# COTI V2: Confidential Computing Ethereum Layer 2

Nir Haloani[1], Avishay Yanai[2], Meital Levy[2], and Yair Lavi[1]

[1]COTI
[2]Soda Labs

April 2024

**Abstract**

The COTI V2 whitepaper presents an innovative approach to addressing one of the most pressing issues in blockchain technology: the lack of confidentiality. As blockchain applications proliferate, the need for privacy-preserving mechanisms becomes paramount to ensure user autonomy and protect against threats like data breaches, theft, and fraud. COTI V2 introduces a sophisticated cryptographic framework centered around garbled circuits and multiparty computation (MPC), offering a new standard for on-chain privacy as an EVM-compatible Layer 2. This framework not only enhances the Ethereum ecosystem's scalability and security but also opens up new possibilities for decentralized applications (dApps) by enabling confidential transactions and data management. Through a comprehensive overview of the challenges and solutions in achieving confidentiality on public blockchains, the whitepaper articulates COTI V2's potential to revolutionize the blockchain space by making it more secure, efficient, and user-friendly, thereby fostering broader adoption and innovation.

## 1 Why Confidentiality Is Blockchain Networks Next Frontier

### 1.1 Introduction

For over a decade, two distinct technical hurdles have dominated the conversation in the realm of blockchain technology: scalability and confidentiality. While the discourse around scaling solutions has intensified, capturing significant industry attention, a smaller group of dedicated teams has diligently pursued the advancement of sophisticated cryptographic techniques to tackle the crucial challenge of confidentiality. This persistent focus on confidentiality reveals a pivotal shift: privacy is poised to become the next frontier in blockchain development. As aptly stated by Vitalik Buterin, co-founder of Ethereum: "One of the largest remaining challenges in the Ethereum ecosystem is privacy. By default, anything that goes onto a public blockchain is public." This inherent transparency, while initially lauded for its potential to foster trust, now presents a significant barrier to broader adoption.

While scalability endeavors aim to address network capacity and transaction processing efficiency, ensuring confidentiality delves deeper, safeguarding the very essence of user privacy and the confidentiality of their sensitive data within the blockchain ecosystem. As the blockchain landscape matures and its applications proliferate, prioritizing robust privacy solutions becomes imperative to foster a secure, inclusive, and trustworthy environment for users and organizations alike. This exploration of on-chain confidentiality, therefore, represents a critical step towards realizing the full potential of blockchain technology and propelling it towards a future of trust and transparency.

## 1.2 Transitioning from Web2 to Web3

From financial transactions to personal information, individuals are increasingly demanding control over their personal data. This demand stems from the understanding that privacy isnt just a luxury, but a fundamental human right. We believe that confidentiality is a fundamental human right and just as in the physical world, where individuals have the right to private conversations, associations and transactions, it's essential that we find ways to protect it also in the digital world.

In many cases, it already does. Web2, or the internet as we know it today, offers a level of confidentiality that was actually critical to its success in the early days. Part of the reason e-commerce exploded in the late 90s was the fact that it inherited many of the confidentiality features of traditional financial systems. People could purchase items online, without the fear of their transactions being made public. eBay and Amazon didnt broadcast your shopping preferences or your credit card details, and your purchases and personal information were only known by you and your bank.

Confidentiality within Web2 also allowed for greater connection and cooperation between global parties and markets around the world. Completely transparent systems are admirable, but they carry a risk of exposing sensitive personal details, business information or intellectual property, especially between distrusting or malevolent trading partners. Despite the counterintuitive nature of keeping secrets, confidentiality actually allows parties to trust each other more.

Its this trust that has made confidentiality the backbone of a whole host of online services. From voting systems to healthcare, social media and banking, there are numerous Web2 applications that rely on individuals being able to keep specific information shielded from public view. This becomes even more apparent as our lives are increasingly conducted online. Additionally, the rise of AI and big data necessitates access to private datasets for learning, but without compromising individual privacy.

Web2 isnt without its flaws, however. While traditional online systems have devised numerous confidentiality features to protect users information, the centralized nature of these systems has resulted in countless hacks and data breaches. Once your data is leaked, it becomes tradable and potentially abused. At this point, you become the product. These cases are not rare - in fact, by 2025, its estimated that nearly 70% of internet-connected people and 55% of businesses have had their data stolen at least on one occasion.

Web3, with its emphasis on data sharing and user empowerment, presents a paradoxical challenge: navigating the delicate balance between individual privacy and inherent blockchain transparency. While liberating users from the control of centralized entities, Web3 inherits the public record characteristic of its underlying technology. Achieving a high level of confidentiality in Web3 enables individuals to exercise autonomy over both their finances and their digital identities. It empowers users to participate in Web3 without fear of surveillance or unauthorized access to their personal data. With confidentiality, youre able to make informed decisions without fear of external interference or judgment. You become less susceptible to undue influence or coercion from external parties, whether it be governments, corporations, or other individuals.

Unlike with Web2, where regulatory frameworks evolved organically in parallel, Web3 is a completely new paradigm, complete with a complex architecture and no central authority. This means that historically, regulators have struggled to define, let alone provide clarity on how they may operate. The fact that a blockchain currently functions as a completely transparent ledger has only made this problem even more difficult. Confidentiality on the blockchain will not only foster trust and confidence, but it will also allow for the creation of new, regulatory frameworks required for enterprise adoption.

## 1.3 Confidentiality on Public Blockchains

Beyond the technological advancements in scalability and security, a fundamental challenge continues to plague most public blockchains, particularly Ethereum: the complete transparency of transactions, assets, and wallet addresses. Initially hailed as a virtue, this characteristic now presents an obstacle to broader adoption. Businesses increasingly handling sensitive information on Web3 platforms require privacy solutions that are no longer simply convenient, but indispensable for the ecosystem's survival, and while blockchain's transparent nature empowers in many ways, it also carries significant privacy risks.

The critical role of privacy in blockchain is gaining traction among industry leaders and observers. Brian Armstrong, CEO of Coinbase, emphasized the importance of privacy for Layer 2 solutions, signifying its vital role in shaping the future of cryptocurrencies. Conversations with industry partners and stakeholders reinforce this sentiment, highlighting privacy as not just a preference, but a legal and operational necessity for businesses and enterprises. It goes beyond mere information concealment, encompassing protection, control over one's data and its disclosure, and ultimately safeguarding individual sovereignty.

In the realm of blockchain technology, anonymity and confidentiality serve distinct but complementary roles in information protection. Anonymity focuses on shielding the identities of participants, ensuring transactions can occur without revealing who is conducting them, often through mechanisms like pseudonymous addresses or advanced cryptographic methods. Confidentiality, conversely, safeguards the content of transactions, ensuring that sensitive data within a transaction is accessible only to authorized parties. While anonymity solutions preserve the anonymity of blockchain participants, confidentiality protects the actual transaction data, which is crucial for a wide range of blockchain applications where security and trust are paramount.

Beyond securing individual and business information, blockchain confidentiality plays a critical role in safeguarding individual autonomy. This is particularly evident in protecting against theft, scams, and fraud within the realm of decentralized finance (DeFi) applications built on Ethereum. One prevalent example is the exploitation known as Maximal Extractable Value (MEV), where malicious actors leverage the transparency of public blockchains to their advantage. They scan incoming transactions, identify legitimate trades, and insert their own before them. This front-running manipulation alters the asset price for the unaware parties, allowing the malicious actors to profit once the transaction completes. This exploitative practice thrives solely on the full visibility of pending transactions on the blockchain. Confidential transactions, however, disrupt this dynamic by obscuring their details. This eliminates the ability to frontrun legitimate trades for illicit gain, potentially rendering MEV losses a relic of the past.

However, the effect that blockchain transparency has on individual autonomy extends beyond frontrunning. While wallet addresses themselves are pseudonymous, different analysis techniques can often link them to real-world identities. This means even seemingly anonymous transactions can be traced back to individuals, potentially exposing their financial activity, trading strategies, and even personal habits. The risk of such de-anonymization can be a significant deterrent for users seeking true privacy on public blockchains.

Furthermore, the public nature of blockchains raises concerns about data breaches or leaks. If a private key associated with a wallet address is compromised, all the information linked to that address, including transaction history and potentially even the value of holdings, becomes publicly accessible. This can have severe financial consequences and even pose physical security risks depending on the nature of the assets.

This lack of confidentiality creates significant hurdles for various Web3 applications seeking wider adoption. By obscuring transaction details and severing the link between wallet addresses and real-world identities, confidentiality safeguards individuals from de-anonymization efforts and protects their financial confidentiality. Additionally, by limiting the information publicly accessible on the blockchain, even in the event of a data breach, the potential damage is significantly reduced. This fosters a more secure environment for users and encourages broader participation in Web3 applications.

### 1.4 Rethinking On-Chain Confidentiality: Pushing Beyond Limitations

Previously, achieving on-chain confidentiality for decentralized applications appeared nearly impossible. Many early attempts relied on Zero-Knowledge Proofs (ZKPs) for concealing transaction information. While ZKPs offer compelling functionalities, they aren't naturally equipped to address confidentiality in the context of on-chain transactions.

Let's explore this limitation by examining Zcash, a popular blockchain utilizing ZKPs for shielded transactions. In Zcash, users can send transactions that conceal the sender, receiver, and transaction amount. However, this approach comes with a fundamental drawback: the inability to execute complex logic or access the current state of the system.

Since ZKPs operate on encrypted data, it's impossible to directly access the underlying information. This creates a crucial limitation: the system, including smart contracts, cannot evaluate the current state or perform complex calculations using this hidden data. Consequently, answering even basic questions like "How many tokens are in the system?" becomes impossible. This inability to access and manipulate encrypted data poses a significant challenge for decentralized applications (dApps), which often rely on complex computations and shared state.

While promising advancements have been made in ZK-based approaches through advanced techniques, inherent limitations in logic persist. No amount of cryptographic innovation can overcome these fundamental barriers. Moreover, regulatory hurdles often impede the progress of attempts to address privacy concerns. Projects like Zcash or transaction mixing applications like TornadoCash prioritized user anonymity over compliance, leading to regulatory scrutiny and eventual shutdown.

Recent breakthroughs in cryptography, like Fully Homomorphic Encryption (FHE), delivered a new approach to on-chain confidentiality by allowing computations to be performed directly on encrypted data, essentially preserving the overall logic of a transparent financial system. This means developers can design smart contracts and dApps as if they were operating on clear data, even though the underlying information remains encrypted.

While Fully Homomorphic Encryption (FHE) offers a significant leap forward in on-chain confidentiality, it's not without its limitations. FHE computations are inherently complex, demanding significantly more processing power compared to traditional methods, potentially leading to slower transactions and higher operational costs for dApps built on FHE. Additionally, important FHE schemes currently lacks the full spectrum of functionalities available with unencrypted data, potentially restricting the types of dApps it can support.

In summary, a fresh approach is imperative to reconcile privacy with regulatory compliance and user expectations in the Web3 landscape.

**COTI V2 introduces a novel approach that ensures performance without sacrificing confidentiality. To enable a performant solution with strong privacy guarantees, COTI V2 will utilize a novel combination of well-established privacy-enhancing technologies (PETs), with the main ingredient being a Garbling Protocol.**

## 2 The Core Technology: Garbled Circuits-Based MPC[1]

### 2.1 Introduction to Garbling Protocols

Privacy has always been a pivotal aspect of human life, cherished across all ages. However, in the current era, marked by the internet revolution, its significance has magnified. Society is racing to keep pace with technological advancements to ensure personal information remains confidential. In today's digital world, where nearly every action leaves a trace, safeguarding our privacy is not just importantit is essential. We find ourselves constantly navigating the fine line between harnessing technology's benefits and protecting our private lives from intrusion.

The mid-20th century's introduction of sophisticated cryptography signaled a crucial shift towards secure communication. This period was defined by the urgent need to transmit messages safely across potentially compromised mediumsbe it physical documents carried by unreliable couriers or digital data transmitted through the airwaves or wires vulnerable to interception. The burgeoning field of cryptography focused on devising methods that guaranteed not only secure communication but also the efficiency and resilience of these communications against eavesdropping or tampering. This era witnessed cryptography's evolution from an arcane practice to a fundamental, science-driven toolkit essential for military units, governments, and eventually the general public, shaping the complex digital security landscape we navigate today.

