The Lava Loans Protocol v2

Lava Global Inc

July 22, 2024

Abstract

Bitcoin has secured its position as a digital store of value. As a result, many people want access to bitcoin-secured loans. However, there exist no trust-minimized solutions enabling the use of native-bitcoin as collateral for loans. We present a solution that uses bitcoin smart contracts (discreet log contracts) to enable bitcoin-secured loans that offer users cryptographic security rather than exclusively reputational or legal assurances.

1 Background

Borrowing against your bitcoin can be a strategic move as means to streamline tax commitments or amplify BTC exposure. Yet, accessing bitcoin-secured loans is costly and slow, and all available solutions require users to trust centralized entities, which in 2022 alone lost over 24 billion in user funds. People need a better solution: the Lava Loans Protocol.

1.1 Objectives

Borrowers want the lowest cost loan. Lender want to earn a risk-adjusted rate of return. Both want an accessible and simple user-experience. Just as importantly, they want a secure and reliable loan process that meets the following objectives:

Initiation Loan initiation should be trustless. At no point should the lender be able to decline giving the borrower the loan capital after the borrower has deposited their bitcoin collateral, and neither should the borrower be able to run away with their collateral while having access to the loan capital.

Repayment If the borrower repays the loan according to the original terms, their collateral should be returned atomically.

Liquidation Loans should only be liquidated if the collateral value falls below the loan-to-value threshold agreed to at loan initiation.

Expiration If the borrower does not repay the loan before the loan term ends, some of the collateral, equivalent to principal plus interest, is released to the lender, and the rest is sent to the borrower.

Fail-safe Collateral should never be lost.

The Lava Loans Protocol is the first bitcoin-secured loans experience that meets all the objectives of lenders and borrowers.

2 How It Works

Let's start by defining some terms. Loan capital asset refers to the asset being borrowed (ex. a digital dollar). Interest accrues in the loan capital asset.

r	APR (annual percentage interest rate) at which the loan is issued
T	number of days after loan initialization that the loan will expire
t	number of days that have transpired since loan initialization
V	collateral amount, denominated in bitcoin
L	borrowed amount, denominated in the loan capital asset
$X(t)$	price of bitcoin on day t, denominated in the loan capital asset

Table 1: Definitions

Loan Life-cycle

1. **Matching** Borrower and lender are matched.
2. **Initialization** Loan is initialized via a trustless swap, resulting in the bitcoin collateral being locked in the bitcoin smart contract and the loan capital being accepted by the borrower.
3. **Contract Execution**

   Liquidation If the loan value reaches a pre-determined LTV, the collateral is immediately released to the lender for liquidation.

   Expiration If the borrower does not repay the loan before the loan term ends, some of the collateral, equivalent to the principal plus interest, is released to the lender, and the rest is sent to the borrower.

   Repayment If the borrower repays the loan, the collateral is released to the borrower atomically with the repayment being accepted by the lender.

   Fail-safe If the loan expires and the oracles do not attest to the price of bitcoin, the borrower can still repay their loan and receive their collateral. Failing that, the collateral can eventually be claimed by the lender who should return the borrower's share. In the further extremity of the lender and oracle both being completely offline, the collateral can be claimed by the borrower.

2.1 Matching the borrower and lender

The borrower chooses the best loan offer where every offer includes:

Rates Offered interest rate for a specific loan request (amount, LTV, duration).

Liquidation LTV Loan-to-value ratio at which liquidation will occur (see section 2.3.1).

Oracles Set of oracles accepted by the lender and the threshold number required for execution.

Inter-oracle Precision If the set of oracles is of size > 1, this parameter specifies the minimum supported and maximum allowed difference (in bits) between oracles' prices for them to be used in contract execution. See the Multi-Oracle spec for details.

Expiry Precision Price precision used for distributing collateral at loan expiration, in bits (see section 2.3.2).

After the borrower has selected a loan offer, loan initialization can begin.

