Mind the Gap!

Secure File Sharing, from Theory to Practice

Matilda Backendal, **David Balbás**, **Nicola Dardanis**, Miro Haller, Matteo Scarlata

Sofia, 28 March 2025

E2EE Cloud Storage Providers

"WITH MEGA, YOU CONTROL THE ENCRYPTION"

300 MILLION USERS

MEGA

INSECURE!

[BHP23]
[AHMP23]

"ULTIMATE SECURITY"

Nextcloud

INSECURE!

[ABCP23]

"EXCEPTIONALLY PRIVATE CLOUD"

sync.com

"EUROPE'S MOST SECURE CLOUD STORAGE"

pCloud

INSECURE!

Jonas Hofmann &
Kien Tuong Truong
[HT24]

icedrive

Seafile

"THE STRONGEST ENCRYPTED CLOUD STORAGE IN THE WORLD"

"SUPPORTS CLIENT-SIDE END-TO-END ENCRYPTION"

"FREE, ENCRYPTED, AND SECURE CLOUD STORAGE. YOUR PRIVACY, SECURED BY MATH"

Proton Drive

NO IND-CCA SECURITY
Léa Micheloud [thesis]

Secure Shared Folders

E2EE file sharing and **secure shared folders (SSF)** are particularly tricky.

Secure Shared Folders

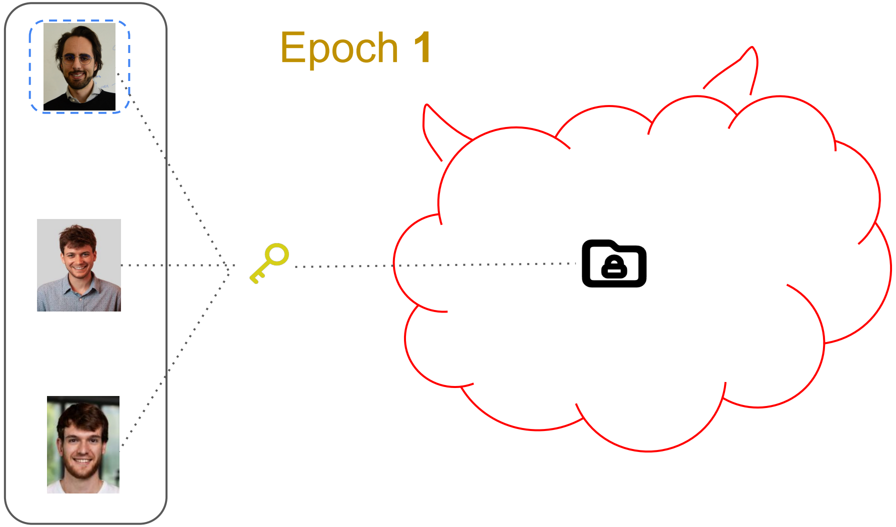
E2EE file sharing and **secure shared folders (SSF)** are particularly tricky.

- Uploader shares files pairwise

Secure Shared Folders

E2EE file sharing and **secure shared folders (SSF)** are particularly tricky.

- Uploader shares files pairwise
- MEGA: shared static **folder key**

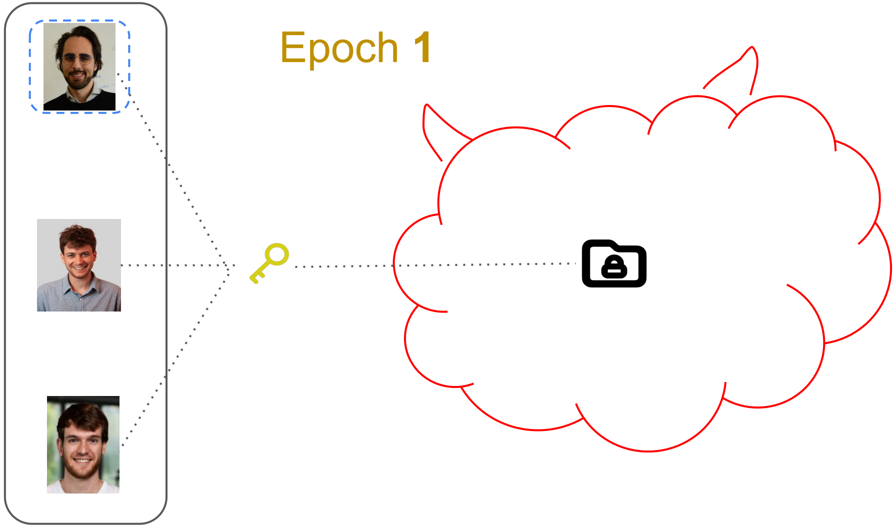


Secure Shared Folders

E2EE file sharing and **secure shared folders (SSF)** are particularly tricky.

- Uploader shares files pairwise
- MEGA: shared static **folder key**

❌ **Dynamic members**: access rights change!

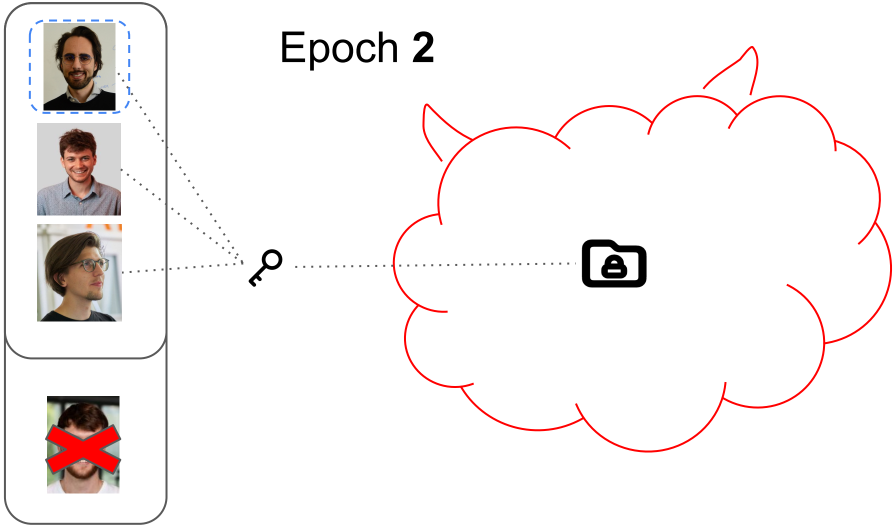


Secure Shared Folders

E2EE file sharing and **secure shared folders (SSF)** are particularly tricky.

- Uploader shares files pairwise
- MEGA: shared static **folder key**

❌ **Dynamic members**: access rights change!



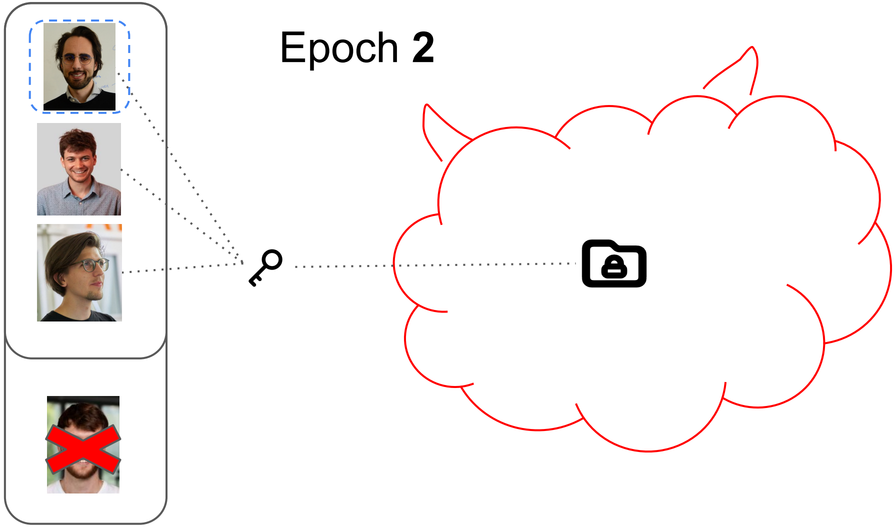
Secure Shared Folders

E2EE file sharing and **secure shared folders (SSF)** are particularly tricky.

- Uploader shares files pairwise
- MEGA: shared static **folder key**

❌ **Dynamic members**: access rights change!

❌ **State exposure**



Secure Shared Folders

E2EE file sharing and **secure shared folders (SSF)** are particularly tricky.

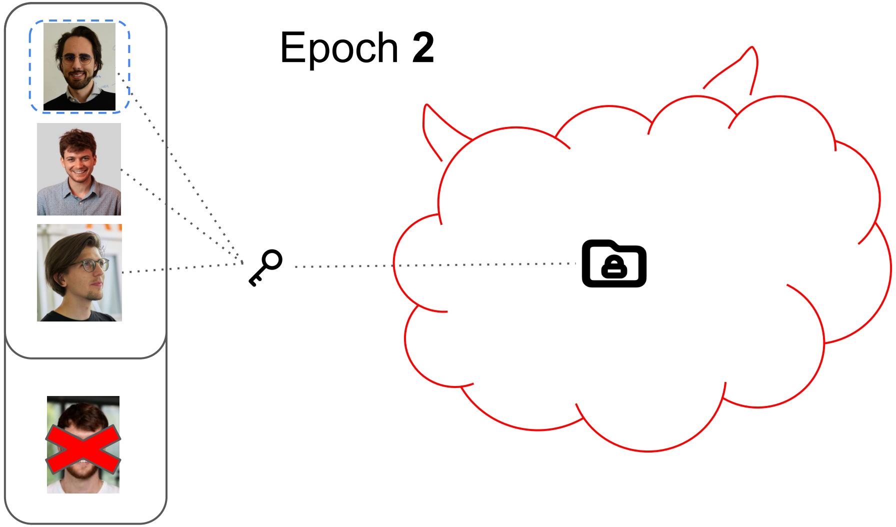
- Uploader shares files pairwise
- MEGA: shared static **folder key**

❌ **Dynamic members**: access rights change!

