Maxy Mint Documentation

Published by Maxy. Last updated: 1 November 2026.

Contents

1. Introduction
   1.1 Purpose
   1.2 What distinguishes the system
   1.3 The issuer's token
   1.4 Scope and audience
2. Design overview
   2.1 Three steps
   2.2 Properties
   2.3 Comparison with custodial transfer
   2.4 Direct-custody mode
3. Key concepts
   3.1 Bitcoin terms
   3.2 Token and contract terms
   3.3 Roles
4. System components
   4.1 Components
   4.2 Enforcement layers
5. The deposit and minting process
   5.1 Steps
   5.2 Worked example
   5.3 The refund case
6. Flow of funds and custody
   6.1 Stages
   6.2 Encumbrance while in the HTLC
   6.3 Custody after the claim
   6.4 Reserve accounting
   6.5 Fees
7. Compliance and auditability
   7.1 The screening window
   7.2 Declining a deposit

7.3 The audit trail
7.4 Verification by a third party
7.5 Controls
8. Deposit lifecycle
8.1 States
8.2 Transitions
8.3 Time parameters
9. Redemption
9.1 Steps
9.2 Quote fields
9.3 Minimum payouts
10. Integrating with the issuer's token
10.1 Requirements on the token
10.2 Deployment options
10.3 Coexistence with other minting methods
10.4 Upgrades and replacement
11. Integration guide for wallets and front ends
11.1 Requesting an order
11.2 Registering the deposit
11.3 Funding the HTLC
11.4 Submitting the funding proof
11.5 Monitoring the mint
11.6 Refunding
11.7 Changing the recipient
11.8 Redeeming
11.9 Errors
12. Issuer operations
12.1 Keys and secrets
12.2 Signing an order
12.3 Watching and proving
12.4 Screening
12.5 Claiming
12.6 Revealing
12.7 Declining and closing
12.8 Paying redemptions
12.9 Monitoring

13. Security model
    13.1 Guarantees
    13.2 Assumptions
    13.3 Capabilities of each party
    13.4 Failure cases
    13.5 Operating rules
    13.6 Direct-custody mode
14. Contract reference
    14.1 Mint contract
    14.2 Token
    14.3 Verifier
    14.4 Redemption contract
15. Bitcoin script reference
    15.1 The witness script
    15.2 The address
    15.3 Spending the HTLC
    15.4 Verifying a supplied script
16. Deployment and parameters
    16.1 Networks
    16.2 Reference deployment (direct-custody mode)
    16.3 HTLC-mode deployment
    16.4 Parameters
    16.5 Build settings
17. Direct-custody mode
    17.1 How the mode differs
    17.2 Steps
    17.3 The order authorization
    17.4 The assignment and registration
    17.5 The deposit-first rule
    17.6 Refunds
    17.7 Trust model
    17.8 EIP-712 domains by mode
    17.9 Parameters

1. Introduction

1.1 Purpose

Maxy Mint is an issuance system for Bitcoin-backed tokens on Robinhood Chain. A Bitcoin-backed token is a token on Robinhood Chain whose every unit is matched by Bitcoin held in reserve by the issuer. Maxy Mint provides the contracts and procedures through which a user deposits Bitcoin, the deposit is verified on chain, and the issuer's token is minted to the user. It also provides a redemption path through which a holder returns the token and receives Bitcoin.

Maxy Mint has two modes. **Direct-custody mode** is live on Robinhood Chain today: the user pays Bitcoin to a deposit address held by the issuer, and the mint contract mints the token against a proof of that payment. **HTLC mode**, in which the deposit stays in a hash time-locked contract until the issuer claims it, is planned. Chapters 2 to 15 describe HTLC mode, the design the system is built toward. Chapter 17 describes direct-custody mode as deployed, and Chapter 16 gives the addresses of both.

1.2 What distinguishes the system

In a conventional custodial issuance, the user transfers Bitcoin to the issuer first and relies on the issuer's internal records for the token to follow. The transfer and the mint are separate events connected only by the issuer's bookkeeping. A third party cannot verify that a given mint was backed by a specific deposit at the time it occurred.

In HTLC mode, Maxy Mint changes the order of operations. The user deposits Bitcoin into a hash time-locked contract on the Bitcoin network. The issuer can take custody of the deposit only by publishing a secret value on the Bitcoin blockchain. The mint contract on Robinhood Chain requires that same secret before it will mint. The consequences are the following.

- The issuer cannot take the Bitcoin without also enabling the mint.
- If the issuer does not take the Bitcoin within a fixed time, the Bitcoin network returns it to the user. No action by the issuer is required.
- The deposit, the issuer's acceptance of it, and the mint are each recorded on a public blockchain. The backing of every minted unit can be verified by anyone.
- The issuer can examine the origin of the Bitcoin before accepting it. A deposit that does not pass the issuer's screening is never taken into custody. It returns to the user when the time lock expires.

1.3 The issuer's token

The token belongs to the issuer. Its name, symbol and governance are the issuer's decisions. Maxy Mint requires only that the token permits its mint contract to mint. An issuer may keep other minting methods in parallel, for example minting against Bitcoin received through its existing custody operations, and may operate its own redemption process instead of, or alongside, the one described here. An exchange that already issues a wrapped-Bitcoin token can therefore add Maxy Mint as an additional, verifiable minting path without replacing its current arrangements. Chapter 10 describes the integration options. Throughout this document the issuer's token is referred to as the token.

1.4 Scope and audience

This document describes the system on Robinhood Chain (chain ID 4663), with deposits settled on the Bitcoin mainnet: direct-custody mode as deployed, and HTLC mode as specified for deployment. It is written for readers who are evaluating or building on the system:

- Issuers and their compliance, treasury and operations teams: Chapters 2, 6, 7 and 12.
- Developers integrating wallets or front ends: Chapter 17 for the live mode, and Chapters 5, 8, 11 and 15 for HTLC mode.
- Reviewers of the contracts: Chapters 13, 14, 16 and 17.

Chapter 3 defines the terms used throughout. It does not assume prior familiarity with Bitcoin scripting or with token issuance.

2. Design overview

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

This chapter describes HTLC mode, which is planned. Section 2.4 sets out how direct-custody mode, live today, differs.

2.1 Three steps

A Maxy Mint deposit proceeds in three steps.

1. **Lock.** The user sends Bitcoin to an address generated for that one deposit. The address is a hash time-locked contract, or HTLC. Bitcoin held at the address can be spent in exactly two ways. The issuer may spend it by presenting a secret value chosen for this deposit. The user may spend it after a fixed time has passed. No other spending is possible.
2. **Prove.** A proof that the Bitcoin arrived at the address is submitted to the mint contract on Robinhood Chain. The mint contract checks the proof against a copy of the Bitcoin block headers that it maintains. After this step the contract has established, without relying on any party's statement, that the address holds a specific number of satoshis.
3. **Reveal.** The issuer spends the HTLC using its secret. Doing so places the secret on the Bitcoin blockchain, where anyone can read it. The issuer then submits the secret to the mint contract. The contract verifies that the secret matches the one fixed at registration and mints the token to the user.

The secret is the link between the two chains. Spending the HTLC requires publishing it. Minting requires presenting it. The issuer therefore cannot complete the first act without enabling the second.

2.2 Properties

Property	How it is achieved
The user cannot lose the Bitcoin without receiving the token	The HTLC can only be claimed by publishing the secret that unlocks the mint. If it is not claimed, the time lock returns the funds.
The issuer never holds Bitcoin it has not accepted	Custody begins only with the claim, which is a deliberate act taken after the deposit has been examined.

Property	How it is achieved
Every minted unit is traceable to a Bitcoin transaction	The funding proof records the Bitcoin transaction and output. The mint event records the amount. The claim transaction is visible on the Bitcoin blockchain.
Minting through this path cannot exceed deposits	The contract mints exactly the amount proven on chain, once per deposit.
A declined deposit is returned without the issuer's involvement	Declining to claim is sufficient. The Bitcoin network enforces the refund after the lock time.
Operation is transparent	Every change of state is a public transaction on Bitcoin or on Robinhood Chain.

2.3 Comparison with custodial transfer

	Custodial transfer	Maxy Mint
Location of the Bitcoin while the mint is pending	The issuer's custody	An HTLC that neither party controls alone
What the user relies on	The issuer's internal process	The Bitcoin network and the mint contract
What a third party can verify	The issuer's published statements	The deposit, the acceptance and the mint, on public chains
Outcome if the issuer declines	A manual refund by the issuer	An automatic return after the lock time
Timing of compliance screening	After custody	Before custody
Record of the mint	The issuer's ledger	A contract event tied to a specific Bitcoin transaction

2.4 Direct-custody mode

In direct-custody mode, live today, the three steps become order, pay and mint. The user signs an order, registers the issuer's signed assignment on Robinhood Chain, and pays the exact amount to a deposit address held by the issuer. The issuer's service then proves the payment to the mint contract, which mints the token. There is no secret and no time lock: the Bitcoin is in the issuer's custody as soon as the payment confirms, and a payment that cannot be minted is refunded by the issuer on the user's signed request. Of the properties in 2.2, those that rest on the HTLC, in particular the automatic

return of a declined deposit, apply only in HTLC mode. Chapter 17 describes direct-custody mode in full.

3. Key concepts

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

The terms below are used throughout this document. Each is defined once here.

3.1 Bitcoin terms

Transactions, outputs and addresses. A Bitcoin transaction moves value from one or more inputs to one or more outputs. Each output holds an amount and a rule that determines who may spend it. An address is a readable encoding of that rule. An output that has not yet been spent is identified by the transaction identifier (txid) and the output's position within the transaction. This pair is called an outpoint.

Satoshi. The smallest unit of Bitcoin. One bitcoin is 100,000,000 satoshis, abbreviated sats.

Confirmations. A transaction is confirmed when it is included in a block. Each later block adds one confirmation. Six confirmations are widely treated as final. The system records a required number of confirmations for each deposit.

Block header. An 80-byte summary of a block that commits to every transaction in it. A system that holds the chain of headers can verify that a transaction belongs to a block without holding the block itself.

Hash and preimage. A hash function converts any input into a fixed-size value called a hash. The input is called the preimage. Maxy Mint uses SHA-256. Computing the hash from the preimage is immediate. Recovering the preimage from the hash is infeasible. In this system the issuer chooses a 32-byte secret preimage `S` and publishes its hash `H`. `H` is public from the moment an order is signed. `S` remains with the issuer until the issuer claims the deposit.

Hash lock and time lock. A hash lock is a spending rule that is satisfied by presenting the preimage of a stated hash. A time lock is a spending rule that is satisfied only after a stated time.

HTLC. A hash time-locked contract is a Bitcoin script that combines the two rules. It allows the issuer to spend by presenting `S` and signing with the issuer's Bitcoin key, and it allows the user to spend by signing with the user's Bitcoin key once the lock time has passed. The Bitcoin network enforces the script. The HTLC is not a contract on Robinhood Chain; the mint contract only verifies proofs about it. Chapter 15 gives the script in full.

Lock time. The Unix timestamp written into the HTLC, called `btcLockTime` in the contracts. Before it, only the issuer's path is available. After it, both paths are available, and the first transaction to confirm

takes the funds.

Proof of inclusion. A compact proof that a transaction is contained in a specific Bitcoin block. It consists of the raw transaction, the header of the block, and the Merkle path that connects them. The verifier contract on Robinhood Chain maintains a copy of the Bitcoin headers and checks proofs against it. This is how Robinhood Chain establishes that a Bitcoin payment occurred.

3.2 Token and contract terms

The token. The issuer's token on Robinhood Chain. In the reference implementation it is an ERC-20 token with 18 decimal places. One satoshi corresponds to 10,000,000,000 token units; this factor is the contract constant `SATS_TO_B18`. A deposit of 100,000 sats therefore mints 10^15 units, displayed as 0.001.

Mint contract. The Robinhood Chain contract that registers deposits, verifies funding proofs, checks the secret and mints the token. It is called `ProofMint` in the source.

Assignment. The signed order for one deposit. It is a message signed by the issuer's `operator` key that fixes the depositor's address, the hash `H`, the two Bitcoin public keys, the lock time, the expected amount, the required confirmations, and two deadlines: `validUntil`, by which the user must register, and `expiry`, which starts the clock for closing the record. The user submits the Assignment and the signature to the mint contract to register the deposit.

Deposit and deposit identifier. The record the mint contract creates at registration. Its identifier, `depositId`, is the hash of the HTLC address, so there is exactly one record per address. A deposit passes through the states described in Chapter 8.

Custody and reserve. Custody means control of Bitcoin by the issuer's keys. The reserve is the Bitcoin the issuer holds to back the token in circulation. Bitcoin enters the reserve when the issuer claims an HTLC. It leaves the reserve when a redemption is paid.

