Escrow-Less Bitcoin-Collateralized Lending

The Lendasat Team

Lendasat

11 August 2024

Abstract. Bitcoin is trustless, self-custodial money. Every attempt at using Bitcoin should respect these core tenets. And yet, every day, we watch more and more Bitcoin flow in the direction of a select few custodians. Bitcoin-collateralized lending is no exception to this unsettling pattern. With this work, we reject the status quo and show how to borrow using Bitcoin as collateral, the Bitcoin way.

Keywords: Bitcoin · Lending · DLCs · HTLCs

1 Introduction

Bitcoin—a scarce, decentralized currency—is the most liquid and secure asset in the world. However, unlocking the liquidity of Bitcoin without selling it has always been a challenge. For those who believe in Bitcoin's long-term value, selling is not an attractive option. It triggers capital gains taxes events, incurs delays, and——most importantly—risks losing a position in a highly profitable market, where re-entering at a favorable price may be impossible.

To circumvent these inefficiencies, Bitcoiners are compelled to borrow against Bitcoin. By using Bitcoin as collateral to borrow stablecoins or fiat currency, holders can access liquidity while retaining ownership of their coins. Their financial needs can be met without sacrificing their long-term investment.

Regrettably, traditional lending platforms require borrowers to transfer their Bitcoin to a third party, effectively giving up control of their assets. These custodial services pose significant risks:

- **Counterparty Risk**: The borrower is exposed to the risk of the custodian mismanaging, losing, or even confiscating their Bitcoin. The collapse of several exchanges and lending platforms in recent years highlights the dangers of entrusting your bitcoin to centralized entities.
- **Opaqueness**: Custodial platforms operate in a black box, often providing little to no visibility into how your assets are being managed or what security measures are in place.
- **Regulatory Uncertainty**: Centralized custodial services are subject to regulatory pressures, which can lead to frozen assets, forced liquidations, or other government-imposed restrictions that could leave borrowers in a precarious position.

We propose an alternative way: a self-custodial loan protocol to allow users to borrow against their Bitcoin without giving up control. This approach eliminates counterparty risk by ensuring that the borrower retains control of their assets throughout. We identify several key advantages:

- **Secure**: The borrower's Bitcoin never leaves their control, reducing the risk of loss or theft.
- **Transparent**: All transactions and collateral management are executed on the blockchain, providing complete transparency and trustless verification.
- **Peer-to-peer**: Aligned with the core principles of Bitcoin—empowering users to maintain control over their financial sovereignty.

In the following sections, we present a protocol that harnesses the capabilities of Discreet Log Contracts (DLCs) to fundamentally improve the state of Bitcoin-collateralized lending.

2 Hash Time Lock Contracts (HTLCs)

The basic building block for conditional payments on Bitcoin is the HTLC. With a HTLC, Alice can lock up coins in a Bitcoin contract with two predefined spending paths:

1. Her counterparty, Bob, can spend if he learns a *secret* only known to Alice: the preimage to a *secret hash*.
2. Alice can spend after a timelock expires.

Bob is able to purchase the *secret* from Alice to get the coins. The Bitcoin HTLC does not define how Alice may reveal the *secret*, although it is common for Bob to lock up value in a second HTLC using the same hashlock. Alice has to reveal the *secret* to claim whatever Bob locked up, allowing Bob to unlock the original HTLC.

A Miniscript policy for a basic HTLC may look like this:

```
or(
    and(pk(bob), sha256(secret))
    and(pk(alice), older(10)),
)
```

Alice is sending coins to Bob on the condition that Bob learns the *secret*. If Bob does not learn the secret before the timelock expires, Alice is able to recover her funds.

To date, HTLCs have been used extensively in Lightning payment channels, cross-chain atomic swaps and other contract protocols. In this work we show how HTLCs can be composed with other contracts to define nested spending conditions.

3 Discreet Log Contracts (DLCs)

DLCs are peer-to-peer Bitcoin contracts that can be unilaterally settled with the help of an oblivious oracle. Traditionally, both Alice and Bob provide coins as a wager, although single-funded DLCs are valid. The coins are divvied up based on the oracle's attestation to the outcome of a predefined event.

The original DLC protocol proposed by Dryja[2] leverages the linearity of Schnorr signatures[6] to sum participant public keys with oracle signature points. The computed public keys are used to lock coins that can only be unlocked if the oracle publishes one of several anticipated signatures.

