Jade DIY

Create your Jade!

👤 Riccardo Casatta
👤 Valerio Vaccaro

Plan ₿ Week

📅 2025-10-20

👤 Riccardo Casatta 👀

- 💻 Bitcoin Developer and Software Engineer
- 🦀 Rust enthusiast and maintainer of several Bitcoin projects
- ₿ Bitcoin and Liquid Engineer at Blockstream

👤 https://x.com/RCasatta

🐙 https://github.com/RCasatta

👤 Valerio Vaccaro 👀

- 💻 Bitcoin Developer and Hardware Expert
- 🔥 Contributor to Open Source Bitcoin Projects
- ⚠️ Passionate about DIY Hardware
- ₿ Bitcoin and Liquid Engineer at Blockstream

👤 https://www.linkedin.com/in/valeriovaccaro/

🐙 https://github.com/valerio-vaccaro/

Meme

⚠️ **Not your keys, not your Bitcoin!**

License

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

Summary

- 🔒 What is a Hardware Wallet? (main functionalities, examples, etc.)
- 💻 Jade
- 🔥 Jade DIY (build your Jade)
- 🤖 Simple test
- 💀 Advanced topics 🚀
- 👀 Questions?

🔒 What is a Hardware Wallet?

- **A dedicated device designed to securely store your Bitcoin private keys**
- **Signs transactions safely on-device:** your private keys never leave the hardware, and you can always review what you are signing
- Can be connected to a computer or smartphone, but the secrets are never exposed
- Adds an extra layer of security and control to your funds
- Protects against malware, remote attacks, and phishing attempts
- 🔥 Makes self-custody of Bitcoin both **practical and secure**

What is a hardware wallet?

- **Securely stores** your Bitcoin private keys and keeps them isolated from your computer or phone
- **Signs** transactions, ensuring you can check the details directly on the device's screen
- Allows you to generate **public keys** and receive Bitcoin **addresses**
- Supports creating recovery phrases (mnemonics) with the option of extra security using external entropy
- Adds an extra layer of protection against malware, remote attacks, and phishing
- Makes self-custody of Bitcoin practical, safe, and accessible for everyone

What is NOT a hardware wallet?

- **Not a backup solution** for mnemonics; you must handle backups yourself
- **Not a transaction creator**; you use a software wallet for that
- **Not a portfolio management tool**; it does NOT calculate balances or track transaction history—this is done by your wallet software
- ⚠️ You can generate mnemonics on-device but ...

🔒 Entropy

- 🎲 **Entropy**: measure of randomness/unpredictability
- Used to generate:
  - Mnemonic phrases (BIP39)
  - Nonces
  - Random outputs, GUI randomness, and more
- ⚠️ Security depends on entropy: Weak entropy = weak keys, and predictable entropy = predictable keys

Entropy

Hardware Sources

- True Random Number Generators (TRNG)
- Temperature sensors
- Electronic and radio noise
- Clock jitter

User Sources

- GUI movements
- Timing of incoming messages
- Key press intervals
- Camera input

But how can we tell if the entropy is random enough?

Entropy Quality Analysis

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
import os

with open('random.bin', 'wb') as file:
    for i in range(1,1024*1024):
        seed = os.urandom(32)
        file.write(seed)
```

In [2]:
```
from hashlib import sha256

seed = 'pippo'.encode('utf-8')

with open('random2.bin', 'wb') as file:
    for i in range(1,1024*1024):
        seed = sha256(seed).digest()
        file.write(seed)
```

In [ ]:

Entropy Quality Analysis

In [1]:
```
cat bible.txt | ent
```

```
Entropy = 4.596758 bits per byte.

Optimum compression would reduce the size
of this 4451368 byte file by 42 percent.

Chi square distribution for 4451368 samples is 68853960.24, and randomly
would exceed this value less than 0.01 percent of the times.

Arithmetic mean value of data bytes is 86.2069 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.109292 (totally uncorrelated = 0.0).
```

In [2]:
```
cat random.bin | ent
```

```
Entropy = 7.999995 bits per byte.

Optimum compression would reduce the size
of this 33554400 byte file by 0 percent.

Chi square distribution for 33554400 samples is 249.31, and randomly
would exceed this value 58.87 percent of the times.

Arithmetic mean value of data bytes is 127.4935 (127.5 = random).
Monte Carlo value for Pi is 3.142053501 (error 0.01 percent).
Serial correlation coefficient is 0.000032 (totally uncorrelated = 0.0).
```

In [3]:
```
cat random2.bin | ent
```

```
Entropy = 7.999995 bits per byte.

Optimum compression would reduce the size
of this 33554400 byte file by 0 percent.

Chi square distribution for 33554400 samples is 252.71, and randomly
would exceed this value 52.87 percent of the times.

