# Scalable Private World Computer via Root iO

**Application-Agnostic iO and Our Roadmap for Making It Practical**

Sora Suegami*[1] and Enrico Bottazzi*[1]

[1]Machina iO, Ethereum Foundation

November 29, 2025

**Abstract**

Ethereum has established itself as a world computer, enabling general-purpose, decentralized, and verifiable computation via smart contracts on a globally replicated state. However, because all computations and state are public by default, it is fundamentally unsuitable for confidential smart contracts that jointly process private data from multiple users. This motivates the notion of a private world computer: an ideal future form of Ethereum that preserves its integrity and availability guarantees while supporting such confidential smart contracts. Prior constructions based on implementable cryptographic primitives such as fully homomorphic encryption (FHE) inevitably rely on committees that hold secret shares and perform computations using those shares, a capability that is not provided by today's Ethereum validators. We cannot simply modify the Ethereum protocol so as to shift the committees role onto the Ethereum validators, because the computational and communication costs borne by the committee grow with the demand for confidential smart contracts, forcing higher hardware requirements for participation, undermining decentralization, and increasing the risk of malicious collusion. Hence, there remains a fundamental trade-off between committee decentralization and scalability for confidential smart contracts.

In this position paper, we make two contributions toward a scalable private world computer. First, we show how indistinguishability/ideal obfuscation (iO), combined with FHE and succinct non-interactive arguments of knowledge (SNARK), yields a private world computer that, after a one-time obfuscation process, introduces no additional ongoing trust assumptions beyond Ethereums validators, incurring no additional overhead for validators to process confidential smart contracts compared to public smart contracts. In this design, a single application-agnostic obfuscated circuit, called root iO, suffices to realize arbitrary confidential

*Equal contributions.

smart contracts. The outputs of root iO can be verified on-chain at a cost comparable to signature verification, and the obfuscation process can be distributed among multiple parties while remaining secure as long as at least one party is honest. As the second contribution, we outline our roadmap toward a practical implementation of root iO. Assuming that the underlying assumptions of our lattice-based iO construction remain secure, the remaining missing pieces are technically concrete: namely, practical implementations of verifiable FHE and of homomorphic evaluation of a pseudorandom function (PRF) and SNARK verification over key-homomorphic encodings, which together would allow us to implement root iO without incurring prohibitive overhead.

## 1 Introduction

Ethereum has established itself as a *world computer*, a decentralized platform capable of executing arbitrary programs, called smart contracts, on the Ethereum virtual machine, and of achieving global consensus on a verifiable, timestamped shared state. To achieve these two capabilities, all computations and all state data are public by default. This fundamental lack of privacy severely limits the scope of decentralized applications, preventing the creation of any system that relies on computation over private data coming from multiple parties.

The ambition to solve this challenge has prompted us to introduce the concept of a *private world computer* as an ideal future form of Ethereum. Such a system extends the capabilities of Ethereum to support arbitrary computation over encrypted data, enabling applications to interact with each other's private state data. Smart contracts specify the application logic executed on ciphertexts, as well as the access policies governing their evaluation and decryption.

We aim for any smart contract on the private world computer to achieve integrity (tamper-resistance), availability (censorship-resistance), and additionally, confidentiality (privacy), at the same security level as today's public smart contracts on Ethereum. Since the security of Ethereum is primarily derived from the decentralization of its validators, any new functionality added to Ethereum to support confidentiality should neither compromise this decentralization nor introduce new trust assumptions in parties that are likely to be less decentralized than the validators.

Unfortunately, existing solutions for confidential smart contracts—including those based on secret-sharing multiparty computation (MPC) or fully homomorphic encryption (FHE)—are not ideal for realizing a private world computer. This is because they rely on ad-hoc trusted parties whose required capabilities are not compatible with today's Ethereum validators; we refer to these parties as committees in this paper to distinguish them from the validators. These committees must hold secret shares specific to the underlying primitive and perform computations using those shares.

We cannot simply modify the Ethereum protocol so as to offload the committee's role onto the existing validators because the computational and communication costs borne by the committee scale with the demand for the private world computer. Even with a solution for confidential smart contracts based on fully homomorphic encryption (FHE), which allows the committee to delegate computation on ciphertexts to permissionless parties, called evaluators, a trusted ad-hoc committee is still necessarily to manage secret key shares and perform FHE decryption of output ciphertexts via MPC [SA21, Dai22, Zam25]. This causes the computational and communication costs of each committee party to grow proportionally to the number of decryption requests. Therefore, improving the scalability of the private world computer—in particular, the throughput of ciphertexts that the committee can decrypt per unit time—requires all committee parties to upgrade their hardware capabilities[1].

However, raising the hardware requirements for participating in the committee undermines its decentralization and could increase the likelihood that a threshold number of malicious committee parties collude. Especially, requiring a sufficiently high bandwidth for the hardware not only increases its cost, effectively limiting committee participation to well-funded individuals or organizations, but also risks undermining geographic decentralization because only those located in regions with enough bandwidth can operate such hardware [Rev25]. Thus, for currently implementable cryptographic primitives, including FHE, there is a trade-off between the decentralization of the committee and the throughput at which outputs of confidential smart contracts can be obtained. This implies that achieving scalability for private smart contracts remains a significant challenge.

**Our contributions**: In this position paper, we make the following two contributions toward realizing a scalable private world computer:

- In Section 3, we show that, using indistinguishability/ideal obfuscation (iO), one can construct a private world computer that, after a one-time initial trusted setup for obfuscation, does not introduce any additional trust assumptions beyond those already placed on the Ethereum validators (Table 1), incurring no additional overhead for validators to process confidential smart contracts compared to public smart contracts. Our construction still requires a committee that is distinct from the Ethereum validators to run an MPC protocol for the one-time trusted setup that is secure as long as at least one out of $n$ parties is honest; however, as we discuss in Subsection 3.4, this is a significantly weaker assumption than those made in prior constructions. Remarkably, by combining iO with FHE and succinct non-interactive arguments of knowledge (SNARK), we can realize arbitrary confidential smart contracts on this private world computer by obfuscating a single application-agnostic circuit, which we refer to as the root iO. We further show how to verify the output of the root iO on-chain—i.e., within smart contract program—at a cost comparable to signature verification, as well as how to distribute the obfuscation pro-

[1]It is not sufficient for only a small subset of committee parties to upgrade their hardware capabilities. Since reconstructing the plaintext requires outputs from at least a threshold number of committee parties, the presence of even a single slow party among them becomes a bottleneck.

Table 1: Comparison of private world computers realized from different cryptographic primitives, where $n$ denotes the number of committee parties, and $t < n$ is a threshold.

| **Primitive** | **Trust Assumptions** |
|---|---|
| FHE + MPC for decryption | Ethereum validators, Ongoing $t$-out-of-$n$ committee |
| FHE + MPC for obfuscation + Root iO | Ethereum validators, $n$-out-of-$n$ committee for one-time setup |

cedure among multiple obfuscators in such a way that the security of the root iO is guaranteed as long as at least one obfuscator is honest.

- In Section 4, we review the limitations of existing iO constructions and implementations, and outline our roadmap toward a practical implementation of the root iO (Figure 1). In a nutshell, assuming that the assumptions underlying Diamond iO—our lattice-based iO construction [SBP25]—remain secure, the remaining missing pieces are practical implementations of verifiable FHE and of homomorphic evaluation of a pseudorandom function (PRF) and SNARK verification over key-homomorphic encodings. Once these are available, we can combine them to construct the root iO without incurring prohibitive overhead.

