

BTC++

Learn how Bitcoin Multi-Sig works

Michael Evans

Aleksa Jovanovic



3-OF-5 QUORUM

Desired Learning Objectives

- What is multi-sig and why use it?
- How does it work under the hood?
- Hands on experience!



What Is a Multi-Sig Wallet?

- A wallet that requires **multiple signatures** to spend from.
- Spending requires M of N signatures: ex. ("3 of 5") ("2 of 3").
- You can lose some keys without losing funds.
- The keys can live in different places, with different people, and be on different device brands.

A normal single-key wallet with your life savings is risky, loss or theft means you lose it all.

3-OF-5 QUORUM



Cold Card



Trezor



Ledger



Device 4



Device 5



Use Multi-Sig to safeguard extreme value:
Enterprise Treasuries, Small Business, Life Savings

Know Your Threat Model

What are we actually defending against?



Remote Attackers

Exchange hacks, Malware



Physical Attackers

“Wrench attacks”: coercion, robbery



Ourselves

Lost seeds, broken devices. Pass on your bitcoin with a multi-sig Inheritance scheme.



Insiders

A rogue employee. Protect your company with an auditable policy-driven treasury: who can spend and how many approvals does it take?



Natural Disasters

Fire, flooding, hurricanes

Multi-Sig best practices

01

Never store a quorum of keys in one place

A thief shouldn't be able to break in and find a quorum of keys to steal funds.

02

Make yourself travel

Obtaining a spending quorum should take time.

03

Use access-controlled locations

Don't let someone stumble upon your devices/seed phrases.

04

Diversify hardware

Mix device manufacturers to avoid single-vendor exploits.

05

Health-check every 6 months

Confirm each device still works, update firmware. Make sure you haven't already lost a key.

06

Never put private keys online

Not in email, not in password managers, not in cloud photos, not in notes app.



Under the Hood:

We'll use Bitcoin RPC (Remote Procedure Commands) to walk through the concepts.



Primer on Digital Signatures

Private keys and public keys exist in pairs. A private key creates a signature and its matching public key verifies that the signature was created by its corresponding private key.

So to sign something:

```
signature = signature_algorithm(private_key, hash(data))  
is_valid = verify_signature(public_key, hash(data), signature)
```

In Bitcoin,

The **seed phrase** is a master backup of all your private keys in your wallet.

Whoever has a **private key** can create a valid signature to spend the coins controlled by that key.

The **public key** lets the network verify a signature was created by someone with access to the private key.

The bitcoin **address** is a user-friendly string for receiving bitcoin (**bc1q57jkgfznapz76kycdkvu7jpynpdqnfq3rtvyv4**). It is derived from a single public key, or from a script containing multiple public keys.

```
signature = sign(private_key, hash(transaction_details))
```




First, we need our 5 XPUBs

Your seed phrase is used to derive a master **extended private key** or **XPRV**. The master XPRV is the root of your public+private key-pair tree. An **XPUB** or **Extended Public Key** gives you the public side of a branch in the key tree. It can generate receive addresses for that branch but not spend funds or reveal private keys. **Derivation Paths** tell us where we are in the key tree.

```
Seed phrase
|
v
Master XPRV (m)
|
+-- m/48h/...
|   BIP48 multi-sig branch
|   |
|   +-- XPUB -> multisig receive addresses
|   +-- XPRV -> multisig signing keys
|
+-- m/84h/...
|   BIP84 single-sig branch
|   |
|   +-- XPUB -> singlesig receive addresses
|   +-- XPRV -> singlesig signing keys
```

History: introduced by BIP 32 “Hierarchical Deterministic Wallets” (Pieter Wuille, 2012). Before BIP 32, one seed produced one address; now one xpub derives an endless tree of watch-only addresses.



