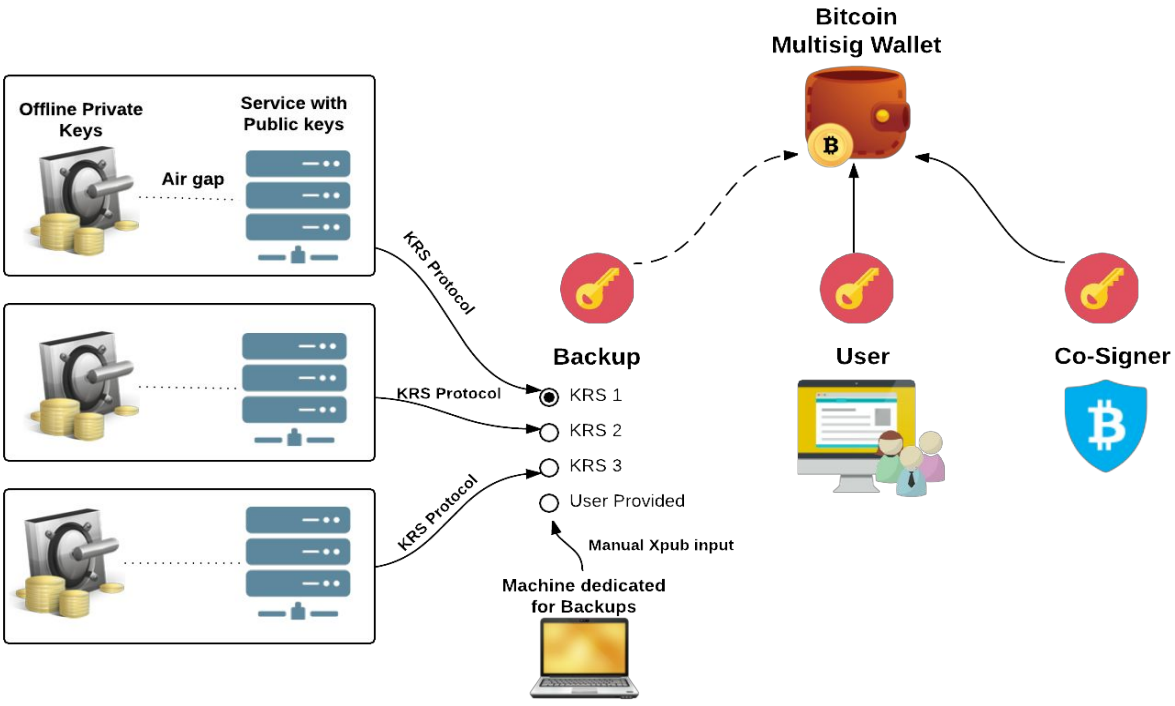
Key Recovery Service Protocol

Background

In multi-signature wallets it can be common to designate a backup key during key generation. This key can be used in cases where one of the primary keys is unavailable for transaction signing. There are, however, security risks and UX challenges inherent in generating backup keys, particularly if another operational key is generated on the same machine (common with web wallets today). An entity known as a Key Recovery Service (KRS) can provision backup keys on behalf of the user to alleviate some of these challenges and risks.



We propose a protocol where an automated KRS (as shown in the box above, built with a reference open source implementation at https://github.com/BitGo/key-recovery-service) can be used to provision public keys in an online environment. The operation of a KRS in this model involves:

Key Creation

The KRS operator first provisions the private key as part of a key ceremony. This is performed on an offline machine in a secure environment that is never taken online. The key is stored encrypted, in one or more parts, on that offline machine. The public part of this private key is then exported and taken online, to be used as the root key with the online service.

Online Service for Key Provisioning (KRS Protocol)

The service uses the Key Recovery Service Protocol (described later in this document) to accept key provisioning requests in an automated fashion. These requests may come from

wallet providers or user web browsers. Each request for a new key involves the BIP32 derivation of a unique path from the root public key, and that derivation path and username (email) is stored in a database. The KRS provider also establishes a relationship with the user in this step.