2.2 Initializing the contract

The process of contract initialization closely resembles that of a trustless atomic swap, with the bitcoin collateral transferring to the bitcoin smart contract atomically when the borrower receives their loan capital.

2.2.1 Escrow Funding

To initiate this "swap", the borrower sends the collateral (V) to a P2WSH address with the witness script shown below where `pub_b` and `pub_l` are the borrower's and lender's respective public keys, `locktime` is a relative locktime that releases V to the borrower if initialization fails, and `H(X)` is a SHA256 hashvalue whose preimage is known by the borrower.

2.2.2 Signature Exchange

After the escrow transaction has confirmed, the borrower sends to the lender certain information, including the confirmed escrow funding txid, initialization signature, and initialization hashvalue. The initialization signature is only usable in the `OP_ELSE` case due to the use of `OP_CODESEPARATOR` and restricts the transfer to only the collateral address. The lender verifies and stores this information. Then, the lender computes the repayment hashvalue and returns it to the borrower. Then, the borrower creates their adaptor signatures and sends them to the lender. The lender verifies the borrower's signatures, creates their own signatures, and returns them to the borrower. At this point, the contract is complete and the parties can proceed to the final steps of loan initiation.

2.2.3 Funding Alt-Chain Contract

After successfully exchanging information with the borrower, the lender requires only the preimage to `H(X)` to transfer V to the collateral address. The alt-chain contract requires that preimage from the borrower to claim L. The lender funds the alt-chain contract with $L + S$ (see section 2.3.3 for details on S) of loan capital asset.

2.2.4 Loan Acceptance

Once $L + S$ are secured in the alt-chain contract, the borrower verifies the alt-chain contract and then accepts L by revealing the preimage to `H(X)`. The lender uses the revealed preimage to transfer V to the collateral address.

If the borrower does not claim L, the lender can reclaim $L + S$ after a timeout.

Likewise, if the lender does not transfer V, the borrower can reclaim the bitcoin collateral via the `OP_IF` path after `locktime`. The escrow `locktime` must end after the alt-chain timeout to prevent the borrower from claiming L at the latest moment and then also reclaiming the collateral. The alt-chain contract can require a signature release from the lender to reclaim $L + S$ after the alt-chain timeout so the borrower does not have to wait for the full escrow `locktime` and can reclaim from escrow earlier.

At this point, loan initialization is complete: V is held in the collateral address, L is held by the borrower, and S remains in the alt-chain contract.

Below, you can see the witness script for the bitcoin escrow. It defines two cases:

1. **Cooperative case:** borrower and lender provide signatures, and borrower reveals the preimage to `H(X)`.

2. **Timeout case:** borrower provides a signature after `locktime`.

opcodes	stack after execution
	`<sig_l?> <preimage?> <sig_b>`
`<pub_b>`	`<sig_l?> <preimage?> <sig_b> <pub_b>`
`OP_DEPTH`	`<sig_l?> <preimage?> <sig_b> <pub_b> <depth>`
`OP_2`	`<sig_l?> <preimage?> <sig_b> <pub_b> <depth> OP_2`
`OP_EQUAL`	`<sig_l?> <preimage?> <sig_b> <pub_b> 0|1`
`OP_IF`	`<sig_b> <pub_b>`
`  OP_CHECKSIGVERIFY`	
`  <locktime>`	`<locktime>`
`  OP_CHECKSEQENCEVERIFY`	`<locktime>`
`OP_ELSE`	`<sig_l> <preimage> <sig_b> <pub_b>`
`  OP_CODESEPARATOR`	`<sig_l> <preimage> <sig_b> <pub_b>`
`  OP_CHECKSIGVERIFY`	`<sig_l> <preimage>`
`  OP_SIZE`	`<sig_l> <preimage> <size>`
`  64`	`<sig_l> <preimage> <size> 64`
`  OP_EQUALVERIFY`	`<sig_l> <preimage>`
`  OP_SHA256`	`<sig_l> <hash>`
`  H(X)`	`<sig_l> <hash> H(X)`
`  OP_EQUALVERIFY`	`<sig_l>`
`  <pub_l>`	`<sig_l> <pub_l>`
`  OP_CHECKSIG`	`0|1`
`OP_ENDIF`	`OP_1|OP_0|<locktime>`

2.3 Contract Execution

CETs (contract execution transactions) for loan liquidation or expiration are constructed as per the DLC specification. From the specification, it is important to understand ECDSA-Adaptor, NumericOutcomeCompression, MultiOracle and Transactions. The rest of the paper assumes knowledge of these. In addition to these CETs, the contract may terminate during repayment or the fail-safe.

