L2 Ethereum ZK Rollup for Private and Compliant Transactions

Calum Moore, Sidhant Gandhi

Feb 12, 2024 - Draft 0.6.0

1 Abstract

This paper describes the Payy Network, an Ethereum L2 zero knowledge rollup with built in privacy and compliance. We describe the protocol's interaction interface, network topology and security mechanisms, that provide privacy preserving and compliant UTXO transactions at scale.

2 Introduction

Blockchain technology has revolutionised the digital realm, enabling an era of decentralisation, security, and trust without intermediaries. These benefits have spurred many innovations across diverse sectors, enabling more transparent, immutable, and efficient transactions. Yet, for all their transformative advantages, blockchains have encountered a significant hurdle: privacy. This limitation has, in many ways, restrained the true potential of blockchains, especially in sectors where confidentiality is paramount, such as traditional finance, real world assets and payments.

Historically, achieving both transparency and privacy in blockchains appeared mutually exclusive. However, with the emergence of Zero-Knowledge (ZK) proofs, a form of cryptography for provable computation in zero knowledge domains, it has become feasible to ensure transaction privacy without compromising the integrity that blockchain platforms promise.

A side effect of enabling privacy, however, is that it complicates regulatory compliance. With public blockchains, the natural transparency enables regulators to screen all transactions for potential compliance breaches. With privacy, everything becomes more opaque and it becomes significantly more difficult to monitor transactions effectively. Yet this too, can be solved through zero knowledge, with clients generating relevant proofs that their activity has been compliant, so called "Proof of Innocence".

In addition, Layer 2 (L2) solutions have taken center stage in the quest to address

blockchain scalability, reducing the strain on the main chain and ensuring faster, more efficient operations. By combining the benefits of L2 with the privacy and compliance capabilities of ZK proofs, a new paradigm emerges.

In this paper, we present a full end to end solution for providing on chain privacy and compliance via an Ethereum L2 rollup, powered by zero knowledge.

3 Architecture

The Payy Network is an L2 Ethereum validium rollup with the following architecture:

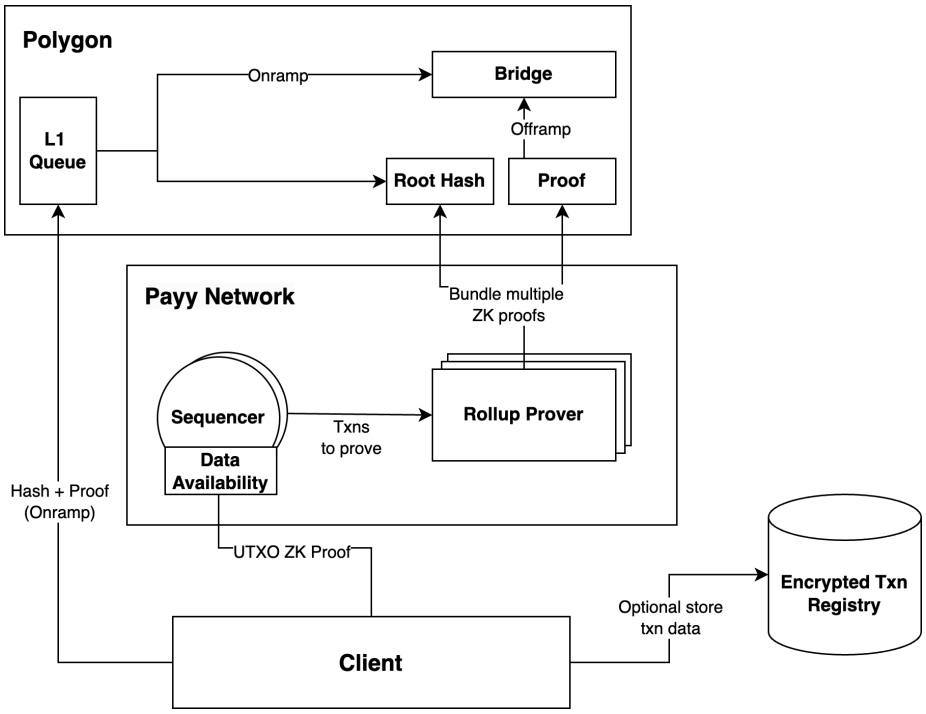

Figure 1: The Payy Network L2 architecture

3.1 Roles

Each node on the the Payy Network protocol can perform one or more of the following distinct roles:

- **Sequencer** - sequences transactions into a block, guaranteeing the order of transactions and providing data availability
- **Prover** - prove in zero knowledge that an existing sequenced block is valid or invalid and submit proof to the L1

- **Client** - proves valid UTXO transactions, so transaction data can remain private
- **Encrypted Transaction Registry (Optional)** - stores transactions so that transactions can be made with offline users
- **Ethereum** - provides security for the network

The following sequence diagram demonstrates the flow of actors in the network during a transfer of notes.