❌ **State exposure**

Build SSF based on **group keys**:

- Strong security
- Efficiency
- Real-world usability



Strong Security for Shared Folders

What epoch keys can we protect?

$$k_0 \longrightarrow k_1 \longrightarrow k_2 \longrightarrow k_3 \longrightarrow k_4 \longrightarrow k_5 \longrightarrow k_6 \longrightarrow \cdots$$

Strong Security for Shared Folders

What epoch keys can we protect?

$$k_0 \longrightarrow k_1 \longrightarrow k_2 \longrightarrow k_3 \longrightarrow k_4 \longrightarrow k_5 \longrightarrow k_6 \longrightarrow \cdots$$

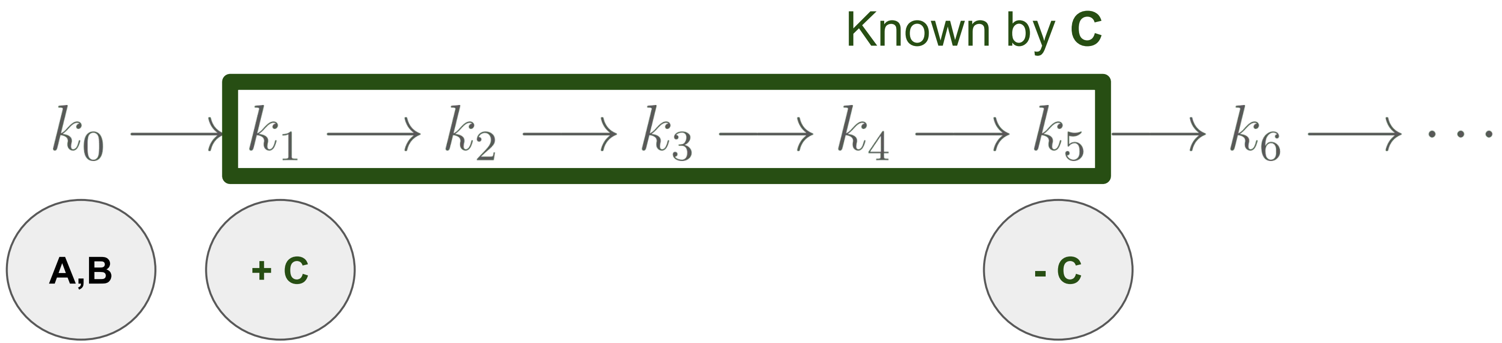
A,B (under k_0)

+ C (under k_1)

- C (under k_5)

Strong Security for Shared Folders

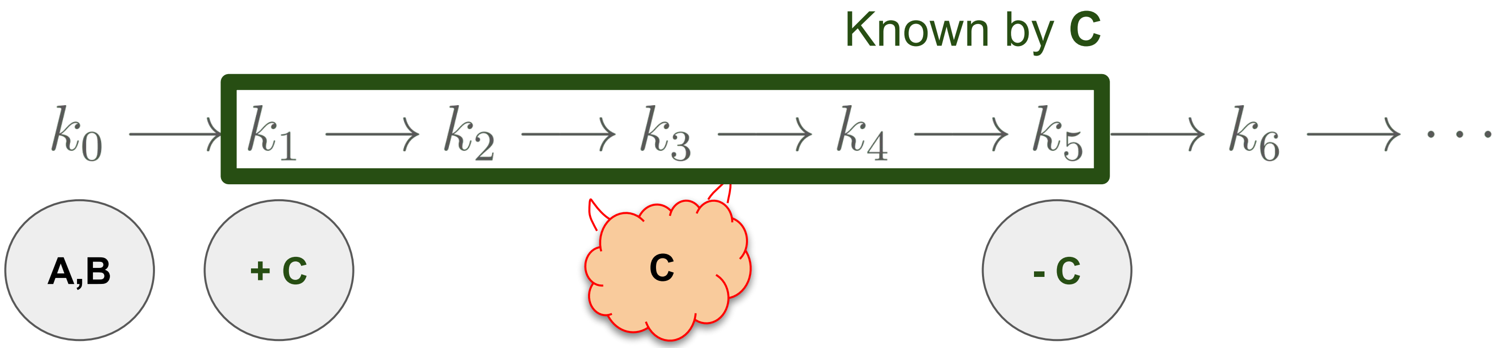
What epoch keys can we protect?



Interval Security

Strong Security for Shared Folders

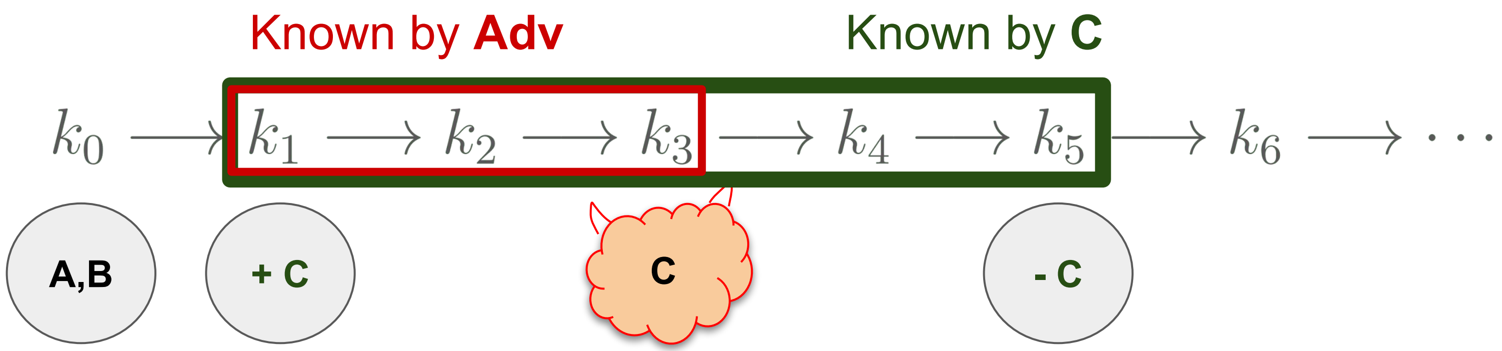
What epoch keys can we protect?



Interval Security

Strong Security for Shared Folders

What epoch keys can we protect?

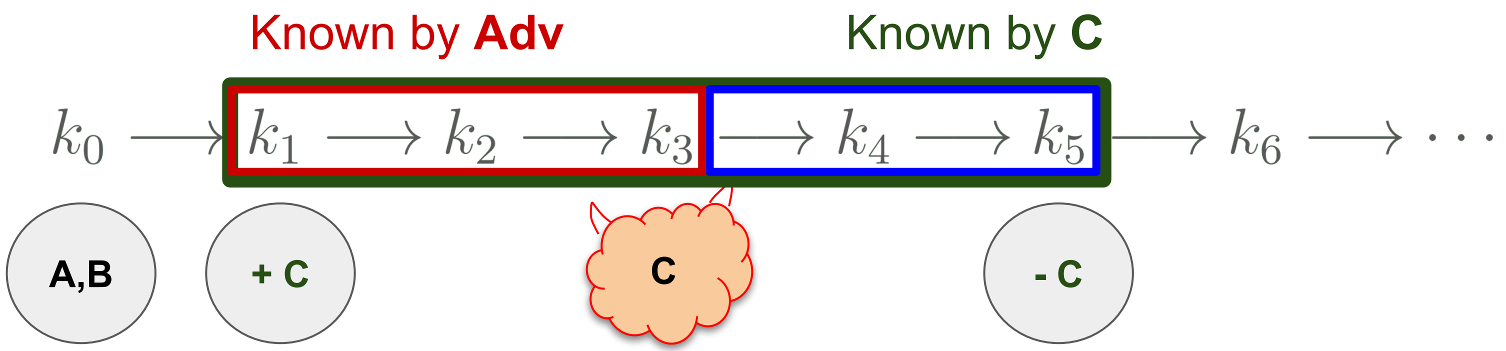


Interval Security

Persistency: leakage of k_1, k_2 unavoidable

Strong Security for Shared Folders

What epoch keys can we protect?



Post Compromise Security (PCS)

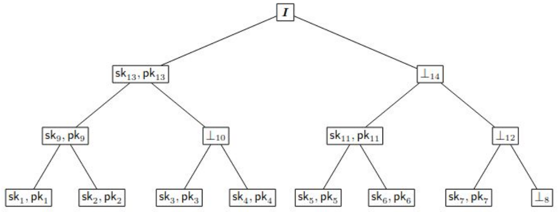
Interval Security

Persistency: leakage of k_1, k_2 unavoidable

Agreeing on Keys for Persistent Data

Natural attempt:

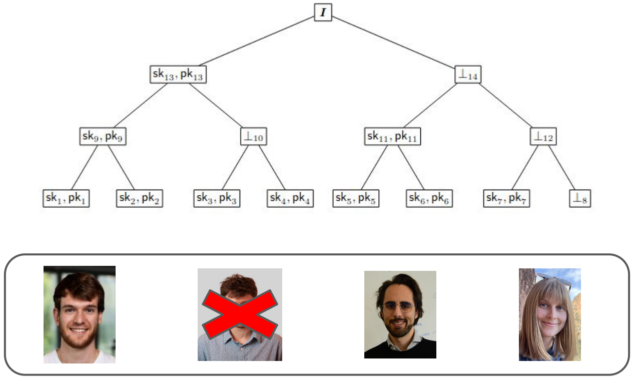
- Run group key agreement (e.g. CGKA as in MLS)



Agreeing on Keys for Persistent Data

Natural attempt:

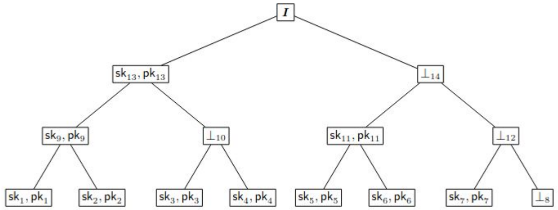
- Run group key agreement (e.g. CGKA as in MLS)
- Derive fresh k_i per epoch



Agreeing on Keys for Persistent Data

Natural attempt:

- Run group key agreement (e.g. CGKA as in MLS)
- Derive fresh k_i per epoch
- Store all keys

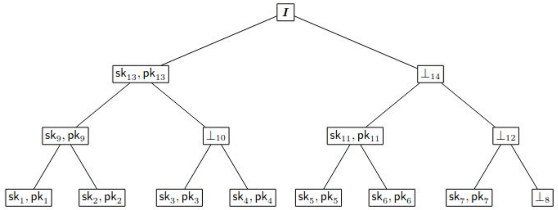


Agreeing on Keys for Persistent Data

Natural attempt:

- Run group key agreement (e.g. CGKA as in MLS)
- Derive fresh k_i per epoch
- Store all keys

✅ Security (from CGKA) & MLS implementation



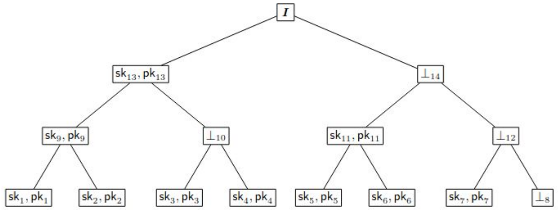
Agreeing on Keys for Persistent Data

Natural attempt:

- Run group key agreement (e.g. CGKA as in MLS)
- Derive fresh k_i per epoch
- Store all keys

✅ Security (from CGKA) & MLS implementation

❌ State grows linearly on number of epochs



Agreeing on Keys for Persistent Data

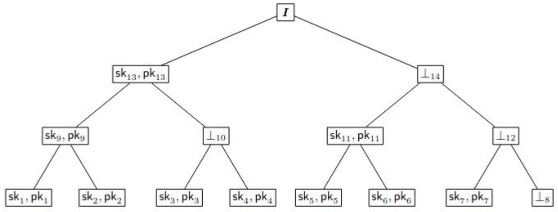
Natural attempt:

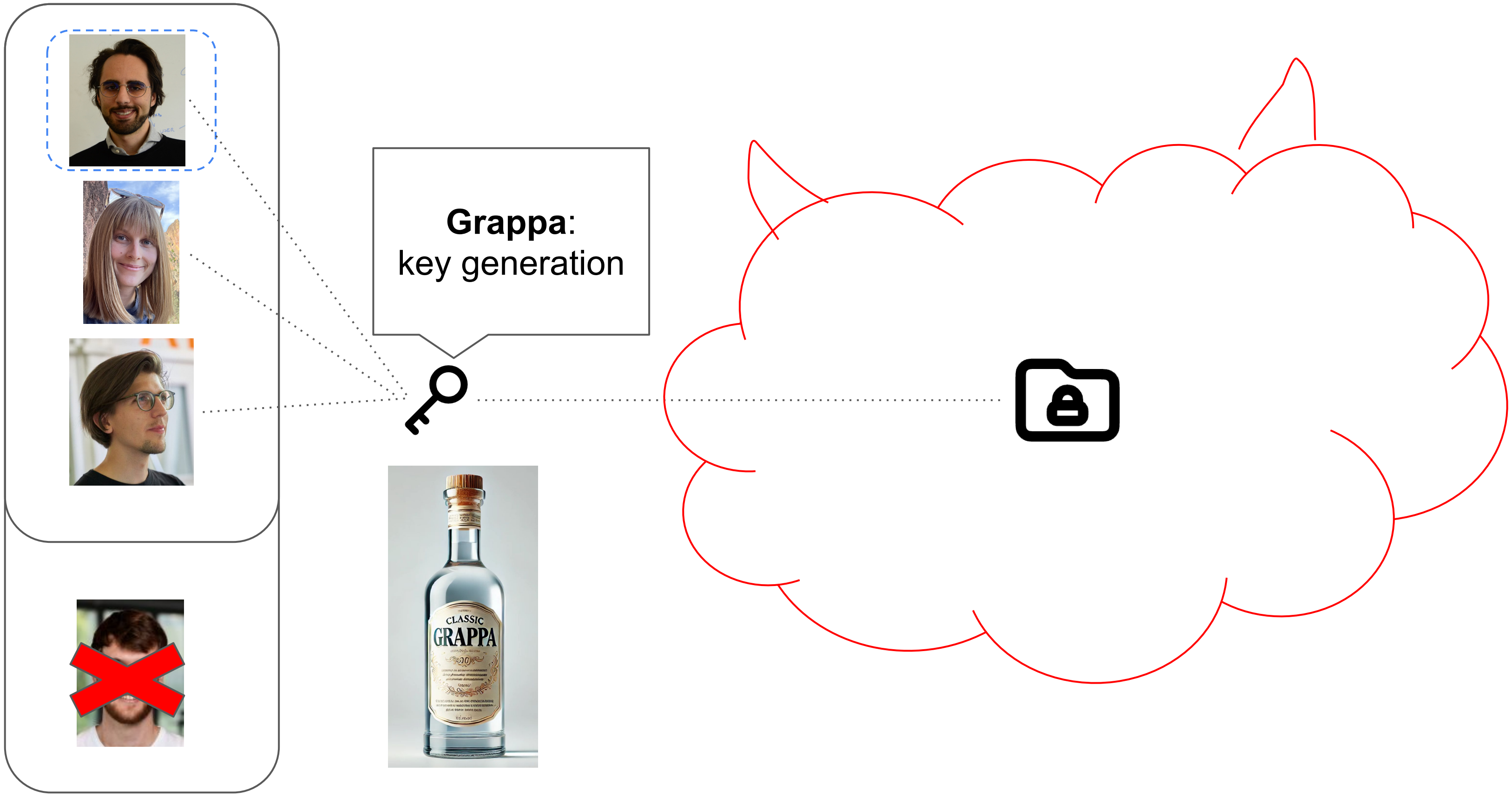
- Run group key agreement (e.g. CGKA as in MLS)
- Derive fresh k_i per epoch
- Store all keys

✅ Security (from CGKA) & MLS implementation

❌ State grows linearly on number of epochs

Can we get a good trade-off?




Grappa:
key generation
CLASSIC
GRAPPA

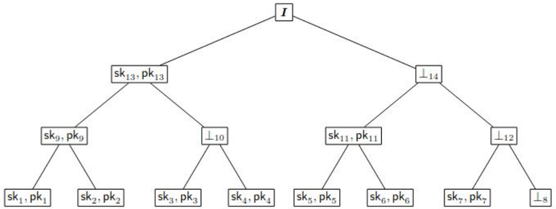
Grappa: Group Key Progression for Persistent Access

Epoch-based **progression of keys for persistent use**

Grappa: Group Key Progression for Persistent Access

Epoch-based **progression of keys for persistent use**

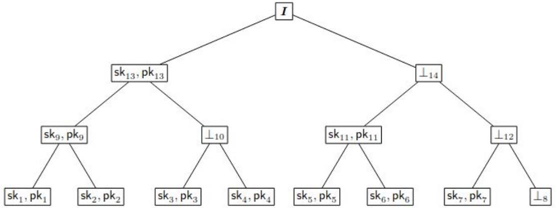
CGKA: Continuous Group Key Agreement



Grappa: Group Key Progression for Persistent Access

Epoch-based **progression of keys for persistent use**

CGKA: Continuous Group Key Agreement



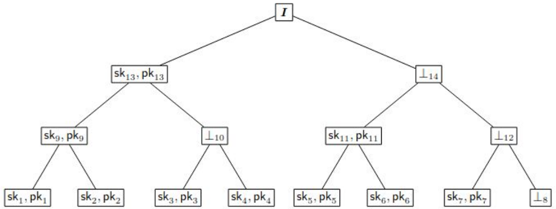
Interval scheme: compact symmetric-key primitive to produce interval keys

$$\textbf{st} \longrightarrow [k_\ell, \ldots, k_r]$$

Grappa: Group Key Progression for Persistent Access

Epoch-based **progression of keys for persistent use**

CGKA: Continuous Group Key Agreement



Interval scheme: compact symmetric-key primitive to produce interval keys

$$\textbf{st} \longrightarrow [k_\ell, \ldots, k_r]$$

CGKA keys encrypt interval scheme states – **CGKA as transport layer**

Grappa Insights

- Grappa: **strong security** for **persistent** data in **group** settings

Grappa Insights

- Grappa: **strong security** for **persistent** data in **group** settings
- Provable security, compact state

Grappa Insights

- Grappa: **strong security** for **persistent** data in **group** settings
- Provable security, compact state
- **Novel use-case for CGKA** beyond messaging

Grappa Insights

- Grappa: **strong security** for **persistent** data in **group** settings
- Provable security, compact state
- **Novel use-case for CGKA** beyond messaging

*What are the **challenges** of building a secure shared folder system using CGKA?*

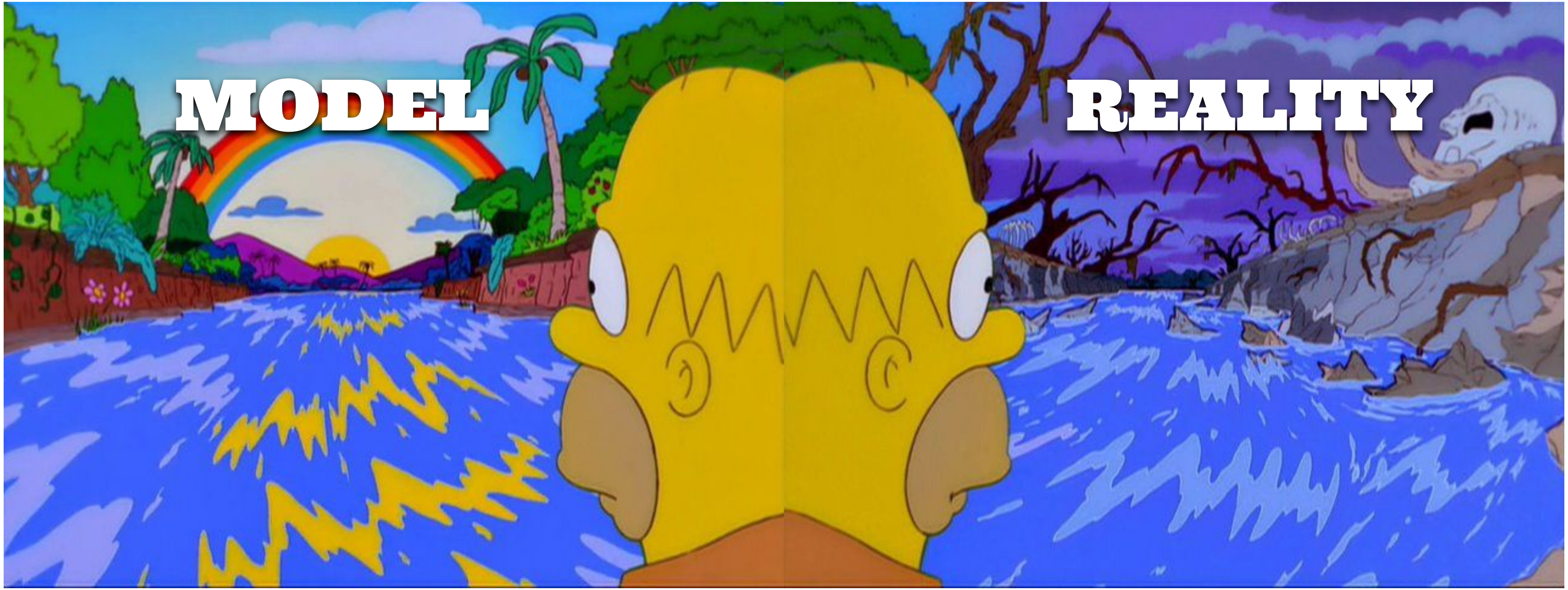
Grappa Insights

- Grappa: **strong security** for **persistent** data in **group** settings
- Provable security, compact state
- **Novel use-case for CGKA** beyond messaging

What are the ***challenges*** *of building a secure shared folder system using CGKA?*

Implementation in a real-world setting!