While a loan is active within the Lava Loans Protocol, the bitcoin collateral is locked by the witness script shown below. Unlike the plain multisig used in standard discreet log contracts, this script has separate multisig execution paths for the borrower and lender and a third path that resembles a reverse of the atomic swap used in loan initialization. It uses definitions identical to the escrow script above, except that the preimage to `H(Y)` is known by the lender.

At a high level, the witness script before for escrow execution defines three validation cases:

1. **Lender executes:** the lender combines the borrower's adaptor signature with an attestation from the set of oracles to execute the contract.
2. **Borrower executes:** the borrower combines the lender's adaptor signature with an attestation from the set of oracles to execute the contract. This path is also used for refund via a time-locked signature from the lender.
3. **Repayment:** the borrower provides their signature and the preimage to `H(Y)`

opcodes	stack after execution
	0|1|2 <sig_b> <sig_l>|<preimage>
<pub_l>	0|1|2 <sig_b> <sig_l>|<preimage> <pub_l>
OP_2SWAP	<sig_l>|<preimage> <pub_l> 0|1|2 <sig_b>
<pub_b>	<sig_l>|<preimage> <pub_l> 0|1|2 <sig_b> <pub_b>
2	<sig_l>|<preimage> <pub_l> 0|1|2 <sig_b> <pub_b> 2
OP_PICK	<sig_l>|<preimage> <pub_l> 0|1|2 <sig_b> <pub_b> 0|1|2
OP_NOTIF	<sig_l> <pub_l> 0 <sig_b> <pub_b>
OP_CHECKSIGVERIFY	<sig_l> <pub_l> 0
OP_DROP	<sig_l> <pub_l>
OP_CHECKSIG	0|1
OP_ELSE	<sig_l>|<preimage> <pub_l> 1|2 <sig_b> <pub_b>
OP_CODESEPARATOR	<sig_l>|<preimage> <pub_l> 1|2 <sig_b> <pub_b>
OP_CHECKSIGVERIFY	<sig_l>|<preimage> <pub_l> 1|2
OP_1SUB	<sig_l>|<preimage> <pub_l> 0|1
OP_NOTIF	<sig_l> <pub_l>
OP_CHECKSIG	0|1
OP_ELSE	<preimage> <pub_l>
OP_DROP	<preimage>
OP_SIZE	<preimage> <size>
64	<preimage> <size> 64
OP_EQUALVERIFY	<preimage>
OP_SHA256	<hash>
H(Y)	<hash> H(Y)
OP_EQUAL	0|1
OP_ENDIF	0|1
OP_ENDIF	0|1

2.3.1 Liquidation

Let ϵ be a reasonably small value ($0 \le \epsilon \ll 1$), the liquidation slippage allowance. This defines the minimum ratio between the amount owed to the lender and the collateral value at which the lender may collect and liquidate the collateral. ϵ is non-zero because collateral sales are neither atomic nor instant. Therefore, the price of bitcoin may shift either before or during liquidation.

$$
\begin{aligned}
L_t &= L \cdot \left(1 + \frac{r}{365}\right)^t && \text{amount owed on day } t \\
LTV(t) &= \frac{L_t}{V \cdot X(t)} && \text{ratio between the amount owed and the collateral value} \\
LTV_l(t) &= 1 - \epsilon && \text{lowest loan to value ratio at which the loan can be liquidated} \\
X_l(t) &= \frac{L \cdot \left(1 + \frac{r}{365}\right)^t}{V \cdot (1 - \epsilon)} && \text{highest liquidation price on day } t
\end{aligned}
$$

If the set of oracles attests to a price of bitcoin below $X_l(t)$; then the lender can decrypt the borrower's corresponding signature adaptor, complete, countersign, and broadcast the liquidation CET.