3.3 Roles

Issuer. The organization that operates the system and owns the token. The issuer signs orders, claims deposits, submits proofs and secrets to the mint contract, holds the reserve and pays redemptions.

User, or depositor. The person or entity that deposits Bitcoin. In the contracts the depositor is the Robinhood Chain address named in the Assignment. Only that address may register the deposit or change its recipient.

Recipient. The Robinhood Chain address that receives the token. The depositor chooses it and may change it until the mint occurs.

Holder. Any owner of the token. A holder may redeem the token for Bitcoin.

4. System components

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

The system spans three environments. The Bitcoin network holds the user's deposit in an HTLC. Robinhood Chain holds the deposit records, the verifier, the token and the redemption contract. The issuer's off-chain service connects the two: it signs orders, claims deposits, submits proofs and secrets, and pays redemptions. The user's wallet or front end interacts with all three.

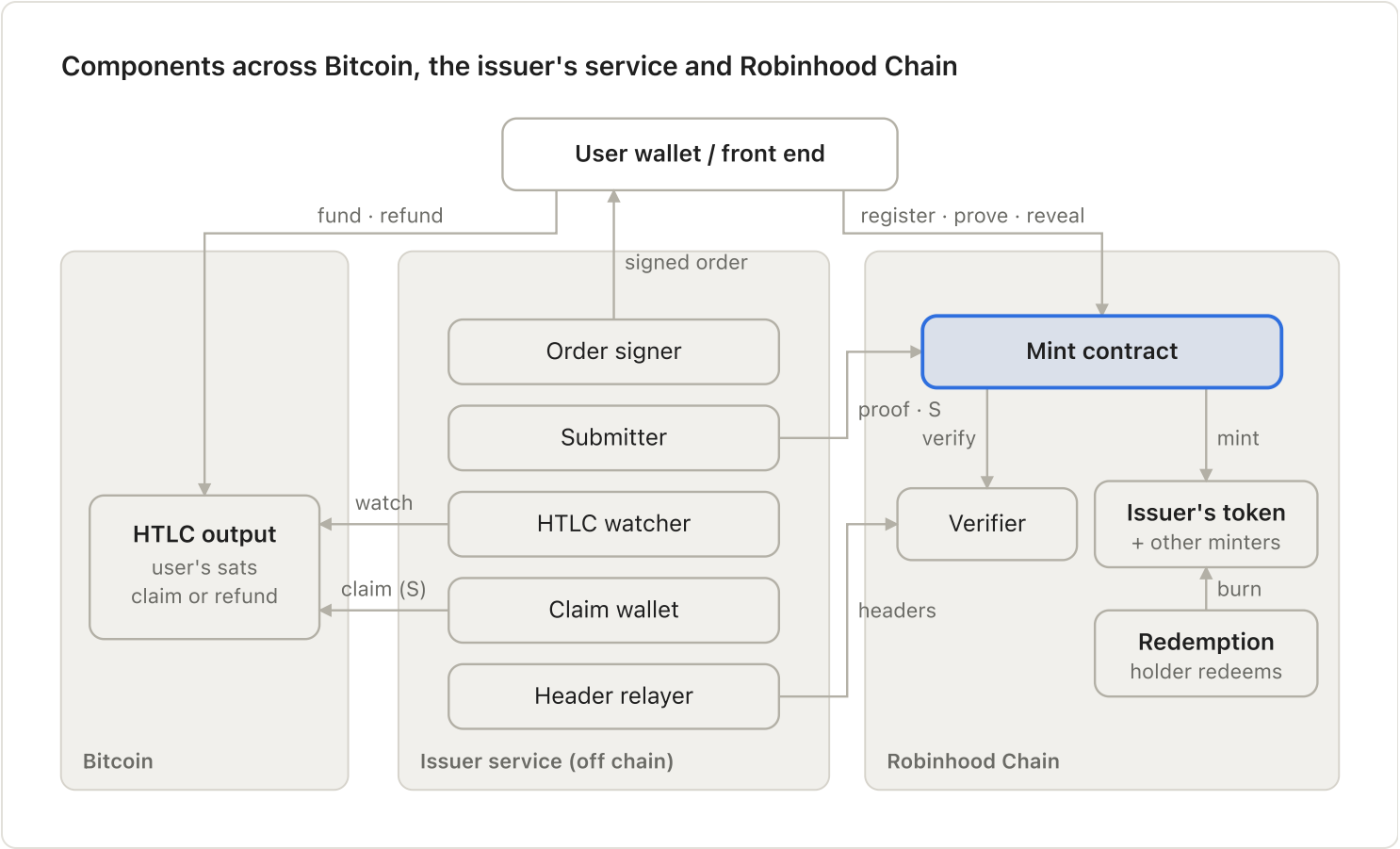


Figure: System components across Bitcoin, the issuer's service and Robinhood Chain.

The user funds and refunds on Bitcoin, and registers, proves, reveals and redeems on Robinhood Chain. The issuer signs orders, claims the HTLC, and supplies the mint contract with proofs, secrets and Bitcoin headers.

4.1 Components

Component	Where it runs	Function
HTLC output	Bitcoin	Holds the user's deposit. Spendable by the issuer with the secret, or by the user after the lock time.
User wallet or front end	User's device	Requests orders, registers deposits, constructs the funding and refund transactions, submits proofs where required, and redeems.
Order signer	Issuer, off chain	Generates the secret, assembles the HTLC terms and signs the Assignment with the `operator` key. Also signs redemption quotes with the `quoteSigner` key.
HTLC watcher	Issuer, off chain	Monitors each HTLC address for the funding transaction, its confirmations and its amount. Supplies the inputs for compliance screening.
Claim wallet	Issuer, off chain	Holds the Bitcoin key named in the HTLC. Spends accepted deposits through the claim path, which publishes the secret.
Submitter	Issuer, off chain	Sends `submitFundingProof` and `revealPreimage` transactions to the mint contract.
Header relayer	Issuer, off chain	Supplies Bitcoin block headers to the verifier so that proofs can be checked. Any party may operate one.
Payout wallet	Issuer, off chain	Pays Bitcoin to holders whose redemptions have been recorded on chain.
Mint contract	Robinhood Chain	Registers deposits, verifies funding proofs, checks the secret and mints the token.
Verifier	Robinhood Chain	Stores Bitcoin block headers and verifies that a given transaction output is contained in a given block.
Token	Robinhood Chain	The issuer's token. It grants the mint contract permission to mint and may grant the same permission to the issuer's other minting systems.
Redemption contract	Robinhood Chain	Burns the token against a signed payout quote and records the payout obligation. Optional; see Chapter 10.

4.2 Enforcement layers

Three distinct mechanisms protect the parties. The remainder of this document refers to them separately.

- **Bitcoin consensus** enforces the HTLC. The issuer cannot spend the deposit without the secret, and the user cannot spend it before the lock time.
- **The mint contract** enforces the mint. No token is created without a verified funding proof and a preimage that matches the registered hash.
- **Issuer policy** governs what no chain can enforce: claiming only deposits that have been verified on chain and screened for compliance, claiming with a safety margin before the lock time, revealing the secret only after the claim is confirmed, and paying redemptions. A breach of the first three costs the issuer rather than the user. Chapter 13 sets out the consequences.

5. The deposit and minting process

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

This chapter follows one deposit in HTLC mode from the order to the minted token. A direct-custody deposit, the mode live today, follows the steps in 17.2. The diagram shows the messages exchanged between the user, the Bitcoin network, the issuer and the mint contract. Steps 1 to 7 are the normal path. Step 8 is the alternative that applies when the issuer does not claim.

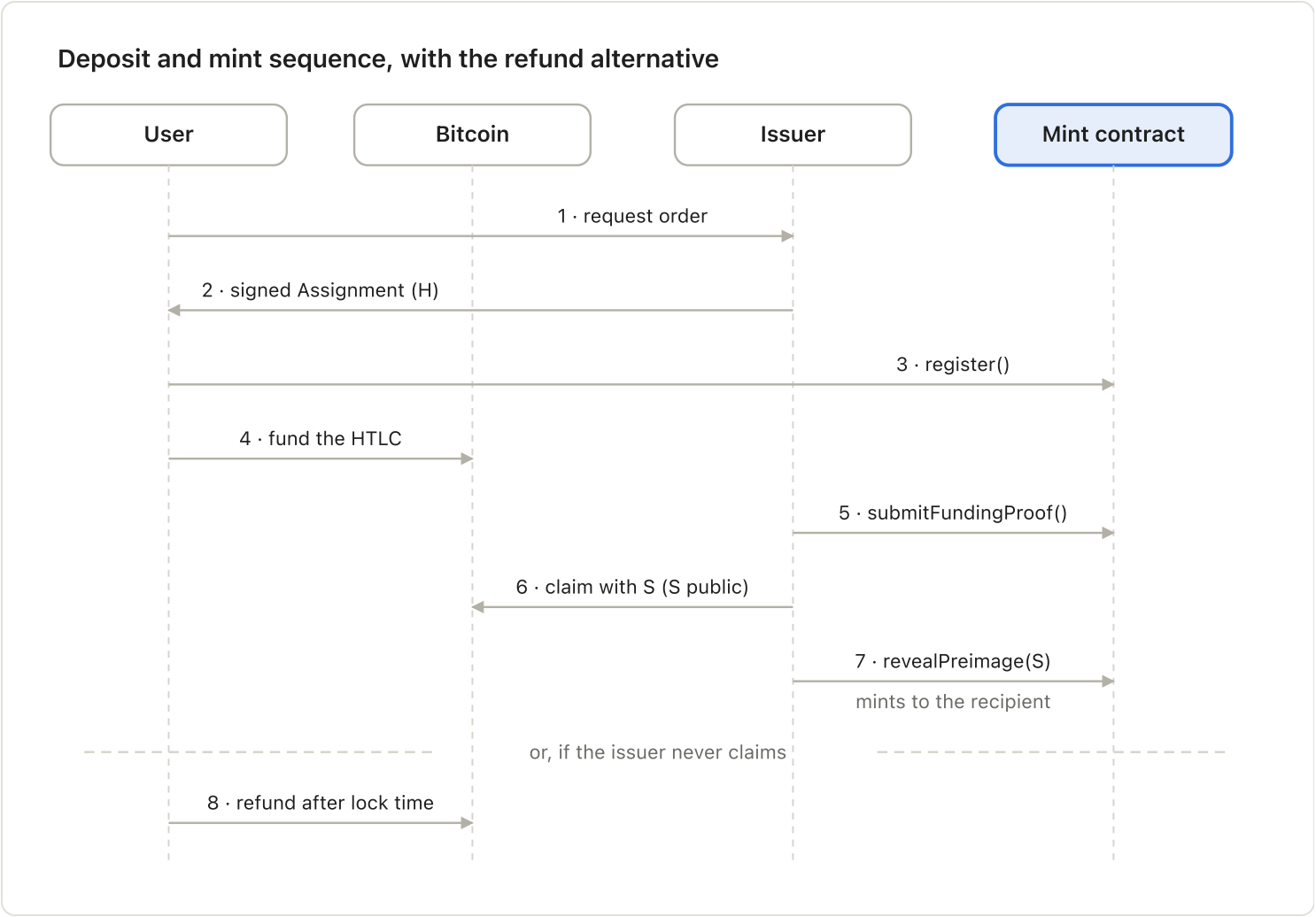


Figure: Deposit and mint sequence, with the refund alternative.

5.1 Steps

1. **The user requests an order.** The user's wallet or front end sends the issuer the user's Robinhood Chain address, a Bitcoin public key that the user controls (the refund key), the amount to deposit

and the address that should receive the token. If the issuer requires identity verification, it is completed at this point.

2. **The issuer signs the Assignment.** The issuer generates a new secret `S`, computes `H`, sets the lock time (typically 24 hours ahead) and signs the Assignment with its `operator` key. The user receives the Assignment, the signature and the HTLC script.
3. **The user registers the deposit.** The user submits the Assignment and the signature to the mint contract. The contract verifies the signature, constructs the HTLC script from the signed terms, derives the Bitcoin address, and records the deposit as `REGISTERED`. The address is published in the `DepositRegistered` event. Because the contract derives the address from the signed terms, the user can confirm that the address is the agreed HTLC before sending any Bitcoin.
4. **The user funds the HTLC.** The user sends the agreed amount in a single payment to the address and waits for the required number of confirmations.
5. **The funding proof is submitted.** The issuer's service, or the user, submits a proof of inclusion for the funding transaction together with the amount it carries. The verifier confirms that the transaction is in a confirmed Bitcoin block and that the output pays exactly that amount to the registered address. The deposit becomes `FUNDED` and the amount is recorded as `fundedSats`. No token has been minted.
6. **The issuer claims the deposit.** Having confirmed that the deposit is `FUNDED`, that it has passed compliance screening, and that sufficient time remains before the lock time, the issuer spends the HTLC through the claim path. The claim transaction carries `S` in its witness data. From this point `S` is public.
7. **The secret unlocks the mint.** After the claim transaction has confirmed, the issuer submits `S` to the mint contract. The contract verifies that `SHA256(S)` equals the registered hash, mints `fundedSats` worth of the token to the recipient, and records the deposit as `ISSUED`.

