End-to-end encryption
for things that matter.

Keybase is secure messaging
and file-sharing.

Keybase File System

Presenter: Yingjie Xue

Date: 11/10/2020

Outline

- Problems addressed
- Approaches(At high-level)
- Detailed System Design

Problems Addressed

- Similar problem that PGP Web of trust tackled with: Is this public key trustable?

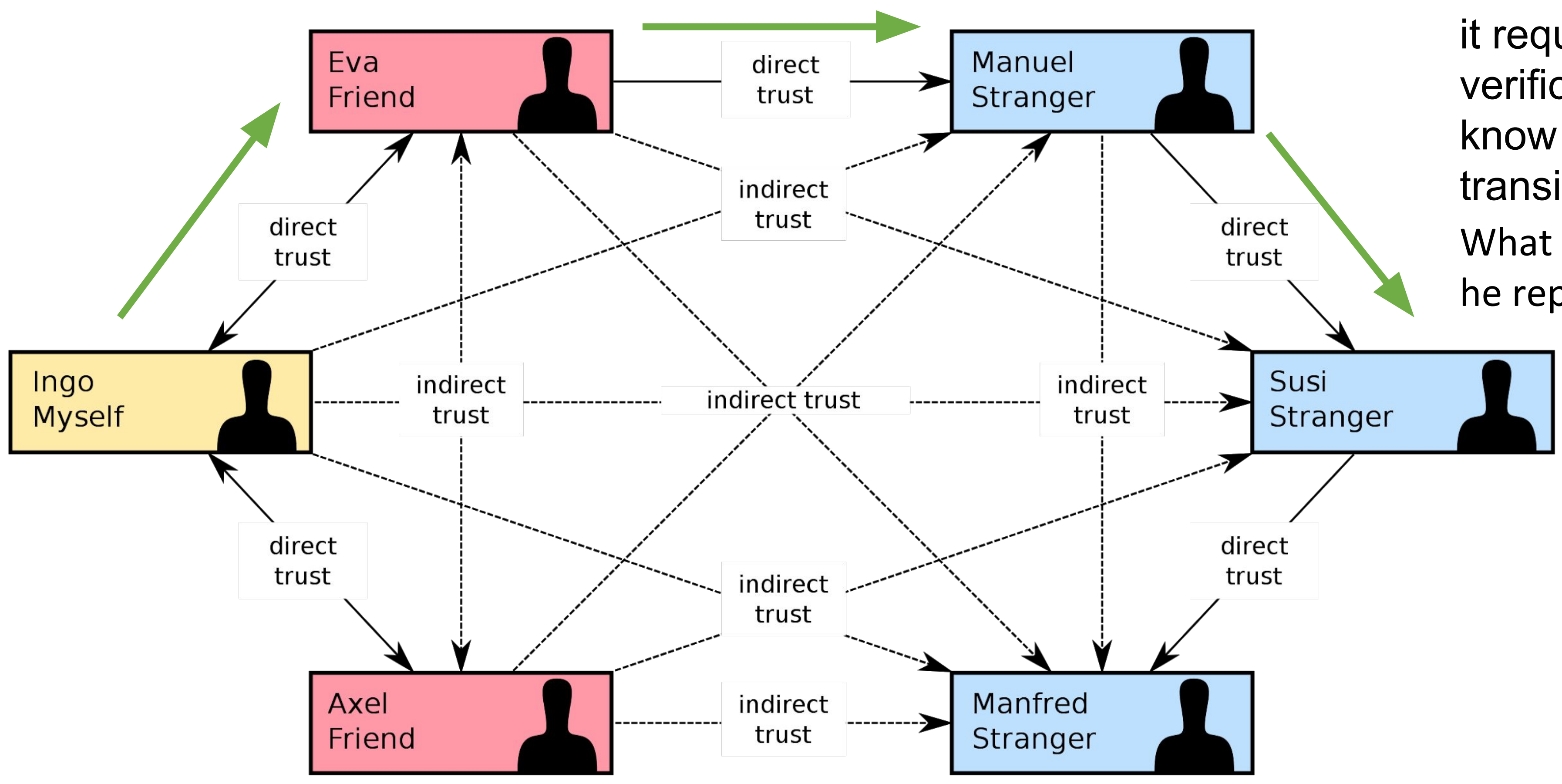


Figure 1: PGP web of trust

it requires in-person verifications, and it's hard to know what trust level to assign transitively.

What if Manuel is drunk when he reports Susi has pubkey_1?

Three kinds of Attacks

- Server DDOS’ed
- Server compromised:
  - attacker corrupts server-side code and keys to send bad data to clients
- Server compromised:
  - attacker distributes corrupted client-side code [by Open API and state that the open-source client is simply a *reference client and signing update to client]*

Key Components

- **Identity proofs**: "I am Joe on Keybase and MrJoe on Twitter"
- **Follower statements**: "I am Joe on Keybase and I just looked at Chris's identity"
- **Key ownership**: "I am Joe on Keybase and here's my public key"
- **Revocations**: "I take back what I said earlier"
- Write the Merkle hash root of signatures into Stellar Blockchain

The goal of Keybase:

Provide public keys that can be trusted

- **Identity proofs**: "I am Joe on Keybase and MrJoe on Twitter”.
  - *Effect:* Two accounts are controlled by the same person.
- **Follower statements**: "I am Joe on Keybase and I just looked at Chris's identity"
  - *Effect:* Enable transitive trust and do not need to verify every time.
- **Key ownership**: "I am Joe on Keybase and here's my public key”.
- **Revocations**: "I take back what I said earlier”.

Questions

Suppose Joe is your friend, which component(s) do you use to verify that the username “joe” on keybase is your friend Joe? And public_key that joe claims to have belongs to joe?

What is the underlying assumption for this approach to be valid?

[**Identity proof example**]

Organization of statements: Sigchain

- Every Keybase account has a **public signature chain** (*sigchain*)
  - An **ordered list** of statements about how the account has changed over time.
  - Signing a new statement must ***sign the hash of previous sigchain***.
- Follow someone, add a key, or connect a website, your client signs a new statement (a *link*) and publishes it to your sigchain.

Sigchain is the core to prevent corrupted data, e.g. rollback

Attack 1: Server DDOS’ed

- Server is not accessible;

- Keybase claimed to be able to defend this attack, but indeed it cannot.

Attack 2: Attacker corrupts server-side code and keys to send bad data to clients

- Q: Can an attacker
    - Add fake update data?
    - Roll back data?

- *A: Rollback attack can be resisted, if you cached the previous data.*
- *All user signature chains grow monotonically and its hash is written to Stellar, thus it can never be "rolled back”.*

Question(cont’d):

- What if you did not cache it (i.e. No cache and no blockchain)? Can you detect the data is corrupted?
    - Your own data is corrupted:?
    - Others’ data is corrupted:?

Attack 2 cont’d: Omission

- Whenever a user posts an addition to a signature chain, the site must sign and advertise a change in global site state.

- Q: Can it be resisted/detected if server is compromised?

A: Clients proactively check if their new signature is appended.

Attack 2 cont’d :Fork two different views to user_a and user_b

- Send two different merkle roots hash to user_a and user_b.
- Can it be resisted/detected?
  - Without stellar blockchain?
  - With stellar blockchain?

To sum up

- The integrity of public keys and signatures highly rely on two parts besides sigchain:
  - **Keybase Client Integrity**
  - **Merkle root hash written to Stellar Blockchain**

Keybase Client Integrity

- Check the integrity of user signature chains,
- Find evidence of malicious rollback(how?),
- Alert Alice when her following of Bob breaks (how?),
- Check the site's published Merkle tree root for consistency against known signature chains(how?)[demo follows],
- Sign proofs when all these checks complete, setting up known safe checkpoints to hold the server accountable to in the future.