By the late 20th century and in subsequent decades, the cryptographic community experienced a revolutionary shift in its research focus, heralding the era of secure multiparty computation (MPC). This cryptographic breakthrough allowed multiple parties to collaboratively compute functions over their private data without revealing the actual data to one another. These advancements extended beyond theoretical significance, offering profound practical implications. They facilitated the secure exchange and processing of information in a manner that preserved both privacy and confidentiality, addressing a growing concern in an increasingly interconnected world. From enabling confidential electronic voting systems to secure data sharing among organizations, multiparty computation marked a considerable advancement. This period of research broadened the horizons of cryptography beyond traditional encryption and decryption, catering to the nuanced requirements of a society ever more reliant on digital interactions and the perpetual challenge of balancing technological convenience with the imperative to preserve privacy.

Opting for a garbled-circuit-based MPC to achieve on-chain privacy aligns with several critical metrics:

- **Modularity**. The garbled-circuit-based solution is structured into two independent phases, termed 'Garbling' and 'Evaluation'. The Garbling phase, involving significant computation by the network nodes, is conducted 'offline' in a pre-processing stage, producing a garbled circuita secure container for data processing. This phase continually generates garbled circuits for subsequent use during the Evaluation phase, where actual transactions are processed. The Evaluation phase is executed by the network nodes in an exceedingly efficient manner. This modular approach is elegant and establishes a 'privacy supply chain'.

- **Security**. Amidst various encryption schemes, our solution aspires to align with industry standards right from the start, rather than introducing a proprietary, untested encryption scheme or waiting for a lengthy standardization process. We employ encryption schemes already widely adopted by the world's most secure systems, including those managed by governments and large banks. This approach involves standardized symmetric-key schemes for encryption (e.g., AES-CTR) and standardized asymmetric-key schemes for key distribution (e.g., RSA), enhancing adoption by eliminating the

[1]Written by Soda Labs

need for additional, unverified security assumptions. Contrary to other MPC methods, garbled-circuit-based solutions facilitate an efficient integration of these encryption schemes within a circuit that can be securely evaluated in a distributed manner.

**Privacy**. In recent years, numerous initiatives have sought to enhance on-chain privacy via the powerful cryptographic tool known as zero-knowledge proof (ZKP). While ZKP allows data owners to verify the correctness of statements about their data without revealing the data itself, it falls short in scenarios involving multiple data owners who wish to collaborate based on their private data. This is vital for a range of blockchain applications, from dynamic identity systems and DeFi applications like AMM to portfolio management, social trading, peer-to-peer messaging, auctions, and governance. Our approach to on-chain privacy is driven by a secure MPC protocol, where users' data is sent to a private data pool. Here, any process, public or private, can be applied to the data without disclosing anything but the process result as intended by the application developer, and only with user consent.

- **Performance**. With the objective of optimizing real-time transaction processing involving private data, having pre-prepared garbled circuits enables nodes participating in the Evaluation phase to achieve a high transaction throughput. The low-latency characteristic of garbled-circuit-based MPC ensures that the number of communication rounds between nodes is constant and does not depend on the number of parties involved or the complexity of the transaction. Crucially, the technologies underpinning both the Garbling and Evaluation phases are ready for implementation on current devices (including smartphones), without the need for specialized hardware or awaiting significant advancements in research.

- **End-user experience**. Maintaining an unaltered security experience (as highlighted in the security discussion), submitting private data to the network resembles sending data over a TLS channel, employing a symmetric-key encryption scheme. This means devices today are already equipped to interact with the network using standard protocols and widely known software libraries. This compatibility extends to software operating on personal computers, web browsers, smartphones, smart sensors, and potentially any IoT device.

This holistic approach to on-chain privacy underscores our dedication to ensuring security, privacy, efficiency, and a user-centric experience, thereby setting a new standard for privacy in the digital domain.

### 2.2 Introduction of gcEVM

Ethereum virtual machine (EVM) is the leading blockchain-native virtual machine, interleaving computer architecture aspects with incentive mechanisms, which made it the first state-transition engine for a decentralized and permissionless general system. One of the main obstacles to the mass adoption of the EVM is its lack of confidentiality, leaving all information public, which is obviously not in par with what businesses, communities, and individuals expect from a system. To this end, we introduce the gcEVM, an extension to the EVM that supports confidentiality by offering a series of functionalities for keeping private data and performing operation on it without ever exposing it (unless required by the execution itself). Because of the nature of the EVM, where everything is visible, handling ciphertexts must be meticulously done, in order to protect those ciphertexts from theft, replication, etc. We demonstrate our solution via Garbled Circuit, hence the extension is called gcEVM.

In the rest of this section we give the necessary background on the EVM and argue about the importance of confidentiality in the EVM for a real impact. We briefly describe the notion of garbled circuits and garbling protocols and lastly we describe our EVM extension that relies on a garbling protocol.

## 2.3 Background on the EVM

**The EVMs Accounts and State**. At a very high-level, the EVM takes an ordered list of valid transactions and executes them. Execution of transactions may change the state of the machine and so, in an abstract way, we refer to this execution process as the state-transition function of the machine. The state of the EVM is composed of many sub-states, each is associated with an account (also known as address); and these accounts may be either external or internal. An external account (also known as externally owned account, or EOA) is an account that may initiate a transaction whereas an internal account (also known as a smart contract) only reacts to requests that were initiated by EOAs. In the following, we use the terms internal account and smart contract interchangeably. The sub-state associated with every account contains its balance and nonce, where balance refers to the number of coins that account owns and the nonce refers to the number of transactions that account has issued so far, such that each newly issued transaction increment that number by exactly one; the latter is used as a protection from replay attacks  so that no transaction may by launched twice. Since internal accounts (smart contract) cannot really initiate transactions, their nonce is incremented only when they trigger the creation of a new internal account, thus, the nonce of a smart contract simply refers to the number of other smart contracts that they have created. While the sub-state associated with an EOA consists of its balance and nonce only, the sub-state of a smart contract may additionally contain an arbitrary data structure along with a set of interfaces (or methods) that can change its sub-state (either its balance, nonce, or the additional data structure).

Transactions may be simple or complex; in a simple transaction an EOA transfers some amount of the native coin to another account (external or internal); in a complex transaction an EOA may either create a new smart contract (also referred as smart contract deployment) or call a method of an already existing smart contract. In the latter, the reaction of the smart contract to that call may involve further calls to methods of (potentially other) smart contracts, and so on. This way, a complex transaction initiated by an EOA may trigger a chain-reaction that causes changes in multiple sub-states. On the other hand, a complex transaction may also cause no change to any sub-state at all. When a method is said to be a view-method it is guaranteed that it never changes any sub-state, which implies that it can call only other view-methods.

**The EVMs Execution, Memory and Storage**. Execution in the EVM is fueled by gas, which is paid in the native coin (Ether in Ethereum) and serves as a measure of computational effort. This gas is paid from the balance of the EOA that initiates the transaction. Each operation within the EVM, from arithmetic calculations to data storage, consumes a certain amount of gas, incentivizing efficiency and preventing network abuse (e.g., mounting a denial-of-service attack by sending an infinite stream of transactions).

Memory management in the EVM is unique; it maintains a volatile memory store (simply called memory hereafter) during execution but does not retain it after the execution completes. This memory is linear (instantiated with a stack data structure) and expands as needed by a contracts execution but is wiped clean after the process ends. For persistent storage, the EVM utilizes a key-value store known as storage, which persists between transactions but is significantly more costly in terms of gas to utilize. This design encourages developers to optimize their use of storage and memory, balancing the need for persistent data against the gas costs of operations, ensuring that the EVM remains scalable and efficient. The storage is the one that manages the EVMs state, and so each sub-state is maintained as an isolated key-value store.

Let us describe a typical execution process: When a smart contract method is called, and it needs to read data from its storage (e.g., a variable value), the EVM fetches this data from the contracts storage (a sub-state) and loads it into memory for quick access during this specific execution. This is done through specific EVM opcodes such as sload, which reads data from the variables location at the storage. The data read is then available in memory for processing or computation during the contract execution. After the smart contract performs computations or manipulates data within the memory, there may be a need to persist some of this data back to storage. To store data back from memory to storage, the contract uses opcodes like sstore. This opcode takes the data from memory and writes it to the specified location in the contracts storage.

**Data Passing Between Smart Contracts**. The EVM facilitates data passing from one contract to another via a function call; it employs a mechanism that allows contracts to interact and invoke functions on each other. This process is foundational to the composability and interoperability of smart contracts on the Ethereum platform.

When a contract (caller) wants to invoke a function on another contract (callee), it specifies the callees address and the function to be called, along with any arguments required by that function. This can be done using op-codes like call and delegatecall. The data passed to the callee includes information about the function to be executed (identified by its signature) and the arguments for that function. The callee contract then executes the specified function using the provided arguments. This execution can read from or write to the callees storage, depending on the functions logic, and eventually gets back to the caller contract using the opcode ret. A run-time variable, called depth, is incremented on every function call (to a different contract) and decremented when the function returns to the caller.

**Error Handling**. The notion of reverting a transaction or a contract call is a fundamental concept designed to ensure the integrity and security of smart contract operations. A revert operation undoes all changes made to the state during a transaction or call, except for the consumption of gas, and returns an error message to the caller. This mechanism is crucial for handling errors and ensuring that failed transactions do not alter the blockchain state in an unintended manner. A revert error is triggered intentionally by using the revert statement in Solidity or when conditions specified by require statements are not met. Its used to handle logical errors in contract execution, where a certain condition or prerequisite is not satisfied. The gas consumed up to the point of revert is not refunded, but any remaining gas is.

An assert function is used in Solidity to handle internal errors and to check for invariants within the code. If an assert statement fails, it indicates a serious bug in the contract code, leading to a panic error. Unlike revert, assert failures consume all the gas provided with the transaction, signaling a more severe form of error that should not occur during normal operation.

Other types of error are possible in the EVM, like out-of-gas, stack overflow or underflow, invalid opcode, and more, each has a different cause and consequences on the EVM.

## 2.4 The EVMs (lack of) Confidentiality

One of the most controversial properties of the EVM is that everything is public, meaning that the sub-state of smart contracts, which may include financial, social (and practically any kind of) information, is completely visible to all. As we argue below, this property of the EVM is a double-edged sword, which, for a long time, formed a dichotomy between decentralization and adoption.

On the one hand, this fact played as key-contributor to the decentralization of Ethereum (who is the first to deploy an EVM), as it allows anyone with a computer to run it and take part in the execution of agreed upon lists of transactions (with an adequate financial incentive mechanism); thereby increasing the validity of the systems state, or in other words, increasing the trust that the information laying in the state is the product of correct execution of the past transactions.

On the other hand, the fact that everything is visible to all poses a significant setback to the usability and adoption of the EVM. To date, the EVMs prominent use-case is de-fi, which paved the way to collaborative liquidity pools and automatic movement of funds. In many cases however, de-fi is used by bad actors for scam (e.g., rug pools), fraud (e.g., money laundering), and many times is considered as funds streaming in a loop with no real impact. Arguably, one of the reasons for this is the lack of confidentiality, leaving honest actors (smart contracts, DAOs and users) unable to prove and verify each others identity, thereby being more vulnerable to exploits and manipulations. Furthermore, the lack of confidentiality sets an obstacle to real social impact, as most real world social activities deal with private information that must be treated adequately. For example, an election requires the independence of votes and the freedom to vote ones opinion without fearing any consequences. Sealed bid auction, as another example, requires an independence of bids as well as their confidentiality, since public bids expose the bidders sensitive financial state.

## 2.5 Garbled Circuits

In this section we use a variation of the notation and definitions from '*Foundations of Garbled Circuits*' by Bellare, Hoang and Rogaway [BHR12].

### 2.5.1 Circuits

For simplicity, we consider circuits with fan-in of two, even though our solution is not limited to those.

A *circuit* is a six-tuple $f = (n, m, q, A, B, G)$, where $n \geq 2$ is the number of *inputs*, $m \geq 1$ is the number of *outputs*, and $q \geq 1$ is the number of *gates*. The inputs, wires, outputs, and gates sets are indexed by $\mathsf{Inputs} = \{1, \ldots, n\}$, $\mathsf{Wires} = \{1, \ldots, n+q\}$, $\mathsf{Outputs} = \{n+q-m+1, \ldots, n+q\}$, and $\mathsf{Gates} = \{n+1, \ldots, n+q\}$, respectively. Then, the functions $A$ and $B$ are of the form $\mathsf{Gates} \to \mathsf{Wires} \setminus \mathsf{Outputs}$, where $A(g)$ (resp. $B(g)$) returns the first, or left, (resp. second, or right) incoming wire of gate $g$. Finally, $G$ is a function of the form $\mathsf{Gates} \times V^2 \to V$, where $V$ is the domain of values that wires can take. Here $V$ is defined abstractly while typically it is defined by finite group, ring or field. For instance, many times $V$ is instantiated by the domain $V = \mathbb{F}_2 = \{0, 1\}$ and the function $G$, which define a binary (Boolean) gates; alternatively, it can be defined by $V = \mathbb{F}$ for some finite field $\mathbb{F}$ and $G$, which define arithmetic gates over $\mathbb{F}$.