Arithmetic mean value of data bytes is 127.4897 (127.5 = random).
Monte Carlo value for Pi is 3.142155783 (error 0.02 percent).
Serial correlation coefficient is -0.000041 (totally uncorrelated = 0.0).
```

In [ ]:
```

```

🔒 Secure Elements

Some hardware wallets have a Secure Element (SE) chip, others don’t. What is it?

✅ Advantages:

- Dedicated hardware chips for security with physical tamper resistance
- Secure storage of cryptographic keys
- Cryptographic operations in an isolated environment
- Protection against side-channel attacks

❌ But there are drawbacks:

- Often closed-source chips
- Potential backdoors/uncontrollable entropy
- Added cost

Hardware Wallet & Secure Element

Hardware wallets may include (from least to most secure):

- 0 secure elements
- 1 secure element
- More than 1 secure element
- Software-based and open-source secure elements (like Jade)

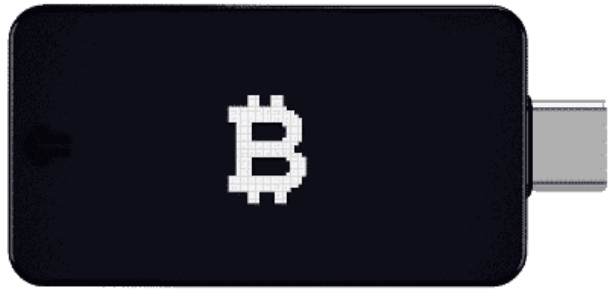
And then, there are devices that don't store anything at all... (stateless)

Examples: Jade

- Open-source hardware wallet developed by Blockstream
- Supports Bitcoin and the Liquid Network
- USB-C and Bluetooth compatible
- Large color screen, QR code support
- Designed for privacy and air-gapped operation
- Extensively documented DIY build process

Examples: BitBox02

- Open-source hardware wallet by Shift Crypto
- Focused on Bitcoin and security best practices
- Touch sliders for PIN and navigation
- MCU and secure chip architecture (with interface and code fully open)
- MicroSD backup



Examples: SeedSigner

- DIY, fully open-source Bitcoin hardware wallet
- Uses standard off-the-shelf parts (Raspberry Pi Zero, camera, screen)
- No specialized secure chip; stateless design—no secrets stored on device
- Camera-based QR code signing
- Targets maximum transparency and low-cost, accessible hardware
- Perfect for air-gapped cold storage and multisig setups

💻 Jade: Introduction

- **Jade** is Blockstream’s open-source hardware wallet for Bitcoin and Liquid.
- Built for transparency, privacy, and full user control.
- DIY-friendly: assemble from open-source instructions or buy pre-made.
- Rich documentation for users, tinkerers, and developers.
- Supported by many software wallets and libraries.

💻 Jade: Secure Hardware Features

- Large, high-resolution color screen for clear transaction review and device prompts.
- USB-C and Bluetooth for flexible connectivity.
- Cameras and QR support for fully air-gapped operation.
- Update/sign via USB memory stick (Plus only)

💻 Jade: Open Source and Auditable

- 100% open source firmware and hardware blueprints.
- Reproducible builds: verify that firmware matches the publicly reviewed code.
- Supports both Bitcoin and Liquid assets out of the box.
- Strong community of makers provides support for new boards.

💻 Jade: Backup, Recovery, and Compatibility

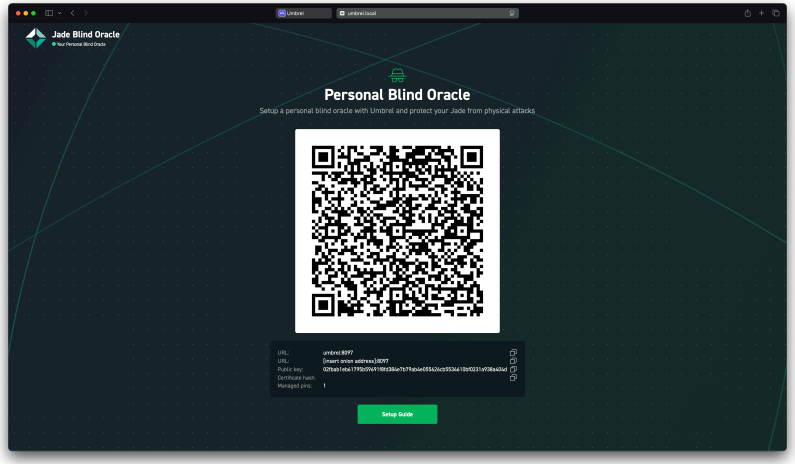
- Use standard 12- or 24-word mnemonic backups.
- Supports passphrase (BIP39) for plausible deniability.
- Allows mnemonics with external entropy
- Simple guided onboarding, recovery, and backup procedures (QR code/on-screen keyboard).
- Works with industry-standard wallet software (BlueWallet, Sparrow, Specter, Electrum, Green, and more).

💻 Jade: Advanced Functions & Multisig

- Fully supports multisig workflows and collaborative custody for institutions or groups.
- Implements BIP85
- Firmware is frequently updated to support the latest Bitcoin upgrades.
- Integrates with **Anti-Exfil** to avoid malicious signatures

💻 Jade + Blind PIN Server

- Supports **Blind PIN Server** (or Blind Oracle).
- A blind oracle functions as a virtual secure element and instead holds the decryption mechanism to your wallet off-device, making Jade invulnerable to physical key extraction. This is unlike typical secure element hardware devices, which hold everything needed to extract your keys on the actual device itself.
- By default, Jade communicates with Blockstream's blind oracle; however, you also have the choice to run your own if you'd like.



💻 Open Source

Why Open Source Matters:

- 🔍 Code can be audited by anyone
- 🐛 Faster discovery of vulnerabilities
- 👥 Community contributions
- 🛡️ No hidden backdoors

Risks of Closed Source:

- ❓ Unknown security practices
- 🕵️ Hidden functionalities
- 🔒 Vendor lock-in
- 💣 Susceptibility to supply chain attacks

⚠️ **But open source is not enough by itself.**

💻 Reproducible Builds 🚀

Reproducible builds are a software development process in which the same source code, built in the same environment, always produces identical binary outputs. This allows anyone to independently verify that a given binary was created from specific, unmodified source code.

✅ Benefits

- Supply chain security
- Build verification
- Community validation
- Malware detection

⚠️ **However, achieving reproducible builds requires significant effort.**

Jade DIY

The same firmware can also run on the following boards:

- M5Stack Core Gray/Black/Fire and also Core 2/S3
- M5StickC-Plus and Plus 2
- LilyGO TTGO T-Display
- LilyGO T-Display S3
- LilyGO T-Display S3 Pro Camera
- LilyGO t-Watch S3
- LilyGO T-Camera Plus
- ESP32-cam
- ... and many more.

You can find testing firmware on this page.

LilyGO T-Display-S3 Pro Camera

Prepare the system

This tutorial is developed on a Debian 13 AMD64 Linux machine. It can also work in a similar way on many Linux systems and on macOS.