Implementation



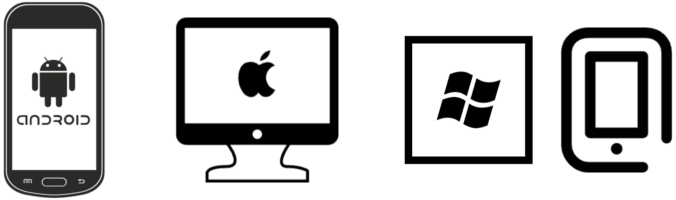
Engineering Gaps

Model

1. Abstract client device / capabilities

Reality

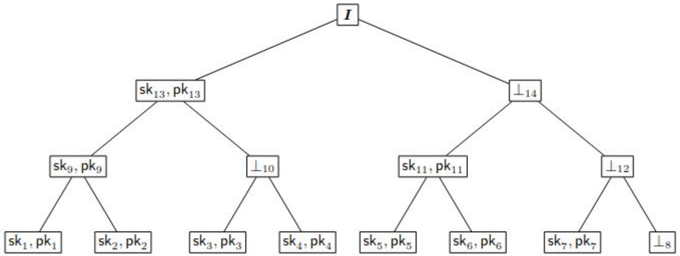
1. Unharmonized capabilities / portability



Engineering Gaps

Model

1. Abstract client device / capabilities
2. Crypto primitives as mathematical objects



Reality

1. Unharmonized capabilities / portability
2. Crypto primitives support in the execution platform / libraries

List of existing MLS implementations

- MLSpp (C++) https://github.com/cisco/mlspp (Status: RFC)
- OpenMLS (Rust) https://github.com/openmls/openmls (Status: RFC)
- mls-kotlin (Kotlin) https://github.com/Traderjoe95/mls-kotlin (Status: RFC)
- mls-rs (Rust) [https://github.com/awslabs/mls-rs] (Status: RFC)
- RingCentral proprietary implementation (C++) (Status: draft-11; RFC in progress)
- `MLS*` (F*) (Status: RFC in progress)
- BouncyCastle (Java) https://github.com/bcgit/bc-java (Status: RFC)
- go-mls (Go) (Status: RFC in progress)

Why crypto in Browsers?

- Browser: **cross platform runtime** to access applications in cloud

Why crypto in Browsers?

- Browser: **cross platform runtime** to access applications in cloud
- **Standardised Web Crypto API** (W3C) for JS Runtimes

Web Cryptography API

W3C Recommendation *26 January 2017*

This Version:
https://www.w3.org/TR/2017/REC-WebCryptoAPI-20170126/

Latest Published Version:
https://www.w3.org/TR/WebCryptoAPI/

Latest editor's draft:
https://w3c.github.io/webcrypto/Overview.html

Previous Version:
https://www.w3.org/TR/2016/PR-WebCryptoAPI-20161215/

Editor:
Mark Watson, Netflix <watsonm@netflix.com>

Errata for this document will be gathered from issues.

See also **translations**.

Participate:
We are on GitHub.
Send feedback to public-web-security@w3.org (archives).
File a bug (see existing bugs).

Web / JS Crypto Ecosystem

- Exhibits non-standard behaviours (loose specifications)

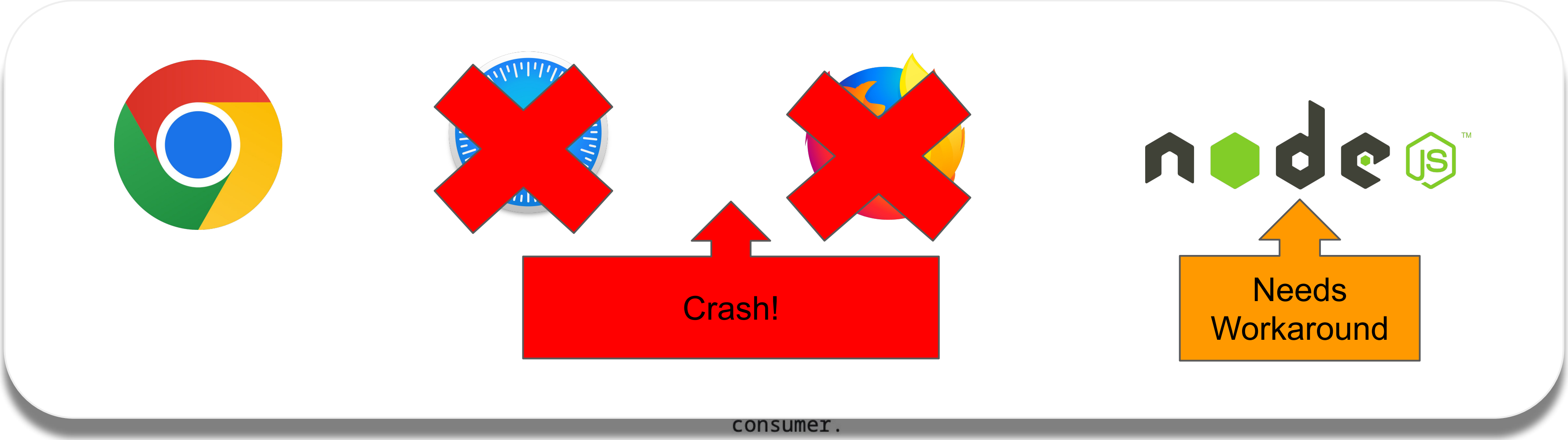
MLS implementation relies on public keys computed from private keys

```
o  publicKey contains the elliptic curve public key associated with
   the private key in question.  The format of the public key is
   specified in Section 2.2 of [RFC5480].  Though the ASN.1 indicates
   publicKey is OPTIONAL, implementations that conform to this
   document SHOULD always include the publicKey field.  The publicKey
   field can be omitted when the public key has been distributed via
   another mechanism, which is beyond the scope of this document.
   Given the private key and the parameters, the public key can
   always be recomputed; this field exists as a convenience to the
   consumer.
```

Web / JS Crypto Ecosystem

- Exhibits non-standard behaviours (loose specifications)

MLS implementation relies on public keys computed from private keys



Web / JS Crypto Ecosystem

- Exhibits non-standard behaviours (loose specifications)
- Introduction of new primitives takes very long!

Secure Curves in the Web Cryptography API

Draft Community Group Report 21 October 2024

Status of This Document

This specification was published by the Web Platform Incubator Community Group. It is not a W3C Standard nor is it on the W3C Standards Track. Please note that under the W3C Community Contributor License Agreement (CLA) there is a limited opt-out and other conditions apply. Learn more about W3C Community and Business Groups.

This is an unofficial proposal.

> 8y, not a standard yet!!

```
ernet Research Task Force (IRTF)                              A. Langley
uest for Comments: 7748                                           Google
egory: Informational                                          M. Hamburg
: 2070-1721                                Rambus Cryptography Research
                                                               S. Turner
                                                                   sn3rd
                                                            January 2016
```

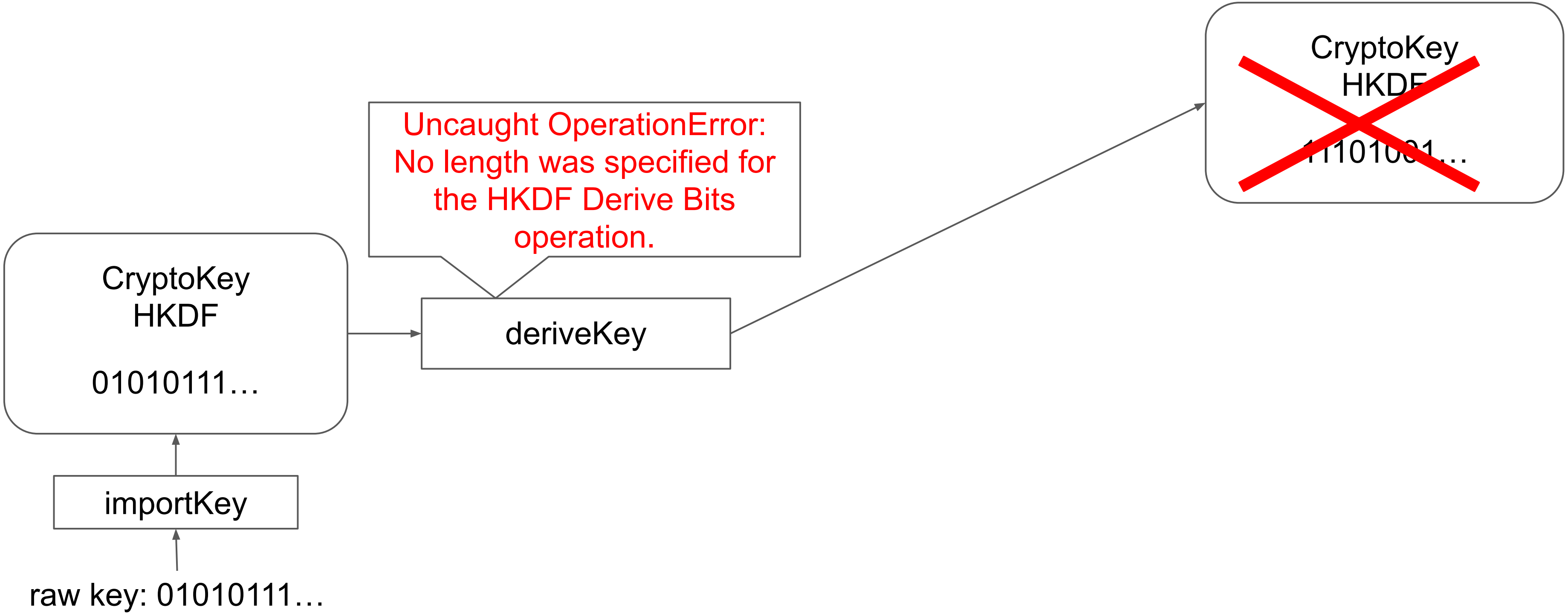
: Curves for Security

```
liptic curves over prime fields that offer
security in cryptographic applications,
including Transport Layer Security (TLS).  These curves are intended
to operate at the ~128-bit and ~224-bit security level, respectively,
and are generated deterministically based on a list of required
properties.
```

