Revault: a multi-party Bitcoin vault architecture

Kevin Loaec, Antoine Poinsot

November 17, 2020 (second revision)

The secure storage and transaction 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.

We propose a self-managed custody architecture for multi-party holders (such as a company or their third party custodian) aiming to **dis-incentivize theft while limiting the impact on the practical movements of funds**. A multi-layer access process is enforced wherein parameters (e.g. the length of time-locks value or the spending policy) of the system can be customized depending on the use-case and the needs of each user.

The architecture

Revault aims to increase the security for the control of multiple UTXOs co-owned by N stakeholders and enable delegation of fund management to M managers.

While traditional multi-signature schemes are looking for an impossible sweet-spot between security (large number of signatories required) and usability (low number of parties required), Revault allows for a delegation of the spending process without sacrificing the large multi-sig security.
In practice, this delegation only allows the managers to follow a pre-approved spending policy, thus retaining control at the stakeholder level. This is achieved by using valid pre-signed transactions falling back to the large stakeholder (N of N) multi-signature script. A threat deterrent is added to the scheme with another pre-signed transaction moving all funds to an unrevealed and stricter script, in case of an attack on the stakeholders.

For custody with N stakeholders, Revault protects against theft if up to N-1 stakeholders are compromised. As a consequence, Revault does not cover the signing refusal by one of the parties.

Process

Funds enter the architecture through unspent and (deeply) confirmed Bitcoin transaction outputs paying to all the stakeholders (N-of-N multisig).
After reception of a deposit, the stakeholders exchange signatures for a set of 4 non-broadcast transactions. These “pre-signed” transactions allow the managers to spend the funds according to a previously agreed-upon policy or any participant (either a stakeholder or a manager) to cancel such a transaction by sending it back to a deposit vault or to the Emergency Deep Vault.
We name the Emergency Deep Vault (EDV) a Bitcoin transaction output paying to a script harder to unlock than usual deposits. Keeping this locking script private is an additional deterrent against a theft attempt.

The first transaction to be signed is the Emergency transaction which spends a deposit and pays to the EDV.
Then, the signatures for the transactions re-vaulting an Unvault transaction are exchanged:

- the Unvault Emergency transaction, which spends the Unvault transaction to pay to the EDV,
- the Cancel transaction which spends the Unvault transaction and pays back to the same deposit script unlocked by the Unvault transaction.

Finally, the Unvault transaction is signed: it spends a deposit and pays either to the managers along with N cosigning servers after X blocks, or to the stakeholders without delay (for the re-vaulting transactions).
The transaction spending from the Unvault transaction to an external address is called the Spend transaction.