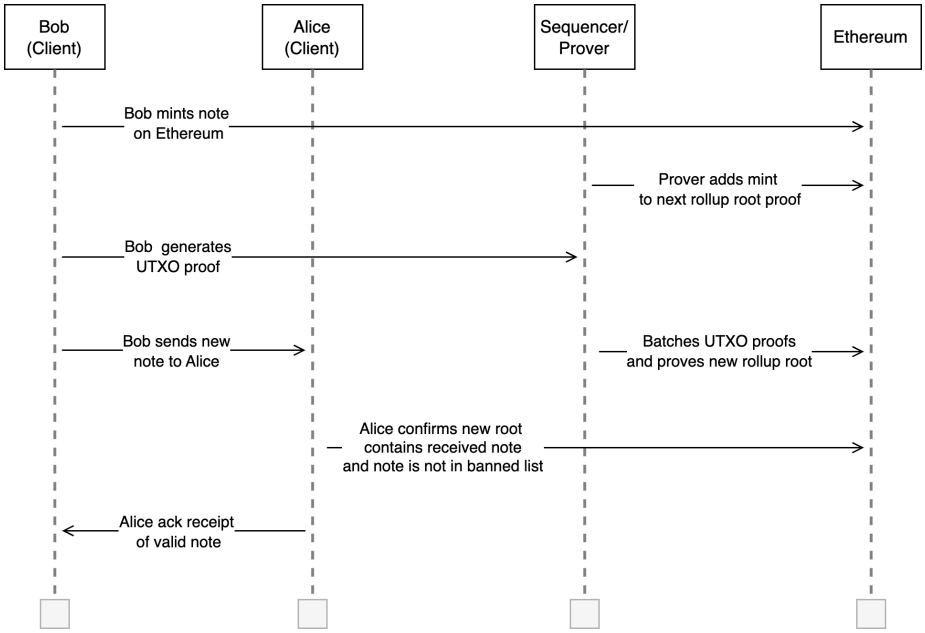

Figure 2: Sequence for transacting on the Payy Network

3.2 Sequencers

Sequencers are responsible for ordering transactions and validating transaction proofs. Sequencers form a Proof of Stake (PoS) network using the HotStuff consensus protocol [AGM18]. This ensures that the sequencers can operate in a decentralised and censorship resistent form, enabling it to provide soft finality and data availability to the network.

Sequencers only need to store blocks since the last rollup (i.e. the last proven block on Ethereum), significantly reducing the overhead of running a sequencer. This dramatically increases the decentralisation of the protocol, as many commodity devices can participate in sequencing.

3.2.1 Soft Finality

Sequencers provide fast (~1s) soft finality for transactions through the PoS consensus. This security is only required for the short period between a transaction being sequenced into a block and that block being rolled up by the prover. This period of time is dependent on the number of provers and other network configuration. As such, a smaller stake can be used to secure the soft finality.

It is recommended that clients use soft finality for smaller transaction sizes. The determination for using soft finality would be based on the amount of stake, divided by the proving interval. For larger transactions, it is recommended that clients wait for full Ethereum finality (i.e. when the prover has rolled up the root hash to Ethereum). However, the responsibility for clearly communicating risk profiles for transaction sizes is delegated to the client.

3.2.2 Data Availability

Availability of transaction data (UTXO proof and public input hashes) is essential, so all network participants can determine for themselves the full merkle tree (as the merkle tree algorithm is predictable). Without this, a prover could change the root hash on Ethereum without providing the txn data to other nodes, making it impossible for other nodes to know what the current state of the merkle tree is (only that it is valid). Without the current state of the merkle tree, no other nodes would be able to create a proof, as the merkle tree is used in proof generation.

3.3 Provers

Provers prove that the L2 sequenced blocks are valid or invalid. If the block is valid, the proof and root merkle hash of the rollup is updated in Ethereum. The Ethereum smart contract needs only the proof and the new root hash from the prover to the verify and ensure the security of the network.

3.3.1 Slot Allocation

It would be expensive and inefficient for all provers to submit proofs in parallel, as many of the proofs would be invalidated by a different recent proof included just before it. To prevent this, provers will be allocated slots which determine when they are eligible for submitting a sequencer of proof to Ethereum.

Ordering of slot allocation will based on a random deterministic ordering derived from a distributed randomness protocol, such as drand [DLJ20]. Nodes are ranked for each L2 block, the primary node has priority for the first 8 blocks, then the next in line has priority for the next 8 blocks, and so on. Nodes can submit a block before the transaction deadline. If a prover fails to submit the block in a timely manor, other nodes can submit the block proof instead.

3.3.2 Prover Rewards

To join as a prover, the prover is required to submit a small stake (e.g. 1ETH), designed to prevent nuisance behaviour. For example, provers could register but then fail to batch any transaction proofs on their allocated turn, or submit invalid proofs. There will be no limit to the number of sequencers that can join, improving the decentralisation of the network. Economic factors should limit the total number of provers to a reasonable level, as each additional prover reduces the overall gain for all other provers.

