Nomos Privacy Whitepaper (v0.1)

Álvaro Castro-Castilla
Nomos

April 2023

1 Introduction

At the core of blockchains is the principle of transparency, which allows participants to view and verify transactions within the network. While transparency is one of the key strengths of blockchain technology, it also raises important privacy concerns that must be addressed to ensure widespread adoption and trust.

Privacy is a fundamental human right, and the protection of personal information has become increasingly important in today's digital age. As blockchains are being used for various applications, maintaining privacy becomes crucial to safeguard sensitive information, protect users from malicious activities, and maintain regulatory compliance. Privacy has two equally important but distinct aspects:

1. **Confidentiality:** Transactions on a blockchain can involve sensitive information, such as financial transactions, medical records, or personal identification data. Ensuring confidentiality is critical to protect users' privacy and prevent unauthorized access to this information. Confidentiality refers to the hiding of the action being conducted (ie. the *what* of an on-chain action).

2. **Anonymity:** The ability to transact anonymously is essential for preserving individual privacy rights. While some blockchain networks, such as Bitcoin, offer a level of pseudonymity, the public nature of their transaction history can still lead to the identification of users. Adequate privacy measures can provide stronger anonymity and protect users from potential surveillance or targeted attacks. Anonymity refers to the hiding of the identity conducting an action (ie. the *who* of an on-chain action).

1.1 A blockchain for the Network State

The importance of anonymity and confidentiality in a blockchain designed for the Network State is paramount. A Network State is a concept that envisions a decentralized, digital, and self-governing system where citizens, organizations, and institutions interact within a distributed network. In such a system, ensuring anonymity and confidentiality plays a crucial role for various reasons:

1. **Individual Privacy:** In a Network State, protecting the privacy rights of individuals is of utmost importance. Users should have the ability to control their personal information and maintain anonymity, allowing them to participate in the network without fear of surveillance, profiling, or discrimination.

2. **Confidential Transactions:** To encourage broader adoption and trust in a blockchain-based Network State, it is essential to ensure the confidentiality of transactions. By concealing transaction details such as sender, receiver, and transaction amounts, users can securely exchange assets, conduct business, or engage in other activities without revealing sensitive financial information.

3. **Secure Communication:** A Network State relies heavily on communication between its participants. Ensuring the confidentiality of messages exchanged between users is vital to maintain privacy, protect sensitive information, and prevent unauthorized access or manipulation.

4. **Political Freedom:** Anonymity and confidentiality play a key role in preserving political freedom within the Network State. By enabling anonymous participation in decision-making processes, such as voting or policy proposals, the blockchain ensures that individuals can freely express their opinions without fear of influence from external forces. Furthermore, mechanisms to limit power accumulation can be implemented.

5. **Censorship Resistance:** Ensuring anonymity and confidentiality in a blockchain-based Network State helps maintain its censorship-resistant nature. By preventing the disclosure of user identities and data, the system can resist attempts by centralized authorities to censor or control the flow of information.

1.2 Privacy requirements for Nomos

A blockchain can have privacy concerns at various layers, each with its unique set of challenges. In this document we will elaborate the different approaches employed to ensure privacy at the various components of the system.

1. **Network Layer:** At the network layer, privacy concerns arise from the exposure of users' IP addresses, which can be used to link transactions to specific users. Membership of the p2p network is also ideally concealed under a larger anonymity set to avoid identification and targeted attacks to any node simply for its participation in the protocol.

2. **Mempool Layer:** Privacy concerns at the mempool layer involve the exposure of transparent transaction data, such as sender, receiver, amounts, or contract call data. Participation in the mempool (identity and dissemination) are also to be considered from a privacy perspective.

3. **Consensus Layer:** We want to protect the privacy of the participants in the consensus mechanism (validators or other roles). This includes both concealing the role itself, as well as the actions they perform and the opinions or decisions they take as part of the distributed protocol.

4. **Execution Layer:** Privacy concerns at the application layer arise from the potential leakage of sensitive information through interactions with decentralized applications (dApps) or smart contracts. This layer cannot be completely solved at the protocol level (since any developer can design an application incorrectly, or an user can perform actions that actively leak information that is meant to be private). In this layer, Nomos attempts to provide good defaults and a flexible (and ideally extensible) set of privacy-preserving technologies to build apps with the highest degree of control and customizable privacy.

5. **Inter-chain communication:** Transferring assets in and out of the blockchain can reveal patterns that undermine privacy. By analyzing this transaction history in sender and receiver blockchains, it may be possible to identify patterns of behavior that can reveal sensitive information about the owner of the address. Other information leakages, like time of the performed actions can help statistical analysis to extract patterns about the Nomos blockchain usage.