The modern approach specified in `dlcspecs`[1] moves the oracle signature points out of the contract's public key. Coins are locked in a 2-of-2 multisig output shared by Alice and Bob. Multiple spend transactions, known as Contract Execution Transactions (CETs), are constructed and pre-signed. Instead of exchanging valid CET signatures, the participants share *adaptor signatures*[5][3], verifiably encrypted on different oracle signature points. The set of CETs and their corresponding adaptor signatures define the financial contract.

3.1 Hash Contract Execution Transactions (HCETs)

In the DLC protocol, a compliant oracle unlocks a single CET by attesting to a single outcome per event. The unlocked CET pays directly to either Alice, Bob or both, depending on the terms of the contract. The payout addresses are chosen by Alice and Bob, and they are usually P2WPKH addresses owned by either party.

The `dlcspecs` protocol does allow the participants to choose arbitrary payout addresses. We can use this to impose additional conditions on the CET payouts. For instance, one of the two participants may be required to provide a secret to claim the coins distributed to them by the oracle.

A HCET is a CET where one of the two payout scripts is a HTLC. In a DLC protocol involving HCETs, one party would act as a secret-holder and the other as the secret-learner. As such, all HCET payout outputs belonging to the secret-learner would be encumbered by a HTLC with two spending conditions:

```
or(
    and(pk(secret_learner), sha256(secret))
    and(pk(secret_holder), older(10)),
)
```

If the secret-learner discovers the secret in time, they are able to claim their share of any HCET. Otherwise, the secret-holder can claim the secret-holder's output (as well as their own regular output in the HCET).

HCETs can be used in protocols where one party provides all the collateral in the DLC, where the counterparty's claim on the collateral is contingent on other value being exchanged between the two participants. A loan protocol fits that description.

4 Protocol

The protocol we describe here involves two active participants: a borrower and a lender. The borrower holds bitcoin on the Bitcoin blockchain. The lender holds stablecoins on another blockchain, also known as the *loan blockchain*. The loan blockchain may also be the Bitcoin blockchain itself, provided there is a supply of stablecoin tokens on Bitcoin (e.g. stable coins issued on Taproot Assets or RGB).

The loan blockchain must have certain properties:

- Support for smart contracts. In particular, support for the SHA256 hash function and timelocks.
- Support for tokens, such as stablecoins.

Suitable candidates for the loan blockchain include Ethereum, TRON or Solana. L2s, such as Starknet or Polygon, may also qualify, despite not being traditional blockchains.

The borrower wants to borrow stablecoins, which they may use for payments or to access DeFi products outside of the Bitcoin ecosystem. The lender wants to earn interest on their stablecoin loan.

When describing this protocol, we assume that borrower and lender have already agreed to the terms of the financial contract, including principal amount, interest, loan term and loan-to-value (LTV) ratio.

4.1 Locking up the Bitcoin collateral

The protocol starts with the borrower locking up their collateral in a smart contract on Bitcoin. This *collateral contract* must have the following properties:

1. The borrower will be able to unilaterally claim back the collateral, if the lender never provides the loan principal.
2. The lender will be able to unilaterally liquidate the collateral, if the value of the collateral falls below an agreed upon level.
3. The lender will be able to unilaterally claim principal plus interest (*in bitcoin*) at maturity, if the borrower does not pay back the loan on the loan blockchain.
4. The borrower will be able to unilaterally claim their collateral (minus principal and interest), if they do not pay back the loan on the loan blockchain.

Before the *collateral contract* is published on Bitcoin, borrower and lender must collaborate to build the funding transaction and any possible spending transactions. This process is similar to the setup of a 2-party DLC, with certain tweaks.

Collateral secret generation The borrower generates a random 32-byte *collateral secret* and a *borrower keypair*. The borrower sends *collateral secret hash* and *borrower public key* to the lender.

Loan secret generation The lender generates a random 32-byte *loan secret*, a *lender keypair* and a *lender Bitcoin address*. The borrower sends *loan secret hash*, *lender public key* and *lender Bitcoin address* to the borrower.

Building the Bitcoin transactions The borrower constructs an *unsigned* Partially Signed Bitcoin Transaction (PSBT) of the *collateral funding transaction*.

The *collateral funding transaction* PSBT specifies: a list of borrower inputs to fund the collateral; a *collateral output*; a borrower change output. The *collateral output* specifies the collateral amount as well as the *collateral script*, built with the data shared by the lender in the previous step. The *collateral script* is described in detail in section 5.