Additionally, at initiation, the borrower provided the lender with signature adaptors on the string `<contract_id> liquidation` which are verified in the alt-chain contract to release the lender's stake (S).

2.3.2 Expiration

At loan expiration, bitcoin is owed to both the lender and the borrower:

$$
\begin{aligned}
Q_l &= \frac{L_T}{X(T)} && \text{collateral owed to lender} \\
Q_b &= V - Q_l && \text{collateral owed to borrower}
\end{aligned}
$$

There is a trade-off between the number of CETs and signature adaptors created and exchanged between the parties and the precision of the payout calculation. The NumericOutcome specification does not account for protocols where the desired outcome precision is expressed as a constant percentage of the outcome. We propose an alternative payout mechanism with a 1:1 relationship between CETs and signature adaptors, and a payout precision expressed as a number of bits of pricing precision for expiration prices, relative to the oracle-attested price. Each additional bit of precision approximately doubles the number of expiration CETs and signatures exchanged during initialization, which could make initialization prohibitively time consuming on some devices.

The maximum pricing error for a given number of bits can be calculated as follows:

$$
\begin{aligned}
bits_X &= 18 \rightarrow && \text{overall bit size of oracle price attestations, initially 18} \\
bits &= 1 \rightarrow (bits_X - 1) && \text{expiration CET price precision} \\
X_{trunc} &= 2^{bits_X - 1 - bits} - 1 && \text{maximum value truncated from price} \\
X_{minimax} &= 2^{bits_X - 1} + X_{trunc} && \text{minimum price with maximum truncation} \\
E_{max}(bits, bits_X) &= \frac{X_{trunc}}{X_{minimax}} && \text{maximum error ratio} \\
E_{max}(bits) &= \lim_{bits_X \rightarrow \infty} E(bits, bits_X) && \text{limit of maximum error} \\
E_{max}(bits) &= \frac{1}{2^{bits} + 1} && \text{simplified}
\end{aligned}
$$

This yields the following maximum error ratios:

bits	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
E_{max} bips	3333	2000	1111	588	303	154	78	39	20	10	5	2	1	<1	≈0

Given these maximum error ratios, $bits = 4 \rightarrow 15$ is the maximum recommended range. This range yields the following numbers of expiration CETs and signature adaptors for 18 and 20-bit oracle price attestations:

bits	4	5	6	7	8	9	10	11	12	13	14	15
CETs $bits_X = 18$	240	448	832	1536	2816	5120	9216	16k	29k	49k	82k	131k
CETs $bits_X = 20$	272	512	960	1792	3328	6144	11k	20k	37k	66k	115k	197k

This result demonstrates starkly the increased computation and bandwidth required for greater pricing precision.

The above maximum error ratios assume the use of either the floor or ceiling price within a rounding range for the calculation of the distribution. Alternatively, the midpoint could be used, resulting in half the maximum error at the expense of making the beneficiary of the error unpredictable.

For this protocol, the floor price within a rounding range is used for building the expiration CETs. Loan offers include the "expiration precision bits", allowing borrowers to select offers by this parameter.

The lender is able to claim their stake (S) from the alt-chain smart contract at expiration.

