Revault : a multi-party Bitcoin vault architecture

Kevin Loaec, Antoine Poinsot

May 6, 2020

The secure storage and usage of funds is a big challenge to Bitcoin users and especially to companies managing a substantial amount of bitcoins. The censorship resistance and lack of identity **features** of Bitcoin make it hard to setup a traditional control of expenses.

Such companies currently mostly rely on agreements with insured custodians or on multisig schemes. The former only shifts the risk from the company to the custodian while the latter either allows a third-party to lock the funds or leaves the company be a single point of failure, as backup keys are used most of the time.

We propose a non-custodial architecture targeting multi-party holders (such as a company or its third party custodian itself) which aims to **disincentivize theft while limiting the impact on the practical movements of funds**. Some parameters (e.g. the timelocks value or the spending policy) of the system can be customized depending on the use case and the needs of each user.

Threat model

Revault covers:

- theft for corruption of up to N-1 parties (with N the number of participants)
- potential theft mitigation for N (all) corrupted parties.

Revault does not cover:

- The signing refusal by one of the parties: we assume the users to solve this by legal agreement, as the intended users are likely to be stakeholders of a company.

The architecture

Revault aims at securing (i.e. mitigating the incentive to attempt theft) the management of multiple UTXOs co-owned by N parties. This management is in practice often handled by M parties, a subset of the overall owners of the funds. Revault retains this usability in practice for the parties managing the funds day-to-day, while allowing the use of a higher-threshold multisig (N ≥ M) at the root of the vault with **minimal overhead**.

Revault is named after the revocation transactions it uses which pay back to a vault (revaulting transactions).

Process

Funds enter the architecture through unspent and confirmed Bitcoin transaction outputs paying to the N parties (P2WSH multisig). We will call such utxos «vaults» thereafter.

After the reception of a vault, the N parties exchange signatures for a set of 4 non-broadcast transactions allowing either M parties to spend the output according to a previously agreed-upon policy or any participant to cancel such a transaction by spending it back to an ordinary vault or to the Emergency Deep Vault.

We name the Emergency Deep Vault a Bitcoin transaction output paying to a script harder to unlock than usual vaults. This locking script being kept private is a significant deterrent against a theft tentative.

For the rest of this document we chose the Emergency Deep Vault to be a script spendable by N static keys (of the N stakeholders) after a long X blocks timelock.

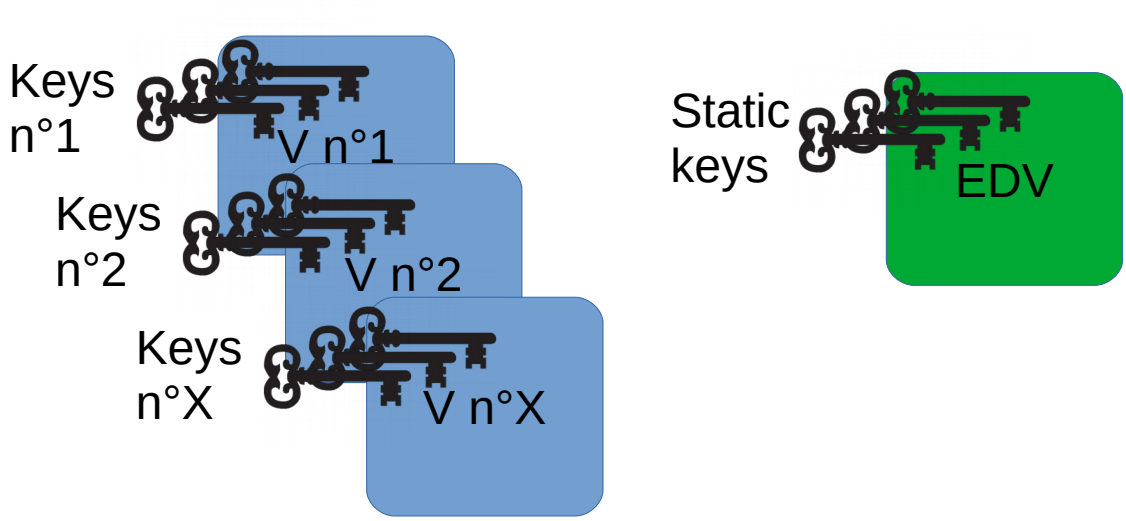

Figure 1: While there are many vault scripts, there is only one emergency deep vault script

The first transaction to be signed is the *emergency transaction* which spends a vault and pays to the Emergency Deep Vault.