Recovery Process

To initiate a recovery, the wallet owner accesses a recovery page hosted on the KRS. This page contains client-side logic which helps the owner find their funds on the blockchain, and collect the required unspents to build a recovery transaction. The signing occurs client-side and uses the wallet owner's user key to provide a single signature to the transaction. This transaction is then sent to the KRS operator for co-signing.

The KRS operator must verify and implement any security processes at this point, using methods such as email, phone calls, notarized letters, time delays, etc. When they are ready to proceed, the KRS operator brings the half-signed transaction offline to the air-gapped environment, where it can be signed. The fully signed transaction is then brought back online. Through the entire process, the private key never leaves the offline environment.

Protocol Specification

This document describes the API protocol endpoints exposed by an automated Key Recovery Service (KRS) to deliver, validate and recover (use) public keys. The service is meant to be used with other wallet providers and applications. As such, the user-interface is out of protocol scope. User verification on the KRS is limited in this specification by design - the service should only hold or expose public keys, with the private keys being held in offline storage. Further, it is up to individual KRS operators to perform due diligence before signing recovery transactions.

The KRS protocol runs on top of the HTTP protocol and uses industry-standard JSON REST API practices. Where an API call is a GET, the request's URI and query component will be used for parameters. For other HTTP verbs (POST/PUT), we expect a body of JSON parameters to be sent. All responses will contain a body in JSON. The expected HTTP “Content-Type” and resultant “Accept” headers should be “application/json” unless otherwise specified. HTTPS is required and should be enforced by wallet services connecting to KRS providers, so as to prove the authenticity of the KRS.

Requesting a new public key

Used directly by users or wallet services to obtain a key on behalf of a user.

Endpoint: POST /key

Parameters (JSON in body):

- **userEmail**: The email of the user who will have sole control of the key for recovery purposes. A relationship should be set up between this user and the KRS.
- **userCustom**: Custom object of information about the user, often containing a secondary form of validation other than the email for recovery purposes. Example: phone.
- **requesterId**: An optional ID of the wallet service requesting the key.
- **requesterSecret**: An optional secret that may be required by the KRS of requesters.
- **notificationURL**: HTTP URL that will receive a webhook notification request when there is important information concerning this key (see "Webhook Notification Format")
- **hmac**: HMAC authenticating BitGo to the KRS. Calculated by applying sha256 to the *userEmail* and using the shared secret for the HMAC

Result (JSON Body):

- **userEmail**: The email (in the path) that has control of the key
- **xpub:** The freshly provisioned public key (BIP32 HD) associated with the userEmail provided. The wallet service is strongly encouraged to use a HD scheme which derives from this key for privacy purposes.
- **path**: An optional path for which the xpub was derived from on the KRS (assuming the KRS derives xpubs instead of storing a pre-created list). Given a pre-agreed upon root public key between wallet service and KRS, the wallet service can make the path derivation to verify the KRS has not been compromised (and an incorrect xpub issued).
- **custom**: Custom information returned by the KRS, such as a unique ID for the xpub

Example Request

```
curl -X POST -H "Content-Type: application/json" \
-d '{ "userEmail": "benchan@bitgo.com", "hmac" : "abcd12ef", "custom": {
"phone": "+14085551234" } }' http://localhost:6833/key
```

Example Response

```
{
    "custom": {
        "created": "2015-07-27T12:07:37.502Z"
    },
    "path": "m/1251361386/1591816709/1656196735/199349137/1573330069",
    "userEmail": "benchan@bitgo.com",
    "xpub":
"xpub6JaA22mn9W2h6R9y8n3Ug9hsWqZYVXkcQNcgBvXeWMVdtmaKEkSWGgkUbgotJDMCMMz6931rVy
vykKuX1raDY1HaDvzF5WandCsfDyL6WA4"
}
```

Validating a previously created key

This endpoint is optional. It allows an end user application to validate a previously created key exists on the KRS. If exposed, note the privacy concern that would allow anyone with a list of user emails and public keys (or vice versa) to query en masse and create a relationship between the two. In order to prevent an attacker trying public keys seen in redeemScripts on the blockchain, wallet services should derive an HD path from the public keys (and not use xpubs directly on the blockchain, where they could be easily scanned).