## 2 Preliminaries

### 2.1 Signature-based Blockchain

In Ethereum, each smart contract—deployed by each application developer—has its own program and storage [Woo25]. The storage consists of multiple slots, and the values stored in each slot persist across blocks indefinitely. A user's transaction finalized on the Ethereum blockchain can modify values in slots only in accordance with the contract's program.

Rather than defining the fully fledged specifications of the Ethereum consensus layer, we define a more abstract and general notion of a "signature-based blockchain [Sta25]". That is, for each block number $T \in \mathbb{N}$, we simply model each $T$-th block $\mathsf{block}_T$ as the following tuple:

$$\mathsf{block}_T := (\mathsf{storage}_T, \mathsf{pks}_T, \mathsf{txs}_T, \mathsf{prevhash}_T)\,,$$

where

- $\mathsf{storage}_T$ is a list of values in the storages of all smart contracts at the block number $T$.
- $\mathsf{pks}_T$ is a list of the assigned validators' public keys at the block number $T$.

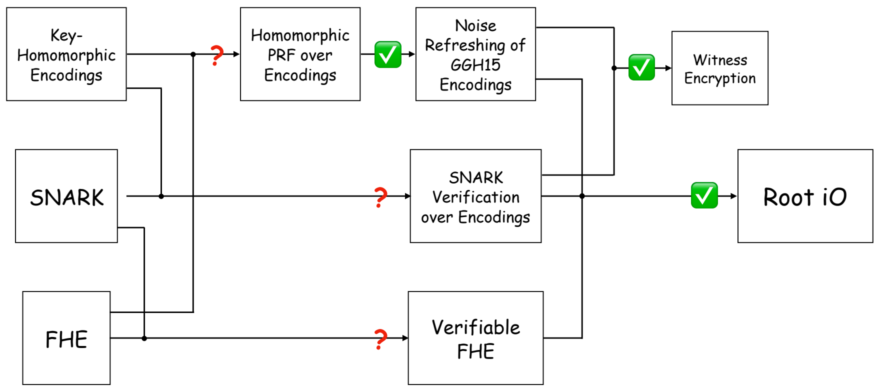
Key-Homomorphic Encodings
SNARK
FHE
Homomorphic PRF over Encodings
Noise Refreshing of GGH15 Encodings
SNARK Verification over Encodings
Verifiable FHE
Witness Encryption
Root iO

Figure 1: Our roadmap toward practical implementation of root iO. A component at the head of each arrow depends on the component(s) at its base. An arrow overlaid with a red question mark indicates a transformation for which we have not yet found a practical implementation. An arrow overlaid with a check mark indicates a transformation that we know how to efficiently implement, provided that all components at the tail of the arrow can be implemented.

- $\mathsf{txs}_T$ is a list of user transactions in the $T$-th block.
- $\mathsf{prevhash}_T$ is a hash of the previous block $\mathsf{block}_T$. The initial hash $\mathsf{prevhash}_1$ is a fixed string.

We assume a function $\mathsf{Transit}$ that derives a next storage and the validators' public keys as follows:

$$(\mathsf{storage}_T,\ \mathsf{pks}_T) := \mathsf{Transit}(T,\ \mathsf{storage}_{T-1},\ \mathsf{pks}_{T-1},\ \mathsf{txs}_T,\ \mathsf{prevhash}_T)$$

We define an NP relation $\mathcal{R}_{\mathsf{SW}}$: an instance is a tuple of the initial block $\mathsf{block}_1^\star$, a smart contract address (an ID to identify a smart contract) $\mathsf{contAddr}$, a list of slot indices $\mathsf{slots}$, and a list of integers $\mathsf{v} \in \mathbb{Z}^\star$, a witness is a block number $T \leq T_{\max}$ and a list of $T$ tuples, each of which consists of the $i$-th block $\mathsf{block}_i$ and a list of digital signatures $\mathsf{signs}_i$, i.e., $(\mathsf{block}_i, \mathsf{signs}_i)_{i\in[T]}$. The instance-witness pair is said to be in $\mathcal{R}_{\mathsf{SW}}$ if and only if all of the following conditions hold:

- $\mathsf{block}_1$ is identical to $\mathsf{block}_1^\star$.
- For every $i \in \{2, \ldots, T\}$:
  - The outputs of $\mathsf{Transit}(i, \mathsf{storage}_{i-1}, \mathsf{pks}_{i-1}, \mathsf{txs}_i, \mathsf{prevhash}_i)$ are identical to $\mathsf{storage}_i$ and $\mathsf{pks}_i$, respectively.

  - The length of $\mathsf{signs}_i$ is equal to the protocol-defined threshold times the length of $\mathsf{pks}_i$.
  - For each signature in $\mathsf{signs}_i$, there is a distinct corresponding public key in $\mathsf{pks}_i$ such that the signature is valid for the public key and the hash of $\mathsf{block}_i$.
  - The hash of $\mathsf{block}_{i-1}$ is equal to $\mathsf{prevhash}_i$.

- $\mathsf{txs}_T$ contains a transaction that calls the smart contract with address $\mathsf{contAddr}$ and, during that call, writes the integers $\mathsf{v}$ to the storage slots $\mathsf{slots}$ of that contract in $\mathsf{storage}_T$[2].

## 2.2 FHE-based Private World Computer

We review a design of a private world computer inspired by the fhevm protocol introduced by Zama [Zam25]. It employs FHE to enable arbitrary computation on encrypted data originating from different parties.

**Syntax and properties of FHE**: We define the syntax of algorithms provided by a public-key FHE scheme as follows:

- $\mathsf{FHE.Setup}(1^\lambda) \to (\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}})$: Given a security parameter $\lambda$, outputs a public key $\mathsf{pk}_{\mathsf{FHE}}$ and a secret key $\mathsf{sk}_{\mathsf{FHE}}$.

- $\mathsf{FHE.Enc}(\mathsf{pk}_{\mathsf{FHE}}, x) \to c$: Given a public key $\mathsf{pk}_{\mathsf{FHE}}$ and a message $x$, outputs a ciphertext $c$.

- $\mathsf{FHE.Eval}(\mathsf{pk}_{\mathsf{FHE}}, C, (c_1, \ldots, c_L)) \to (c'_1, \ldots, c'_M)$: Given a public key $\mathsf{pk}_{\mathsf{FHE}}$, a circuit $C$ with $L$ inputs and $M$ outputs, and $L$ input ciphertexts $(c_1, \ldots, c_L)$, outputs $M$ output ciphertexts $(c'_1, \ldots, c'_M)$.

- $\mathsf{FHE.Dec}(\mathsf{sk}_{\mathsf{FHE}}, (c'_1, \ldots, c'_M)) \to (y_1, \ldots, y_M)$: Given a secret key $\mathsf{sk}_{\mathsf{FHE}}$ and $M$ ciphertexts $(c'_1, \ldots, c'_M)$, outputs messages $(y_1, \ldots, y_M)$.

The correctness property guarantees that decrypting the ciphertext output by the $\mathsf{FHE.Eval}$ algorithm yields the circuit output $C(x_1, \ldots, x_L)$, i.e., it holds that

$$\mathsf{FHE.Dec}(\mathsf{sk}_{\mathsf{FHE}}, (c'_1, \ldots, c'_M)) = C(x_1, \ldots, x_L).$$

The security property guarantees that, informally, a ciphertext leaks no non-trivial information about the message it encrypts.