2.3.3 Repayment

To initiate repayment, the borrower sends L_t to the alt-chain smart contract. The alt-chain contract requires the lender to reveal the preimage to `H(Y)` to claim the repaid funds, and that preimage allows the borrower to reclaim the collateral.

$$i = L \cdot \left(1 + \frac{r}{365}\right)^t - L \qquad \text{the interest owed at time } t$$

$$I = L \cdot \left(1 + \frac{r}{365}\right)^T - L \qquad \text{total interest to be paid over the course of the loan}$$

$$S = I \qquad \text{stake held in the alt-chain contract while the loan is active}$$

$$I_{ro} = I - i \qquad \text{remaining interest on the original loan}$$

$$L_t = L + i \qquad \text{alternate representation of } L_t$$

$$L_t + I_{ro} = L + i + I - i \qquad \text{repayment amount plus remaining interest}$$

$$L_T = L + I \qquad \text{maximum loan value}$$

It is important that the lender cooperates when the borrower attempts to repay the loan. The incentive for the lender not to cooperate only arises when the interest rate on the existing loan is greater than the rate on a new loan. Therefore, the maximum incentive not to accept repayment exists when the new interest rate is 0, and is equal to the remaining interest to be collected from the existing loan (I_{ro}). This incentive is greatest at loan inception when the incentive is equal to the total interest to be paid over the course of the loan (I). The lender stakes S equal to I to incentivize them to accept even an immediate repayment of a high interest loan when no new loan could be issued.

If the lender does not accept the repayment by a specified timeout, the attempted repayment amount (L_t), plus a portion of the stake S equal to I_{ro} will be redeemable by the borrower. The total amount released to the borrower if acceptance times out is $L_t + I_{ro} = L_T$. If the lender does accept the loan repayment, the alt-chain contract releases the full $L_t + S$ to the lender.

2.3.4 Fail-safe

The fail-safe conditions are very unlikely. It happens if, through inaction by the oracles, inaction by the borrower, or any other reason, the loan collateral is not transferred by some reasonable time after the end of the loan term.

In this case, to make sure no bitcoin is burned, the lender is able to claim the collateral via a long-dated signature given to the lender by the borrower at loan initiation. If the lender doesn't do that, the borrower can also claim the collateral after a longer time-delay to make sure no bitcoin is ever burned.

3 Risks

There are three notable risks worth discussing:

1. **Oracle risk:** The set of oracles could collude with either of the parties. If the oracles collude with the lender, they could cause a liquidation event to execute even if the collateral is worth well over the loan value, causing the borrower to lose collateral value - loan value. If the oracles collude with the borrower, they could prevent liquidations from occurring or attest to a high price of bitcoin at expiration such that the majority of the collateral is unduly returned to the borrower.

2. **Loan capital risk:** Let us say the loaned capital is a stablecoin not backed 1:1 with dollars, and it loses its peg. Paying back the stablecoin amount becomes much easier for the borrower, and the borrower is inclined to do so. As such, the loan capital risk is largely put on the lender.

3. **Altchain risk:** If there is some issue with the altchain, then the borrower won't be able to repay the loan. This means the loan will necessarily expire, and some of the collateral will go to the lender.

While these risks exist, there are many ways to respond to them:

1. **Oracle risk:** Oracles for the Lava Loans Protocol are not aware that a contract is relying on their data. They have zero knowledge about the contracts. The oracles also have no direct incentive to misbehave if they are not a contract participant because they can not directly earn from being malicious since the bitcoin and alt-chain contracts restrict fund movements to only borrowers and lenders. Oracles are also chosen at loan initiation so borrowers and lenders can choose reliable oracles. Users can also choose multiple oracles. To make this easier, Lava has released an open-source oracle implementation called Sibyls which anyone can use to run an oracle.

   Compared to multisig signers, oracles are:

   a) more censorship-resistant because there are typically no limitations on how many oracles can influence a contract
   b) more easily chosen and more customizable since oracles require less coordination between parties
   c) private in the sense that they cannot see how/by whom their information is used

2. **Loan capital risk:** Lenders and borrowers can decide which loan capital asset they want to use.

3. **Altchain risk:** Lenders and borrowers can decide what altchain they want to use. Altchain failures don't cause safety issues, just liveness failures.

4 Conclusion

We present this protocol to enable more secure and simpler bitcoin-secured loans. Loan capital can theoretically live on any blockchain with the ability to execute smart contracts.