The borrower also constructs three types of pre-signed transactions, spending from the *collateral output*: the *non-collaborative repayment transactions*, a set of HCETs used to enforce the loan contract at maturity; the *liquidation transactions*, a set of evenly distributed HCETs, allowing the lender to enforce liquidation throughout the lifetime of the loan; and the *refund transaction*, letting the borrower recover the collateral after the *collateral refund timeout*, a while after *loan term*. These pre-signed transactions are necessary to secure the protocol in case any party stops cooperating.

All HCETs in this protocol use the *collateral secret hash* to ensure that the lender can only access the collateral they may be owed if they know the *collateral secret*.

The borrower signs the *refund transaction* and provides an adaptor signature for every generated HCET. As specified in the DLC protocol, the lender will be able to transform any given adaptor signature into a valid signature if the oracle (or oracles) attests to a specific price at a specific time.

The borrower sends the unsigned *collateral funding transaction* PSBT, the *non-collaborative repayment transactions*, the *liquidation transactions*, the *refund transaction*, with corresponding signatures or adaptor signatures where necessary.

Complete signature exchange The lender verifies the validity of the transactions, signatures and adaptor signatures sent by the borrower. If the verification is successful, the lender proceeds with generating their own signature for the *refund transaction*, as well as an adaptor signature for every HCET received in the previous step. The lender sends these signatures and adaptor signatures to the borrower.

Publish collateral funding transaction The borrower verifies the validity of the signatures and adaptor signatures sent by the lender. If the verification is successful, the collateral contract is complete and the borrower is able to safely sign the *collateral funding transaction* and publish it to the Bitcoin blockchain.

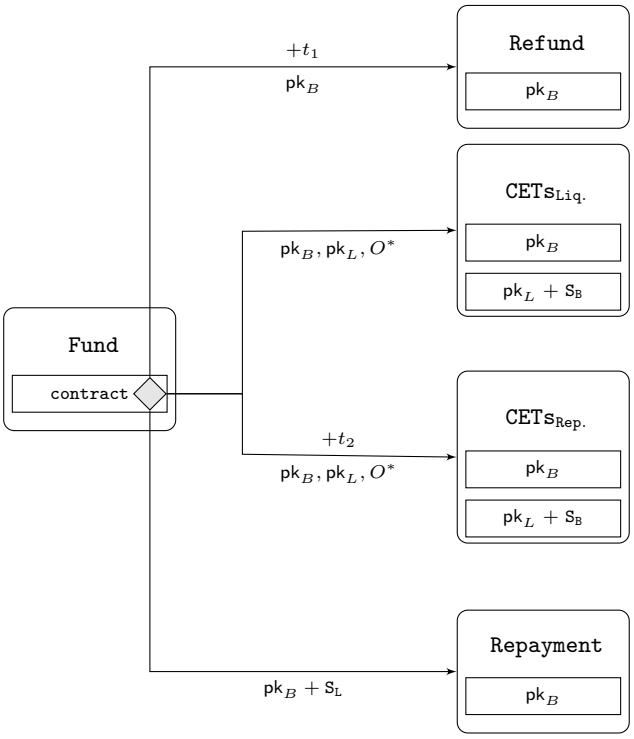

Fig. 1. Bitcoin collateral transaction schema.

4.2 Claiming the principal on the *loan blockchain*

The lender is now able to verify that the collateral is locked up in the Bitcoin blockchain. Once the *collateral funding transaction* reaches their desired number of confirmations, the lender proceeds with the protocol on the *loan blockchain*.

Using the smart contract capabilities of the *loan blockchain*, the lender locks up the principal in the *loan contract*, a smart contract with two spend conditions:

1. The borrower can spend if they authenticate based on the *borrower public key* and if they reveal the preimage to the *collateral secret hash* i.e. the *collateral secret*.
2. The lender can spend after a *loan refund timeout*. The timeout must be shorter than the *collateral refund timeout*, to prevent the borrower from both refunding their Bitcoin collateral and claiming the principal.

Once the borrower sees the locked-up principal on the *loan blockchain*, they claim it using the first spending path. The lender sees this and learns the *collateral secret*. The loan is now established.

Assuming normal oracle behavior, with knowledge of the *collateral secret* the lender will be able to unilaterally claim some or all of the collateral if the loan

value dips below the maintenance margin, or if the borrower fails to pay it back before maturity.

