Liquid: A Bitcoin Sidechain

Jonas Nick
Cryptographic Engineer
jonas@blockstream.com

Andrew Poelstra
Director of Research
andrew@blockstream.com

Gregory Sanders
Liquid Technical Lead
gsanders@blockstream.com

May 22, 2020

Abstract. Bitcoin is a digital currency for which no authorities are responsible for tracking the provenance or ownership of coins. Instead, the complete transaction history is publicly available for all users to cryptographically validate. To address the double-spending problem, in which multiple conflicting valid histories are created, Bitcoin uses a proof-of-work algorithm which economically incentivizes extending the current canonical history while disincentivizing the creation of alternate histories.

This innovation allowed, for the first time, the creation of a leaderless, censorship-resistant digital currency with no single point of failure. However, certain classes of users face challenges when using such a system. For the high-speed world of the cryptocurrency trader, transaction fees, variance in settlement times, publicly-broadcast transactions, as well as the risks of chain reorganizations can incur significant costs during their trading activities. However, bitcoin-denominated payments and contracts do not necessarily require settlement directly on the Bitcoin blockchain. The Liquid Network enables Bitcoin traders to achieve faster, more private settlements, and enjoy many other technological innovations, in exchange for working within a different security model.

In addition, the Liquid Network supports the issuance of new assets native to the sidechain, enabling atomic exchange of assets via multi-asset transactions. All assets on the network (Liquid Bitcoin and Issued Assets) enjoy the same consistent one-minute block times and cryptographic confidentiality of asset types and transaction amounts. Liquid supports Bitcoin Script and protocols built on top of it, e.g., the Lightning Network, as well as script extensions that enable advanced features such as covenants, keytrees, and more.

As a sidechain, Liquid supports transfers of bitcoins into and out of the system by means of a cryptographic peg. Bitcoin pegged into Liquid is referred to as Liquid Bitcoin or L-BTC. The forward progress of the Liquid ledger and custody of the underlying bitcoin are controlled by a federation, and remain secure as long as over 2/3 of its members are honest.

1 Introduction

Deployed in 2009, Bitcoin [Nak09] is a digital currency tracked by a publicly verifiable globally distributed ledger. Bitcoin has no central issuer, no single point of failure, and is designed to be resistant to censorship even against nation-state level adversaries. The backbone of the system is a *blockchain*, which contains every transaction since its network's genesis. The current state of the network, implied by the blockchain, is termed the *unspent transaction output set (UTXO set)*, and describes the amount and spending policy of all coins.

The rules governing Bitcoin transaction validity are known as *consensus rules*, so named because all participants in the system must agree on them to ensure they have a consistent view of the ledger state. This makes any changes to these rules slow and difficult, and because they affect the soundness of the system they must necessarily be very conservative. To allow quick innovation without the legal and logistical burden of creating independent digital currencies, Back et al. proposed the idea of *sidechains* in 2014 [BCD+14]. A sidechain is an independent blockchain which supports a peg mechanism to allow automatic transfers of Bitcoin between the Bitcoin blockchain and the sidechain.

Federated sidechains. The vision in the sidechains whitepaper is one of a *trustless peg*, in which transfers to and from the sidechain are cryptographically enforced. Since the point of a sidechain is that its validity rules are independent of those of Bitcoin, this presents a problem: how can the Bitcoin network validate that transfers from the sidechain were initiated according to the sidechains rules?

There are three high-level solutions to this problem:

1. One option is to extend Bitcoin with the ability to verify sidechain block headers, rather than entire blocks, and to trust that the presence of valid headers implies that the associated blocks are valid according to whatever the sidechain rules might be. This is the approach promoted by the Drivechain [SC17] project, and is known as an *SPV peg*.

   While this strategy would achieve the goal of independent sidechain rules without requiring ongoing complex changes to the Bitcoin protocol, it is very unclear how its incentive structure would play out in the real world. In particular, it allows a sufficiently powerful and well-funded subset of sidechain miners to steal all of the pegged coins.

   Further, it requires nontrivial changes to the Bitcoin protocol, which will prevent its deployment in the immediate future.

2. Another is to extend Bitcoin with the ability to verify arbitrary computations, using a cryptographic tool such as STARKs [BSBHR18] to allow Bitcoin verifiers to check the sidechain rules with far less resource consumption than it would take for them to track the sidechain itself.

   Unfortunately, while the last five years have seen tremendous advances, the cryptographic tooling is simply not sufficient for this option to be workable. We are excited to watch this situation develop over the coming years.

Relatedly, like the SPV peg, this option would require changes to Bitcoin, and there are no widely-accepted proposals in existence at the time of this writing.

3. Finally, in Appendix A of the sidechains whitepaper [BCD+14], a *federated peg* is proposed, in which sidechain coins are custodied by a consortium of *functionaries*, collectively called a **Strong Federation** [DPW+16]. These functionaries are responsible for verifying the sidechain rules and processing transfers to Bitcoin by signing transactions on the Bitcoin network.

   This option requires no changes to Bitcoin, since the peg is enforced by means of ordinary multisignature transactions. It does require a consortium to exist, and for participants of the system to trust that at least 2/3 of the federation is acting honestly.

   However, when these conditions are met, we believe that a federated peg is the best available option, and the one that we have chosen for Liquid.

Since the peg is enforced by a federation anyway, there is no additional loss of security by requiring that this federation also sign the blocks of the sidechain, rather than using a proof-of-work. In this trust model, sidechain reorganizations are impossible and we do not need to add compensating logic to our peg mechanism. In addition, users of the sidechain benefit from quickly and regularly produced blocks, which they can be assured will not be reversed.

Confidential Transactions and Bulletproofs. Unlike in Bitcoin, Liquid transactions typically do not have explicit amounts on any of their inputs or outputs. Instead, these amounts are replaced by *Pedersen commitments*, cryptographic commitments which have the special property that they are *homomorphic*, meaning they can be added together to produce new commitments. This means that while they reveal no information about the underlying amounts, validators can add up both the input commitments and the output commitments, and check that the results are equal, assuring themselves that no inflation or destruction of coins has occurred. Meanwhile, the absence of visible amounts greatly improves user privacy by hiding the total transferred amount of each transaction, making change outputs indistinguishable from ordinary outputs, and greatly amplifying the privacy guarantees of CoinJoin [Max13].

This feature, called *Confidential Transactions*, was developed for Liquid by Gregory Maxwell at Blockstream, and has since been adopted by many other blockchain systems, such as Monero [vS13] and MimbleWimble [Jed16].

To allow users to provide cryptographic proof of properties of their coins, such as to prove the total amount under control in a *proof of reserve system*, Blockstream also helped develop **Bulletproofs**, a general zero-knowledge proof system optimized for use with Pedersen commitments. This allows such properties to be proven while preserving the perfect privacy of the underlying amounts.