If the issuer's service fails after step 6, the user can read `S` from the claim transaction on the Bitcoin blockchain and submit it to the mint contract. The token is minted to the registered recipient regardless of who submits the secret.

5.2 Worked example

A user deposits 100,000 sats (0.001 BTC). The issuer signs an Assignment with `amountSats = 100000`, `minConfirmations = 3` and a lock time 24 hours ahead. The user registers the deposit and sends 100,000 sats to the derived address. After three confirmations the issuer submits the funding proof; the deposit becomes `FUNDED` with `fundedSats = 100000`. The issuer screens the funding transaction, claims the HTLC, and after the claim confirms submits `S`. The contract mints 10^15 token units, displayed as 0.001, to the recipient address.

If the user had sent 80,000 sats instead, the funding proof would have to be submitted by the user, since only the depositor may prove an amount that differs from the order. The deposit would be

recorded with `fundedSats = 80000`, and the issuer would decide whether to claim at that amount. If it claimed, 0.0008 of the token would be minted.

5.3 The refund case

If the issuer does not claim, the user waits for the lock time to pass, spends the HTLC through the refund path, and recovers the full deposit less the Bitcoin network fee. Nothing is required on Robinhood Chain. The secret was never published, so no token can ever be minted for that deposit. The deposit record remains `FUNDED` until it is closed, and closing it changes nothing about the funds.

The two outcomes are mutually exclusive. A claim publishes the secret and leads to a mint. A refund never publishes the secret and leads to a closed record. There is no sequence of events in which the user holds neither the Bitcoin nor the token.

6. Flow of funds and custody

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

This chapter states where the Bitcoin is at each stage of a deposit, who can move it, and what the token side records. Readers responsible for treasury, audit or compliance should find here every transfer of value the system performs.

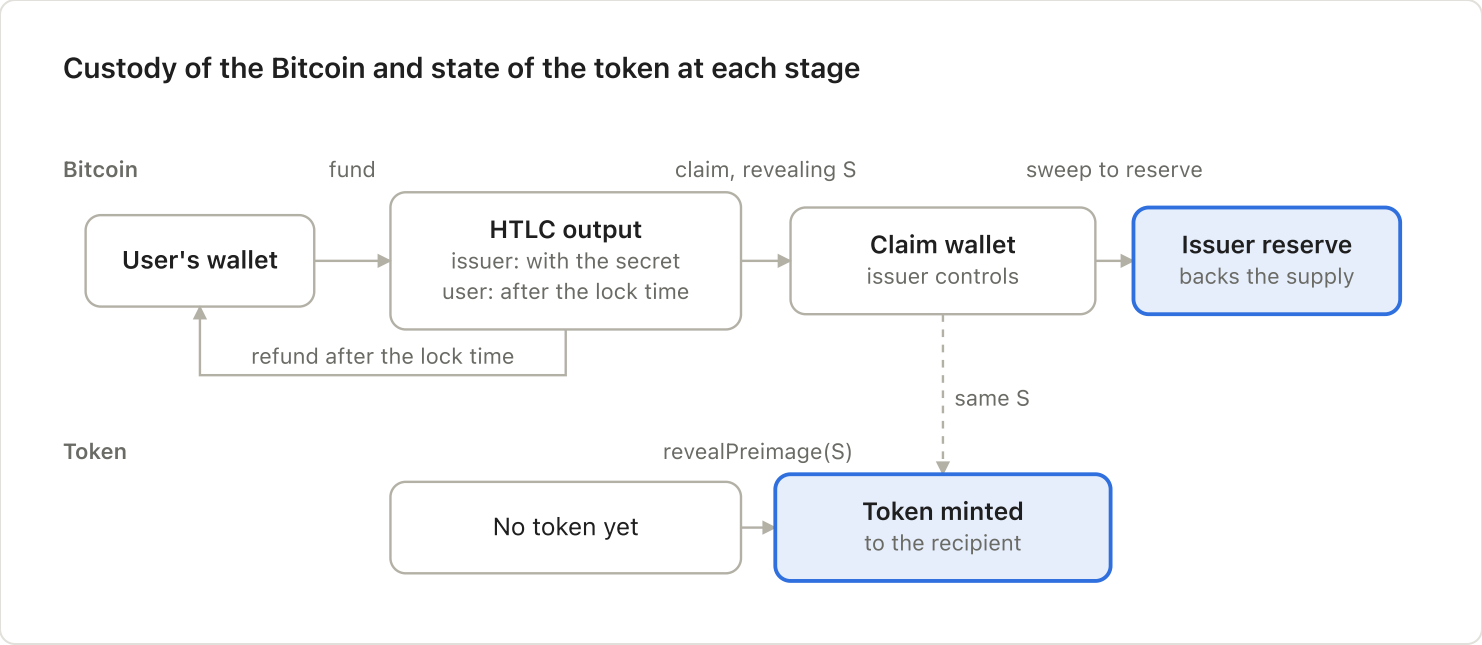


Figure: Custody of the Bitcoin and state of the token at each stage.

The Bitcoin moves from the user's wallet into the HTLC, then into the issuer's claim wallet, then into the issuer's reserve. The token is minted only after the claim, and only with the same secret that the claim published. If no claim occurs, the Bitcoin returns from the HTLC to the user's wallet.

6.1 Stages

Stage	Location of the Bitcoin	Who can move it	Token	Record on chain
Order signed	User's wallet	User	None	None; the signed Assignment is held by the user and the issuer
Deposit registered	User's wallet	User	None	`DepositRegistered` on Robinhood Chain, including the HTLC address and terms

Stage	Location of the Bitcoin	Who can move it	Token	Record on chain
HTLC funded	HTLC output on Bitcoin	Issuer, by publishing the secret; user, after the lock time	None	The funding transaction on Bitcoin
Funding proven	HTLC output on Bitcoin	As above	None	`DepositFunded` on Robinhood Chain, naming the funding transaction, output index and amount
HTLC claimed	Issuer's claim wallet	Issuer	None, but mintable by anyone who reads the secret	The claim transaction on Bitcoin, which contains the secret
Token minted	Issuer's claim wallet or reserve	Issuer	Minted to the recipient	`PreimageRevealed` and `ProofMinted` on Robinhood Chain
Swept to reserve	Issuer's reserve custody	Issuer	In circulation	The sweep transaction on Bitcoin
Refund (alternative)	User's wallet	User	None, and none can be minted	The refund transaction on Bitcoin

Redemption, which moves Bitcoin from the reserve back to a holder, is described in Chapter 9.

6.2 Encumbrance while in the HTLC

While the deposit sits in the HTLC, neither party has ordinary control of it. The issuer can move it only by publishing the secret, and publishing the secret entitles the user to the token. The user can move it only after the lock time, and only if the issuer has not already claimed. Neither party can direct the funds to any third party except through one of those two paths. The HTLC is therefore not an escrow held by an intermediary. It is a rule enforced by the Bitcoin network with no custodian at all.

For accounting purposes the deposit remains the user's property until the issuer claims it. The issuer acquires no right to the funds by their arrival at the address. It acquires the funds only by the act of claiming, which is also the act that commits it to the mint.

6.3 Custody after the claim

When the claim transaction confirms, the Bitcoin is under the sole control of the issuer's claim key and is no longer encumbered by any script. The claim wallet is an operational wallet. The issuer sweeps claimed outputs from it into reserve custody, which may be cold storage operated by the issuer or an account with a qualified custodian. From that point the Bitcoin is reserve backing the issuer's token.

The chain of custody is public at every step. The funding outpoint is recorded on Robinhood Chain in the `DepositFunded` and `ProofMinted` events. The Bitcoin transaction that spends that outpoint is the claim. The outputs of the claim transaction lead to the sweep, and the sweep leads to the reserve addresses. An issuer that publishes its reserve addresses enables any third party to follow every deposit from the user's wallet to the reserve.

6.4 Reserve accounting

The token supply created through Maxy Mint equals the sum of `fundedSats` over all deposits in the `ISSUED` state, multiplied by `SATS_TO_B18`. The supply retired through the redemption contract equals the sum of `grossSats` over all redemptions. Both sums can be computed by anyone from contract events.

On the Bitcoin side, reserve inflow equals the sum of the claimed outputs, and reserve outflow equals the sum of redemption payouts plus network fees. An issuer that also mints through other paths accounts for those separately; the Maxy Mint portion of the supply and of the reserve is reconstructible from chain data alone.

6.5 Fees

Cost	Paid by
Bitcoin network fee for the funding transaction	User
Bitcoin network fee for the refund transaction	User, only in the refund case
Robinhood Chain gas for `register`	User
Robinhood Chain gas for the funding proof	Whoever submits it; normally the issuer
Bitcoin network fee for the claim and the sweep	Issuer
Robinhood Chain gas for `revealPreimage`	Whoever submits it; normally the issuer
Redemption fee (`feeSats`)	Deducted from the holder's payout
Bitcoin network fee for the redemption payout	Issuer, from the fee

7. Compliance and auditability

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

The design gives the issuer a decision point between receiving a deposit and taking custody of it, and it records every step of the process on a public blockchain. This chapter describes how an issuer uses those properties to meet its compliance obligations and how any third party can audit the result.

7.1 The screening window

A deposit arrives in the HTLC before the issuer has any control over it. The funding transaction is visible on the Bitcoin blockchain with its inputs, so the addresses and prior transactions from which the funds came can be examined. The issuer's service has the entire interval between the funding and the lock time, typically close to a day, in which to screen the deposit before deciding whether to claim it.

Screening can include any check the issuer's program requires: sanctions list matching on the source addresses, risk scoring of the funds' history, reconciliation against the identity established when the order was requested, and transaction-size or velocity limits. The user's identity is tied to the deposit through the Assignment, which names the user's Robinhood Chain address and is issued only after whatever onboarding the issuer requires.

The secret is not published until the issuer claims. Until then the issuer has made no commitment and has received nothing.

7.2 Declining a deposit

To decline a deposit, the issuer does nothing. It does not claim the HTLC and does not reveal the secret. When the lock time passes, the user recovers the Bitcoin through the refund path. The result is the following.

- The issuer never takes possession of the declined funds. There is nothing to freeze, segregate or return.
- The user is not dependent on the issuer for the return of the funds. The Bitcoin network enforces it.
- The on-chain record shows a deposit that was registered and funded but never minted. This is the evidence that the funds were examined and not accepted.

Compared with a custodial process, in which a declined deposit must be returned by the issuer from its own wallets, this removes both the operational burden and the period during which the issuer holds

funds it does not wish to hold.

7.3 The audit trail

Each step of a deposit leaves a record that can be read without the issuer's cooperation.

Record	Where	What it establishes
Signed Assignment	Held by the user and the issuer; verifiable against the `operator` address	The issuer authorized a deposit on these terms for this user
`DepositRegistered` event	Robinhood Chain	The user accepted the terms; the HTLC address, hash lock and lock time
Funding transaction	Bitcoin	The amount deposited and the inputs it came from
`DepositFunded` event	Robinhood Chain	The funding transaction, output index and amount were verified against the Bitcoin headers
Claim transaction	Bitcoin	The issuer took custody; the secret is in the witness
`PreimageRevealed` and `ProofMinted` events	Robinhood Chain	The secret matched; the amount and the recipient of the mint
Sweep transaction	Bitcoin	The claimed funds entered the reserve
`RedemptionRequested` event	Robinhood Chain	A holder burned a stated amount for payout to a stated destination, with the fee
Payout transaction	Bitcoin	The redemption was settled

7.4 Verification by a third party

An auditor, a regulator or a holder can verify the backing of the token without access to the issuer's systems.

1. Read every `ProofMinted` event from the mint contract. Each names a deposit, a Bitcoin transaction, an output index and an amount.

2. For each event, locate the Bitcoin transaction and confirm that the named output paid the amount to the registered HTLC address.
3. Confirm that the output was spent by a claim transaction, and that the claim's witness contains a value whose SHA-256 equals the deposit's hash lock. This establishes that the issuer, not the user, took the funds.
4. Follow the claim outputs to the issuer's reserve addresses.
5. Sum the amounts over all `ProofMinted` events, subtract the amounts over all `RedemptionRequested` events, and compare the result with the token supply attributable to this path.

Steps 1 to 3 establish that every unit minted through Maxy Mint was backed by a specific deposit at the time it was minted. Steps 4 and 5 establish whether the reserve has been maintained since. The first three steps need no information from the issuer. The last two need the issuer's reserve addresses, which an issuer committed to transparency publishes.

