BDK e Descriptors: Una Guida Pratica

Valerio Vaccaro

Satoshi Spritz Milano

26 Novembre 2025

Valerio Vaccaro ∞

- 💻 Sviluppatore Bitcoin ed Esperto Hardware
- 🔥 Contributore a progetti Bitcoin open source
- ⚠️ Appassionato di hardware fai-da-te (DIY)
- Ingegnere Bitcoin e Liquid presso Blockstream

Social

- 🧑‍💼 **LinkedIn** linkedin.com/in/valeriovaccaro
- 🐙 **Github** github.com/valerio-vaccaro
- **Telegram** t.me/valeriovaccaro

Licenza

Questa presentazione è distribuita sotto la licenza Creative Commons CC BY-SA 4.0.

Le immagini utilizzate in questa presentazione sono proprietà dei rispettivi autori e sono incluse solo a fini educativi e illustrativi.

May this presentation inspire you to become more self-sovereign!

Meme

Introduzione

Argomenti della Presentazione

- Descriptors e Miniscript: cosa sono e perché sono importanti
- BDK e bdk_cli: strumenti pratici per lavorare con Bitcoin
- Esempi pratici su Signet: singlesig, multisig, timelock e script complessi

Cosa sono i Descriptors?

Definizione

Un **descriptor** è una stringa testuale che descrive completamente come spendere fondi Bitcoin.

- Contiene tutte le informazioni necessarie per derivare chiavi e script
- Standardizzato in BIP 380
- ✓ Elimina la necessità di memorizzare script complessi
- Portabile e standardizzato

Vantaggi dei Descriptors

- **Portabilità**: stesso descriptor funziona su diversi wallet
- **Sicurezza**: non serve esportare chiavi private
- **Flessibilità**: supporta script complessi e timelock
- ✓ **Standardizzazione**: compatibilità tra diversi software

Tipi di Descriptors Base

Descriptors Principali

- `wpkh()` - Witness Public Key Hash (Segwit v0, P2WPKH)
- `sh(wpkh())` - Script Hash di Witness PKH (P2SH-P2WPKH)
- `tr()` - Taproot (Segwit v1, P2TR)
- `wsh()` - Witness Script Hash (P2WSH)
- `sh(wsh())` - Script Hash di Witness Script Hash (P2SH-P2WSH)

Esempio Base

`wpkh(pk(A))`

Questo descriptor descrive un output P2WPKH spendibile con la chiave pubblica specificata.

Miniscript

Cos'è Miniscript?

Miniscript è un linguaggio per rappresentare script Bitcoin in modo composabile e analizzabile.

- **Componibile**: script complessi da componenti semplici
- **Analizzabile**: proprietà verificabili automaticamente
- **Sicuro**: previene errori comuni negli script
- **Standardizzato**: BIP 383

Esempio Miniscript

`wsh(or_d(pk(A),and_v(v:pkh(B),older(144))))`

Questo script può essere speso da: - Chiave A, oppure - Chiave B dopo 144 blocchi

Miniscript: Componenti Base

Operatori Principali

- `pk(key)` - Verifica firma con chiave
- `pkh(key)` - Verifica hash della chiave pubblica
- `older(n)` - Timelock relativo (n blocchi)
- `after(n)` - Timelock assoluto (block height o timestamp Unix)
- `or_d(a,b)` - Disgiunzione (a o b)
- `and_v(a,b)` - Congiunzione (a e b)
- `multi(k,keys...)` - Multisig k-of-n

Timelock: Blocchi vs Timestamp

`older(n)`: Timelock relativo basato su numero di blocchi

- Esempio: `older(144)` = spendibile dopo 144 blocchi dalla creazione dell'UTXO
- Vantaggio: Indipendente dal tempo di clock
- Svantaggio: Tempo variabile (dipende dalla velocità di mining)

Miniscript: Componenti Base

Timelock: Blocchi vs Timestamp

`after(n)`: Timelock assoluto

- Se `n` < 500000000: interpretato come block height
- Se `n` >= 500000000: interpretato come timestamp Unix
- Esempio: `after(800000)` = dopo il blocco 800000
- Esempio: `after(1735689600)` = dopo il timestamp Unix 1735689600 (1 Gen 2025)
- Vantaggio timestamp: Tempo preciso e prevedibile

Compilatore disponibile su bitcoin.sipa.be/miniscript/

⚙ BDK (Bitcoin Development Kit)

Cos'è BDK?

