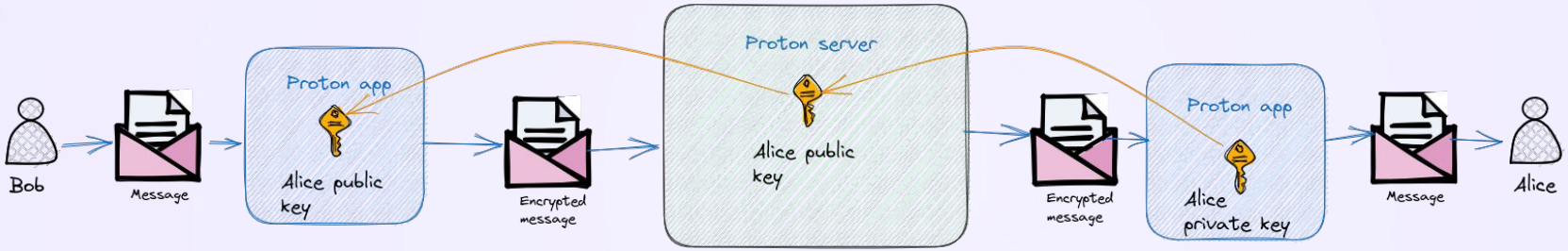
Key Transparency in Proton Mail

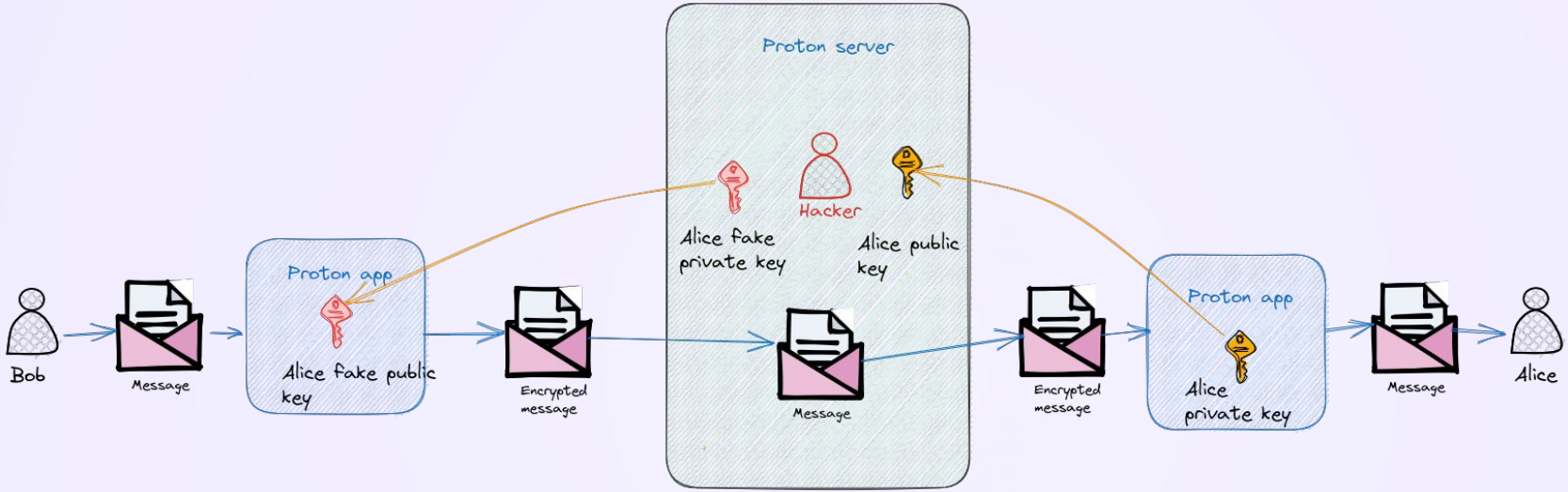
Transparency Summit

Proton | Privacy by default

Proton's End-to-End Encryption

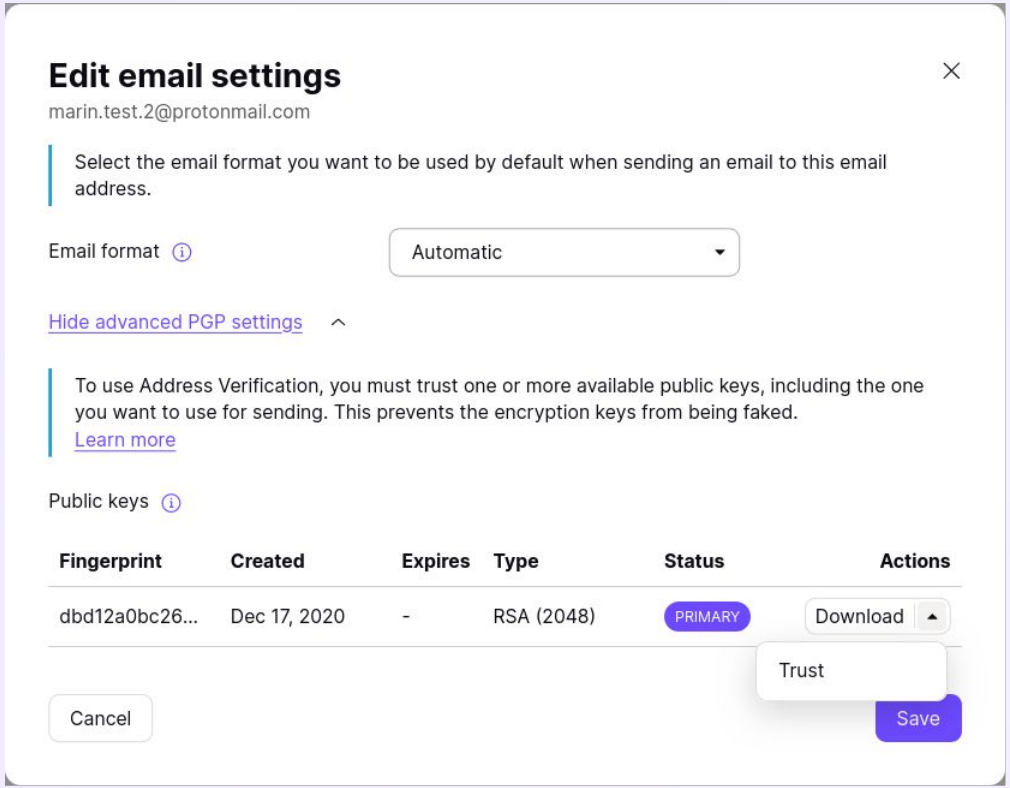


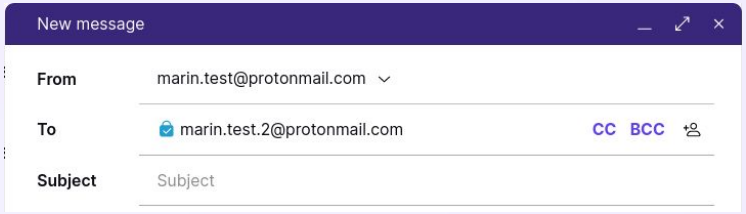
Problem: Man-in-the-middle attack

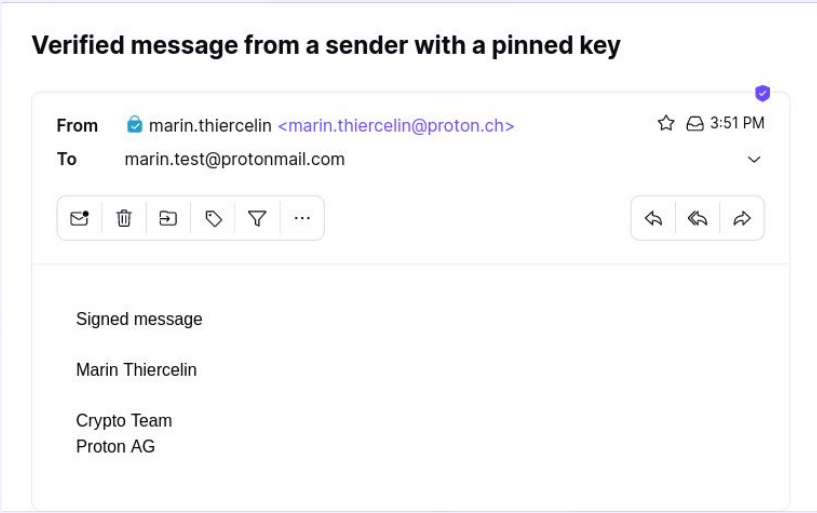


Key Pinning

