Cross-chain Auctions via Bitcoin Double Spends

James Prestwich

Follow

Summa

--

Listen

Share

On November 15 at 10 am PT, Summa began theworld's first cross-chain auction. We are auctioning 10 non-fungible tokens (NFTs) on Ethereum for Bitcoin in a completely trustless process featuring direct interaction between Solidity smart contracts and Bitcoin payments. These 10 tokens are unique badges of participation in the first cross-chain auction. For a higher-level introduction to our products and instructions on how to participate in our auction, head over to this post. If you're interested in the SPV proof that drives auction settlement, definitely check out our previous blog post.

We leverage Bitcoin protocol features in a novel way to ensure bidding in our auction is fast and safe. Unlike Ethereum and other account based chains, many of Bitcoin's protocol rules lend themselves naturally to auctions and other public marketplaces. Bitcoin's transaction validation process enforces most of the rules of our auction.

The most compelling feature for a marketplace is the ability to make transactions mutually exclusive. Which is to say, with Bitcoin, we can have ten people create transactions, and ensure that at most one of those will ever go on-chain. In the context of an auction, this means that if a bid is accepted, all others become invalid immediately. There is no reasonable way to accomplish this in Ethereum. Instead, each other bid would error during smart contract execution, incurring transaction fees. With Bitcoin-based auctions, if your bid is not accepted, your funds do not move, and you do not pay a fee.

Double Updates

One of the fundamental goals of a blockchain is resolving the "double spend" problem. In a nutshell, this means preventing someone from sending the same coin to two people. However, beyond just simple spend transactions, it applies any time two transactions want to update the same state. This could be someone trying to duplicate Bitcoin, or two people trying to buy the same CryptoKitty. For the sake of generality, we'll call it the "double update"

problem. Fundamentally it's about ordering: when we see two things, how do we decide which is first, and what happens to the second one?

As it turns out, ordering is nearly impossible in an untrusted distributed system. Proof of Work and Proof of Stake are attempts at reliably coming to consensus on (among other things) transaction order. While the goal — reaching a shared state — is the same, different chains have different approaches to recording the current state. And in practice, the state model determines a given chain's approach to the double update problem.

State Models

There are two major state models in use today: Account

, which maps an address to its balance and current state, and UTXO

(unspent transaction output), which tracks specific coins by the transaction which last moved them and the current restrictions on moving them again (represented by an address). Account systems work more like a bank account, while UTXO systems work more like cash in your wallet. When you write a check, your bank balance is updated, but your account number doesn't change. But when you pay with cash you give up the bills you have and get different ones back with different values and serial numbers.

Ethereum-style Account chains resolve the double update problem in the contract execution phase. This is because it'snot always apparent whether a set of transactions contains a double update or not. On Ethereum, we can't check for a double update without running all transactions against the current state in varying orders. For example, if two people try to buy the same coins from Etherdelta, miners would include both transactions in the miner's preferred order. Instead of being able to rely on consensus rules, we have to rely on the Etherdelta smart contract to implement logic that causes the first caller to succeed and the second caller to fail. In other words, because we can't efficiently check for double updates, all transactions go on-chain, a smart contract sorts it all out, and everyone pays mining fees.

On the other hand, in Bitcoin-like UTXO chains, if two transactions try to spend the same coins, it is immediately apparent. The coins are uniquely identified, so we can easily check that identifier against the other transactions in our mempool. And

unlike Ethereum, where two transactions may not be double updates if broadcast at different times or with different contract states, Bitcoin double spends are state- and time-invariant

. Double update transaction sets will always and forever be double update transactions sets. Because Bitcoin resolves the double update problem at the block admission level, we can prevent double updates from ever going on chain. Conflicting transactions can never both go on-chain. In fact, nodes typically don't even allow conflicting transactions in their mempools. If a new transaction conflicts with a known one, it either replaces the old one in the mempool or is rejected.

To tie it back to Summa's cross-chain transactions, Bitcoin's rules against double spending align well with what we want in a market: a single definitive state (one buyer) where all other possible states (other bids) are invalidated and there are no unnecessary mining fees. All that's left is to ensure that Bitcoin will see all bid transactions as double updates.

Inputs and Witnesses

You may be thinking "how can we reliably turn all the bid transactions into double spends?" Easy: we add the same money to all of them. Yes, that's a bit tautological. But I promise this will make sense in a moment. First, we have to talk about how Bitcoin validates spending permissions. As mentioned earlier, Bitcoin addresses aren't like accounts with balances. Rather than addresses having balances, balances (specific UTXOs) have addresses. You can think of an address as "controlling" a UTXO. They're small tests that spenders have to pass. When someone wants to use that UTXO as an input to a new transaction (i.e. spend the coins), they have to pass the test.

Most addresses are public key hash (PKH or pubkeyhash) addresses. These addresses encode the hash of an ECDSA public key. PKH addresses require their associated public key and a valid signature to pass the test. The process is straightforward: hash the pubkey to make sure it matches the expected hash, then check that the signature both matches the pubkey and correctly signs the current transaction. Put simply, we test that the transaction is cryptographically connected to the signature, the signature to the public key, and the public key to the PKH address that controls the coins. Therefore, passing the test is a proof that the owner of the UTXO intended to sign this specific

transaction. In a transaction, each input gets its own proof (called a "witness", or a "scriptSig" in legacy transactions), to show that the owner of that input approved the transaction.