7.5 Controls

The following controls are inherent in the system or are standard operating practice for an issuer using it.

Control	Provided by
Customer identification before a deposit is authorized	The order process; no Assignment is signed for an unidentified user
Screening of the source of funds before custody	The screening window between funding and claim
No custody of declined funds	The refund path, enforced by Bitcoin
Minting limited to verified deposits	The mint contract; no token is minted without a proof and a matching secret
No unilateral minting through this path	The mint contract has no function that mints without a deposit
Segregation of duties among keys	Separate keys for signing orders, claiming deposits, holding secrets, paying gas, signing quotes and paying redemptions (Chapter 12)
Limited administrative powers	The pause affects only new registrations; funded deposits always proceed; the token's mint and burn authorities are fixed
Complete record of minting and redemption	Contract events on Robinhood Chain and transactions on Bitcoin

Control	Provided by
Verifiable reserves	The chain of custody from each deposit to the reserve

Taken together, these controls provide the safeguards expected of a regulated issuer: identification of the customer, screening of the source of funds before acceptance, separation of duties, complete and tamper-evident records, and reserves that can be verified independently. They do so while keeping both minting and redemption entirely on chain. Legal requirements differ between jurisdictions, and the system supplies the technical controls on which an issuer's compliance program is built.

8. Deposit lifecycle

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

A deposit record is created by `register` and passes through at most three further states. The state is stored in the record and can be read with `getDeposit(depositId)`.

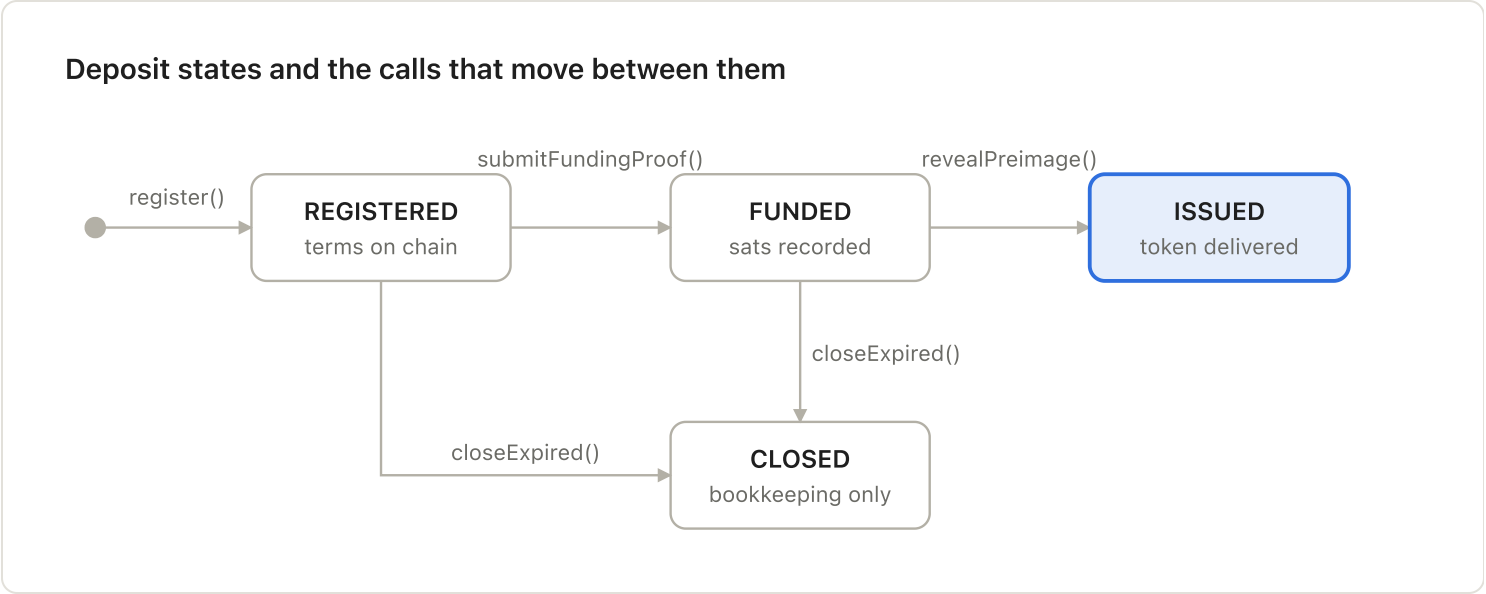


Figure: Deposit states and the calls that move between them.

`ISSUED` is reached only through `revealPreimage`, and therefore only with the secret. `CLOSED` can be reached from either open state, but only after the expiry and, for most of a long period, only by the depositor.

8.1 States

State	Code	Meaning
`MISSING`	0	No record exists for this identifier.
`REGISTERED`	1	The terms are on chain and the HTLC address is known. No funding proof has been accepted.
`FUNDED`	4	A funding proof has been accepted and `fundedSats` is recorded. The secret has not been revealed.
`ISSUED`	2	The secret has been revealed and the token has been minted. Final.

State	Code	Meaning
`CLOSED`	3	Closed without minting. Final. Closing affects records only; it moves no funds.

8.2 Transitions

From	To	Call	Who may call	Conditions
`MISSING`	`REGISTERED`	`register`	The depositor named in the Assignment	Valid operator signature; `validUntil` not passed; HTLC terms well formed; lock time and expiry within the limits in 8.3; hash lock and address unused; contract not paused
`REGISTERED`	`FUNDED`	`submitFundingProof`	Anyone if the proven amount equals `amountSats`; otherwise only the depositor	Proof verifies against the registered address with at least `minConfirmations`; amount at or above the minimum; output not previously used
`FUNDED`	`ISSUED`	`revealPreimage`	Anyone	`SHA256(preimage)` equals the registered hash lock
`REGISTERED` or `FUNDED`	`CLOSED`	`closeExpired`	The depositor after `expiry`; anyone after `expiry + LATE_CLOSE`	None beyond the timing

While a deposit is `REGISTERED` or `FUNDED`, the depositor may change the recipient with `setDelivery` and may replace the expected amount and the expiry with `amendAmount`, using a newly signed Assignment. Neither call changes the state.

8.3 Time parameters

Parameter	Set by	Meaning	Rule or default
`btcLockTime`	Issuer, in the Assignment	Time after which the user may refund on Bitcoin	At least `MIN_LOCK_LEAD` (6 hours) after registration; 24 hours is typical
`validUntil`	Issuer, in the Assignment	Last moment at which `register` accepts the Assignment	No later than `expiry`
`expiry`	Issuer, in the Assignment	Start of the closing period	At least `btcLockTime + PREIMAGE_GRACE` (24 hours)
`LATE_CLOSE`	Contract constant	Period after `expiry` during which only the depositor may close	90 days
`minConfirmations`	Issuer, in the Assignment	Confirmations the funding proof must show	1 to 6, and never below the verifier's own floor; 3 is typical

Bitcoin compares a lock time with the median timestamp of the preceding eleven blocks rather than with the current clock. That median trails real time by about an hour. A refund therefore becomes spendable roughly an hour after `btcLockTime`, and the issuer's claim margin in Chapter 12 allows for this.

9. Redemption

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

A holder converts the token back into Bitcoin through the redemption contract. The token is burned on Robinhood Chain first, and the burn is permanent. The Bitcoin payout follows from the issuer's reserve. An issuer may instead, or in addition, operate its own redemption process; see Chapter 10.

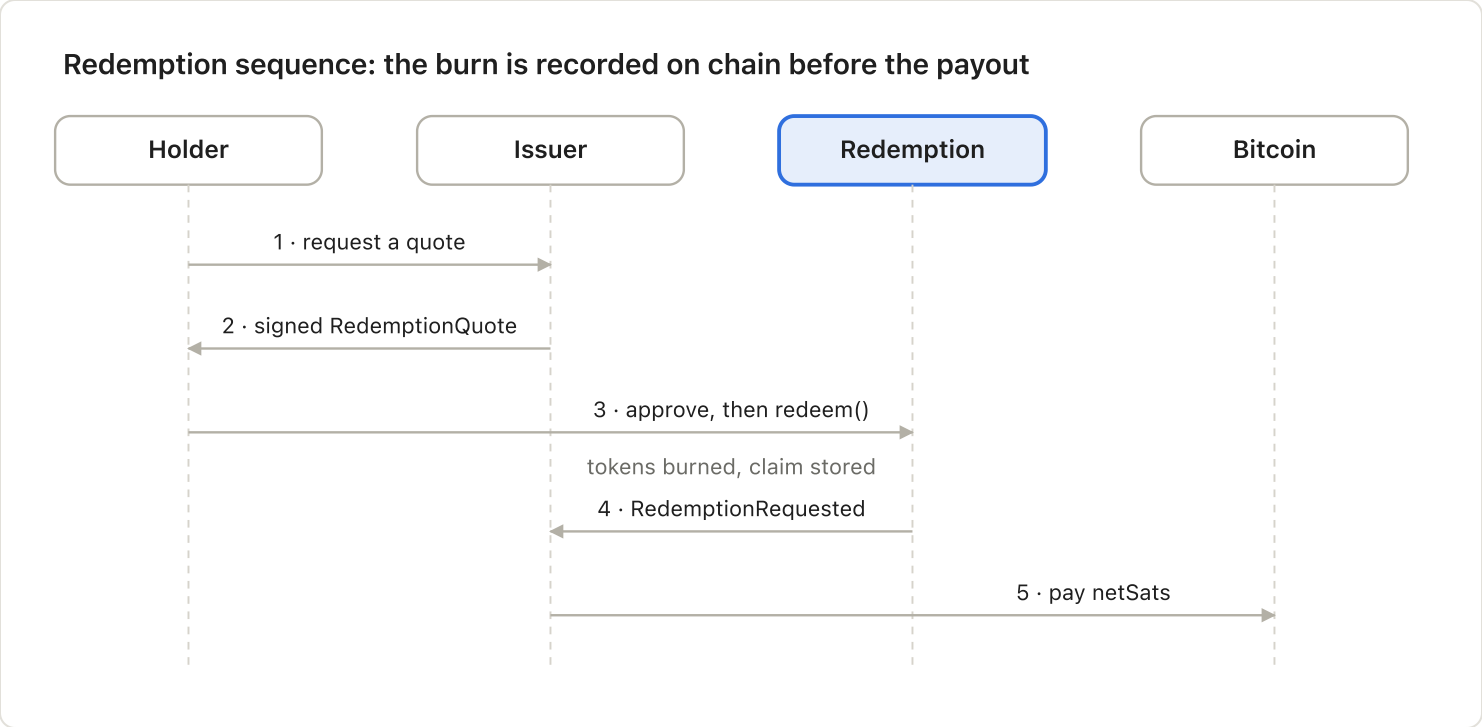


Figure: Redemption sequence: the burn is recorded on chain before the payout.

Steps 1 to 3 are performed by the holder. Steps 4 and 5 are performed by the issuer. Once step 3 has confirmed, the holder has an on-chain record of the issuer's obligation.

9.1 Steps

1. **The holder requests a quote.** The holder gives the issuer the Bitcoin destination and the amount to redeem.
2. **The issuer signs the quote.** The issuer's `quoteSigner` key signs a `RedemptionQuote` that names the holder, the destination, the gross amount to be burned, the fee retained by the issuer and the net amount to be paid, together with a deadline.
3. **The holder approves and redeems.** The holder authorizes the redemption contract to spend the gross amount of the token and calls `redeem(quote, signature)` from the address named in the

quote, before the deadline. The contract verifies the quote and the signature, burns the token from the holder, and stores the quote under its identifier.

4. **The contract records the obligation.** It emits `RedemptionRequested`, which the issuer's service monitors.
5. **The issuer pays.** The issuer's payout wallet sends the net amount to the destination on Bitcoin and reconciles the payment against the stored quote. There is no cancellation, re-minting or expiry after the burn.

9.2 Quote fields

Field	Meaning
`quoteId`	Unique, non-zero identifier; accepted once
`holder`	The Robinhood Chain address that must call `redeem`
`token`	The token's address
`bitcoinNetwork`	`keccak256("bitcoin-mainnet")`
`destinationScript`	The Bitcoin output script to be paid: P2WPKH, P2TR, P2PKH or P2SH
`grossSats`, `feeSats`, `netSats`	Amount burned, fee retained, amount paid; `netSats + feeSats` must equal `grossSats`, and `feeSats` must be positive
`validUntil`	Deadline for `redeem`

9.3 Minimum payouts

The net amount must exceed the Bitcoin network's dust threshold for the destination type: 294 sats for P2WPKH, 330 sats for P2TR, and 546 sats for P2PKH and P2SH. Other destination types are rejected. The stored quote for any redemption can be read with `getClaim(quoteId)` by anyone auditing a payout.