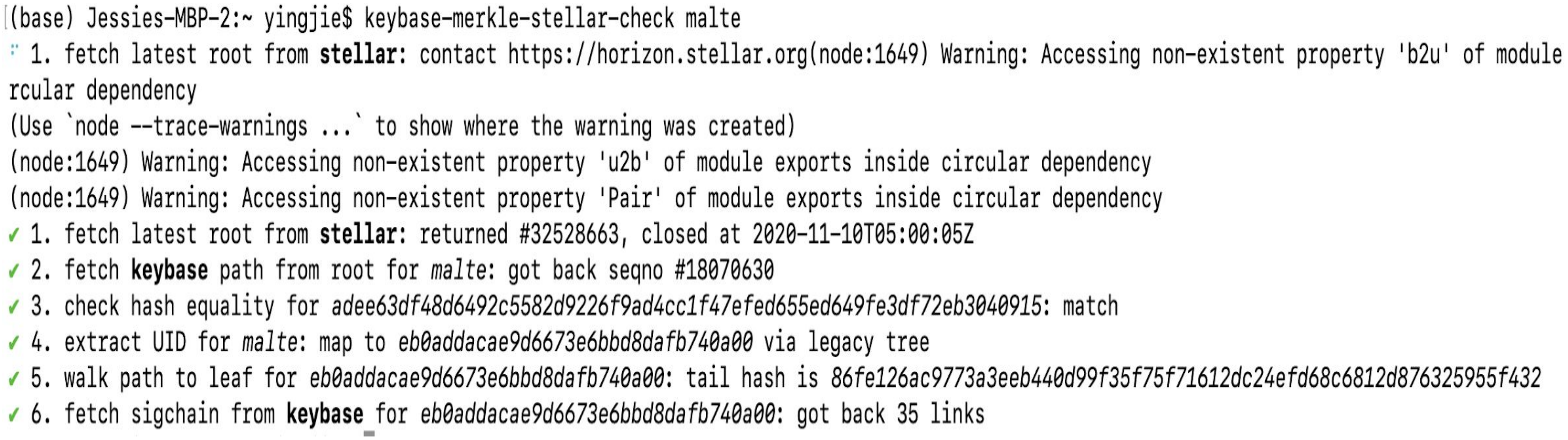
Merkle tree root hash written to Stellar

- All of signatures by users are signed by Keybase and hashed into the Stellar blockchain:
  - ✔ announcing your Keybase username
  - ✔ adding a public key
  - ✔ identity proofs (twitter, github, your website, etc.)
  - ✔ public bitcoin address announcements
  - ✔ public follower statements
  - ✔ revocations
  - ✔ team operations

Check sigchain

- fetchLatestGroveHashFromStellar — Fetch the last groveHash that Keybase published into Stellar
- fetchPathAndSigs — Fetch from Keybase the signatures over the grove that matches the hash we just got from Stellar
- checkSigAgainstStellar — Open the signature, verify it against Keybase's known signing key, and check that the hash of this signature matches what we got from Stellar.
- walkPathToLeaf — Walk down a path to the leaf for the user
- fetchSigChain — The user's leaf gives us the tail of her sigchain. Fetch the whole chain to check it matches the leaf in the tree.

Demo

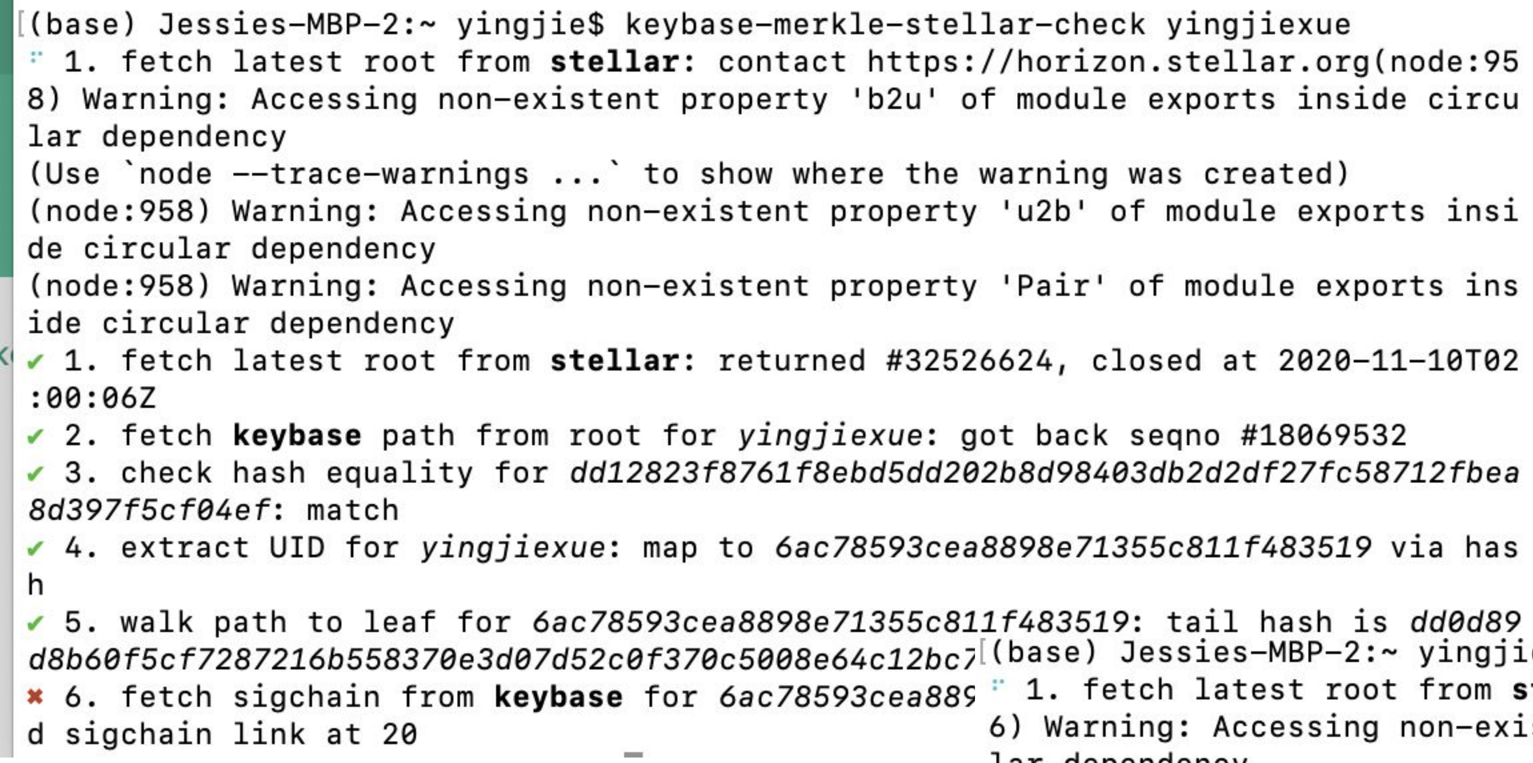


```
(base) Jessies-MBP-2:~ yingjie$ keybase-merkle-stellar-check malte
⠋ 1. fetch latest root from stellar: contact https://horizon.stellar.org(node:1649) Warning: Accessing non-existent property 'b2u' of module e
rcular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:1649) Warning: Accessing non-existent property 'u2b' of module exports inside circular dependency
(node:1649) Warning: Accessing non-existent property 'Pair' of module exports inside circular dependency
✓ 1. fetch latest root from stellar: returned #32528663, closed at 2020-11-10T05:00:05Z
✓ 2. fetch keybase path from root for malte: got back seqno #18070630
✓ 3. check hash equality for adee63df48d6492c5582d9226f9ad4cc1f47efed655ed649fe3df72eb3040915: match
✓ 4. extract UID for malte: map to eb0addacae9d6673e6bbd8dafb740a00 via legacy tree
✓ 5. walk path to leaf for eb0addacae9d6673e6bbd8dafb740a00: tail hash is 86fe126ac9773a3eeb440d99f35f75f71612dc24efd68c6812d876325955f432
✓ 6. fetch sigchain from keybase for eb0addacae9d6673e6bbd8dafb740a00: got back 35 links
```

How long to wait before update on Keybase written to Stellar?