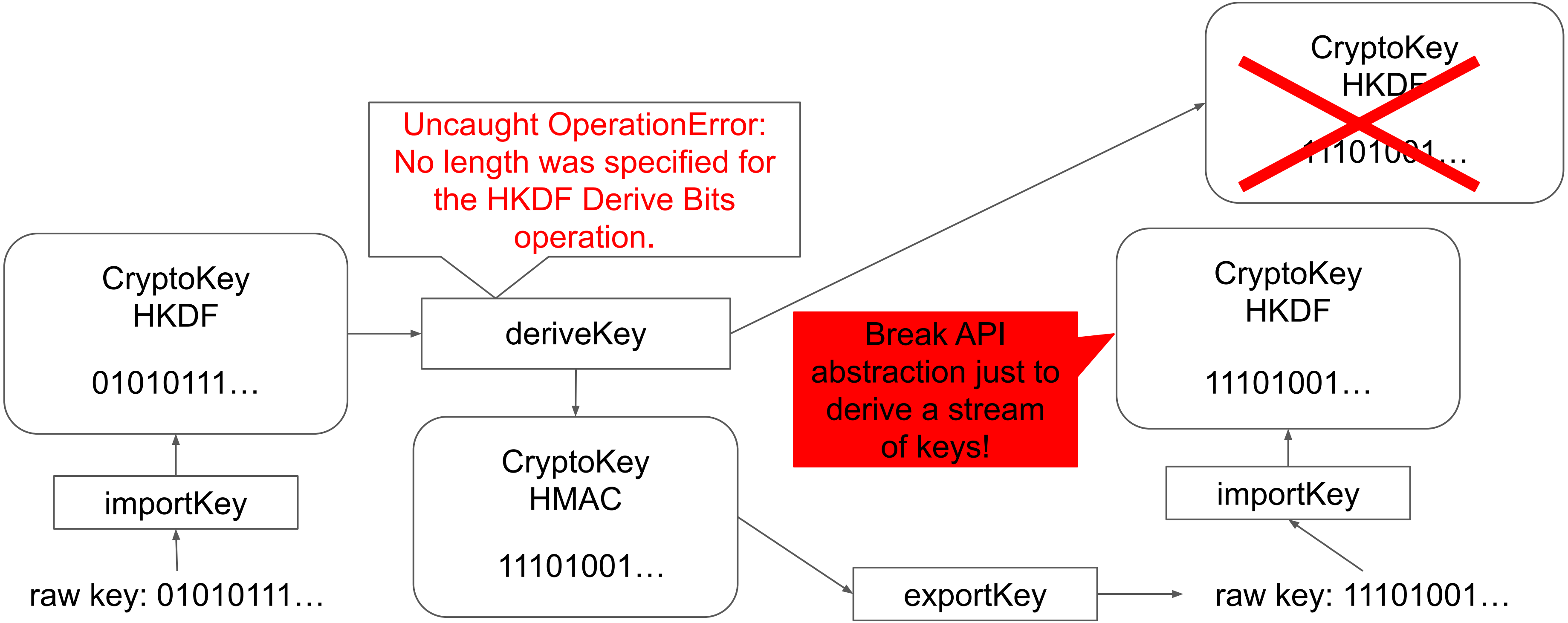
Web / JS Crypto Ecosystem

- Exhibits non-standard behaviours (loose specifications)
- Introduction of new primitives takes very long!
- Overprotective, too restrictive to implement advanced crypto

Web / JS Crypto Ecosystem



Web / JS Crypto Ecosystem



Engineering Gaps

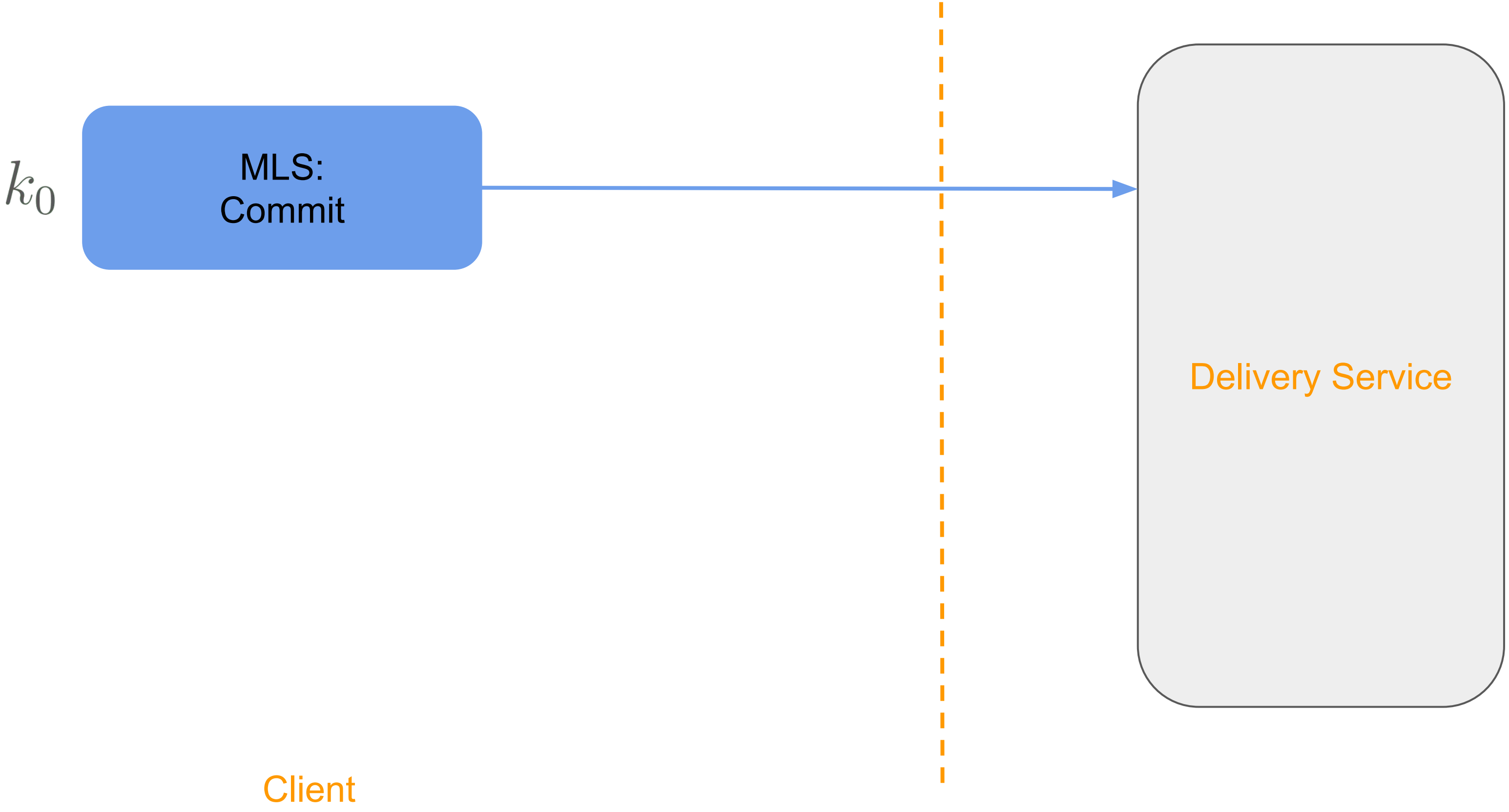
Model

1. Abstract client device / capabilities
2. Crypto primitives as mathematical objects
3. Atomic operation of the scheme

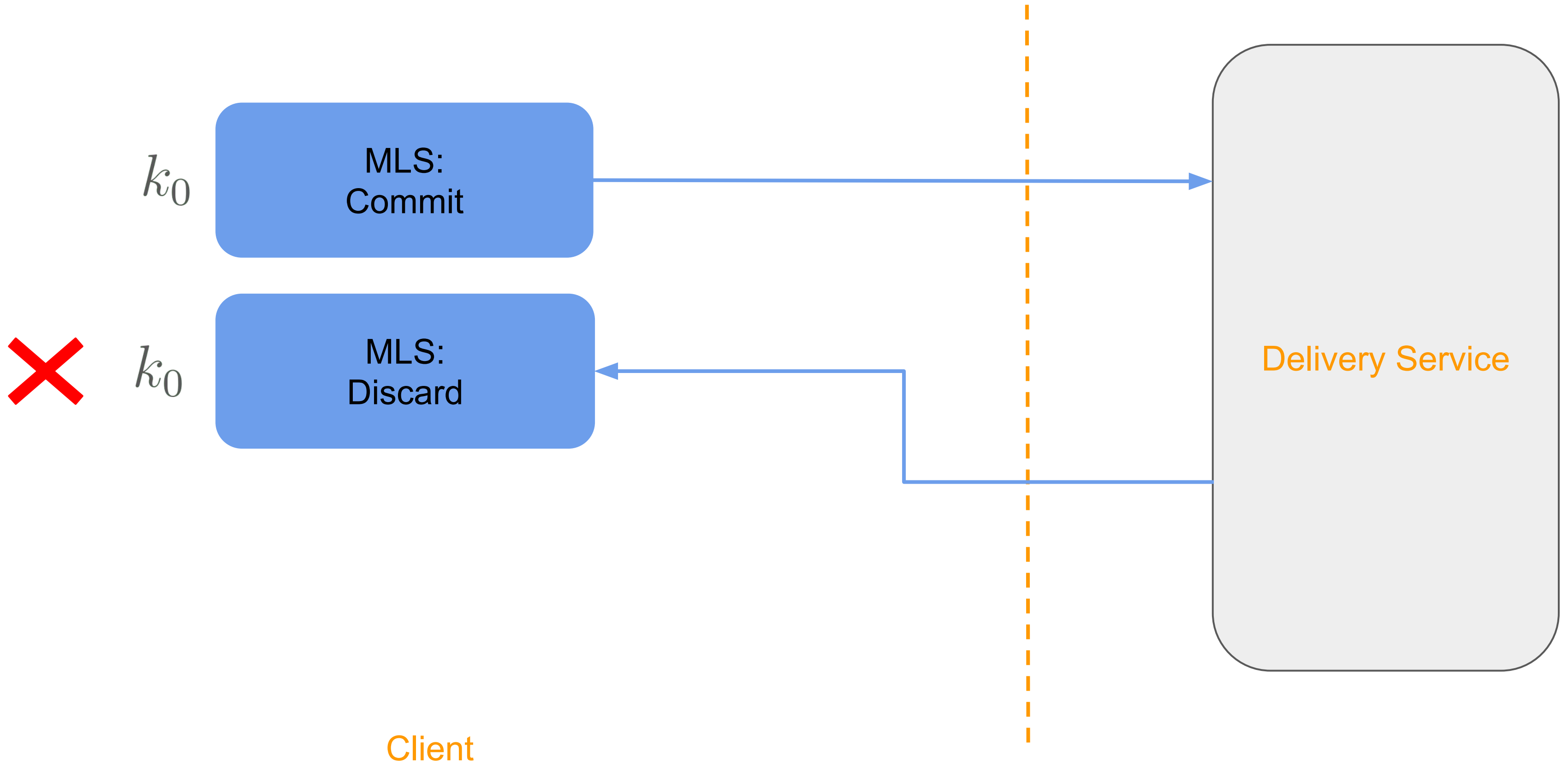
Reality

1. Unharmonized capabilities / portability
2. Crypto primitives support in the execution platform / libraries
3. Multiple schemes, non atomic interactions between components