BDK è una libreria Rust per costruire applicazioni Bitcoin.

- ⚙ Scritto in Rust, binding per Python, Swift, Kotlin
- 💻 Multi-platform: desktop, mobile, server
- 🔑 Supporto completo per descriptors
- 🚀 Performance elevate e sicurezza

Caratteristiche Principali

- 📜 Supporto completo per descriptors e miniscript
- 🪙 Gestione UTXO e wallet
- ✉ Creazione e firma di transazioni
- 🚀 Supporto per mainnet, testnet, signet, regtest

bdk_cli: Strumento da Linea di Comando

Installazione

```
cargo install bdk-cli
```

Oppure da sorgente:

```
git clone https://github.com/bitcoindevkit/bdk
cd bdk/cli
cargo install --path .
```

Comandi Principali

- **`wallet`** - Gestione wallet
- **`descriptor`** - Operazioni con descriptors
- **`address`** - Generazione indirizzi
- **`balance`** - Controllo saldo
- **`send`** - Invio fondi
- **`sync`** - Sincronizzazione con blockchain

bdk_cli: Struttura Base

Generazioni indirizzi

```
bdk-cli wallet ... \
  -d "wpkh(...)" \
  new_address
```

Sincronizzazione

```
bdk-cli wallet ... \
  -d "wpkh(...)" \
  sync
```

Controllo Saldo

```
bdk-cli wallet ... \
  -d "wpkh(...)" \
  balance
```

Setup Iniziale

Generazione Chiavi di Test

Per gli esempi useremo chiavi di test generate con:

```
export NETWORK=signet
export DATABASE_TYPE=sqlite
bdk-cli key generate
```

Ed a fine della presentazione userò:


```
{
  "fingerprint": "e7cdc822",
  "mnemonic": "envelope frame ten end original stumble blade bless unlock
  soon enter soccer",
  "xprv": "tprv8ZgxMBicQKsPeWmRT6CwSDXdR1Anq5YTHtA2DyS9Pu4fK7Pr2rbgnFV6Bp
  XviXZtgTYVKmtuQBuAL6bcSg7pmDmRrWEzFqCXmy35rf8u2dK"
}
```

Setup Iniziale

Generazione Chiavi di Test

E poi

```
bdk-cli key derive --xprv "tprv8ZgxMBicQKsPeWmRT6CwSDXdR1Anq5YTHtA2DyS9Pu4fK7
  Pr2rbgnFV6BpXviXZtgTYVKmtuQBuAL6bcSg7pmDmRrWEzFqCXmy35rf8u2dK"
  --path "m/84'/1'/0'"
```

Ottenendo

```
{
  "xprv": "[e7cdc822/84'/1'/0']tprv8gzeeyNhJUbWpdZvHLavBTwoaCrhLcfLm6KQ5m
  9HngP5y6Cdw9oupQhZxjia5hjsTiuSD5XeBfSmMWpBWA6tgEbQajZCEJxCaDSPBa4qz8d/*",
  "xpub": "[e7cdc822/84'/1'/0']tpubDDggoP.../*"
}
```

Per semplicità userò sempre queste derivazioni

Esempio 1: SingleSig P2WPKH

Comandi

```
export EXT_DESCRIPTOR="wpkh([e7cdc822/84'/1'/0']tpubDDggoP.../0/*)"
export INT_DESCRIPTOR="wpkh([e7cdc822/84'/1'/0']tpubDDggoP.../1/*)"

# Ottenere un nuovo indirizzo
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL new_address

# Sincronizzare
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL sync

# Controllare saldo
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL balance
```

Esempio 2: SingleSig P2SH-P2WPKH

Comandi

```
export EXT_DESCRIPTOR="sh(wpkh([e7cdc822/84'/1'/0']tpubDDggoP.../0/*))"
export INT_DESCRIPTOR="sh(wpkh([e7cdc822/84'/1'/0']tpubDDggoP.../1/*))"

# Ottenere un nuovo indirizzo
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL new_address

# Sincronizzare
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL sync

# Controllare saldo
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL balance
```