Provers are refunded for the cost of the Ethereum transaction to add the block, and also rewarded for their participation on the network.

3.3.3 Block Frequency

The maximum frequency that sequencers should add transactions to Ethereum is 1 epoch (~ 6 minutes), adding batches faster than this is redundant, as finality will not be improved due to reliance of the finality of Ethereum. The actual block frequency will be determined by the algorithm on the Ethereum smart contract, and will take into consideration the level of activity on the network, fees and other concerns.

3.4 Ethereum

Ethereum provides the settlement layer for the Payy Network. It also holds the smart contracts required to administrate the network, for example, adding/removing sequencers and provers, slashing rules, and the bridge contracts.

3.4.1 L1 Queue

The L1 Queue allows network participants to call other Ethereum smart contracts as part of any transaction that occurs on the Payy Network. Essentially, this allows for an atomic transactions across both Ethereum and the Payy Network. This is primarily used for the bridge functionality, so that funds can be locked on Ethereum via the bridge smart contract, whilst being simultaneously minted on the Payy Network, and vice versa. The L1 queue can also be used to better prevent censorship resistance, allowing any client to submit transactions directly and ensure they are included, albeit at a higher cost and slower transaction time.

L2 transactions pushed into the L1 Queue should be picked up by the next sequencer and included in the next batch/block. If no sequencer takes these transactions after a pre-determined period of time, then the transaction can be force included in its own block. This should be used only as an escape hatch.

3.4.2 Bridge

The bridge smart contract on Ethereum is responsible for bridging assets from Ethereum to the Payy Network, and vice versa. Given the Payy Network does not currently provide smart contract functionality, a specific transaction is used for bridging assets. When this transaction type is used, the proof circuit would modify its rules (e.g. to allow new funds to be minted or burned, outside of the normal balanced txn rules).

3.4.2.1 Onboard: From Ethereum to the Payy Network To use the bridge, a user would submit a single transaction to Ethereum directly, that both transfers the required funds to the Ethereum bridge contract, and adds the mint txn to the L1 Queue so the mint is propagated to the Payy Network. Before accepting a block from the prover, the smart contract would verify that the required funds were previously transferred.

3.4.2.2 Offboard: From the Payy Network to Ethereum Withdrawing an asset from the Payy Network, involves proving that a UTXO note was burned, and no output on the Payy Network was derived. This would allow the Ethereum smart contract to release funds (previously locked during the Onboard process). This unlocking would occur once the L2 block has been proven, as this is the point the transactions are proven to be valid.

3.4.3 Ethereum Reorgs

Finality on Ethereum takes 2 epochs and therefore within that time reorgs can occur. Reorgs could impact the the Payy Network sequencing and proving of transactions, but the worst case outcome would be that sequencers and provers would need to rework and submit their transactions. The state would never become invalid as a result of an Ethereum reorg.

3.5 Encrypted Transaction Registry

The encrypted transaction registry is an optional protocol component that allows transactions to be sent to the receiver, while the receiver is offline. These transactions are stored encrypted with the receiver's public key, ensuring only the receiver can decrypt the data. When a client joins the network, they scan the registry for new transactions that have been sent to them.

4 Rollup

The rollup represents the entire state of the network, where each piece of state is referred to as a note, is represented by a commitment hash [Merkle87]. Hashes result in data loss, so the original messages can provably never be reconstructed, improving the privacy of the protocol.

Each of the hash record states are stored in a merkle tree, so a single root hash can represent the entire state of the network. No underlying data is stored in the rollup, and therefore any required shared data must be stored in the Encrypted Transaction Registry. In addition to privacy, an additional advantage of storing only hashes in the rollup is that the rollup on disk data size can be somewhat constrained and deterministic. This is a result of all hashes being a consistent size regardless of the size of the underlying data. This reduction in the disk size requirements for clients, improves the number of clients able to join the network, thereby improving decentralisation.

The merkle tree must enable the following operations:

- Prove inclusion - prove a hash exists in the tree
- Prove non-inclusion - prove a hash does not exist in the tree
- Insert - insert a new hash into the tree if it does not already exist

To satisfy these properties, the Payy Network uses a sparse merkle tree [DPP16]. A sparse merkle tree has a defined position for every possible value that can be inserted into the tree. In order to support a 256 bit (32 byte) hash, a tree of size 2^{256} is required. This means that for every operation, 256 tree nodes need to be traversed and validated. This may results in an unacceptable performance.

There are two possible optimisations to improve the performance:

1. A smaller tree could be used, such as 2^{128}. This would result in a higher likelihood, but possibly still acceptable, risk of hash collision. For a tree of size 2^{128} it would take ~8 million years, generating 10,000 hashes per second to have a 1% chance of collision.
2. The tree can be sharded at multiple levels, allowing these verifications to occur in parallel

4.0.1 Merkle Proofs

As every hash has a unique position in a sparse merkle tree, we can derive its position by decomposing each bit of the hash and traversing the tree based on whether the bit is 0 or 1. For 0, the tree is traversed to the left child, and for 1 the tree is traversed to the right child.