Assets and Issuance. Liquid is an asset issuance platform, meaning that the blockchain supports creation of new asset classes and permissionless (re)issuance of assets by any participant in the system. By extending Confidential Transactions to this context, we are able to hide the

asset class of each transaction input and output, along with the amount. Supporting multi-asset transactions in this way enables new use-cases such as atomic exchange of assets, while preserving privacy by ensuring that such exchange transactions are indistinguishable from ordinary ones.

Liquid allows for multisignature re-issuance of assets which allows for more secure asset management. For example, a tokenized fiat asset in Liquid may require 3 of 5 signatures for any given issuance, which decreases the likelihood of an unauthorized increase of the token supply.

2 Strong Federations

The backbone of the Liquid platform is a Strong Federation [DPW+16], a consortium of signers who are responsible for producing Liquid blocks and for processing transfers out of the system. Essentially, a Strong Federation is a consortium which is only able to take action given digital signatures from threshold-many members. In order to resist attacks outside of the technological realm, the Strong Federation members are distributed around the world, spanning a variety of jurisdictions and cultures. They have skin in the game because they are Bitcoin exchanges or brokers and are therefore incentivized to cooperate in advancing Liquid.

These digital signatures are produced by separate hardware called key modules which enforce business logic to ensure that their signatures are not used to sign forks of the Liquid blockchain or process unauthorized transfers. The underlying secret keys stored on these key modules are backed up offline such that both the participant responsible for operating the key module and Blockstream need to cooperate to decrypt the backup. This protects against hardware failure while ensuring that no individual participant has the ability to bypass the key module's protections.

Before exploring the details of this federation, we review some definitions.

- A *peg-in* is a transfer from Bitcoin onto the Liquid blockchain.
- A *peg-out* is a transfer from Liquid back onto the Bitcoin blockchain.
- A *blocksigner* is a member of the consortium responsible for signing Liquid blocks.
- A *watchman* is a member of the consortium responsible for signing the Bitcoin transactions that process peg-outs.
- Together, blocksigners and watchmen are referred to as *functionaries*, so named because their correct operation is fully deterministic, based on their view of the network.

We observe that the blocksigners and watchmen have conceptually independent roles, and these roles may be fulfilled by different federations. However, to simplify the trust model, in Liquid the same participants fulfill both roles.

2.1 The Peg

The core of the Liquid sidechain, or any sidechain, is the cryptographic peg which allows coins to move from Bitcoin onto the sidechain and vice-versa. There are two parts to the peg, which in Liquid work quite differently.

2.1.1 Peg-Ins

Transfers from Bitcoin to Liquid, or **peg-ins**, consist of two transactions. First a transaction on the Bitcoin network is created which transfers custody of some amount of Bitcoin to the Liquid federation. Next, a transaction is created on the Liquid blockchain which claims this transfer, creating an equal amount of Liquid Bitcoin (L-BTC) and transferring it to a destination of the user's choosing.

To ensure that the claimed coins go to the correct user, a cryptographic tool called *pay-to-contract* is used to bind the claimant destination to the Bitcoin transaction. Concretely, call the multisignature scriptPubKey describing the watchman coins (explicitly given in Section 2.3 below) `spk`.

Then a user transferring coins into Liquid acts as follows.

1. She chooses a Liquid scriptPubKey `dest` that she controls.
2. For each key P in `spk`, she computes the modified public key $P + H(P \| \texttt{dest})G$ and substitutes this into `spk`. Here H is a hash function, specifically HMAC-SHA256.
3. She sends her coins to the P2SH-P2WSH-wrapped version of the modified `spk`.

We observe that the modified `spk` is indistinguishable from random to anybody who does not know `dest`, so at this point her transaction cannot be determined to be a Liquid peg-in. This improves censorship resistance.

After the above transaction has been buried by 100 blocks, she may now claim the coins on the Liquid blockchain. To do so, she creates a new Liquid transaction sending coins to herself, which is ordinary in all respects except that it contains a *peg-in input*. Peg-in inputs differ from ordinary inputs in the following respects.

- Rather than referencing an unspent output of a previously created Liquid transaction, peg-in inputs reference unspent Bitcoin outputs: specifically, the output of the transaction that she created in the previous step.

  This is indicated to Liquid verifiers by setting the second-highest bit of the `vout` field to 1.

- The peg-in transaction contains a *peg-in witness* field with the following data:
  - `dest`, which allows validators to verify the commitment created in the pay-to-contract construction.
  - a satisfying witness for `dest` which signs the Liquid transaction, proving that the claimant actually controls the coins she is claiming
  - the serialized Bitcoin transaction and a Merkle proof connecting the transaction to a block header. Liquid users verify peg-ins by checking that the block is in their best chain and has at least 100 confirmations.

The federation monitors the Liquid blockchain and verifies such claim transactions, recognizing that the corresponding coins on the Bitcoin chain are now custodied by them. This is described in more detail in the following sections.

Anyone that is running the Liquid client is able to create a peg-in of Bitcoin to the Liquid network, although only federation members are able to peg-out.

2.1.2 Peg-Outs

Transfers from Liquid back to Bitcoin, termed **peg-outs**, are conceptually much simpler. They are simply Liquid outputs which destroy L-BTC using the `OP_RETURN` Script opcode, and which include the following data:

- The blockhash of the Bitcoin genesis block, to indicate that these L-BTC are intended to go to the Bitcoin chain.
- A scriptPubKey indicating the destination that the coins should have on the Bitcoin chain.
- A *PAK proof*, described in more detail in Section 2.3, which proves that this peg-out was initiated by an authorized Liquid participant.

The federation monitors the Liquid blockchain looking for such transactions, and when it sees them, it creates Bitcoin transactions transferring the appropriate number of coins to the appropriate destination.

While ordinary Liquid outputs have their amount and asset type hidden using Confidential Transactions, described below in Section 3, it is required that peg-out outputs explicitly reveal both their amount and asset type. Otherwise, it would be impossible for the federation to confirm that the user is actually transferring L-BTC and not some other hypothetical asset.

For network security, only members of the federation are able to create peg-out transactions, which are sent to approved whitelisted addresses controlled by federation members.

2.2 Consensus: Blocksigning

The purpose of federation of blocksigners is to consistently progress the linear chain of blocks that comprise the Liquid blockchain. They do so using a Byzantine Fault Tolerant (BFT) consensus algorithm [LSP82] to come to agreement on the next block in the chain, which they then publish to the network, and repeat.

In Liquid, blocks are signed every minute. Our BFT algorithm is designed such that once a block is produced, it is final — unlike in Bitcoin, users do not need to worry about chain reorganizations. As a further check, blocksigners have hardware signing modules which will refuse to sign a fork deeper than one block. This means that users are guaranteed finality of transactions after two blocks have been produced, which means an average delay of 2.5 minutes.

