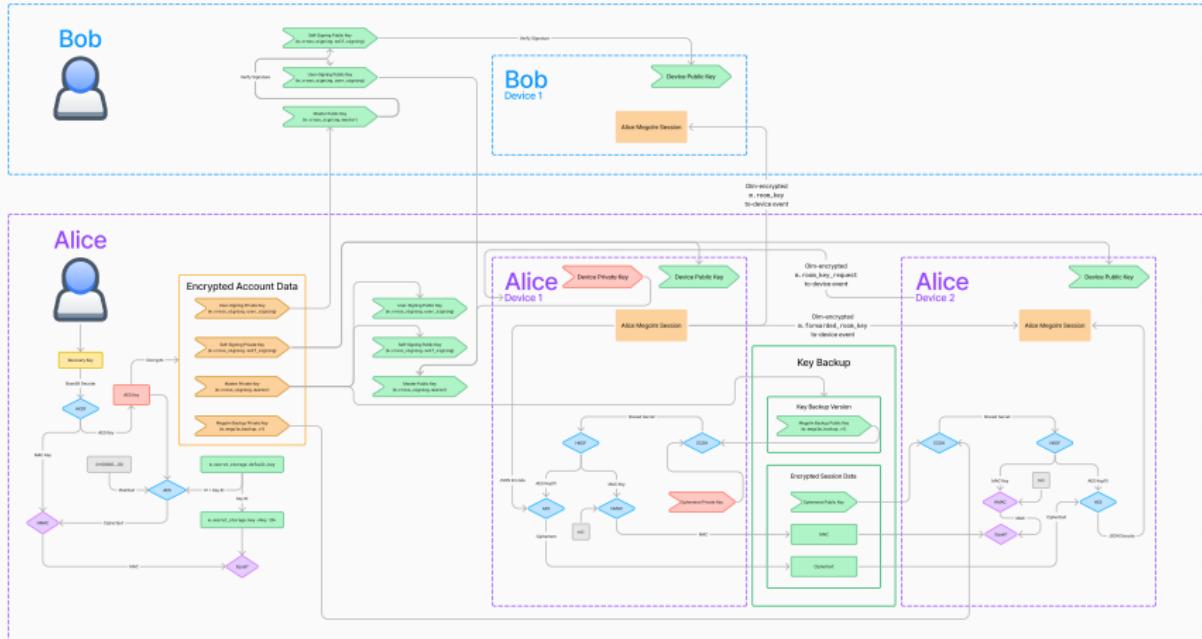
# Additional Identity Verification



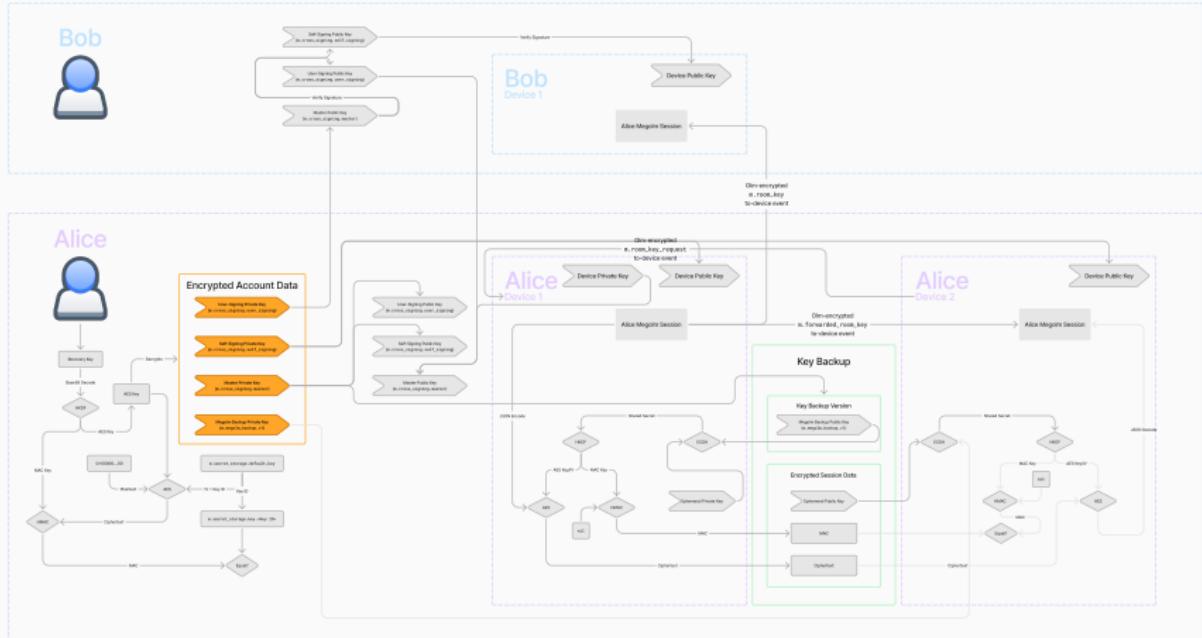
# Secure Secret Storage and Sharing (SSSS)