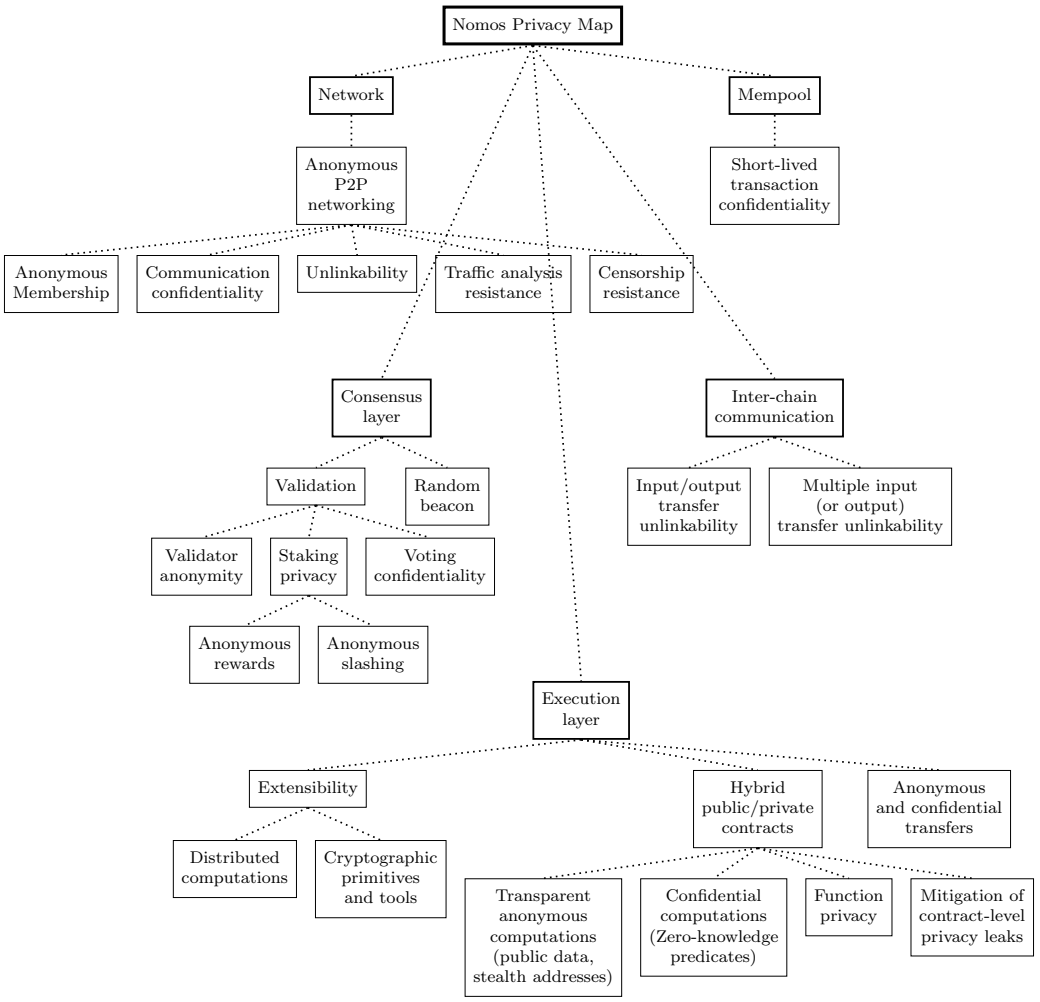

Figure 1: Map of privacy requirements for Nomos

1.2.1 Network layer privacy

Privacy at the network level refers to the protection of user data and communication as it traverses the network infrastructure, concealing the identity of the users, the content of their communication, and their interactions with other nodes or services. This type of privacy aims to prevent eavesdropping, traffic analysis, and other forms of surveillance by unauthorized parties or malicious actors. It involves several aspects:

- **Anonymity:** Protecting the identities of the users participating in the network. This can be achieved by using anonymizing network protocols, such as Waku or Nym, that route data through multiple nodes to obscure the origin and destination of the communication. For anonymity to be achieved,

the network is disguised within a larger set of completely unrelated networks (the anonymity set), which are effectively indistinguishable even for the network participants.

- **Communication confidentiality:** Ensuring that the content of the communication, such as transaction details or messages exchanged between nodes, remains private and cannot be intercepted or read by unauthorized parties. End-to-end encryption and secure communication protocols, can be used to maintain confidentiality.
- **Unlinkability:** Preventing the ability to correlate different transactions, messages, or interactions involving the same user, making it difficult for an attacker to build a profile of the user's activities on the network, eventually deanonymizing an individual or a group of them. Mix networks, onion routing, or zero-knowledge proofs can be employed to make this harder or impossible for an adversary to do.
- **Traffic analysis resistance:** Protecting against the analysis of communication patterns, such as the frequency, timing, or size of data packets, which can reveal information about users even if the content is encrypted. Network-level privacy mechanisms like constant-rate communication, traffic obfuscation, cover traffic, or dummy traffic can be used to mitigate traffic analysis risks.
- **Resistance to censorship:** Ensuring that users can access and participate in the network without being blocked, filtered, or restricted by governments, ISPs, or other entities. Decentralized networks, domain fronting, and alternative routing methods can help bypass censorship efforts and maintain network-level privacy.

References: Waku, Nym

1.2.2 Mempool layer privacy

A mempool refers to a collection of unconfirmed transactions that are not yet publicly visible on the blockchain network. These transactions are generally transparent, as they need to be validated and ordered by the validator. Building a *private mempool* can help increasing the censorship resistance of the blockchain, and additionally, act as a powerful MEV protection mechanism. However, it is important to note that a private mempool is not a long-term privacy-preservation mechanism, since the transaction data will be exposed as it is committed on-chain. This is the reason why its effect is limited to censorship resistance and

MEV protection, which benefit from this short-lived confidentiality. Conversely, the transaction data that is encrypted doesn't benefit from mempool privacy.

The fundamental concept involves enabling users to submit transactions that are encrypted. To execute an encrypted transaction, a user first encrypts the transaction and broadcasts it to the network. The encrypted transaction is then committed to by the network. The transaction is later decrypted, and the resulting plaintext is used to execute the transaction. In other words, To achieve this, block producers commit to these transactions before decrypting them. The commit, decrypt, and execute phases can potentially be performed in a single step, depending on the network's design, to preserve user experience. There are different approaches to implementing this:

- **Enclaves**. One approach is to use trusted hardware to process plaintext data. The trusted hardware can operate on the data without decrypting it publicly until after the transactions have been committed to the blockchain.

- **Threshold Encryption/Decryption (TE)** is a method that involves a committee of signers who are jointly responsible for decrypting ciphertext data. A minimum number of signers, or *threshold*, are required to decrypt the data. For example, a threshold of 2/3 of the validator set may be required to agree to decrypt the data.

  The downsides of threshold encryption/decryption (TE) include lack of accountability, a strong honest majority assumption, destabilization of the protocol, and weakened liveness. TED can lead to malicious behavior without attribution, and external agents outside the validator set could influence the process. TED also relies on a truly honest and altruistic majority, which increases incentives for centralization and malicious behavior. A higher decryption threshold improves safety but weakens liveness, making it vulnerable to massive network outages that can halt the protocol.

- **DED (Delay Encryption/Decryption)** is a method where encrypted information is automatically decrypted after a certain amount of time, and it requires VDF (Verifiable Delay Function) ASICs for practical implementation. Ethereum Foundation and Protocol Labs have been working on producing VDF ASICs, which are capable of fast sequential computation and also work for time-lock puzzles and DED. However, producing SNARKs (Succinct Non-Interactive Argument of Knowledge) for VDFs also require specialized ASICs. Compared to TED, DED is preferred because it doesn't require trusting a committee, but specialized hardware and delays may add latency to the system.

References: Shutterized Beacon Chain, Chainlink FSS, Ferveo, The Ethereum Foundation and Protocol Labs references

1.2.3 Consensus layer privacy

In PoS systems, validator anonymity, voting privacy, and stake privacy are critical aspects of the network privacy preservation:

- **Validator Anonymity:** Maintaining validator anonymity is crucial in PoS systems to prevent targeted attacks and protect validators' real-world identities. There are two aspects to consider for this:

  1. Network-level anonymity: Anonymous networks provide an additional layer of privacy and security by routing validator traffic through a series of intermediate nodes. This makes it difficult for an adversary to trace the origin of the messages back to the validator node.
  2. Secret Validator Registration: Using zero-knowledge proofs or group signatures, validators can register their nodes in a privacy-preserving manner, proving their eligibility without revealing their actual stake or identity.

- **Voting Privacy:** In PoS systems, validators often participate in voting processes to reach consensus or as part of some secondary protocol (such as the Random Beacon). Ensuring privacy in these voting processes is essential to avoid revealing validators' preferences. One example technique to achieve this is the use of Somewhat Homomorphic Encryption (SHE). This cryptographic method allows validators to encrypt their votes such that they can be combined and tallied without decrypting the individual votes, maintaining vote privacy while still enabling accurate vote aggregation.

- **Stake Privacy:** In PoS systems, preserving stake privacy is vital to avoid exposing users' wealth. Different approaches to achieve this include leveraging confidential assets, such as the Sapling scheme, or coin mixing protocols applied to staking.

TODO: Elaborate further. References: Sapling

1.2.4 Inter-chain communication privacy

There are several risks associated with analyzing the flow of transfers between interconnected blockchains. Linking transfers in and out of a private blockchain can lead to various levels of account de-anonymization. The aim is to provide protection against:

- **Correlating input and output transfers.** When considering the correlation of input/output (I/O) transfers between two privacy-preserving blockchains, the risk of exposing sensitive information is mitigated but not entirely eliminated.