10. Integrating with the issuer's token

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

Maxy Mint is a minting path, not a token. It can create the token it mints, or it can be attached to a token the issuer already operates. This chapter sets out what the token must provide, the ways it can be connected, and how an issuer accounts for supply when more than one minting path exists.

10.1 Requirements on the token

The mint contract makes one call on the token: `mint(address recipient, uint256 amount)`. The token must allow the mint contract to make that call. Nothing else is required of the token for minting.

If the issuer uses the Maxy Mint redemption contract, the token must also allow that contract to burn from a holder who has approved it: `burnFrom(address holder, uint256 amount)`. If the issuer operates its own redemption process, this is not required.

The reference contracts assume 18 decimal places and convert satoshis to token units with the constant `SATS_TO_B18`. A token with a different number of decimals requires the constant to be set accordingly at deployment.

10.2 Deployment options

Option 1: a new token created with the system. The mint contract's constructor deploys a token and a redemption contract, and fixes itself as the token's only minter and the redemption contract as its only burner. These authorities cannot be changed afterwards. This is the most restrictive arrangement and the simplest to audit: the entire supply is attributable to Maxy Mint deposits.

Option 2: an existing token grants minting permission. The issuer deploys the mint contract against its existing token and grants the mint contract permission to mint. Tokens built on a role-based permission model, such as a minter role, support this directly. The issuer's other minting methods continue to operate. The issuer may grant the Maxy Mint redemption contract permission to burn, or keep its own redemption process.

Option 3: a new token with open permissions. The issuer deploys a token whose minter and burner permissions are managed by the issuer's governance, grants them to the Maxy Mint contracts, and retains the ability to add other minting paths later.

Under options 2 and 3 the issuer's governance can revoke the mint contract's permission. Doing so stops new mints through this path. It does not affect deposits already funded, which the issuer should

not claim once minting is no longer possible, and it does not affect tokens already in circulation.

10.3 Coexistence with other minting methods

An issuer that mints through several paths maintains a single token supply backed by a single reserve. The Maxy Mint path contributes a portion of the supply that is reconstructible from chain data, as described in Chapter 6. The other paths contribute according to the issuer's own records.

For reserve verification, the issuer should publish which minting paths exist, which contracts hold minting permission, and the reserve addresses associated with each. A third party can then verify the Maxy Mint portion independently and treat the remainder according to the evidence the issuer provides for it. Over time an issuer may choose to route a growing share of issuance through Maxy Mint precisely because that share requires no trust in its records.

10.4 Upgrades and replacement

The contracts are not upgradeable. A new version of the mint contract is a new deployment. Under option 1 this requires a new token. Under options 2 and 3 the issuer grants minting permission to the new contract and revokes it from the old one, and the token is unaffected. Issuers who expect to upgrade should prefer options 2 or 3.

Before revoking permission from an old mint contract, the issuer should allow every `FUNDED` deposit on it to reach `ISSUED` or be refunded. A revealed secret on a contract that can no longer mint would leave the user with neither the Bitcoin nor the token, which the design otherwise prevents.

11. Integration guide for wallets and front ends

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

This chapter describes what a wallet or front end must do to take a user through a deposit, a refund and a redemption. It assumes access to a Robinhood Chain RPC endpoint and to a Bitcoin node or indexer that can return raw transactions, block headers and Merkle branches.

11.1 Requesting an order

The front end sends the issuer's order service the user's Robinhood Chain address, which becomes the depositor; a 33-byte compressed Bitcoin public key that the user controls, which becomes the refund key; the amount in satoshis; and the recipient address. The issuer returns the `Assignment`, the operator `signature`, and the HTLC witness script and address for display.

Before proceeding, the front end verifies the Assignment locally.

1. `depositor` is the user's address.
2. `userRefundPubKey` is the key the user supplied.
3. `amountSats` is the amount the user requested.
4. `btcLockTime` allows enough time to complete the deposit, and `validUntil` has not passed.
5. `hashAssignment(assignment)` on the mint contract, when recovered against the signature, yields the address returned by `operator()`.

11.2 Registering the deposit

From the depositor's account, call:

```
register(Assignment assignment, bytes signature, address recipient, uint8 delivery) returns
(bytes32 depositId)
```

`delivery` must be `DELIVERY_WALLET` (0). The contract derives the HTLC address from the signed terms and publishes it in the `DepositRegistered` event as a 34-byte output script of the form `00 20 <32-byte hash>`. Encode it as a bech32 address beginning `bc1q` for display. Retain `depositId`; every later call uses it.

The address can be verified independently by calling the view `htlcFor(hashLock, minterPubKey, userRefundPubKey, btcLockTime)` through `eth_call`, which returns the script and the address the contract derives. The call costs nothing.

If a deployment uses issuer-supplied addresses instead of contract-derived ones, the front end must verify the issuer's script before displaying the address: the SHA-256 of the script must equal bytes 2 to 33 of the registered output script, and the script's hash lock, lock time and refund key must match the registered terms. Chapter 15 gives the byte offsets. An address that fails this check must not be shown.

11.3 Funding the HTLC

Construct a Bitcoin transaction with one output paying the HTLC address. The following rules apply.

- Send the whole amount in one output. The proof covers a single output, and the contract keeps one record per address.
- Send the exact amount in the order where possible. A proof for the exact amount may be submitted by anyone, including the issuer's service. A proof for any other amount may be submitted only by the depositor, so the front end must be prepared to submit it.
- Send at least the minimum amount set at deployment (Chapter 16).
- Register before paying. Bitcoin sent to an address whose deposit is not yet `REGISTERED` has no record on chain.

Wait for the required confirmations before building the proof.

11.4 Submitting the funding proof

The proof is the `BitcoinProof` structure.

Field	Type	Source
`blockHeight`	`uint256`	Height of the block containing the funding transaction
`transaction`	`bytes`	The raw funding transaction; witness serialization is accepted
`txIndex`	`uint256`	Position of the transaction within the block, starting at 0
`txOutputIndex`	`uint256`	Index of the output that pays the HTLC
`merkleProof`	`bytes32[]`	Sibling hashes from the transaction to the Merkle root, in order
`header`	`bytes`	The 80-byte header of the containing block
`prevHeader`	`bytes`	The 80-byte header of the preceding block

With Bitcoin Core, `getrawtransaction <txid> true` returns the block hash and the raw transaction; `getblock <hash> 1` returns the height and the ordered transaction identifiers, from which `txIndex`

and the Merkle branch are computed with double SHA-256; `getblockheader <hash> false` returns the header, and the same call on `previousblockhash` returns the preceding header.

Then call:

```
submitFundingProof(bytes32 depositId, BitcoinProof proof, uint64 fundedSats)
```

`fundedSats` is the output's actual value. The verifier rejects any other value. The verifier must already hold the header at `blockHeight`, and the number of blocks it has seen above that height must satisfy `minConfirmations`. If the call reverts with `InsufficientAssignmentConfirmations`, retry later. The proof remains valid after the output has been spent, because it proves inclusion in a block rather than the current state of the output. The mint contract is a fee-exempt caller of the verifier, so the only cost is gas.

11.5 Monitoring the mint

After the deposit is `FUNDED`, the front end monitors two things: the `PreimageRevealed` and `ProofMinted` events on the mint contract, and the HTLC output on Bitcoin. If the output is spent through the claim path, the spending transaction's witness is `[signature, S, 0x01, witnessScript]`, and `S` is the second element. If the issuer's reveal does not follow within a reasonable period, the front end may call:

```
revealPreimage(bytes32 depositId, bytes32 preimage)
```

Anyone may make this call. The token is minted to the registered recipient regardless of the caller. `ProofMinted` reports the funding transaction, the output index, the recipient and the amount in token units.

11.6 Refunding

If the issuer has not claimed by `btcLockTime`, construct the refund transaction as follows.

- Input: the HTLC outpoint, with `nSequence` set below `0xffffffff` (for example `0xfffffffe`).
- `nLockTime`: at least `btcLockTime`. The transaction becomes valid once the chain's median time past exceeds this value, which is roughly an hour after real time reaches it.
- Output: the user's own address, less the network fee.
- Signature: the BIP-143 digest for SegWit version 0, with the witness script as the script code, signed with the refund key.
- Witness: `[signature, <empty>, witnessScript]`. The middle element is an empty byte string. A single `0x00` byte is not accepted by the network's MINIMALIF rule.

A refund or claim with one input and one output is approximately 150 virtual bytes. Broadcast the refund promptly and wait for confirmation. No action is required on Robinhood Chain. After `expiry` the depositor may call `closeExpired(depositId)` to mark the record `CLOSED`; this is optional.

11.7 Changing the recipient

Until the mint occurs, the depositor may call `setDelivery(depositId, recipient, DELIVERY_WALLET)` to change the address that will receive the token.

11.8 Redeeming

To redeem, the front end requests a quote from the issuer, has the holder approve the redemption contract for `grossSats * SATS_TO_B18` token units, and calls `redeem(quote, signature)` from the holder's address before `validUntil`. Chapter 9 describes the fields and the minimum amounts.

11.9 Errors

Error	Meaning	Response
`AssignmentExpired`	`validUntil` has passed	Request a new order
`InvalidSignature`	The signature does not recover to the operator	Reject the order
`UnauthorizedDepositor`	The caller is not the depositor	Use the depositor's account
`ScriptAlreadyUsed`, `HashLockAlreadyUsed`	The terms reuse an address or hash lock	Request a new order
`InsufficientAssignmentConfirmations`	The verifier has not yet seen enough blocks	Retry later
`VerifierRejectedProof`	The proof does not match the registered address and amount	Check `txOutputIndex`, `fundedSats` and the Merkle branch
`OutpointAlreadyUsed`	The output has already been proven	None; the deposit is funded
`DepositNotRegistered`, `DepositNotFunded`	The call is not valid in the current state	Read `getDeposit` and consult Chapter 8
`InvalidPreimage`	The value does not hash to the registered hash lock	Re-read `S` from the claim transaction

Error	Meaning	Response
`SatsOutOfRange`	The amount is below the minimum	Refund after the lock time
`InvalidDelivery`	`delivery` is not `DELIVERY_WALLET`	Pass 0

12. Issuer operations

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

This chapter is written for the team that operates the issuer. It covers the keys the issuer holds, the procedures for each stage of a deposit, the handling of redemptions, and the conditions that should be monitored.

12.1 Keys and secrets

The issuer holds six distinct credentials. They should be held on separate systems with separate access controls.

Credential	Network	Purpose	Consequence of compromise
`operator` key	Robinhood Chain	Signs Assignments	An attacker can sign orders but cannot claim any deposit or mint any token. Pause registration and redeploy.
`quoteSigner` key	Robinhood Chain	Signs redemption quotes	An attacker can sign quotes that burn a consenting holder's tokens against a destination of the attacker's choosing. The burn still requires the holder's own call. Redeploy.
Bitcoin claim key	Bitcoin	Spends accepted deposits through the claim path	An attacker can claim funded deposits. Each claim publishes the secret, so the user can still mint. The issuer loses the Bitcoin.
Secret store	Off chain	Holds `S` for each deposit	An attacker can reveal secrets for deposits whose users have already refunded, minting unbacked tokens to those users. The exposure is bounded by `LATE_CLOSE`.
Submitter key	Robinhood Chain	Pays gas for proofs and reveals	An attacker can spend gas. No other capability.
Payout wallet key	Bitcoin	Pays redemptions from the reserve	An attacker can move reserve funds. This key is protected as reserve custody.

Each secret is 32 bytes from a cryptographically secure source and is never reused. Deriving each secret from an offline master secret and the deposit identifier, for example with HMAC-SHA256,

protects against loss of the secret store: a secret can be recomputed, and a deposit cannot be stranded by a database failure. Secrets are made available only to the claim and reveal processes.

12.2 Signing an order

In HTLC mode, an order is an EIP-712 signature over the `Assignment` structure, with the domain name `ProofMint`, version `2`, chain ID 4663 and the mint contract's address. Direct-custody orders use a different assignment under version `1` (17.4). Before signing, the order service verifies that:

- `depositor` is the Robinhood Chain address of the authenticated user;
- `userRefundPubKey` is a 33-byte compressed key supplied by the user, and `minterPubKey` is the current claim key;
- `hashLock` is new, and `btcLockTime` is at least `MIN_LOCK_LEAD` ahead and long enough for the full cycle, with 24 hours as a sound default;
- `expiry` is at least `btcLockTime + PREIMAGE_GRACE`, and `validUntil` is no later than `expiry`;
- `amountSats` is at or above the minimum, and `minConfirmations` is between 1 and 6 and at least the verifier's floor;
- `nonce` has not been used for this depositor.