For an inclusion or non-inclusion proof, we can simply check that each of the provided siblings combined with the the computed child results in the root hash. For non-inclusion we are proving that the leaf node is a null value, which does not need to be passed to the proof, as it is a static value.

Insertion proofs are a combination of a non-inclusion proof (existing position must be null) and inclusion proof (proving the new root based on the inserted hash).

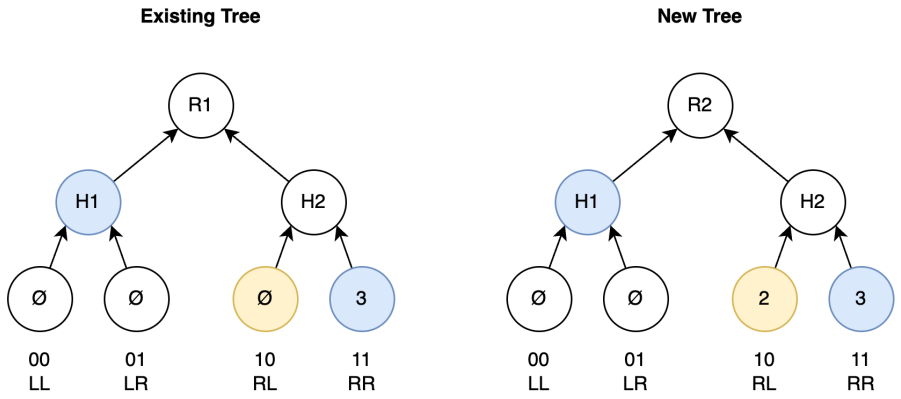

Figure 3: 4 bit tree, demonstrating bit decomposition and insertion

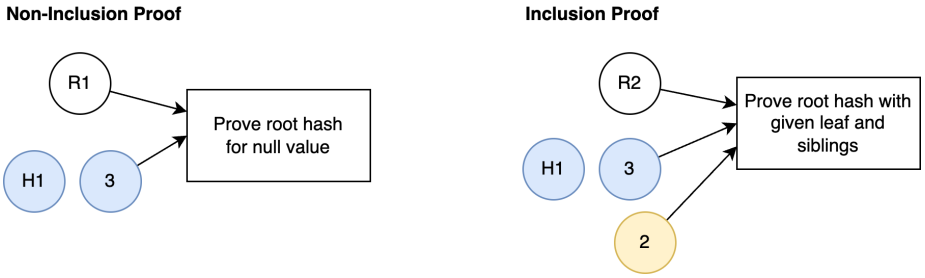

Figure 4: Inclusion and non-inclusion proof

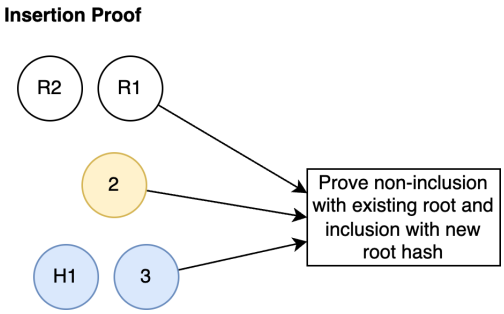

Figure 5: Inserting into merkle tree

5 UTXO

UTXO (Unspent Transaction Output) is a state model introduced by Bitcoin [Nak08] and additionally used in Zcash [GMRA13], and other blockchains to store the balances that can be spent. In this model, each store of value is a note that belongs to a specific authenticated account. Notes can be of any value, but can only be used once. If the note contains a value larger than you wish to transfer, you can create a new note with the output. You can use multiple input notes, so long as the total value of incoming notes is the same (or less) than the total value of outgoing notes.

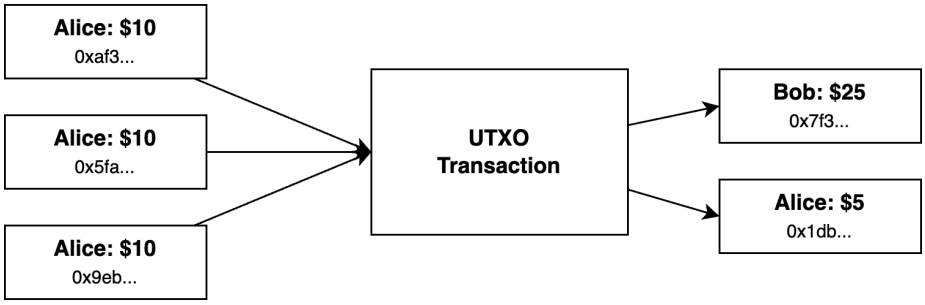

Figure 6: UTXO

This model has a number of advantages:

- **Scalability** - because a user can have multiple notes, multiple payments can be included in the same block (even in the case where the underlying data is hidden)
- **Privacy** - each transaction creates a new output record/hash. This prevents adversaries from looking for addresses with high activity or attempting to match accounts based on the source IP of the transaction, as is possible in account based models like Ethereum.