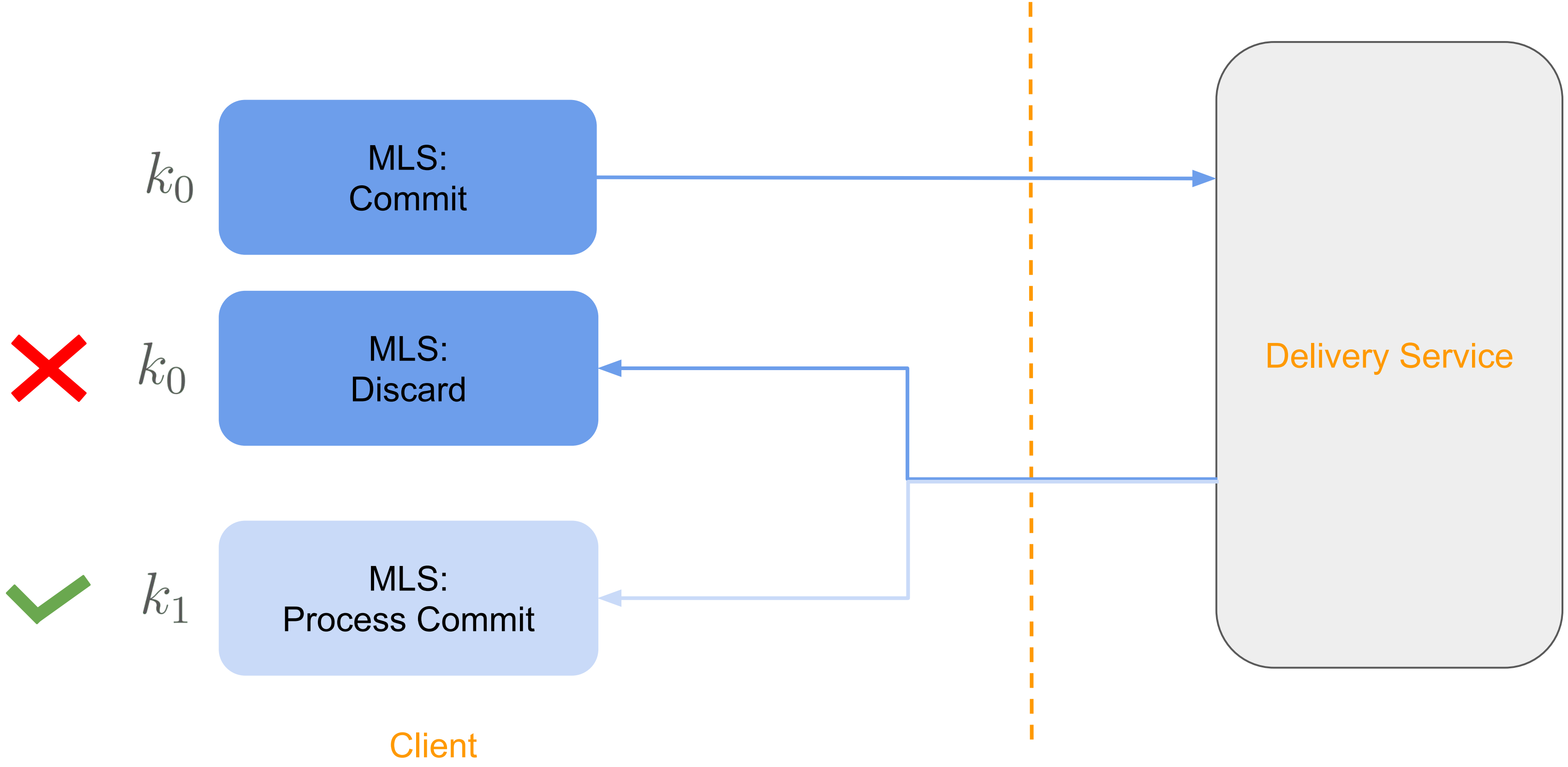
MLS ⇨ Atomic Updates



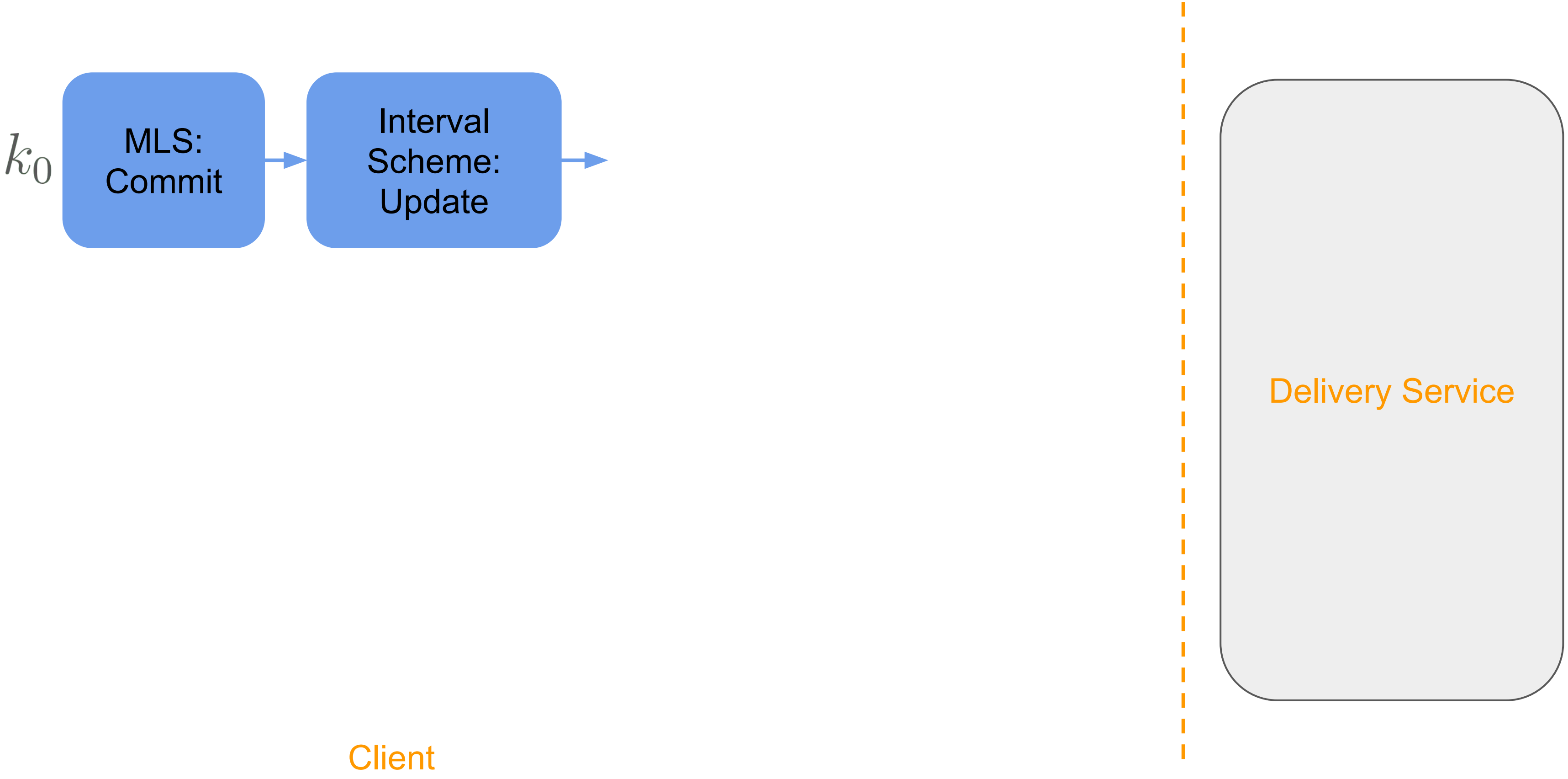
MLS ⇨ Atomic Updates



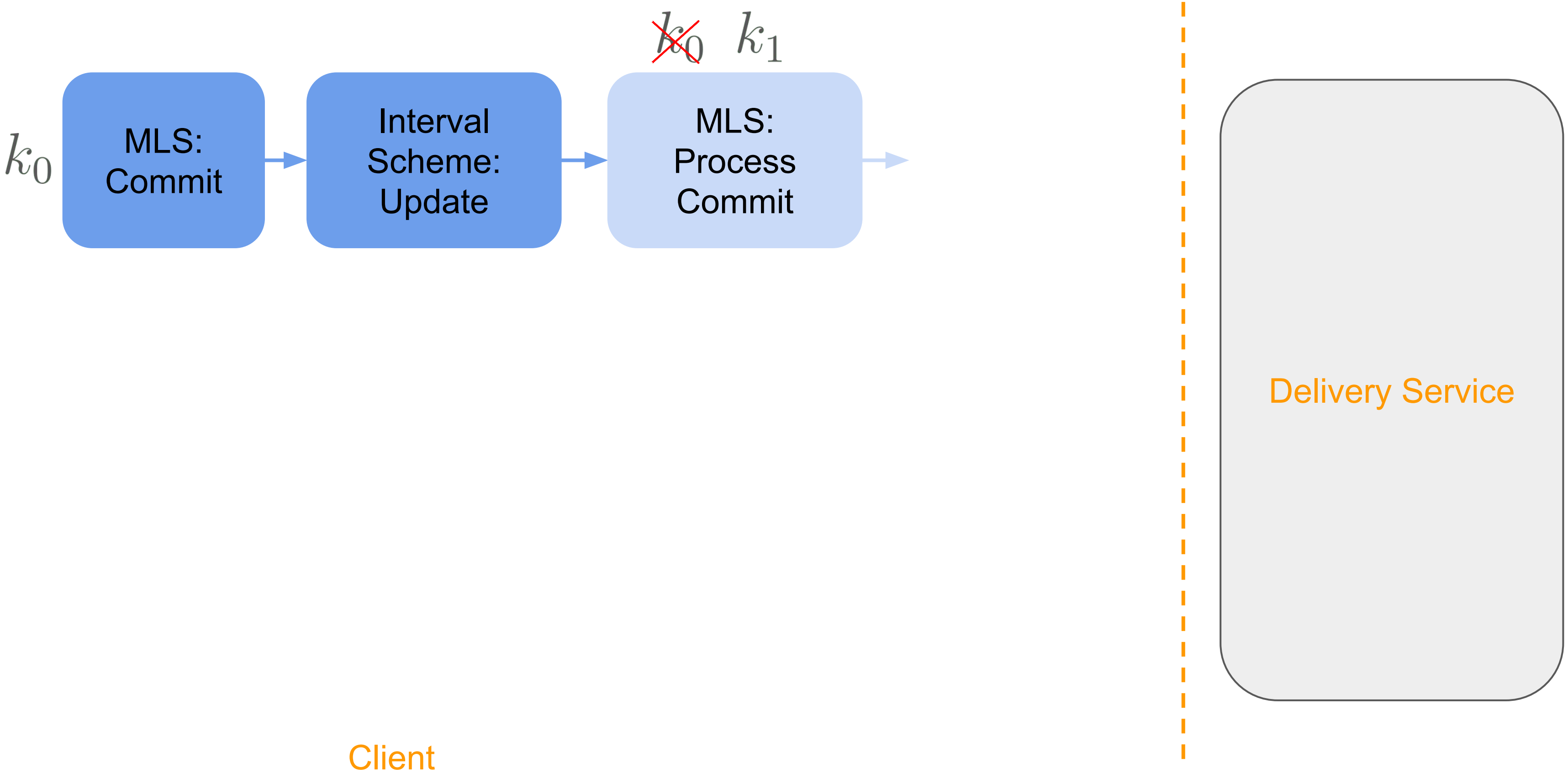
MLS ➪ Atomic Updates



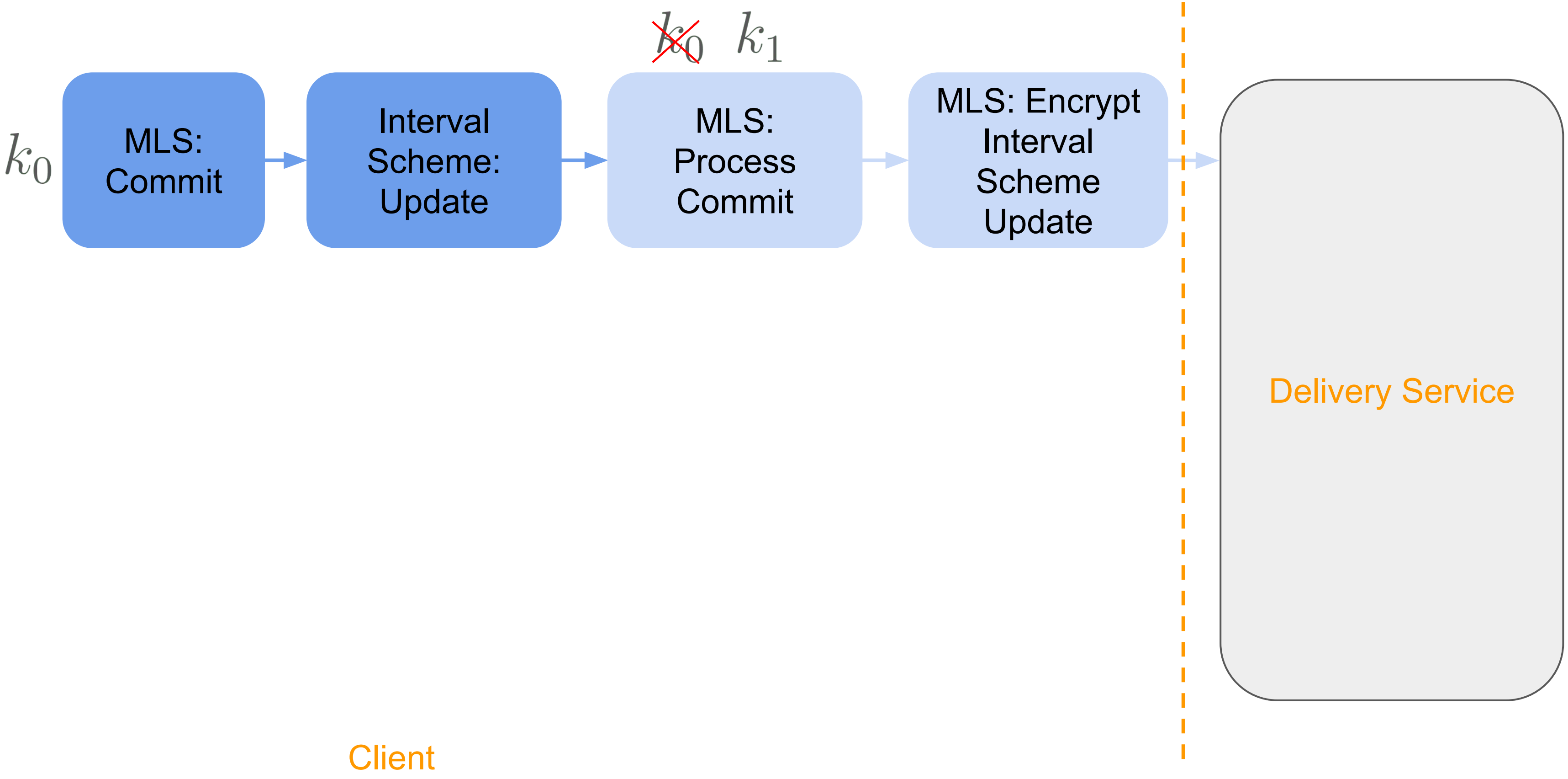