- **Correlating multiple input (or output) transfers of the same account**. Similar approaches to those employed for correlating inputs with outputs can potentially be used also for multiple inputs.

References. Analysis of Address Linkability in Tornado Cash on Ethereum

These problems are akin to the privacy leakages incurred when transferring from shielded to unshielded environments in blockchains that support both types internally, as the transfer between different (sub)-systems is performed without the same privacy-preservation measures as the shielded system. As a consequence, adversaries have the possibility of analyzing the transaction graph to uncover patterns, relationships, and flows of assets, potentially leading to the identification of high-value targets or sensitive business operations.

- **Timing Analysis:** By analyzing the timing of transfers into and out of the private blockchain, adversaries can establish correlations between transactions. If the timing of transfers is distinctive or if there are only a few participants, the adversary can infer relationships between transactions and potentially reveal user identities.

- **Amount Correlation:** If the amounts involved in transfers into and out of the private blockchain are unique or follow a pattern, adversaries can correlate these amounts to link transactions. This can lead to user deanonymization and expose the transaction graph.

- **Intersection Attacks:** If a user participates in multiple private blockchains or engages in off-chain transactions, an adversary can perform intersection attacks by analyzing the common elements in the different systems based on transaction patterns. This can enable the adversary to link transfers and compromise user privacy.

- **Side-Channel Information:** Adversaries can leverage side-channel information, such as IP addresses, metadata, or network traffic patterns, to establish connections between transfers in and out of the private blockchain. By correlating this side-channel information with transaction data, adversaries can potentially uncover relationships between transfers and users.

Some of the countermeasures that can be taken to protect against these risks include timing and amount randomization. Introducing random delays in transaction processing or using decoy transactions with varying amounts can help mitigate timing and amount correlation attacks. Additionally, the techniques employed to provide execution-layer privacy help mitigating these risks during inter-chain communications.

1.2.5 Execution layer privacy

The execution layer brings additional complexity to the pursuit of privacy on blockchains. When the set of operations is limited (such as in cryptocurrency-only blockchains), it is possible to design the system in such a way that privacy leaks are completely eliminated, or at least their extent is well-known. In the case of quasi-Turing complete execution environments, the space of possibilities is much larger, leading to many potentially overseen privacy leakages entirely caused by the application designs. This is particularly true when we intend to build a partial privacy solution (required for many real world use cases, such as decentralized exchanges), where certain information must be made public in order for applications to become useful.

Consequently, the understanding of requirements for the execution layer is much more nuanced and is highly dependant of the cryptographic tools at our disposal. We will establish here the main requirements to satisfy in the design of the protocol, in the ideal case.

- **Enabling Transaction Confidentiality and Anonymity:** To achieve transaction confidentiality, privacy solutions should effectively obfuscate transaction details, such as sender, receiver, and transaction amounts. Generally speaking, there are two main ways in which this can be done:

  (a) *Encrypting and obfuscating transactions on-chain.* All the information pertaining a transaction can be published in the blockchain but its privacy maintained by using different encryption schemes. Generally speaking, encrypted on-chain data can be further processed in several ways, with the highest potential found in homomorphic encryption and secret multi-party computation techniques. Furthermore transactions can be obfuscated with cryptographic techniques to anonymize or conceal aspects of it (such as ring signatures). The idea is that the data is available on-chain, but protected.

  (b) *Proving off-chain computations.* Cryptographic operations can be used to "commit" to a value while keeping it hidden. In a general sense, the idea is to not make any data public even in an encrypted or anonymized form, but just publish enough information to prove publicly that the

operation is performed correctly off-chain. This can be achieved with commitment schemes and Zero-Knowledge Proofs (e.g., zk-SNARKs or zk-STARKs). These cryptographic constructions allow a prover to demonstrate the validity of a statement without revealing any information about the statement itself.

- **Hybrid public/private contracts.** A sufficiently powerful execution layer requires maintaining several conditions for privacy preservation. Many solutions focus on only one or two, since enforcing all three can potentially limit the usability of the system:

  (a) *Transparent anonymous computations.* These computations are performed openly and can be verified by anyone, while preserving the anonymity of the parties involved. This allows for trust in the computation process without revealing sensitive information about the participants. In it by itself only provides partial privacy (ie anonymity), but is a fundamental component of a comprehensive solution.

  (b) *Confidential computations.* Enabling secure computations on encrypted or private data. This could be solved using FHE in the former case, ZK proofs in the latter. The aim in both approaches is to allow data privacy to be maintained while still enabling the necessary calculations to be performed.

  (c) *Function Privacy.* This concept pertains to maintaining the privacy of which function is being called in a given computation. By keeping the function call private, it becomes more difficult for outside observers to determine the exact nature or purpose of the computations being performed.

- **Extensibility**. The third core requirement of the execution layer is the ability to incorporate new privacy-preserving primitives with relative ease. While unlimited extensibility is not something that can be accounted for, there are some considerations that we can take from the beginning, by looking at the expectations of certain cryptographic technologies of becoming practical. We divide the concept of extensibility into two more specific areas:

  (a) *Distributed computations.* We need to support new operations that might require distributed algorithms or network operations. An example would be threshold encryption (TE) or multi-party computations (MPC), which require network communicate as part of the protocol.

  (b) *Cryptosystems.* Similarly, cryptographic systems such as fully or somewhat homomorphic encryption (FHE/SHE) are either practical today

or have a high chance of becoming practical in the near term, making the consideration of extensions support them relevant for the design of Nomos.

TODO:

Plausible deniability enables a party to disclaim involvement in a confidential transaction. Both Monero and Quisquis facilitate this characteristic, as the individual initiating the transaction can select any public key within the system to be included in the anonymity set, regardless of the key owner's consent. However, Zerocash lacks this feature, as its anonymity set comprises all private coins in the system. Consequently, in Zerocash, a user consents to be included in the anonymity set as soon as they possess a private coin.

2 Network layer privacy

TODO

Current thoughts:

1. Describe requirements.
2. Describe how Waku solves them

3 Mempool privacy

TODO?

1. Describe requirements.
2. Purpose: MEV protection and censorship resistance, not privacy preservation.
3. Probably not necessary to elaborate much, as we might not implement this.

4 Consensus layer privacy

TODO

Current thoughts:

1. If delegation: manual choice of where to delegate is a problem if the stakes are private, and other reputation data like reliability. Idea: random assignation.
2. If no delegation (or equivalent, such as random assignation of delegations): just need to hide validator IDs. **Verify this claim**.
3. Does the consensus algorithm use explicit or implicit voting? If the vote is implicit (ie. a *Yes* vote is casted by attesting the block for example), no voting confidentiality needs to be implemented beyond hiding the participation of the validator.

4. Secret Leader Election. Not a privacy measure, but a security one. If the validator identities are already private, secret leader election becomes less critical in the context of PoS blockchains. In this case, the main goals of secret leader election (privacy, security, and censorship resistance) are already addressed by the anonymity of the validators.

On random assignation of delegated stakes. Two potential lines of research:

1. Distribute such that the equal delegation takes place just initially when committed

2. Readjusts dynamically based on the set of current validators every period.

The difference is that the latter just treats every validator the same regardless of history, the former generates a variation in staked tokens per validator implicitly via slashing over time (for which you can compensate with higher block rewards), so it is an "opt-out" in a way and also leads to higher allocation to reputable validators.