---

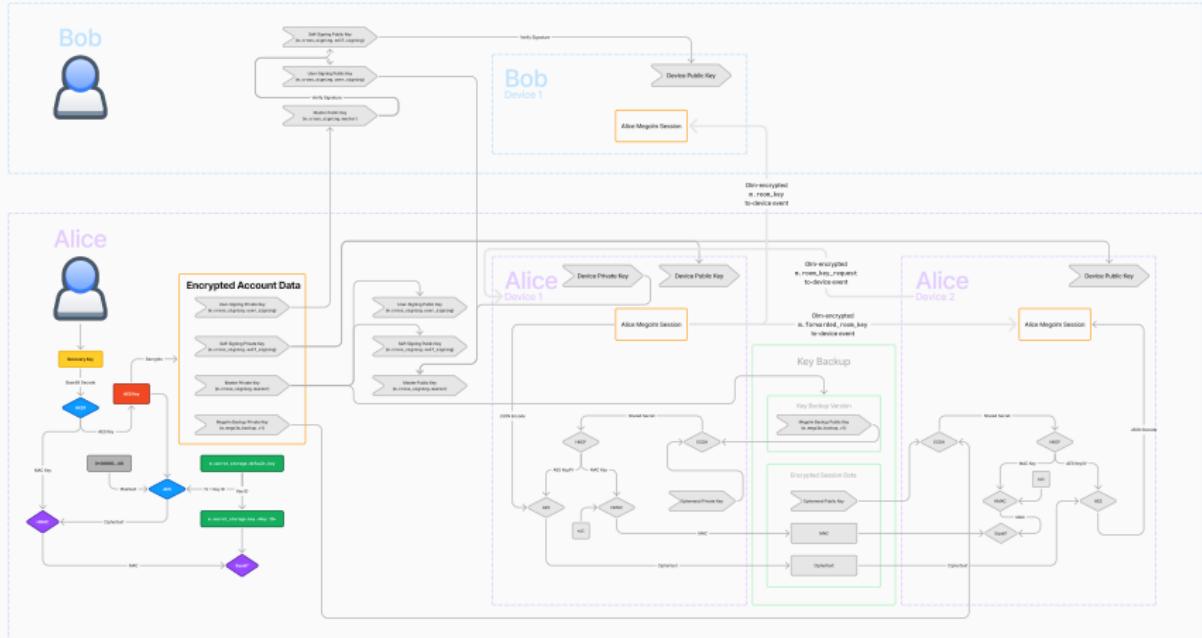
# Don't Forget Your Keys



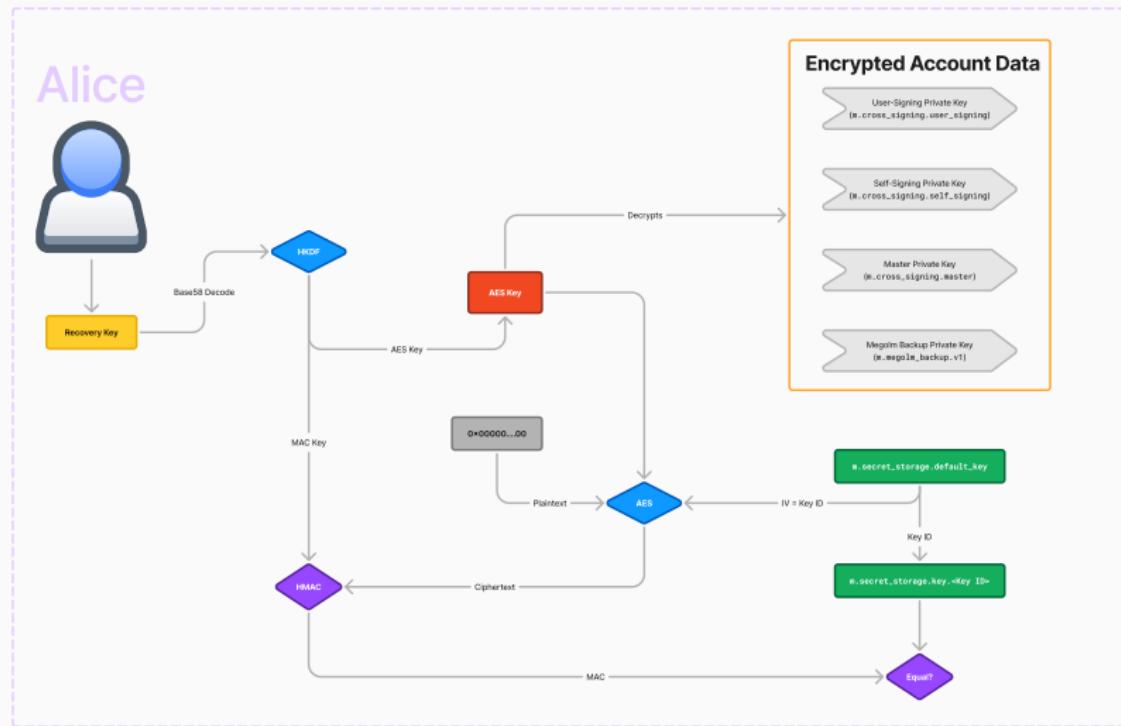
# Don't Forget Your Keys



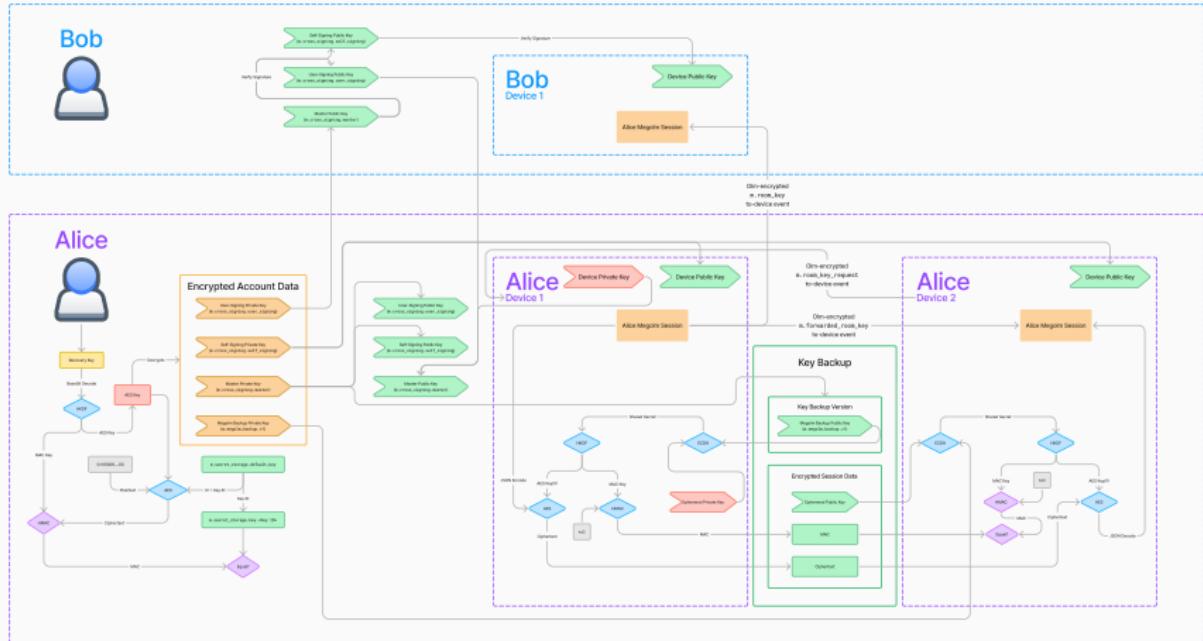
# Don't Forget Your Keys



# Don't Forget Your Keys



# Big Picture



# Thank You for Listening!

Questions?



[sumnerevans.com/posts/matrix/cryptographic-key-infrastructure](https://sumnerevans.com/posts/matrix/cryptographic-key-infrastructure)