- Keybase writes to Stellar once per hour;
- Stellar ledger update every five seconds

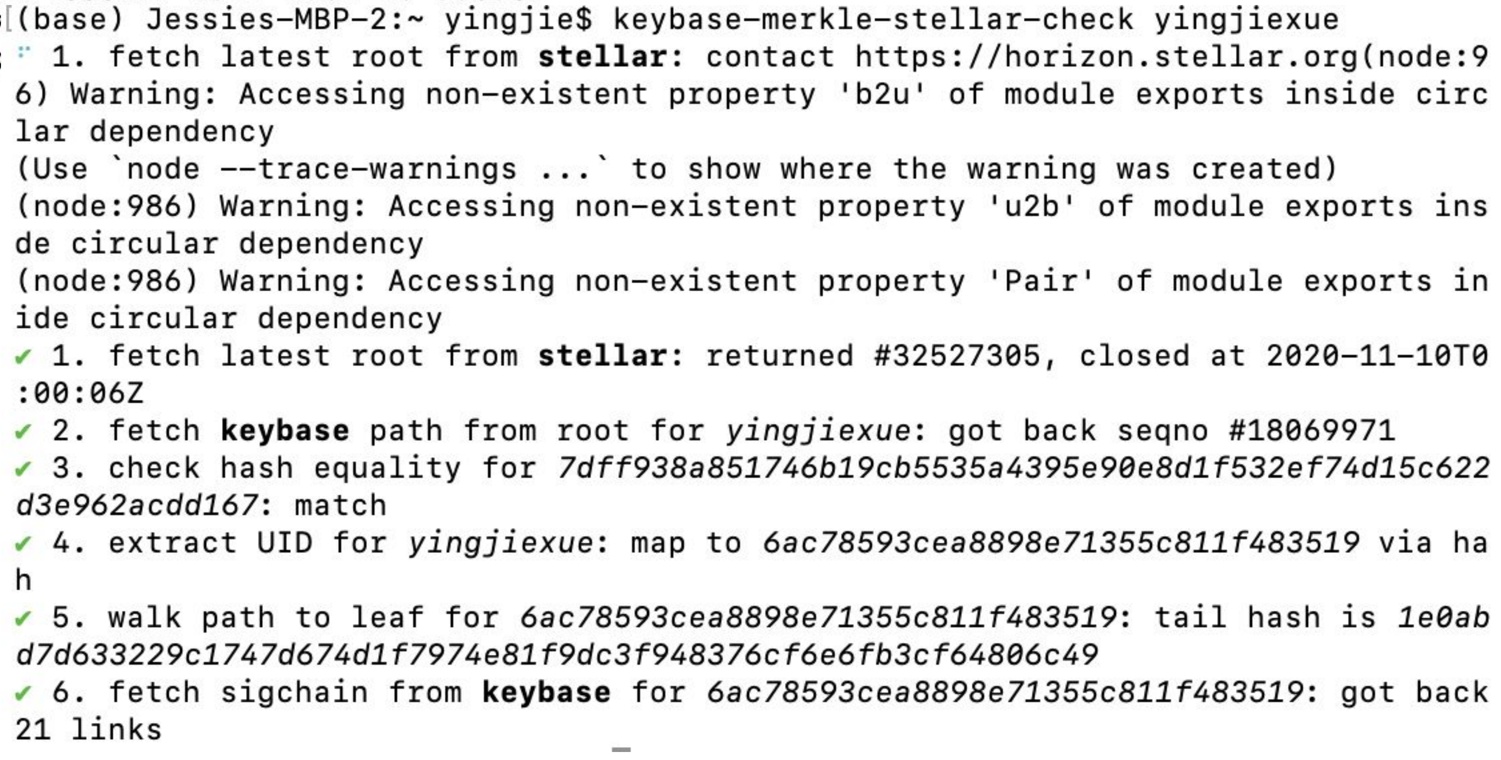
What happens between the two updates?



```
(base) Jessies-MBP-2:~ yingjie$ keybase-merkle-stellar-check yingjiexue
⠋ 1. fetch latest root from stellar: contact https://horizon.stellar.org(node:95
8) Warning: Accessing non-existent property 'b2u' of module exports inside circu
lar dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:958) Warning: Accessing non-existent property 'u2b' of module exports insi
de circular dependency
(node:958) Warning: Accessing non-existent property 'Pair' of module exports ins
ide circular dependency
✔ 1. fetch latest root from stellar: returned #32526624, closed at 2020-11-10T02
:00:06Z
✔ 2. fetch keybase path from root for yingjiexue: got back seqno #18069532
✔ 3. check hash equality for dd12823f8761f8ebd5dd202b8d98403db2d2df27fc58712fbea
8d397f5cf04ef: match
✔ 4. extract UID for yingjiexue: map to 6ac78593cea8898e71355c811f483519 via has
h
✔ 5. walk path to leaf for 6ac78593cea8898e71355c811f483519: tail hash is dd0d89
d8b60f5cf7287216b558370e3d07d52c0f370c5008e64c12bc7
✖ 6. fetch sigchain from keybase for 6ac78593cea889
d sigchain link at 20
```

11/09/2020 21:58 pm

11/09/2020 22:01 pm



```
(base) Jessies-MBP-2:~ yingjie$ keybase-merkle-stellar-check yingjiexue
⠋ 1. fetch latest root from stellar: contact https://horizon.stellar.org(node:9
6) Warning: Accessing non-existent property 'b2u' of module exports inside circ
lar dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:986) Warning: Accessing non-existent property 'u2b' of module exports ins
de circular dependency
(node:986) Warning: Accessing non-existent property 'Pair' of module exports in
ide circular dependency
✔ 1. fetch latest root from stellar: returned #32527305, closed at 2020-11-10T0
:00:06Z
✔ 2. fetch keybase path from root for yingjiexue: got back seqno #18069971
✔ 3. check hash equality for 7dff938a851746b19cb5535a4395e90e8d1f532ef74d15c622
d3e962acdd167: match
✔ 4. extract UID for yingjiexue: map to 6ac78593cea8898e71355c811f483519 via ha
h
✔ 5. walk path to leaf for 6ac78593cea8898e71355c811f483519: tail hash is 1e0ab
d7d633229c1747d674d1f7974e81f9dc3f948376cf6e6fb3cf64806c49
✔ 6. fetch sigchain from keybase for 6ac78593cea8898e71355c811f483519: got back
21 links
```

Check sigchain

- fetchLatestGroveHashFromStellar
- fetchPathAndSigs
- checkSigAgainstStellar
- walkPathToLeaf
- fetchSigChain

The user's leaf gives us the tail of her sigchain. Fetch the whole chain to check it matches the leaf in the tree.