There are a variety of BFT consensus algorithms in the literature, but they are all forced to make a fundamental tradeoff between *safety* (the risk of a colluding subset of functionaries convincing the others to sign two conflicting blocks at the same height) and *liveness* (the risk of the chain failing to advance). This tradeoff is reflected in the choice of threshold value.

Concretely, suppose that there are n functionaries and threshold value of k, meaning that a block is considered valid only if it has signatures from k functionaries. It is clear that if $n-k+1$ functionaries are disabled, that the remaining $k-1$ functionaries cannot sign blocks, and the

chain will not progress. Conversely, if $2k - n$ functionaries collude to sign a pair of conflicting blocks, and partition the remaining functionaries into two sets of size $n - k$, they can get k signatures on both blocks and force the network. In other words, for a given value of k, we need k honest functionaries for liveness and $n - (2k - n - 1) = 2n - 2k + 1$ honest functionaries for safety.

These two risks are approximately equalized when $k = \lfloor 2n/3 \rfloor + 1$, which is the decision we made for the first iteration of Liquid. In particular, for a network of 15 functionaries, 11 signatures are required on blocks and at least 9 honest functionaries are required to prevent partitioning.

Since blocks can be produced by a single functionary in one pass, the BFT algorithm used in Liquid can be very simple: as a round-robin three-phase commit scheme which works as follows:

1. Every minute, the *round master*, chosen from the set of participants based on the clock time, creates a Liquid block proposal and sends to all peers.

2. Each peer, if they consider the block to be valid, indicates a *precommitment* to sign the block and broadcasts this precommitment.

3. Each peer, if they see sufficiently many precommitments to the correct block, actually signs the block and broadcasts this signature.

4. Each peer may then construct the complete signed block, using the original block proposal and received signatures, and broadcast this to the wider network.

The purpose of the precommitment phase is twofold. First, it ensures that functionaries do not produce actual signatures without being assured that a quorum of other signers have seen this block and signalled intent to sign it. Otherwise, if a signer was requested to sign two blocks at the same height, it would be unable to distinguish an attack from the benign scenario in which the first attempt had simply failed to achieve quorum. Second, the use of the precommitment phase gives a second threshold value, which individual functionaries can increase beyond k to improve the safety of the network (at the expense of liveness). Since this is not a network-wide parameter, it is easy for individual signers to adjust it according to real-time network conditions.

While this consensus algorithm is easy to reason about and allowed rapid deployment of the network, it comes with significant limitations. In particular, it requires that all functionaries have fairly well-synchronized clocks, and in the case that the round master is offline or malfunctioning, the other functionaries are unable to compensate and the round will fail.

Future versions of Liquid will use a more robust BFT protocol such as HoneyBadger [MXC+16] which does not have these limitations.

2.3 Custody: Watchmen

At first glance, the federation of watchmen appears to have a similar function to the federation of blocksigners, except that it is signing peg-out transactions rather than blocks. However, there are significant differences between these tasks which make their design considerations and operation very different.

- First, because watchman transactions are published to the Bitcoin network, they need not, and can not, worry about BFT consensus. Instead, watchmen must handle the case that transactions may not confirm even if they have sufficiently many signatures, in the case of volatile fee market dynamics or outright miner censorship. To ensure that this cannot result in double-processing of any peg-outs, watchmen enforce the invariant that any two transactions processing the same peg-out conflict with each other and cannot both be committed to the Bitcoin blockchain.

- Relatedly, because the Bitcoin blockchain confirms transactions only in a slow and probabilistic manner, peg-outs cannot be processed without a variable delay. This obviates the limitations described in the previous section for our round-robin three-phase commit, so this is the algorithm used for transaction signing in Liquid.

- Unlike for blocksigners, if watchman signing keys are lost or compromised, there is no recourse, even if the entire federation agrees to replace the keys. The custodied Bitcoins would simply be lost. Therefore liveness of the network and key management are much more serious concerns for the watchmen.

- Similarly, if the watchmen are compromised to process an invalid peg-out, there is no recourse. This requires the watchmen do much more extensive cross-checking of transaction proposals, reasoning about the actual semantics of the transactions rather than simply mechanically checking their validity against consensus rules.

- Adding to the complexity of these cross-checks, Bitcoin transactions have a *network fee* associated to them, which must be estimated by each watchman according to its view of the Bitcoin network. Since these views may be inconsistent, cross-checking transaction fees is necessarily complex and error-prone during times of highly volatile feerates.

Emergency Withdrawal. In the case that watchman keys are destroyed or unavailable, or the case that the Liquid Network is degraded and unable to process peg-outs, it is an essential design requirement that the coins custodied by the system are not lost. To this end, we have two lines of defense.

First, as mentioned above, all signing keys are backed up and encrypted to both Blockstream and the participant responsible for operating the watchman software.

Second, all coins custodied by the system have a Script equivalent to

```
11 <watchman keys> 15 CHECKMULTISIG NOTIF 4032 CSV <backup keys> CHECKMULTISIG ENDIF
```

The first half of this script, up to the first `CHECKMULTISIG`, represent the 11-of-15 watchman threshold. The second half, between `NOTIF` and `ENDIF`, give an alternate spending path in case the first is unusable. This alternate spending path is only available if the coins have been unmoved for 28 days ('4032 CSV' indicates that the Script can only be executed after 4032 blocks, i.e. 28 days, have elapsed), and allows the coins to instead be moved by an alternate *emergency withdrawal* set of keys. During normal operation the watchmen spend their coins well before the emergency timeout. Otherwise, the emergency script allows the coins to be recovered.

The Liquid emergency keys are held offline by non-overlapping components of Blockstream, in geographically distributed locations.

Peg-out Authorization Keys. To protect against software compromise or intrusion which allows an attacker to request a key module signature on unauthorized peg-out transactions, we could implement logic in the key module to confirm that every peg-out transaction is supported by the current state of the Liquid blockchain. However, this would require implementing complete Liquid consensus rules inside the key module, which would increase the complexity of the key module software by orders of magnitude, correspondingly magnifying the risk of an exploitable fault within the key module trust boundary.

Instead, we use a simpler mechanism in which every peg-out has a so-called **peg-out authorization key proof (PAK proof)**. This is a zero-knowledge proof which demonstrates that the peg-out destination is controlled by a known Liquid participant, without revealing which participant. In cryptographic terms, a PAK proof is a ring signature over a set of keys of the form

$$P_{\text{online}} + H(A - P_{\text{offline}}) \cdot (A - P_{\text{offline}}),$$

one for each participant. Here H is a hash function, A is the public key corresponding to the peg-out destination address, $A - P_{\text{offline}}$ represents (typically) a BIP32 tweak used to derive the address from a master key P_{offline}, and P_{online} is an additional key.

As suggested by the names, P_{online} needs to be online to produce a PAK proof, while P_{offline} only needs to come online to spend the pegged-out coins. Its presence in the PAK proof equation ensures that even if P_{online} is compromised, it is only possible to send coins to destinations partially controlled by the participant, who owns P_{offline}. However, an attacker could send coins to an output jointly controlled by P_{offline} and himself, giving him the opportunity to ransom coins, which is why we need the online key.