The considered drawbacks for the UTXO model are:

- **Increased tree depth** - UTXO transactions create one new insert hash for every incoming and outgoing note in a transaction. In contrast, an account based model (such as used in Ethereum) uses only a single hash insert per account, reducing the size of the tree.

Due to the strict privacy constraints desired for the Payy Network, the UTXO model was selected.

5.1 Notes

Each note has the following state properties:

- **Application Type** - used to differentiate between different applications

running on the Payy Network (each application type would have a different set of ZK circuit constraints)
- **Authentication** - used to verify which actor is allowed to "spend" a note
- **Value** - the balance of the note, balances between incoming and outgoing notes must match
- **Nullifier entropy** - additional nullifier entropy to increase security and privacy

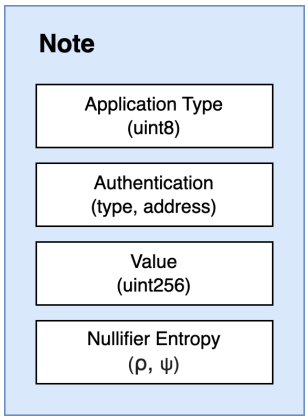

Figure 7: UTXO note structure

5.2 Authentication

Authentication identifies the user that is allowed to spend a note. The auth commitment is used in the Nullifer constraint to ensure that the user generating the nullifier (i.e. spending the note), knows the underlying secret key represented by the commitment. the Payy Network supports multiple authentication systems, denoted by a type enum, to provide flexible ownership over notes. Additional authentication types may be added over time.

5.2.1 Poseidon

Users can generate a random 32-byte secret key, and use the poseidon commitment of the secret key as the authentication address for a note. This provides a highly performant authentication mechanism, but may not be well supported by existing wallets or tooling.

5.2.2 Ethereum

The Payy Network natively supports Ethereum addresses. In this case, the auth commitment would be the Ethereum address. To spend with the Ethereum address a proof must be generated that proves ownership of the underlying

private key. This can be achieved in zero knowledge by passing the Ethereum private key as a private input and and deriving the address from the private key.

5.3 Minting and Burning

Special transaction types will allow the UTXO model to be modified in order to allow bridging of assets to and from the Payy Network:

- **Mint** - allows a note to be created without a corresponding input, minting will be further validated on Ethereum in the data availability layer before it will be accepted as a valid transaction
- **Burn** - allows a note to be used without a corresponding output, when this is proved on Ethereum, the funds will be released from the bridge contract

5.4 Nullifier

As per the UTXO model, a note can only be used once. As such, the protocol must keep track of which notes have been used or spent, and which have not. Naively, this could be performed by removing used notes from the merkle tree, however this would reveal to a sender when a note they have sent to another user has been spent, reducing privacy.

Instead, to enhance privacy, we can use a deterministic nullifier record that for each spend must be inserted into the tree, and proven not to exist in the tree via a non-inclusion proof. The presence of the nullifier represents the record being spent. The nullifier is constructed in such a way that it is impossible for an external party to determine which nullifier matches a spent record (or a record to be spent).

The nullifier is calculated as follows:

$$N = \text{Poseidon}(nk, \psi, cm)$$

Where:

- nk is the Nullifier Key, a unique secret associated with each user (this is a auth commitment to the auth)
- ψ (psi) sender controlled randomness, additional entropy provided by Blake2b [Aumasson et al. 2013] hash. Blake2b provides additional entropy and privacy security.
- cm is the note commitment, which is a Poseidon commitment to the note.

6 Compliance

Public blockchains allow regulators to easily verify on chain activity is compliant with regulations. As the Payy Network is privacy preserving, compliance

capabilities need to be defined at the protocol level to ensure the network is not useful to those seeking to perform illicit activity.

There are a number of different mechanisms for enforcing compliance on the network:

1. Transaction lineage - tracking of full or partial transaction lineage, so that illicit funds can be tracked and disabled across the network. Transaction details are still private.
2. Privacy pools - bundling transactions and actors into pools of good and or bad actors and treating them as a single entity
3. Compliance ZK proofs - enable us to hide information while proving specific compliance constraints

These techniques could be used independently or in combination to ensure the required level of compliance. In addition, these capabilities could be applied at either onramp/offramp or for each individual transaction depending on the regulatory or user requirements.

6.1 Transaction lineage

Transaction lineage allows the network to trace the source of funds (but not individual transaction details) across the network. This can generally be split into two categories:

1. **Source lineage** - source tracking tracks only the source of each transaction as it enters and exists the network. This enables onramp and offramp providers to reject transactions that are from external sources which are later found to be illicit.
2. **Full lineage** - all transaction lineage can be tracked. This reduces the level of privacy, as any new holder of the note obtains a significant portion of lineage, and could reveal this publicly without consent of the other parties.