Current best idea: the end state would be something like an in-protocol private liquid derivative, with uniformly random distribution of the delegated stakes across all validators. Could work as a hybrid between both able to home stake and buy the staked derivative asset. This gives the best of both worlds and enables privacy.

Incentives should be put in place to favor home stakers, perhaps by setting a minimum fee for delegated rewards.

5 Inter-chain privacy preservation

TODO

Research: are there mitigation techniques?

6 Execution layer privacy

The execution layer is, due to its programmability, the most complex layer to design for privacy purposes. It's also highly exposed to the rapid advances in cryptographic protocols, allowing it to benefit directly from advances in FHE, MPC and ZK proofs.

6.1 Introduction

In previous sections we have reviewed the requirements for this layer, so we will explore the main options we have to achieve our purposes.

UTXO Model vs. Account Model. The choice of the underlying blockchain model can have a significant impact on the privacy properties of a protocol. Each model has its own implications for privacy:

(a) *UTXO Model:* Used by Bitcoin and its derivatives, the UTXO (Unspent Transaction Output) model represents the transaction history as a set of unspent outputs that can be spent by a valid signature. This model enables better privacy by allowing for transaction mixing protocols, such as CoinJoin, which obfuscates the relationship between inputs and outputs by combining multiple transactions into one, making it difficult to trace the flow of tokens.

(b) *Account Model:* Used by Ethereum and similar blockchains, the account model maintains a global state that tracks each account's balance and associated data. This model may require more complex privacy solutions, such as fully-homomorphic encryption, or a combination of regular encryption and zero-knowledge proofs.

Add references, based on enumerated examples. Add more examples.

Off-chain vs on-chain privacy. Closely related to the underlying data model is the pursuit of two different approaches for privacy depending on where the data lives and the computations are performed:

(a) *Off-chain:* The data is held at the user's device, and the computation is privately performed there with no need for encryption. In this case, the user needs to provide the system with a proof in zero knowledge that the operations were performed and satisfy certain predicates. The proofs go on-chain, and allow consensus on the computations performed by the user, that they were done correctly and satisfy certain constraints. This is done

without revealing any information about the user's data thanks to Zero-knowledge proofs.

Another approach to compute with off-chain is Multi-party computation. In this approach, multiple devices can contribute to a computation, without knowledge of the entire computation or all the inputs. This collaborative approach to secure computation is generally not enough per so to provide privacy-preservation, but is a powerful off-chain computation mechanism that can unlock certain use cases.

(b) *On-chain:* The data is stored encrypted on-chain. This is made possible by using homomorphic encryption. Homomorphic encryption enables privacy-preserving operations on blockchain by allowing users to encrypt sensitive data before submitting it to the public ledger. Validators can then perform computations directly on the encrypted data, with the results remaining encrypted. Once computations are completed, the encrypted results can be shared with relevant parties who possess the appropriate decryption keys, ensuring that only those with proper permissions can access the decrypted data.

Describe more aspects that matter in the design, current state of the art, etc

6.2 Design rationale

Given the complex landscape of solutions available and recent progress in all the cryptosystems involved, as well as current efforts in building practical systems with them, it is reasonable to assume that any system designed with a very strong set of assumptions is bound to be obsolete quite soon. Taken this into account, we will attempt to design a system that

(a) is founded in strong design principles that solve fundamental problems in privacy and are consistent with state-of-the-art approaches (even if not completely practical yet),

(b) is flexible enough to accommodate improvements in such cryptosystems over time without requiring a major re-design,

(c) enables expressing different types of computations and privacy-preserving application designs.

Taking these over-arching principles, there are some specific requirements that we can derive as a starting point. These are:

- **Avoid contention issues** naturally arising from off-chain computation. All transactions performed over shared data need to be totally ordered, which is complicated by the fact that off-chain computation doesn't happen in the same environment than the ordering is.
- **Enable separation between public/private** in some form between parts of the transaction, to account for hybrid models. It is not possible to build a practical decentralized application where users interact without any public information at all. Consequently we need to account for hybrid models that allow app developers to decide which parts need to be made public. Note that this requires some effort from the developer's side in understanding that public data can leak knowledge of the private side if the application is not carefully designed.
- **Support account abstraction** in the data model. This allows for both externally owned accounts and contract wallets to have the same functionality, unlocking a series of use cases otherwise impossible. Some examples are: customizable security rules, account recovery options, shared security across trusted devices or individuals, gas payment flexibility, batched transactions, and enhanced innovation opportunities for dapps and wallet developers.
- **Support authorization keys**, to enable a private data owner to delegate the execution of a specific.

  TODO: Research pending. Unclear.

6.3 Design overview

The design of the data and execution models is approached throughout the progressive addition of features that can be built atop the previous ones. We will start solving the most fundamental problems (such as the concurrent state issues) and progressively add features in a manner that allows tackling different problems separately. Following this structure, there are four aspects that we will focus on:

1. **Shared and owned state separation.** Achieving hybrid public/private contracts (in other words, contracts where some data is made public but some remains private though still respects the rules of the contract), is a nuanced topic. Both extremes of the spectrum (all computations are public, or all computations are private) define a clear model. Once we want to introduce partial privacy, the boundary between the private and the public becomes fuzzy, primarily as a consequence of public data effectively leaking part of what is intended to be private. An example of this are public

DEXes, which by providing a pair price, make it particularly hard to conceal the values of each swap (even if they are never published), unless specific measure to mitigate this are taken.

The solutions to handle this boundary between public and private data, and the way they are crossed (such as re-keying, threshold encryption or homomorphic batching), are multiple and offer potential for research. As a consequence, in the design we focus on the aspect that really matters for the data model: ownership of the data. By ownership we refer specifically to *the ability to access and compute the data within the context of the blockchain*. In other words, it refers to two possible options: the data is shared among several entities (and thus requires coordination), or is exclusively owned by one. This is implemented with a data cell model, a straightforward extension of the UTXO model that tags outputs as *owned* or *shared*.

This simple approach offers the flexibility to employ different public-private hybrid models based on one or several cryptosystems. Note that this separation opens up the possibility of optimization at the consensus layer for owned data, as well as a generalization of what private means (such as in MPC, in which a single entity is comprised of multiple subentities that achieve the computation collaboratively).

Reference: Cardano, Nervos

Think of other potential naming, such as local/global, single/multiple

2. **User-level privacy with ZK proofs.** Once we have defined the generic data model based in ownership, we can start identifying which cryptosystems can help us achieve certain privacy qualities on top. We will start with the private data processing model, and base our design on a model considered robust and efficient enough, based on Zexe, Aleo.

   The model, in its essence, can be summarized like this: the private data is living off-chain in the user's device. Any creation or destruction of private data needs to satisfy two predicates: a creation predicate and a destruction predicate, which constraints the computations that can be performed by the user in his or her device. Transforming the data is the process of destroying old data and creating new one (very much in the UTXO classic model from Bitcoin).