The service returns the Assignment, the signature, the witness script and the derived address. The contract rejects anything these checks would reject, but a rejected registration costs the user gas.

12.3 Watching and proving

For each `REGISTERED` deposit, the watcher monitors the HTLC address. When a funding transaction reaches `minConfirmations`, the watcher reads the output's value.

- If the value equals `amountSats`, the submitter calls `submitFundingProof`, unless the user has already done so.
- If the value differs, the contract will not accept a proof from the issuer. The issuer either asks the user to submit the proof or signs an amended Assignment with the actual amount, after which the proof may be submitted by anyone.

The header relayer is kept current. A proof for a block the verifier has not received reverts, and every proof waits on the verifier's view of the chain.

12.4 Screening

Between the funding and the claim, the issuer applies its compliance program to the deposit. The inputs available are the funding transaction with its source addresses and history, the identity associated with the depositor at order time, and the amount. The program's outcome is a decision to

claim or to decline. A declined deposit requires no further action; it returns to the user through the time lock. Chapter 7 describes the window and its properties.

12.5 Claiming

The issuer claims a deposit only when all of the following hold.

1. The deposit is `FUNDED` on Robinhood Chain. The issuer never claims an HTLC whose deposit is not `FUNDED`, and never claims any output other than the one that was proven.
2. The deposit has passed screening, and the amount is one the issuer is prepared to honor.
3. The time remaining before `btcLockTime` exceeds the claim margin. A margin of two hours, or twelve blocks, covers the median-time-past lag and ordinary confirmation delays. Close to the lock time, the user's refund may confirm first and the claim would fail.

The claim transaction spends the HTLC with the witness `[signature, S, 0x01, witnessScript]`. From that moment the secret is public.

12.6 Revealing

The issuer waits until the claim transaction has the number of Bitcoin confirmations its policy requires, and only then calls `revealPreimage(depositId, S)`. Revealing before the claim is confirmed is the one error that can cost the issuer the deposit: if the user's refund confirms instead of the claim, the token has been minted and the Bitcoin has been returned.

12.7 Declining and closing

To decline a deposit the issuer takes no action. The issuer does not issue manual refunds in this system.

The issuer cannot close a deposit early. `closeExpired` is available to the depositor after `expiry` and to anyone after `expiry + LATE_CLOSE`. Closing deposits whose users refunded long ago is good practice, because it ends the exposure of their secrets.

12.8 Paying redemptions

The issuer's service monitors the redemption contract for `RedemptionRequested`. Each event records a burn that has already occurred and a payout the issuer now owes: `netSats` to `destinationScript`, with `feeSats` retained. The payout wallet makes the payment, the Bitcoin transaction identifier is recorded against the `quoteId`, and the reserve balance is reconciled against the token supply. There is no on-chain cancellation; a burned redemption is either paid or outstanding.

12.9 Monitoring

The following conditions warrant an alert.

- A `FUNDED` deposit within the claim margin of `btcLockTime` that has not been claimed. A decision is required: claim now or allow the refund.
- A claim transaction that has not confirmed within the expected time, in particular one competing with a refund in the mempool.
- A confirmed claim with no `PreimageRevealed` within a set period.
- Bitcoin received at an HTLC address that is not `REGISTERED`, or a second output to a funded address. Such outputs are not claimed; the user is contacted or the refund is awaited.
- A `RedemptionRequested` event without a payout within the issuer's service window.
- The verifier's `getLatestHeight()` falling more than a few blocks behind the Bitcoin chain tip.
- Any `register` call that reverts, which indicates that the order service and the contract disagree about validity.

13. Security model

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

13.1 Guarantees

Property	Enforced by	Mechanism
The issuer cannot take the Bitcoin without publishing the secret	Bitcoin consensus	The claim path requires a preimage whose SHA-256 equals the hash lock
The user can always recover an unclaimed deposit	Bitcoin consensus	The refund path opens at `btcLockTime` and never closes
No token is minted without a funded deposit	Mint contract	`revealPreimage` requires `FUNDED`, which requires a verified proof of an output paying the registered address
No token is minted without the secret	Mint contract	`revealPreimage` requires `SHA256(preimage)` to equal the registered hash lock
The amount minted equals the amount deposited	Mint contract and verifier	The verifier checks the output value for exact equality; the contract mints `fundedSats`
The HTLC address is the agreed HTLC	Mint contract	`register` derives the address from the signed terms
Only the user can accept a non-matching amount	Mint contract	A proof for an amount other than `amountSats` is accepted only from the depositor
A funded deposit cannot be closed out from under the user	Mint contract	Only the depositor may close before `expiry + LATE_CLOSE`
A pause cannot trap a funded user	Mint contract	`submitFundingProof`, `revealPreimage` and `closeExpired` are never subject to the pause
A redemption cannot be burned without the holder	Redemption contract	`redeem` must be called by the quoted holder, who must also have approved the burn

13.2 Assumptions

- Bitcoin and Robinhood Chain are operating and are not reorganized beyond the confirmation depths in use.
- The verifier holds the canonical Bitcoin header chain. A relayer that supplies an invalid header is rejected by the verifier's proof-of-work checks; a relayer that supplies nothing only delays proofs.
- The user retains the refund key and registers before paying.
- The user acts within `LATE_CLOSE` of `expiry` if the issuer has not claimed. In practice this means refunding soon after `btcLockTime`.
- Redemption payouts depend on the issuer's reserve. The chain enforces the burn and records the obligation; it cannot compel the Bitcoin payment. This is the one point at which a holder relies on the issuer, and the on-chain claim is the holder's evidence.

13.3 Capabilities of each party

Party	Can	Cannot
Issuer	Decline to claim; claim a funded deposit and reveal the secret; take any output the user sent beyond the proven one; delay a redemption payout	Mint without the secret; mint for an unfunded deposit; substitute a smaller funding; take a funded deposit without enabling the mint; burn a holder's tokens without the holder's call
User	Refund after `btcLockTime`; submit proofs; reveal a secret once it is public; change the recipient before the mint	Refund before `btcLockTime`; mint without the secret; retain both the Bitcoin and the token unless the issuer breaches its own procedures
Holder	Redeem with a signed quote; read any stored claim	Redeem without a quote; reverse a burn
Holder of a stolen `operator` key	Sign Assignments	Claim any deposit; mint anything
Holder of stolen secrets	Reveal secrets for deposits whose users have already refunded, minting unbacked tokens to those users	Take any Bitcoin
Anyone	Submit exact-amount proofs; reveal public secrets; close deposits after `expiry + LATE_CLOSE`; relay headers	Register on another's behalf; prove a non-matching amount; close early

13.4 Failure cases

Event	Outcome	Cost borne by
The user funds an amount other than the order	The user proves the actual amount; the issuer claims or declines; otherwise the user refunds after the lock time	No one, or the user's fees
The user splits the payment across two outputs	One output is minted; the other can be refunded unless the issuer takes it	The user, up to the second output
The user pays an unregistered address	An honest issuer does not claim and the user refunds. A dishonest issuer could claim with nothing to mint	The user, only against a dishonest issuer
The issuer's service fails after funding	The user refunds after `btcLockTime`	No one
The issuer claims but does not reveal	The secret is public; anyone reveals	The caller's gas
The issuer reveals before its claim confirms and the refund wins	The token is minted and the Bitcoin is refunded	The issuer
The issuer claims too close to the lock time and loses	The claim is invalid; the user keeps the Bitcoin; the issuer must not reveal	The issuer's fee
The issuer attempts to prove a small output of its own	Rejected: non-matching amounts are accepted only from the depositor	No one
The issuer claims before the proof is on chain	The proof remains valid; anyone reveals	No one
The user leaves a deposit unclaimed for `LATE_CLOSE`; the issuer closes it and then claims	The Bitcoin is lost to the issuer	The user, after a prolonged failure to act
The issuer's secrets are stolen	Unbacked tokens for deposits already refunded, until those are closed	The token's backing, bounded by `LATE_CLOSE`
The issuer fails to pay a redemption	The burn and the quote remain on chain as evidence; the issuer holds an unpaid liability	The holder, against a dishonest issuer
The verifier lags the Bitcoin chain	The proof reverts and is retried	Delay only

Event	Outcome	Cost borne by
The `operator` key is stolen	Orders can be signed but nothing can be minted; registration is paused	Contained

13.5 Operating rules

1. The issuer reveals the secret only after its claim is confirmed.
2. The lock time covers the full cycle with a margin for the median-time-past lag.
3. Functions that move a funded deposit forward are never subject to the pause.
4. Closing a deposit is a bookkeeping act, not a refund; nobody but the depositor may close before `expiry + LATE_CLOSE`.
5. An amount that differs from the order is proven only by the depositor.
6. Every burned redemption is paid, and the reserve is reconciled against the token supply.

13.6 Direct-custody mode

The guarantees in 13.1 and the failure cases in 13.4 describe HTLC mode. In direct-custody mode, live today, the user pays Bitcoin to a single-key address held by the issuer, so the issuer has custody of the deposit as soon as the payment confirms. The mint and any refund are then the issuer's actions, and the user relies on the issuer for both. What the chain still records is the signed order and assignment, the registered deposit and the mint against the payment's proof, and a refund can be paid only to the refund address the depositor signed. Chapter 17 gives the full trust model (17.7).

14. Contract reference

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

A deployment consists of the mint contract, the verifier, the token and, optionally, the redemption contract. All are written in Solidity 0.8.24 on OpenZeppelin Contracts 5.4.0. This chapter documents their interfaces.

14.1 Mint contract

Structures

```
struct Assignment {
    address depositor;
    bytes32 hashLock;          // H = SHA256(S)
    bytes   minterPubKey;      // issuer's 33-byte compressed key
    bytes   userRefundPubKey;  // user's 33-byte compressed key
    uint32  btcLockTime;       // Unix time
    uint64  amountSats;        // expected amount
    uint16  minConfirmations;  // 1 to 6
    uint64  expiry;
    uint64  validUntil;
    uint256 nonce;
}

struct Deposit {
    address depositor;
    address recipient;
    bytes   scriptPubKey;      // derived P2WSH, 34 bytes
    bytes32 hashLock;
    uint32  btcLockTime;
    uint64  amountSats;
    uint64  fundedSats;        // 0 until FUNDED
    uint16  minConfirmations;
    uint64  expiry;
    uint8   delivery;          // always DELIVERY_WALLET
    uint8   status;
}

struct BitcoinProof {
    uint256   blockHeight;
    bytes     transaction;
    uint256   txIndex;
```

```
    uint256   txOutputIndex;
    bytes32[] merkleProof;
    bytes     header;
    bytes     prevHeader;
}
```

EIP-712. Domain name `ProofMint`, version `2`, chain ID 4663, the contract address. This is the HTLC-mode Assignment. The direct-custody mint contract signs a different, 7-field assignment under version `1`; 17.4 gives it, and 17.8 lists every domain by mode. Type string: `Assignment(address depositor,bytes32 hashLock,bytes minterPubKey,bytes userRefundPubKey,uint32 btcLockTime,uint64 amountSats,uint16 minConfirmations,uint64 expiry,uint64 validUntil,uint256 nonce)`. The two `bytes` fields enter the structure hash as their `keccak256`.

Constants

Constant	Value	Meaning
`SATS_TO_B18`	10,000,000,000	Token units per satoshi for an 18-decimal token
`MIN_SATS`	Set at deployment	Smallest funded amount accepted
`DELIVERY_WALLET`	0	The only delivery: mint to the recipient. Named `DELIVERY_MAXY` in the reference source.
`STATUS_REGISTERED`, `STATUS_ISSUED`, `STATUS_CLOSED`, `STATUS_FUNDED`	1, 2, 3, 4	Deposit states
`MIN_LOCK_LEAD`	6 hours (default)	Minimum interval between registration and `btcLockTime`
`PREIMAGE_GRACE`	24 hours (default)	Minimum interval between `btcLockTime` and `expiry`
`LATE_CLOSE`	90 days (default)	Period after `expiry` during which only the depositor may close
`PAUSER_ROLE`	`keccak256("PAUSER_ROLE")`	May pause and unpause registration

Functions

Function	Caller	Preconditions	Effect
`register(Assignment, bytes signature, address recipient, uint8 delivery) returns (bytes32)`	The depositor named in the Assignment	Not paused; valid operator signature; `validUntil` not passed; terms well formed; `delivery == DELIVERY_WALLET`; hash lock and derived address unused; nonce unused	Derives the HTLC address, records the deposit as `REGISTERED`, emits `DepositRegistered`
`amendAmount(bytes32 depositId, Assignment, bytes signature)`	The depositor	Not paused; `REGISTERED` or `FUNDED`; same depositor, derived address and `minConfirmations`; new nonce	Replaces `amountSats` and `expiry`; emits `AmountAmended`
`setDelivery(bytes32 depositId, address recipient, uint8 delivery)`	The depositor	`REGISTERED` or `FUNDED`; `delivery == DELIVERY_WALLET`	Replaces the recipient; emits `DeliveryUpdated`
`submitFundingProof(bytes32 depositId, BitcoinProof, uint64 fundedSats)`	Anyone if `fundedSats == amountSats`; otherwise the depositor	`REGISTERED`; proof verifies; confirmations sufficient; `fundedSats >= MIN_SATS`; output unused	Records `fundedSats`, sets `FUNDED`, emits `DepositFunded`
`revealPreimage(bytes32 depositId, bytes32 preimage)`	Anyone	`FUNDED`; `sha256(preimage) == hashLock`	Sets `ISSUED`, mints `fundedSats * SATS_TO_B18` to the recipient, emits `PreimageRevealed` and `ProofMinted`
`closeExpired(bytes32 depositId)`	The depositor after `expiry`; anyone after `expiry + LATE_CLOSE`	`REGISTERED` or `FUNDED`	Sets `CLOSED`; emits `DepositClosed`

Function	Caller	Preconditions	Effect
`pause()`, `unpause()`	`PAUSER_ROLE`		Affects `register` and `amendAmount` only. The contract is deployed paused and unpaused after release checks.