Manual verification of keys





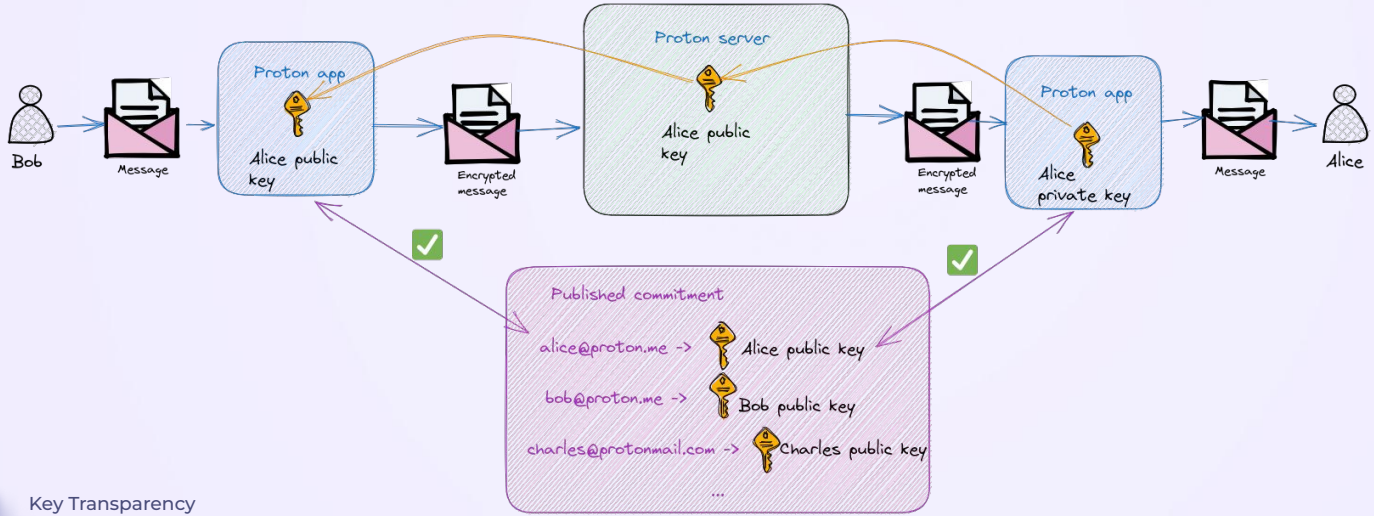


Key Transparency

Automatic verification of keys

High-level protocol:

1. The server **publishes all keys** for each Proton Mail address
2. The user **verifies that their own keys matches** what has been published for their address
3. When sending a message, the user **verifies that the recipient public keys matches** what has been published for the recipient address