First, we need our 5 XPUBs

Your seed phrase is used to derive a master **extended private key** or **XPRV**. The master XPRV is the root of your public+private key-pair tree. An **XPUB** or **Extended Public Key** gives you the public side of a branch in the key tree. It can generate receive addresses for that branch but not spend funds or reveal private keys. **Derivation Paths** tell us where we are in the key tree.

```
# These are our 5 wallets
bitcoin-cli createwallet "signer1"    # My Cold Card (fingerprint: f242c5cc)
bitcoin-cli createwallet "signer2"    # My Trezor      (fingerprint: 8a2f52d6)
bitcoin-cli createwallet "signer3"    # My Ledger
bitcoin-cli createwallet "signer4"    # My other device
bitcoin-cli createwallet "signer5"    # My other device

# Each hardware signer exports a BIP48 multisig account xpub. Fingerprints are short device identifiers.
hwi --fingerprint f242c5cc getxpub "m/48h/1h/0h/2h"
# master key / purpose' / coin type' / account' / script type'
# master XPRV / BIP48 multisig / testnet / account index 0 / native SegWit P2WSH

# Here are the resulting XPUBS (tpub in regtest. can be ypub, zpub)
signer1  [f242c5cc/48h/1h/0h/2h]tpubDFYoCXRHHTquZjL8oEs4...vyLK5xmr
signer2  [8a2f52d6/48h/1h/0h/2h]tpubDEjWCPLR6JzWeav6a6g...UTbTp1V9
(+ signer3, signer4, signer5)
```

History: introduced by BIP 32 “Hierarchical Deterministic Wallets” (Pieter Wuille, 2012). Before BIP 32, one seed produced one address; now one xpub derives an endless tree of watch-only addresses.



The Wallet Descriptor

A **wallet descriptor** is a recipe for generating addresses and defining how coins sent to those addresses can be spent. It states the script type, keys, derivation paths, plus (for multisig) the spending threshold.

```
wsh(sortedmulti(3ofTotal, key1, key2, key3, key4, key5))
```

This is a **native-segwit multi-sig** script requiring **3** signatures from these 5 key expressions. We'll use X PUBs in here (instead of single public keys) so our wallet can automatically derive new addresses and avoid address reuse, which is a best practice for privacy.

`sortedmulti()` lets participants list the same cosigner xpubs in any order. It sorts derived public keys lexicographically. This makes coordination easier than using `multi()` where different orderings derive different addresses.

```
# This is a multipath descriptor. It combines receive and change paths together using <0;1>.
# Branch 0 is used for receive addresses and branch 1 is used for change addresses.
# Before we had to import receive and change wallet descriptors separately.
wsh(sortedmulti(3,
  [f242c5cc/48'/1'/0'/2']tpub.../<0;1>/*,
  [8a2f52d6/48'/1'/0'/2']tpub.../<0;1>/*,
  ...))
```

<https://github.com/bitcoin/bitcoin/pull/22838> "Be able to specify change and receiving in a single descriptor string"



How to create a Watch-Only wallet for our Treasury:

```
# Create a blank watch-only wallet
bitcoin-cli -named createwallet \
    wallet_name=treasury \
    disable_private_keys=true \ # This wallet will never store private keys
    blank=true # create with no keys or default descriptors, we will import ours

# Import the descriptor
bitcoin-cli -rpcwallet=treasury importdescriptors '[
    { "desc": "wsh(sortedmulti(3,
        [f242c5cc/48h/1h/0h/2h]tpub.../<0;1>/*,
        [8a2f52d6/48h/1h/0h/2h]tpub.../<0;1>/*,
        ...,
        ...,
        ...))#ujyqzadm",
      "active": true,
      "timestamp": "now" }
    ]'
```



Receiving Payments

Our new watch-only wallet has everything needed to derive receive+change addresses and spot incoming balances. Every **getnewaddress** rpc call derives the next public key from each signer's xpub and generates a multisig receive address. No signing devices or private keys need to be involved to generate a new receive address.

```
# Generate a fresh address
bitcoin-cli -rpcwallet=treasury getnewaddress

bcrt1qtlg3fhlqm42fwvty4qpfeplepft94qqjsnjf287wj0lyn5y7qu8qyzcdss
```

Spending: The PSBT

Partially Signed Bitcoin Transactions are a standard file format for transactions that aren't fully signed.