Views. `htlcFor(bytes32 hashLock, bytes minterPubKey, bytes userRefundPubKey, uint32 btcLockTime) returns (bytes witnessScript, bytes scriptPubKey)` (pure); `hashAssignment(Assignment) returns (bytes32)`; `depositIdFor(bytes scriptPubKey) returns (bytes32)`; `outpointKey(bytes32 txid, uint256 vout) returns (bytes32)`; `getDeposit(bytes32) returns (Deposit)`; the public mappings `usedOutpoints`, `usedNonces`, `usedScripts` and `usedHashLocks`; and the immutable addresses `verifier`, `token`, `redemption` and `operator`.

Events

Event	Fields
`DepositRegistered`	`depositId`, `depositor`, `recipient`, `amountSats`, `expiry`, `delivery`, `scriptPubKey`, `hashLock`, `btcLockTime`, `minterPubKey`, `userRefundPubKey`
`AmountAmended`	`depositId`, `amountSats`
`DeliveryUpdated`	`depositId`, `recipient`, `delivery`
`DepositFunded`	`depositId`, `txid`, `vout`, `fundedSats`
`PreimageRevealed`	`depositId`, `preimage`
`ProofMinted`	`depositId`, `txid`, `vout`, `recipient`, `amount` in token units
`DepositClosed`	`depositId`

Errors. `ZeroAddress`, `InvalidVerifier`, `InvalidDelivery(delivery)`, `InvalidAssignment`, `InvalidHtlcKey`, `InvalidSignature(recovered)`, `AssignmentExpired(now, validUntil)`, `LockTimeTooSoon(btcLockTime, earliest)`, `ExpiryTooSoon(expiry, earliest)`, `ConfirmationsImmutable(stored, requested)`, `NonceAlreadyUsed(owner, nonce)`, `ScriptAlreadyUsed(depositId)`, `HashLockAlreadyUsed(hashLock)`, `DepositMissing(depositId)`, `DepositNotRegistered(depositId, status)`, `DepositNotFunded(depositId, status)`, `DepositNotClosable(now, closeAfter)`, `UnauthorizedDepositor(caller, depositor)`, `SatsOutOfRange(amount)`,

`OutpointAlreadyUsed(outpoint)`, `InsufficientAssignmentConfirmations(actual, required)`, `VerifierRejectedProof`, `InvalidPreimage`, `DeploymentAddressMismatch`.

14.2 Token

The mint contract requires `mint(address recipient, uint256 amount)` to be callable by its own address. The redemption contract, if used, requires `burnFrom(address holder, uint256 amount)` to be callable by its own address, spending the holder's allowance.

The reference implementation is an ERC-20 token with 18 decimals whose two authorities are fixed at construction: `mint` is callable only by the mint contract, and `burnFrom` only by the redemption contract. It has no owner and no pause. An issuer deploying its own token substitutes a role-based permission model as described in Chapter 10.

14.3 Verifier

The verifier maintains the Bitcoin header chain and verifies proofs against it. The mint contract uses the following functions.

Function	Purpose
`verifyTransactionWithOutputPaid(blockHeight, transaction, txIndex, txOutputIndex, merkleProof, outputValue, outputScript, header, prevHeader, payAsset, maxPayment) returns (bool)`	Confirms that the header is canonical and sufficiently confirmed, that the transaction is in the block, and that the named output pays exactly `outputValue` to `outputScript`. The verifier may charge a fee; the mint contract passes `maxPayment = 0` and is registered as fee-exempt.
`getLatestHeight()`, `getBlockHash(height)`, `minConfirmations()`	Chain state and the verifier's confirmation floor
`calculateTxId(transaction)`	Transaction identifier of a raw transaction, with witness data excluded
`quoteVerificationFee(outputValueSats, payAsset)`, `isFeeExemptCaller(caller)`	Fee discovery

Any party may extend the header chain with `updateHeader(height, header, prevHeader)` or its batch and reorganization variants. The verifier validates proof of work and difficulty and may pay a rebate from a treasury when one is configured.

14.4 Redemption contract

Function	Caller	Effect
`redeem(RedemptionQuote quote, bytes signature) returns (bytes32 quoteId)`	The `holder` named in the quote	Verifies the quote and the `quoteSigner` signature, burns `grossSats * SATS_TO_B18` from the holder, stores the quote, emits `RedemptionRequested`
`hashQuote(RedemptionQuote) returns (bytes32)`	View	EIP-712 hash under domain `ProofMintRedemption`, version `1`
`getClaim(bytes32 quoteId) returns (RedemptionQuote)`	View	The stored quote; reverts with `ClaimMissing` if unused
`destinationDust(bytes script) returns (uint64)`	Pure	Minimum payout for the destination type; reverts with `UnsupportedDestination` otherwise

Errors: `InvalidQuote`, `QuoteExpired`, `QuoteAlreadyUsed(quoteId)`, `UnauthorizedHolder(caller)`, `InvalidSignature(recovered)`, `UnsupportedDestination`, `DustPayout(netSats, minimum)`, `ClaimMissing(quoteId)`.

15. Bitcoin script reference

This chapter describes HTLC mode, which is planned. The mode live today is direct custody (chapter 17).

15.1 The witness script

Every deposit uses the same script with four values filled in. All data pushes are of fixed size, so the script is always 119 bytes and each value sits at a fixed offset.

Offset	Bytes	Opcode or data	Meaning
0	`63`	`OP_IF`	Branch on the last witness element
1	`82`	`OP_SIZE`	Push the length of the preimage
2 to 3	`01 20`	push 1 byte: `0x20`	32
4	`88`	`OP_EQUALVERIFY`	The preimage must be exactly 32 bytes
5	`a8`	`OP_SHA256`	Hash the preimage
6	`20`	push 32 bytes	
7 to 38	`<hashLock>`		`H`
39	`88`	`OP_EQUALVERIFY`	The hash must equal `H`
40	`21`	push 33 bytes	
41 to 73	`<minterPubKey>`		Issuer's claim key
74	`ac`	`OP_CHECKSIG`	Issuer's signature
75	`67`	`OP_ELSE`	Refund branch
76	`04`	push 4 bytes	
77 to 80	`<btcLockTime>`	little-endian	Lock time as a script number
81	`b1`	`OP_CHECKLOCKTIMEVERIFY`	The spending transaction must be locked to at least this time
82	`75`	`OP_DROP`	
83	`21`	push 33 bytes	

Offset	Bytes	Opcode or data	Meaning
84 to 116	`<userRefundPubKey>`		User's refund key
117	`ac`	`OP_CHECKSIG`	User's signature
118	`68`	`OP_ENDIF`	

Lock time encoding. `btcLockTime` is a Unix timestamp. It is at least 500,000,000, so Bitcoin interprets it as a time rather than a block height. Until January 2038 it fits in four bytes with the top bit clear, so the minimal encoding is a four-byte little-endian push. The value 1,760,000,000 encodes as `04 00 78 e7 68`.

15.2 The address

The deposit address is the pay-to-witness-script-hash (P2WSH) form of the script:

```
scriptPubKey = 00 20 <SHA256(witnessScript)>     34 bytes
```

For display, witness version 0 and the 32-byte program are encoded as a bech32 address with the `bc` prefix. The result is 62 characters beginning `bc1q`. The mint contract's `htlcFor` view returns both the script and the output script, and the `DepositRegistered` event carries the output script.

15.3 Spending the HTLC

Both spends are SegWit version 0 inputs. The signature covers the BIP-143 digest with the witness script as the script code and uses `SIGHASH_ALL`.

	Claim (issuer)	Refund (user)
Witness, bottom to top	`<signature>`, `<S>` (32 bytes), `01`, `<witnessScript>`	`<signature>`, `<empty>`, `<witnessScript>`
Signing key	`minterPubKey`	`userRefundPubKey`
`nLockTime`	Any	At least `btcLockTime`
Input `nSequence`	Any	Below `0xffffffff` (`0xfffffffe` is conventional)
Valid when	At any time while the output is unspent	Once the chain's median time past exceeds `nLockTime`
Publishes	`S`, readable by anyone	Nothing

The branch selector must be minimal: a single `0x01` byte for the claim and an empty byte string for the refund. The network's MINIMALIF rule rejects any other encoding, including a `0x00` byte.

Size and fees. A claim or refund with one input and one output is approximately 150 virtual bytes: about 41 non-witness bytes for the input, 31 to 43 for the output, 11 of overhead, and a witness of 195 to 230 weight units. The user pays the funding and refund fees; the issuer pays the claim fee.

15.4 Verifying a supplied script

Where the script is supplied by the issuer rather than derived by the contract, it is verified in three steps: compute `SHA256(witnessScript)` and compare it with bytes 2 to 33 of the registered output script; compare bytes 7 to 38 with the registered `hashLock`; decode bytes 77 to 80 as a little-endian number and compare with `btcLockTime`; and compare bytes 84 to 116 with the user's refund key. Any mismatch means the address must not be paid.

16. Deployment and parameters

16.1 Networks

Network	Chain ID	Role
Bitcoin mainnet		Deposits, HTLCs, redemption payouts
Robinhood Chain	4663	Mint contract, verifier, token, redemption contract

16.2 Reference deployment (direct-custody mode)

The direct-custody deployment on Robinhood Chain is live and has accepted new orders since 15 October 2026. Admission is an operational setting and may change. Chapter 17 describes how it works.

Contract	Address
Mint contract	0xb2b8dAB207810586e2b8cf8daEfFbe0D92990A9d
Token (BTC.R; on-chain symbol MAXYBTCb, 18 decimals)	0xA32Ec9e411aCF2ea403233e5e7f2f8366CC0d6C8
Redemption contract	0x9B21339E641bA89DfC1Fda27684D009B15b64D8A
Verifier	0xDbA3FaD120aE7bBBD42a39eAB65D2A41E030e29A

The operator key of this deployment, returned by the mint contract's `operator()`, is 0x1AA9B5542Ec1fd6221935dE1CF7BA612aFe17c58.

16.3 HTLC-mode deployment

HTLC mode is planned. It will be a new deployment of the mint contract. Depending on the option chosen in Chapter 10, the deployment either creates its own token and redemption contract or is granted permission on an existing token. The verifier is reused. Addresses are published when the deployment is live. Until then, any address presented as the HTLC-mode mint contract should be treated as unverified.

Contract	Address
Mint contract (HTLC mode)	Not yet deployed
Token	Not yet deployed, or the issuer's existing token
Redemption contract	Not yet deployed, or the issuer's own process
Verifier	0xDbA3FaD120aE7bBBD42a39eAB65D2A41E030e29A (reused)

16.4 Parameters

The parameters below are those of HTLC mode, except the EIP-712 row, which covers both modes. The parameters of direct-custody mode are in 17.9.

Parameter	Value
`SATS_TO_B18`	10,000,000,000 token units per satoshi (18-decimal token)
`MIN_SATS`	Set at deployment; 10,000 sats suggested
`minConfirmations`	1 to 6, never below the verifier's floor
`MIN_LOCK_LEAD`	6 hours (default)
`PREIMAGE_GRACE`	24 hours (default)
`LATE_CLOSE`	90 days (default)
Delivery	`DELIVERY_WALLET` (0) only
Verifier fee	The mint contract is fee-exempt
EIP-712 domains	`ProofMint` version 1 for direct-custody orders, assignments and refunds (17.8); `ProofMint` version 2 for HTLC-mode Assignments (14.1); `ProofMintRedemption` version 1 for redemption intents and quotes

16.5 Build settings

Solidity `0.8.24+commit.e11b9ed9`; optimizer enabled with `runs = 1`; `viaIR = true`; `evmVersion = paris`; OpenZeppelin Contracts 5.4.0. Imports resolve from the package root, including `@openzeppelin/contracts/`.