   Satisfying these predicates is only part of the answer, as the data is intended to be kept private so these proofs aren't trivial. To solve the privacy part, proving these constraints needs to be done in zero-knowledge, to avoid revealing any information about the computation performed by the user,

and just proving to the network that the computation is done within the allowed terms. Zero-knowledge proofs, commitments and nullifiers, combined provide a strong level of privacy in which it is possible to perform private p2p transactions as well as private user-level computations. These are hardly enough to build the kind of generic environments we would require for Nomos, but with a little help from other cryptosystems, we can get much further.

Reference: Zexe, Aleo

3. **Actor model.** The third building block of the execution layer tackles the computation happening on-chain, at the validators. This data isn't necessarily private (although it could be as we will see). Regardless of the visibility of the data, as soon as the data is owned by multiple entities (ie is on-chain), access and modification of this data needs to be coordinated. For this purpose, we base our execution environment on the actor model, inspired by asynchronous programming. The purpose of an actor is to handle on-chain computations (as well as deciding their order), while checking the validity of these with the constraints proven by the ZK proofs. There are two key components to the actor model in Nomos:

   (a) **Batched processing of message.** Batching is a key enabler of our architecture. There are two critical reasons to perform batching: construct per-block anonymity sets to avoid leaking private information in the individual transactions, and handle concurrent operations on shared state within the same block. At the actor (handles anonymity, concurrency and increases performance).

   (b) **Asynchronous programming.** In certain cases, one of the complications of batching multiple operations is that the result will either be incompatible with the private data computations (as provided in the zk proofs). For example, a zk proof will try to mint a new value equal to an expected trade's output. However, this output is not known until after the batch is processed. To solve this, the private computations need to be defined in terms of intents, which will then be finalized in a second step that *materializes* the transaction from the outcome of the batch. This is usually done proportionally to the participation in the batched operation, but that is highly dependent on the application.

   The implementation of this is done via NFTs of the intermediate state, that is used to prove afterwards the participation in the batched processing within the block. A second transaction to spend the NFTs using the public data after the batch processing is required. However,

this second transaction doesn't need a new signature so the user agent can handle that automatically.

Reference Penumbra for async and actor model. Reference Anoma for intents.

At this point, with a foundational data model, and an execution model for user-centric privacy as well as for transparent shared data, many useful real-world applications are viable already. We have all the building blocks to build an execution layer for Nomos capable of supporting basic private/public contracts.

4. **Multi-user private computation with encrypted shared state.** The possibility of performing computation over encrypted data in a way that is protected from the execution environment itself, is known as fully-homomorphic encryption. Furthermore, if we require that the encryption is done to a shared key with a threshold (like in threshold encryption schemes), this cryptosystem is a Threshold Fully-Homomorphic Encryption. With this functionality, plus the ability to re-key, we gain much more control on the kind of data that we can publish. This in turn unlocks a whole new set of hybrid applications where private computations between parties can be performed without requiring online MPC protocols directly between the users.

   This fourth stage of the design is arguably not feasible yet. But there are high chances that the THFE technology will be practical in the next 2-5 years, so considering its feasibility a very real possibility within a reasonable time-frame, inspired us to introduce it as the last component.

   Reference Zama

6.4 Data model

The data model serves as the foundation of the execution model. It enables storage of arbitrary blobs of data, where computation may or may not be performed.

Extended Unspent Transaction Output model. The eUTXO model is a variation of the traditional UTXO model used in cryptocurrencies like Bitcoin. The eUTXO model aims to provide enhanced functionality and programmability while preserving the benefits of the original UTXO model. In the UTXO model, a transaction consumes one or more unspent transaction outputs (UTXOs) and creates new ones. Each UTXO represents a specific amount of cryptocurrency and can be associated with a locking script that defines the conditions under which the UTXO can be spent.

The eUTXO model extends the UTXO model by introducing the concept of *extended* transaction outputs. These extended outputs can contain not only a value and a locking script but also custom data and state information. This additional data allows for more complex and expressive smart contracts to be built on top of the eUTXO model, similar to what can be achieved using the account-based model found in Ethereum.

One of the advantages of the eUTXO model is that it maintains the ability to process transactions concurrently, as each transaction input operates independently of the others. This results in better scalability and parallelism, as nodes can process multiple transactions simultaneously without waiting for others to complete. However, in its naive implementation this can only be achieved over data owned by a single party. If multiple parties try to modify shared state simultaneously, only one can succeed per block. To solve this, we will employ a two-signing process described later on.

Data cells. The Nomos Data Model operates on data units known as cells. It aims at decoupling the notion of private/public state with that of ownership of the data. The purpose of this is to tackle concurrency first, and enable the construction of a flexible execution model on top by combining several cryptosystems. The design consequently distinguishes between two types of cells:

- **Owned cells:** stores state that can only be modified by a single party.
- **Shared state:** contains state that is modified by multiple party, and thus needs to be coordinated.

Owned cells can be used arbitrarily to store any data, but they have always to satisfy one of two predicates in every transaction: a create predicate if they are created anew, or a destroy predicate if they are being spent. Valid transactions generate and consume records. A transaction is considered valid if, when provided with the transaction's local data as input, all the destroy predicates of consumed records and the create predicates of generated records are satisfied simultaneously.

TODO: Elaborate details, serial numbers, replay protection, function privacy, etc

TODO: Diagram of data cell types and contents

Code cells. Code cells are shared cells that contain executable code. They can

- **Arbitrary contract code** as defined by the Virtual Machine (see corresponding section).
- **Predicates** are functions that return a boolean.

- **Scripts** facilitate sophisticated on-chain interactions that do not need to be permanent. Complex transactions usually involves several steps, such as voting on proposals, staking tokens, and claiming rewards. With the help of scripts, these intricate transactions can be carried out more seamlessly in a single step.

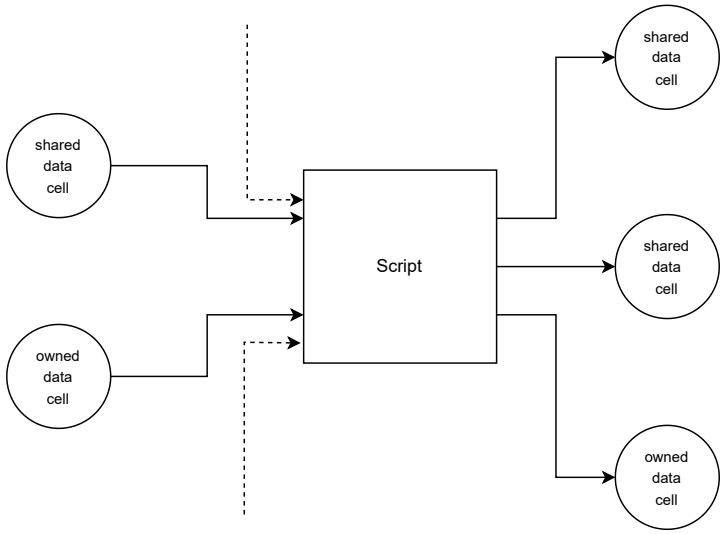

Figure 2: A script running a transaction consuming both shared and owned data, and generating new ones. The script itself calls arbitrary contract code and create/destroy predicates

Reference: Sway language

Transactions. Transactions can be comprised of either owned and shared data cell transformations. If these need to be combined, a script can be used to tie together arbitrary contract code for modifying public state, as well as predicates validating private transformation. Chaining these operations can be performed freely, as long as each step is valid.