Esempio 3: SingleSig Taproot (P2TR)

Comandi

```
export EXT_DESCRIPTOR="tr([e7cdc822/84'/1'/0']tpubDDggoP.../0/*)"
export INT_DESCRIPTOR="tr([e7cdc822/84'/1'/0']tpubDDggoP.../1/*)"

# Ottenere un nuovo indirizzo
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL new_address

# Sincronizzare
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL sync

# Controllare saldo
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL balance
```

Esempio 4: Multisig 2-of-3

Comandi

```
export EXT_DESCRIPTOR="wsh(
  multi(2,
    [e7cdc822/84'/1'/0']tpubDDggoP.../0/*,
    [4d633db5/84'/1'/0']tpubDD39Bo1.../0/*,
    [693e4e5e/84'/1'/0']tpubDC6YrbM.../0/*
  ))"
export INT_DESCRIPTOR="wsh(
  multi(2,
    [e7cdc822/84'/1'/0']tpubDDggoP.../1/*,
    [4d633db5/84'/1'/0']tpubDD39Bo1.../1/*,
    [693e4e5e/84'/1'/0']tpubDC6YrbM.../1/*
  ))"
```

Nota: Tre chiavi pubbliche (altre due sono state generate analogamente alla prima), necessarie 2 firme su 3.

Esempio 5: Multisig P2SH-P2WSH

Comandi

```
export EXT_DESCRIPTOR="sh(wsh(
  multi(2,
    [e7cdc822/84'/1'/0']tpubDDggoP.../0/*,
    [4d633db5/84'/1'/0']tpubDD39Bo1.../0/*,
    [693e4e5e/84'/1'/0']tpubDC6YrbM.../0/*
  )))"
export INT_DESCRIPTOR="sh(wsh(
  multi(2,
    [e7cdc822/84'/1'/0']tpubDDggoP.../1/*,
    [4d633db5/84'/1'/0']tpubDD39Bo1.../1/*,
    [693e4e5e/84'/1'/0']tpubDC6YrbM.../1/*
  )))"
```

Esempio 6: Timelock Relativo

Comandi

```
export EXT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tpubDDggoP.../0/*),
    older(144)
  ))"
export INT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tpubDDggoP.../1/*),
    older(144)
  ))"
```

Nota: Spendibile solo dopo **144** blocchi dalla creazione dell'UTXO.

Esempio 7: Timelock Assoluto

Comandi

```
export EXT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tpubDDggoP.../0/*),
    after(800000)
  ))"
export INT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tpubDDggoP.../1/*),
    after(800000)
  ))"
```

Nota: Spendibile solo dopo il blocco **800000**.

Esempio 7b: Timelock con Timestamp Assoluto

Comandi

```
export EXT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tpubDDggoP.../0/*),
    after(1735689600)
  ))"
export INT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tpubDDggoP.../1/*),
    after(1735689600)
  ))"
```

Nota: Spendibile solo dopo il timestamp Unix **1735689600** (1 Gennaio 2025, 00:00:00 UTC). I timestamp devono essere $>=$ 500000000 per essere interpretati come timestamp invece di block height.

Esempio 7b: Timelock con Timestamp Assoluto

Calcolo Timestamp

```
# Converti data in timestamp Unix
date -d "2025-01-01 00:00:00 UTC" +%s
# Output: 1735689600

# Oppure usa Python
python3 -c "import datetime;
print(int(datetime.datetime(2025, 1, 1, tzinfo=datetime.timezone.utc).timestar
```

Esempio 7c: Timelock Relativo con Timestamp Calcolato

Comandi

```
# Calcola timestamp tra 30 giorni
TIMESTAMP=$(date -d "+30 days" +%s)
export EXT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tpubDDggoP.../0/*),
    after($TIMESTAMP)
  ))"
export INT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tpubDDggoP.../1/*),
    after($TIMESTAMP)
  ))"
```

Nota: Spendibile solo dopo una data specifica calcolata al momento della creazione. Esempio pratico: per un timelock di 30 giorni da oggi, usa `TIMESTAMP=$(date -d "+30 days" +%s)`.