**FHE-based private world computer**: The FHE-based private world computer relies on three core components, illustrated in Figure 2:

1. **FHE co-processor**: This component comprises permissionless evaluators to execute the $\mathsf{FHE.Eval}$ algorithm on the specified circuit and the input ciphertexts.

[2]For simplicity, we assume that the same storage slots are not written more than once by transactions within a single block.

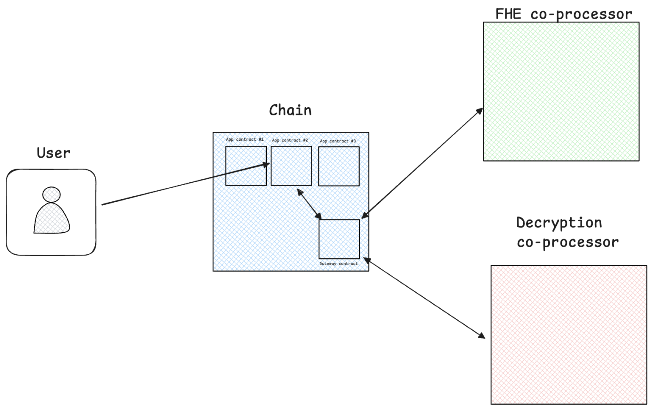

Figure 2: An architecture of an FHE-based private world computer.

2. **Decryption co-processor**: This component handles decryption of ciphertexts.

3. **Smart contracts**: A set of smart contracts on Ethereum[3], which include:

    - Various application-specific smart contracts, each specifying the evaluation functions, input ciphertexts to be evaluated and output ciphertexts to be decrypted
    - A single gateway smart contract that orchestrates the interactions among the FHE co-processor, the decryption co-processor, and the application contracts.

In what follows, we focus on the decryption co-processor, because our design for a private world computer improves precisely this component using iO, while keeping the other components identical to those in the fhevm protocol. In Appendix A, we illustrate in more detail the interface of the gateway contract and how an application running on the FHE-based private world computer interacts with it.

The decryption co-processor in the fhevm protocol is instantiated via a $t$-out-of-$n$ MPC that performs threshold FHE decryption [Zam25]. Delegating control of the decryption key to a single trusted party is extremely risky, as this party can decrypt any ciphertexts in violation of the decryption policies defined by the application contracts. On the other hand, running the decryption MPC among all users is impractical, since users cannot be assumed to be online full-time and able to coordinate for each decryption request. Therefore, their protocol relies

[3]The original fhevm protocol [Zam25] considers a multi-chain design where these contracts do not necessarily reside on the same chain

on an MPC committee shared across all applications, which holds shares of the FHE secret key and performs the decryption MPC. Since such functionality is not provided by today's Ethereum validators, this must be a new, protocol-specific committee.

We require the decryption co-processor to satisfy the following two properties:

- **Security against malicious evaluators**: Even if the evaluator is malicious—namely, the malicious evaluator attempts to extract as much private information as possible from the ciphertexts and does not necessarily evaluate the requested circuit correctly. While the confidentiality of ciphertexts is protected by FHE, FHE by itself cannot force the evaluator to correctly evaluate a specific circuit. This is addressed by verifiable FHE [KVMGH24], which enable the evaluator to make a succinct proof that the output ciphertexts are obtained by executing the $\mathsf{FHE.Eval}$ algorithm on the input ciphertexts and circuit specified by the application contracts. The committee for the decryption co-processor should decrypt the provided output ciphertexts only when the evaluator provides a valid proof[4].

- **Application-agnostic implementation**: Although the decryption policies differ from one application to another, the implementation of the decryption co-processor—specifically, the operations executed by the committee parties—must be the same regardless of the application. Such application-specific differences can instead be absorbed by the gateway contract. As describe in [Zam25], the gateway contract maintains an "access control list (ACL) [Zam25]" that specifies, for each ciphertext, which application contracts are allowed to access it. An application contract evaluates its application-specific decryption policy in its own program, and if the policy is satisfied for some ciphertexts, it calls a decryption-request function of the gateway contract, passing those ciphertexts as an argument, where we assume that the aforementioned proof for the correct homomorphic evaluation is verified within the smart contract program before the decryption policy is evaluated[5]. If and only if the calling contract is authorized to access the supplied ciphertexts according to the ACL, the gateway contract writes the ciphertexts, as a sequence of integers, into certain designated slots in its storage[6]. Consequently, for all applications, it suffices for the committee to decrypt only the ciphertexts written into the designated slots of the gateway contract's storage.

[4]In the fhevm protocol [Zam25], output ciphertexts are optimistically published on-chain and, since the FHE evaluation is deterministic and can be recomputed by anyone, any misbehavior by the FHE co-processor can be detected and punished via a slashing mechanism.

[5]In the fhevm protocol, handle IDs of ciphertexts are passed rather than ciphertexts themselves [Zam25].

[6]In the fhevm protocol, the gateway contract emit events rather than updating the storage.

## 2.3 Indistinguishability/Ideal Obfuscation

We use the abbreviation iO for both indistinguishability obfuscation [BGI+01] and ideal obfuscation [JLLW23]. Both primitives provide the following algorithms:

1. $\mathsf{iO.Obf}(1^\lambda, C) \to \widetilde{C}$: Given a security parameter $1^\lambda$ and a circuit $C$ with $L$ inputs and $M$ outputs, outputs an obfuscated circuit $\widetilde{C}$.

2. $\mathsf{iO.Eval}(\widetilde{C}, \mathbf{x}) \to \mathbf{y}$: Given an obfuscated circuit $\widetilde{C}$ and $L$ inputs $\mathbf{x}$, outputs the evaluation results $\mathbf{y} = C(\mathbf{x})$.

Indistinguishability obfuscation and ideal obfuscation exhibit different tradeoffs between the strength of the security they provide and the strength of the security model they require. Specifically, indistinguishability obfuscation only guarantees that obfuscations of two circuits with the same size and functionality (namely, the input-output behavior) are indistinguishable from each other, but the construction for all circuits exists in a standard model based on standard cryptographic assumptions [BGI+01, JLS21]. In contrast, ideal obfuscation guarantees that, informally, an obfuscated circuit leaks no nontrivial information beyond its input-output behavior for most natural (not contrived) circuits; however, its construction is possible only in a heuristic model called pseudorandom oracle model [JLLW23][7].

If we can implement indistinguishability obfuscation, then we can immediately extend it to an implementation of ideal obfuscation[8]. Therefore, in Section 3 we use iO in the form of ideal obfuscation, while in Section 4 we describe constructions of indistinguishability obfuscation.

# 3 Root iO: Obfuscation for All Applications

## 3.1 Definition

Root iO is a single obfuscation of an application-agnostic circuit that is available to all applications on the private world computer. We introduce this to realize the decryption co-processor described in Subsection 2.2 without relying on any ad-hoc committee after a one-time setup. The obfuscation and evaluation algorithms for root iO are defined with the following syntax:

- $\mathsf{RiO.Obf}(1^\lambda) \to \widetilde{C}$: Given a security parameter $1^\lambda$, outputs an obfuscated circuit $\widetilde{C}$.