History of Key Transparency at Proton

- 2018: Started working on KT based on CONIKS
- 2023: Thore Göbel from ETHZ brought in some ideas from SEEMless and Parakeet
- 2023: Launched in beta (opt-in)
- 2024: Whitepaper published

Publishing the list of keys

Creating an epoch

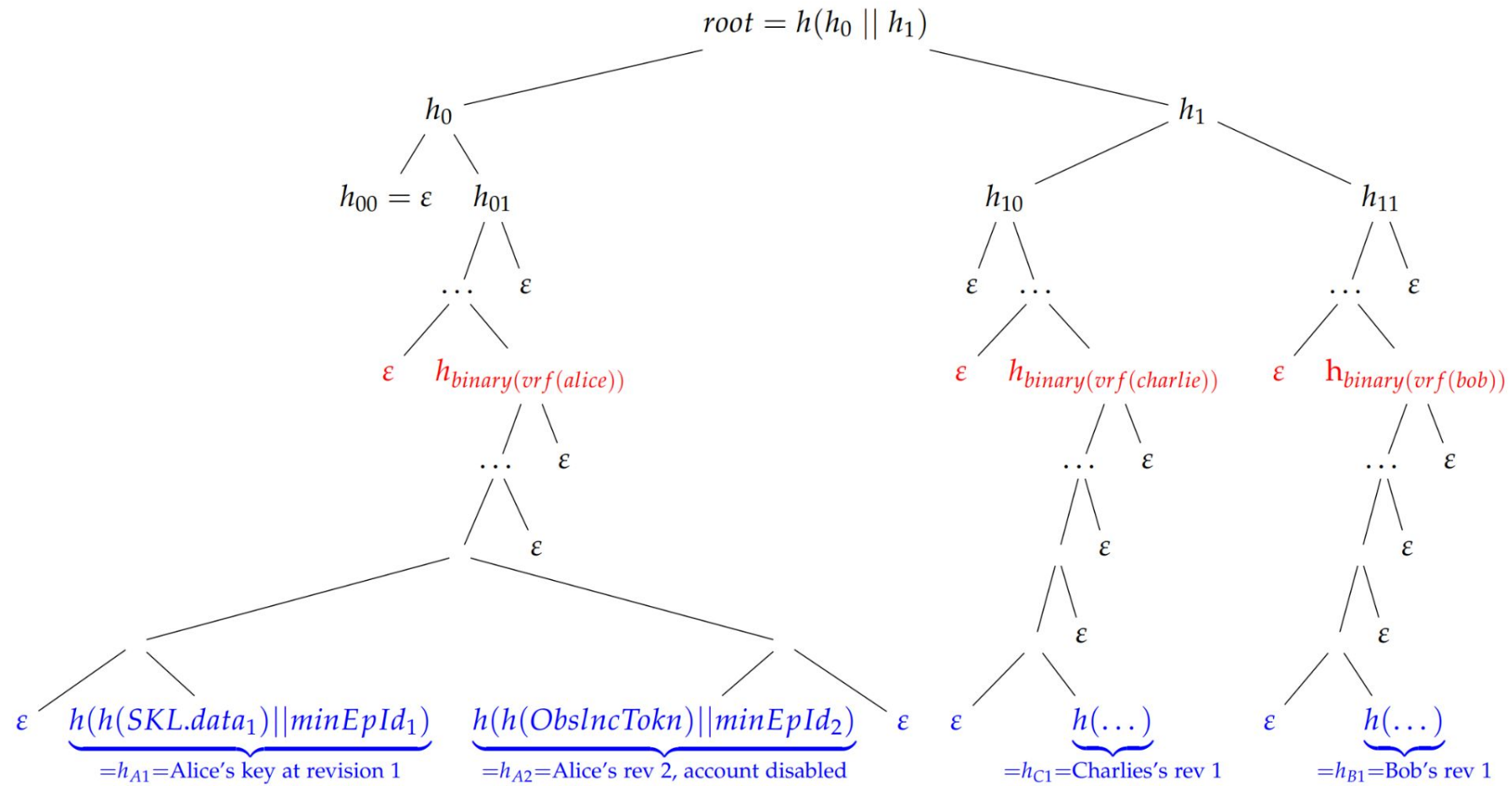
High-level protocol:

1. The server **publishes all keys** for each Proton Mail address
2. The user **verifies that their own keys matches** what has been published for their address
3. When sending a message, the user **verifies that the recipient public keys matches** what has been published for the recipient address

Publishing all keys

Every 4 hours:

- Build a Merkle tree of the keys of every email address at every revision
- Compute tree root hash
- Compute chain hash (hash of root hash and previous chain hash)
- Request certificate for `{chainhash[0:32]}.{chainhash[32:64]}.{issuanceTime}.{epochid}.1.keytransparency.ch`
- Certificate gets published to Certificate Transparency logs


$root = h(h_0 \,||\, h_1)$
h_0
h_1
$h_{00} = \varepsilon$
h_{01}
h_{10}
h_{11}
$h_{binary(vrf(alice))}$
$h_{binary(vrf(charlie))}$
$h_{binary(vrf(bob))}$
$h(h(SKL.data_1)||minEpId_1)$
$h(h(ObslncTokn)||minEpId_2)$
$h(\ldots)$
$h(\ldots)$
$=h_{A1}$=Alice's key at revision 1
$=h_{A2}$=Alice's rev 2, account disabled
$=h_{C1}$=Charlies's rev 1
$=h_{B1}$=Bob's rev 1

Auditing process

Auditors have to check:

- There is only one root hash per epoch ID in CT logs
- The epochs form a consistent chain
- Subtrees are append-only (except for stale entries)

Verifying the user own keys

The self audit

High-level protocol:

1. The server **publishes all keys** for each Proton Mail address
2. The user **verifies that their own keys matches** what has been published for their address
3. When sending a message, the user **verifies that the recipient public keys matches** what has been published for the recipient address

The self audit procedure

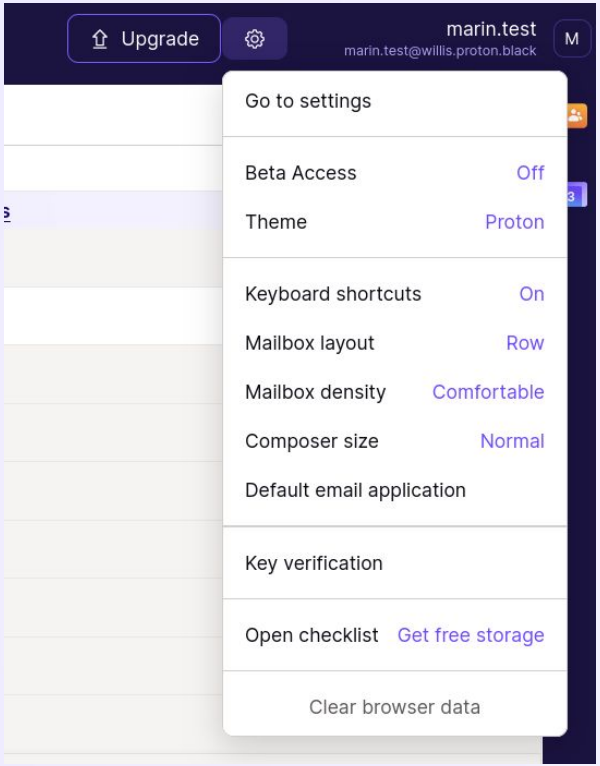
At regular intervals:

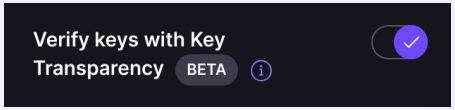
1. Fetch the latest epoch & verify the certificate
2. Check that the public key fingerprints in the latest revision match the private keys
3. Check that previous modifications were included in the epoch
4. Check that additional included revisions are signed