4.3 Cooperative repayment

If the borrower is ready to repay the loan, they can do so on the *loan blockchain*. To initiate repayment, the borrower locks up the repayment amount in the *repayment contract*. Borrower and lender must agree on the repayment amount for the repayment protocol to succeed. The *repayment contract* has two spend conditions:

1. The lender can spend if they authenticate based on the *lender public key* and if they reveal the preimage to the *loan secret hash* i.e. the *loan secret*.
2. The borrower can spend after a *loan repayment timeout*. The timeout must end before the loan reaches maturity, to prevent the lender from both unilaterally claiming principal plus interest from the Bitcoin *collateral contract* and claiming the repayment from the *repayment contract*.

Once the lender sees the locked up repayment amount on the *loan blockchain*, they claim it using the first spend path. The borrower sees this and learns the *loan secret*. The borrower now spends the *collateral contract*, using their *borrower keypair* to authenticate and the *loan secret* that was just revealed. The borrower must do this before the loan reaches maturity, to prevent the lender from triggering the unilateral repayment on the Bitcoin blockchain.

4.4 Alternative paths

After the borrower locks up the collateral on the Bitcoin blockchain, the loan protocol can deviate from the happy path in a number of ways. In the previous sections we have alluded to some of them, but here we describe them all explicitly.

The lender fails to lock up the principal There is no guarantee that the lender will lock up the principal on the *loan blockchain* in response to the borrower locking up the collateral. If the lender does not come through with the principal, the borrower is able to recover their collateral after the *collateral refund timeout*.

Since loan terms usually span multiple months, the borrower's collateral can end up stuck in the contract for a very long time. In section 6.2 we discuss how the protocol can be adapted to mitigate this problem.

The borrower never claims the principal Similarly, there is no guarantee that the borrower will claim the principal after the lender locks it up on the *loan blockchain*. In such a scenario, both participants will have to wait for their respective timelocks to expire:

- The lender waits for the *loan refund timeout*.

– The borrower waits for the *collateral refund timeout*.

Given that, for safety, the *loan refund timeout* has to be smaller than the *collateral refund timeout*, we are able to choose a much shorter *loan refund timeout* relative to the loan term. For example:

– Loan term: 12 months.
– *Collateral refund timeout*: 13 months.
– *Loan refund timeout*: 1 day.

The borrower fails to pay back the loan cooperatively Even though it is quicker and cheaper for the borrower to repay the loan cooperatively, it can happen that repayment has not been made by loan maturity.

To guarantee repayment of the loan value (including interest), the lender relies on the DLC path of the *collateral contract*. At loan maturity, the oracle attests to the price of Bitcoin in the stablecoin currency (often USD), unlocking a single HCET. The unlocked HCET contains one output for the lender, holding a Bitcoin amount equivalent (at the time of attestation) to the total repayment value in the stablecoin currency.

The lender publishes the signed HCET to the Bitcoin blockchain. Once confirmed, the lender spends their output using the *lender keypair* and the *collateral secret*.

The unlocked HCET pays any leftover collateral directly to the borrower. In fact, the borrower is able to execute the non-collaborative repayment themselves, as they possess their own version of the same HCET, also unlocked by the same oracle attestation that unlocked the lender's HCET.

The lender fails to claim the repayment If the borrower initiates the cooperative repayment protocol by locking up the repayment amount in the *loan blockchain*, they still depend on the lender to claim it, thus revealing the *loan secret*.

If the lender fails to do so, the borrower is able to refund the repayment amount after the *loan repayment timeout*.

As mentioned earlier, for safety, the *loan repayment timeout* must unlock the repayment amount for the borrower *before* loan maturity.

Once the loan repayment has been refunded back to the borrower, both participants can rely on the non-collaborative repayment path to settle the loan fairly, as described in the previous section.

Liquidation According to the financial terms of the loan, the loan may be liquidated before maturity. This can happen if the LTV ratio approaches 100%, although it will depend on the specific financial contract set up by borrower and lender.

If the value of the loan dips below the agreed upon threshold, the lender is able to unilaterally liquidate the collateral. Unilateral liquidation uses the same

mechanism as non-collaborative repayment i.e. HCETs unlocked by the oracle. The lender publishes an unlocked HCET which gives most or all of the collateral to the lender, if they can authenticate using the *lender keypair* and provide the *collateral secret*.