Then, the signatures for transactions revaulting an *unvault transaction* are exchanged: there is the *unvault emergency transaction* which spends the *unvault transaction* to pay to the EDV, and the *cancel transaction* which spends the *unvault transaction* and pays back to an ordinary vault.

Finally, the signatures for the *unvault transaction* are exchanged: it spends a vault and pays either to M parties and N cosigning servers after X blocks, or to the N parties (for the revaulting transactions).

The transaction spending from the *unvault transaction* to an external address is called the *spend transaction*.

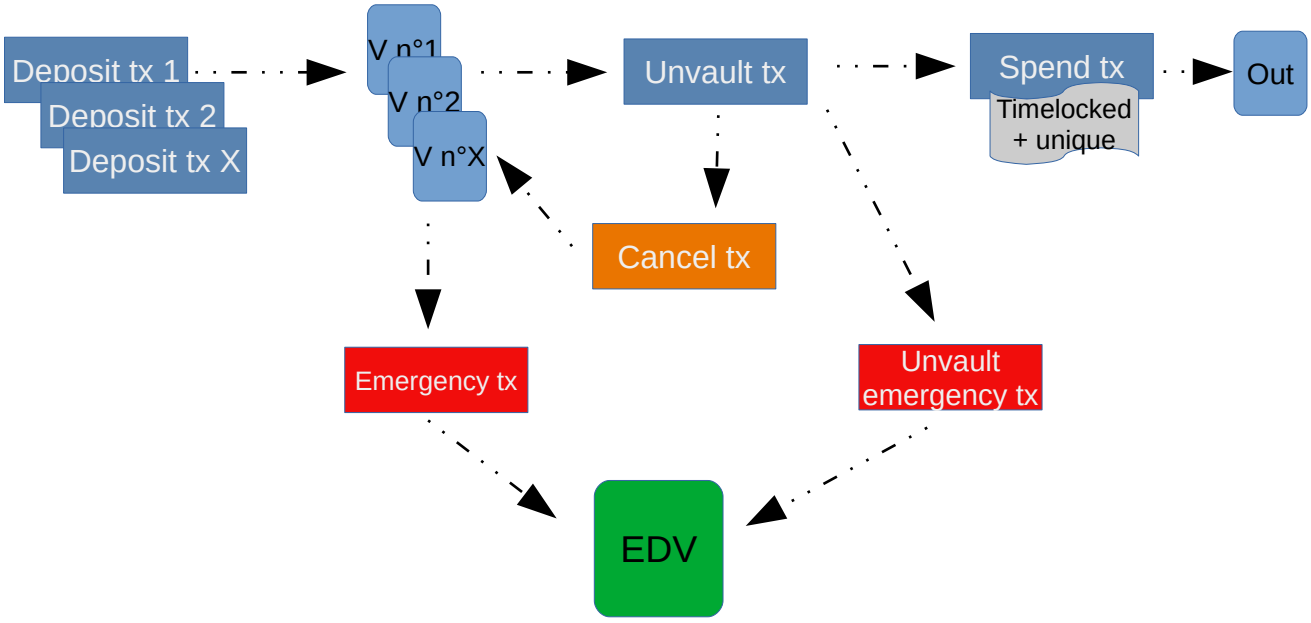

Figure 2: The set of transactions

Cosigning servers are very basic anti-replay oracles: they will accept to sign **any** *spend transaction*, but **only once**. This is a necessary evil to avoid the M parties to change the *spend transaction* destination after the *unvault transaction*'s locktime has matured and to enforce the *spend transaction* to be known for at least the entire duration of the locktime.