Esempio 8: Script Complesso - OR con Timelock

Comandi

```
export EXT_DESCRIPTOR="wsh(
  or_d(
    pk([e7cdc822/84'/1'/0']tpubDDggoP.../0/*),
    and_v(
      v:pkh([4d633db5/84'/1'/0']tpubDD39Bo1.../0/*),
      older(144))))"
export INT_DESCRIPTOR="wsh(
  or_d(
    pk([e7cdc822/84'/1'/0']tpubDDggoP.../1/*),
    and_v(
      v:pkh([4d633db5/84'/1'/0']tpubDD39Bo1.../1/*),
      older(144))))"
```

Nota: Spendibile da chiave 1 immediatamente, oppure da chiave 2 dopo 144 blocchi.

Esempio 9: Multisig con Timelock

Comandi

```
export EXT_DESCRIPTOR="wsh(
  and_v(
    v:multi(2,
      [e7cdc822/84'/1'/0']tpubDDggoP.../0/*,
      [4d633db5/84'/1'/0']tpubDD39Bo1.../0/*,
      [693e4e5e/84'/1'/0']tpubDC6YrbM.../0/*
    ),
    older(1008)
  ))"
export INT_DESCRIPTOR="wsh(...)"
```

Nota: Richiede 2 firme su 3 E almeno 1008 blocchi (circa 1 settimana).

Esempio 10: Script Complesso Multi-Condizione

Comandi

```
export EXT_DESCRIPTOR="wsh(
  andor(
    pk([e7cdc822/84'/1'/0']tpubDDggoP.../0/*),
    older(144),
    and_v(
      v:multi(2,
        [4d633db5/84'/1'/0']tpubDD39Bo1.../0/*,
        [693e4e5e/84'/1'/0']tpubDC6YrbM.../0/*,
        [0af3b444/84'/1'/0']tpubDDNg4Jv.../0/*
      ),
      older(1008)
    )))"
export INT_DESCRIPTOR="wsh(...)"
```

Nota: Spendibile da chiave singola dopo 144 blocchi, oppure da multisig 2-of-3 dopo 1008

Esempio 10b: Script Complesso con Timestamp

Comandi

```
export EXT_DESCRIPTOR="wsh(
  andor(
    pk([e7cdc822/84'/1'/0']tpubDDggoP.../0/*),
    after($TIMESTAMP_2025),
    and_v(
      v:multi(2,
        [4d633db5/84'/1'/0']tpubDD39Bo1.../0/*,
        [693e4e5e/84'/1'/0']tpubDC6YrbM.../0/*,
        [0af3b444/84'/1'/0']tpubDDNg4Jv.../0/*
      ),
      after($TIMESTAMP_2026)
    )))"
```

Esempio 10b: Script Complesso con Timestamp

Comandi

```
export INT_DESCRIPTOR="wsh(...)"
```

I timestamps sono stati calcolati con i seguenti comandi.

```
TIMESTAMP_2025=$(date -d "2025-01-01 00:00:00 UTC" +%s)
TIMESTAMP_2026=$(date -d "2026-01-01 00:00:00 UTC" +%s)
```

Nota: Spendibile da chiave singola dopo il 1 Gennaio 2025 (timestamp 1735689600), oppure da multisig 2-of-3 dopo il 1 Gennaio 2026 (timestamp 1767225600). I timestamp sono indipendenti dalla velocità di mining e più precisi per date specifiche.

Esempio 11: Taproot con Script Path

Comandi

```
export EXT_DESCRIPTOR="tr(
  [e7cdc822/84'/1'/0']tpubDDggoP.../0/*,
  {
    and_v(
      v:pk([4d633db5/84'/1'/0']tpubDD39Bo1.../0/*),
      older(144)
    ),
    pk([693e4e5e/84'/1'/0']tpubDC6YrbM.../0/*)
  })"
export INT_DESCRIPTOR="tr(...)"
```

Nota: Taproot con key path (spesa immediata con chiave principale) e script path (timelock o chiave alternativa).