The above embodies the following. Gates have fan-in of two (two inputs), arbitrary functionality, and arbitrary fan-out (an output wire may serve as an incoming wire to unlimited number of gates). The wires are numbered 1 to $n+q$. Every non-input wire is the outgoing wire of some gate. The $i$-th value of an input is presented along wire $i$. The $i$-th value of an output is collected off wire $n+q-m+i$. The outgoing wire of each gate serves as the name of that gate. Output wires may not be input wires and may not be incoming wires to gates. No output wire may be twice used in the output. Requiring $A(g) < B(g) < g$ ensures that the directed graph corresponding to circuit $f$ is *acyclic*, and that no wire twice feeds a gate; the numbering of gates comprise a topological sort.

**Circuit evaluation.** The canonical evaluation function $\mathsf{ev}_{\mathsf{circ}}$ takes a circuit $f$ and a list of inputs $x = x_1, x_2, \ldots, x_n$ and returns a list of outputs $x_{n+q-m+1}, \ldots, x_{n+q}$. See Listing 1 for a formal description:

### 2.5.2 Garbling Schemes

A *garbling scheme* is a five-tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, where $\mathsf{Gb}$ is probabilistic and the rest are deterministic. Let $f = (n, m, q, A, B, G)$ be a circuit that we wish to garble. Recall that $f$ represent a function of the form $V^n \to V^m$. On input $f$ and a security parameter $\kappa \in \mathbb{N}$, the garbling algorithm $\mathsf{Gb}$ returns the triple $(F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, f)$, where $e$ describes an encoding function, $\mathsf{En}(e, \cdot)$, that maps an initial input $x \in V^n$ to a *garbled input* $X = \mathsf{En}(e, x)$; $F$ describes a *garbled circuit*, $\mathsf{Ev}(F, \cdot)$, that maps

**Algorithm 1** Canonical Evaluation $\mathsf{ev}_{\mathsf{circ}}(f, x)$

```
1: $(n, m, q, A, B, G) \leftarrow f$.
2: for $g \leftarrow n + 1$ to $g \leftarrow n + q$ do
3:     $a \leftarrow A(g)$ and $b \leftarrow B(g)$
4:     $x_g \leftarrow G(g, x_a, x_b)$
5: end for
6: return $x_{n+q-m+1}, \ldots, x_{n+q}$
```

each garbled input $X$ to a *garbled output* $Y = \mathsf{Ev}(F, X)$; and $d$ describes a decoding function, $\mathsf{De}(d, \cdot)$ that maps a garbled output $Y$ to a final output $y = \mathsf{De}(d, Y) \in V^m$.

**Projective garbling schemes.** A common approach in existing garbling schemes is for $e$ to consist of a vector of sets of *labels*, such that a set of labels $L_i$ is associated with the $i$-th input or output wire ($i \in \{1, \ldots, n\} \cup \{n + q - m + 1, \ldots, n + q\}$). For example, if the circuit is Boolean then we have $V = \{0, 1\}$ and there are two labels for the $i$-th input wire, namely, $L_i = \{L_i^0, L_i^1\}$. The encoding function $\mathsf{En}(e, \cdot)$ then uses the values $x = x_1, \ldots, x_n$ to select from $e = (L_1, \ldots, L_n)$ the subvector $X = (X_1, \ldots, X_n) = (L_1^{x_1}, \ldots, L_n^{x_n})$.

### 2.5.3 Example: Secure Two-Party Computation (2PC) via Garbled Circuits

Garbling schemes were originally designed as a solution for secure two-party computation, which work as follows. Suppose Alice has $n_A$ inputs, denoted $x_A = (x_1, \ldots, x_{n_A})$ and Bob has $n_B$ inputs, denoted $x_{n_A+1}, \ldots, x_{n_A+n_B}$, where $x_i \in \{0, 1\}$ for all $i$, and $n = n_A + n_B$. Alice and Bob wish to securely compute the circuit $f = (n, m, q, A, B, G)$ over their joint inputs $x_1, \ldots, x_n$. Alice and Bob can use a garbling scheme to do so, as shown in Algorithm 2 in which Alice and Bob take the roles of the Garbler and Evaluator, respectively. The garbler (Alice) generates the garbled circuit $F$ and its own garbled input $X_A$, and sends both to the evaluator (Bob). Then, the evaluator obtains its own garbled input $X_B$ via a cryptographic protocol called oblivious transfer (OT). Finally, using the garbled circuit and the garbled inputs, the evaluator can obtain the garbled output $Y$ and using the decoding information $d$ it can obtain the actual (plaintext) output $y$.

**Algorithm 2** Secure Two-Party Computation $f(x_A, x_B)$

```
1: Interpret $(n, m, q, A, B, G) \leftarrow f$.
2: Alice (the garbler) computes $(F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, f)$ ($\kappa$ is the security parameter).
3: Alice computes her garbled input $X_A = \mathsf{En}(e, x_A)$.
4: Alice sends $F, d$ and $X_A$ to Bob.
5: Alice and Bob invoke a two-party protocol called oblivious transfer (OT). In this protocol Alice privately
   has $e$ and Bob privately has $x_B$, and Bob (and only Bob) obtains the output $X_B = \mathsf{En}(e, x_B)$.
6: Bob computes $Y = \mathsf{Ev}(F, X)$ where $X$ is the concatenation of $X_A$ and $X_B$.
7: Bob computes $y = \mathsf{De}(d, Y)$ which is the output of the computation. At this stage Bob may share
   that output with Alice; in case Bob is suspicious of being malicious other security measurements are in
   place.
```

**Security notions of garbling schemes.** Garbling schemes may be associated with different security guarantees, like *privacy*, *obliviousness*, *authenticity*, and more; in our context we are mostly interested in privacy and authenticity. In a high level, *privacy* refers to the fact that the evaluator is unable to tell which actual (plaintext) value passes through wires of the circuit, even after having the garbled circuit and the garbled inputs (unless it has prior information about those values). Authenticity means that the evaluator cannot produce garbled output $Y'$ that is different than the correct garbled output $Y$ that is obtained through honest evaluation of the garbled circuit.

**Extension to Secure Multiparty Computation (MPC).** The above is an example of how to distribute the computation of $f$ between two parties where each party holds part of the inputs. In our context, we are interested in secure computation that can be executed by many parties where the private inputs themselves might not even be theirs, e.g., private input may contain some identity information of a client (who is not necessarily running an execution node). In a very high level, this is solved by two techniques, the first is *cryptographic secret sharing* and the second is *multiparty garbling protocol*. In such a solution, an external client is the one who holds the private input and the execution parties are responsible for holding it securely and performing secure computation over it when needed.[2]. The client's private data is stored by the execution parties in a way that requires many of them to behave maliciously in order to disclose that data, otherwise, no information can be inferred about that data. With regard to computation over such secret shared data, the parties invoke a garbling protocol that can be run by many parties. Various protocols for multiparty garbling have been proposed in the literature since the '90, optimizing for various metrics like communication rounds, bandwidth and computation complexity. The protocol that we use for the gcEVM (see next section) is fundamentally different than those protocols, although building on similar techniques.

## 2.6 The gcEVM

The gcEVM involves extension of the EVM in multiple dimensions. First, we introduce new data types in order to capture the fact that information of this type must be kept secret; then, we introduce new operations that can perform manipulation on secret data types without disclosing the secrets; and finally, we must take extra care on the way we manage and protect these new data type against attackers who wish to mount some sort of a replay attack. We discuss all these topics in this section.

### 2.6.1 gcEVM-Related Data Types

Similar to other systems, all the information in the EVM sub-state, including the balance, nonce, and anything residing in the data structure maintained by smart contracts, is stored in atomic *typed* variables, namely, variables that capture a certain type of information, be it small or large, signed or unsigned integers, strings, or bytes. Let `Types` be the set of data types supported by the EVM.

For the purpose of supporting confidentiality, we introduce a new set of data types, denoted `STypes`, that is analogous to `Types`; each data type in `STypes` is basically the secure version of one data type from `Types`. For example, we have `uint8`, `uint16`, `uint32`, `uint64` $\in$ `Types`, then their secure version are `suint8`, `suint16`, `suint32`, `suint64` $\in$ `STypes`. Generally speaking, the secure version of an EVM data type will have the same name, prepended with the letter '`s`' (to indicate a `s`ecret). A smart contract developer must use these types if it wishes the underlying information to remain secret.

These data types in `STypes` are all referred as an abstract data type, called `CT` (for 'ciphertext'), which is essentially a re-definition of `uint256`, whereas data types in `Types` are referred as `PT` (for 'plaintext'). Looking ahead, having a ciphertext in a smart contract state, or in the memory during an execution does not necessarily mean that it is *authenticated*; the gcEVM must make sure that a ciphertext is authenticated before entering it into any secure computation procedure.

[2]Computation over a client's private data must be under the client's consent, a constraint we deal with in the next sections.

We make a distinction between secret data types that are used for security 'at rest', 'in transit', or 'in use'. That is, while the *ciphertext* data type (denoted `CT`) are use to secure data at rest (be it the persistent storage or the volatile memory used in the course of an execution of a transaction), we use the *inputtext* data type (denoted `IT`) for protecting data in transit and the *garbledtext*™ data type (denoted `GT`) for protecting data in use.

Protecting data in transit means protecting the ciphertexts that a user wish to send to some smart contract function in a transaction. Specifically, the goal is that when a user incorporate some ciphertext in its transaction, this ciphertext will be used only in the context of this transaction and cannot be re-used in other transactions (by malicious actors). For example, if a user participates in a sealed bid auction and sends a ciphertext in its transaction that hides its plaintext bid, we must prevent an adversary from copying that user's ciphertext and submitting it as its own bid; furthermore, we must prevent an adversary from using that ciphertext in any way, so that the gcEVM will not agree to perform any secure operation on it.

Protecting data in use refers to the fact that even when data is secure on storage or on memory, its security might be broken when performing some operations on it, like using it within a secure computation protocol.

In the following we give a formal description of the three data types:

- **ciphertext (`CT`).** This data type represents the result of a CPA-secure encryption scheme and used for securing data at rest. It is the actual datatype visible in the system's state. Due to other security mechanisms employed in the system, like authenticated memory and storage, and the fact that decryption is performed to 'well formed' ciphertexts only, we do not need to use a CCA-secure encryption scheme (the attacker does not get to choose the ciphertexts to be decrypted by the system).

  Formally, let $\mathsf{Enc} = (\mathsf{kgen}, \mathsf{enc}, \mathsf{dec})$ be a CPA-secure encryption scheme, for a message $m = (\{0,1\}^{\ell})^*$ (i.e., the message length is a multiple of the encryption block length $\ell$) we have:

  $$\mathtt{CT} = c \leftarrow \mathsf{enc}_k(m) \tag{1}$$

  where $(ek, dk) \leftarrow \mathsf{Enc.kgen}(1^{\kappa})$[3] and $\kappa$ is the computational security parameter. Looking ahead, by default, instances of `CT` will be the result of encryption using the system's key, whereas some `CT`s will be the result of encryption using a client's key.

  The encryption scheme we use is `AES128` in the counter mode (CTR), thus, for a message $m = m_1\|m_1|\ldots$ (with $|m_i| = 128$) the encryption result is $c = c_0\|c_1\|\ldots$ where $c_0 = r$, $c_i = \mathtt{AES128}_k(r+i) \oplus m_i$, and $r$ is chosen uniformly at random. CTR mode is advantageous as it allows a random access to a specific slot in the plaintext, and it only performs a forward evaluation of the underlying PRF (`AES`).

- **inputtext (`IT`).** This data type is a wrapper of `CT` only used to infiltrated data to the gcEVM from the outside world. The role of `IT` is to make sure that the wrapped `CT` is used only for the purpose it is intended to by the user who sent it. An `IT` may be formed of an authenticated encryption (e.g., using the encrypt-then-authenticate approach) or a signcryption;[4] in both cases the associated data being authenticated must contain the identities of the sender and the receiver. The gcEVM inherits the transaction format from the EVM and so every message is already signed, and the signature is applied on those identities, as required. The sender's identity (which is the user) is extracted from

[3]Since we use a symmetric encryption scheme we have $k = ek = dk$, but asymmetric schemes may be used in the same way

[4]See [KL20] for a discussion about authenticated encryption and signcryption, and [BSW06, SPW07] for a signatures with strong security; in [BMP22] they argue that `ECDSA` has strong security (also called enhanced unforgeability).

the signature itself, while the receiver's identity is combined of the contract address and the function within that contract to be invoked.