- $\mathsf{RiO.Eval}(\tilde{C}, \mathbf{c}, T, (\mathsf{block}_i, \mathsf{signs}_i)_{i \in [T]}) \to \mathbf{y}$: Given the obfuscated circuit $\tilde{C}$, FHE ciphertexts $\mathbf{c}$, a block number $T \in \mathbb{N}$, and a list of $T$ tuples, each

[7]Achieving such a strong security for all circuits is known to be impossible in the standard model [BGI+01]

[8]More concretely, we first construct a public-key functional encryption scheme from indistinguishability obfuscation using the method introduced in [GGH+13b], and then combine it with a hash function to obtain ideal obfuscation, as described in [JLLW23].

of which consists of the $i$-th block $\mathsf{block}_i$ and a list of digital signatures $\mathsf{signs}_i$, outputs plaintexts $\mathbf{y}$.

Notably, the above obfuscation algorithm is assumed to be executed by a single trusted third party; in practice, the private world computer cannot assume the existence of such a party. We address this issue by introducing a decentralized and verifiable obfuscation process in Subsection 3.4.

## 3.2 Instantiation

At a high-level, root iO is instantiated by obfuscating a circuit $C_{\mathsf{RiO}}$ that simulates the behavior of the committee used for the decryption MPC. Recall that to decide whether given ciphertexts $\mathbf{c}$ are allowed to be decrypted, by the application-agnostic implementation of the decryption co-processor, it suffices to simply check whether those ciphertext has been written into designated slots in the storage of the gateway contract. Such a condition can be decided by the NP relation $\mathcal{R}_{\mathsf{SW}}$ defined in Subsection 2.1. Therefore, we design the circuit $C_{\mathsf{RiO}}$ so that it takes as input ciphertexts $\mathbf{c}$ and a SNARK proof for the relation $\mathcal{R}_{\mathsf{SW}}$, and outputs the decryption of $\mathbf{c}$ if and only if it successfully verifies that the proof is valid for $\mathbf{c}$[9].

We hardcode the following values in the circuit $C_{\mathsf{RiO}}$.

- A verification key $\mathsf{vk}_{\mathsf{SW}}$ of the SNARK scheme for the NP relation $\mathcal{R}_{\mathsf{SW}}$.
- The first block of Ethereum $\mathsf{block}_1^\star$.
- The address of the gateway contract $\mathsf{contAddr}$.
- The designated slots $\mathsf{slots}$ in the storage of the gateway contract.
- The FHE secret key $\mathsf{sk}_{\mathsf{FHE}}$.

We formally define $C_{\mathsf{RiO}}$ as below:

- $C_{\mathsf{RiO}}(\mathbf{c}, \pi) \to \mathbf{y}$:
  1. Verify the provided proof $\pi$ with the verification key $\mathsf{vk}_{\mathsf{SW}}$ and an instance $(\mathsf{block}_1^\star, \mathsf{contAddr}, \mathsf{slots}, \mathbf{c})$. If the verification does not pass, return $\mathbf{y} := \mathbf{0}$.
  2. Return the decryption results $\mathbf{y} := \mathsf{FHE.Dec}(\mathsf{sk}_{\mathsf{FHE}}, \mathbf{c})$.

Using $C_{\mathsf{RiO}}$ and the fixed initial block $\mathsf{block}_1^\star$, we instantiate the obfuscation and evaluation algorithms of the root iO as follows:

- $\mathsf{RiO.Obf}(1^\lambda) \to \widetilde{C}$:
  1. Execute $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$.

[9]Our instantation of root iO is highly inspired by a technique introduced in [GGH+13b] that transform iO for conditional FHE decryption to one for any polynomial-sized circuits; however, ours makes the condition verified within the obfuscated circuit agnostic to applications.

2. Sample a proving key $\mathsf{pvk}_{\mathsf{SW}}$ and a verification key $\mathsf{vk}_{\mathsf{SW}}$ of the SNARK scheme for the NP relation $\mathcal{R}_{\mathsf{SW}}$.
3. Depoly a gateway contract, obtaining its contract address $\mathsf{contAddr}$ and the designated slots $\mathsf{slots}$.
4. Construct a circuit $C_{\mathsf{RiO}}$, in which $\mathsf{sk}_{\mathsf{FHE}}$, $\mathsf{vk}_{\mathsf{SW}}$, $\mathsf{contAddr}$, $\mathsf{slots}$ produced in the previous steps are hardcoded.
5. Execute $C' \leftarrow \mathsf{iO.Obf}(1^\lambda, C_{\mathsf{RiO}})$.
6. Output $\widetilde{C} := (\mathsf{pk}_{\mathsf{FHE}}, \mathsf{pvk}_{\mathsf{SW}}, \mathsf{contAddr}, \mathsf{slots}, C')$.

- $\mathsf{RiO.Eval}(\tilde{C}, \mathbf{c}, T, (\mathsf{block}_i, \mathsf{signs}_i)_{i\in[T]}) \rightarrow \mathbf{y}$:

  1. Parse $\widetilde{C}$ as $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{pvk}_{\mathsf{SW}}, \mathsf{contAddr}, \mathsf{slots}, C')$.
  2. Generate a SNARK proof $\pi$ using the proving key $\mathsf{pvk}_{\mathsf{SW}}$, a witness $\Big(T, (\mathsf{block}_i, \mathsf{signs}_i)_{i\in[T]}\Big)$, and an instance $(\mathsf{block}^\star_1, \mathsf{contAddr}, \mathsf{slots}, \mathbf{c})$.
  3. Execute $\mathbf{y} \leftarrow \mathsf{iO.Eval}(C', \mathbf{c}, \pi)$.
  4. Output $\mathbf{y}$.

The obfuscation circuit $\widetilde{C}$ contains the FHE public key $\mathsf{pk}_{\mathsf{FHE}}$, even though this key is not used within the evaluation algorithm itself, since $\mathsf{pkFHE}$ is required to execute the $\mathsf{FHE.Eval}$ algorithm in the FHE co-processor.

The above instantiation satisfies the following correctness property: if the ciphertexts $\mathbf{c}$ have been written to the slots specified by $\mathsf{slots}$ at block number $T$ on Ethereum, then the output of the $\mathsf{RiO.Eval}$ algorithm equals the decryption of $\mathbf{c}$, i.e.,

$$\mathsf{RiO.Eval}(\tilde{C}, \mathbf{c}, T, (\mathsf{block}_i, \mathsf{signs}_i)_{i\in[T]}) = \mathsf{FHE.Dec}(\mathsf{sk}_{\mathsf{FHE}}, \mathbf{c}).$$

**Trust assumption during evaluation**: Once the obfuscated circuit $\widetilde{C}$ is generated honestly, all evaluation can be delegated to permissionless evaluators. Users do not need to place any trust in them because evaluators can neither learn the private data being processed nor deliberately provide incorrect results. Specifically, ideal obfuscation guarantees that $\widetilde{C}$ leaks no non-trivial information beyond its input-output behavior (Subsection 2.3), implying that the embedded FHE secret key $\mathsf{sk}_{\mathsf{FHE}}$ remains unknown to the evaluators. Besides, as described in Subsection 2.2, the proof verification inside $\widetilde{C}$ ensures that confidentiality of any ciphertexts that are not authorized for decryption by the decryption policies is preserved. We discuss the verification of the obfuscated circuit outputs provided by the evaluators in Subsection 3.3. Therefore, unlike existing constructions of a private world computer that rely on FHE and MPC for decryption [SA21, Dai22, Zam25], our root iO-based construction, after a one-time setup for obfuscation, does not require any additional trust beyond that already placed on the Ethereum validators.