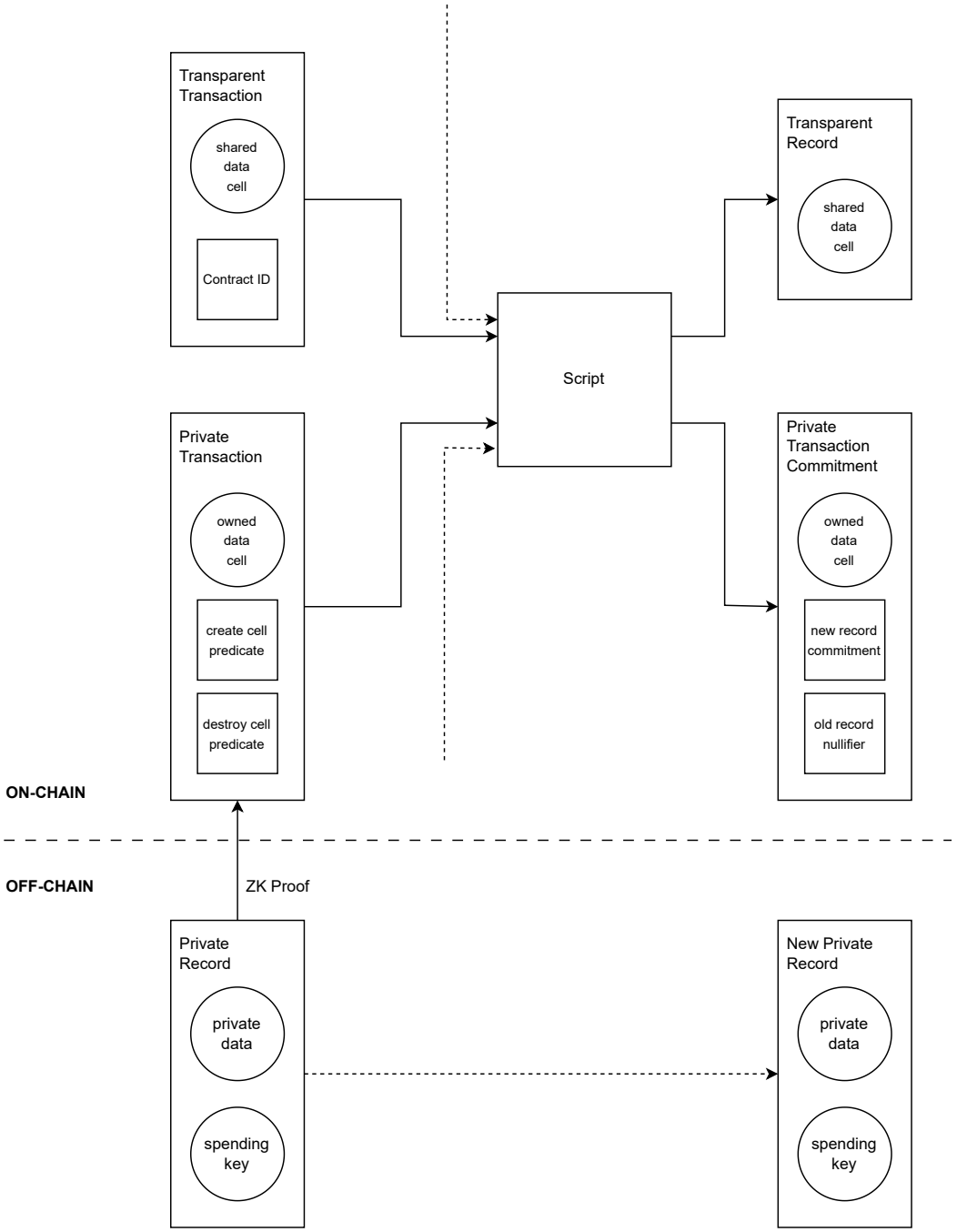

Figure 3: A complex transaction in the Nomos data model involving both public and private computations

6.4.1 Stealth addresses

TODO

Open questions:

1. Maintain privacy of participants in protocol (ie their addresses shouldn't be reused by others)
2. Mechanism to easily generate new addresses for each transaction, that only the sender and receiver can identify

6.4.2 Fuzzy Message Detection

TODO

Open questions:

1. Mechanism for receivers to privately detect messages for them.

6.4.3 State growth mitigation

TODO

Open questions:

1. Study the problem of state growth.
2. Research accumulators.
3. Research state pruning.

6.4.4 Other privacy-preservation measures

TODO

Open questions:

1. With a proving system with no constant size proof, there is the possibility of larger private chunks (data cells) forming an identifiable pattern. Study the possibility of splitting in several in order to obfuscate.

6.5 Execution model

The first component of the execution model that will focus on is on purely private computation, on one single user. From there we will work on ways to combine that with computation of shared states, using the UTXO framework describe before. The last step is to bring multi-user privacy to shared state.

6.5.1 User-level privacy using ZK proofs

Private computation on a blockchain using zero-knowledge proofs (ZKPs) enables secure and confidential processing of data without revealing the underlying information. ZKPs allow a prover to demonstrate the validity of a statement without disclosing any additional information, ensuring that sensitive data remains undisclosed during on-chain computations. However, the use of ZKPs for privacy comes with its own set of limitations.

One such limitation is the need for concurrent-access-to-state. In smart contract chains like Ethereum and Solana, the state of an application is shared between all users, and any transaction can modify it. This shared state poses a challenge for applications that require privacy. When users attempt to modify the shared state on the client side, conflicts can arise between different users, leading to race conditions.

To overcome this limitation, an intermediary agent, known as the prover, can be introduced. Users can send transactions to the prover, who orders and executes the transactions off-chain before advancing the state on-chain. While this approach can address the concurrent access to state problem, it's less than ideal for privacy reasons, since the prover needs access to the cleartext, while also introducing a centralized point of failure. This is not an option that we can consider for our purposes.

References: Aleo, Zexe?

Another, previously discussed, limitation of ZKPs is that they provide single-user privacy. This means that the user must generate their own proof, if they want to preserve full privacy, as opposed to delegating the task to a third party. Additionally, the need for a user to know all private inputs relevant to the proof to generate the proof further complicates matters.

To address these limitations, we need to design on-chain applications with user-level privacy in mind. Instead of a shared global state that everyone can modify, we can use a fragmented state model where individual state fragments are owned by individual users. This approach ensures that users can execute their transactions on the client side without conflict, as each fragment is owned by a single user. The problem with this is that it limits the design space for many applications. For example, it is possible to design decentralized AMMs using only user-level privacy by bypassing the need for users to interact with each other. Instead of transacting with each other, users can transmute one asset type to another, which updates their private state in a provable manner without interacting with other users' private state. However, to achieve this, users burn their input assets privately and encrypt the amounts to validators, which would require additional mechanisms like threshold encryption. The validators then aggregate the encrypted amounts, decrypt the batch total, compute the effective

clearing prices (inclusive of fees), and commit them to the chain state. Users can then privately mint output funds of the new type in a later block, proving consistency with the inputs they burned.

As we can observe, user-level privacy alone has several limitations that need to be overcome by the use of additional cryptographic mechanisms for relatively complex applications. It is however a powerful tool that can be used as a foundational building block of a flexible execution model.