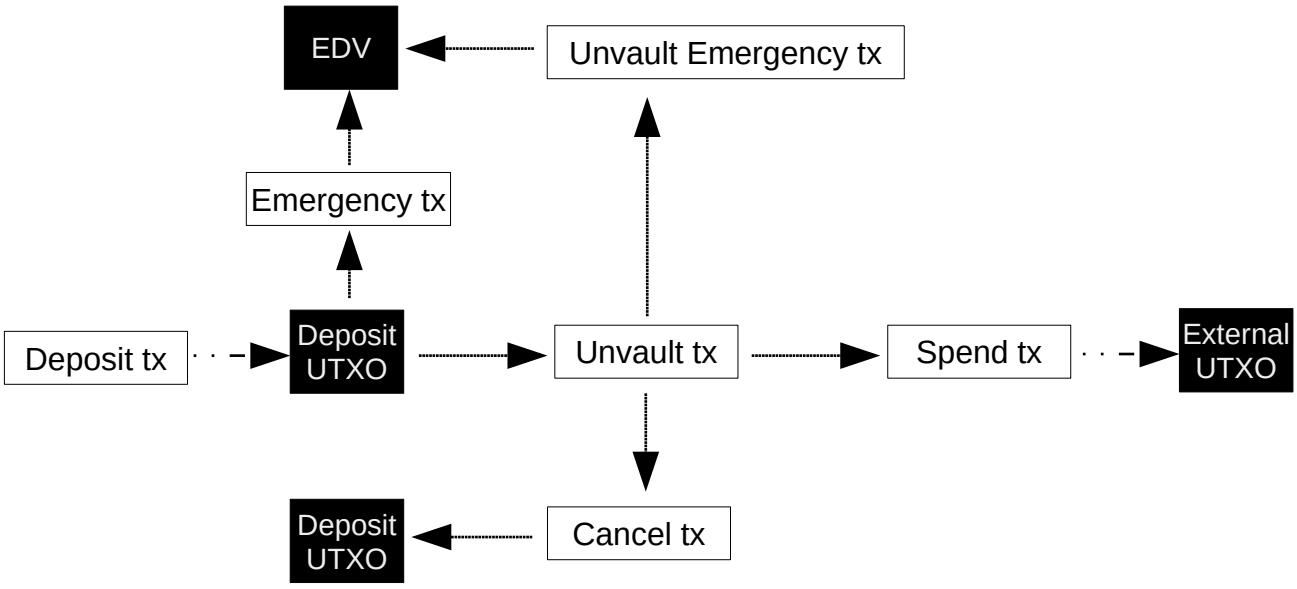

Figure 1: The set of transactions

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 stop the managers from changing the Spend transaction destination after the Unvault transaction's locktime has matured.
Each stakeholder runs a cosigning server to enforce the single-spend policy.

In order to spend from a vault, the managers 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.
A network monitor noticing the broadcast of an Unvault without being aware of its Spend transaction will automatically broadcast a Cancel transaction. This implies funds cannot be spent without prior acknowledgement of a network monitor.
Any participant can also manually push a cancel transaction, effectively sending it back to a deposit.

This flexibility brought by the active defense mechanism allows any spending policy to be set and enforced by watchtowers. This therefore largely extends the scope of the possibilities compared to on-chain enforcement and is more in-phase with custodians' needs.

At any point, a network monitor can broadcast an Emergency transaction. To further reduce the incentive of a theft attempt the broadcast of the Emergency transaction associated with any vault triggers the broadcast of the Emergency transaction associated with all the remaining vaults.

Infrastructure

Each manager controls a wallet and a signing device.
Each stakeholder controls a wallet, a signing device, a cosigning server and at least one network monitor.

Wallet: an open-source software running against a *bitcoind* back-end used to keep track of the user's (co-)owned coins and to generate transactions. It also handles the communication with the other participants. A manager's wallet also shares fully-signed transactions with the watchtowers.

Signing Device: a hardware module used to store keys and handle signing operations. As user verification is an important part of the Revault security model, such a device must allow the user to check the message integrity before signing it. It would ideally not require strong technical skills to perform the validation, Bitcoin Scripts could for example be parsed to Miniscript and displayed as an abstract Miniscript policy to the user.

Cosigning Server: a server providing an authenticated API for the managers to request signatures, responding with its own signature **only once per deposit UTXO**.

Watchtower: a server running a Bitcoin node to monitor the network. It enforces compliance with business logic and responds with emergency or cancel transactions if a breach is detected.

While an externally ran wallet (part of the N parties) or cosigning server would introduce a trusted third-party, externally ran watchtowers neither introduce a trusted third-party risk nor a higher on-chain 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 we manage (this excludes the Emergency Deep Vault script). 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.

Transactions

Transaction Scripts are hereby described using the Miniscript abstract policy, and scriptPubKeys described using output descriptors.
We denote **N** the number of stakeholders, **M** the number of managers and **X** the value in blocks of the relative timelock to spend the unvault transaction.
All amounts are expressed in the smallest fraction of a bitcoin, usually referred to as a Satoshi (1 sat = 10^{-8} BTC).