2.4 Dynamic Federations

The previous sections have taken the blocksigner and watchman federations to be fixed, immutable sets. However, these federations are composed of real-world entities whose interests change throughout time, and we need a mechanism to ensure continuity of the system as new participants join and old participants leave. It is also possible that the same participant may wish to rotate their keys in order to upgrade their key module or otherwise maintain security of the system.

On the other hand, it is important that such transitions happen with the consent of the existing federation, to not allow "coups" which would undermine the entire purpose of the federated trust model.

In designing such a scheme, it is also important to recognize that the consensus rules of Liquid, including the choice of federation keys, ultimately derive from a social consensus. If all participants agree to change the rules, they are free to do so, and there is no technical means to prevent this. Therefore the existence of a transition scheme is not a way to *enable* transitions as much as it is a way to allow such transitions to happen in an orderly fashion.

We have termed our transition scheme **Dynamic Federations**.

Liquid v1. The first version of Liquid does not include dynamic federations, meaning that the first change in federation will require a hardfork to the network and replacement of the key modules which ultimately control the custodied Bitcoin. To prevent future transitions from being so disruptive, we will ensure that the first one includes a hardfork to enable Dynamic Federations.

Dynamic Federations. In future, the Liquid blockchain will commit to its set of consensus parameters, by means of an additional Merkle tree in each block containing

- the current blocksigning Script;
- the current watchman Script;
- a freeform byte vector, which has no consensus meaning but which the federation will use to encode its PAK list;
- and "proposed" versions of the above 3 components

The Blocksigner federation will be responsible for choosing the contents of this Merkle tree; the Watchman federation will determine its signing policy by reading this tree, rather than having an independent configuration.

Transitions from one functionary set to another are executed as follows.

1. Once a transition plan has been decided (that is, a list of public keys describing the new federation has been agreed upon by the existing federation), the current federation updates their Blocksigner software configuration to initiate a transition.

   The Blocksigner software will validate that the new Watchman federation is able to satisfy the old Watchman policy (that is, at most 1/3 of the original watchmen have been removed or replaced in one transition). This ensures that coins pegged into the old federation will have time to be moved under control of the new federation.

   The Blocksigner software will further validate that it knows how to communicate with all peers listed in both the current and proposed federation scripts.

   If either of these checks fail, the Blocksigner software will simply refuse to operate, awaiting operator correction of its configuration.

2. Similarly, the watchmen, on startup, will check the Liquid blockchain and determine their current configuration from the tip. If they do not know how to communicate with any peers in the current configuration, they will refuse to start up.

   If they see a proposed transition which includes peers they do not recognize, they will raise a warning, and shut down when/if this transition becomes active.

3. As a safety measure, each Blocksigner will check whether 4/5 of its peers are signalling awareness of a proposed transition. If not, it will continue to set both the "current" and "proposed" consensus parameters of its blocks to the current consensus rules, although it will agree to sign others' blocks in which the "proposed" rules match the new federation indicated in its configuration.

   If 4/5 of signers indicate they are aware of the transition, the Blocksigner will propose blocks in which the "current" consensus parameters match the current federation, but for which the "proposed" consensus parameters match the new federation described in its configuration.

4. Validators of the Liquid blockchain, once they have seen one month's worth (43200) of blocks, for which 4/5 of them (34560) have their "proposed" consensus parameters set to some particular value, will consider the proposed parameters to be the true ones from that point onward.

   For an additional one month (43200 blocks), peg-ins using the old watchman federation will be recognized by the network. This gives users time to claim any outstanding peg-ins they initiated during the transition, and time for their software to recognize the new federation and peg-in addresses.

   This transition period is the reason that we require the new federation be capable of signing coins belonging to the old federation.

   After this transition period, coins sent to the old federation will not be recognized by the network as peg-ins, and manual intervention (perhaps using the emergency withdrawal keys, at great expense to the participant) will be required to recover the coins.

   During this transition, and potentially for some time afterward, the watchmen will move all coins controlled by the old federation under control of the new federation. This will occur naturally using the watchmen's ordinary transaction creation rules, which ensure that all coins are moved before the emergency withdrawal policy becomes active.

3 Confidential Transactions

Confidential Transactions (CT) [PBF+19] is a method to hide amounts in a transaction from third parties, while proving to the same third parties that input and output amounts add up. This ensures that no new coins are fraudulently created out of thin air. Furthermore, CT is used by default in Liquid to enable confidential assets and asset issuance (section 3.1). Users are still able to create non-CT transactions in Liquid if they choose to do so.

In blockchains following the *unspent transaction outputs* (UTXO) model, such as Liquid, every transaction spends existing outputs and creates new outputs. The total amount of spendable coins in the system are represented by unspent outputs. An output contains an amount and a specification of the data which is required to be provided by an eventual spender of the output. In Liquid the specification is written in a slightly more powerful version of Bitcoin Script and

usually requires a signature of the spending transaction which must verify with a public key contained in the specification.

CT consists of three main components that are explained below in an informal way: homomorphic commitments, key exchange and rangeproofs. The basic idea of CT is that plain amounts in outputs are replaced by *Pedersen Commitments* to those amounts. Using the fact that these commitments are additively homomorphic, the transaction creator proves that the difference between spent commitments and commitments in newly created outputs is zero. The blinding factors of the new commitments are communicated to the receiver by attaching an elliptic curve Diffie-Hellman key exchange with the blinding factor's entropy to the transaction. In order to prevent modular reductions when summing commitments, every transaction output contains *rangeproofs* showing that the committed amounts are small enough to not wrap around when adding them. Instead of implicitly computing the transaction fee as the difference between input and output amounts, the fee is specified using an explicit output with a plain amount to allow comparing blinded input and output amounts as we will see below.

Cryptographic commitments are binding and hiding. Binding means that a commitment to a value can not be opened to another value. Hiding means that the committed value can not be determined given a commitment. Pedersen commitments to value v with randomness r are constructed as $\mathrm{Com}\,(v, r) = vH + rG$ where H and G are generators of a group where the discrete logarithm problem is hard. Furthermore, the discrete logarithm between H and G must be unavailable. Otherwise, the commitment is not binding. In Liquid G is the standard generator of the secp256k1 group and H is a nothing-up-my-sleeve (NUMS) number generated by using the SHA256 hash of G as x-coordinate. The blinding factor r is drawn from a uniformly random distribution by the prover.