Each party runs a cosigning server to enforce the single-spend policy, they are of course not required if M=N.

In order to spend from a vault the M parties present a signed (by themselves and all the cosigning servers) *spend transaction* to all the network monitors (there are at least N of them) to advertise their willingness to spend from this vault.

In case the spenders broadcast the *unvault transaction* without the approval of all the watchers, a *cancel transaction* is broadcast.

At any point, a stakeholder's network monitor can broadcast an *emergency transaction*. To further reduce the incentive of a threat the broadcast of the *emergency transaction* of any vault triggers the broadcasting of the *emergency transaction* of all the vaults.

Infrastructure

Each party controls a wallet, a cosigning server and at least one network monitor («watchtower»).

Wallet: an open-source software running against a bitcoind backend used to keep track of the user's (co-)owned coins and to generate transactions. A secure signing device must be used to store keys and handle signing operations. As user verification is an important part of the security model, such a device must allow the user to check transaction integrity before signing it.

Cosigning server: a server providing an authenticated API for the M parties managing the funds to request signatures.

Watchtower: a server running a Bitcoin node to monitor the peer-to-peer network and the block chain for a breach. A breach is defined by a policy set by the user: it can for instance be set according to the destination of the funds, the maximum amount to unvault, a remote interaction from the user (red button), or even the lack of some action from the user.

While an externally ran wallet (part of the N parties) or cosigning server introduces a trusted third-party, externally ran watchtowers neither introduce a trusted third-party nor a higher onchain footprint.

Keys

Although there are weaknesses with regard to jeopardized derived private keys, we use unhardened key derivation as per bip-0032 for all the transactions scripts except for the *emergency* transactions. We do so in order to limit interactivity between stakeholders when creating scripts and to limit the storage of either sensitive or crucial (to construct redeem scripts) data.

The N static private keys used in the emergency script are generated once and the corresponding public keys reused for all the *emergency transaction* outputs.

The N emergency private keys and extended private keys are to be generated and the corresponding public (resp. extended public) keys to be exchanged during an in-person ceremony.

Transactions

All transactions are version 2 transactions using version 0 Segwit scripts.

Vault transaction

<vault_script> = N <pubkey 1> … <pubkey N> N OP_CHECKMULTISIG
If N > 15, the following vault script can be used to keep the transaction standard :
<pubkey 1> CHECKSIGVERIFY … <pubkey N> CHECKSIGVERIFY

- IN
  - N / A (from external source)
- OUT
  - At least one output paying to **0x00 0x20 SHA256(<vault_script>)**

Unvault transaction

The unvault script really depends on the needs of the user, but in the usual case with N overall parties and M of them managing the funds day to day, the following one is used (with the M pubkeys of the parties managing the funds placed first) :

```
<pubkey 1> CHECKSIGVERIFY … <pubkey M> CHECKSIGVERIFY

IF

      <pubkey M + 1> CHECKSIGVERIFY <pubkey N> CHECKSIG

ELSE

      X CHECKSEQUENCEVERIFY
      <cosigner M + 1> CHECKSIGVERIFY … <cosigner N> CHECKSIG

ENDIF
```

NB : As per the vault script, **CHECKMULTISIG** can be used if N ≤ 15.