Deposit transaction

Version	2
Locktime	N/A (already confirmed)
IN	N/A (from external source)
OUT	A least one output paying to the **deposit descriptor**

With the **deposit descriptor** being

```
wsh(thresh(N, pubkey1, pubkey2, ..., pubkeyN))
```

Unvault transaction

Version	2		
Locktime	0		
IN	0	txid	**Deposit** transaction id

	<table><tr><td></td><td><table><tr><td>Vout</td><td>N/A (depends on the actual deposit transaction)</td></tr><tr><td>sequence</td><td>0xfffffffd</td></tr><tr><td>scriptSig</td><td>empty</td></tr><tr><td>Witness</td><td>satisfy(**deposit descriptor**)</td></tr></table></td></tr></table>
OUT	<table><tr><td>0</td><td><table><tr><td>value</td><td>Deposit transaction output 0 value - **fees**</td></tr><tr><td>scriptPubKey</td><td>**Unvault descriptor**</td></tr></table></td></tr><tr><td>1</td><td><table><tr><td>value</td><td>330</td></tr><tr><td>scriptPubKey</td><td>**Cpfp descriptor**</td></tr></table></td></tr></table>

With the **unvault descriptor** being

```
wsh(
    or(
        1@thresh(N, stakeholders),
        9@and(thresh(custom_threshold, managers),
            and(thresh(N, cosigners),
            older(X))
        )
    )
)
```

and the **cpfp descriptor** being

```
wsh(thresh(1, pubkey1, pubkey2, ..., pubkeyM))
```

With the **fees** being computed at the current minimum relay feerate of 253 satoshis per kilo-weight. A CPFP output is present as a fee update mechanism since the **unvault transaction** is likely to be signed significantly prior to broadcast.

Spend transaction

Version	2
Locktime	0
IN	<table><tr><td>0</td><td><table><tr><td>txid</td><td>**Unvault** transaction id</td></tr><tr><td>Vout</td><td>0</td></tr><tr><td>sequence</td><td>**X**</td></tr><tr><td>scriptSig</td><td>empty</td></tr><tr><td>Witness</td><td>satisfy(**unvault descriptor**) via the managers path</td></tr></table></td></tr></table>
OUT	<table><tr><td>0</td><td><table><tr><td>value</td><td>Unvault transaction output 0 value – **fees** [- change]</td></tr><tr><td>scriptPubKey</td><td>N/A (externally controlled)</td></tr></table></td></tr></table>

	1	value	330
		scriptPubKey	**Cpfp descriptor**
	2 (optional)	value	N/A
		ScriptPubkey	**Deposit descriptor**

The **spend transaction** *may* reference multiple **unvault transaction**s (batching).
The **spend transaction** *may* include multiple externally controlled outputs.
The **fees** are computed at the time of signing, as per the user's preference. A CPFP output is present as, depending on the timelock imposed by the unvault descriptor, the fee market may have evolved significantly during the wait time.

Cancel transaction

Version	2		
Locktime	0		
IN	0	txid	**Unvault** transaction id
		Vout	0
		sequence	0xfffffffd
		scriptSig	empty
		Witness	satisfy(**unvault descriptor**) via the stakeholders path
OUT	0	value	Unvault transaction output 0 value – **fees**
		scriptPubKey	**Deposit descriptor**

With the **fees** being computed at the minimum relay feerate of 253 satoshis / kilo-weight.
The **cancel transaction** may (and will likely) include a fee-bumping input at the time of broadcast.

Deposit emergency transaction

Version	2		
Locktime	0		
IN	0	txid	**Deposit** transaction id
		Vout	0
		sequence	0xfffffffd
		scriptSig	empty
		Witness	satisfy(**deposit descriptor**) via the stakeholders path

OUT	0	value	Deposit transaction output 0 value – **fees**
		scriptPubKey	**Emergency descriptor**

With the **Emergency descriptor** being

```
wsh(emergency script)
```

and the `emergency script` being set at the discretion of the stakeholders.
The **fees** are computed at the minimum relay feerate of 253 satoshis / kilo-weight.
The **emergency transaction** may (and will likely) include a fee-bumping input at the time of broadcast.

Unvault emergency transaction

Version	2		
Locktime	0		
IN	0	txid	**Unvault** transaction id
		Vout	0
		sequence	0xfffffffd
		scriptSig	empty
		Witness	satisfy(**unvault descriptor**) via the stakeholders path
OUT	0	value	Unvault transaction output 0 value – **fees**
		scriptPubKey	**Emergency descriptor**

The **fees** are computed at the minimum relay feerate of 253 satoshis / kilo-weight.
The **unvault emergency transaction** may (and will likely) include a fee-bumping input at the time of broadcast.