It is easy to verify that Pedersen commitments are additively homomorphic, i.e. $\mathrm{Com}\,(v_1, r_1) + \mathrm{Com}\,(v_2, r_2) = \mathrm{Com}\,(v_1 + v_2, r_1 + r_2)$ holds. In CT that means that the sum of spent commitments should be a commitment to a sum of the input values v_I and correspondingly for the sum of the newly created commitments v_O. If $v_I = v_O$ then the difference of input and output commitment is a commitment to 0. Additionally, The prover chooses the sum of blinding factors of the inputs r_I and of the outputs r_O to be equal. Therefore, when receiving the transaction the verifier checks that $\mathrm{Com}\,(v_I, r_I) - \mathrm{Com}\,(v_O, r_O) = \mathrm{Com}\,(0, 0) = 0$.

This does not yet prove that the total amounts in unspent transaction outputs remains the same after applying the transaction. The transaction creator could add two outputs $v_{O,0} = 1, v_{O,1} = n - 1$ (where n is the group order) to undetectably forge n coins. CT uses rangeproofs to prove in zero knowledge that the value in each commitment is small enough to avoid wrapping around. For efficiency reasons the limit is set as small as reasonably possible without introducing privacy leaks.

To create the rangeproofs the output amount is decomposed into a base 2 representation. The optimal basis depends on various factors – in Liquid it is base 4 – but for simplification we assume base 2 here. For every digit d_i of the base 2 representation the prover creates a commitment to $d_i 2^i$. The verifier will check that the sum of the digit commitments is equal to the commitment to the output amount. Now it remains to prove that the committed d_i are indeed either 0 or 1.

Assuming that is correct, every digit commitment $\text{Com}(d_i 2^i, r_i)$ is either a commitment to 0 or $\text{Com}(d_i 2^i, r_i) - d_i 2^i H$ is a commitment to 0. Because Pedersen commitments to 0 are computed as $0H + r_i G$, the transaction creator knows its discrete logarithm with respect to G, namely r_i. This would be impossible if it would not commit to 0 because the discrete logarithm between G and H is not known to the prover.

Liquid uses *ring signatures* to prove knowledge of the discrete logarithm of either the digit commitment or the digit commitment minus $2^i H$ without revealing which one exactly. Basically, these ring signatures are similar to Schnorr signatures, which are two elements (e, s) such that $e = \text{hash}(P||m||sG - eP)$ for public key P and message m. Then a signature for ring of public keys $\{P_1, P_2\}$ are three elements (e, s_0, s_1) such that

$$e = \text{hash}(P_1||P_2||m||s_2 G - \text{hash}(s_1 G - eP_1)P_2).$$

To get an intuition for how this works let us look at the signing algorithm. Assuming the signer knows the discrete logarithm x_1 of P_1, it first chooses random scalars k and s_2 and computes

$$\begin{aligned} e &= \text{hash}(P_1||P_2||m||s_2 G - \text{hash}(kG)P_2) \\ s_1 &= k + ex_1. \end{aligned}$$

It should be straight forward to see that such a signature (e, s_1, s_2) is correct by plugging it into the verification equation above. Conversely, assuming the signer knows the discrete logarithm x_2 of P_2, it first chooses a random scalar k and s_1 and computes

$$\begin{aligned} e &= \text{hash}(P_1||P_2||m||kG) \\ s_2 &= k + \text{hash}(s_1 G - eP_1)x_2. \end{aligned}$$

In CT there is not only one ring signature, but one per digit. This allows an optimization called Borromean Ring Signatures [MP15] where essentially the e value is shared between rings. Additionally, the message m includes a list of all digit commitments. The above range proof without Borromean Rings requires 1 digit commitment, and $m + 1$ scalars per digit for a base m representation. Assuming the size of group elements and scalars is the same, for n digits the required space is $(m + 2)n$ elements. Borromean ring signatures reduce the number of elements to $(m + 1)n + 1$.

In order to successfully send a CT payment, the recipient needs to receive the opening of the amount commitment. For that purpose, Liquid addresses do not only encode the hash of a public key as Bitcoin addresses do, but they also contain a public key allowing the sender to initiate a Diffie-Hellman (DH) key exchange. First, the sender creates an ephemeral key pair, multiplies the secret part with the recipient's DH public key and hashes the result to obtain the randomness seed of the rangeproof. In order to allow the receiver to derive the same seed, the sender attaches

its ephemeral key to the output. The seed is used in the rangeproof to deterministically derive the digit commitment blinding factors. This gives the receiver the ability to recover the amount commitment blinding factor as the sum of the digit commitment blinding factors. The receiver can further "rewind" the proof to get the committed amount[1] It is easily possible for the receiver to export the key used for the DH exchange and give it to a third party. This allows auditing the receiver's incoming and outgoing amounts.

As a result, to support CT in Liquid for each output there is the following additional data attached to transaction outputs: 33 bytes for the amount commitment, 33 bytes for the sender's ephemeral DH key and about $((4+1)(36/4)+1)32 = 1472$ bytes for the rangeproof using above formula for Borromean ring signatures with base 4 and a default upper limit of 2^{36}. We will see in the next section what additional data is required to support confidential Issued Assets.

3.1 Assets and Issuance

As a sidechain, Liquid allows the transfer of bitcoin which are secured by the Liquid federation. Furthermore, users are allowed to issue additional assets, referred to as Issued Assets. This means that when making a payment they can choose which of their available assets are transferred in a transaction. By building on Confidential Transactions, the actual asset types in a transaction are hidden from third parties. Liquid only deals with transferring the ownership of an asset. Additional rules such as pegs to other assets and allowing users to redeem assets have to be enforced by the issuer and trusted by the user. In that sense the native Bitcoin asset can be understood as being issued by the Liquid federation with the agreement that the coins can be redeemed for main chain bitcoin through peg-outs.

Among other advantages over alternative approaches such as "colored coins", native assets on Liquid allow *atomic exchanges*. If Alice wants to sell two L-BTC for one asset A, she can create a transaction spending two L-BTC and an output with one asset A to herself. This transaction is invalid because the input and output assets of a transaction do not match. Therefore, she can broadcast the transaction to other users. Bob is interested, adds an input that spends one asset A and an output with two L-BTC to himself. Now the transaction is valid and can be broadcast to the network. Neither Alice nor Bob can be cheated and, as opposed to atomic swap based exchanges, there are no phases where Alice or Bob can back out.

In Liquid, assets are represented by *asset tags*. They are NUMS points and therefore sampled uniformly at random from the group and without known discrete logarithm with respect to group generator G. We will see later how they are derived exactly. Let us first look at simpler scheme than what is deployed in Liquid where amounts are hidden but assets are not. In such a system the asset tag is information added to every output. Moreover, amount commitments do not use second generator H anymore but they use their respective asset tag as second generator. That means that for asset tag H_A the commitment to amount v with randomness r is $\mathrm{Com}_{H_A}(v,r) = vH_A + rG$. It is not possible to verify from the commitment alone that it indeed was created with asset tag H_A. But cheating by using a different generator than H_A is prevented by the

[1] See Adam Gibson's excellent exposition for further details. `https://github.com/AdamISZ/ConfidentialTransactionsDoc/blob/master/essayonCT.pdf`

rangeproofs discussed above which now use the asset tag as generator H_A instead of H. Just as in the above section, the verifiers check that the difference between spent and output amount commitments is 0.