- locktime : 0

- IN
  - Count : 1
  - inputs[0] :
    - txid : <vault_txid>
    - sequence : 0xffffffff
    - scriptSig : <empty>
    - witness :
      - if N <= 15,
        `0 <sig pubkey1> … <sig pubkeyN> <vault_script>`
      - else,
        `<sig pubkey1> … <sig pubkeyN> <vault_script>`
- OUT
  - Count : 1
    - outputs[0] :
      - value : <vault_tx output value> - <tx_fee>
      - scriptPubkey : **`0x00 0x20 SHA256(<unvault_script>)`**

Spend transaction

- locktime : 0
- IN
  - Count : 1
  - inputs[0] :
    - txid : <vault_txid>
    - sequence : **X**
    - scriptSig : <empty>
    - witness :
      `<sig subset member 1> … <sig subset member M> <sig cosigner M + 1> … <sig cosigner N> <unvault_script>`
- OUT
  - Count : 1 or 2 (optional change)
  - outputs[0] :
    - value : <unvault_tx output value> - <tx_fee> [- <change>]
    - scriptPubkey : N / A
  - outputs[1] *(optional)*:
    - value : <change>
    - scriptPubkey : **`0x00 0x20 SHA256(<vault_script>)`**

Cancel transaction

- locktime : 0
- IN
  - Count : 1
  - inputs[0] :
    - txid : <unvault_txid>
    - sequence : 0xfffffffe
    - scriptSig : <empty>
    - witness :
      - if N <= 15,
        0 <sig pubkey1> … <sig pubkeyN> <unvault_script>
      - else,
        <sig pubkey1> … <sig pubkeyN> <unvault_script>
- OUT
  - Count : 1
  - outputs[0] :
    - value : <unvault_tx output value> - <tx_fee>
    - scriptPubkey : **0x00 0x20 SHA256(<vault_script>)**

Emergency transactions

Both emergency transactions pay to the same emergency script :

<emer_script> = X CHECKSEQUENCEVERIFY N <emer_pubkey1> …
<emer_pubkeyN> N CHECKMULTISIG

As per the vault script, if N > 15 the following one can be used :
X CHECKSEQUENCEVERIFY <emer_pubkey 1> CHECKSIGVERIFY …
<emer_pubkey N> CHECKSIGVERIFY

Vault emergency transaction

- locktime : 0
- IN
  - Count : 1
  - inputs[0] :
    - txid : <vault_txid>
    - sequence : 0xfffffffe
    - scriptSig : <empty>
    - witness :
      - if N <= 15, **0 <sig pubkey1> … <sig pubkeyN> <vault_script>**

        - if N > 15, **<sig pubkey1> … <sig pubkeyN> <vault_script>**
- OUT
    - Count : 1
    - outputs[0] :
        - value : <vault_tx output value> - <tx_fee>
        - scriptPubkey : **0x00 0x20 SHA256(<emer_script>)**

Unvault emergency transaction

- locktime : 0
- IN
    - Count : 1
    - inputs[0] :
        - txid : <unvault_txid>
        - sequence : 0xfffffffe
        - scriptSig : <empty>
        - witness :
            - if N <= 15,
              0 <sig pubkey1> … <sig pubkeyN> <unvault_script>
            - else,
              <sig pubkey1> … <sig pubkeyN> <unvault_script>
- OUT
    - Count : 1
    - outputs[0] :
        - value : <unvault_tx output value> - <tx_fee>
        - scriptPubkey : **0x00 0x20 SHA256(<emer_script>)**

Fees

It is crucial for revaulting transactions to be confirmed in a timely manner in case of a breach. As such transactions are signed in advance, their feerate may not be viable anymore according to the block space market at the time of an urgent broadcast. In addition -and since they are spending co-owned outputs- we can’t rely on all parties accepting (or even being able) to sign a new version of the transaction at confirmation time before the timelock expires.

In order to mitigate this, all parties exchange SIGHASH_ANYONECANPAY | SIGHASH_ALL signatures for the revaulting transactions. This allows each party to attach an additional input to

revaulting transactions in order to bump the fees. Should this input not be consumed entirely, it can originate from a first stage transaction creating a change output.