```
sudo apt update
sudo apt upgrade
sudo apt install git python3-pip python3-venv cmake
```

Do you have privileges? If not, from root ...

```
/sbin/adduser my_user dialout  # for serial device privileges
/sbin/adduser my_user docker   # for docker usage
/sbin/adduser my_user sudo     # for root privilege, if you need it
```

Clone code

We can clone the code from the repository.

```
git clone --recursive https://github.com/Blockstream/Jade.git
cd $HOME/Jade
```

We want to install the latest release. We can get it based on the latest tag, in our case 1.0.36! (You can check using `git tag`.)

```
git checkout 1.0.36
git submodule update --init --recursive
```

Code Structure

The code is written in C using the ESP-IDF toolchain and libraries. The source code is organized in various directories:

- **configs**: This directory holds SDK configuration files (sdkconfig) for compiling the Jade firmware across different hardware variants and testing setups.
- **components**: A collection of custom ESP-IDF components used in the Jade firmware build process. These extend the base framework with Jade-specific functionality, such as tools for firmware diffing and updates.
- **diy**: Dedicated to do-it-yourself (DIY) hardware adaptations of the Jade firmware.

Code Structure

- **main**: The core directory for the Jade firmware source code, serving as the primary application entry point in the ESP-IDF project structure. It contains C/C++ source files, headers, and build scripts for the wallet's logic.
- **tools**: Utility scripts and Python tools for firmware preparation, building, and deployment.

A Python library is also provided in order to interact with Jade.

🔥 Using Docker

From the cloned repository, build images and log into the shell.

```
docker-compose up -d
docker-compose exec dev bash
```

Copy the configuration and flash your board.

```
cp configs/sdkconfig_jade.defaults sdkconfig.defaults
idf.py flash --port /dev/ttyACM0
```

⚠️ Fast, but you need Docker!

Install Environment

Install dependencies:

```
mkdir ~/esp
cd ~/esp
git clone -b v5.4 --recursive https://github.com/espressif/esp-idf.git
cd ~/esp/esp-idf
git checkout 67c1de1eebe095d554d281952fde63c16ee2dca0
./install.sh --enable-gdbgui esp32 esp32s3
python ~/esp/esp-idf/tools/idf_tools.py install qemu-xtensa
```

You can find the branch and commit in the project's README and in the Dockerfile.

⚠️ Each time you need esp commands you need to run the following command:

```
. $HOME/esp/esp-idf/export.sh
```

Configure

```
cp configs/sdkconfig_display_ttgo_tdisplays3procamera.defaults sdkconfig.defaults
```

And fix debug messages by adding if not present

```
CONFIG_LOG_DEFAULT_LEVEL_NONE=y
```

and removing

```
CONFIG_DEBUG_MODE=y
```

Or you can use

```
./tools/mkdefaults.py ./configs/sdkconfig_display_ttgo_tdisplays3procamera.defaults NDEBUG
```

Build/Flash

Now you can finally build the firmware and flash your device.