Detailed Design

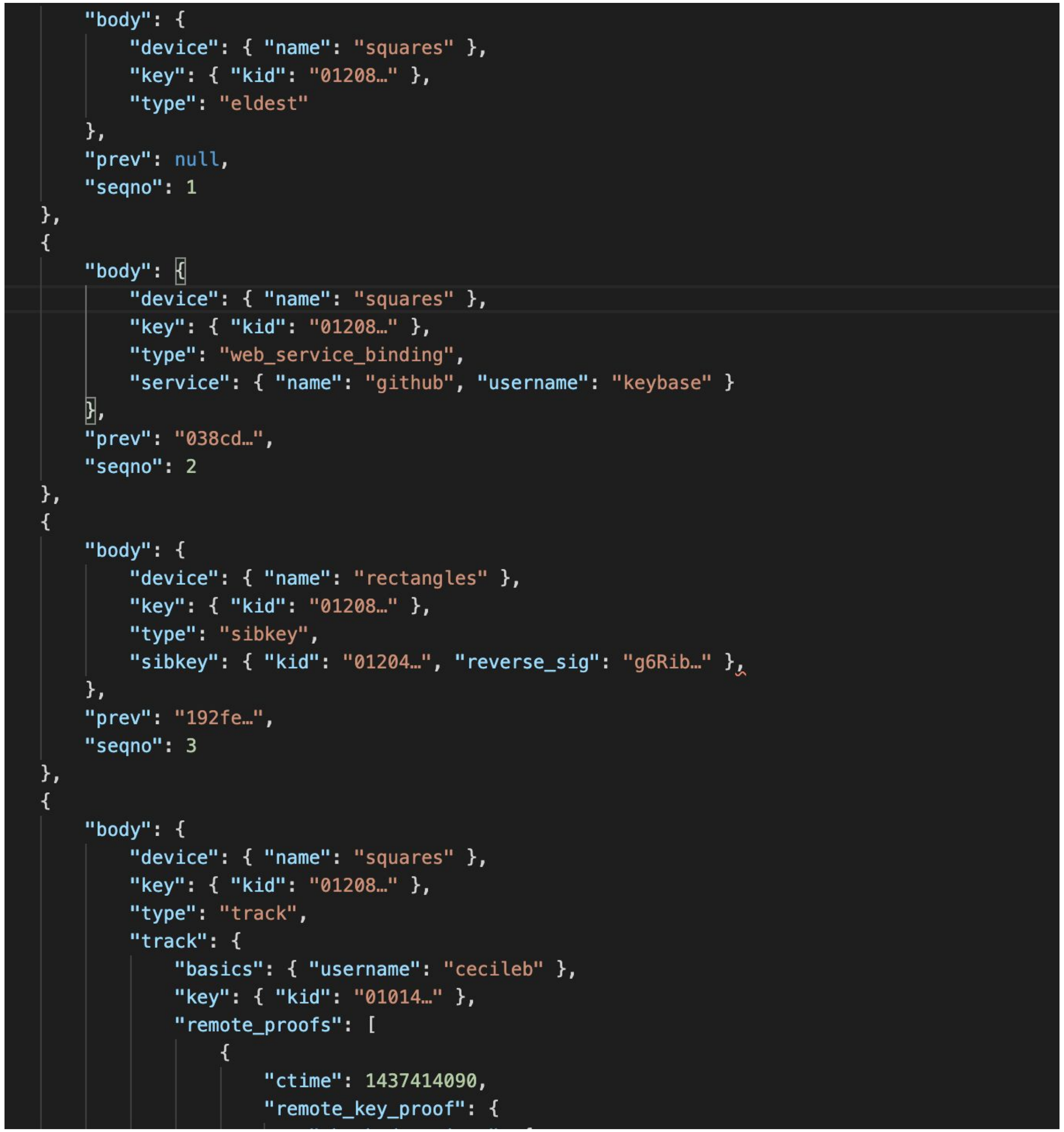
Sigchain

.Signed up for Keybase from a device called “squares”

.Proved their GitHub account

.Used squares to add another device called “rectangles” with its own key

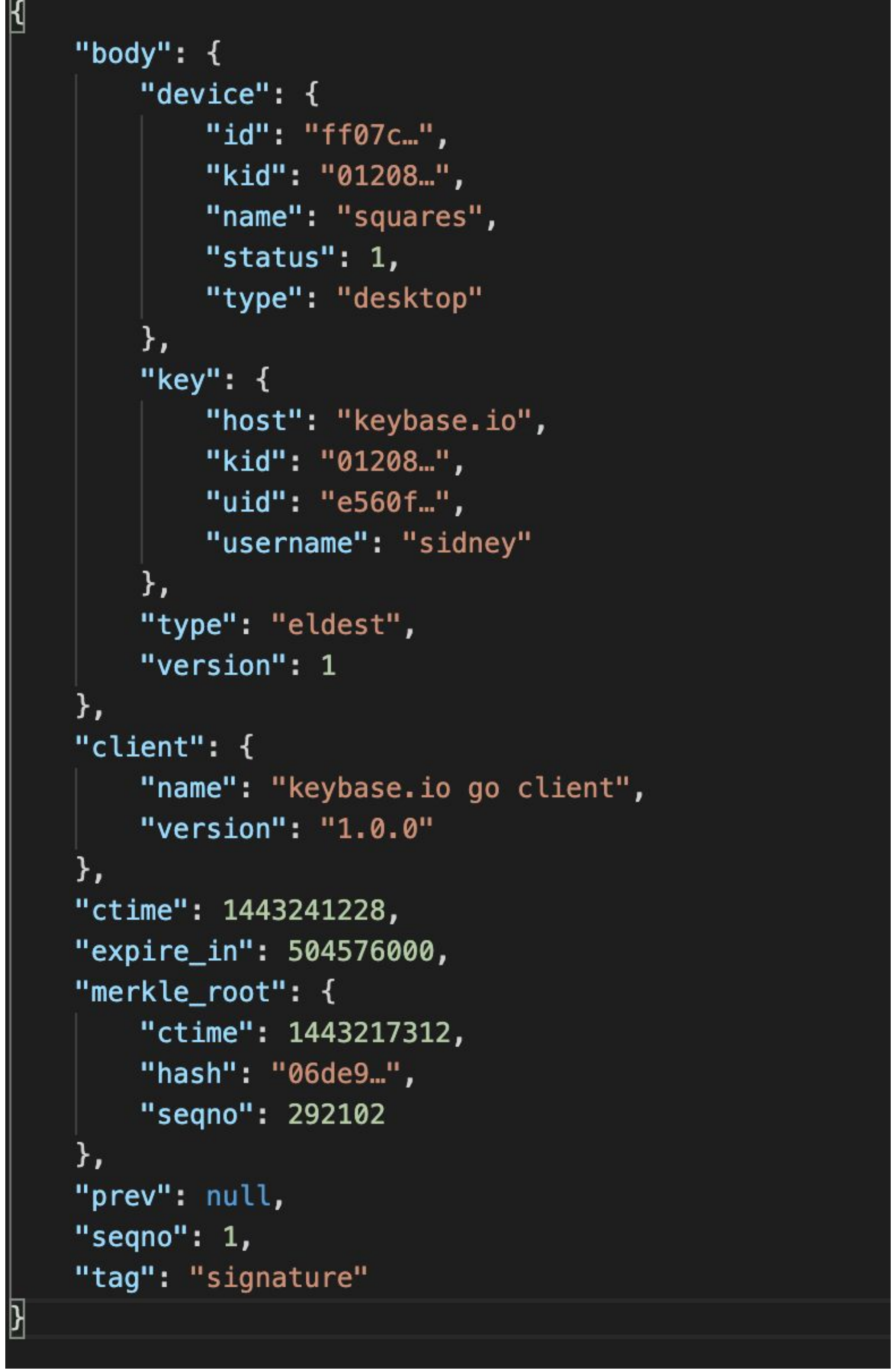
.Used rectangles to follow cecileb



```
        "body": {
            "device": { "name": "squares" },
            "key": { "kid": "01208…" },
            "type": "eldest"
        },
        "prev": null,
        "seqno": 1
    },
    {
        "body": {
            "device": { "name": "squares" },
            "key": { "kid": "01208…" },
            "type": "web_service_binding",
            "service": { "name": "github", "username": "keybase" }
        },
        "prev": "038cd…",
        "seqno": 2
    },
    {
        "body": {
            "device": { "name": "rectangles" },
            "key": { "kid": "01208…" },
            "type": "sibkey",
            "sibkey": { "kid": "01204…", "reverse_sig": "g6Rib…" },
        },
        "prev": "192fe…",
        "seqno": 3
    },
    {
        "body": {
            "device": { "name": "squares" },
            "key": { "kid": "01208…" },
            "type": "track",
            "track": {
                "basics": { "username": "cecileb" },
                "key": { "kid": "01014…" },
                "remote_proofs": [
                    {
                        "ctime": 1437414090,
                        "remote_key_proof": {
```