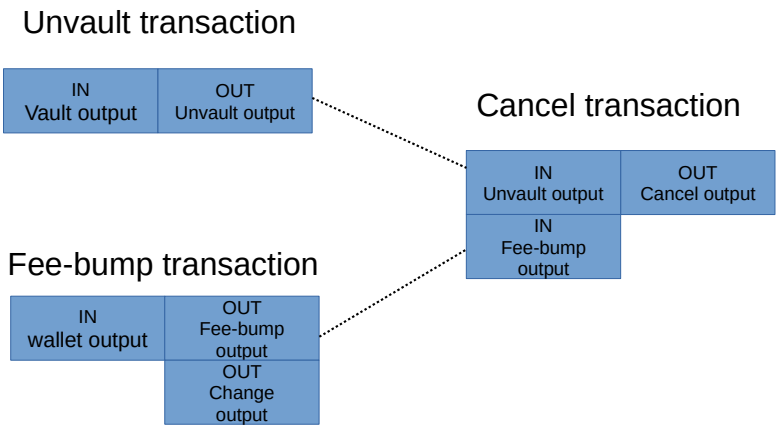

Figure 3: A fee bumped cancel transaction

All transactions should be appended a SIGHASH_ALL signature before being broadcast to prevent malleability.

Discussion

Cosigning servers

Cosigning servers introduce an additional cost ([N-M] * 71 WU) but permit high flexibility for the day-to-day managers of the funds, as they allow to choose both the destination and amount at spending time. They can however be omitted if M=N (every participant sign already) or in a scenario where there is only a small set of possible destinations and no need for change (pre-signed *spend transactions*).

They also introduce potentially “weaker” points of failure, but in practice the flexibility brought by their use largely outweigh this as it allows to have a higher-threshold multisig (N) for the vault scripts.

Finally, it might be worth mentioning that even if all the M parties and N-M cosigners are compromised and sign a malicious *spend transaction*, such transaction must signal replaceability (as it spends the CSV-encumbered path of the unvault script).

Watchtowers trustlessness

Watchtowers are a mean of increasing the reliance of the system, without putting the funds at risk if they are not fair. However even if funds are not put at risk there is always a tradeoff with regard to their trustlessness when they are given pre-signed transactions, especially when such transactions can lock the funds for some time.

An instance of this problem is present on Revault with the exchange of the *emergency transaction*s : the higher the number of network monitors are given this signed transaction, the lower is the success probability of an attack against the N participants but the higher the number of point of failures (where the failure here is defined as locking access to the funds for a long time period).

Proposed Bitcoin improvements

Some of the Bitcoin protocol proposed improvements discussed for integration into vault architectures were BIP119 and BIP118 (and friends) based covenants. Both these methods would require a setup phase (between the *vault transaction* and the *unvault transaction*) to have the *vault transaction* commit to the *unvault transaction* (as we **cannot** assume user won't reuse addresses). We could not however use them to get rid of the cosigning servers as long as the flexibility of choosing the amount and destination at spending time is required.

However, as our security model heavily relies on timelocks and revocation transactions we are more concerned about non-protocol Bitcoin improvements. Depending on the amount behind each vault, we must consider attacks from miners. Efforts to reduce mining centralization (such as STRATUM V2) would greatly improve the underlying security of the architecture.

Privacy

As *spend transaction*s are not pre-signed, the use of Revault will always be revealed after a spend attempt (or only after a succeeded spend if bip-0341 and bip-0342 are accepted by the Bitcoin network).

Acknowledgments

Thanks to Bryan Bishop for sparking our interest in Bitcoin vault protocols.

Thanks to Bryan Bishop, Spencer Hommel and Jacob Swambo for useful discussions around different ways of constructing such protocols.

Thanks to NOIA who, seeking a non-custodial secure way of storing and using their bitcoins, sponsored a previous version of Revault and a prototype for it.

Thanks to Lea Thiebaut (who also coined the name), Pierre Lorcery, Aaron Van Wirdum, Michael Folkson, and Udi Wertheimer for inviting us to talk about Revault which permitted to gather early feedback about the architecture.