Buried inside this process is a powerful but infrequently-used feature. When we sign the transaction, we can choose

which parts of it we sign. We take the parts we want to sign, order them a specific way, and then hash them to create the "sighash." The sighash ties the transaction to the signature. By choosing what goes into it, we declare which parts of the transaction we "care about." The parts we care about can't be changed without invalidating our signature, but anyone can change the parts we don't care about. We call the process of building this hash the "sighash algorithm."

SIGHASH Algorithms

The Bitcoin protocol defines three different sighash types: SIGHASH_ALL, SIGHASH_SINGLE, and SIGHASH_NONE. And before you ask, the convention is to capitalize them. It's annoying. For convenience, we'll call the types "ALL" and "SINGLE." We can ignore "NONE" because it's a bad idea and we won't be talking about it again. When you sign your input, you choose one of these sighash types.

ALL commits to the entire transaction: every input and every output. Once you sign with ALL, nobody can change the transaction in any way without invalidating your signature. This means you have a complete veto over any change. On the other hand, SINGLE commits to all inputs, but only a single output. Signatures made with SINGLE ensure that nobody can add more inputs, but anybody can add more outputs.

In addition to the sighash types, there's a modifier we can set: ANYONECANPAY. It can be mixed in with ALL or SINGLE. If ANYONECANPAY is set, then rather than committing to every input, we only commit to the specific one we're signing. This means other people can add new inputs to the transaction — just like it says, anyone can pay! ALLANYONECANPAY locks in all outputs, while SINGLEANYONECANPAY (SACP

) commits to only one output. As such, SACP gives others leeway to add inputs and/or outputs.

For cross-chain transactions, we care about SACP. In a sale order or an auction, the seller specifies a price (or a series of prices in a dutch auction) and delivery information. We use SACP to encode that information into a signed partial transaction. This partial transaction becomes a template that bids must follow. SACP does exactly what we want. The buyer adds enough input value to pay the seller, and any other outputs the buyer wants (their change, for example). Signing SACP lets the seller specify a price and a delivery address encoded in the one output. And because all bids include the partial transaction, they all share the partial transactions single input and are thus mutually exclusive!

Dutch Auctions

It's easy to extend this to Dutch auctions using Bitcoin's time locking features. All sighash algorithms sign the locktime field. So to turn our partial transaction into an auction, we simply have to add a locktime and a few more partial transactions sharing the same input. Bitcoin transactions with locktimes become valid at specific block heights or Unix timestamps. To build a Dutch auction, we construct a series of partial transactions where each successive transaction has a lower output value than the last, and is locked to a later time. For example, we could build a transaction paying the seller 3 BTC valid now, another transaction paying 2.9 BTC valid in an hour, a third paying 2.8 BTC valid in two hours, etc. In this way, the seller can decide on a price curve over time, and control an auction without being online.

See an example dutch auction commitment set in the event logs of this transaction. And if you find the bug in this set of transactions, shoot me an email.

Summa's Transaction Format v1

Now that we have covered the fundamentals, here's the rundown of Summa's Bitcoin transaction format. The first input and first output (at index 0) are the seller's input and output. These can come from any valid partial transaction signed SACP by the seller. The buyer provides any number of additional inputs. The second output (at index 1) is an OP_RETURN output. These are special 0-value outputs that carry a small amount of data, broadly similar to Ethereum's event logs. This OP_RETURN output contains 20 bytes encoding the buyer's Ethereum address. The buyer can provide any other number of outputs at indexes 2+. The buyer should sign with ALL, to ensure that nobody else, not even the seller, can alter the transaction.

With this format, it's easy to see that the seller can't steal money because the buyer signed with ALL. Under Bitcoin's protocol rules, only one bid can be accepted, because all bids (even at different price points in the Dutch auction) share the seller's input. In addition, by including the OP_RETURN output, the buyer has committed to their Ethereum delivery address in a verifiable way. Following this format guarantees the buyer only pays if their bid is accepted, that delivery can be made automatically on-chain to both parties, and that no third party can interfere in the process.

Here's an example. Make sure to expand the details to see the Ethereum address in the second output!

Smart Contract Enforcement

In addition to the SPV proof validation, the smart contract needs to take two additional steps. To provide safety for the buyer, the contract forces the seller to commit to a partial transaction. Later it should ensure that the seller's input was used in the transaction contained within the SPV proof. Inclusion of the partial transaction is direct proof that the seller received acceptable payment. Fortunately, all inputs have a unique identifier (with some interesting but inconsequential exceptions), so our contract can easily check that. Because each auction has a specific associated input, that UTXO's identifier also identifies the auction or sale order.

After the transaction is validated, our contract extracts the buyer's address from the OP_RETURN output, and automatically delivers the sale asset to the buyer. Now both the purchase price and the sale asset have been delivered, so the auction is completely settled.

Why Link Chains?

Chains don't have to be siloed ecosystems. Summa aims to leverage the unique advantages of each chain to make a stronger and more united ecosystem. By leveraging Bitcoin's UTXO model and sighash algorithms in combination with Ethereum's expressive smart contract features we can build marketplaces that outperform anything that either chain could accomplish alone. As we expand these techniques to new assets and more complex instruments, we expect to see smart contracts interacting directly with cross-chain markets. Whether by buying Bitcoin, observing cross-chain markets, hedging exchange risk, or using Bitcoin to resolve double update problems, we anticipate that cross-chain smart contracts allow for instruments never before thought possible.

We believe direct observation of other chains is an incredibly powerful tool and we are determined to keep building. Thank you for joining us on our journey, and happy auctioning!

https://auction.summa.one/