## 3.3 On-chain verification of obfuscated circuit outputs

Anyone can verify that given decryption results $\mathbf{y}$ are indeed the outputs of the obfuscated circuit $\widetilde{C}$ by re-executing $\widetilde{C}$ on the same input. However, such a re-execution within a program of a smart contract, i.e., on-chain, is expected to incur prohibitive costs[10].

One way to perform on-chain verification at only a small cost is to have the obfuscator sample a signing key of a digital signature scheme and embed this key into the obfuscated circuit, whose functionality is modified to additionally output a digital signature on the decryption result $\mathbf{y}$. Suppose that the obfuscator publishes the corresponding verification key, smart contracts can then verify that a given $\mathbf{y}$ was output by $\widetilde{C}$ simply by checking this signature under the published verification key.

Furthermore, the signature need not necessarily be generated directly inside the circuit $C_{\mathsf{RiO}}$ itself. Concretely, the obfuscator publishes an FHE encryption of the signing key. Using this ciphertext together with the ciphertext $\mathbf{c}$, the evaluator homomorphically evaluates under FHE the signing algorithm on the output $\mathbf{y}$ encrypted in $\mathbf{c}$, obtaining a ciphertext $\mathbf{c}_{\mathsf{sign}}$ that encrypts the signature. The evaluator then proves, in addition to satisfying the NP relation $\mathcal{R}_{\mathsf{SW}}$, that $\mathbf{c}_{\mathsf{sign}}$ is precisely the ciphertext obtained by the above homomorphic evaluation. The circuit $C_{\mathsf{RiO}}$ outputs the decryptions of both $\mathbf{c}$ and $\mathbf{c}_{\mathsf{sign}}$ if and only if both proofs are valid. In this manner, we can keep the computation inside the obfuscated circuit limited to SNARK verification and FHE decryption, while reducing the on-chain verification cost to be as small as that of a signature verification.

## 3.4 Decentralized and Verifiable Obfuscation

While the definition of iO assumes a single trusted obfuscator, in practice, a private world computer cannot rely on the existence of such a party. Moreover, for the root iO obfuscation, we must keep certain values—most notably the FHE secret key embedded in the obfuscated circuit $C'$ and the private random coin used in its obfuscation—private, while still allowing public verification that $C'$ is indeed an obfuscation of the intended circuit $C_{\mathsf{RiO}}$ and that the public values embedded in $C'$ (such as $\mathsf{vk}_{SW}$ and $\mathsf{contAddr}$) match those published by the obfuscator. One naive approach to addressing these two issues is to run the obfuscation algorithm via a publicly verifiable MPC [BDO14] among committee parties with a sufficiently high threshold; however, since this stacks the computational cost of MPC on top of that of obfuscation, its practical feasibility is doubtful. Instead, we propose a more efficient method that leverages the concrete structure of the obfuscation algorithm of root iO.

**Decentralization**: Recall that the $\mathsf{RiO.Obf}$ algorithm in Subsection 3.2 uses

[10]Indeed, if this re-execution is carried out on a zk-rollup, the validators of the Ethereum blockchain would only need to verify a single SNARK proof, generated by a powerful server, attesting to the correctness of the re-execution. However, given the computational cost of iO, it is unclear whether generating such a proof is practically feasible.

a private random coin for key samplings of the SNARK and FHE schemes, as well as obfuscation[11]. For the SNARK scheme, one can either use a transparent SNARK that does not require a private coin for the setup [BBHR18], or employ an MPC protocol specialized for the SNARK setup process [BGM17, KMSV21, NRBB24] that requires only one honest participant.

To efficiently decentralize the generation of the remaining components, namely the FHE key pair and the obfuscation $C'$, we employ a non-interactive multi-party or multi-key FHE scheme for the decryption process [AJL$^+$12, MW16, LWSH23]. In these schemes, each party holding a secret key share only needs to release a partial decryption for ciphertexts being decrypted, and a threshold number of partial decryptions is sufficient to recover the plaintext. Let $n$ be the committee size. For every $i \in \{1, \ldots, n\}$, we define a circuit $C_{\mathsf{RiO},i}$ obtained from $C_{\mathsf{RiO}}$ by hardcoding the $i$-th party's secret key share $\mathsf{sks}_i$ in place of the FHE secret key $\mathsf{sk}_{\mathsf{FHE}}$, and by making it output the $i$-th party's partial decryptions for the provided ciphertexts $\mathbf{c}$ if the verification of the provided proof $\pi$ succeeds. We set the threshold for decryption to $n$, i.e., the FHE secret key remains confidential if at least one party is honest.

Using the modified circuit $C_{\mathsf{RiO},i}$, each $i$-th party only needs to obfuscate $C_{\mathsf{RiO},i}$ because the evaluator can non-interactively recover the decryption result by combining the partial decryptions obtained from each obfuscation of $C_{\mathsf{RiO},i}$. This implies that the committee does not need to execute the iO algorithm within the MPC. Formally, the decentralized obfuscation algorithm for root iO is defined as follows:

- $\mathsf{RiO.DObf}(1^\lambda, 1^n) \to \widetilde{C}$:

    1. The $n$ parties execute a MPC protocol to setups a FHE public key $\mathsf{pk}_{\mathsf{FHE}}$ and $n$ secret key shares $\{\mathsf{sks}_i\}_{i \in \{1,\ldots,n\}}$, where the $i$-th party holds $\mathsf{sks}_i$.
    2. The $n$ parties execute a MPC protocol to produce a proving key $\mathsf{pvk}_{\mathsf{SW}}$ and a verification key $\mathsf{vk}_{\mathsf{SW}}$ of the SNARK scheme for the NP relation $\mathcal{R}_{\mathsf{SW}}$.
    3. The first party depolys a gateway contract and publishes its contract address $\mathsf{contAddr}$ and the designated slots $\mathsf{slots}$ to the other parties.
    4. For every $i \in \{1, \ldots, n\}$, the $i$-th party constructs a circuit $C_{\mathsf{RiO},i}$, in which $\mathsf{sks}_i$, $\mathsf{vk}_{\mathsf{SW}}$, $\mathsf{contAddr}$, $\mathsf{slots}$ produced in the previous steps are hardcoded.
    5. For every $i \in \{1, \ldots, n\}$, the $i$-th party executes $C'_i \leftarrow \mathsf{iO.Obf}(1^\lambda, C_{\mathsf{RiO},i})$.
    6. Output $\widetilde{C} := \big(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{pvk}_{\mathsf{SW}}, \mathsf{contAddr}, \mathsf{slots}, \{C'_i\}_{i \in \{1,\ldots,n\}}\big)$.

**Trust assumption during obfuscation**: The above decentralized obfuscation algorithm only requires that at least one party is honest to ensure the confidentiality of the FHE secret key embedded in the obfuscated circuits

[11]We ignore the random coin used for depolying a gateway contract—in particular, the depolyer's signing key to sign a deployment transaction—because it does

$\{C'_i\}_{i\in\{1,\ldots,n\}}$. Since the Ethereum validators cannot themselves act as obfuscators, one might suspect that our root iO-based construction ultimately requires the same trust assumptions as existing designs in [SA21, Dai22, Zam25]. However, in our case the committee is needed only during a one-time setup: once the obfuscated circuits are produced, the system no longer depends on the committee. This allows us to set the threshold to its maximum value $n$ and simply wait for all parties to output their obfuscated circuits, re-running the setup with a fresh committee if the previous one fails—without any impact on users, as this concerns only the setup phase.