The above suggests that `IT` can be in the exact same format as `CT`, however, there are subtleties that require us taking some extra care. Specifically, instead of fully relying on the signature on the transaction as a whole, we ask the user to individually sign each `CT` (as well as the identities). This is important for security at least for the support of view functions in a setting of a single gcEVM node (i.e., the entire system consists of a single node). Since invocation of view functions do not trigger the verification of a signature on the transaction (in fact, calls to view functions do not have to be signed at all), it means that an attacker may 'steal' an honest user's ciphertext: the attacker's contract will have a function like `leakData(CT c, address sender)`, which onboards the ciphertext `c` to the system's memory using the `sender`'s key, and then decrypts it, so that the plaintext hidden by `c` is revealed to everyone (and to the attacker in particular). The attacker now takes some ciphertext `c` sent to the gcEVM earlier by an honest user of address `user_addr`, and calls the above function with `leakData(c, user_addr)`, which reveals the value that the honest user intended to keep private. Signing each ciphertext individually prevent such an attacks.

Then, we formalize an inputtext as follows. Let $ct_m$ be a ciphertext for message $m$ and let $\mathsf{Sig} = (\mathsf{kgen}, \mathsf{sign}, \mathsf{verify})$ be an unforgeable signature scheme; the inputtext format for $m$ is:

$$\mathtt{IT} = (ct_m, \sigma) = (ct_m, \mathsf{sign}_{sk}(d \| ct_m)) \tag{2}$$

where $(sk, vk) \leftarrow \mathsf{Sig.kgen}(1^\kappa)$, $\kappa$ is the computational security parameter, and $d$ encapsulates the identities, namely,

$$d = \mathtt{user\_addr} \| \mathtt{contract\_addr} \| \mathtt{func}. \tag{3}$$

Given $\mathtt{IT} = (ct, \sigma)$, before the gcEVM agrees to work with $ct$ it must first check the identity of the sender and then decrypt $ct$ using that sender's key. Specifically, this is done by:

$$m = \begin{cases} \mathsf{dec}_k(c) & \text{if } \mathsf{verify}_{pk}(d \| c, \sigma) = 1 \\ \perp & \text{otherwise} \end{cases} \tag{4}$$

We instantiate $\mathsf{Sig}$ with the `ECDSA` scheme over `secp256k1`.

- **garbledtext™(`GT`).** This datatype is used to securely handle data while in use. Unlike inputtext and ciphertext, garbledtext is in a form that is readily available for manipulation (e.g., making arithmetics over the plaintext it hides) which is made inside the garbled execution environment (see below).[5] Using a binary-projective implementation of a garbling scheme the garbledtext version of a message $m = (m_1, \ldots, m_\ell)$ where $m_i \in \{0, 1\}$, is `gt` of type `GT`, such that

$$\mathtt{gt} = \mathtt{L}_1, \ldots, \mathtt{L}_\ell \tag{5}$$

where $\mathtt{L}_\ell$ is a $\kappa$-bit label (typically $\kappa = 128$). Overall, a garbled-text expands the underlying data by a factor of $\kappa$. This expansion has no effect on the long-term storage requirement of the system, as the lifetime of garbledtexts is short (it is only valid during the execution time of a transaction). The garbledtexts themselves are not utilized by smart contracts and are not appear in their raw form in the gcEVM memory or storage, instead, their *handles* are being used, where a handle of a garbledtext is

[5]We note that in FHE-based solutions there is no distinction in the representation of hidden data at rest and in use.

simply a hash on the list of its labels; namely, $h_{\texttt{gt}} = H(\texttt{gt})$ is the handle of $\texttt{gt}$, where $H$ is a hash function. We instantiate $H$ by Keccak, which is a collision resistant hash function, therefore, the probability of two garbledtexts having the same handle is negligible.

The nodes who take part in evaluation or verification of garbled circuits do obtain the raw representation of garbledtext (the labels). Those nodes store a map of the form $h_{\texttt{gt}} \to \texttt{gt}$; whenever a secure operation is invoked on garbledtext(s) the evaluation nodes perform the actual computation, which mostly results with another garbledtext, whereas verification nodes have the result of the computation and verify its correctness.

The fact that evaluation nodes obtain the raw form of garbledtexts forces the protocol design to assume the worst-case scenario, that the attacker obtains them too (e.g., an attacker who corrupts an evaluation node). Thus, the system must protect itself from *theft of garbledtexts*. For example, suppose that a corrupted evaluation node knows that $\texttt{gt}$ is a garbledtext result of some secure operation in the next block, then it may inject $h_{\texttt{gt}}$ to a function of some contract in the next block, such that the function performs decryption of that garbledtext. Fortunately, the garbling scheme ensures that all garbledtexts are *unpredictable* and are only known to the evaluation nodes at the moment of evaluation and never before.

### 2.6.2 The gcEVM Data-Flow

Before delving into technical details, let us describe the data-flow at a high level, which is also depicted in Figure 1.

In order to preserve security in the course of the execution, the network and the users maintain multiple keys:

- *Network key*: $nk$ is the network symmetric key. This key is being distributively generated on the network's startup (via a key-generation protocol) which results with a key share $nk_i$ to MPC node number $i$. A refresh protocol is applied to the network key, which results with a new key share $nk_i'$ to the nodes, but the secret key $nk$ itself remains the same; such mechanism is intended to protect from adaptive adversaries, who can corrupt a dynamic subset of the parties at every given moment, and thwart accumulation of their power. In addition, a re-generation of the network key would take place from time to tiem, according to the network's policy (e.g., upon accumulation of additional 20% of staked funds); this re-generation protocol would generate a new network key $nk^*$ and re-encrypt all ciphertexts under this new one.

- *User key*: Apart from the secret signing key that users usually maintain in their wallet, the user has a symmetric key $uk$ with which it enters new data to the network. Like the network key $nk$, the user key $uk$ is distributively generated and is secret shared among the network nodes (so node number $i$ holds share $uk_i$). Thus, when a new data is to be entered (e.g., to a function of a smart contract) the user encrypts it and the function asks to decrypt and use it. As will be shortly explained, such a decryption does not reveal the plaintext to the function (or in public in any way), rather, it transform the data into a garbledtext, which enables the function to securely operate on it.

- *Key-retrieval key*: This is a *public-key* that is generated by a user in order to retrieve its symmetric key from the network. This is done by a generation of asymmetric encryption and decryption keys $ek, dk$ by the user, and submitting $ek$ to the network (via a transaction). Upon receiving the encryption key $ek$, the network encrypts the user's symmetric key $uk$ under the temporary public encryption key $ek$, and returns the encryption result to the user. The user then uses the secret decryption key $dk$ to decrypt that message and obtain the symmetric key $uk$ that is also maintained by the network. The user can now use $uk$ in order to enter encrypted data to the network.

The network maintains its own symmetric encryption key as well as a symmetric encryption key for every user that wants to benefit from data privacy in the system. In contrast to a signature key-pair, which is generated at the user and can be used without any on-boarding process, the encryption key for each user is generated by the network and can be used by the user only after on-boarding, which entails a simple query to the system to generate its key (or to retrieve it if it already exists) via the key-retrieval key. Note that the key may be generated prior to the user's query, in cases a smart contract already performed an encryption toward that user (meaning that the smart contract decided that some information should be decipherable by that user only).

For a user to bring encrypted data to the gcEVM, it has to encrypt it using its own symmetric key, and sign it using its own signing key. These two form an inputtext (as detailed above). Once this inputtext reaches a function of a contract, the function first has to verify its authenticity and that the inputtext has landed where the sender (user) really intended it to land; the result of a successful verification will be a garbledtext that is ready to work with (it can serve as an input for secure operations) inside the garbled execution environment (GEE). The verification procedure is according to Equation 4 above.

The above is one path to the GEE; a second path to the GEE would be to 'onboard' a ciphertext, which turns a ciphertext into a garbledtext (both hiding the same plaintext). Ciphertexts reside in the state of each contract and 'belong' to the contract where they reside, meaning that a function on one contract cannot request the onboarding of a ciphertext in another contract, which is critical for the security and privacy of users' data.

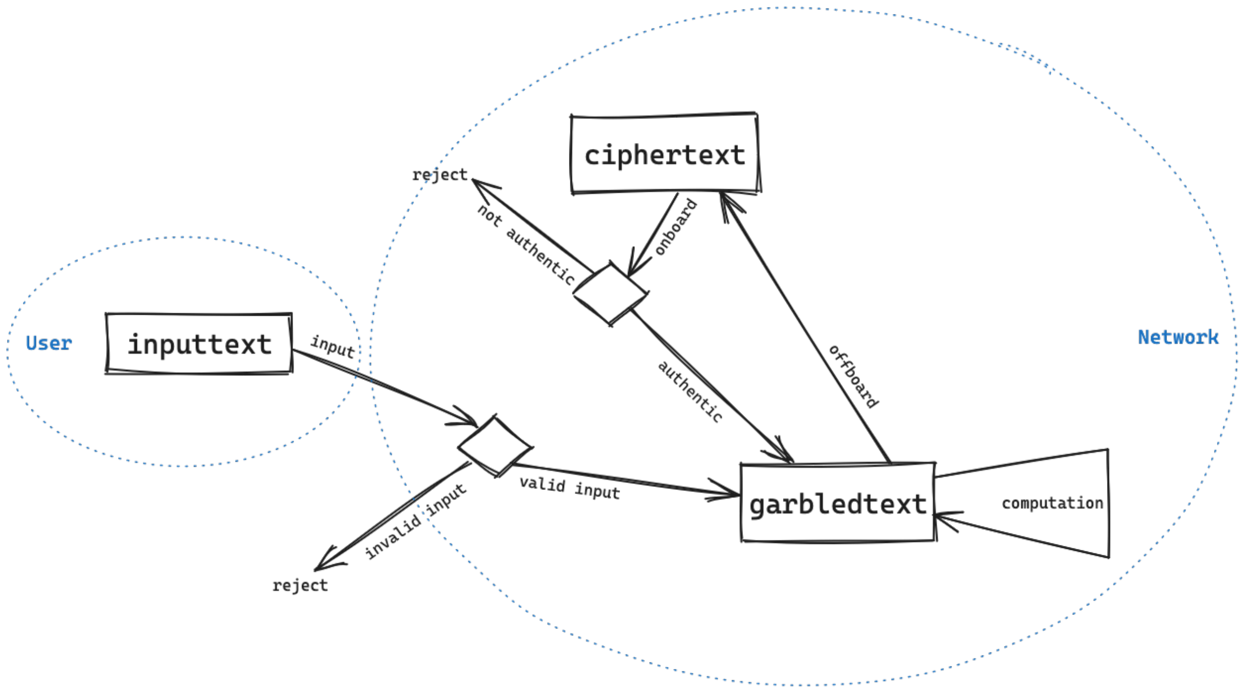

Figure 1: Transition between data types.

## 2.7 Authentic Memory and Storage

It is crucial to ensure the authenticity of all three data types within the operation of the network, safeguarding them against any potential malicious manipulation. Given the transparency of the blockchain, *inputtext* and *ciphertext* are susceptible to malicious copying or unauthorized acquisition. Contrarily, we argue that *garbledtexts* cannot be predicted, copied or obtained outside the transaction's execution.

Generally speaking, the challenge of protecting private data in the context of blockchains mostly deals with ensuring an adequate and tight access control to those ciphertexts. The use of the plaintext behind an inputtext or a ciphertext must be permitted only if this usage complies with the intentions of the contract that is owning or receiving them, and the users who contributed those inputtexts.