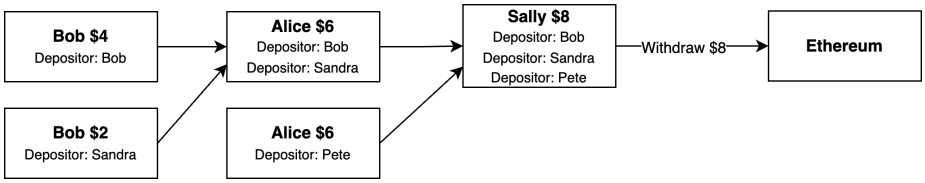

Figure 8: L2 Rollup-Privacy Pools.drawio (31).png

The transaction lineage would be added to each note, so any receiver of a note would obtain the transaction lineage, but outside parties would not. In order to avoid exponentially increasing note size, the lineage of transactions would only be maintained for a specified time period (i.e. number of blocks, where block time is constant), for example 1 year.

6.2 Privacy Pools

Privacy pools [BJMFA23] is a mechanism that group users into pools that share the same level of regulatory and compliance risk. Although the focus of the privacy pools paper is based around the use of privacy pools in so called "mixers", it does discuss the implications of extending this to a system where intermediate transfers are possible.

Extending this to such a system requires users to accept the risk of loss of funds if a particular pool they are part of is deemed to be non-compliant. It would therefore be essential that users choose a pool which matches their risk appetite.

There are a number of different types of pools that could used, such as:

- Add with delay, exclude bad actors
- KYC pool
- N per month per person
- N per month per trusted community member
- Real-time AI-based scoring

Transactions between pools would be allowed according to the inbound/outbound rules of both pools.

6.3 Compliance ZK-Proofs

ZK proofs enable us to prove certain attributes of a transaction or chain of transactions in a privacy preserving environment. These proofs could be used as a layer on top of privacy pools or transaction lineage to provide additional privacy while maintaining the same level of compliance.

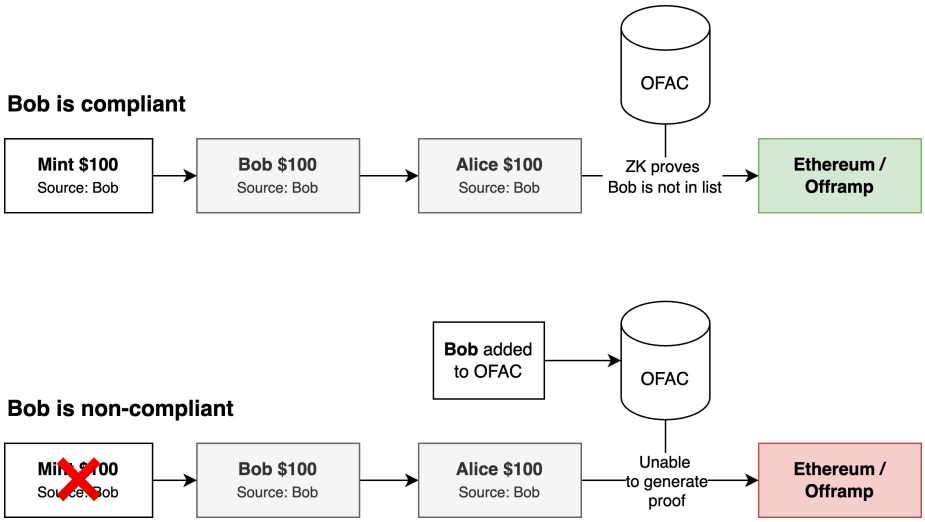

Figure 9: Compliance ZK proof

In addition, ZK could be used to prove/enforce additional constraints. Offramp providers could even enforce their own additional ZK constraints. These additional constraints can be applied by the offramp in a permission-less manner outside of the protocol, as the proofs would be generated locally on the client and submitted in addition to the protocol proof to the offramp.

7 ZK Proofs

There are three ZK circuits required for the the Payy Network ZK rollup:

1. **UTXO proof (client)** - runs on the client and proves that a user has permission to spend an input note and generate an output note
2. **Insert proof (prover)** - calculates the new root given a number of UTXO proofs
3. **Aggregation proof (prover)** - combines multiple UTXO proofs and the insert proof into a single proof to be verified on Ethereum

In addition, a compliance proof can optionally be generated at any time to prove compliance of given notes:

1. **Compliance proof (client)** - allows the client to prove that the source of their funds is not from a blacklisted address

The following diagram represents the high level of flow of input constrains to the zero knowledge circuits.