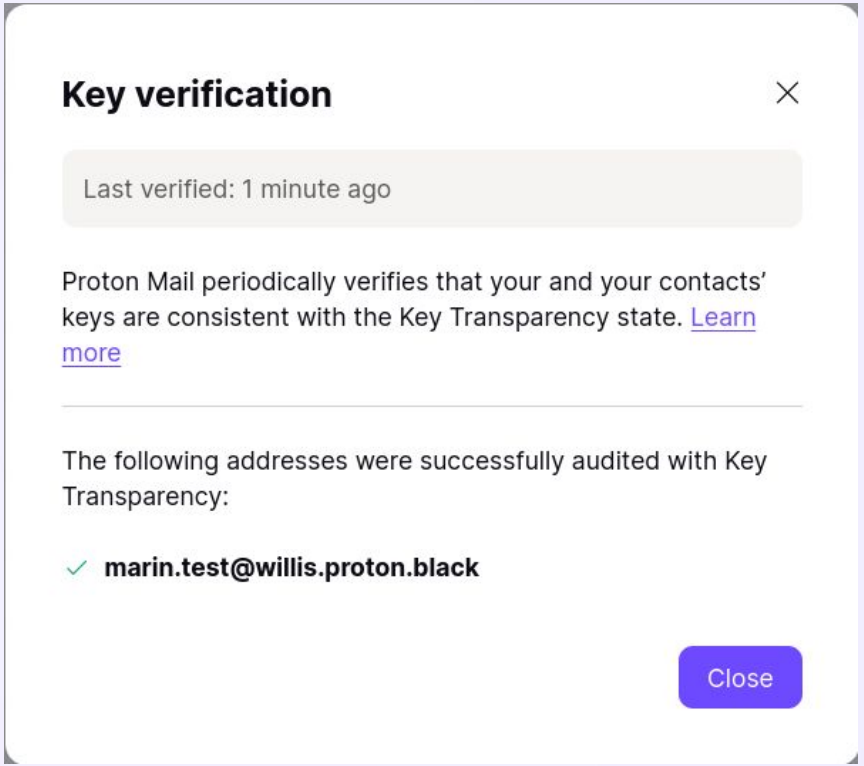
Signed Key List

```
{
  Data: [
    {
      Primary: true,
      Flags: 0,
      SHA256Fingerprints: ["0123ABCD...", "4567EFAB..."]
    },
    {
      Primary: false,
      Flags: 1,
      SHA256Fingerprints: ["8901CDEF...", "2345ABCD..."]
    },
    ...
  ],
  Signature: Sign(PrimaryKey, JSON.stringify(Data))
}
```

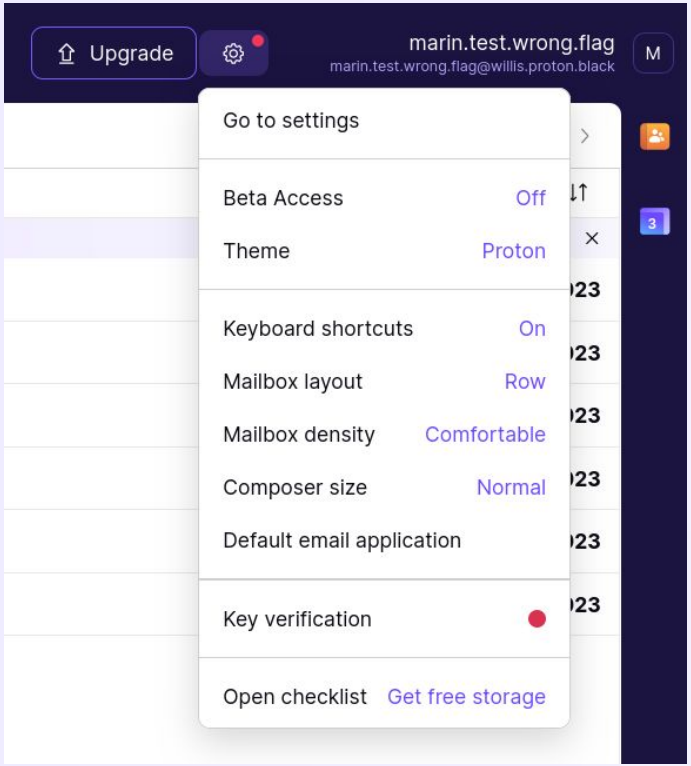
Self audit user interface

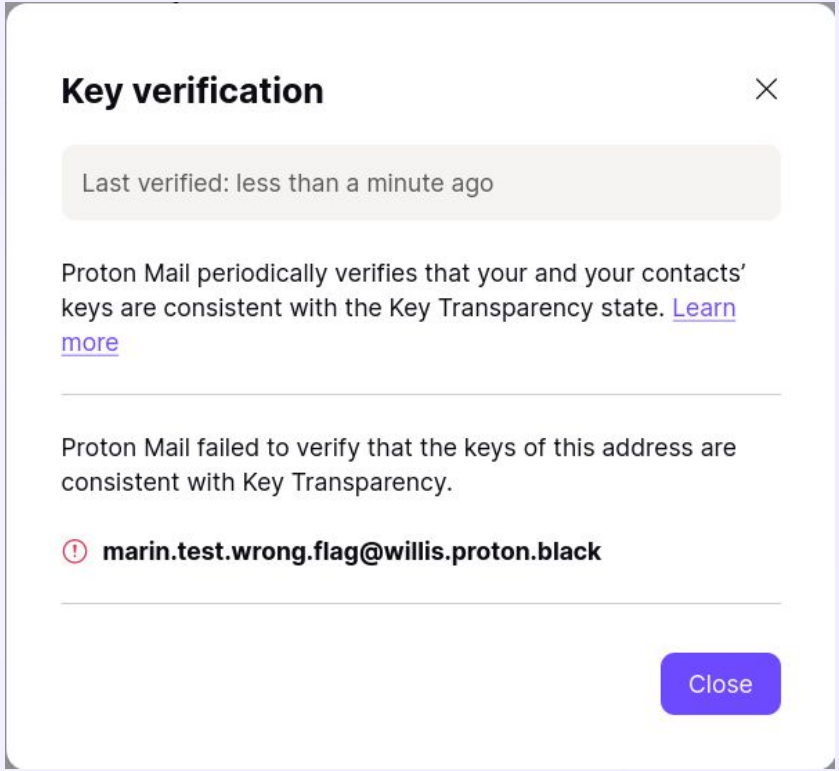






Self audit user interface





Verifying the public keys of other users

High-level protocol:

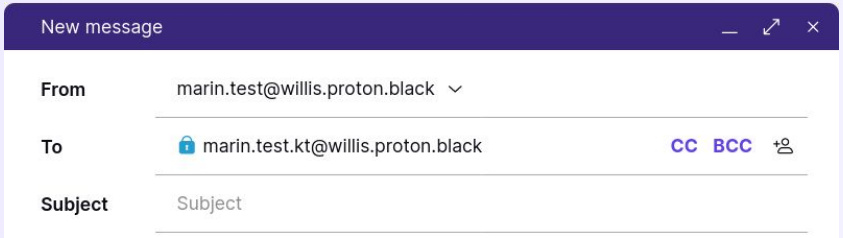
1. The server **publishes all keys** for each Proton Mail address
2. The user **verifies that their own keys matches** what has been published for their address
3. When sending a message, the user **verifies that the recipient public keys matches** what has been published for the recipient address

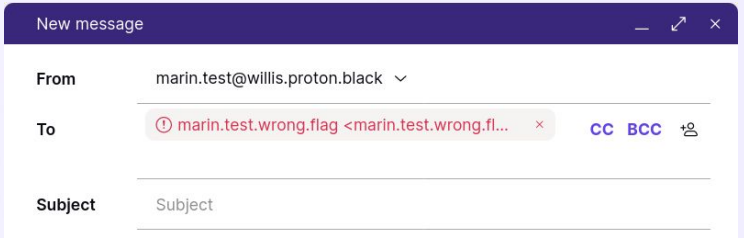
Verifying public keys from the server

In the composer

When Alice wants to write to bob@proton.me :

1. The client asks for the public keys for bob@proton.me
2. Fetch the latest epoch & verify the certificate
3. Check that the epoch includes the public keys





Special case: the recipient keys changed recently

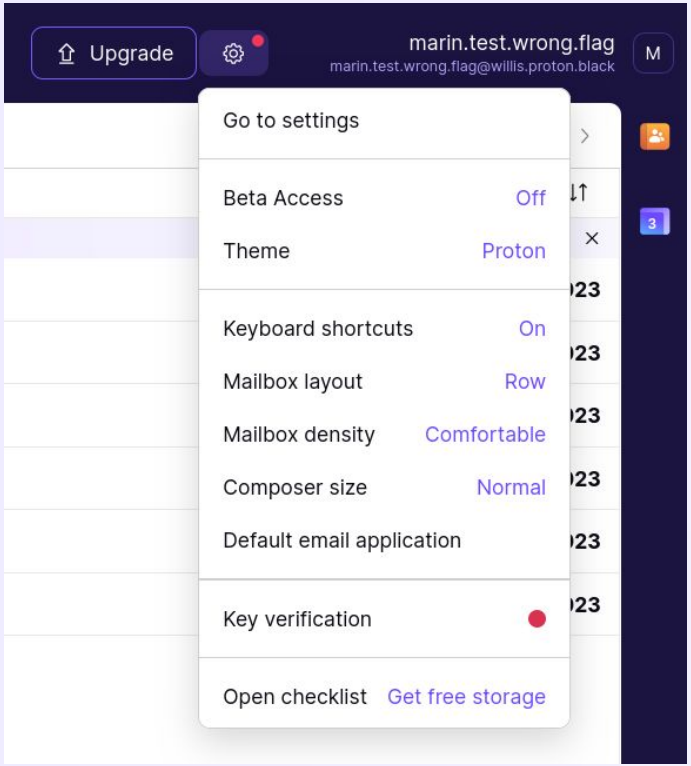
Asynchronous verification

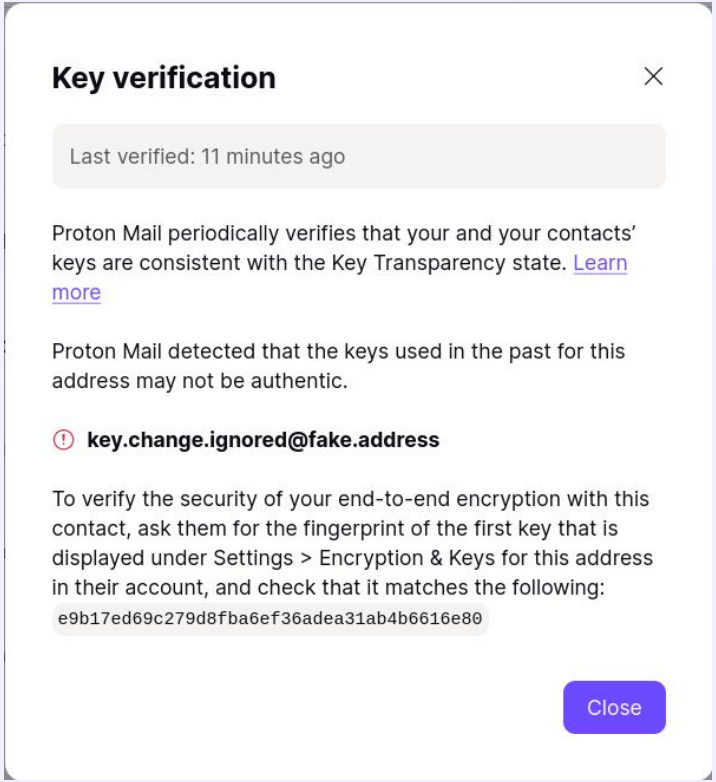
If Bob's current keys are not in KT yet:

1. Alice's client accepts the keys without verifying the proof
2. Alice's client stores the keys in local storage
3. At the next periodic audit, Alice's client verifies that the new keys are included in the next epoch

When the asynchronous verification fails

The server might have lied

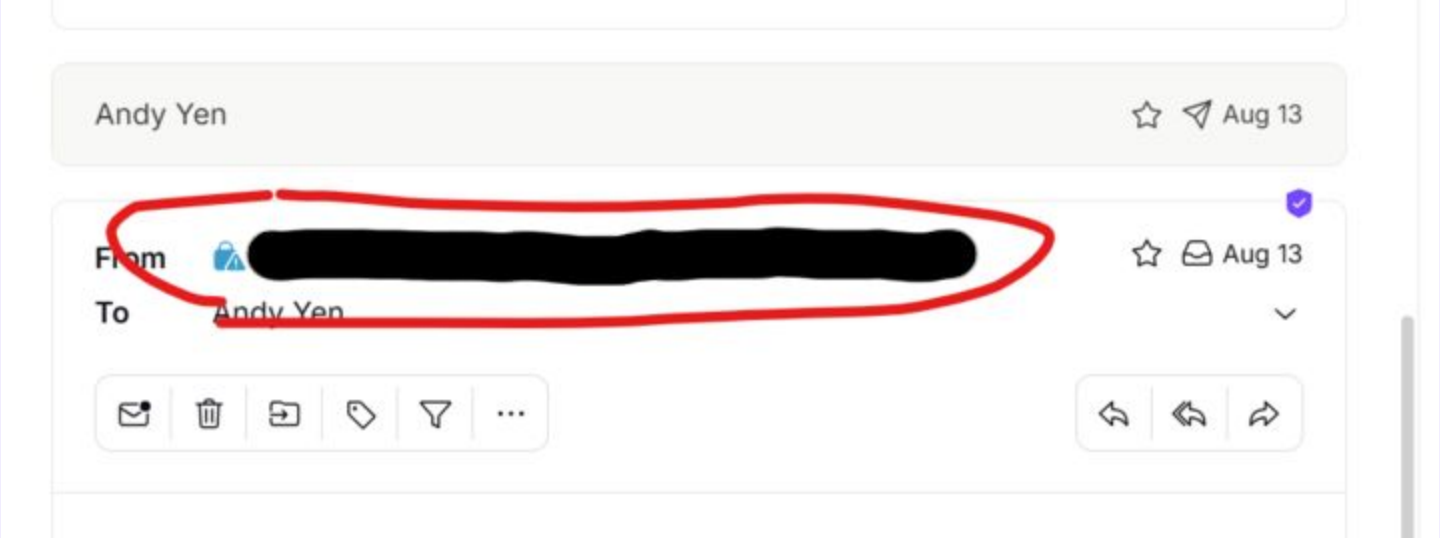




Other Edge Cases

Deleted & Transferred Accounts

Account ownership might have changes



Sampling of Other Issues

Random Sentry errors we've gotten

S	sentry@protontech.ch	...WEB-MAIL-3TQ9 - [KeyTransparency] Error: Error signing message: Could not find valid self-signature in key 70a65c...
S	sentry@protontech.ch	...WEB-MAIL-3TP6 - [KeyTransparency] SyntaxError: Invalid character: '\0'
S	sentry@protontech.ch	...WEB-MAIL-3TNT - [KeyTransparency] TypeError: Cannot create property '10' on number '9'
S	sentry@protontech.ch	...WEB-MAIL-3TM6 - [KeyTransparency] No keys detected
S	sentry@protontech.ch	...WEB-MAIL-3TJJ - [KeyTransparency] TypeError: Right side of assignment cannot be destructured
S	sentry@protontech.ch	...WEB-MAIL-3TJA - [KeyTransparency] Error: Epoch certificate alternative name does not match
S	sentry@protontech.ch	...WEB-MAIL-3TH6 - [KeyTransparency] StatusCodeError: Not Found
S	sentry@protontech.ch	...WEB-MAIL-3TGM - [KeyTransparency] Error: Minified Redux error #3; visit https://redux.js.org/Errors?code=3 for th...
S	sentry@protontech.ch	...WEB-MAIL-3TFE - [KeyTransparency] SyntaxError: Unexpected token '<', "<!DOCTYPE "... is not valid JSON

More details

https://proton.me/files/proton_keytransparency_whitepaper.pdf

Thanks!

Any questions?

Proton | Privacy by default

The signed key list: a text representation of keys

Instead of publishing a list of keys for each address, the server publishes a text representation called the Signed Key List.