In order to hide the asset tags being used in transactions from third parties, asset tags are not actually put into transaction outputs. Instead, there are *asset commitments* to asset tags. For an asset tag H_A the asset commitment with randomness r is $\mathrm{Com}_{H_A}(1, r) = H_A + rG$. Correspondingly, the amount commitment is computed using the asset commitment instead of the asset tag as second generator and, because the asset tag is unknown to verifiers, they use the asset commitment in the rangeproof. The blinding factor r of the asset commitment is chosen uniformly random per asset, but in such a way that the verifier can still check that the difference between spent and output amount commitments is 0. For example, assuming a transaction spends one unit of asset H_{A0} and two units of H_{A1} and creates three outputs with one unit each, the verifier checks that

$$\begin{aligned}&(H_{A0} + r_{I0}G) + r_{I1}G + 2(H_{A1} + r_{I2}G) + r_{I3}G \\ &= (H_{A0} + r_{O0}G) + r_{O1}G + (H_{A1} + r_{O2}G) + r_{O3}G + (H_{A1} + r_{O4}G) + r_{O5}G.\end{aligned}$$

This holds if the transaction creator chooses all output blinding factors at random except $r_{O5} = r_{I0} + r_{I1} + 2r_{I2} + r_{I3} + r_{I4} - \sum_{i=0}^{4} r_{Oi}$. As a result, no one but the sender and the receiver know which output asset corresponds to which input asset. The more transactions have happened between asset issuance and asset commitment the harder it is to associate the commitment with a particular issuance because they hide more and more among other assets. Asset tag and blinding factor are communicated to the recipient in the same way as the amount: with a DH exchange and a rangeproof rewind.

The scheme we outlined above is not sufficient to prevent fraudulent behavior. A transaction creator would be able to forge an arbitrary asset without spending a corresponding asset. For example, this would work by adding two outputs, one with an asset commitment to the desired asset tag and one with the negation of the commitment. This will not be spotted when summing input and output commitments. Therefore, every output contains an *asset surjection proof* (ASP) which proves that the asset tag in the asset commitment is one of the spent asset tags without revealing which one. Essentially, in an ASP the prover chooses a subset of spent asset commitments, subtracts the output asset commitment from every asset commitment in the set and creates a ring signature over the resulting list. As an example assume the transaction spends asset commitments $\{H_{A0} + r_0G, H_{A1} + r_1G, H_{A1} + r_2G\}$ and creates a new output with asset commitment $H_{A1} + r_OG$. Then the ASP is a ring signature over the ring

$$\{H_{A0} - H_{A1} - r_3G, (r_1 - r_O)G, (r_2 - r_O)G\}.$$

Remember that the ring signature is a zero-knowledge proof of knowledge of the discrete logarithm with respect to G of least one member of the ring. In above example the prover knows the discrete logarithm of two ring members. Conversely, if the output asset tag would not be equal to any spent output tag, the prover would not know the discrete logarithm of any ring

member. Because there needs to be one ring signature per output, proofs for transactions with many inputs and outputs can be quite large. Therefore, the default ring size in Liquid ASPs is set to three.

So far we have not covered how assets come into existence. Essentially, the asset issuance consists of additional data that is added to a regular transaction input, mainly the new asset tag and asset amount. It is necessary that asset tags are unique. So in order to derive the new asset tag the tuple of transaction identifier and index of the spent output (which is itself unique) is hashed along with a Ricardian contract [Gri04] hash and a constant. The Ricardian contract can be a legal document specifying the conditions of using the asset enforced outside of Liquid. In order to speed up derivation of the asset tag we use an optimization to Merkle tree hashing. Instead of a normal SHA256 application on the concatenation of left and right branch, the returned hash is the state after the first compression round. As a result, the padding is not taken into account and a compression round is saved. This is secure because left and right branch together are always exactly 64 bytes.

Amounts in issuances can be either plain or blinded by using an amount commitment. Additionally, issuances contain a plain or blinded reissuance token amount. These tokens can be transferred just as any other asset and allow the holder to reissue more of the corresponding asset. If no reissuance capability is desired the reissuance token amount is set to 0. In summary, asset issuance requires a special extra input attached to a normal transaction input. The normal input is necessary to provide entropy for the asset tag. In practice, this is no problem because there will always be an input because just like any other transaction, issuances require fees. They must be paid in an asset specified by the federation, which is L-BTC by default.

4 Tooling

The Liquid blockchain is an asset issuance platform. However, much of the power and capability of Liquid comes not from the blockchain itself, but on the tooling on top of this. In this section we describe some of the tools we have developed and are continuing to develop.

4.1 Proof of Reserve

Proof of Reserve (PoR) is a tool developed by Blockstream to meet the challenges of trading desks, OTC market makers, and exchange customers needing assurances that claimed funds actually exist, and are within the custody of the claimant. PoR is an open-source tool which creates a *Partially Signed Bitcoin Transaction (PSBT)* [Cho17], an interoperability format for Bitcoin signers also developed in part by Blockstream. It passes the PSBT to a signer, such as a hardware wallet, when then passes it to the requester of the proof for validation. This PSBT, unlike ordinary PSBTs, is not actually a valid Bitcoin transaction (even after signing), meaning that funds are not at risk of moving or being stolen, even if the destination is not controlled by the signer. This also prevents any Denial of Service risk related to moving coins unnecessarily.

Near-term extensions to PoR will support Liquid directly, including confidential amounts and asset types. The prover will not have to reveal the amount or asset type of the specific UTXOs,

and yet still prove authorization of a total amount of funds of each type. With identical tooling an exchange can prove their on-chain assets, both BTC, L-BTC, and any other issued asset on the Liquid chain without adding any additional tooling.

Future work will include enhancing the privacy of PoR using zero-knowledge proofs, which would allow these invalid "transactions' to hide which inputs they reference. This would let a prover demonstrate control over some number of coins without revealing the exact coins that they own. However, current-generation zero knowledge proofs are not yet efficient enough to realize this vision.

This project can be found at `https://github.com/ElementsProject/reserves`.

4.2 Vaults

Liquid's extensions to Bitcoin Script include a new opcode `CHECKSIGFROMSTACK`, which enable spending policies that require a digital signature on some arbitrary data, not just the spending transaction. This opcode can also be used to enforce arbitrary restrictions on the transactions that spend some coins, a capability known as *covenants*. Using covenants, Russell O'Connor was able to implement a scheme known as a *Möser-Eyal-Sirer Vault* [MIGS16].

Vaults are transaction outputs which cannot be directly spent, but which must first be sent to a "staging area", an output type which restricts the coins from moving again without a significant time delay. During this delay, the coins may be returned to the vault, effectively cancelling the original transaction. This way, even in the case of key theft, the original owner is therefore able to retain coins in the vault indefinitely.