**Verifiability**: We next discuss how to make the provided obfuscated circuit verifiable, namely verifying that $\widetilde{C}$ obfuscates the expected circuits $\{C_{\mathsf{RiO},i}\}_{i\in\{1,\ldots,n\}}$ without revealing the hardcoded secrets and the used private coins. Since existing MPC protocols for the SNARK setup already provide an efficient method to verify the soundness of the MPC outputs [NRBB24], it therefore suffices to make the FHE public key $\mathsf{pk}_{\mathsf{FHE}}$ and the obfuscated circuits $\{C'_i\}_{i\in\{1,\ldots,n\}}$ verifiable. A naive approach is to prove the entire FHE setup and obfuscation procedures in a SNARK, which is unclear whether this can be realized with a practical computational cost.

A more promising approach is to adapt the cut-and-choose technique used for garbled circuits [LP07] to proving the honest obfuscation. Specifically, the parties repeat $v$ times the procedure that, using a fresh random coin, generates a pair $(\mathsf{pk}_{\mathsf{FHE}}, Ci'_{i\in1,\ldots,n})$. The verifier then uniformly at random selects a subset $\mathcal{T} \subseteq \{1,\ldots,v\}$ of $t$ distinct indices and, for each index in $\mathcal{T}$, requests the committee to open the random coins used in the corresponding obfuscation procedure. After these coins are revealed, the verifier honestly re-runs the obfuscation algorithm with the revealed randomness and checks that all outputs exactly match those originally provided by the committee.

The remaining $v - t$ obfuscations whose random coins are not opened are then used by the evaluator to actually obtain the output of the root iO. If their outputs are not all identical, the evaluator adopts the value given by the majority of these obfuscations. We believe that by choosing $v$ to be sufficiently large—yet still polynomial in the security parameter $\lambda$—the soundness error can be bounded by a negligible function in $\lambda$, although a more careful analysis is required. As in [CZ21], the above protocol can be made non-interactive in the random oracle model via the Fiat-Shamir transform.

## 4 Roadmap for Making Root iO Practical

### 4.1 Limitations of Existing iO Constructions and Implementations

Despite recent theoretical advances in iO, it remains far from being practical. In fact, to the best of our knowledge, all existing implementations of iO are incomplete for one of the following reasons.

- No vulnerabilities have been found in the constructions and their implementations, but the program functionalities supported by the constructions are significantly restricted, e.g., point functions [DCBC$^+$16] and conjection programs [CDG$^+$18].

- The theoretical construction supports arbitrary programs, but in practice existing implementations omit some of the computations required for security due to efficiency limitations, or vulnerabilities have now been found in the theoretical constructions themselves [AHKM14, LMA$^+$16, HHSS17, Zon, Mi25].

In the following, we classify (a subset of) existing iO constructions for arbitrary programs by their underlying assumptions and, for each category, describe their limitations from security and efficiency perspectives.

**iO from multilinear maps**: The first iO construction introduced in [GGH$^+$13b] relies on multilinear maps, a generalization of bilinear maps, where bilinear maps can be instantiated by pairings on elliptic curves. Several works [GGH13a, CLT13, GGH15] have proposed candidates of multilinear maps. Because of their relatively simple and algebraic structures, several implementations of these candidates have been explored in practice.

Apon et al. [AHKM14] and Lewi et al. [LMA$^+$16] have implemented CLT13 multilinear maps [CLT13]. However, subsequent works have demonstrated polynomial-time attacks against CLT13 [CHL$^+$14, CGH$^+$15, CFL$^+$16].

Halevi et al. [HHSS17] employ GGH15 multilinear maps [GGH15] to implement iO for read-once branching programs, a class that can represent arbitrary oblivious NFAs but is still far too limited to realize our root iO construction. Unfortunately, several works [CLLT16, CGH17] have demonstrated attacks against GGH15, and the attack introduced in [CVW18] is applicable to the iO implementation in [HHSS17]. Following these works, new attacks and countermeasures have been proposed [CCH$^+$19, CHVW19], whereas there is no known GGH15-based iO construction that is secure under standard assumptions.

**iO from standard assumptions**: A new line of work, initiated in [AJL$^+$19], constructs iO without relying on multilinear maps and ultimately leads to the first construction based solely on standard assumptions [JLS21]. Its subsequent works [JLS22, RVV24] prove that the following three standard assumptions suffice to construct iO:

- The Decision Linear assumption on bilinear groups.
- The Learning Parity with Noise (LPN) assumption over large prime fields.
- Either the existence of a Boolean pseudo-random generator (PRG) represented by a constant-depth circuit [JLS22]—instantiated by the Goldreich's PRG [Gol00]—or the sparse LPN assumption over binary fields [RVV24].

An important point to note is that these constructions are not post-quantum secure because the first assumption relies on bilinear maps. Constructing iO from standard and post-quantum secure assumptions remains an open problem.

Despite their well-established security, to the best of our knowledge, there is currently no complete implementation of any iO construction from standard assumptions[12]. A major obstacle to implementation is the complexity and inefficiency that arise from composing cryptographic primitives in a black-box manner across multiple layers.

One of the main bottlenecks lies in the transformation from functional encryption (FE) to iO [AJ15, BV15]. FE is a generalization of public-key encryption: a holder of a secret key for a public function $f$—called functional secret key—can decrypt a ciphertext encrypting an input $x$ to obtain the output $f(x)$, but learns nothing else about $x$ [BSW11]. Most recent iO constructions—both those based on standard assumptions and those based on the new lattice assumptions discussed later—construct FE schemes that are sufficiently efficient to be used in the FE-to-iO transformation methods, rather than directly constructing iO. However, the transformation is costly because it recursively invokes the FE encryption algorithm on every input bit: during decryption, the encryption algorithm is evaluated as the function $f$ to produce an FE ciphertext involving the next input bit [AJ15, BV15]. This implies that even if we only need to obfuscate simple programs, the lower bound on the complexity of $f$ that must be supported by the underlying FE scheme is essentially dictated by the complexity of the FE encryption algorithm.

In addition to the inefficiency in the transformation, the FE constructions in [JLS21, JLS22, RVV24] are already far from being concretely efficient due to the multi-layer composition of cryptographic primitives. For example, the construction in [JLS22] represents a function by multivariate polynomials that outputs Yao's garbled circuits [Yao86] by computing PRG on provided private seeds. In these polynomials, if the degree in the variables corresponding to the private inputs is at most $d$, then the FE ciphertext must contain $\mathcal{O}(2^{d/2})$ elements of a bilinear group (specifically, points on an elliptic curve). Although $d$ is independent of the size or depth of the function $f$ being evaluated, this does not imply that $d$ is small enough to be practical. In fact, if the PRG is instantiated by the Goldreich's PRG [Gol00], $d$ is at most 56, incurring the impractical number of the elements.

**iO from new lattice assumptions**: Another line of research has proposed lattice-based iO constructions [BDGM23, BDGM22, DQV$^+$21, GP21, WW21, AKY24b, BDJ$^+$25, SBP25, AMYY25, HJL25]. Most of these constructions rely on newly introduced, simple-to-state lattice assumptions, whose common structure is summarized by Hsieh et al. [HJL25] under the umbrella term "LWE-with-hints. [HJL25]" assumptions. Informally speaking, they claim that some LWE samples remain pseudorandom even if an adversary can access to hints that are specifically designed to leak additional information about the LWE samples. These lattice-based iO constructions employ such hints to allow an evaluator of an obfuscated circuit to decrypt output ciphertexts obtained via homomorphic evaluation on input ciphertexts, without revealing any information beyond the

[12]There are ongoing implementation efforts [Sue24], but to the best of our knowledge no implementation to date instantiates all the building blocks of the iO constructions.

outputs.