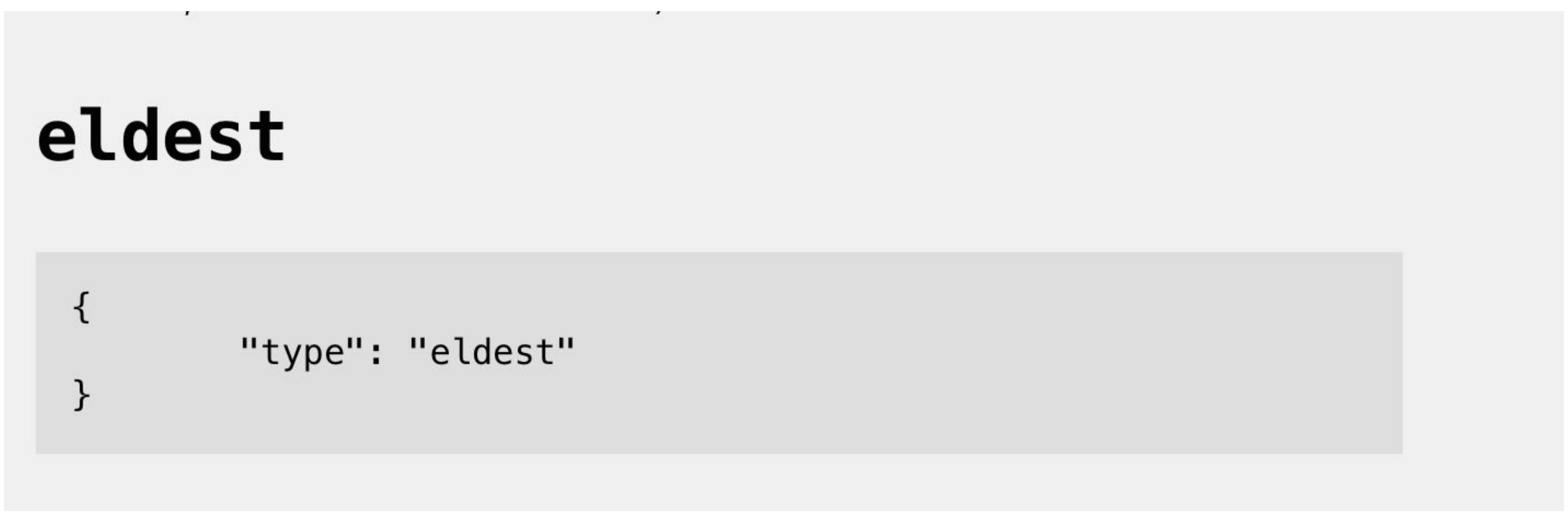
Link structure

- type – The type of the link
- device
- key – Information about the key that will sign the link. Contains these properties:
  - host – Currently always “keybase.io”
  - kid – The key’s KID
  - uid: The user ID of the sigchain’s owner
  - username: The username of the sigchain’s owner
- client
- ctime – When the link was created, as a Unix timestamp
- expire_in – How long the statement made by the link should be considered valid, in seconds,
- merkle_root – The creation time, hash, and sequence number of the Merkle tree root at the time the link was created
- prev – The hash of the previous sigchain link or null if this is the first link
- seqno – Specifies that this is the *n*th link in the user’s sigchain
- tag – Currently always “signature”.



```
{
    "body": {
        "device": {
            "id": "ff07c…",
            "kid": "01208…",
            "name": "squares",
            "status": 1,
            "type": "desktop"
        },
        "key": {
            "host": "keybase.io",
            "kid": "01208…",
            "uid": "e560f…",
            "username": "sidney"
        },
        "type": "eldest",
        "version": 1
    },
    "client": {
        "name": "keybase.io go client",
        "version": "1.0.0"
    },
    "ctime": 1443241228,
    "expire_in": 504576000,
    "merkle_root": {
        "ctime": 1443217312,
        "hash": "06de9…",
        "seqno": 292102
    },
    "prev": null,
    "seqno": 1,
    "tag": "signature"
}
```

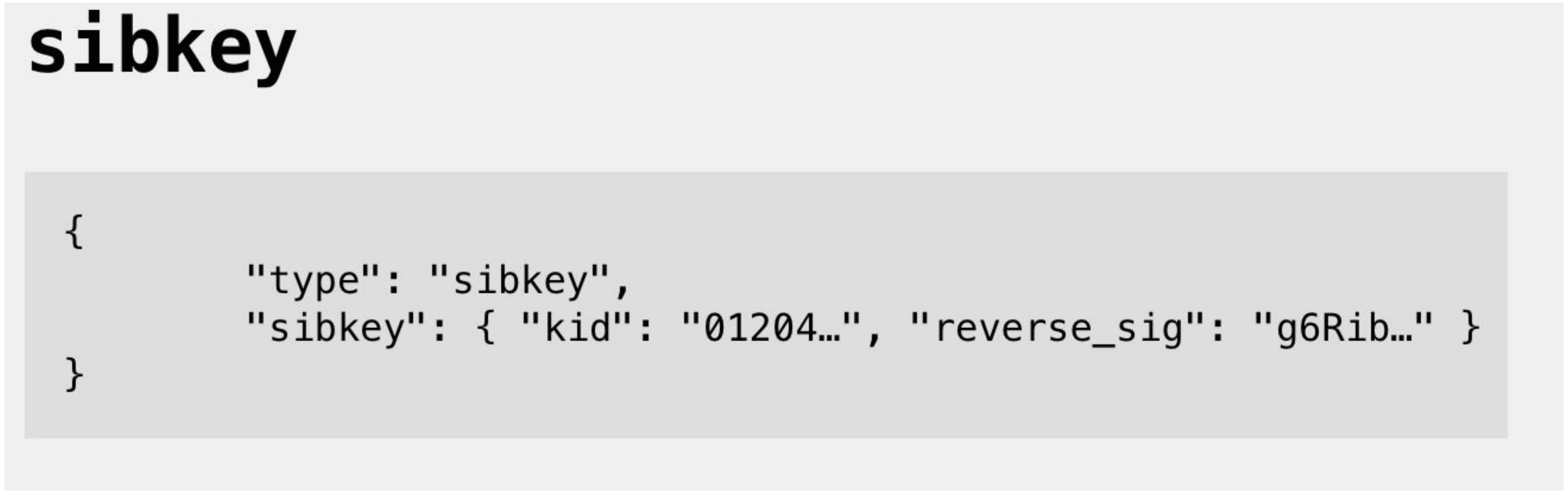
Link type



eldest

```
{
        "type": "eldest"
}
```

Appears at the beginning of a sigchain or after an account reset. The link’s signing key becomes the account’s first sibkey.



sibkey

```
{
        "type": "sibkey",
        "sibkey": { "kid": "01204…", "reverse_sig": "g6Rib…" }
}
```