```
idf.py flash --port /dev/ttyACM0
```

If it is not working, check if you have privileges to the serial port (e.g. by adding yourself to the dialout group).

Simple test: Introduction

Sparrow Wallet is a powerful Bitcoin wallet designed for desktop use. It is ideal for Bitcoiners who value privacy, security, and versatility:

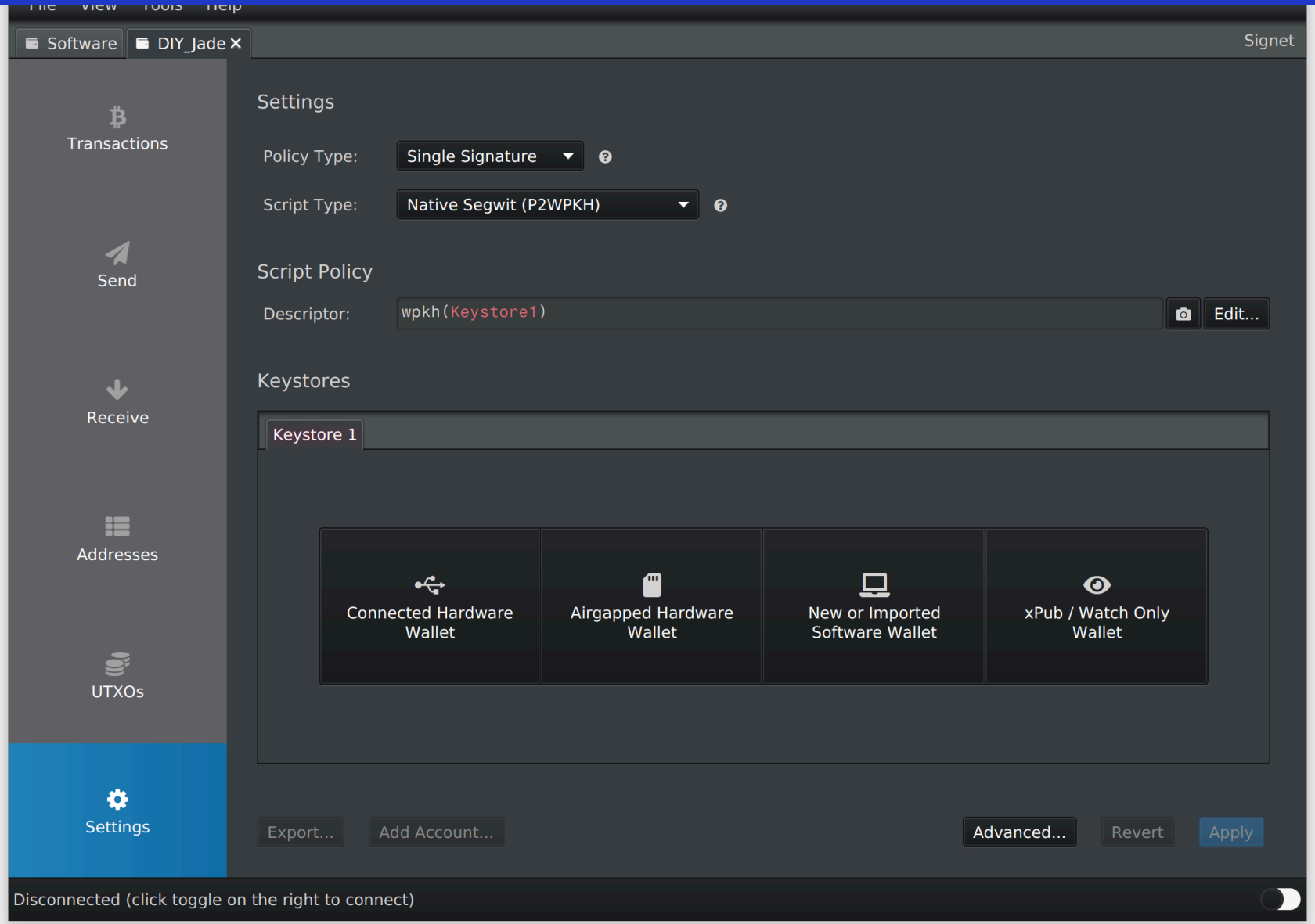
- **Open Source & Focused on Self-sovereignty**
- **Supports Airgapped Hardware Wallets**: including DIY devices like Jade, Specter, and Passport
- **Advanced Features**: multisig wallets, coin control, custom scripts, PSBT (Partially Signed Bitcoin Transaction) workflow
- **Works on Testnet, Signet, and Mainnet**
- **Great Interface**: intuitive UI for managing addresses, UTXOs, and coin selection

Simple test: Introduction

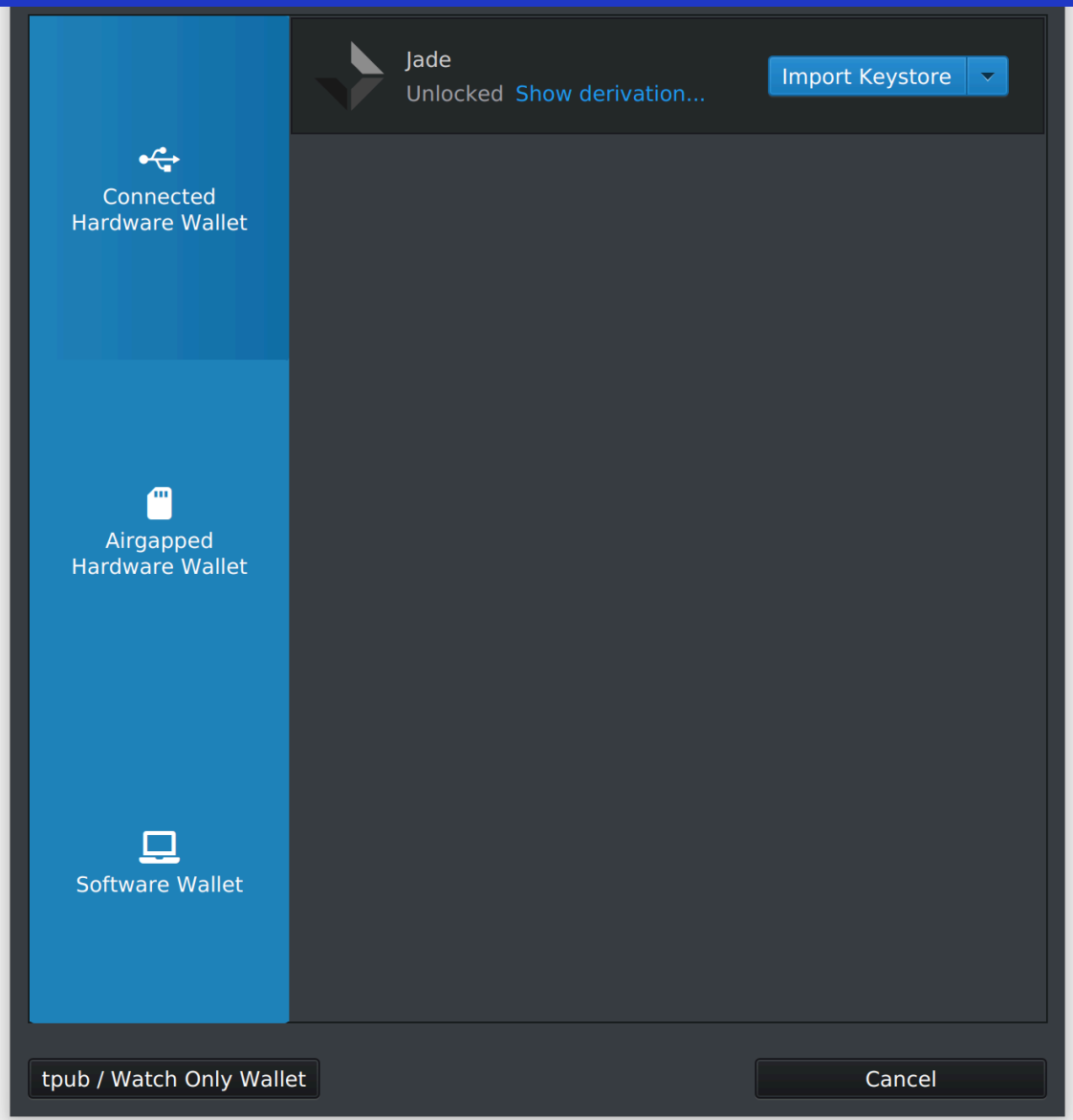
Signet is a public Bitcoin test network, designed for safe experimentation and development, without risking real bitcoin:

- **"Fake bitcoin"** is used on signet—no real value, free to obtain
- **Safer for Testing**: unlike testnet, blocks on signet are signed and reliable, reducing spam and instability
- **Similar Features to Mainnet**: allows you to experiment with real Bitcoin software and devices, simulating mainnet scenarios
- **Perfect for wallet development, testing firmware, or playing with new tools**

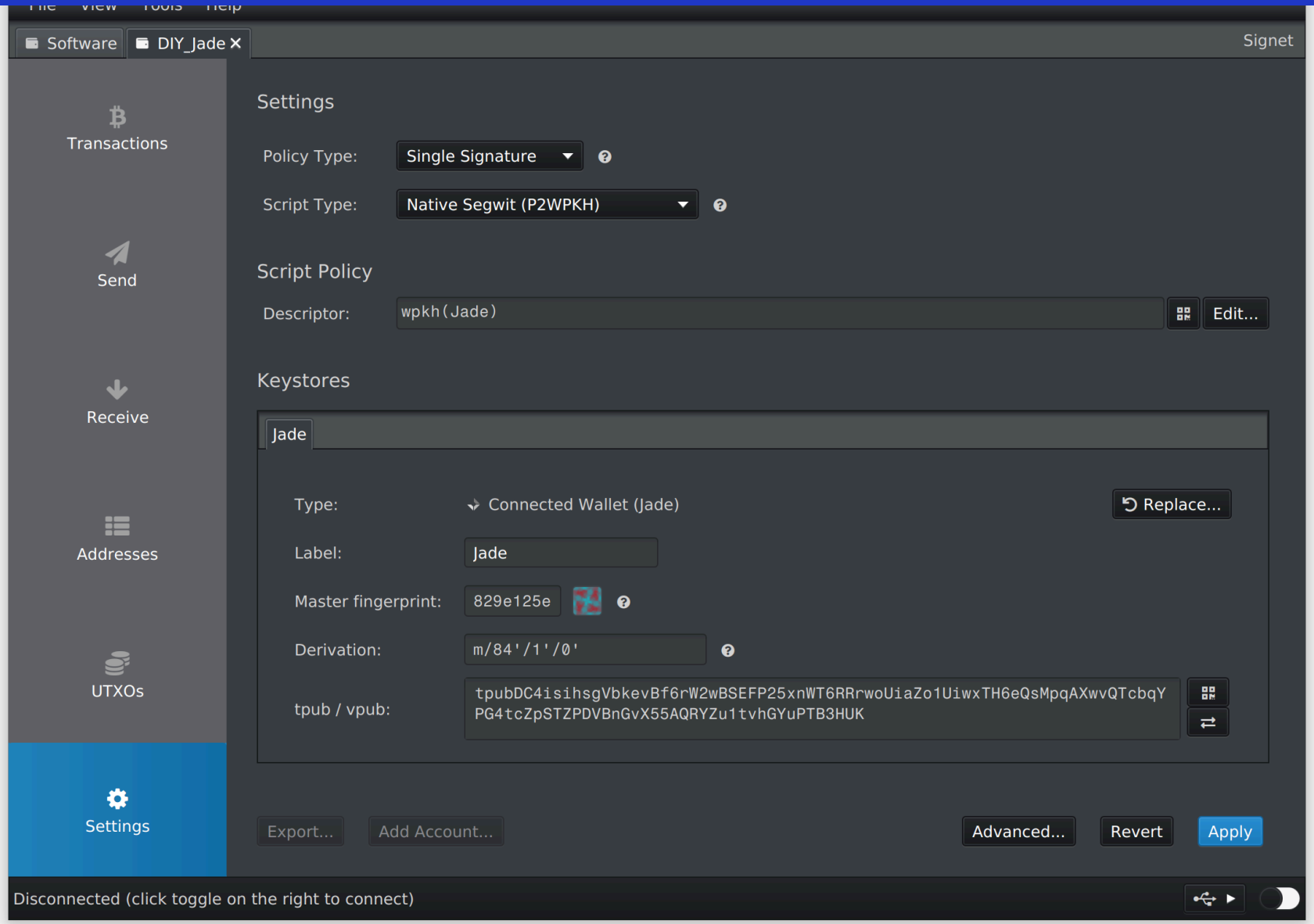
Simple test: Load mnemonic



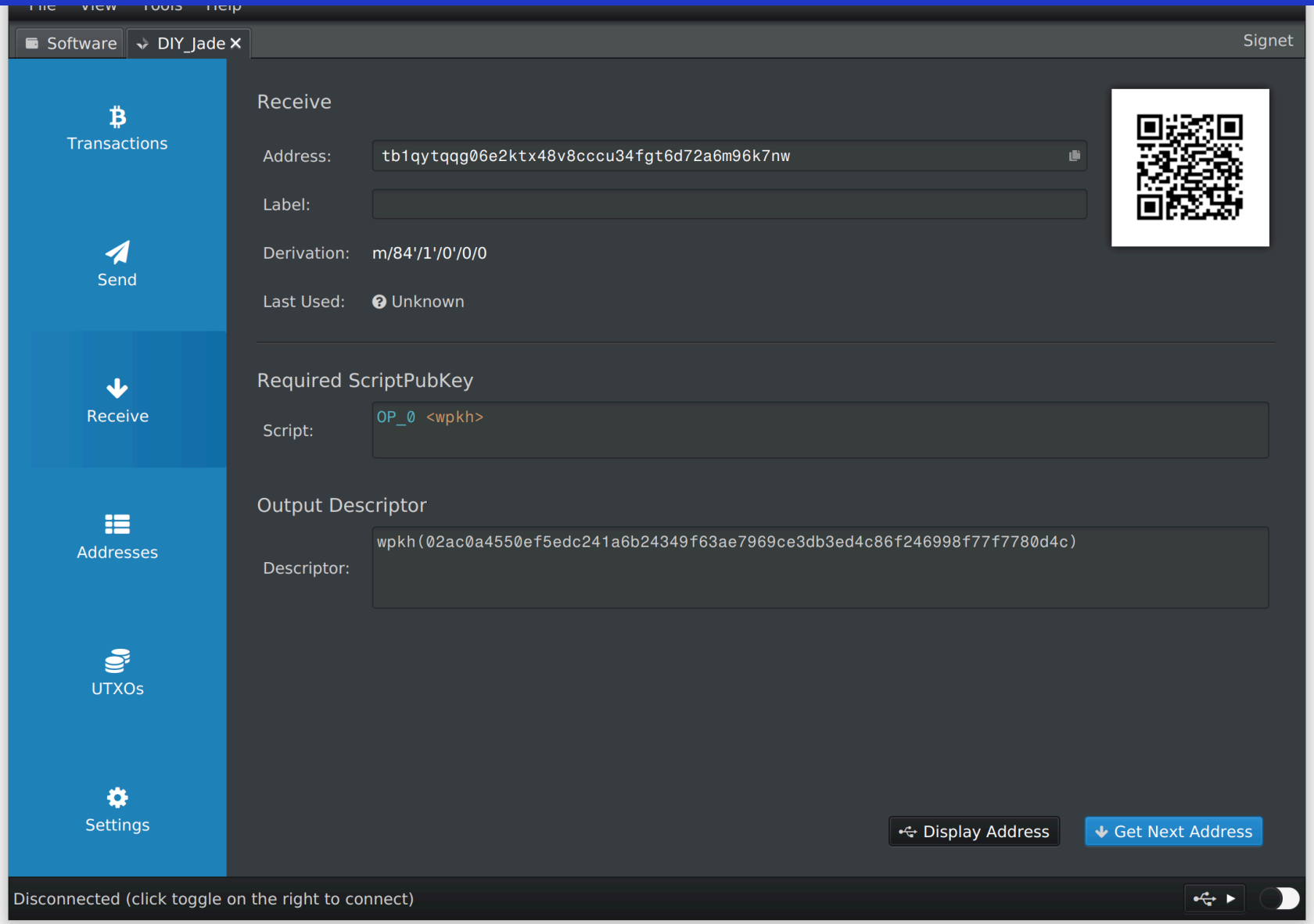
Simple test: Create new signer



Simple test: Create wallet



Simple test: Receive



Simple test: Spend

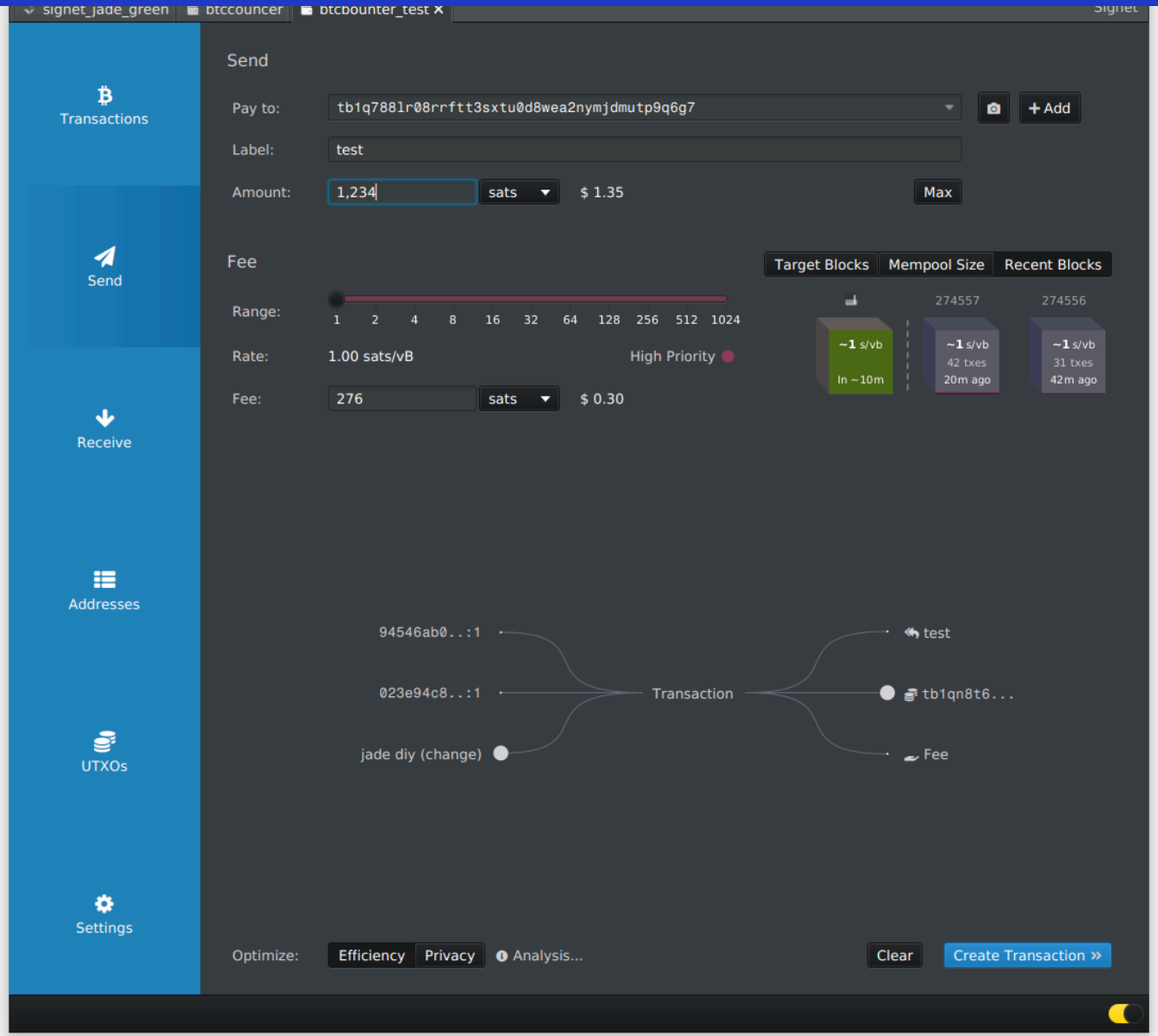
Exercise:

- Create a transaction that sends 1234 sats to an address.
- Sign the transaction using Jade and checking informations provided by Jade.
- Broadcast the signed transaction.

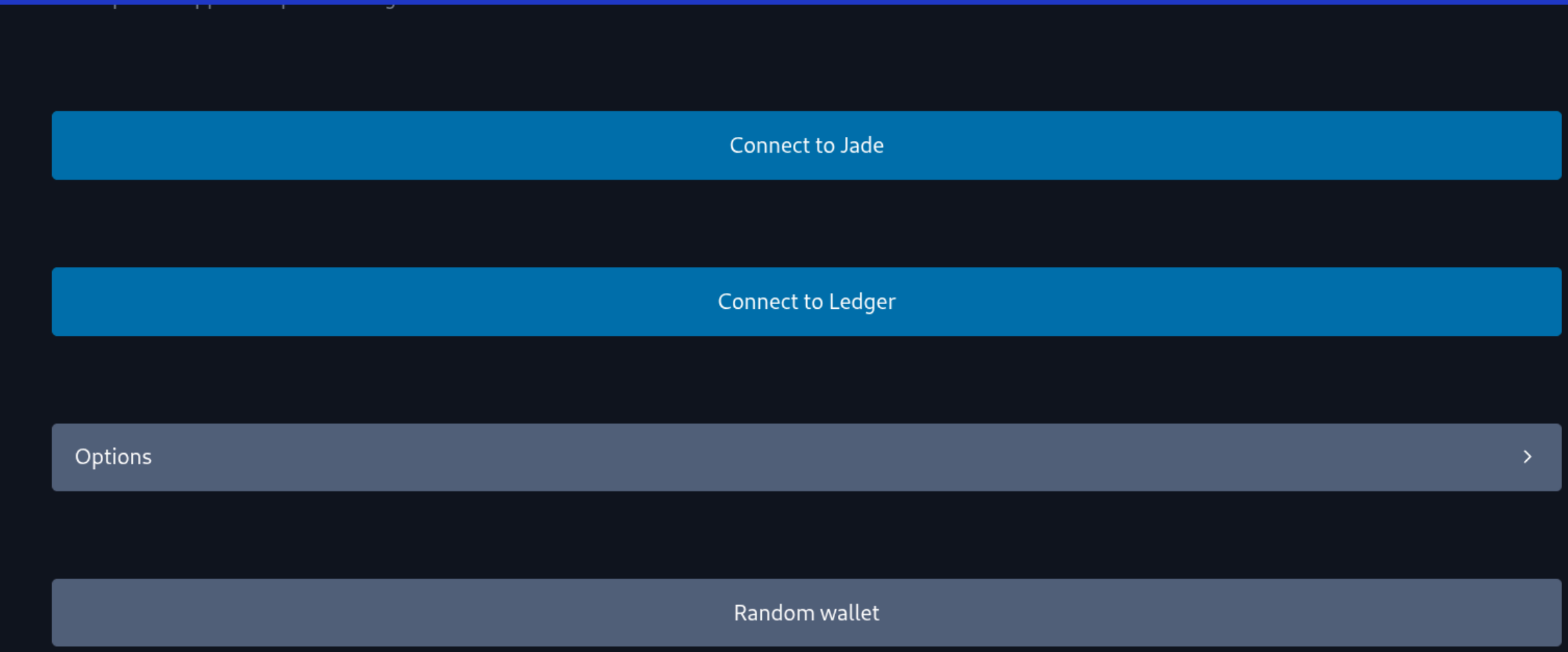