The primary difference with non-collaborative repayment is that liquidation HCETs are unlocked before loan maturity.

Oracle malfunction The ability to unilaterally enforce the rules of the loan depends on the oracle operating correctly. If the oracle attests to incorrect values, either party may be hurt. If the oracle fails to attest when expected, the lender may be hurt.

In particular, if the oracle fails to generate an attestation corresponding to loan maturity time, before the *collateral refund timeout*, the borrower will be able to recover their collateral without repaying the loan. In section 6.1 we discuss how to prevent this kind of scenario.

5 The collateral contract script

The *collateral contract script* is used to lock up the collateral on the Bitcoin blockchain. We use a Miniscript policy to specify the contract:

```
or(
    99@and(pk(borrower_0), sha256(loan_secret)),
    1@or(
        99@thresh(2, pk(borrower_1), pk(lender)),
        1@and(pk(borrower_2), after(1754611832))
    )
)
```

If we pass this policy to the `rust-miniscript:12.2.0` compiler we get the following SegWit Miniscript descriptor:

```
andor(
    pk(borrower_0),sha256(loan_secret),
    or_i(
        and_v(v:pkh(borrower_2),after(1754611832)),
        and_v(v:pkh(borrower_1),pkh(lender))
    )
)
```

We can also use the compiler to generate the following Tapscript Miniscript with an unspendable internal key:

```
tr(
    UNSPENDABLE_KEY,
    {
        {
            and_v(v:pk(borrower_2),after(1754611832)),
            and_v(v:pk(borrower_1),pk(lender))
        },
        and_v(v:pk(borrower_0),sha256(loan_secret))
    }
)#az4ls4sf
```

5.1 The Miniscript policy explained

The most likely path is the one that corresponds to the cooperative repayment of the loan:

```
and(pk(borrower_0), sha256(loan_secret))
```

This borrower can spend the collateral contract using this sub-policy by using their *borrower keypair* and the *loan secret*.

The second most likely outcome is that the *collateral contract* is spent with signatures from both the borrower and the lender:

```
thresh(2, pk(borrower_1), pk(lender))
```

This corresponds to the DLC path. Prior to the publication of the *collateral funding transaction*, borrower and lender collaborate to construct HCETs spending from this sub-policy. Instead of sharing valid signatures, they share adaptor signatures which will only become valid depending on the future attestation of an agreed upon oracle (or oracles).

Finally, the refund path is the least likely sub-policy:

```
and(pk(borrower_2), after(1754611832))
```

The borrower will have to wait until the timelock expires to spend using their *borrower keypair*. This path ensures that the collateral can be recovered in full if all else fails i.e. if borrower and lender fail to agree on a cooperative repayment of the loan *and* if the oracle fails to generate an attestation that enables non-collaborative repayment through the DLC.

6 Limitations and future work

6.1 Trusting the oracle

In the context of Bitcoin-collateralized loans, replacing the escrow with an oracle is advantageous for several reasons:

- The oracle is oblivious to the contract. They do not learn about the terms of the loan or even the existence of the loan.
- The oracle does not touch any coins. There is no multisig involving the oracle, so it's much harder for the oracle and one of the participant to collude and run away with the collateral.

The oracle is in a weaker position than the escrow, but borrower and lender still need to trust that the oracle will behave correctly.

A strategy to deal with this problem is to decentralize the oracle. Instead of relying on one oracle, who may be unreliable or untrustworthy, the DLC protocol can be carried out using a quorum of oracles. The CET (HCET in our protocol) adaptor signatures are locked in such a way that they require a threshold number of attestations from a set of oracles. If the required minimum number of oracles agree, a CET is unlocked.

The multi-oracle setup comes at the cost of complexity and speed. It is more work to manage announcements and attestations from multiple oracles, and generating and decrypting multi-oracle adaptor signatures is considerably slower. More efficient oracle attestation schemes can be used to mitigate these problems[4].

6.2 The borrower moves first

In our loan protocol, the borrower locks up the Bitcoin collateral first. After doing so, the borrower is at the mercy of the lender, who can back out of the deal at no additional cost. In that case, the longer the borrower has to wait to recoup their collateral, the more damaging the situation is for them. Since Bitcoin-collateralized loans can be open for months and even years, this problem cannot be ignored.