Endpoint: GET /key/:xpub?userEmail=benchan@bitgo.com&hmac=abcd12ef

Parameters (in path):

- **xpub**: The public key to be verified against that user

Parameters (in query):

- **userEmail**: The email of the user that has control of the key.
- **hmac**: HMAC authenticating BitGo to the KRS. Calculated by applying sha256 to the *xpub* and using the shared secret for the HMAC

Result (JSON body):

If the key was found to be provisioned for that user, the result will have a 200 status and contain

- **userEmail**: The email (in the path) that has control of the key
- **xpub**: The xpub (in the path) that is associated with the user email
- **path**: An optional path for which the xpub was derived from the master xpub on the KRS
- **custom**: Optional custom information about the key as provided by the KRS operator. The KRS operator should take care not to expose any private information here.

Otherwise a HTTP 404 status will be returned.

Example Request

```
curl
http://localhost:6833/key/xpub6JaA22mn9W2h6R9y8n3Ug9hsWqZYVXkcQNcgBvXeWMVdtmaKE
kSWGgkUbgotJDMCMMz6931rVyvykKuX1raDY1HaDvzF5WandCsfDyL6WA4?userEmail=benchan@bi
tgo.com
```

Example Response

```
{
    "custom": {
        "created": "2015-07-27T12:07:37.502Z"
    },
    "path": "m/1251361386/1591816709/1656196735/199349137/1573330069",
    "userEmail": "benchan@bitgo.com",
    "xpub":
"xpub6JaA22mn9W2h6R9y8n3Ug9hsWqZYVXkcQNcgBvXeWMVdtmaKEkSWGgkUbgotJDMCMMz6931rVy
vykKuX1raDY1HaDvzF5WandCsfDyL6WA4"
}
```

Requesting a recovery

May be used during a recovery event to provide transaction recovery information to prepare a recovery. During recovery, the derived paths used in multi-signature scripts will need to be known (in order to derive the correct private key to sign a recovery transaction). BIP32 specifies a path format to derive new keys, but does not specify what paths to use. Different wallet services use different paths and combine public keys in different orders. For the purposes of simplicity, this KRS endpoint is not expected to know the exact HD scheme. Instead, the individual paths and redeemScript will be provided by the caller. It is necessary for wallet services to share their HD schemes and create recovery scripts meant to generate the input for this endpoint. These scripts could be combined with a UI and hosted on the KRS for a better user recovery experience.

Endpoint: POST /recover

Parameters (in aJSON body):

- **xpub**: Public key that will be used for the recovery
- **userEmail**: The email of the user that has control of the key
- **hmac**: HMAC authenticating BitGo to the KRS. Calculated by applying sha256 to the *xpub* and using the shared secret for the HMAC
- **transactionHex:** Hex form of the incomplete transaction (typically half signed) that is requested for signing with the private key corresponding to the public key above.
- **inputs:** Array containing data required to sign the individual transaction inputs:
    - path: Derivation path from the public key above to make a redemption
    - redeemScript: P2SH redeem script for the input to be signed.

  The array is ordered (corresponding to the inputs in the incomplete transaction). If an input is not to be signed, an empty object should be used as a placeholder.
- **custom:** Optional custom object that may contain external information, such as a comment to the KRS operator.

Result (JSON body):

- **id**: The ID of this recovery request.
- **created**: The date the request was created.

In order to minimize leaking information, it is recommended to have this endpoint always return a HTTP 200 status as long as the inputs are not invalid (but no matter if the xpub is found or not). If the xpub and userEmail match, an email should be sent to indicate a successful request and request further authentication and proof of identity. It is recommended the endpoint store the request internally (to be prepared for offline signing). The KRS operator should have stringent processes in place to validate the user, such as via the phone number provided during key creation, or other more secure means.