17. Direct-custody mode

Maxy Mint operates in direct-custody mode today. It has been live on Robinhood Chain since 15 October 2026, at the addresses in 16.2. In this mode the user pays Bitcoin straight to a deposit address held by the issuer, and the issuer's service proves the payment to the mint contract, which mints the token to the user's Robinhood Chain wallet. Chapters 2 to 15 describe HTLC mode, which is planned and not yet deployed (16.3). This chapter describes what differs in direct-custody mode: the order, the assignment, registration, payment, refunds and the trust model.

The token minted in this mode is BTC.R, an ERC-20 token with 18 decimal places whose on-chain symbol is MAXYBTCb. The service and the contracts carry the name ProofMint. Redemption works as Chapter 9 describes, through the redemption contract in 16.2.

17.1 How the mode differs

	Direct-custody mode (live)	**HTLC mode (planned)**
Where the user pays	A single-key deposit address in the issuer's wallet	An HTLC that neither party controls alone
Custody of the Bitcoin	The issuer's, from the moment the payment confirms	The user's until the issuer claims with the secret
What triggers the mint	The issuer's service submits the payment proof to the mint contract	The secret, published by the claim, presented to the mint contract
A payment that cannot be minted	Refunded by the issuer on the depositor's signed request (17.6)	Returned by the Bitcoin network after the lock time
Screening of the deposited Bitcoin	After it is in the issuer's custody	Between funding and claim, before custody
Who the user relies on	The issuer, for the mint and for any refund	The Bitcoin network and the mint contract
Signed order	`ProofMintBitcoinOrder` from the depositor, then a 7-field assignment from the operator	The 10-field HTLC Assignment of 14.1 from the operator
EIP-712 domain for the assignment	`ProofMint`, version `1`	`ProofMint`, version `2`

17.2 Steps

1. **The user requests an order.** The wallet or front end chooses the amount in satoshis, the depositor (the user's Robinhood Chain address), the recipient, a Bitcoin mainnet refund address, and the confirmations the payment must reach, from 1 to 6. The depositor signs an order authorization (17.3), and the front end sends the order and the authorization to the service. Every request that creates something carries an `Idempotency-Key`, so a retry with the same key and body returns the same order rather than a second one.
2. **The service returns the order and the assignment.** The assignment (17.4) is signed by the mint contract's `operator` key. Its `scriptPubKey` is the deposit address for this order. The front end checks that the order and the assignment match the request it signed (17.4) before going further.
3. **The user registers the deposit.** From the depositor's account the front end calls `register(assignment, signature, recipient, DELIVERY_WALLET)` on the mint contract. The contract accepts the assignment only if its signature recovers to `operator()`. Registration costs gas in Robinhood Chain's gas token, ETH.
4. **The user pays.** One Bitcoin payment of exactly `amountSats` to the deposit address, as output 0 of the transaction (17.5).
5. **The service mints.** When the payment reaches the confirmations chosen in step 1, the issuer's service submits the payment's proof of inclusion to the mint contract with `submitProof(depositId, proof)`, and the contract mints BTC.R to the recipient. The service reports the transaction in which the mint happened.
6. **Refund, if the payment cannot mint.** A payment of the wrong amount, a late payment or a second payment is not minted. The depositor requests a refund, and the issuer returns the Bitcoin to the order's refund address (17.6).

Until the mint, the depositor may change the recipient with `setDelivery(depositId, recipient, DELIVERY_WALLET)`. Delivery is to the wallet only; `delivery` is always `DELIVERY_WALLET` (0).

17.3 The order authorization

The depositor signs an EIP-712 `ProofMintBitcoinOrder` under the domain name `ProofMint`, version `1`, chain ID 4663 and the mint contract's address as the verifying contract.

Field	Type	Meaning
`deployment`	`address`	The mint contract
`epoch`	`uint256`	The custody epoch number published in the service's configuration
`evmChainId`	`uint256`	4663

Field	Type	Meaning
`orderId`	`string`	An identifier the front end chooses for the order
`depositor`	`address`	The user's Robinhood Chain address
`amountSats`	`uint256`	The amount to deposit
`refundAddress`	`string`	The Bitcoin mainnet address to which a refund is paid
`refundScriptPubKeyHash`	`bytes32`	SHA-256 of the refund address's output script
`requestPayloadHash`	`bytes32`	SHA-256 of the order request, described below
`nonce`	`string`	A fresh value for each authorization
`expiresAt`	`uint256`	Unix time after which the authorization is no longer accepted
`minConfirmations`	`uint16`	The confirmations the payment must reach, 1 to 6
`custodyPolicy`	`string`	`robinhood-backed-v1`, the custody policy under which the order is accepted

`requestPayloadHash` binds the fields that are not in the structure itself, in particular the recipient. It is the SHA-256 of these values joined by a NUL byte, in this order: the mint contract's address in lower case, the custody epoch number, the chain ID, the order identifier, the depositor and the recipient in lower case, the delivery value, the amount, the refund address, the refund output script in lower-case hex, the custody policy, and the confirmations. SHA-256 is used for the three inner hashes; keccak-256 is used only inside EIP-712 itself.

17.4 The assignment and registration

The assignment is the operator's signed authorization for one deposit. It is not the HTLC Assignment of 14.1: it has no hash lock and no Bitcoin keys, because the deposit address is the issuer's own.

```
struct Assignment {
    address depositor;
    bytes   scriptPubKey;      // the deposit address, as an output script
    uint64  amountSats;        // the exact amount to pay
    uint16  minConfirmations;  // 1 to 6, as chosen in the order
    uint64  expiry;            // end of the deposit window
    uint64  validUntil;        // last moment register accepts the assignment
    uint256 nonce;
}
```

EIP-712. Domain name `ProofMint`, version `1`, chain ID 4663, the mint contract's address. Type string: `Assignment(address depositor,bytes scriptPubKey,uint64 amountSats,uint16`

`minConfirmations,uint64 expiry,uint64 validUntil,uint256 nonce)`. The `scriptPubKey` field enters the structure hash as its `keccak256`. The contract's view `hashAssignment(assignment)` returns the same digest.

Before registering, the front end checks that:

1. the order identifier, depositor, recipient, amount, confirmations, refund address and delivery in the service's answer are those of the request it signed;
2. the assignment's `depositor`, `amountSats` and `minConfirmations` are those of the request;
3. `scriptPubKey` is a standard Bitcoin output script, and the deposit address the service displays encodes exactly that script;
4. `validUntil` has not passed;
5. the signature, recovered against `hashAssignment(assignment)`, yields the address returned by `operator()`;
6. an `eth_call` of `register` with the same arguments succeeds.

An answer that fails any of these checks is neither registered nor paid.

Contract interface. The mint contract of 16.2 offers the following.

Function	Purpose
`register(Assignment, bytes signature, address recipient, uint8 delivery) returns (bytes32 depositId)`	Records the deposit under the operator's signed assignment
`amendAmount(bytes32 depositId, Assignment, bytes signature)`	Replaces the amount and expiry under a newly signed assignment
`setDelivery(bytes32 depositId, address recipient, uint8 delivery)`	Changes the recipient before the mint
`submitProof(bytes32 depositId, BitcoinProof proof) returns (uint256 mintedAmount)`	Takes the payment's proof of inclusion and mints
`getDeposit(bytes32 depositId)`	Returns the depositor, recipient, `scriptPubKey`, `amountSats`, `minConfirmations`, `expiry`, `delivery` and `status`. Status 0 means no record and 1 means registered and open; any other value means issued or closed.
`closeExpired(bytes32 depositId)`	Closes an expired record

Function	Purpose
`hashAssignment(Assignment) returns (bytes32)`, `operator()`, `paused()`	Views used by the checks above

`BitcoinProof` is the structure of 11.4.

17.5 The deposit-first rule

The payment is one Bitcoin transaction whose **output 0** pays exactly `amountSats` to the deposit address. Change, if any, is the last output and returns to the payer.

The reason is how the service matches a payment to its order. It reads output 0 of the transaction that paid the deposit address and checks it against the assignment's script and amount. Fixing the deposit's position keeps that match unambiguous: the exact-amount check is made at one known output, and a change output can never be taken for the deposit. A payment that places the deposit at any other position is not minted.

A wallet's ordinary "send" chooses its own output order, so a front end builds the payment itself, deposit first and change last, asks the wallet only to sign it, checks that the signed transaction is the one it built and that output 0 still pays the exact amount to the deposit address, and only then broadcasts it.

17.6 Refunds

A payment that cannot be minted stays in the issuer's custody until the depositor asks for it back. The service records the reason as the payment's state:

State	Meaning
`UNDERPAID`	Less than `amountSats` arrived
`OVERPAID`	More than `amountSats` arrived
`LATE`	The payment arrived after the order's deposit window closed
`REPEATED`	More than one payment reached the deposit address

The refund has three steps.

1. **The depositor requests a refund quote.** The service answers with a quote: `quoteId`, `quoteDigest`, the outpoints to be refunded (`inputOutpoints`), `grossInputSats`, `feeSats`, `refundSats`, `maxFeeSats`, the refund address and output script, and an expiry. `grossInputSats`

equals `feeSats + refundSats`, and `feeSats` is at most `maxFeeSats`. A front end accepts a quote only if it pays the order's own refund address.

2. **The depositor signs the refund.** An EIP-712 `ProofMintBitcoinRefund` under the same domain as the order (`ProofMint`, version `1`, chain ID 4663, the mint contract), with the fields `deployment`, `epoch`, `evmChainId`, `orderId`, `quoteId`, `quoteDigest`, `inputOutpointsHash`, `refundScriptPubKeyHash`, `maxFeeSats`, `nonce` and `expiresAt`. `inputOutpointsHash` is the SHA-256 of the refunded outpoints, each written as the byte-reversed txid, a colon and the output index, sorted and joined with `|`.
3. **The issuer pays.** The issuer sends `refundSats` to the refund address and reports the refund transaction.

The refund address is fixed when the order is signed: its output script is committed in the order authorization's `refundScriptPubKeyHash` and again in the refund authorization. A quote expires; a depositor who misses it requests a new one.

17.7 Trust model

In direct-custody mode the user relies on the issuer from the moment the Bitcoin is paid. The deposit address is a single-key address in the issuer's wallet, so the Bitcoin is in the issuer's custody as soon as the payment confirms, and both the mint and any refund are the issuer's actions. The guarantees of 13.1 that rest on the HTLC do not apply in this mode: the user has no time-locked path back to the Bitcoin, and the issuer's acceptance of a deposit is not tied to a published secret.

What remains verifiable on chain is the following.

- The order the user signed, and the operator's assignment, fix the deposit address, the amount, the confirmations and the refund address before any Bitcoin moves. The user can check all of them before paying.
- Registration records the deposit on Robinhood Chain under the operator's signature, and the mint is made on chain by `submitProof`, which carries the Bitcoin proof of the payment.
- A refund can only be paid to the refund address the depositor signed.

HTLC mode is designed to remove the remaining reliance on the issuer for the mint and the refund. Until it is deployed, an integrator presents direct-custody mode to its users as a custodial deposit with an on-chain record.

17.8 EIP-712 domains by mode

Message	Signed by	Domain	Mode
`ProofMintBitcoinOrder` (17.3)	The depositor	`ProofMint`, version `1`, chain 4663, the mint contract	Direct custody
`ProofMintBitcoinRefund` (17.6)	The depositor	`ProofMint`, version `1`, chain 4663, the mint contract	Direct custody
Assignment, 7 fields (17.4)	The `operator` key	`ProofMint`, version `1`, chain 4663, the mint contract	Direct custody
Assignment, 10 fields (14.1)	The `operator` key	`ProofMint`, version `2`, chain 4663, the HTLC-mode mint contract	HTLC (planned)
`RedemptionIntent`: `quoteId`, `holder`, `destinationScript`, `grossSats`, `validUntil`	The holder, to request a quote	`ProofMintRedemption`, version `1`, chain 4663, the redemption contract	Both
`RedemptionQuote` (9.2)	The `quoteSigner` key	`ProofMintRedemption`, version `1`, chain 4663, the redemption contract	Both

17.9 Parameters

Parameter	Value
Chain	Robinhood Chain (4663); Bitcoin mainnet
Contracts	The direct-custody deployment in 16.2
Token	BTC.R (on-chain symbol MAXYBTCb), 18 decimals
Confirmations	Chosen per order, 1 to 6
Minimum order	10,000 sats
Delivery	`DELIVERY_WALLET` (0) only
Custody policy	`robinhood-backed-v1`
Gas token	ETH

The service publishes its current limits and its active custody epoch in its configuration, and a front end reads them from there rather than from this table.