Fees

It is crucial for revaulting transactions to be confirmed in a timely manner in case of a breach. Since these 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 being able) to sign a new version of the transaction at any time.

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.

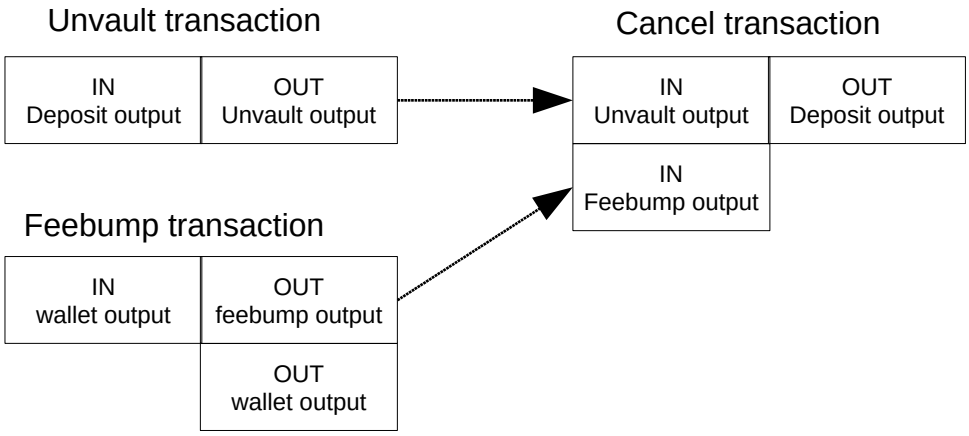

Figure 2: A fee-bumped Cancel transaction

Threat Model

While trying to fix (part of) the threats on existing custody protocols, this architecture introduces new challenges and different assumptions.
Revault aims to prevent theft with up to N-1 parties being compromised (including internal attacks). Here we discuss different attack scenarios by going through the previous sections, trying to identify weaknesses in each of them and eventually deduce a set of assumptions introduced by the usage of the architecture.

Regarding the architecture

While the Emergency Transaction is the biggest attack deterrent in the Revault architecture, it introduces a potential risk of blackmail or a loss external to the architecture (block-and-short).
An attacker (may be internal) getting hold of a single Emergency Transaction can threaten to push it to the Bitcoin network, triggering a push of all Emergency Transactions by the watchtowers. The funds are not at risk in this scenario but getting the funds back online would be long and difficult. On the other hand, an important part of the security model (the deterrent) is enforced by the Emergency pre-signed transactions being accessible.

An “active defense” mechanism is used for decreasing the multi-signature threshold from the N stakeholders to the M managers. Contrary to traditional multi-signature-only custody, transactions are accepted by default and reverted if they breach a predefined policy. As a consequence we rely on at least one entity being able to both notice the breach and react with a revocation transaction.

Regarding the process

As a consequence of the N-1 resilience model, there are multiple attacks against the practical usage of the funds:

- signing refusal by a stakeholder,
- unjustified Cancel transaction push,
- taking down a co-signing server,

Regarding the keys

The usage of a BIP32 unhardened derivation introduce even more reliance of the signing devices security.
As the chaincodes are exchanged and stored on the wallets (running on untrusted online devices) of each participant, the leak of a single child private key could be fatal to the entire set of private keys of one stakeholder. Therefore the leak of a single child private key of **each** of the N stakeholders would be equivalent to the loss of **all** the funds.
Using unhardened derivation implies that there is no partial loss of funds in case of a key leak by each stakeholder.

Regarding the transactions

Pre-signing the unvault transaction at 253sat-per-kiloweight is in practice a significant threat to the usage of the funds. With the current state of the Bitcoin network, even with a child fee-paying transaction the Unvault would likely not be propagated if there is an increase of the minimum transaction fee of the mempools of most network's nodes.