Best Practices

Sicurezza

- 🔑 Non condividere mai chiavi private
- 🔒 Usa sempre testnet/signet per test
- 👀 Verifica sempre i descriptor prima di usarli
- ✓ Testa script complessi prima di usarli su mainnet

Performance

- 🚀 Usa `sync` regolarmente per aggiornare lo stato
- ⚙ Per script complessi, considera il costo delle fee
- ⚙ Salva i descriptor in modo sicuro

Esempio 12: Invio Fondi

Descriptors

```
export EXT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tprv8gzeeyNhJUbWpdZ.../0/*),
    older(2)
  )
)"
export INT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tprv8gzeeyNhJUbWpdZ.../1/*),
    older(2)
  )
)"
```

Esempio 12: Invio Fondi

Comandi

```
# Setup wallet
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL new_address
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL sync
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL balance
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL unspent
```

Esempio 12: Invio Fondi

Comandi

```
# Crea transazione con UTXO specifici
DESTINATION_ADDRESS="tb1q788lr08rrftt3sxtu0d8wea2nymjdmutp9q6g7"
AMOUNT=1000
UTXO_TXID="cdc030697e18badcaeb1c95f8785dc181944aa6c00db825a4de30b633859e4e6"
UTXO_VOUT=1

# Crea PSBT
PSBT=$(bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL create_tx --utxos $UTXO_TXID:$UTXO_VOUT \
  --to $DESTINATION_ADDRESS:$AMOUNT | jq -r .psbt)
```

Esempio 12: Invio Fondi

Comandi

```
# Firma la PSBT
SIGNED_PSBT=$(bdk-cli wallet --client-type electrum --database-type=$DATABASE_
  --url=$URL sign $PSBT | jq -r .psbt)

# Estrae la transazione finalizzata (hex)
FINAL_TX=$(bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYP
  --url=$URL extract_psbt "$SIGNED_PSBT" | jq -r .raw_tx)

echo "Finalized transaction hex:"
echo $FINAL_TX
```

Nota: Questo esempio mostra come creare una transazione spendendo un UTXO specifico. Il processo prevede: creazione della PSBT, firma con le chiavi private del wallet, e estrazione della transazione hex finalizzata. La transazione hex può essere poi inviata alla rete Bitcoin.

Esempio 13: Timelock Fisso Domani

Comandi

```
# Calcola il timestamp per domani (o usa un timestamp fisso)
TOMORROW=1733664000
# Oppure calcola dinamicamente:
# TOMORROW=$(date -d "tomorrow 12:00:00 UTC" +%s)

echo "Timestamp domani (UTC): $TOMORROW"
echo "Data/ora: $(date -d "@$TOMORROW" -u '+%Y-%m-%d %H:%M:%S UTC')"
```

Esempio 13: Timelock Fisso Domani

Comandi

```
export EXT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tprv8gzeeyNhJUbWpdZ.../0/*),
    after($TOMORROW)
  )
)"
export INT_DESCRIPTOR="wsh(
  and_v(
    v:pk([e7cdc822/84'/1'/0']tprv8gzeeyNhJUbWpdZ.../1/*),
    after($TOMORROW)
  )
)"
```

Esempio 13: Timelock Fisso Domani

Comandi

```
# Setup wallet
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL new_address
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL sync
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL balance
bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL unspent
```

Esempio 13: Timelock Fisso Domani

Comandi

```
# Crea transazione spendendo un UTXO con timelock
DESTINATION_ADDRESS="tb1q788lr08rrftt3sxtu0d8wea2nymjdmutp9q6g7"
AMOUNT=1000
UTXO_TXID="3ac761498f9aacca2b8b8ceb02ca131a4f591befe89fa05ac6ea1638c57c4d7f"
UTXO_VOUT=0

# Crea PSBT
PSBT=$(bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYPE \
  --url=$URL create_tx --utxos $UTXO_TXID:$UTXO_VOUT \
  --to $DESTINATION_ADDRESS:$AMOUNT | jq -r .psbt)
```

Esempio 13: Timelock Fisso Domani

Comandi

```
# Firma la PSBT
SIGNED_PSBT=$(bdk-cli wallet --client-type electrum --database-type=$DATABASE_
  --url=$URL sign $PSBT | jq -r .psbt)

# Estrae la transazione finalizzata (hex)
FINAL_TX=$(bdk-cli wallet --client-type electrum --database-type=$DATABASE_TYP
  --url=$URL extract_psbt "$SIGNED_PSBT" | jq -r .raw_tx)

echo "Finalized transaction hex:"
echo $FINAL_TX
```

Nota: Questo esempio mostra un timelock assoluto basato su timestamp Unix. Gli UTXO creati con questo descriptor possono essere spesi solo dopo il timestamp specificato. Il timelock è verificato al momento della creazione della transazione: se il timestamp corrente è inferiore al timelock, la transazione non sarà valida e verrà rifiutata dalla rete.