Unfortunately, their subsequent works [HJL21, JLLS23, DJM$^+$25, AMYY25, HJL25] has exhibited counterexamples to the assumptions, demonstrating that these hints can in fact leak more information than originally intended. However, this does not immediately imply that the iO constructions based on these attacked assumptions are broken, because some of the underlying assumptions, such as private-coin evasive LWE [Tsa22, VWW22], are non-falsifiable; in particular, the existence of a few counterexamples does not imply that the assumption is invalid in all cases.

Recall that except for our work in [SBP25], those existing lattice-based iO constructions in fact first build a FE scheme under their new lattice assumptions and then obtain iO by applying the FE-to-iO transformation methods [AJ15, BV15]. From the efficiency perspective, their FE constructions are conceptually simpler than those based on standard assumptions; in particular, they can be realized using cryptographic primitives such as FHE with fewer layers of composition. Especially, Agrawal et al. [AKY24a] construct FE directly from lattices, using only PRF as a black-box primitive.

However, their FE scheme still has a bottleneck that makes implementation challenging: simulating the homomorphic evaluation algorithm of the FHE scheme on the key-homomorphic encodings introduced by Boneh et al. (BGG+ encodings) [BGG$^+$14]. We describe this bottleneck in more detail in the next section.

**Diamond iO**: One of common bottlenecks in most lattice-based iO constructions is dependence on the FE-to-iO transformation. To address it, we proposed a lattice-based iO construction called Diamond iO [SBP25], removing the dependence on the conventional transformation methods in [AJ15, BV15]. This replaces the recursive FE encryption with simple matrix multiplications by using encodings introduced in GGH15 multilinear maps (GGH15 encodings) [GGH15][13]. The construction additionally uses the AKY24 FE scheme [AKY24a] in a non-black-box manner, and is proven secure under the LWE and private-coin evasive LWE assumptions, as well as our new lattice assumption called all-product LWE.

We implemented Diamond iO and evaluated its performance [Mi25]. However, we later discovered a vulnerability in the optimization introduced only in the implementation[14]. Specifically, the implementation replaces the homomorphic PRF evaluation over BGG+ encodings introduced in [AKY24a] with a more direct construction whose security is based on a variant of the all-product LWE assumption, in which samples are rounded to a smaller modulus, in a manner similar to the Learning with Rounding (LWR) assumption. We subsequently realized that, because the rounded values are not reduced modulo the smaller modulus, the security proof of the construction using this PRF is flawed, and that the attack on evasive LWE proposed in [AMYY25, HJL25] is in fact applicable to this variant. To address this vulnerability, we must, as in

[13]Our use of GGH15 encodings avoids the attacks against GGH15 multilinear maps for the reasons described in [VWW22]

[14]The theoretical construction remains secure as long as the underlying assumptions hold.

the theoretical construction, evaluate the homomorphic PRF over the BGG+ encodings, which is impractical at this time.

In addition to this vulnerability, Diamond iO is also severely constrained in the input size of circuits it can obfuscate: for practical parameter choices, it supports at most a dozen input bits. This is because, as the evaluator's input size grows, the noise added to the GGH15 encodings of the inputs grows exponentially, which in turn requires increasing the encoding modulus $q$ accordingly. Equivalently, the bit-length of $q$, i.e., $\log_2 q$, grows proportionally to the input size $L$. The matrices used to update the GGH15 encodings—namely, lattice preimages—then grow proportionally, both in number and in their row and column dimensions, with $\log_2 q$. Consequently, the asymptotic complexity of the obfuscation and evaluation algorithms, as well as the size of the obfuscated circuit, grows on the order of $\Omega(L^3)$.

The benchmark results in [Mi25] demonstrate that, even for four input bits, obfuscation and evaluation take 373 and 85 minutes, respectively, and the obfuscated circuit size is 88 GB. These results lead us to conclude that, with the current Diamond iO construction, it is difficult to scale the input size to a practically meaningful range.

## 4.2 Our Roadmap toward Practical Implementation of Root iO

Figure 1 illustrates our roadmap toward practical implementation of root iO. In this roadmap, we primarily focus on improving efficiency, and further cryptanalysis of the underlying lattice assumptions—particularly private-coin evasive LWE—must proceed in parallel. Moreover, this roadmap reflects what we currently believe, based on our results so far, to be the most direct path toward practical iO; it does not rule out alternative approaches, such as radically different iO constructions based on local mixing [CCMR24].

Apart from verifiable FHE [KVMGH24], the only components for which we have not yet found a practical implementation are homomorphic PRF evaluation and verification of SNARK proofs over key-homomorphic encodings. In other words, we believe that if these two operations can be computed efficiently over the encodings, root iO can be made practical. In the following, we first recall the properties of BGG+ encodings [BGG+14]—a representative instantiation of key-homomorphic encodings—and then describe each component in the roadmap in detail.

**Review of BGG+ Encodings**: Let $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z}$ denotes the integers modulo $q$ for the modulus $q \geq 2$. The BGG+ encoding of integers $\mathbf{x} \in \mathbb{Z}_q^L$ under a secret $\mathbf{s} \in \mathbb{Z}_q^n$ and a public key $\mathbf{A} \in \mathbb{Z}_q^{n \times Lm}$, denoted by $\mathsf{Encode}_{\mathbf{s}}(\mathbf{A}, \mathbf{x})$, is defined as follows:

$$\mathsf{Encode}_{\mathbf{s}}(\mathbf{A}, \mathbf{x}) := \underline{\mathbf{s}\,(\mathbf{A} - \mathbf{x} \otimes \mathbf{G})},$$

where $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ is a fixed matrix called gadget matrix, $\otimes$ denotes a tensor product operation, and a small (norm-bounded) error term is added to each underlined term.

An important property of BGG+ encodings is support of key-homomorphic evaluation of arithmetic circuits over encodings. One can publicly compute an encoding of the circuit outputs from that of the inputs, without knowing the secret $\mathbf{s}$. Additionally, the public key in the output encoding can be deterministically derived only from the circuit and the input public key, independent of the encoded inputs $\mathbf{x}$.

Formally, we can define two deterministic algorithms with the following syntax:

- $\mathsf{EvalPK}(C, \mathbf{A}) \to \mathbf{A}_C$: Given a circuit $C$ with $L$ inputs and $M$ outputs and an input public key $\mathbf{A} \in \mathbb{Z}_q^{n \times Lm}$, returns an output public key $\mathbf{A}_C \in \mathbb{Z}_q^{n \times M}$.
- $\mathsf{EvalEnco}(C, \mathbf{A}, \mathbf{x}, \mathsf{Encode}_{\mathbf{s}}(\mathbf{A}, \mathbf{x})) \to \mathsf{Encode}'_{\mathbf{s}}(\mathbf{A}_C, C(\mathbf{x}))$: Given a circuit $C$ with $L$ inputs and $M$ outputs, an input public key $\mathbf{A} \in \mathbb{Z}_q^{n \times Lm}$, inputs $\mathbf{x} \in \mathbb{Z}_q^L$, an input encoding $\mathsf{Encode}_{\mathbf{s}}(\mathbf{A}, \mathbf{x})$, returns an output encoding $\mathsf{Encode}'_{\mathbf{s}}(\mathbf{A}_C, C(\mathbf{x}))$.