There are multiple ways to approach this problem:

1. Trust the lender to help the borrower recover the collateral early.
2. Introduce an early collateral refund path.

The first option might be sufficient in most scenarios. Having decided that they no longer want to go ahead with the loan, the lender has no incentive to *not* help the borrower. At the same time, the only incentive for the lender to help the borrower is to prevent reputational damage. Depending on the lender's profile, this may not carry any weight. Furthermore, in some cases the lender may simply disappear, with no ulterior motive to harm the borrower.

Thus, we should consider adding an early collateral refund path to the protocol. The properties of this alternative spending path are:

- The borrower can spend after a much shorter timeout relative to the *collateral refund timeout*.
- The lender can punish the borrower's spend attempt if the principal has already been claimed on the *loan blockchain* i.e. with knowledge of the *collateral secret*.

We cannot modify the *collateral contract script* to achieve this, because the lender needs to be able to punish the borrower on the publication of the *early collateral refund transaction*. As such, before the borrower publishes the *collateral funding transaction*, borrower and lender need to collaborate to build the *early collateral refund transaction*. This transaction spends the *collateral output* to a new shared output with the following spending conditions:

- The borrower can spend after an *early collateral refund timeout*.
- The lender can spend if they provide the *collateral secret*.

The timeout can be considerably shorter than the *collateral refund timeout*, but it needs to be long enough to allow the lender to punish the borrower if they are trying to run away with the collateral after claiming the principal. This imposes a sometimes-online requirement on the lender, who needs to periodically monitor the blockchain for the *early collateral refund transaction*, to be safe.

6.3 Partial repayment

The repayment scheme specified here expects the borrower to pay back in full. The repayment amount is defined when the loan is created: the sum of principal and total interest. The stablecoin value to be repaid is actually written into the rules of the DLC. If the borrower fails to make voluntary repayment, the oracle attestation ensures that the bitcoin returned to the lender has the expected stablecoin value (at the time of attestation).

It is often desirable for the borrower to repay part of the loan before maturity. To allow for this, we could add another spending path to the *collateral contract script*:

```
or(
    50@and(pk(borrower_0), sha256(loan_secret)),
    50@or(
        99@thresh(3,
            pk(borrower_1),
            pk(lender_0),
            sha256(partial_repayment_secret)
        ),
        1@or(
            99@thresh(2, pk(borrower_2), pk(lender_1)),
            1@and(pk(borrower_3), after(1754611832))
        )
    )
)
```

In the updated script, the partial repayment sub-policy is:

```
thresh(3,
    pk(borrower_1),
    pk(lender_0),
    sha256(partial_repayment_secret)
)
```

With this new spending path, borrower and lender can pre-sign a completely new *collateral funding transaction* spending from the *collateral output*, as well as all the other protocol transactions spending from the new *collateral output*. The new *collateral output* is reduced based on the repayment amount, with the leftover funds being sent to the borrower in a second output.

The lender is safe to pre-sign these transactions because the borrower is only able to publish them with knowledge of the *partial-repayment secret*. It is also **mandatory** that the lender uses a different signing key for this spend path. For the borrower to learn the *partial-repayment secret*, they will have to lock up the partial repayment amount on the *loan blockchain* using a hashlock with the *partial-repayment secret* as the preimage. Once the lender claims the partial repayment, the borrower is able to publish the new *collateral funding transaction*, recovering part of their collateral.

7 Conclusion

Here we have laid out a path for Bitcoiners to borrow against their Bitcoin, free from the yoke of custodians. If history[1] is anything to go by, the choice is clear.

References

1. dlcspecs: Specification for discreet log contracts. `https://github.com/discreetlogcontracts/dlcspecs` (2019), accessed: 2024-08-10
2. Dryja, T.: Discreet log contracts. `https://adiabat.github.io/dlc.pdf` (2017)
3. Fournier, L.: One-time verifiably encrypted signatures a.k.a adaptor signatures. `https://github.com/LLFourn/one-time-VES/blob/master/main.pdf` (2019)
4. Madathil, V., Thyagarajan, S.A., Vasilopoulos, D., Fournier, L., Malavolta, G., Moreno-Sánchez, P.: Cryptographic oracle-based conditional payments. `https://eprint.iacr.org/2022/499.pdf`
5. Poelstra, A.: Scriptless scripts. `https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf` (2017), accessed: 2024-08-10
6. Schnorr, C.: Efficient identification and signatures for smart cards. Pages 239–252 (1990)

[1] `https://bitcoinmagazine.com/business/sam-bankman-fried-and-ftx-largest-crypto-fraud-history`