Predicates are zero-knowledge verifiers. User-level privacy based on Zero-knowledge proofs is supported directly by our data model, via the *create* and *destroy* predicates. While there is no fundamental restriction on how those predicates need to perform computations as long as they return a boolean value, their power is revealed through the user of Zero-knowledge proofs.

TODO: Elaborate mechanism

References. Shielded transitions: https://github.com/anoma/taiga

6.5.2 Actor model: batching and asynchronous programming

In the context of smart contracts on Unspent Transaction Outputs (UTXOs), the contention problem is a well-known issue that can arise when multiple transactions within a block attempt to spend the same contract UTXO. This problem can occur when using certain decentralized applications, such as Automated Market Makers (AMMs), where many transactions may attempt to swap into the same pool.

In first place, we introduce **batching** in order to solve two main problems: concurrency and confidentiality.

Concurrency: Two-party signing process. A solution to the problem of concurrency required for batching is to implement a two-party signing process, where users only sign over the contract ID rather than the contract UTXO when initiating a transaction. By doing so, users can indicate which contract they want to interact with without forcing the contract to be in a particular state before or after execution. This approach is intuitive because users do not actually know the state of the contract at the time of execution, as the final order of transactions is not determined by them. Block producers, on the other hand, can determine the order of transactions and know the state of the contract at execution time. This allows them to sign over the exact input (contract UTXO) that the transaction will spend. By doing so, they can avoid contention and ensure that the smart contract functions efficiently. Upon execution, the contract UTXO is consumed and replaced by a new UTXO that represents the post-state of the contract.

Overall, implementing a two-party signing process and having block producers determine the order of transactions can help mitigate the contention problem when using smart contracts on UTXOs. This approach allows for efficient and reliable execution of smart contracts in a decentralized environment.

TODO: Elaborate technically. Make diagrams.

Confidentiality: Somewhat Homomorphic Encryption. The aforementioned two-party signing mechanism solves the concurrent execution aspect. But batching needs to be introduced for yet another reason: protecting the private information to be extracted from public data updates in the same transaction. The problem goes as follows. Take again the AMM example, in which pricing is a public piece of data so users can choose whether to swap assets or not. Even if transactions keep the amounts private, the price change after the execution of a swap reveals enough information to know how much was traded, even if that information was never directly exposed. When batching several transactions together, this problem can be solved to a certain degree, since only the price change after all the transactions are performed can be observed. In other words, traders' contributions to batches and pooled liquidity are kept private, whereas the aggregate values such as total value in batches and liquidity pools remain public.

For this to be possible, however, we need to introduce another cryptographic tool: Threshold Additive Somewhat Homomorphic Encryption. This would allow us to threshold encrypt transactions to a committee of validators, and homomorphically aggregate the values into an encrypted total sum. Assuming a threshold of committee validators are honest, they will distribute their decrypted shares of this sum to the network to finalize the committed block.

Note that while this process introduces a new step, it can be avoided for data that doesn't require to be kept private throughout the batching. Batching is both a concurrency and confidentiality solution, but **confidentiality of the public data is only required if it exposes private data**.

TODO: explain in technical detail how decrypted shares are distributed after the block is committed, following threshold decryption

Note that this solution while powerful, is only as good as the size of the set of transactions in the block pertaining to the same contract. This will work well when the app is popular, but in the case of a single transaction per block per contract, this mechanism is rendered useless. For this purpose, and especially initially when an app has little user adoption, a separate mechanism should be introduced for the case of low transaction count. Some options to explore in the future include:

1. Introducing an MPC-generated noise perturbation on the public value.

2. Delay transactions until there are enough to guarantee some degree of privacy. Note that this introduces the challenge that anyone listening to the mempool can activate the inclusion by crafting known transactions, and derive the public values of the single transaction that is being targeted. However this vulnerability is limited in time and only possible before committing on-chain.

3. Introduce decoy transactions

References of papers by Tarun and Guille on AMM privacy

Reference similarities with MEV-mitigation mechanisms with batching

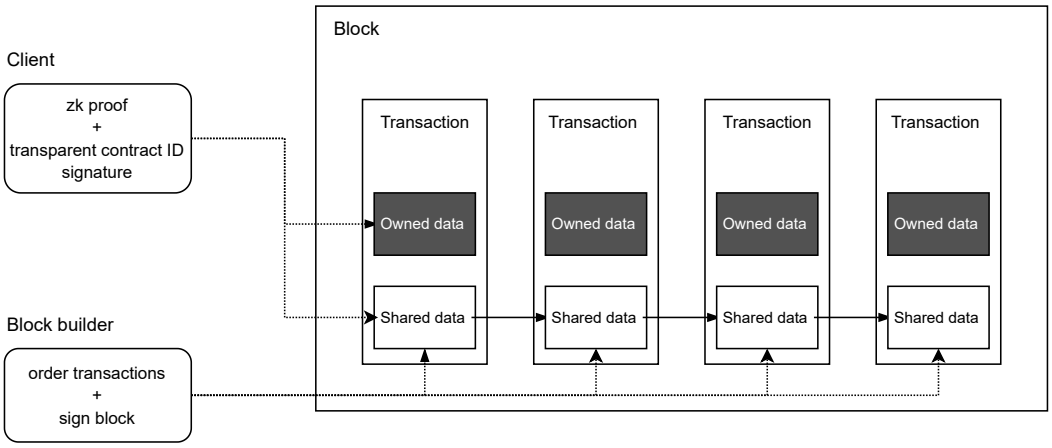

Figure 4: Processing of concurrent transactions in the Nomos data model.

6.5.3 Completing the sequence: asynchronous programming

Batching solves the two problems described above, but introduces a new one: when operations are batched together, the outputs of the computation cannot be obtained until after the batching, since those public values are aggregated and split proportionally among participants.

The solution to this is to use a two-step mechanism, postponing the final computation to after the batch is committed on-chain. In a manner akin to futures in asynchronous programming, the first step would act as a *promise* of a computation, while the second step would *resolve* the computation.

TODO: Elaborate mechanism with NFTs. Make diagram.

Transaction processing.

TODO: Describe how everything fits together here

1. Separation of owned computation (off-chain by user), and shared computation (on-chain, batched on validators).
2. Two-party signing process.
3. Create an NFT of the future.
4. Batching + TE acting similarly to anonymity sets, hiding the effect of each transaction on the public data, and collectivizing the effect, so that only the aggregate change is known. At least Somewhat HE is required for this.
5. Resolve the NFT to extract computation output. This doesn't require a second signing.

On parallelization An account-based model can be designed to enable parallel execution of transactions, but this requires additional considerations such as the use of strict access lists. In contrast, a UTXO model inherently allows for parallel execution since each transaction specifies which UTXOs it intends to spend, and transactions that spend non-overlapping UTXOs can be executed simultaneously.

This is a significant advantage of the UTXO model, as it allows for the benefits of multi-core processing to be fully leveraged, enabling faster and more efficient execution of transactions. Moreover, parallelized execution not only applies to transactions within a block but also to transactions across blocks, which can speed up the syncing process for nodes that are joining or re-entering the network.