Exercise 2:

- Create a transaction with multiple outputs.
- Sign the transaction using Jade and checking informations provided by Jade.
- Broadcast the signed transaction.

Simple test: Spend



Simple test: Liquid Web Wallet 🚀



💀 Bootloader 🚀

On DIY hardware wallets without a securely installed bootloader, there is a serious security risk known as a **firmware swap attack**:

- An attacker can connect to the device via USB or serial and **replace the firmware** with a custom version that secretly exports private keys, bypasses security checks, or uses incorrect addresses.
- **The device then boots the malicious firmware,** which is often indistinguishable from the legitimate version to most users.
- As a result, the attacker may receive your funds or sensitive information, or the compromised device might store your secrets in flash memory for later retrieval.

💀 Bootloader 🚀

How to protect against this:

- Use hardware secure boot and flash encryption.
- **Ensure the bootloader is truly write-protected** (e.g., using ONE-TIME PROGRAMMABLE memory, eFuses, etc.).
- Whenever possible, only use devices that allow you to verify the integrity of the bootloader and firmware at all times.

This is why securing the bootloader is *absolutely critical* for DIY security devices!

💀 Bootloader 🚀

Enabling secure boot and encrypting flash memory are critical steps for protecting your device against unauthorized firmware modifications and data theft.

However, keep in mind:

- You must safely store your signing key in a secure location, as losing it may permanently prevent you from updating your device.
- Future firmware updates (including Over-the-Air, OTA) will require access to this signing key to generate valid update files.

To get started, generate your own signing key as follows:

```
espsecure.py generate_signing_key --version 2 ../jade-diy-v2.pem
```

⚠️ THIS KEY IS EXTREMELY IMPORTANT!

💀 Bootloader 🚀

Modify the configuration by adding

```
CONFIG_ESP32_DISABLE_BASIC_ROM_CONSOLE=y
CONFIG_SECURE_DISABLE_ROM_DL_MODE=y
CONFIG_SECURE_BOOT_SIGNING_KEY=<PATH_TO_YOUR_SIGNING_KEY>
CONFIG_SECURE_BOOT=y
CONFIG_SECURE_FLASH_ENC_ENABLED=y
CONFIG_SECURE_FLASH_ENCRYPTION_MODE_RELEASE=y
CONFIG_ESP32_REV_MIN_3=y
```

and removing

```
CONFIG_ESP32_REV_MIN_1=y
```

Or you can use

```
./tools/mkdefaults.py ./configs/sdkconfig_display_ttgo_tdisplays3procamera.defaults NDEBUG SECURE
```

💀 Bootloader 🚀

🔨 Now you can build a secure bootloader with the new configuration.

```
idf.py bootloader
```

✍️ Adapt the following command or check in logs.

```
esptool.py --chip esp32s3 --port=(PORT) --baud=(BAUD) --before=default_reset --after=no_reset --no-stub write_flash --flash_mode dio --flash_freq 80m --flash_size keep 0x0 ./build/bootloader/bootloader.bin
```

💀 Bootloader 🚀

🔨 Repeat to build the full firmware.

```
idf.py build
```

✍️ Adapt the following command or check in logs.

```
esptool.py --chip esp32s3 -b 460800 --before default_reset --after no_reset --no-stub write_flash --flash_mode dio --flash_size keep --flash_freq 80m 0x8000 build/partition_table/partition-table.bin 0x1a000 build/ota_data_initial.bin 0x20000 build/jade.bin
```

or simply

```
idf.py flash
```

💀 Bootloader 🚀

If you have enabled secure boot with the settings suggested above, you will need to do firmware updates via OTA.

An example command to do this using the jade.bin file in the /build folder would be:

```
python jade_ota.py --noagent
```

💀 Blind Pin Server 🚀

An alternative security model to traditional secure element chips.

- The secret required to decrypt the wallet's seed is stored remotely on a "blind oracle" server.
- The server only sees a hash of the user's PIN combined with a random nonce. It never has access to private keys or wallet addresses.
- When the user enters the correct PIN, the wallet requests the missing piece of the secret from the blind oracle using an encrypted channel (via ECDH key exchange).
- Once obtained, the seed is unlocked to authorize transactions.
- After three incorrect PIN attempts, both the device and the server delete sensitive data, rendering the wallet unusable without the recovery seed.
- Advanced users can run their own blind oracle server, reducing reliance on Blockstream's infrastructure.

💀 Blind Pin Server 🚀

Advantages:

- Fully transparent and open-source architecture.
- Eliminates costs and limitations of proprietary secure elements.
- Strong protection against physical attacks or key extraction attempts.
- Ability for users to self-host their own blind oracle server.

Already available on Umbrel and as a Docker Compose app! More integrations coming soon.

💀 Blind Pin Server Protocol v2: Message Flows 🚀

The Blind Pin Server Protocol v2 is a secure, privacy-preserving way to unlock wallet secrets using a remote PIN oracle, never exposing your seed or keys—even during unlock!

Let’s look at a simplified protocol, all messages are encrypted.

💀 Blind Pin Server Protocol v2: Message Flows 🚀

- **Client generates an ECDH keypair** and establishes a TLS-encrypted session to the server.
- **SET the PIN** via POST messages:

  Device sends:
  - `public_key`: client's ephemeral Curve25519 public key
  - `encrypted_secret`: wallet secret encrypted so only the holder of PIN + device keys can decrypt
  - `pin_hash`: (Argon2 hash of user's PIN and random nonce)
- **Server response:**
  - `device_id`: unique device identifier (used for subsequent requests)

💀 Blind Pin Server Protocol v2: Message Flows 🚀

- **GET the PIN** via POST messages:
- Client sends:
  - `device_id`
  - `public_key`: ephemeral public key for this session
  - `pin`: user's PIN (processed via Argon2id+nonce and sent as hash, never raw PIN)
- **Server verifies PIN hash:**
  - If correct and within allowed attempts:
    - Returns `encrypted_secret` (device decrypts this using its keys and the correct PIN)
  - If incorrect: increases attempt counter.

⚠️ **After 3 wrong PINs, the server deletes wallet information.**

Bibliography

- Jade source code
- Pin server source code
- Video, Windows-based
- Video, Linux-based 🇮🇹 In Italian, sorry 🐧

👀 Any Questions?

On Telegram: @valeriovaccaro

Satoshi Spritz Project

- 💻 Federation of local Bitcoiner groups
- 🎓 Free and privacy-oriented events
- 🤖 BITCOIN ONLY
- 📖 Focused on learning self-sovereignty
- 🍕 Satoshi Spritz Connect every week online

https://satoshispritz.it

https://t.me/SatoshiSpritzConnect

₿ Officine Bitcoin

- 🤝 Italian Bitcoiners community, totally free
- 🤖 BITCOIN ONLY
- 🎓 Focus on education and project development
- 📋 Projects:
  - 💼 Bitcoin node development
  - 👨‍🏫 Using Hardware Wallets
  - 💻 Open source philosophy
  - 🤝 Debian installation
  - ... and much more

https://officinebitcoin.it