Grappa = MLS + Interval Scheme ⇨ Non Atomic Updates



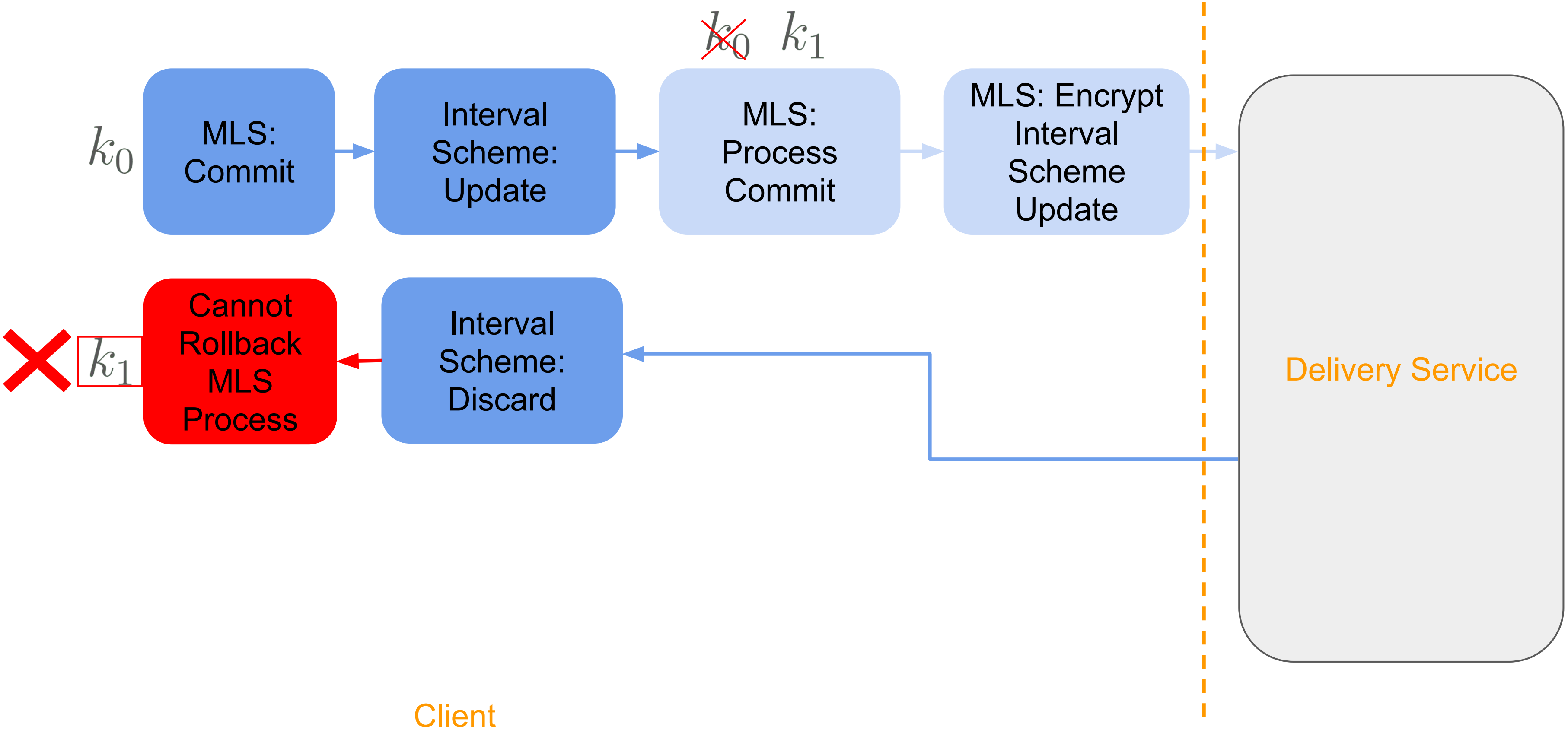
Grappa = MLS + Interval Scheme ➪ Non Atomic Updates



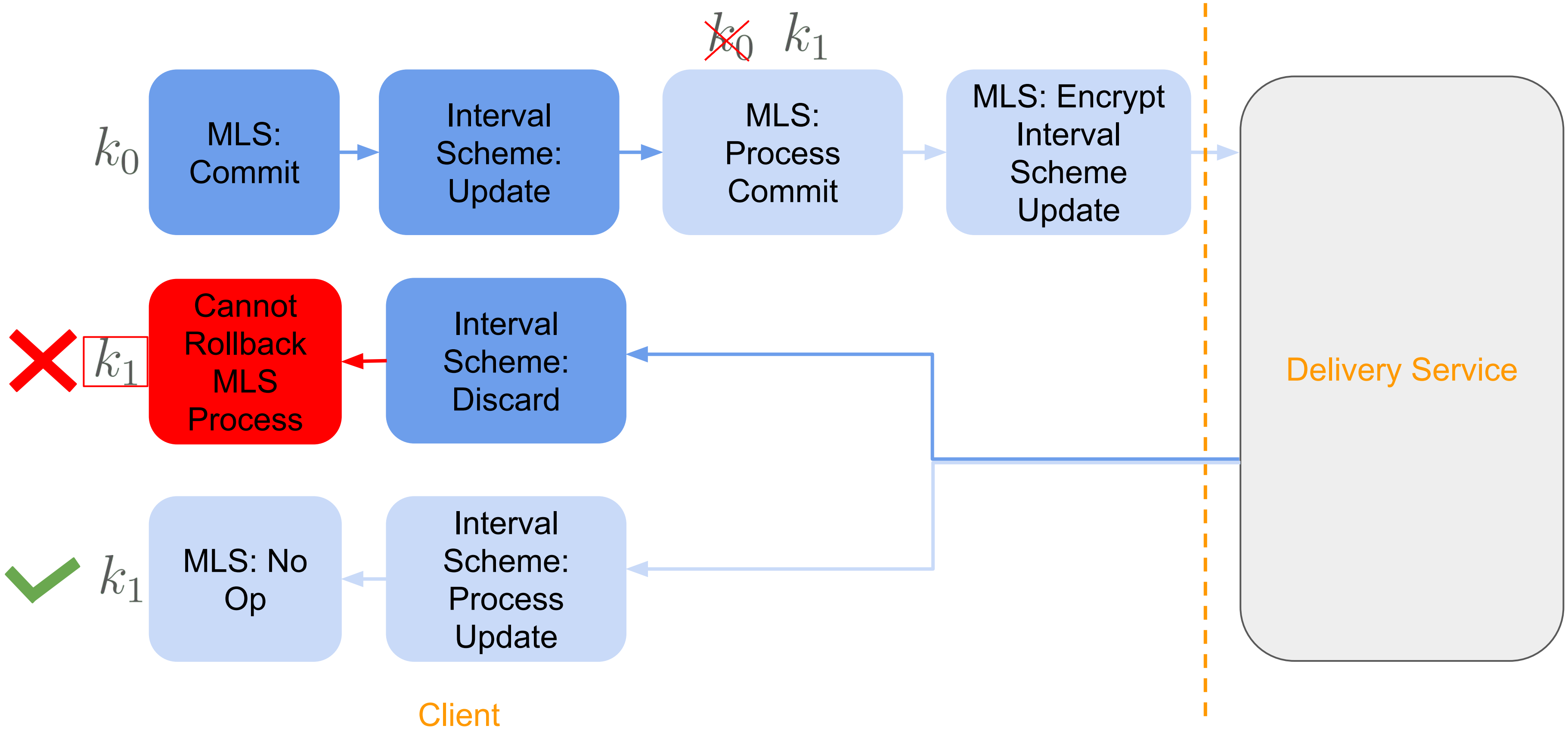
Grappa = MLS + Interval Scheme ➪ Non Atomic Updates