The original vault construction was created for Elements Alpha, a technological precursor to Liquid, which included a primitive version of Bitcoin's "Segregated Witness" (Segwit). This made the vault particularly inefficient, as it required constructing an entire transaction on the Script interpreter's stack, and limited it to non-CT transactions. However, Liquid uses the modern version of Segwit which was deployed on Bitcoin, and these limitations do not apply.

This makes vaults efficient to deploy, and practical even for transactions which use CT.

4.3 Liquid Securities

The Liquid Core software and consensus layer focuses on establishing history of token ownership, not the meaning of a particular token or what abilities ownership of asset imbues to the holder. In other words, a token representing shares of a stock need no special on-chain logic to enforce constraints; these can instead be executed by the issuer or regulator directly. This is both a privacy and a scaling win, with no additional identifying logic on-chain.

To close the gap between customer requirements and low-level chain logic the Liquid Securities suite of software builds on the solid foundations of Liquid Core. The Liquid Securities suite enables feature-rich, legally-compliant issuance and management of security tokens. This suite takes advantage of the separation of concerns, by not encumbering the base layer with complex legal logic, and instead use the Liquid blockchain as a pure settlement network. The higher-layer logic is instead enforced on top.

Features of the suite include:

- Issuance, reissuance, and distribution of securities tokens to holders.
- Investor management, investor identification compliance.
- Non-custodial investor wallets, with co-signer enforcing the required securities logic.
- Dividend payments to securities token holders.

4.4 Lightning

The *Lightning network* is a layer-2 payment network developed for Bitcoin by many contributors throughout the world. It works by chaining payments across paths composed of two-party *payment channels*, which are updated atomically to ensure that payments are not lost partway through a path. These payment channels work by having the two parties jointly create a transaction which is not published to the blockchain. Instead, this transaction is repeatedly replaced to reflect the changing balance in the channel. Only when the parties agree to close the channel, or one party goes offline, is anything published to the blockchain.

While Lightning can be adapted to Liquid with nearly no changes, its design assumes that all amounts and asset classes are visible to all participants. Blockstream is funding research into Lightning along multiple axes:

- First, basic Lightning support in which users of the Lightning-on-Liquid network must reveal their asset types and amounts to allow the ordinary Lightning payment routing logic to work.
- From here it is possible to extend Lightning to cross-chain payments, allowing transfers between Bitcoin and Liquid L-BTC nearly instantaneously, bypassing the peg mechanism.

  While cross-chain Lightning payments have been critiqued because of problems related to price volatility between different blockchains' assets, these problems do not apply between BTC and L-BTC because the peg maintains an almost constant price close to 1.
- In parallel to this, with some additional cryptography it is possible to create Lightning channels where not all assets and amounts are revealed. This research will also apply to other blockchain systems which may want to support Lightning on top of Confidential Assets.
- Finally, Blockstream has spearheaded a research program called *scriptless scripts* [Gib17, MMSS+19] which would allow Lightning channels whose blockchain transactions look like ordinary spends. Scriptless scripts allow payment channels to be linked into paths such that even the participants in the individual channels (except those at the endpoints) cannot see which payments are being routed through which paths.

  This is an exciting area of research which promises to improve the privacy and scalability story for both Bitcoin and Liquid.

5 Future Work

As a Bitcoin sidechain and fork of Bitcoin Core, Liquid research and development has always progressed alongside research in Bitcoin itself. Ideas such as segregated witness [LLW15] and relative timelocks [BFL15] were first introduced in Liquid's open-source counterpart *Elements Alpha* and later proposed for Bitcoin, while more disruptive features such as Confidential Transactions have spurred research across the Bitcoin ecosystem.

Going forward, we hope to maintain this cadence of collaborative innovation, both by developing ideas within Bitcoin community which we can bring into Liquid, and by pushing the boundaries of blockchain research forward in ways that will bring long-term benefits to the wider ecosystem.

eltoo. Leveraging a proposed extension to Bitcoin called `SIGHASH_NOINPUT` [Poo16], the protocol **eltoo** [DRO18] co-developed by Blockstream is a successor to the transaction update mechanism in the Lightning Network which solves many limitations of the original LN. In particular, the Lightning Network requires participants store recovery transactions corresponding to every individual state update, to be used in the case that an adversarial or confused participant publishes an expired state to the blockchain. Each of these transactions must be stored and kept secret for the entire lifetime of a Lightning payment channel, at great cost in both storage and operating complexity of the system.

eltoo uses `SIGHASH_NOINPUT` to create update transactions which may be bound to any previous state of the channel; then even in the case that an expired state is published, users may simply publish the most recent state to bring the blockchain up to speed on the actual state of the channel. This means that users need only store the most recent transaction in a payment channel, and need not worry about keeping this secret.

Like the original Lightning network, eltoo can link payment channels across blockchains. While this results in complications for blockchains whose asset prices are volatile relative to each other, in the case of Liquid L-BTC and Bitcoin, the two assets are in one-to-one correspondence anyway, so no such complications arise. This allows the Lightning network to act as a seamless onboarding and offboarding mechanism for Liquid.

By extending eltoo to the multi-asset case, we can use it as the basis for a second transaction layer on top of Liquid, greatly improving its privacy and scalability.

Taproot and Scriptless Scripts. Developed in part by Blockstream's research team, *Taproot* is a proposed extension to Bitcoin which promises to improve the privacy and scalability of the system by hiding all script policy in the case that all participants in a transaction act cooperatively.

It does so by a combination of two technologies. First, transaction outputs in Taproot are simply elliptic curve keys, and these outputs can be spent using a digital signature corresponding to this key. Using the same pay-to-contract technology used by Liquid peg-ins, this key can also commit to additional script semantics, in such a way that the commitment is never revealed unless these additional semantics are actually necessary.

Secondly, using a combination of MuSig Schnorr multisignatures [MPSW18], co-invented by the Blockstream research team, and *scriptless scripts* [Gib17, MMSS+19], it is possible to encode the majority of common script semantics in the public key itself, eliminating the need to reveal the script commitment for common cases. In particular, an extension of MuSig allows the creation of threshold signatures (or indeed, any policy defined by a number of admissible signer sets), while scriptless scripts allow these threshold signatures to additionally reveal secret data, such as the hash preimages used by Lightning HTLCs or the encryption keys used in Zero-Knowledge Contingent Payments [Max16, CGGN17].