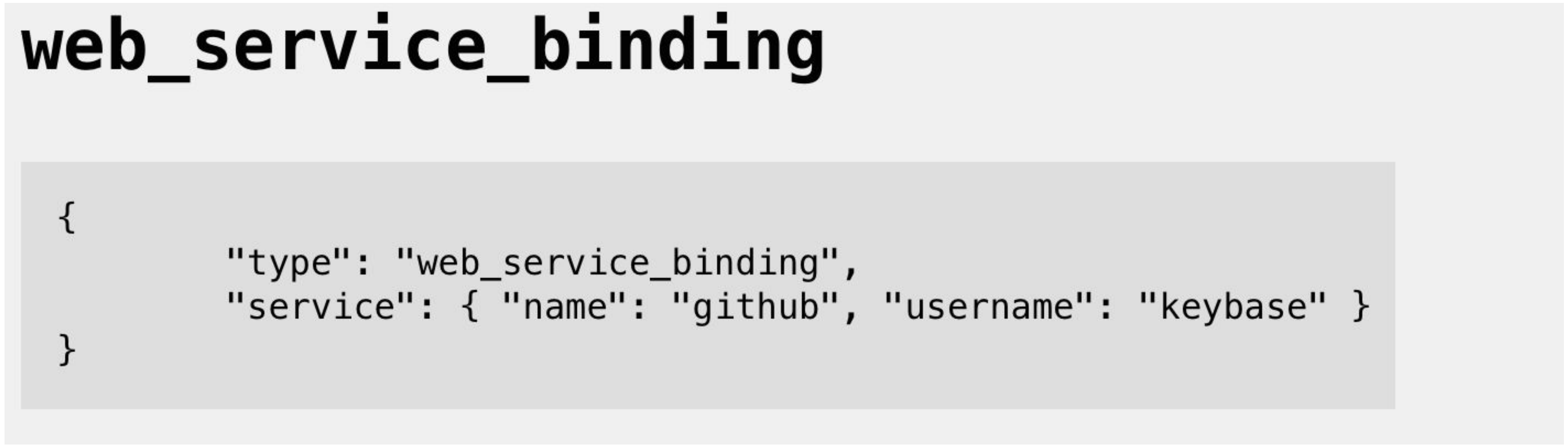
Add a new sibkey to the account. reverse_sig is a signature of the link by the new sibkey itself, made with the reverse_sig field set to null, and makes sure that a user can’t claim another user’s key as their own.

Discussion

Why are sibkeys useful?

Sibkeys

- A Keybase account can have any number of *sibkeys* which can all sign links.
- Revoking a key doesn’t affect your identity proofs, other keys, or followers.



web_service_binding

```
{
        "type": "web_service_binding",
        "service": { "name": "github", "username": "keybase" }
}
```

Claim, “I am username on the website name”.

The client will look for ***a copy of the link*** and ***signature*** on the website.

The server ***searches for the proof*** when the link is first posted, and caches its permalink (e.g. the tweet, on Twitter, the Gist, on GitHub) so that the client doesn’t have to rediscover it each time.

Track

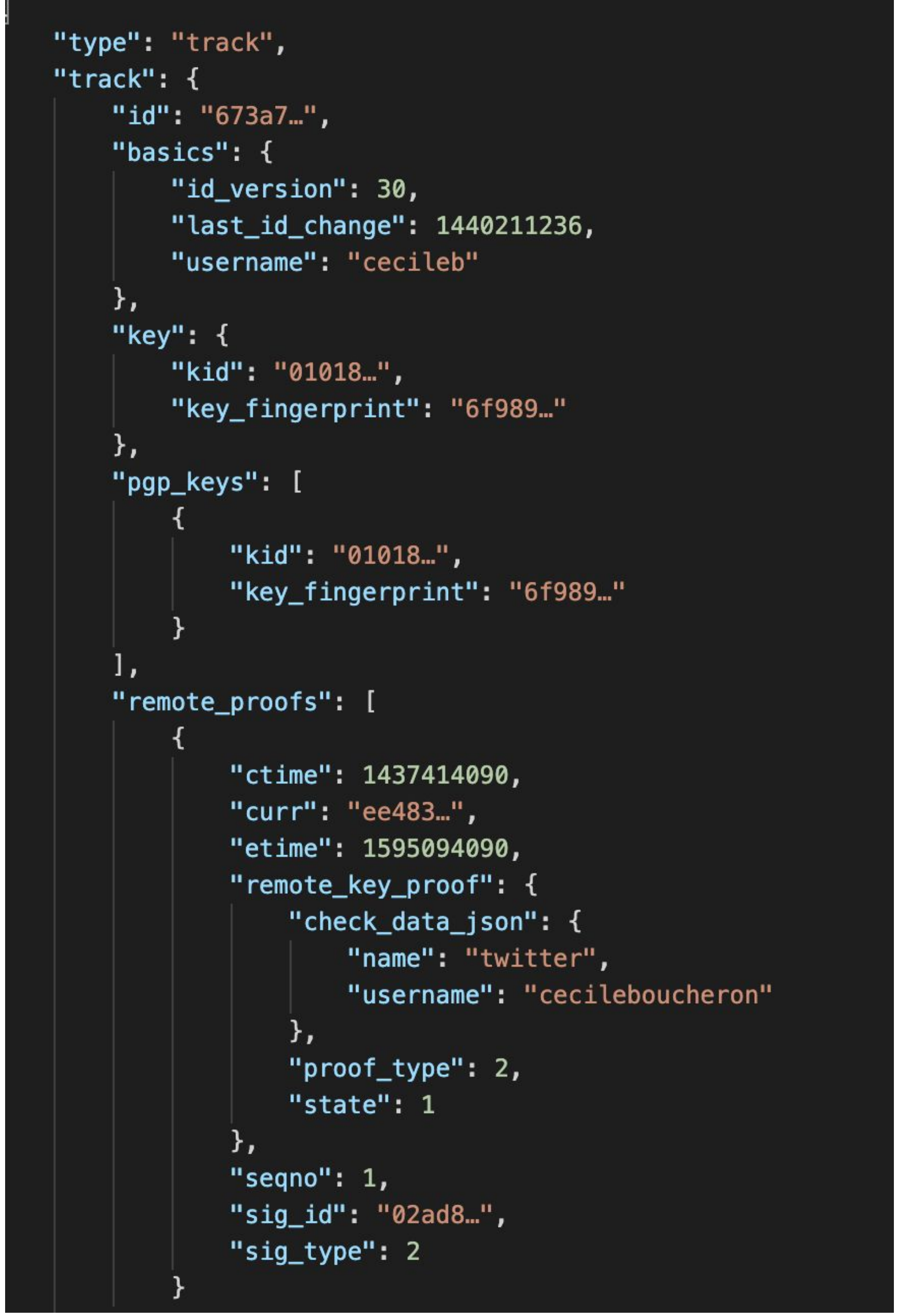
Make a snapshot of another user’s identity that your other devices trust.

•id – Followee user ID

•basics –The **server bumps** the identity generation whenever the state of any of their proof changes.