As for any protocol with pre-signed transactions, there is the concern of invalidating the pre-signed transactions chain with a malleable transaction and/or a block chain reorganization. This concern is further enhanced by the fact we ourselves create malleable deposit transactions (the Cancel Transaction is signed with `ALL|ANYONECANPAY`).

Regarding the fee bumping

Allowing outputs co-owned by more than 2 parties, we cannot use the carve-out rule for safe CPFP-based fee-bumping. Since no second-stage transaction depend on the revocation transactions we can afford introducing some malleability by using `ALL|ANYONECANPAY` as their signature hash flag for the unvault input.
Such malleability could be exploited to strip out fee-bumping inputs from the revocation transaction but an attacker cannot take advantage of the relay policy of *bitcoind* nodes to perform such an exploit. If this policy ever changes, an additional CHECKSIG could be added for watchtowers to attach a `ALL` signature to the input before broadcasting.
It is also assumed that a miner would prefer the fee-bumped transaction, if it is even aware of it.

More generally, fee-bumping is a handy mechanism but only shifts the bet on future block space market from the transaction signer to the transaction fee-bumper. The entity (likely, the watchtowers) bumping the fees is expected to maintain enough funds to be able to revault all the current vaults.
The funds required to avoid an uncompetitive feerate for revocation transactions depends on both the number of vaults monitored and the block space market at the time of broadcast. The former can be controlled by the software (forced batching or consolidation), while the latter cannot and could be subject to an attack (filling next blocks with transactions paying a feerate higher than what the watchtowers are able to in order to delay the confirmation of a revocation transaction).
The choice of the CSV value is therefore critical in this scenario: the higher it is, the higher the cost of a “block space market attack”.

Assumptions

Bitcoin

- We assume we can propagate revocation transactions to miners.
- We assume miners will mine the transaction paying the most feerate, regardless of any other (meta)data.
- We assume a significant hashrate to prevent reorgs of depth higher than the unvault CSV value.

Infrastructure

- We assume a proper ceremony as the root of trust for cryptographic protocols used.
- We assume a secure signing device allowing users to check the validity of the data being signed.
- We assume at least one honest cosigning server AND at least one honest online watchtower per stakeholder.
- We assume the N stakeholders’ keys to not be **all** compromised.
- We assume watchtowers to be financially able to cancel a spending attempt from all the vaults at the same time, at any time.
- We assume the Unvault transaction output relative timelock to be high enough for an attack on the block space market to not be worth it.

In practice, a different threat model considering privacy and blocking of operations is required. One is being worked on as part of the effort for specifying a concrete deployment of the Revault architecture at https://github.com/re-vault/practical-revault/ .

Further Discussion

Cosigning servers

Cosigning servers introduce an additional cost but permit high flexibility for the day-to-day managers of the funds as they allow to choose both the funds destination and amount at spending

time. They can however be omitted if all the stakeholders are the only managers (every participant signs already) or in a scenario where there is only a small set of possible destinations and no need for change (pre-signed Spend transactions).

Funds management

Stakeholders may sign all transactions but the Unvault one at the reception of a deposit. This would effectively create a vault which cannot be unvaulted but is still protected by the deterrent of the pre-signed Emergency transaction.
They may then defer the signing of the Unvault transaction to a later time. We qualify such vaults as being “inactive” and the process of signing the Unvault transaction (effectively allowing managers to use this vault) the “activation”.
Activating vaults is typical fund management by the stakeholders. As an example, owners of a business might be willing to use this process to delegate the management of a specific amount over a specific period of time to their employees.

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 users won’t reuse addresses). We would not be able to 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.
Moreover, bypass of the policy by the N stakeholders is likely to be a **feature** wanted by users.

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 transactions are not pre-signed, the use of Revault will always be revealed after a spend attempt (or only after a successful spend if bip-034[0, 1, 2] are deployed by the Bitcoin network and Musig usage is introduced in Revault).

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 Jacob Swambo and Edouard Paris for ongoing discussions regarding how to actually deploy a Revault architecture, and the threat model of such an implementation.
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.