In a high level, the system employs three types of protection mechanisms: *authenticated storage*, *authenticated memory* and *garbled execution environment* (GEE$^{\text{TM}}$):

- Authenticated storage protects ciphertexts 'at rest'; each ciphertext is associated with one or more contract addresses in a way that permits onboarding of the ciphertext only in the execution of those contracts; this means that copying a ciphertext from one contract to another is futile. By copying a ciphertext $ct$ we mean either hardcoding the content of $ct$ into another contract, or transmitting it within a transaction as an argument.
- Authenticated memory is the vehicle between the storage and the execution environment. The function first loads the ciphertext from storage into the execution environment memory, from which it may turn into a garbledtext. On the other way around, to store the plaintext value behind a garbledtext to storage, it has to be offboarded first into a ciphertext that resides in memory, from which it is actually stored in the storage.
- Ciphertexts cannot be transferred between contracts, which means it is useless to pass a ciphertext as an argument between different contracts. The way to pass private information between contracts is for the caller contract to turn the ciphertext into a garbledtext first, which puts it in the garbled execution environment, and then pass the garbledtext to the callee contract, this way, the garbledtext is available for secure manipulation by the caller as well. Garbledtexts are protected by the fact that they are random and unpredictable values generated 'on-the-fly'.

**Authenticated Memory & Storage.** Memory refers to the EVM run-time memory, which is stack-based, and storage refers to the EVM persistent storage, which is maintained per address. Recall that only contracts' addresses (i.e., to exclude EOA addresses) are associated with actual storage, whereas EOAs are associated with the their balance and nonce only. For a function to perform some computation on a state variable, it has to first load it from the contract's storage and to know its exact location in the storage. To do that, the function provides the variable's location `loc` to the `sload` opcode, which triggers the EVM to execute it and push the variable's content to the memory stack.[6] On the other way around, for a function to persistently save some value into some state variable, it provides the location `loc` of that variable as well as the value to the `sstore` opcode.

The EVM keeps track of the `depth` of the execution, that is, when an EOA calls some function `func1` on a certain contract `contract1`, the execution of `func1` begins at `depth` = 1; if that function calls function, say `func2`, on another contract, say `contract2`, then `depth` changes to 2 (and changes back to 1 when `func2` returns to `func1`), and so on. Note that function calls within the same contract do not change `depth`.

[6]Location is also known as 'key' in the context of the EVM storage, as the storage is simply a key-value store.

The gcEVM provides an extension to the 'normal' EVM memory and storage operation described above, which we call *authenticated memory* and *authenticated storage*.

The authenticated memory maintains a map of the form

$$\mu : \mathbb{N} \to \texttt{CT}^*,$$

that is, for each execution depth the map maintains all authenticated ciphertexts for that depth. To check whether a ciphertext $ct$ is authenticated for depth $d$, we check if $ct \in \mu(d)$. In our context, a ciphertext may arrive in memory by either loading it from storage via the `sload` opcode or as a result of the `Offboard` mechanism (which, given a garbledtext, returns a ciphertext).

Authenticated storage is maintained in a per-contract basis; each contract, say on address `addr`, is associated with another contract at address $\texttt{addr}'$ that contains only storage (and no bytecode). The relation between `addr` and $\texttt{addr}'$ must be one-to-one, so that a malicious entity would not be able to associate another address $\texttt{addr}''$ to neither `addr` or $\texttt{addr}'$. The associated contract at $\texttt{addr}'$ forms the authenticated storage of the contract at `addr`; this authenticated storage is only accessible from the EVM and not by the contract developer. If a valid ciphertext $ct$ resides at location `loc` of the storage of address `addr`, then the authenticated storage, of address $\texttt{addr}'$, contains $ct$ at the same location `loc`. We must ensure that a user cannot cause this to happen on invalid ciphertexts.

The authenticated memory and storage adhere to the following rules, which are also depicted in Figure 9.

1. *Load from storage.* A ciphertext $ct$ resulting from `sload` applied to a storage at location `loc` that is performed in depth $d$ is first checked against the authenticated storage. If the authenticated storage has $ct$ in location `loc` as well then $ct$ is added to the set $\mu(d)$, otherwise $ct$ is *not* add to $\mu(d)$ (but it is pushed to the normal memory).

2. *Onboard.* When Onboard is invoked by a function at depth $d$ on cipher $ct$, if $ct$ is authenticated for depth $d$ (i.e., $ct \notin \mu(d)$) then it is being translated into a garbledtext and that garbledtext is returned to the caller; otherwise, the execution is reverted.

3. *User input.* As explained above, a user input is encrypted by its own symmetric key and upon verification (of authenticity) it is turned directly into a garbledtext, readily available for secure operations.

4. *Public input.* A contract might want to input some public input into the secure execution environment (GEE), which means that value has to be turned into a garbledtext. This is done via the special opcode 'SetPublic'.

5. *Offboard.* A ciphertext $ct$ resulting from the `Offboard` mechanism (applied to a garbledtext) that is performed by a function in $\texttt{depth} = d$ is added to the set $\mu(d)$, upon which we say that $ct$ is *authenticated for depth* $d$.

6. *Offboard to user.* When a contract wishes to disclose some value only to a specific user, that value is turned from garbledtext into a ciphertext that is encrypted under that user's key (the symmetric key $uk$), and that ciphertext is not considered authenticated (it is placed in memory but is not added to $\mu(d)$).

7. *Decrypt.* When a garbledtext is decrypted, the plaintext value is returned directly to the normal (non-authenticated) memory.

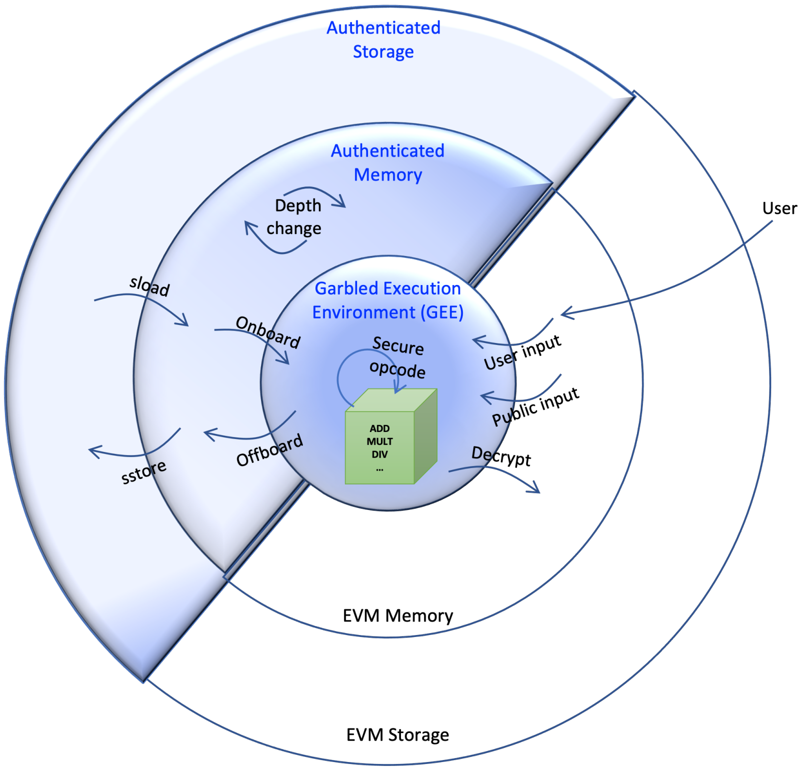

Figure 2: Overview of the gcEVM. The colored part represents the security and privacy extension to the normal EVM, which is represented by the white part.

8. *Store to storage.* When `sstore` is called by a function at `depth` $= d$, to store $ct$ at location `loc`, if $ct \in \mu(d)$ then, in addition to writing $ct$ to location `loc` of the normal storage, write it to location `loc` in the authenticated storage as well, otherwise (if $ct \notin \mu(d)$) write it only to the normal storage, and make sure the authenticated storage at location `loc` is empty.

9. *Cleaning memory.* When `depth` is decreased from $d$ to $d' < d$ (when a function returns, reverts, etc.) the set $\mu(d)$ in the authenticated memory is cleaned. Similarly, when `depth` is increased from $d$ to $d + 1$ (on a function call), we make sure that $\mu(d + 1)$ is empty.

10. *Immobility of ciphertexts.* An authenticated ciphertext cannot transit between depths, namely, calling a function (on another contract) with an argument of type `CT` would deem that ciphertext invalid (i.e., it will not be considered as authenticated for the new depth); similarly, when returning an argument of type `CT` to a caller function from another contract the returned value would not be considered authenticated. The right way to move such private values is by using the `GT` type.

**The Garbled Execution Environment (GEE$^{\text{TM}}$).** As explained, a garbledtexts may be generated from an inputtext or a ciphertext, *only* upon confirming their authenticity. It is crucial to note that garbledtexts bear significance only in the course of the execution of the transaction. Additionally, their inherent safety is derived from the negligible likelihood of predicting them, given that they are formed of random values that are revealed to the parties only at the very moment of execution. This property allows us to treat them with ease rather than keep tracking them across function calls. Since the next batch of transactions to run is fixed before the garbledtexts are revealed, it is not possible for a user or for a contract to hardcode a garbledtext in their transaction or state, as garbledtexts are unpredictable, furthermore, these garbledtexts become useless once the execution of the transaction is completed; the garbledtexts for the next transactions batch would be completely fresh.

## 3 Decentralizing Sequencers in COTI V2

### 3.1 Concept and Rationale

The evolution of blockchain technology has ushered in the need for scalable and secure layer 2 solutions. COTI V2 aims to be at the forefront of this evolution by introducing a decentralized sequencer model. This model is inspired by shared sequencer frameworks, which have shown promise in enhancing network security and reducing centralization risks.

A sequencer in a blockchain network is responsible for ordering transactions before they are finalized on the blockchain. Centralized sequencers, while efficient, pose risks including single points of failure and potential censorship. Decentralizing the sequencer function distributes these responsibilities across multiple participants, thereby enhancing security, redundancy, and resistance to censorship.

### 3.2 How It Works

The decentralized sequencer model for COTI V2 involves the following key components:

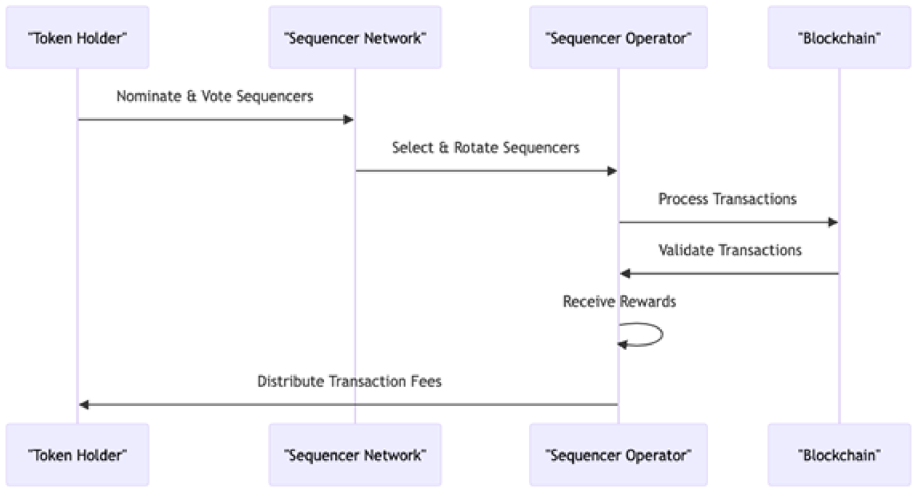

Figure 3: Decentralized sequencer model

- Shared Sequencer Network: A cluster of sequencers that operate in a rollup-agnostic manner. This network ensures that valid transactions are included in the blockchain within a finite time frame, providing the necessary liveness and censorship resistance.

- Operator Selection and Rotation: Sequencer operators are selected through a transparent and fair consensus mechanism, possibly utilizing a Proof-of-Stake (PoS) model. The right to sequence transactions is rotated among operators to prevent centralization and ensure fairness.

- Staking and Slashing Mechanisms: Operators are required to stake tokens as a form of security deposit. Misbehavior or failure to perform duties results in slashing of their stake, aligning operators' interests with the network's integrity.

- Transaction Processing: Transactions are processed off-chain by the sequencers and batched together before being finalized on the blockchain. This approach maintains the scalability benefits of layer 2 solutions while leveraging the security of the underlying layer 1 blockchain.