•remote_proofs: inherit from the followee’s link

- curr – The hash of the link which contains the proof
- remote_key_proof
    - check_data_json – The service section of the identity proof link



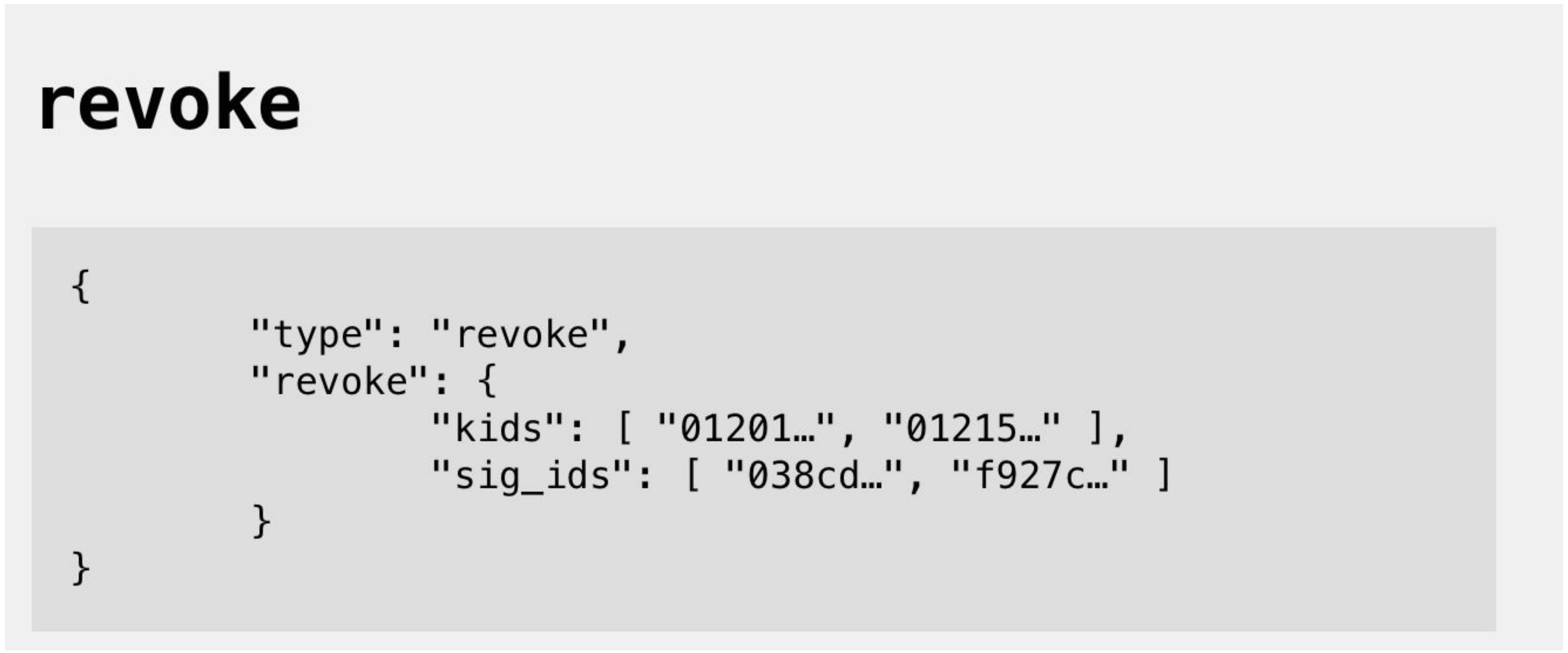
```
"type": "track",
"track": {
    "id": "673a7…",
    "basics": {
        "id_version": 30,
        "last_id_change": 1440211236,
        "username": "cecileb"
    },
    "key": {
        "kid": "01018…",
        "key_fingerprint": "6f989…"
    },
    "pgp_keys": [
        {
            "kid": "01018…",
            "key_fingerprint": "6f989…"
        }
    ],
    "remote_proofs": [
        {
            "ctime": 1437414090,
            "curr": "ee483…",
            "etime": 1595094090,
            "remote_key_proof": {
                "check_data_json": {
                    "name": "twitter",
                    "username": "cecileboucheron"
                },
                "proof_type": 2,
                "state": 1
            },
            "seqno": 1,
            "sig_id": "02ad8…",
            "sig_type": 2
        }
```

untrack (to "unfollow" someone)

```
{
        "type": "untrack",
        "untrack": {
                "basics": { "username": "maria" },
                "id": "47968…"
        }
}
```

cryptocurrency

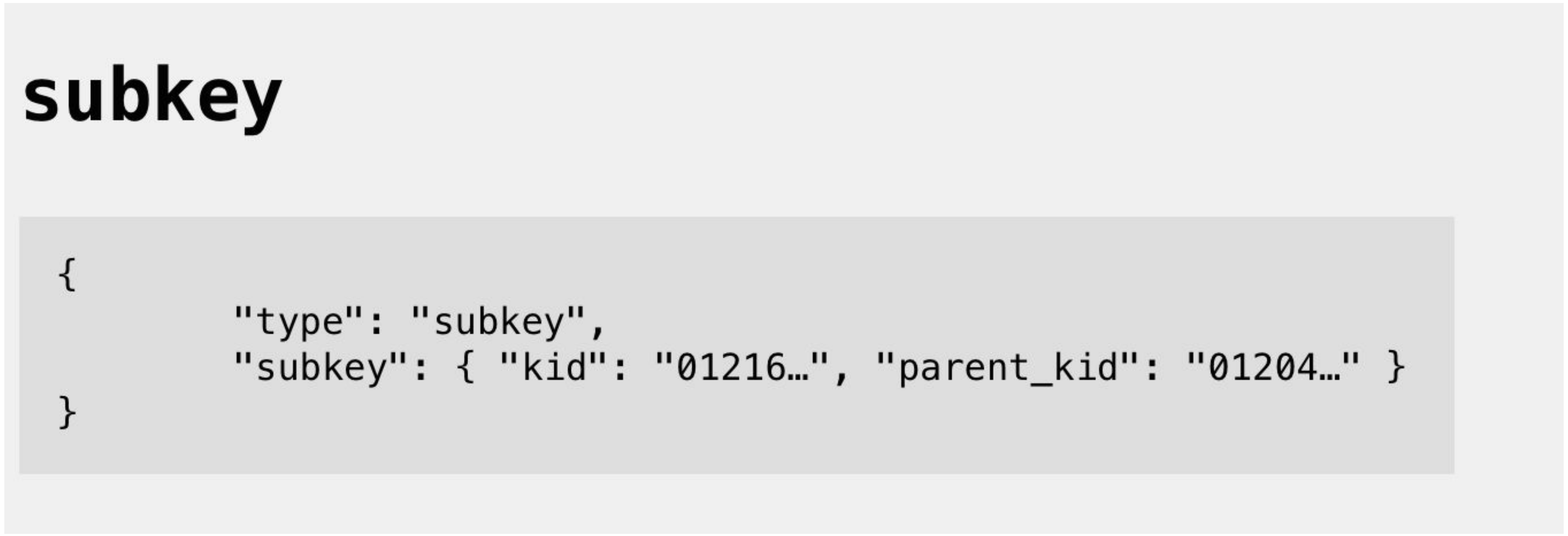
```
{
        "type": "cryptocurrency",
        "cryptocurrency": { "address": "1BYzr…", "type": "bitco
}
```



revoke

```
{
        "type": "revoke",
        "revoke": {
                "kids": [ "01201…", "01215…" ],
                "sig_ids": [ "038cd…", "f927c…" ]
        }
}
```