There are two approaches for multi-sig:



Round-Robin

walletprocesspsbt

One signer appends a signature and passes the PSBT along to the next signer.



Fan-Out & Combine

combinepsbt

The coordinator fans a PSBT out to everyone, each signer adds a signature, then the coordinator merges them with this rpc.

History: BIP 174, by achow (2017), shipped in Bitcoin Core 0.17 (2018). Before it, co-signing meant shuttling raw hex and redeem scripts between incompatible wallets.



Spending: Create the PSBT (a 'draft' transaction)

Use the watch-only wallet to create the transaction (PSBT). You don't need the signing keys to propose a transaction.

```
# Create a PSBT
bitcoin-cli -rpcwallet=treasury walletcreatefundedpsbt "[" \ # [] so it auto-selects UTXOs to send
  '["bcrt1q6ezg3a5...kuu8l": 0.2]]' 0 \ # Send 0.2 BTC to this address, 0 for no locktime restriction
  '{"includeWatching": true}' # Include watch-only coins

# This is the resulting unsigned PSBT, no signatures added yet
{ "psbt": "cHNiGyRYZ/bNEKcC....AAA==", "fee": 0.0000225 }

# Verify the PSBT is correct before we sign!!!
bitcoin-cli decodepsbt "cHNidP8BAH0CAAAA..."
#   out 0.20000000 BTC -> bcrt1q6ezg3a5...kuu8l   (payee)
#   out 0.79997750 BTC -> bcrt1qv99hu0w...lnks    (change)
#   fee 0.00002250 BTC      <-- check amount + destination!
```



Anatomy of a Partially Signed PSBT

```
# Lets add signer1 using walletprocesspsbt and then have a look inside:
bitcoin-cli -rpcwallet=signer1 walletprocesspsbt "cHNidP8B..."
<psbt_with_1_signature_added> # A new PSBT is returned with signer1's signature added

bitcoin-cli decodepsbt "<psbt_with_1_signature_added>" # Below are some fields of interest to point out
{ "tx": {
  "vout": [
    { "value": 0.20000000, "address": "bcrt1q6ezg3a5...kuu81" } # Send 0.2 BTC to this address
    { "value": 0.79997750, "address": "bcrt1qv99hu0w...fsfmq" }, # Change goes back to one of our change addresses
  ],
  "inputs": [{ # This contains info needed for signers to sign this PSBT.
    "witness_utxo": { # The amount + locking script of the coin this transaction spends
      "amount": 1.00000000,
      "scriptPubKey": "0020 35a47c75...",
    },
    "partial_signatures": { # The signatures added so far
      "03d03cc3...b976e42a": "304402200dbce390...0501" # signer1
    },
    "witness_script": { # The multisig rule of the bitcoin being spent. Lets signers verify this input belongs to
      "asm": "3 <pk1>..<pk5> 5 OP_CHECKMULTISIG" # the 3-of-5 multisig wallet.
    },
    "bip32_derivs": [ # Shows which wallet derivation path produced each public key in this UTXOs multisig script:
      { "pubkey": "03d03cc3...", "fingerprint": "f242c5cc", "path": ".../0/3" }, # signer1
      { "pubkey": "02643c58...", "fingerprint": "8a2f52d6", "path": ".../0/3" } # signer2
      # ... signer2, signer3, signer4, signer5
    ]
  }],
  "fee": 0.00002250
}
```

Key idea: Each signature is bound to THIS exact transaction. Change any output and every signature already collected becomes invalid.