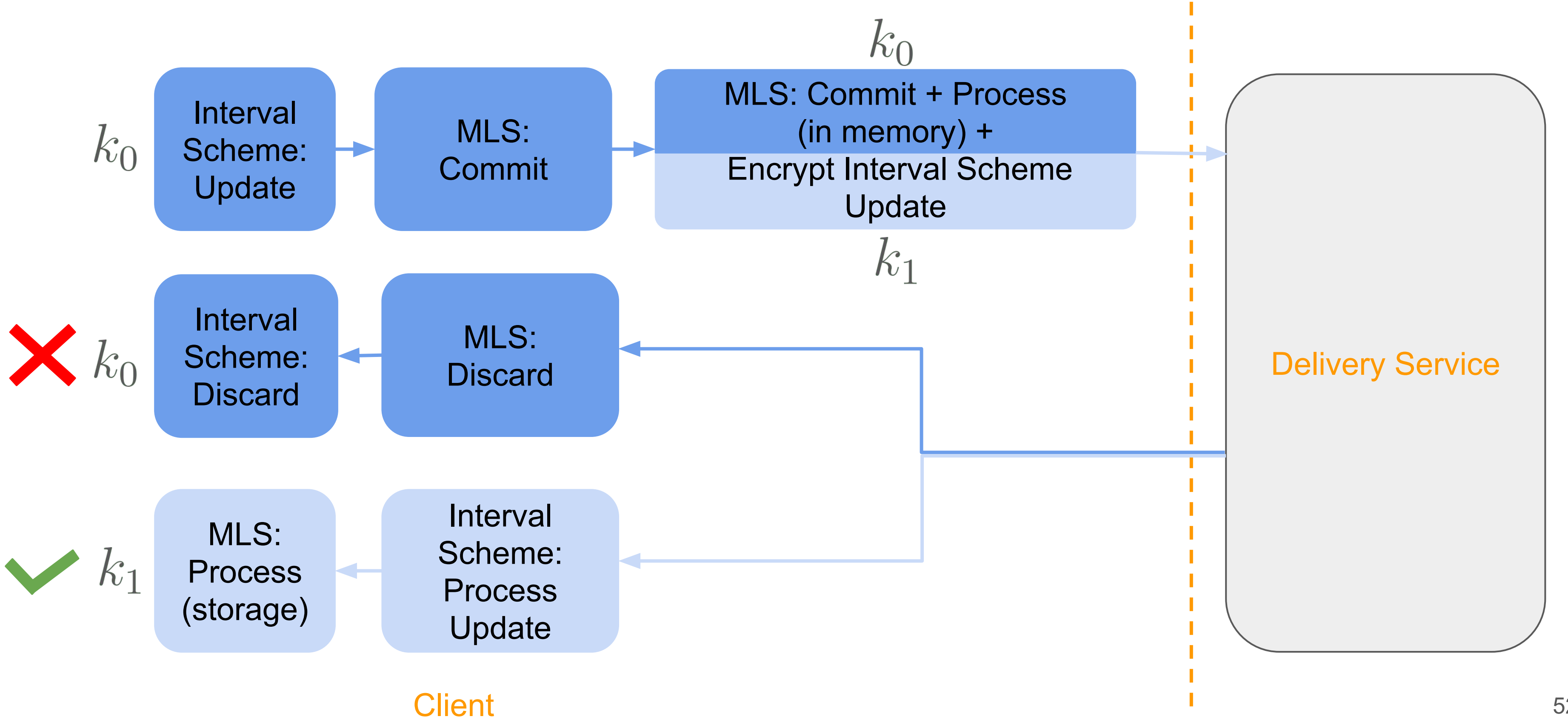
Grappa = MLS + Interval Scheme ➪ Non Atomic Updates



Grappa = MLS + Interval Scheme ➪ Non Atomic Updates

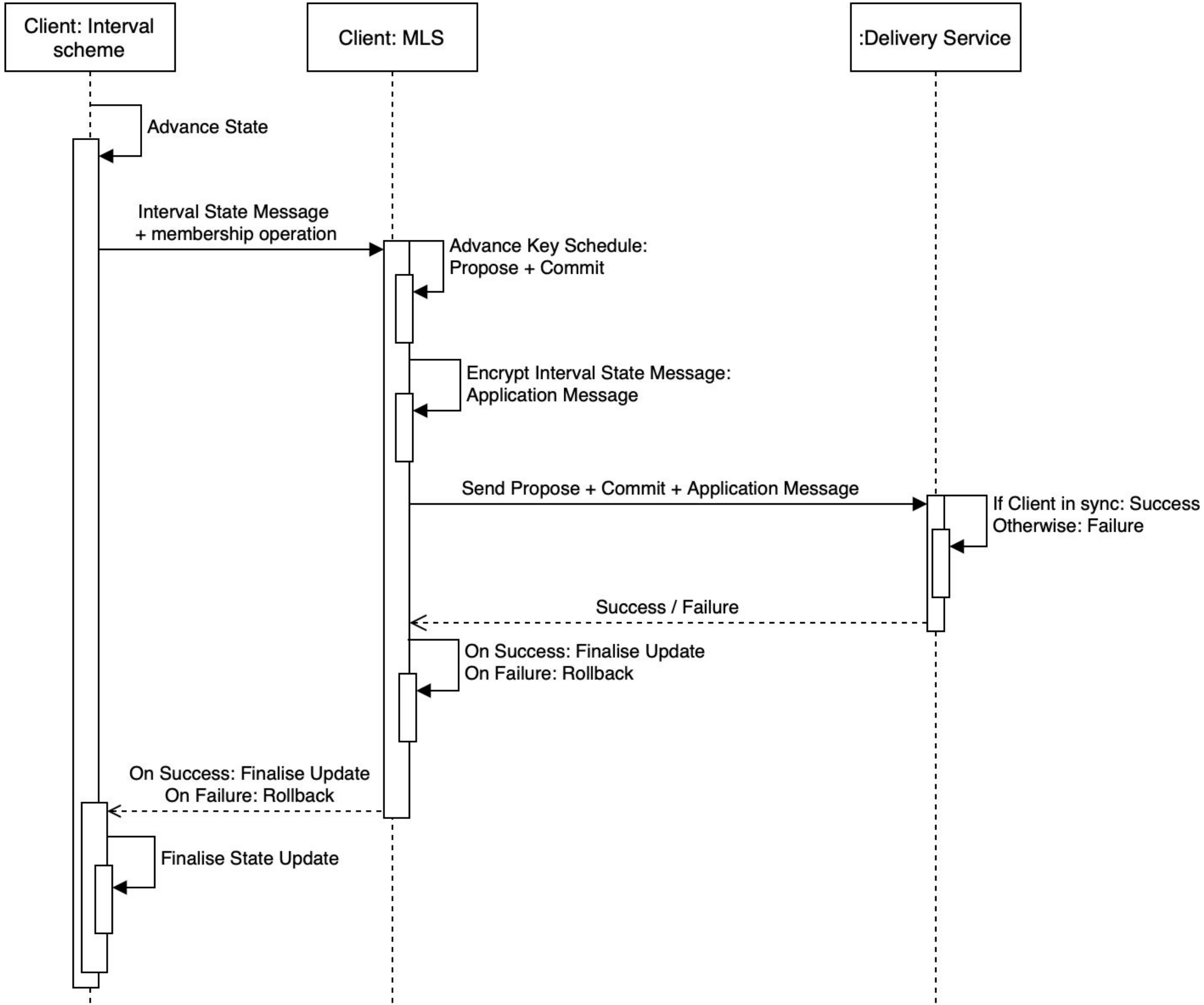


MLS as a Transport Layer ➪ Grappa Atomic Updates



Takeaways

- We introduce **Grappa** for dynamic groups of users to agree on a key progression for **persistent data**:
  - Builds on CGKA and Interval Scheme
  - Advanced security guarantees
  - **Secure Shared Folders** as a real-world application
- We implement Grappa for browser and desktop:
  - Relied on existing MLS implementation and WebCrypto API standard
  - Cryptography in JS environments is lagging behind
- MLS as transport layer to construct new protocols:
  - Gap between security modeling and functionality


Client: Interval scheme
Client: MLS
:Delivery Service
Advance State
Interval State Message
+ membership operation
Advance Key Schedule:
Propose + Commit
Encrypt Interval State Message:
Application Message
Send Propose + Commit + Application Message
If Client in sync: Success
Otherwise: Failure
Success / Failure
On Success: Finalise Update
On Failure: Rollback
On Success: Finalise Update
On Failure: Rollback
Finalise State Update

```rust
#[cfg(not(feature = "node"))]
async fn import_with_public_info(&self, crypto: &SubtleCrypto, key: &Uint8Array, params: &EcKeyImportParams, key_usages: &Array) -> Result<Promise, JsValue> {
    crypto.import_key_with_object(self.format(), &key, &params, true, &key_usages)
}

#[cfg(feature = "node")]
async fn import_with_public_info(&self, crypto: &SubtleCrypto, bytes: &Uint8Array, params: &EcKeyImportParams, key_usages: &Array) -> Result<Promise, JsValue> {
    let crypto_key_promise = crypto.import_key_with_object(self.format(), bytes, params, true, key_usages)?;
    let crypto_key = JsFuture::from(crypto_key_promise).await?.into();
    // Export the key to jwk to force the generation of the public key.
    let jwk_promise = crypto.export_key(&"jwk", &crypto_key)?;
    let jwk  = JsFuture::from(jwk_promise).await?;
    // Re-import into the original requested format with the same usages from jwk.
    crypto.import_key_with_object("jwk", &jwk.into(), params, true, &key_usages)
}

```