Simplicity. To increase the expressiveness of our smart contracting language, we are developing a new language called **Simplicity**. Unlike a traditional imperative language that executes statements in order, Simplicity is a functional language that allows a declarative approach for specifying the conditions required by the smart contract. Expressive smart contract languages come with the added risk of programming errors. Worse is that errors cannot be corrected once they are committed to the blockchain. To address this risk, Simplicity has formal semantics, specified in the Coq proof assistant [CDT17], that allows smart contract developers to use formal methods to eliminate errors from their programs. The functional semantics of Simplicity is compatible with the logic used in Coq, allowing for simple equational reasoning, as opposed to the more difficult to use Hoare-style logic that is needed to reason about imperative languages. We intend Simplicity to become the primary language used in Liquid transactions.

Bulletproofs and General ZKPs. Like Taproot, Simplicity provides much better privacy than Bitcoin Script by hiding unused script semantics. However, it is possible in principle to do much better using a cryptographic primitive called a *zero-knowledge proof*. Such proofs can be used to hide entire scripts, revealing only that the script, whatever it was, was satisfied.

While current techniques for proving arbitrary scripts in zero-knowledge are too inefficient to replace explicit scripts entirely, there are several research directions which promise to bring us close to this goal. In particular, Bulletproofs [BBB+18] provide very small proofs of arbitrary computations and can very efficiently use secret inputs encoded in Pedersen commitments, such as Liquid outputs. Alternately, STARKs [BSBHR18] provide an alternative proof construction which can be constructed and verified orders of magnitude more quickly than Bulletproofs, though the resulting proofs are much larger.

These general zero-knowledge proof systems require programs be encoded in a special form called a *circuit*, which is "executed" by means of assigning values to inputs and outputs, and checking that these are consistent. This programming model is conceptually close to that of Simplicity, and while the tools for developing and analyzing program circuits are immature at best, we expect our research into similar tools for Simplicity will directly improve this situation. Further, the specific encoding of circuits used by the STARK system has many more similarities to Simplicity, which may be exploitable to efficiently map between the two systems.

References

[BBB+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, *Bulletproofs: Short proofs for confidential transactions and more*, IEEE Symposium on Security and Privacy, IEEE Computer Society, 2018, https://eprint.iacr.org/2017/1066, pp. 315–334.

[BCD+14] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, *Enabling blockchain innovations with pegged sidechains*, 2014, https://www.blockstream.com/sidechains.pdf.

[BFL15] BtcDrak, M. Friedenbach, and E. Lombrozo, *BIP112: CHECKSEQUENCEVERIFY*, Bitcoin Improvement Proposal, 2015, https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki.

[BSBHR18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, *Scalable, transparent, and post-quantum secure computational integrity*, Cryptology ePrint Archive, Report 2018/046, 2018, http://eprint.iacr.org/2018/046.

[CDT17] The Coq Development Team, *The Coq proof assistant reference manual, version 8.7*, October 2017, http://coq.inria.fr.

[CGGN17] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, *Zero-knowledge contingent payments revisited: Attacks and payments for services*, ACM SIGSAC Conference on Computer and Communications Security, Association for Computing Machinery, 2017, https://eprint.iacr.org/2017/566, p. 229–243.

[Cho17] A. Chow, *BIP174: Partially signed bitcoin transaction format*, Bitcoin Improvement Proposal, 2017, https://github.com/bitcoin/bips/blob/master/bip-0174.mediawiki.

[DPW+16] J. Dilley, A. Poelstra, J. Wilkins, M. Piekarska, B. Gorlick, and M. Friedenbach, *Strong federations: An interoperable blockchain solution to centralized third party risks*, 2016, http://arxiv.org/abs/1612.05491.

[DRO18] C. Decker, R. Russell, and O. Osuntokun, *eltoo: A simple layer2 protocol for bitcoin*, 2018, https://blockstream.com/eltoo.pdf.

[Gib17] A. Gibson, *Flipping the scriptless script on schnorr*, 2017, https://joinmarket.me/blog/blog/flipping-the-scriptless-script-on-schnorr/.

[Gri04] Ian Grigg, *The Ricardian contract*, First IEEE International Workshop on Electronic Contracting, IEEE Computer Society, 2004, https://iang.org/papers/ricardian_contract.html.

[Jed16] T.E. Jedusor, *Mimblewimble*, 2016, Defunct hidden service, http://5pdcbgndmprm4wud.onion/mimblewimble.txt. Reddit discussion at

https://www.reddit.com/r/Bitcoin/comments/4vub3y/mimblewimble_noninteractive_coinjoin_and_better/.

[LLW15] E. Lombrozo, J. Lau, and P. Wuille, *BIP141: Segregated witness (consensus layer)*, Bitcoin Improvement Proposal, 2015, https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki.

[LSP82] L. Lamport, R. Shostak, and M. Pease, *The byzantine generals problem*, ACM Trans. Program. Lang. Syst. **4** (1982), no. 3, 382–401.

[Max13] G. Maxwell, *CoinJoin: Bitcoin privacy for the real world*, 2013, BitcoinTalk post, https://bitcointalk.org/index.php?topic=279249.0.

[Max16] ———, *The first successful zero-knowledge contingent payment*, 2016, Blog post, https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/.

[MIGS16] M. Möser, E. Ittay, and E. Gün Sirer, *Bitcoin covenants*, Financial Cryptography and Data Security, Springer Berlin Heidelberg, 2016, https://fc16.ifca.ai/bitcoin/papers/MES16.pdf, pp. 126–141.

[MMSS+19] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, *Anonymous multi-hop locks for blockchain scalability and interoperability*, Network and Distributed System Security Symposium, The Internet Society, 2019, https://eprint.iacr.org/2018/472.

[MP15] G. Maxwell and A. Poelstra, *Borromean ring signatures*, http://diyhpl.us/~bryan/papers2/bitcoin/Borromean%20ring%20signatures.pdf, 2015.

[MPSW18] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, *Simple schnorr multi-signatures with applications to bitcoin*, Cryptology ePrint Archive, Report 2018/068, 2018, https://eprint.iacr.org/2018/068.

[MXC+16] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, *The honey badger of BFT protocols*, ACM SIGSAC Conference on Computer and Communications Security, Association for Computing Machinery, 2016, https://eprint.iacr.org/2016/199, p. 31–42.

[Nak09] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2009, https://www.bitcoin.org/bitcoin.pdf.

[PBF+19] A. Poelstra, A. Back, M. Friedenbach, G. Maxwell, and P. Wuille, *Confidential assets*, Bitcoin Workshop at Financial Cryptography and Data Security, Springer Berlin Heidelberg, 2019, https://blockstream.com/bitcoin17-final41.pdf, pp. 43–63.

[Poo16] J Poon, *SIGHASH_NOINPUT in segregated witness*, bitcoin-dev mailing list, 2016, https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2016-February/012460.html.

[SC17] P. Sztorc and CryptAxe, *Drivechain documentation – hashrate escrows bip*, 2017, https://github.com/drivechain-project/docs/blob/master/bip1-hashrate-escrow.md.

[vS13] N. van Saberhagen, *Cryptonote v 2.0*, https://cryptonote.org/whitepaper.pdf, 2013.