Spending: Collect & Combine the Signatures

Each signer starts from the same draft, so each returned PSBT carries only its own signature. **combinepsbt** combines the separate copies into one.

```
# Two more signers sign the SAME starting PSBT:
bitcoin-cli -rpcwallet=signer2 walletprocesspsbt "...
bitcoin-cli -rpcwallet=signer3 walletprocesspsbt "...

# Merge the three separate PSBT copies into one:
bitcoin-cli combinepsbt '["<psbt_1>","<psbt_2>","<psbt_3>"]'
<psbt_with_a_quorum>

# The partial_signatures field in the resulting PSBT now carries all three signatures:
{ "partial_signatures": {
    "03d03cc3...b976e42a": "304402...01",    # signer1
    "02643c58...cbfbbf83": "304402...01",    # signer2
    "02957c23...b2083952": "304402...01"     # signer3
  }
}
```

Spending: Finalize and Broadcast



Propose



Verify



Sign ×3



Combine



Finalize



Broadcast

`walletcreatefundedpsbt`

`decodepsbt`, check amount +
destination

`walletprocesspsbt`, once per signer

`combinepsbt` merges the signatures

`finalizepsbt` completes at M sigs

`sendrawtransaction`

```
# Now that we have a quorum of signatures we need to finalize the PSBT.  
# If this transaction is fully signed it will produce a serialized raw transaction hex.  
bitcoin-cli finalizepsbt "<psbt_with_a_quorum>"  
<raw_transaction>  
  
# Now we can broadcast this transaction to the network:  
bitcoin-cli sendrawtransaction "020000000001011b9367cf2d375ed6acdf1a632071db..."
```

Spending: Finalize and Broadcast

```
# Lets inspect some of the fields:
bitcoin-cli decoderawtransaction "0200000000001011b9367cf2d375ed6acdf1a632071db..."

{
  "txid": "602b4ce7...b14d15",
  "vin": [
    {
      "txid": "b36463c1...931b:0",
      "txinwitness": [
        "",                                # OP_CHECKMULTISIG dummy
        "3044...44601",                    # signer2 sig
        "3044...fec9f01",                  # signer1 sig
        "3044...710501",                    # signer3 sig
        "5321...55ae"                       # witness script: OP_3 <5 pubkeys> OP_5 OP_CHECKMULTISIG
      ]
    }
  ],
  "vout": [
    { "value": 0.79997750, "address": "bcrt1qv99hu0w...fsfmq" },
    { "value": 0.20000000, "address": "bcrt1q6ezg3a5...kuu81" }
  ]
}
```



Lost a Key? Rotate!

If you lose one or even two keys in our 3of5 key vault, no problem. All we have to do is send our bitcoin using the 3 surviving keys into a fresh vault.

```
# First get a replacement signer key X PUB:
hwi --fingerprint <new_fingerprint> getxpub "m/48h/1h/0h/2h"

# 2. New 3-of-5 descriptor: 4 surviving xpubs + the new one:
wsh(sortedmulti(3,<xpub1>,<xpub2>,<xpub3>,<xpub4>,<xpub5b>))
bitcoin-cli getdescriptorinfo "$NEWDESC"
bitcoin-cli createwallet "treasury2" true true
bitcoin-cli -rpcwallet=treasury2 importdescriptors '[ ...new desc ]'

# 3. Sweep old vault -> new vault (a normal 3-of-5 spend):
bitcoin-cli walletcreatefundedpsbt # Then add 3 signatures, finalize, then broadcast
```



Why Casa? Your Personal Swiss Bank

Casa helps the sovereign individual custody their own Bitcoin. **Casa never holds a quorum of your multi-sig private keys** and can **never** spend your funds.

- No KYC required for vaults or inheritance.
- Casa holds 1 of your keys, will only sign after verification. Duress support.
- OTC trades, buy/sell within app, team signing, custom solutions.
- 24/7 white-glove customer service.
- Business and Enterprise solutions, address whitelisting.
- Anti-Phishing support: call detection, optional travel rules, customer support verification codes.
- Sovereign recovery instructions. Import to Sparrow.
- Free Trial

Promo code - 25% off 1st year of standard, premium, or business:

referral-internal-ME

<https://casa.io/pricing?ref=referral-internal-ME>