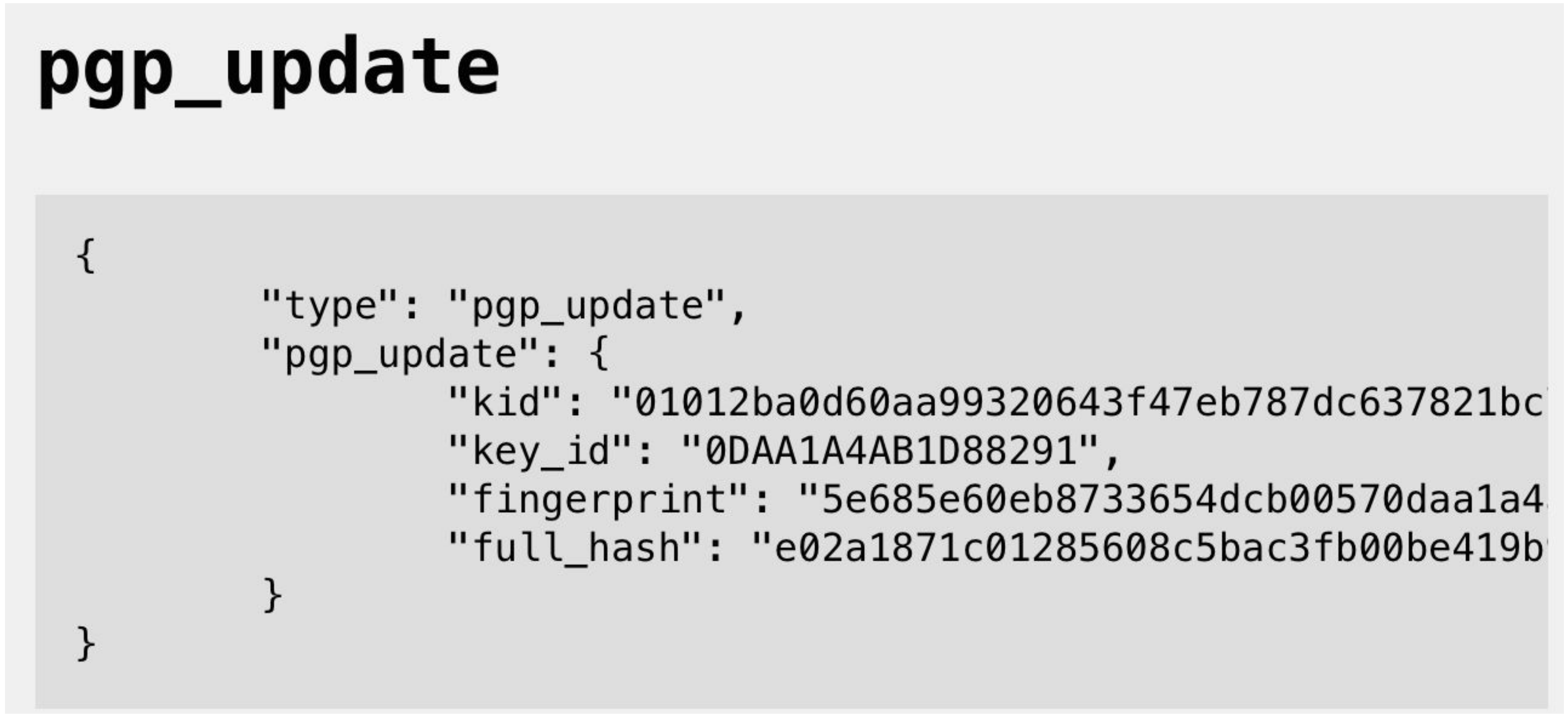
Remove the keys in kids from your account. Any previous links they've signed are still valid, but they can no longer sign new links and other users should no longer encrypt for them after seeing the revoke link. Also **reverse the effects** of the links in sig_ids— this can be used to remove, for instance, a web_service_binding.



subkey

```
{
        "type": "subkey",
        "subkey": { "kid": "01216…", "parent_kid": "01204…" }
}
```

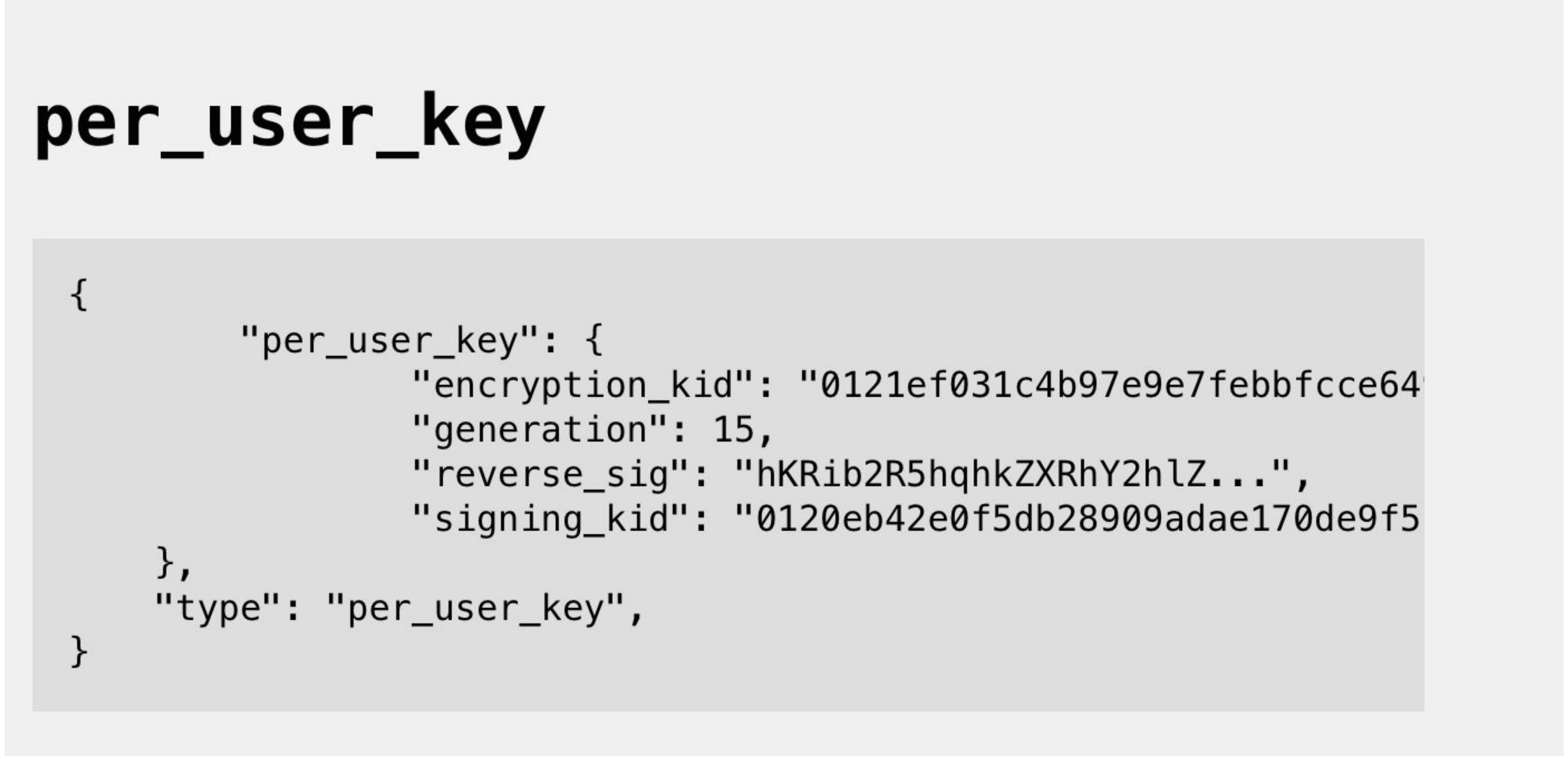
Add a new encryption-only *subkey* to the account.
Plan to use these in the future.



pgp_update

```
{
        "type": "pgp_update",
        "pgp_update": {
                "kid": "01012ba0d60aa99320643f47eb787dc637821bc
                "key_id": "0DAA1A4AB1D88291",
                "fingerprint": "5e685e60eb8733654dcb00570daa1a4
                "full_hash": "e02a1871c01285608c5bac3fb00be419b
        }
}
```

Update a PGP key to a new version (which may have new subkeys, revoked subkeys, new user IDs…). The pgp_update section contains the same properties a key section would have. full_hash is expected to have changed, the other properties should be unchanged.



per_user_key

```
{
        "per_user_key": {
                "encryption_kid": "0121ef031c4b97e9e7febbfcce64
                "generation": 15,
                "reverse_sig": "hKRib2R5hqhkZXRhY2hlZ...",
                "signing_kid": "0120eb42e0f5db28909adae170de9f5
    },
    "type": "per_user_key",
}
```

Add or rotate a per-user signing and encryption key. reverse_sig is the signature over the sigchain link with new per-user signing key itself. The generation number starts at one and increments whenever the per-user keys are rotated, typically after a device revocation.

Thanks for listening!