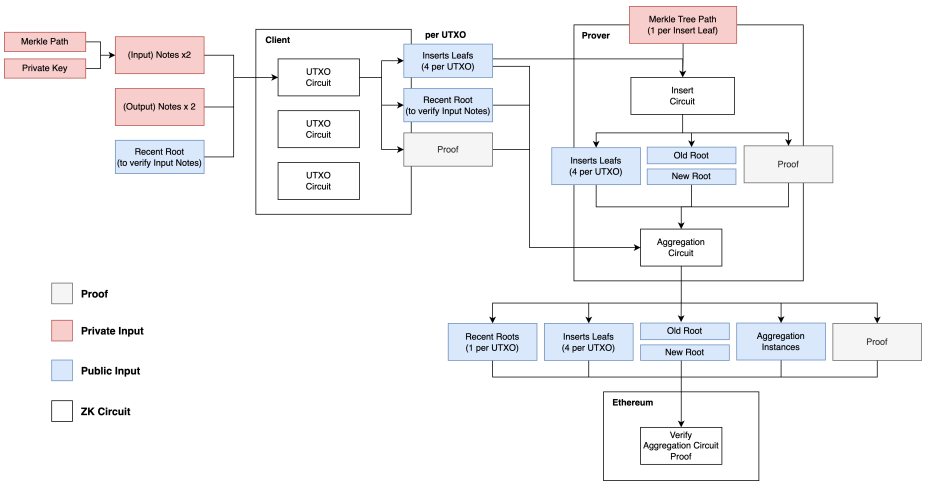

Figure 10: ZK process flow

7.1 Proof Algorithm

There has been rapid improvements in zk-SNARK/zk-STARK algorithms in recent years. Groth-16 [Gro16] has been traditionally the benchmark, which is used in large scale decentralised protocols, such as Filecoin [BPZZ17]. However, this algorithm relies on a trusted setup procedure which makes updating the protocol difficult, and prevents the use of user generated functions (i.e. smart contracts), so it will not be used in the Payy Network.

Instead, the Payy Network uses Halo2 circuits with a HyperPlonk [CBZZ22] arithmetisation scheme and KZG [KZG10] as the polynomial commitment scheme. In addition to the performance characteristics, Halo2 was selected for its high security guarantees having been audited and used in production for numerous projects such as Zcash [Hopwood et al. 2022] and Scroll.

The KZG commitment scheme was selected over the IPA [BBBF18] commitment scheme, for the improved performance, especially in relation to aggregation proofs. Further enhancements to the proving system are anticipated. For example, improved recursion techniques, such as ProtoStar [BC23], could likely yield significant performance increases.

The BN256 pairing friendly elliptic curve [BN06] is used over the pasta curves traditionally used with Halo2, due for the need of verifying the proofs on Ethereum.

7.2 Performance

Performance is a key consideration for zero knowledge circuits, in particular the prover cost is significant compared to running computation in a native turing machine. The following outlines the expected performance characteristics of each proof.

Circuit	Browser	Desktop/Mobile	Optimised Server
UTXO Proof	9s	3s	-
Insert Proof	N/A	N/A	< 1s
Aggregation Proof	N/A	N/A	40s
Compliance Proof	1s	0.3s	-

7.3 Enhanced throughput with multi-level aggregations

To increase the throughput of the network, multiple levels of aggregation can be applied, which can then recursively combine multiple proofs into a single proof. Each additional level of proof increases the overall throughput of the network, at the expense of additional latency.

Aggregators	Layers	TPS	Latency
1	1	10	10s
11 (10 + 1)	2	100	20s
111 (100 + 10 + 1)	3	1,000	30s
1111 (1000 + 100 + 10 + 1)	4	10,000	40s

8 Transmitting Notes

In order to spend a note you must have the data that represents the note being spent, as generating the spend UTXO proof requires the notes data; and only the hash commitment of the note is stored in the rollup. The sender of the note, is responsible for communicating the the new notes data to the receiver of the note. The sender could decide not send or record the output note, but then the receiver would not acknowledge receipt of note (i.e. from the senders perspective, they have not received any notes) and therefore it would be as if the transfer did not complete.

There are two mechanisms to allow senders to communicate the note data to the recipient.

- P2P - increased privacy, but requires both actors to be online
- Encrypted Registry - reduced privacy if encryption key is lost or exploited

8.1 P2P

If both peers are online, the sender can send the transaction hashes and proofs to the sequencer, and the underlying data directly to the receiver. This method approach enhances privacy, as the underlying data is never stored in the public domain.

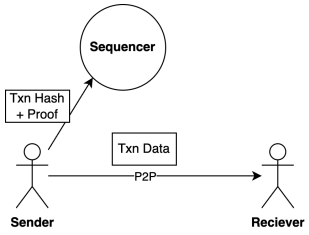

Figure 11: Transmitting notes with P2P

8.2 Encrypted Registry

If the Sender and Receiver are not online at the same time, the transaction data can be stored in the Encrypted Registry. The Encrypted Registry is optional,

and not a core part of the Payy Network protocol. It is provided purely as a convenience feature to store shared user transaction data.

To use the Encrypted Registry, the Sender uses the Receiver's PublicKey to encrypt the data, so only the Receiver can decrypt it. When the Receiver joins the network, they can scan the Encrypted Registry attempting to decrypt all new records since they were last online. If they are able to decrypt a transaction, then it is a transaction they have been sent.