## 3.3 Implementation

**Consensus Protocol:**

In COTI V2's decentralized sequencer model, the modified Practical Byzantine Fault Tolerance (PBFT) algorithm plays a critical role. This consensus protocol operates in several key steps to ensure secure, fair, and transparent sequencer operation:

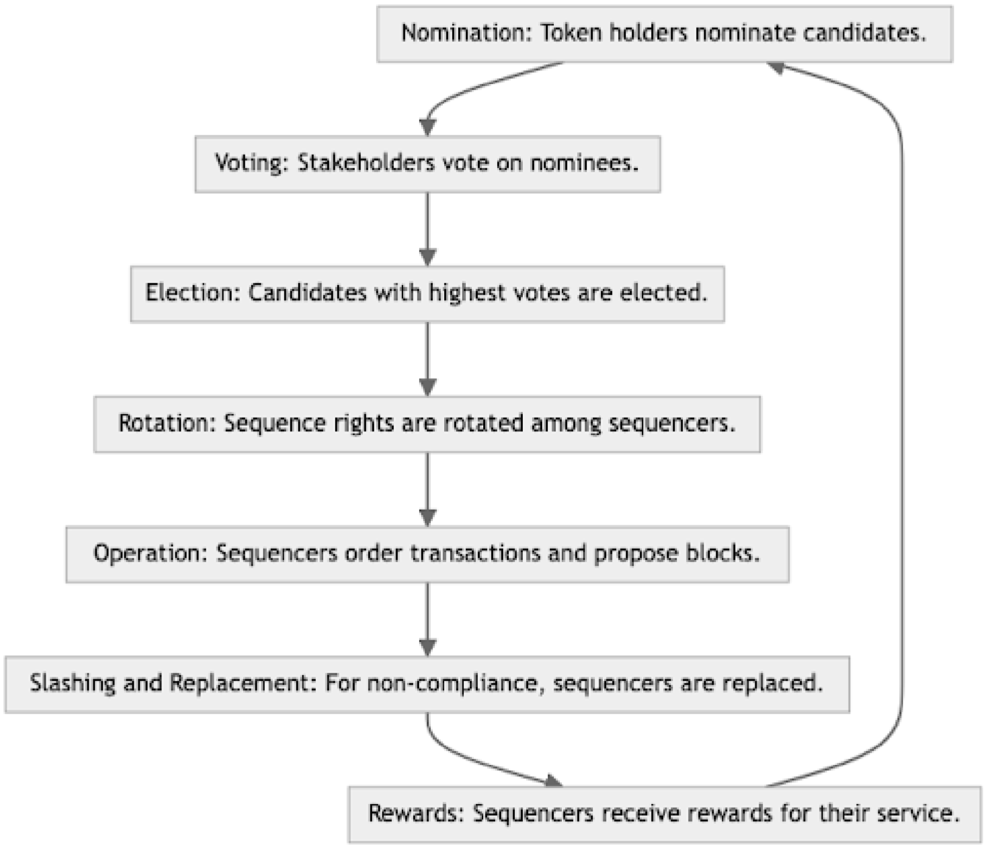
Nomination: Token holders nominate candidates.
Voting: Stakeholders vote on nominees.
Election: Candidates with highest votes are elected.
Rotation: Sequence rights are rotated among sequencers.
Operation: Sequencers order transactions and propose blocks.
Slashing and Replacement: For non-compliance, sequencers are replaced.
Rewards: Sequencers receive rewards for their service.

Figure 4: Consensus protocol steps

1. **Nomination:** Token holders nominate candidates for the sequencer role. Any token holder can become a nominee by locking a specified amount of tokens as collateral, demonstrating their commitment.

2. **Voting:** Stakeholders vote on the nominated candidates, with the voting power of each stakeholder proportional to their token holdings. This ensures that the selection process is democratized and reflects the preference of the network's participants.

3. **Election:** After the voting period concludes, the candidates with the highest votes are elected as sequencers for a predetermined term. This term is designed to be short enough to allow regular rotation but long enough to ensure stability.

4. **Rotation:** The right to sequence transactions is rotated among the elected sequencers based on a predefined schedule. This rotation can be deterministic, based on the order of election, or randomized to enhance security.

5. **Operation:** During their term, sequencers order transactions and propose blocks to the network. Their operations are continuously monitored for compliance with the network's rules, ensuring reliability and integrity.

6. **Slashing and Replacement:** If a sequencer acts maliciously or fails to perform their duties, they can be slashed, meaning a portion of their staked tokens is forfeited. A new election process can be initiated to replace the slashed sequencer.

7. **Rewards:** Sequencers receive rewards for their service, typically in the form of transaction fees or newly minted tokens. This incentivizes participation and ensures that sequencers act in the network's best interest.

**Smart Contracts for Staking and Slashing:**

The implementation of Smart Contracts for Staking and Slashing introduces a rigorous and automated framework designed to maintain integrity and accountability among sequencers within a blockchain network. This process begins with Stake Deposit, where operators commit a certain amount of $COTI tokens into a staking contract to qualify as sequencers, establishing a tangible investment in their role. The mechanism of Stake Locking further cements this commitment by ensuring the stake remains untouchable for a predetermined period, effectively binding operators to their obligations.

Behavior Monitoring plays a crucial role in this ecosystem, employing automated systems to vigilantly oversee sequencer actions, ensuring adherence to the network's protocol by evaluating criteria such as block production fidelity and transaction fairness. In the event of protocol deviation, Violation Detection mechanisms promptly identify and log offenses, potentially triggering automated alerts or necessitating manual reports from network participants. Crucially, Consensus on Violation may be required for certain breaches, safeguarding against unwarranted penalties by ensuring any slashing decision is collectively agreed upon, thereby preventing misuse or errors in violation assertions.

Upon verified breach of protocol, Slashing Execution is carried out, with the offending party's stake being partially or fully confiscated, the severity of which is predefined or calculated based on the infraction's gravity. However, an Appeal Process is available, allowing operators to challenge a slashing verdict through a structured governance mechanism, presenting evidence to contest the violation. For those who uphold the network's standards throughout their term, Stake Return ensures the reimbursement of their commitment, with the contract facilitating the smooth reclamation of funds. Additionally, Reward Distribution recognizes and incentivizes exemplary participation, with rewards allocated based on factors such as staked amount, duration of commitment, and overall network contribution, underlining the system's holistic approach to fostering a secure, efficient, and accountable blockchain environment.

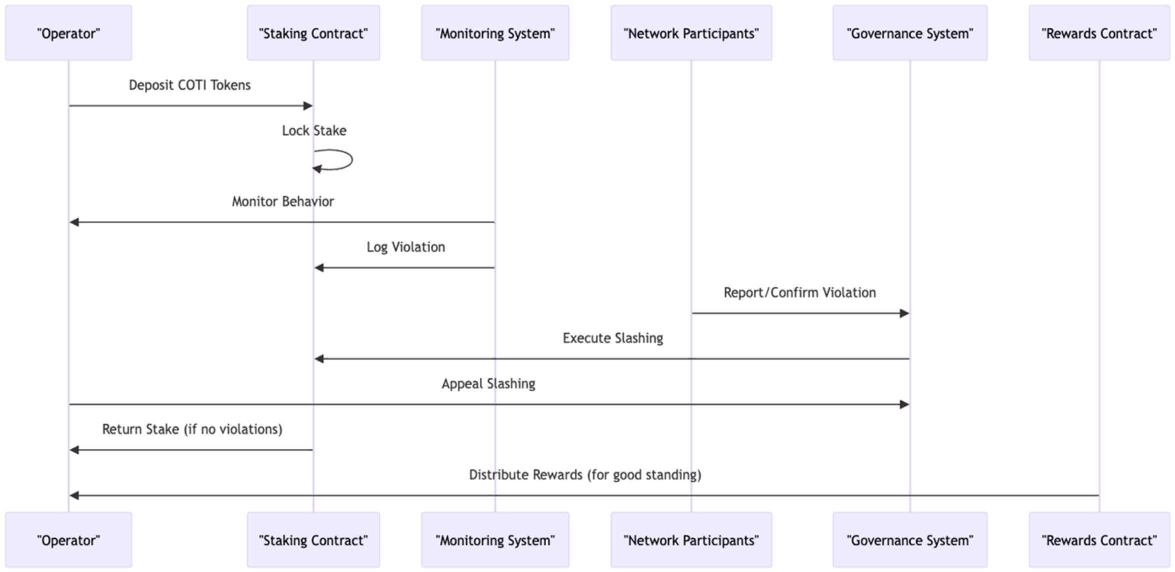

Figure 5: Staking and validation process

1. **Stake Deposit:** Operators deposit a predefined amount of COTI tokens into the staking contract to be eligible as sequencers. This contract records the staked amount and the identity of the depositor.

2. **Stake Locking:** Once deposited, the stake is locked for a specified duration to ensure commitment. The contract prevents withdrawal of the stake during this period, ensuring that sequencers have skin in the game.

3. **Behavior Monitoring:** Sequencer actions are monitored against the network's protocol rules. This could involve automated systems tracking block production fidelity, transaction inclusion fairness, and response times.

4. **Violation Detection:** If a sequencer violates predefined rules (e.g., censoring transactions, failing to produce blocks on time), the violation is detected and logged by the contract. This could trigger an automatic alert or require manual reporting by network participants.

5. **Consensus on Violation:** For certain types of violations, a consensus among other network participants may be required to confirm the offense. This ensures that the slashing mechanism is not triggered by false accusations or errors.

6. **Slashing Execution:** Upon confirmation of a violation, the contract partially or fully slashes the offender's stake. The exact amount can be predetermined in the contract or calculated based on the severity of the violation.

7. **Appeal Process:** Operators may have the option to appeal a slashing decision through a governance mechanism, presenting evidence to dispute the detected violation.

8. **Stake Return:** For operators who complete their term without violations, their stake is returned post

the locking period. The contract handles the automatic release of funds.

9. **Reward Distribution:** Operators in good standing may receive rewards, distributed through another smart contract, which could factor in the staked amount, term length, and network performance.

This structured approach ensures that staking and slashing mechanisms enforce network integrity and performance, deter malicious behavior, and promote a healthy, decentralized ecosystem. Each step involves specific smart contract functions that are transparently executed on the blockchain, providing a secure and automated system for managing sequencer operations in COTI V2.

**Infrastructure for Shared Sequencing:**

The development of an Infrastructure for Shared Sequencing represents a critical endeavor in enhancing the operational efficiency and interoperability of blockchain networks. At its core, this initiative involves the meticulous design of a software framework that sequencers will employ, ensuring a robust architecture for transaction processing, batch creation, and seamless communication across various rollups and the sequencer network. Key to this framework is the establishment of interoperability protocols, which facilitate smooth interactions and standardized transaction processing across the ecosystem. Additionally, paramount importance is placed on implementing stringent security measures to safeguard against unauthorized access and maintain transaction integrity, alongside integrating scalability solutions to adeptly manage high volumes of transactions. This comprehensive approach not only fortifies the blockchain infrastructure but also significantly propels its capacity to support a growing array of applications.

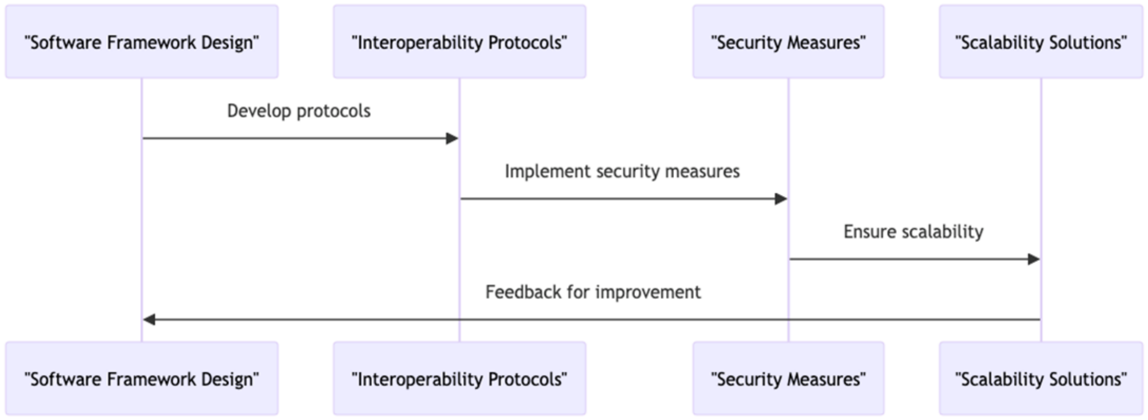

Figure 6: Shared sequencer framework

1. **Software Framework Design:** The initial step involves outlining the software architecture that sequencers will utilize. This includes defining the protocols for transaction processing, batch creation, and communication between different rollups and the sequencer network.

2. **Interoperability Protocols:** Developing protocols that ensure seamless interaction between the shared sequencer infrastructure and multiple rollups is crucial. This involves creating standardized APIs and data formats that enable transactions to be universally processed and understood across the ecosystem.

3. **Security Measures:** Implement robust security protocols within the software to protect against unauthorized access and ensure the integrity of transaction processing. This includes encryption, access

controls, and continuous security audits.