Example Request

```
curl -H "Content-Type: application/json" \
-d '{ "xpub":
"xpub6JaA22mn9W2h6R9y8n3Ug9hsWqZYVXkcQNcgBvXeWMVdtmaKEkSWGgkUbgotJDMCMMz6931rVy
vykKuX1raDY1HaDvzF5WandCsfDyL6WA4",
        "userEmail": "benchan@bitgo.com",
        "hmac": "abcd12ef",
        "transactionHex":
"010000000176f1169fc7252d173b539b72be897408348ff37d72c236424e4026e057bc9cf60000
000000ffffffff01607be660190000001976a914dbb0c5b54a9347cb1ee82dbded41e2302ad5360
488ac00000000",
        "inputs": [
                {
                        "chainPath": "/0/1645",
                        "redeemScript":
"522103cdfebbd122a9fa9ba405efead022b24055273a12bffd7e5af1fc6e5bfdbe8dd321026880
27f13b00377ae19da43f3474f598ade81b967a365d3ee7d34867f2eba5732102c12f3d5579458b8
d8a08fcfd0f47c0b8c1dffe29fcc20fc84f33ef02e28c39a353ae"
                },
                {},
                {
                        "chainPath": "/0/8397",
                        "redeemScript":
"522103147f08b4017d5207470aee49d888cf2a7ae49306a916147475378ebd2dab661d21029f6a
d03f8933b8b83ca54246fc47f0e7d41dccac9629e0602433cee2c02f664421026b432d8b41d8133
8a6b3c4fd4a492cd0b9485a97deb7f68502176b03c6bb17de53ae"
                }
        ],
        "custom": { "message" : "help me!!! "}
}
' \
http://localhost:6833/recover
```

Example Response

```
{"id":"55b624036baa9c74202d3ce8","created":"2015-07-27T12:28:51.188Z"}
```

External Notifications

When creating a key, the requester may specify a **notificationURL** parameter. Setting this parameter will cause the xpub to be marked with the URL. An outgoing webhook request will be sent to the HTTP URL provided when there is an important event concerning the key. Such requests may include:

- When the key is created
- When recovery is requested using the key
- When the key is discovered to be compromised

This could help in situations where the user is creating a key directly at the KRS (and wants to have a way to fire off an event at the wallet service), or for monitoring purposes at the wallet service to mark a key when it has been used or recovered. Note that the receiving endpoint (wallet service) may not be authenticated (unless a secret is sent in the URI), so it is recommended that the service make a request to the validation endpoint (GET /key/xpub above) for confirmation. The webhook functionality is currently not implemented in the key-recovery-service Github repo, so it is up to the KRS operators if they want to provide this functionality to their users.

The KRS will send a POST event to the URL
Endpoint: POST http://notificationURL.com/api/providedPath/webhookReceiver
Parameters (JSON in body):

- **userEmail**: The email of the user who will has sole control of the key for recovery purposes.
- **xpub**: The public part of the key in question which the KRS is calling about.
- **hmac**: HMAC authenticating the KRS to BitGo. Calculated by applying sha256 to the xpub and using the shared secret for the HMAC
- **provider**: The KRS should identify itself so we know who the provider is
- **recoveryId**: Unique ID for the recovery event
- **state**: The latest state which the key is in. Valid states include:
    - **created**: sent upon provisioning a new key
    - **prerecovery**: sent prior to the KRS operator signing using key in recovery event
    - **unsafe**: sent in event the KRS operator believes the key is unsafe for further use
    - requested
    - signed
    - cancelled

This webhook is informational, and should not block any operations on the KRS.

Authentication between KRS Provider and Wallet Service

Not all KRS providers may wish to open up their services to the public. The KRS may use the following scheme (if it chooses) to require authentication by a wallet service.

Every request sent from the wallet provider to the KRS should have the following headers:

- **x-access-key**
  - Given to the wallet provider beforehand by the KRS
- **x-access-timestamp**
  - Current Unix timestamp in milliseconds
- **x-access-hmac**
  - Calculated as thus:
    1. Concatenate the following elements
       a. Timestamp
       b. HTTP method (uppercase)
       c. HTTP path (lowercase, with leading slash, and without trailing slash)
       d. request body if POST, empty string if GET
    2. Hash them using SHA256, and use a secret that had been given by the KRS to the wallet provider for the HMAC
    3. Convert the hash bytes to base64

When a KRS sends a request to the wallet provider (typically using webhooks), the same scheme holds, except the headers are renamed to x-webhook-key, x-webhook-timestamp, and x-webhook-hmac. The value for x-webhook-key as well as the secret for x-webhook-hmac should be shared with the KRS by the wallet provider beforehand.