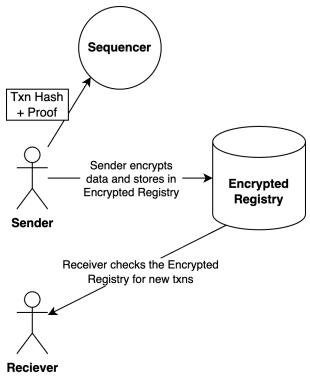

Figure 12: Transmitting notes with encrypted registry store

Users of the Encrypted Registry should be aware that using it reduces the level of privacy protection. If the data is encrypted in a public registry there are two main risks:

1. Exposed secret key - secret key used for encryption is compromised at any point in time
2. Compromised encryption protocol - underlying encryption protocol is broken by advances in cryptography or compute

As the Encrypted Registry is an optional feature, other providers could offer alternative encrypted registries to store transactions. In future, we may provide a sharded decentralised encrypted registry, where the encrypted data is split into chunks and blindly stored across multiple nodes, reducing the exposure to the risks described above.

9 Conclusion

In this paper, we presented the requirements and design for a privacy preserving Ethereum L2 zero knowledge rollup with built in privacy and compliance. We presented the underlying zero knowledge proof algorithms that enable secure and privacy preserving off-chain transactions, as well as the mechanism for transmitting and sending notes. For the first time, the Payy Network protocol

provides the necessary infrastructure to private financial transactions that are privacy preserving yet compliant.

10 References

[AGM18] Abraham, I., Gueta, G., Malkhi, D. 2018. Hot-Stuff the Linear, Optimal-Resilience, One-Message BFT Devil. CoRR, abs/1803.05069. Accessed on: http://arxiv.org/abs/1803.05069

[Aumasson et al. 2013] Aumasson, J-P., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. 2013. BLAKE2: simpler, smaller, fast as MD5.

[BBBF18] Bünz, B., Bootle, J., Boneh, D., Fish, B. Fisch and Poelstra, A. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More, In IEEE Symposium on Security and Privacy (SP), 319-338. IEEE. https://doi.org/10.1109/SP.2018.00020

[BC23] Bünz, B., Chen, B. 2023. ProtoStar: Generic Efficient Accumulation/Folding for Special Sound Protocols. Cryptology ePrint Archive, Paper 2023/620. Accessed on: https://eprint.iacr.org/2023/620

[BINSSA23] Buterin, V., Illum, J., Nadler M, Schär, Fabian., and Soleimani, Ameen. 2023. *Blockchain Privacy and Regulatory Compliance: Towards a Practical Equilibrium.*

[BN06] Barreto, P. and Naehrig, M., 2006. *Pairing-friendly elliptic curves of prime order.* In: Preneel, B., Tavares, S. (eds) Selected Areas in Cryptography. SAC 2005. Lecture Notes in Computer Science (vol 3897). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11693383_8

[BPZZ17] Benet, J., Pontarelli, N., Zigdon, Y., and Zuckerman, E. 2017. *Filecoin: A Decentralized Storage Network.* https://filecoin.io/filecoin.pdf

[CBZZ22] Chen, B., Bünz, B., Boneh, D., and Zhang, Z. 2022. *HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates.* Cryptology ePrint Archive, Paper 2022/1355. Accessed on: https://eprint.iacr.org/2022/1355

[DLJ20] Deco, N., Jovanovic, P., & Barman, L. 2020. *Drand: Verifiable Distributed Randomness.* https://drand.love/whitepaper.pdf.

[DPP16] Dahlberg, R., Pulls, T., and Peeters, R. 2016. *Efficient Sparse Merkle Trees.* In: Brumley, B., Röning, J. (eds) Secure IT Systems. NordSec 2016. Lecture Notes in Computer Science (vol 10014). Springer, Cham. https://doi.org/10.1007/978-3-319-47560-8_13

[Gro16] Groth, J. 2016. *On the Size of Pairing-Based Non-interactive Arguments.* Advances in Cryptology – EUROCRYPT 2016

[GMRA13] Green, M., Miers, I., Rubin, A.D. and Garman, C. 2013. *Zerocoin: Anonymous Distributed E-Cash from Bitcoin.*

[Hopwood et al. 2022] Hopwood, D., Bowe, S., Hornby, T., & Wilcox, N. (2022). *Zcash Protocol Specification.* https://zips.z.cash/protocol/protocol.pdf

[KZG10] Kate, A., Zaverucha, G., and Goldberg, I. 2010. *Constant-Size Commitments to Polynomials and Their Applications.* In: Abe, M. (ed) Advances in Cryptology – ASIACRYPT 2010. Lecture Notes in Computer Science (vol 6477). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17373-8_17

[Merkle87] Merkle, R. 1987. *A digital signature based on a conventional encryption function." Advances in Cryptology* — CRYPTO '87. Springer.

[Nak08] Nakamoto, S. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System.