4. **Scalability Solutions:** Incorporate scalability solutions, such as sharding within the sequencer network or adopting layer 1 scalability technologies, to handle high transaction volumes efficiently.

**Operation Phase:**

The Operation Phase of deploying a Shared Sequencing Infrastructure marks a pivotal transition from theoretical design to practical application within the blockchain network. This phase kicks off with the deployment and integration of sequencer software, ensuring its seamless incorporation with existing rollups through extensive compatibility and performance testing. Following deployment, sequencers commence the processing and batching of transactions, utilizing sophisticated algorithms to enhance network throughput and reduce latency by prioritizing transactions on various criteria. A significant focus is also placed on enabling fluid cross-rollup communication, thereby knitting together a unified and interoperable blockchain ecosystem that facilitates transactions across different rollups with ease. Moreover, an ongoing commitment to monitoring and maintenance ensures the sequencer network remains at the pinnacle of performance, security, and reliability, with continual updates and optimizations driven by network analytics and emerging technological trends. This operational vigilance guarantees the infrastructure's adaptability and enduring relevance in the fast-evolving blockchain landscape.

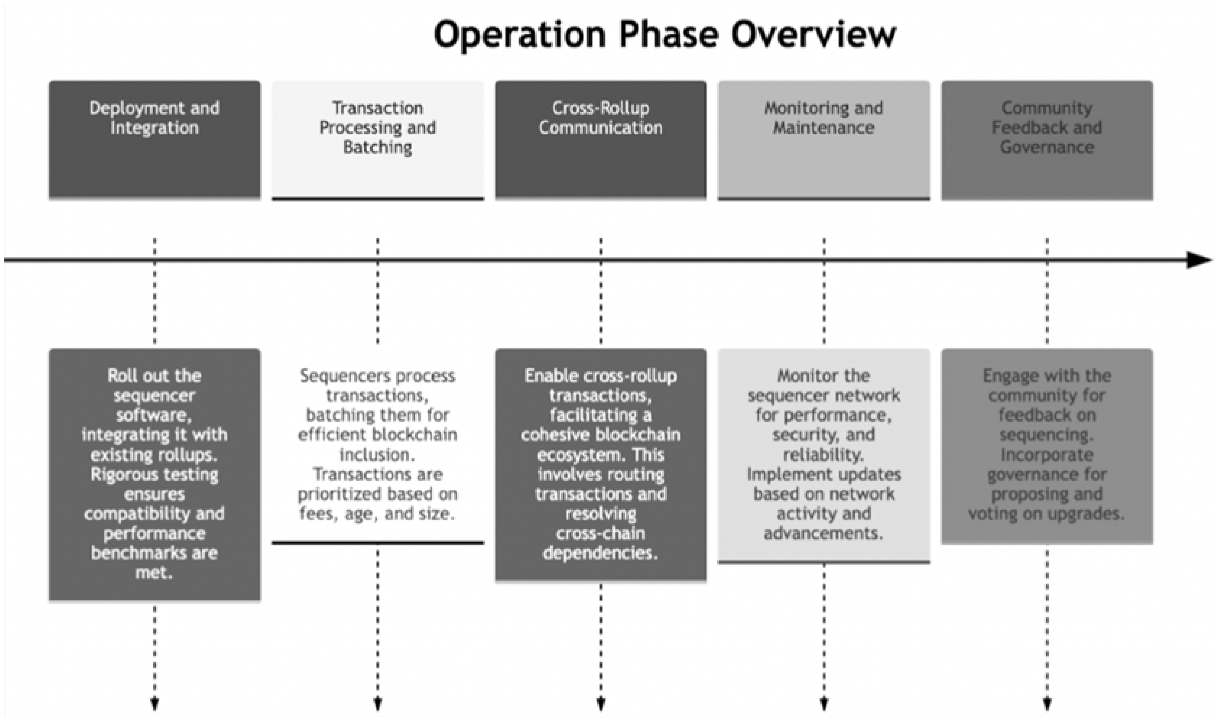
Operation Phase Overview
Deployment and Integration
Transaction Processing and Batching
Cross-Rollup Communication
Monitoring and Maintenance
Community Feedback and Governance
Roll out the sequencer software, integrating it with existing rollups. Rigorous testing ensures compatibility and performance benchmarks are met.
Sequencers process transactions, batching them for efficient blockchain inclusion. Transactions are prioritized based on fees, age, and size.
Enable cross-rollup transactions, facilitating a cohesive blockchain ecosystem. This involves routing transactions and resolving cross-chain dependencies.
Monitor the sequencer network for performance, security, and reliability. Implement updates based on network activity and advancements.
Engage with the community for feedback on sequencing. Incorporate governance for proposing and voting on upgrades.

Figure 7: Operation phase overview

1. **Deployment and Integration:** Roll out the sequencer software across the network, integrating it with existing rollups. This phase involves rigorous testing to ensure compatibility and performance benchmarks are met.

2. **Transaction Processing and Batching:** Sequencers begin processing individual transactions, batch-

ing them for efficient blockchain inclusion. Batching algorithms prioritize transactions based on criteria such as fees, age, and size to optimize network throughput and minimize latency.

3. **Cross-Rollup Communication:** Enable cross-rollup transactions through the shared sequencer infrastructure, facilitating a cohesive and interoperable blockchain ecosystem. This involves routing transactions between rollups and resolving any cross-chain dependencies or conflicts.

4. **Monitoring and Maintenance:** Continuously monitor the sequencer network for performance, security, and reliability. Implement updates and optimizations based on observed network activity and technological advancements.

This detailed approach ensures that the shared sequencing infrastructure for COTI V2 is built on a foundation of security, scalability, and interoperability. By focusing on comprehensive software development and operational protocols, COTI V2 aims to achieve a decentralized, efficient, and user-friendly transaction processing network.

### 3.4 Network Fee Structure

COTI V2 utilizes a dynamic fee model designed to strike a balance between affordability and network efficiency. This approach aims to keep fees lower than the Ethereum mainnet while ensuring transactions are processed promptly. Let's delve into the factors that determine network fees:

- Demand-Based Pricing: Similar to Ethereum's gas fees, COTI V2 employs a demand-driven pricing model. When network congestion rises due to increased transaction volume, fees automatically adjust upwards. Conversely, during periods of low demand, fees decrease. This mechanism ensures resource allocation reflects actual network usage.

- Decentralized Fee Rules: The COTI V2 protocol establishes the core principles for fee calculation. These pre-programmed rules autonomously adjust fees in response to network demand and usage patterns, eliminating the need for manual intervention.

- User-Defined Maximum Fees: Users retain some control over transaction costs, particularly during peak demand periods. They can specify the maximum fee they are willing to pay for their transaction to be included in the next block. Transactions with higher fees are generally prioritized for inclusion by sequencers. It's important to note that some fees may be denominated in the native $COTI token. Furthermore, fees can be tailored to meet the specific needs of network functionalities and resource demands, including support for certain privacy features and services, smart contract fees, and so on. The fees can be adjusted to facilitate access to advanced Zero-Knowledge (ZK) services within the network. These services may include specialized ZK computations, private information retrieval, or access to secure data feeds.

- Sequencers and Fee Competition: Sequencers play a vital role by ordering transactions and submitting them to the Ethereum mainnet. While sequencers have some discretion in pricing transactions, the competitive market and pre-defined protocol rules contribute to ensuring fees remain fair and competitive.

- Layer 1 Costs: A portion of the network fee covers the cost of submitting transaction data to the Ethereum mainnet (Layer 1). COTI V2, as a Layer 2 solution, batches multiple transactions together for submission to Layer 1, minimizing these costs. However, the overall fee structure incorporates these Layer 1 gas prices, which are influenced by Ethereum network activity.

In essence, COTI V2 employs a decentralized approach to network fee determination. Protocol rules govern the core framework, while fees dynamically adjust based on network demand and the cost of interacting with the Ethereum mainnet. Sequencers and users operate within this system, influencing transaction prioritization and costs through market dynamics and individual choices, respectively.

## 3.5 Governance Model

### Framework Development

The Operation Phase of deploying a Shared Sequencing Infrastructure marks a pivotal transition from theoretical design to practical application within the blockchain network. This phase kicks off with the deployment and integration of sequencer software, ensuring its seamless incorporation with existing rollups through extensive compatibility and performance testing. Following deployment, sequencers commence the processing and batching of transactions, utilizing sophisticated algorithms to enhance network throughput and reduce latency by prioritizing transactions on various criteria. A significant focus is also placed on enabling fluid cross-rollup communication, thereby knitting together a unified and interoperable blockchain ecosystem that facilitates transactions across different rollups with ease. Moreover, an ongoing commitment to monitoring and maintenance ensures the sequencer network remains at the pinnacle of performance, security, and reliability, with continual updates and optimizations driven by network analytics and emerging technological trends. This operational vigilance guarantees the infrastructure's adaptability and enduring relevance in the fast-evolving blockchain landscape.

1. **Proposal Submission:** Token holders can submit proposals regarding sequencer network operations, including protocol upgrades, operator selection criteria, and dispute resolution mechanisms. This ensures every stakeholder has a voice in the network's evolution.

2. **Discussion Platforms:** Dedicated forums and discussion boards will be established for the community to debate and refine proposals. This allows for transparent deliberation and collective intelligence to shape the network's policies.

### Decision-Making Process

1. **Voting System:** Implement a decentralized voting mechanism where COTI V2 token holders cast their votes on proposals. Votes are weighted based on token ownership, ensuring that those with a larger stake in the network have a proportionate influence.

2. **Quorum Requirements:** Define quorum requirements for different types of decisions to ensure that significant changes require a broad consensus, while routine matters can be decided with a simpler majority.

3. **Operator Selection:** For sequencer operator elections, a specific voting process will be established, allowing token holders to choose among candidates based on predefined criteria, ensuring the most competent and trustworthy operators are selected.

### Implementation and Enforcement

1. **Smart Contracts:** Deploy smart contracts to automate the governance process, from proposal submission to vote tallying and implementation of decisions. This reduces the potential for human error

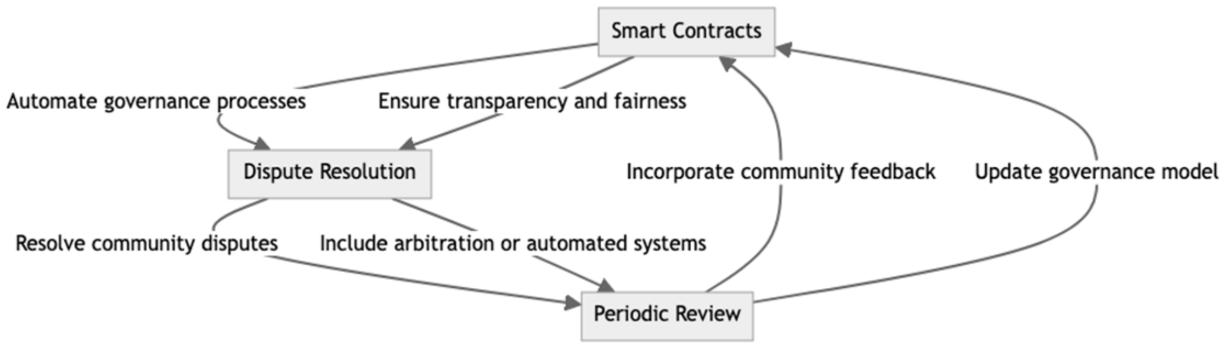
Smart Contracts
Automate governance processes
Ensure transparency and fairness
Dispute Resolution
Incorporate community feedback
Update governance model
Resolve community disputes
Include arbitration or automated systems
Periodic Review

Figure 8: Overview of governance model

and ensures transparency and fairness.

2. **Dispute Resolution:** Establish a clear and fair mechanism for resolving disputes within the community. This could include arbitration by elected community members or an automated system based on smart contract logic.

3. **Periodic Review:** Regularly review and update the governance model to address emerging challenges and incorporate community feedback. This ensures the model remains effective and relevant as the network and its ecosystem evolve.

This governance model for COTI V2 is designed to empower token holders, ensuring that the sequencer network evolves in a way that is beneficial to all stakeholders. By leveraging blockchain technology and smart contracts, the model aims to provide a transparent, fair, and efficient framework for community-driven decision-making.

### 3.6 Conclusion

Decentralizing the sequencer in COTI V2 represents a significant step towards achieving a scalable, secure, and community-driven blockchain network. Through careful design and implementation of the shared sequencer model, COTI V2 can set a new standard for decentralized transaction processing in the blockchain space.