6.5.4 Multi-user level privacy: TFHE

With user-level privacy and the ability to use threshold-encrypted batching, we are able to considerably expand the possibilities for the decentralized application developer, opening the door to building complete privacy-preserving apps with multiple users. However, there are still several limitations with this functionality:

- While it is possible to perform computations with per-user privacy, and also to keep those computations safe from being extracted by the simultaneous updates on public state thanks to threshold-encrypted batching, it is not possible to perform computations that involve *private data of multiple users at the same time*.
- All user-level private computations need to be performed by the client, but it is possible to imagine scenarios in which we would desire those *private computations to be performed on-chain*. For example, this data can be used both for single-user computations as well as multi-user seamlessly, be made public directly but only after transformation, or be encrypted to a different user as a form of on-chain messaging.

Fully-homomorphic encryption is certainly not practical today, but is its feasibility is in a reasonable timeline. This component of the privacy-preserving execution model of Nomos will be implemented after the others, in a manner that could be considered as an extension of the previously described components.

Threshold Fully Homomorphic Encryption scheme.

TODO

Overview of solution:

1. Users encrypt their input with the THFE public key of a validator committee.
2. Users generate a proof of validity of the input data (to avoid providing random bits).
3. The transaction sent by the user to the mempool includes both the encrypted data and the proofs.
4. The block builder checks the proofs and executes the transactions in the block.
5. The network agrees and commits the block.
6. The committee nodes exchange decryption shares of the result. Check how this is done to not expose values to nodes. Is it possible to decrypt only the public parts and then do the key switches in the user private parts?
7. The committee nodes run a threshold keyswitch of the user states.
8. The user states are either delivered to the user or stored encrypted on-chain (TODO).
9. The users are then able to decrypt their own state.

Keep in mind that data cannot remain THFE-encrypted on-chain, since the committees are changing over time.

7 Construction of privacy-preserving apps

TODO

Elaborate:

1. These apps have been selected as foundational for a Network State. We will explore how they can be built on the platform described above.
   (a) The ability to exchange assets p2p (transfers)
   (b) An open marketplace for assets (DEXes)
   (c) An over-collateralized lending protocol
   (d) A structure for governance and organization (DAOs)
2. How can apps be designed with this paradigm.

7.1 P2P token transfers

TODO

How is a Zcash-style private transfer implemented on Nomos?

7.2 Decentralized Exchanges: Automated Market-Makers

TODO

Privacy of an AMM in the ideal scenario

1. Maximally private: swapped values, swapped denominations, users, liquidity providers, LP positions (if concentrated liquidity)
2. Necessarily public: available pools, price of each asset, liquidity or slippage calculation per pool.

Challenges

1. **Price discovery leaks information about the private transactions:** In private AMMs, changes in pricing over time can potentially reveal transaction values, undermining the privacy of the participants. This issue arises because the price of assets in an AMM is determined by the ratio of the assets in the liquidity pool. When users trade assets on the platform, they change the balance of assets in the pool, which in turn affects the price. If an

adversary closely monitors the pricing data of the AMM, they can observe the fluctuations in the prices of the traded assets. By analyzing the timing and magnitude of price changes, the adversary may be able to deduce the approximate transaction values, even if the transaction details are encrypted or obfuscated. This is particularly concerning in private AMMs, where users expect a high level of confidentiality and anonymity. To mitigate this risk, private AMMs can implement various strategies to obscure the relationship between transaction values and price changes:

(a) **Introduce time delays**: By introducing random time delays in processing transactions or batching multiple transactions together, the direct correlation between individual trades and price changes can be weakened, making it more challenging for adversaries to identify specific transaction values.

(b) **Price slippage controls**: Limiting the maximum allowable price slippage for each transaction can reduce the impact of individual trades on the overall pricing, making it more difficult for adversaries to deduce transaction values based on price changes.

(c) **Use of decoy transactions**: Introducing decoy transactions with varying amounts and timings can help obfuscate the relationship between real transactions and price changes, making it more challenging for adversaries to link transactions to price fluctuations.

2. **Liquidity values per pool leak information about positions:** Similarly to price discovery leaking information about individual trades, liquidity positions can be be exposed by changes in pool liquidities. Similiar solutions to those of the price changes can be applied, further improved by the use of a Multi-asset AMM where all positions are confined to the same global anonymity set.

   TODO: References for Multi-Asset Automated Market Maker (Namada)

3. **Timing Analysis:** Examining the timing of transactions enables adversaries to correlate and identify patterns that might disclose information about the parties involved in private transactions. The process usually involves observing transaction timestamps or block inclusion times to detect unusual patterns. These patterns can be particularly revealing when the timing of certain transactions is unique or when there are a limited number of participants engaging in the private transactions. Adversaries can then use these observations to uncover relationships between addresses, potentially leading to user de-anonymization or exposure of sensitive business operations.

4. **Volume and Flow Analysis:** By analyzing the volume and flow of assets in a DEX, adversaries can identify high-value transactions or targets, even when transaction details are encrypted. This method involves monitoring patterns in the frequency, size, and direction of transactions within the DEX ecosystem. For example, an adversary might notice that large transactions tend to occur at specific times or that certain assets frequently move between a set of addresses. By establishing these patterns, the adversary can potentially infer the underlying private transactions or identify strategic targets for further analysis or exploitation.

Construction of a private AMM in Nomos. Description. Elaborate how each components tackles the challenges above.

1. Anonymity of users and LPs: ZK
2. Confidentiality of tx: need batching, or randomness in price. Describe the problem of changes on the pricing in time, being enough to derive transaction values.
3. Batching only possible with enough transactions. Otherwise randomness.
4. Batching works with HE of specific amounts, and decryption of total.

Describe other constructions briefly

1. ZKP+SHE-based: Penumbra
2. MPC+ZKP-based: Renegade

7.3 Lending protocols: Private Money Markets

TODO

7.4 Governance: Private DAOs

TODO

Explore:

1. Private creation.
2. Private proposals
3. Private voting.

8 Nomos Virtual Machine

TODO. Description of requirements and how to approach multiple VMs / computation models

8.1 Owned/shared data types

VM and language that can describe both. References: Sway, Move

8.2 Transparent computations

Language for transparent computations that is compatible with private and TFHE computations

8.3 Private computations

References: Zeestar. Zkay.

8.4 Additive Somewhat Homomorphic Encryption

TODO. Used for batching

8.5 Public value MPC randomness

TODO. Used for confidentiality when batches are too small

8.6 Threshold Fully-Homomorphic Encryption computations

Based on Zama's libraries?

9 Futuristic ideas

TODO

9.1 Intents and solver markets

Intents, also known as expressive bidding or preferences, can encompass a wide range of scenarios. For example, a buyer might value a package of goods A and B at $10, but only $4 for each item individually. A logistics company could quote X dollars for a route if they secure a bidder for the return route at Y dollars. Or, one's willingness to pay more for a ski resort trip could depend on a favorable weather forecast.

In theory, more expressive intents can lead to greater market efficiency. However, implementing this in practice has been challenging due to the complexity of finding feasible and near-optimal solutions at the speed and scale of capital markets. AI offers the potential to create highly efficient intent-based smart markets by leveraging AI-powered solvers to optimally match complex intents. Blockchain technology can then be employed for verification and settlement purposes.