Esempio: Mempush & Broadcast della Transazione

Cos'è mempush?

mempush è un servizio che consente di inviare (broadcast) rapidamente una transazione Bitcoin raw (in formato hex) direttamente ai nodi, inserendola nella mempool pubblica. Utile per testare facilmente su testnet, signet o mainnet senza una propria istanza di nodo Bitcoin.

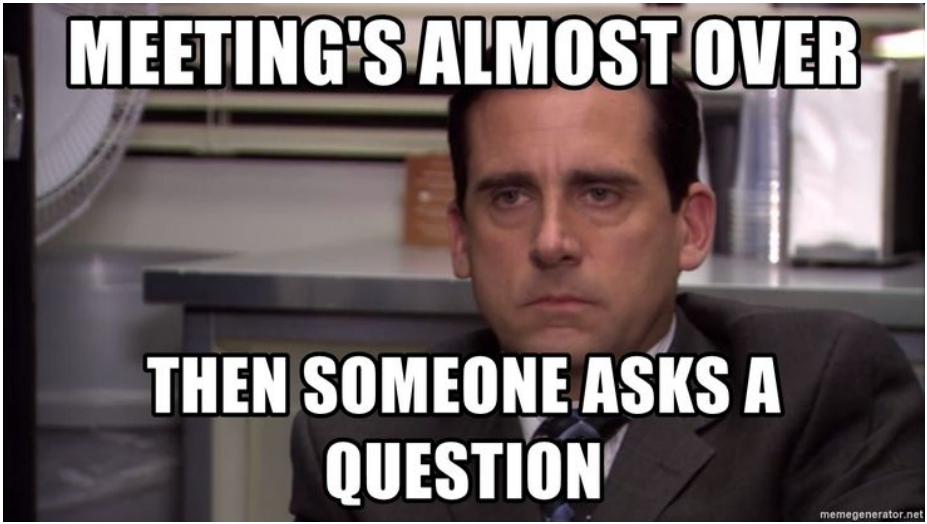
Come broadcastare una transazione con mempush

```
curl -X POST https://mempush.com/signet/api/transaction/push -H
  "Content-Type: application/json" -d '{"raw_tx": "$FINAL_TX"}'
```

Nota:

- Cambia `/signet` con `/testnet` o `/mainnet` a seconda della rete desiderata. - In alternativa, puoi copiare la transazione hex e incollarla nell'interfaccia web su https://mempush.com/signet/
- **Risultato:** Se la transazione è valida, riceverai un JSON di conferma dalla API. - Puoi monitorare la transazione su un explorer, ad esempio mempool.space.

? Domande


MEETING'S ALMOST OVER
THEN SOMEONE ASKS A QUESTION
memegenerator.net

Bibliografia

- BIP 380 - Output Script Descriptors
- BIP 383 - Miniscript
- BDK Documentation
- BDK GitHub Repository
- Miniscript Website

Progetto Satoshi Spritz

- 💻 Federazione di gruppi locali di Bitcoiner
- 🎓 Eventi gratuiti e privacy oriented
- 🤖 BITCOIN ONLY
- 🔧 Satoshi Spritz Connect online settimanale
- 📖 Orientato all'apprendimento della self-sovereign
- 🍕 Tutte le settimane un evento online -> Satoshi Spritz Connect

Links

- satoshispritz.it
- t.me/SatoshiSpritzConnect

Officine Bitcoin

- Comunità Italiana di Bitcoiners, totalmente gratuita
- BITCOIN ONLY
- Focus su educazione e sviluppo di progetti
- Progetti:
  - Sviluppo nodi Bitcoin
  - Uso di Hardware Wallet
  - Filosofia open source
  - Installazione di Debian
  - Mnemoniche & Dadi
  - . . . e molto altro

Links

- officinebitcoin.it