## 4 Tokenomics

### 4.1 The COTI Token

COTI V2 is an EVM-compatible L2, which requires a native token in order to fully realize the potential of the network. This token serves as the lifeblood of the ecosystem, facilitating transactions, securing the network, incentivizing operators, and providing the means to access and utilize the advanced privacy features of COTI V2. In essence, the token is the cornerstone of this transformative solution, bridging the gap between cutting-edge privacy technology and the practical functionality that users and developers require within the ecosystem.

The current $COTI token holds a central and indispensable role within the ecosystem since the inception of the COTI mainnet in 2019. Serving as a foundational native cryptocurrency, it plays a vital part in covering transaction fees and managing various ecosystem functions. COTI recognizes a great value proposition both for COTI token holders and to the COTI V2 future community in retaining the tokens central position in the new economy, enabling it to grow in importance and relevance as the network's capabilities expand.

As such, it needs to undergo a comprehensive technological upgrade, prompting the need for an improved token infrastructure that will gradually replace the current native one. Instead of introducing an entirely new token with a new and unrelated economy, COTI has opted for a migration plan that secures and augments the token's usage within the COTI ecosystem.

## 4.2 COTI V2 Evolution

COTI V2 introduces a fundamentally different economic framework compared to its predecessor, the existing COTI network. This evolution is driven by the network's unique features and its compatibility with EVM, allowing developers, DAOs, and dApps to seamlessly integrate and utilize the network.

COTI V2 embraces an open, public, permissionless network model. Such openness, coupled with proper incentives, paves the way for a diverse array of developers and entities to harness its capabilities, fostering innovation and growth. With EVM compatibility, developers arent required to learn new or obscure coding languages, easing the transition to building dApps within the COTI mainnet.

The economy in COTI V2 is based on the combination of its advanced underlying technology and the existing COTI community, which will create a thriving and valuable ecosystem. The new network will enable new benefits for network participants, contributors, developers and users. Capitalizing on its advanced features and new benefits, COTI V2 has the capacity to drive an upsurge in transaction volumes, leading to increased revenue through transaction fees. Additionally, it lays the foundation for the exploration of fresh business models and revenue streams, stemming from heightened utility and the introduction of new capabilities like smart contract deployments.

## 4.3 Monetary policy

Starting 2025, coinciding with the launch of the new mainnet, COTI V2 will initiate the minting of new tokens every 120 hours (period), featuring a gradually reducing inflation rate that adjusts according to the network's changing demand and usage. The initial expansion of the total supply will occur at a periodic rate of 0.45%, which will subsequently decrease periodically by 0.55% for a decade. Following this period, the inflation rate is expected to stabilize at a rate comparable to the terminal value, though it remains adjustable through the network's governance mechanisms.

Transaction fees collected in $COTI will be pooled into a special Treasury, governed by the community, which will decide on how to utilize these funds. Choices include redistributing these fees as rewards to network participants or implementing a deflationary tactic by destroying some of the tokens. The fee and potential token burn mechanisms suggest the possibility of limiting total supply growth, potentially leading to a deflationary outcome. Under such conditions, the token supply could diminish, illustrating the protocols capacity for adjustment and long-term viability. Figure 9 outlines various potential paths for token supply changes, influenced by network activity and the associated fees.

Out of every new token minted in the network, 58% of it will be dedicated to reward holders, stakers and operators as well as other network participants, encouraging active participation and securing the network. Since COTI V2 will use Proof of Stake, staking rewards and fees will be distributed approximately pro-rata to current holdings as long as everyone in the network is staking. Therefore, assuming high staking partici-

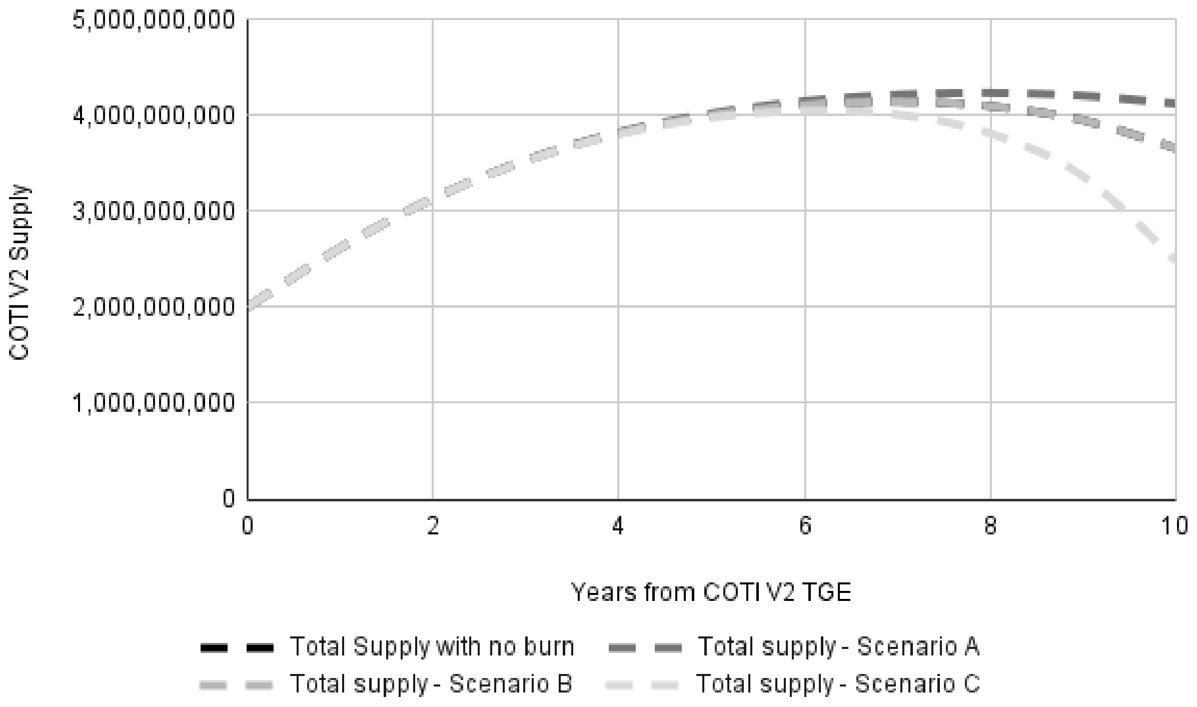

Figure 9: Total supply scenarios

pation, the nominal inflation attributed to this portion will be canceled out by a nominal return denominated in the protocol, ensuring that accounts proportional holdings of the currency remain constant.

## 5 Applications

The COTI V2 framework, employing Garbled Circuits technology, presents an advancement in privacy mechanisms, offering enhancements that result in a computational speed increase by a factor of 1,000 and a reduction in resource intensity by a factor of 250. This development facilitates operation across diverse devices, yielding an improved user experience compared to existing privacy solutions. Distinguished from the constraints of Zero-Knowledge (ZK) based approaches, Garbled Circuits enable transaction processing involving multiple stakeholders. The technology's capacity for secure data utilization in verification and computational tasksparticularly at this scaleheralds novel applications not only within Web3 ecosystems but also across Web2 sectors.

This chapter outlines what are considered pivotal domains where COTI V2 would focus in order to catalyze transformative impacts. It is anticipated that additional sectors and innovative concepts will emerge from the community, contributing to the ecosystem's evolution.

### 5.1 Confidential DeFi

DeFi has redefined the financial industry's landscape, introducing new potential functionalities. Nonetheless, the inherent transparency of on-chain activities necessitates a compromise on privacy, deterring institutional engagement due to regulatory concerns. COTI V2 introduces confidential transactions, enhancing privacy and security for existing services while complying with regulatory standards, thereby expanding DeFi's innovation potential. It addresses long-standing vulnerabilities in Ethereum DeFi, such as exploitation through Miner Extractable Value (MEV), by encrypting transaction details, thus preventing opportunistic behaviors by MEV bots and frontrunners.

The forthcoming COTI V2 Devnet will provide developers with tools to enhance existing decentralized applications (dApps) with advanced data security features or to innovate new DeFi-centric applications previously unfeasible.

## 5.2 Confidential Transactions for Payments, Stablecoins, CBDC and RWA

The traditional financial ecosystem emphasizes transaction confidentiality to build trust and encourage user participation. Mirroring this attribute in Web3 is crucial for its mainstream adoption. COTI V2 ensures transaction confidentiality while maintaining compliance with regulatory frameworks for digital and real-world assets, offering a mechanism for confidential payments that preserves the transparency of fund flows yet encrypts transaction specifics.

## 5.3 Confidential Machine Learning and On-Chain Sensitive Data Management

In the current context of paramount importance placed on data management and privacy, COTI V2 introduces a framework that enables Artificial Intelligence/Machine Learning (AI/ML) models to be trained on sensitive data without compromising the anonymity of individuals. Large Language Models (LLMs) such as ChatGPT depend on substantial volumes of data to enhance their service capabilities. This necessity raises concerns regarding the protection of user privacy and intellectual property (IP). Utilizing Privacy-Preserving Machine Learning (PPML) facilitated by cryptographic methods like garbled circuits (GC), COTI V2 ensures that training on these data does not infringe upon the privacy of data subjects, thus paving the way for new business models.

One example is federated learning, a collaborative approach involving multiple stakeholders, each possessing unique datasets. The limited utility of these datasets due to their size or diversity can be overcome by pooling them together, resulting in a collective data pool that enables the development of more accurate machine learning models.

Furthermore, organizations possessing advanced machine-learning models derived from their proprietary data and seeking to provide these models as a service represent another application scenario. An organization with a model that can, for instance, differentiate between images of dogs and cats with high precision might offer this predictive capability to other entities lacking the resources for similar model development. By deploying such a model on a blockchain platform ("on-chain"), the organization can make its classification service available while safeguarding the privacy of the model's IP and the data being classified. This mechanism, referred to as private inference, allows end-users to access the model's predictive functionalities without the need to publicly disclose their data.

These instances illustrate the initial applications of integrating privacy-preserving techniques with machine learning. As technological developments in this domain continue to advance, the scope and impact of these applications are anticipated to broaden, highlighting the significance of privacy-preserving methodologies in the evolution of AI/ML capabilities.

### 5.4 Dynamic Decentralized Identification (DyDID)

COTI V2 facilitates a paradigm where identity verification and personal data management are executed without exposing actual data to third parties. Users maintain control over their information while fulfilling Know Your Customer (KYC) requirements. This framework allows for secure, privacy-preserving interactions between digital identities and dApps, enabling trustless, regulated environments for global service applications without compromising sensitive data.

The advent of COTI V2, with its state-of-the-art garbled circuit technology, addresses the critical privacy challenges impeding Web3's broader adoption. It offers superior solutions to the privacy issues prevalent in Web2 industries, presenting substantial opportunities for business innovation. Developers worldwide are invited to participate in the upcoming COTI V2 Devnet, contributing to the evolution of a more private, secure, and user-friendly Web3 ecosystem.

## 6 Conclusion

In the conclusion of this research document, it is observed that the development of COTI V2 represents a pivotal advancement in the realm of blockchain privacy mechanisms and system efficiency. Through the deployment of an innovative Layer 2 privacy enhancement for Ethereum, COTI V2 introduces a significant enhancement to confidentiality within blockchain transactions. Utilizing advanced cryptographic methods, this development not only increases security measures but also expands the functional scope for decentralized applications, potentially accelerating innovation and adoption within the Web3 domain.

The application of Garbled Circuits alongside a redesigned tokenomics framework demonstrates COTI's dedication to fostering a secure, scalable, and user-focused blockchain ecosystem. This strategic evolution invites a diverse array of stakeholders, including developers, enterprises, and users, to engage with the emergent opportunities in Web3, characterized by enhanced privacy and reliability.

Therefore, COTI V2 should be viewed as a substantial shift in the approach to blockchain technologies, emphasizing its potential to redefine the standards of privacy, efficiency, and user agency in the digital landscape. The successful integration and wider adoption of COTI V2 could serve as a critical determinant in the future direction of blockchain privacy, establishing a new norm for security, operational efficiency, and user empowerment across the sector.

## References

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796. ACM, 2012.

[BMP22] Constantin Blokh, Nikolaos Makriyannis, and Udi Peled. Efficient asymmetric threshold ECDSA for mpc-based cold storage. *ePrint Archive*, 2022.

[BSW06] Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational diffie-hellman. In *PKC*, pages 229–240. Springer, 2006.

[KL20] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, 2020.

[SPW07] Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. In *CT-RSA*, pages 357–371. Springer, 2007.

[Woo23] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Technical report, Ethereum Foundation, 2023.