It is a JSON string with information about the keys of the address, signed by the user client with PGP.


```
[
    {
        "Primary": 1,
        "Flags": 3,
        "Fingerprint": "25bab8444b7ac056eecdf9c458eaaa73e09a8837",
        "SHA256Fingerprints": [
            "ed6905dfeb980c71d479272ba28b3655ee7c50e5525e02b1ef604632585e3781",
            "80943ea7d8f893139cb40be9c88966445b90fdd114d2b5712a9f4e11d19e3325"
        ]
    },
    {
        "Primary": 0,
        "Flags": 3,
        "Fingerprint": "c86cad61bb292bf924b3473f0500b430f41f19de",
        "SHA256Fingerprints": [
            "1cc6f1246bcd1047b0d05f6cc01aae8302602e341c69ea744d5c9bb3c24c6f6c",
            "fd629daa8a3640e1f2cda4cb1c3905492fa3d0720cbeddf3332b7e6b299e64df"
        ]
    }
]
```

Verifiable random function (VRF)

An asymmetric keyed hash

Usual hash functions:

- hash = Hash.hash(email)

Verifiable random functions

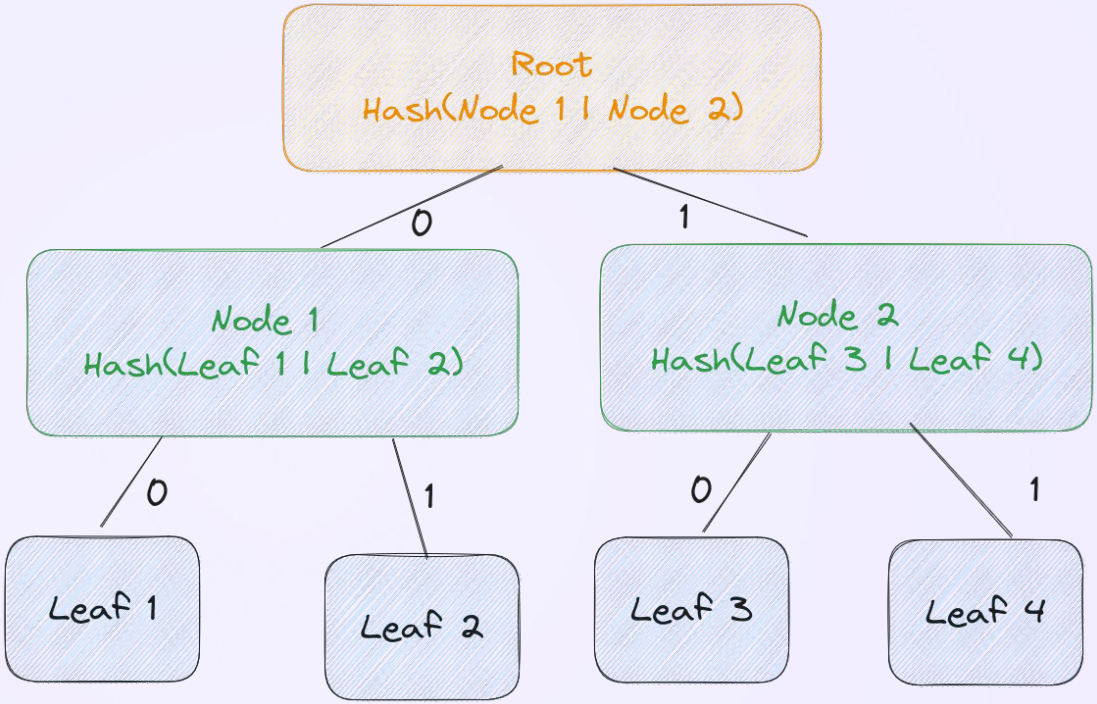
- (hash, proof) = VRF.hash(email, secret_key)
- yes / no = VRF.verify(email, hash, proof, public_key)

Regular hash functions can be computed by anyone, VRFs can only be computed by someone with the secret_key.

Merkle tree

A single hash for a list of values

- Each leaf has an assigned “path” in the binary tree
- The server can efficiently prove that a given value is at the right path



Putting it all together: Creating an Epoch

Periodically, the server releases an Epoch

1. Make a snapshot of the address keys DB
2. For each address, compute the vrf hash: (hash, proof) = Vrf.hash(address.Email)
3. For each address compute the leaf value: leaf = Hash(address.SKL)
4. Create a merkle tree with all leaves, where the path of each leaf is derived from the VRF hash.
5. Publish the root of the merkle tree

01
Guide Lines
Graphics Light Background

Square Text with shadow

Square Circle Text

with shadow

02

Guide Lines Graphics Light Background

Top Line Light

Image with shadow

No shadow

Bottom Line Light

06
Guide Lines
Graphics Dark Background

06

Guide Lines
Graphics Light Background

Proton

Proton

Proton Mail

Proton Calendar

Proton Drive

Proton VPN

Proton Bridge

Proton Pass

Proton

Proton

Proton Mail

Proton Calendar

Proton Drive

Proton VPN

Proton Bridge

Proton Pass

05
Logo Guide Lines

Small

Medium

Large