The outputs of these algorithms satisfy the following equation[15]:

$$\mathsf{Encode}'_{\mathbf{s}}(\mathbf{A}_C, C(\mathbf{x})) = \underline{\mathbf{s}\mathbf{A}_C + C(\mathbf{x})}. \tag{1}$$

At a high level, the output encoding hides the outputs $C(\mathbf{x}) \in \mathbb{Z}_q^M$ in the second term using the mask $\mathbf{s}\mathbf{A}_C$ in the first term, which is unique to the secret $\mathbf{s}$ and the output public key $\mathbf{A}_C$. Therefore, if a trusted party who knows $\mathbf{s}$ releases the mask $\underline{\mathbf{s}\mathbf{A}_C}$ by multiplying $\mathbf{s}$ and the output of the $\mathsf{EvalPK}$ algorithm, an evaluator running the $\mathsf{EvalEnco}$ algorithm can recover $C(\mathbf{x})$ from the output encoding. The trusted party can publish this mask non-interactively, namely without knowing the inputs $\mathbf{x}$, since the output public key $\mathbf{A}_C$ can be derived only from $C$ and $\mathbf{A}$. Furthermore, the mask does not allow the evaluator to learn the outputs of circuits other than the specified circuit $C$.

One might view this mask as a special decryption key that releases only the output of a specific circuit, and therefore regard key-homomorphic encodings as a superior version of FHE. However, this is not true, because performing homomorphic multiplications between encodings requires the evaluator to know the corresponding inputs. In other words, the inputs must be public except when the inputs are used only in linear computations, which is why the $\mathsf{EvalEnco}$ algorithm takes the input $\mathbf{x}$ as an argument.

A common approach to handling private multiplication over encodings is to encode FHE encryptions of the private inputs and then evaluate a circuit $C$ that simulates FHE evaluation over encodings [GVW15, BTVW17, HLL23,

[15]In standard notations of BGG+ encodings, the second term of the output encoding is $-\mathbf{s}\,(C(\mathbf{x}) \otimes \mathbf{G})$ rather than $C(\mathbf{x})$. To derive Eq. 1, we assume that 1) the last element of $\mathbf{s}$ is fixed to -1 and 2) the output encoding is finally multiplied by a gadget-decomposed matrix $\mathbf{G}^{-1}\begin{pmatrix}\mathbf{0}_{n-1} \\ 1\end{pmatrix}$, as described in [SBP25].

AKY24a, SBP25]. By using either the technique introduced in [GVW15] or the one introduced in [BTVW17], one can decrypt FHE ciphertexts over the encodings without revealing the FHE secret key $\mathbf{k}$. However, the circuit output $C(x)$ contains not only the decryption result in its higher bits but also decryption errors in its lower bits because those decryption techniques do not perform rounding after an inner product between $\mathbf{k}$ and the ciphertexts. To prevent these errors from leaking to the evaluator, the AKY24 FE scheme additionally computes a homomorphic PRF over the encodings and adds its output in the lower bits, thereby masking the errors [AKY24a].

**Homomorphic PRF over encodings**: Homomorphic PRF evaluation over encodings is essential for security as discussed above, and is also crucial for improving efficiency via the noise refreshing described below. It requires a circuit that takes as input an FHE encryption of a PRF key $K_{\mathsf{PRF}}$ and a public seed $s$, outputs FHE encryptions of pseudorandom values $\mathbf{r}$ whose size is upper bounded by $p < q$, i.e., $\mathbf{r} \in \mathbb{Z}_p^M$, where $q$ is both the modulus of the BGG+ encodings and of the FHE ciphertexts. This circuit is provided for the $\mathsf{EvalPK}$ and $\mathsf{EvalEnco}$ algorithms to evaluate it over the public keys and the encodings, respectively. An important point is that the modulus of the encoding of $\mathbf{r}$ must remain $q$; therefore techniques that perform modulus switching on the encodings themselves [BTVW17] do not apply.

One promising approach is introduced in [DJL$^+$24]. This evaluates LWR-based PRF via a single programmable/functional bootstrapping of FHE [CJP21, LMS24]. Another promising approach is to evaluate FHE-friendly block ciphers or hash functions [ARS$^+$15, AGR$^+$16, DEG$^+$18, GKR$^+$21, DGH$^+$23] as PRF without FHE bootstrapping.

Unfortunately, at present, even a single FHE multiplication over the encodings is not yet practical. This is mainly because multiplication over encodings requires the encoded integers to be significantly smaller than the encoding modulus $q$—typically just bits—to avoid a large blow-up of errors after multiplication. Therefore, when using conventional BGG+ encodings, we need to simulate FHE evaluation by Boolean circuits, incurring large circuit size and multiplicative depth.

We addressed this inefficiency by introducing a method to evaluate lookup tables (LUTs) over BGG+ encodings [SB25]. It packs many integers into a single encoding and processes multiple bits in a single LUT evaluation, thereby reducing the circuit size and depth. For example, when simulating modulo-$q$ multiplication over encodings, the LUT evaluation reduces the circuit size and multiplicative depth by factors of about 33 and 4, respectively, compared to the Boolean-simulation baseline. However, even after this reduction, the depth still exceeds 150, which is an obstacle to finding practical parameters that satisfy both correctness and security.

**Noise refreshing of GGH15 encodings**: As described in Subsection 4.1, the current Diamond iO construction cannot support a large input size without increasing the modulus $q$ since the noise added to the GGH15 encodings of the evaluator's input bits grows exponentially with the input size. The goal of noise refreshing is to periodically refresh the noise in GGH15 encodings by homomorphically computing new encodings of the same inputs whose error terms (noise) are freshly sampled via a PRF so that both the maximum noise magnitude and the modulus $q$ grow at most polynomially in the input bit-length $L$ of the circuit being obfuscated[16].

In Diamond iO [SBP25], for every $i \in \{0, 1, \ldots, L\}$, the GGH15 encoding of the first $i$ bits of the evaluator's inputs $\mathbf{x}_i \in \{0,1\}^i$ is defined as follows[17]:

$$\mathbf{p}_{\mathbf{x}_i} := \Big( (1, \mathbf{x}_i) \otimes \mathbf{s}_{\mathbf{x}_i} \Big) \mathbf{B}_i + \mathbf{e}_{\mathbf{x}_i},$$

where $\mathbf{s}_{\mathbf{x}_i} \in \mathbb{Z}_q^n$ and $\mathbf{e}_{\mathbf{x}_i} \in \mathbb{Z}_q^{m_b}$ are, respectively, a secret and an error unique to $\mathbf{x}_i$, and $\mathbf{B}_i \in \mathbb{Z}_q^{(i+1)n \times m_b}$ is a public matrix sampled for each $i$ (but independent of $\mathbf{x}_i$). For every $i \in \{1, \ldots, L\}$ and $b \in \{0,1\}$, the obfuscator provides a special matrix $\mathbf{K}_{i,b}$—technically lattice preimages [MP12]—such that the multiplication between $\mathbf{p}_{\mathbf{x}_i}$ and $\mathbf{K}_{i+1,b}$ produces the encoding of $\mathbf{x}_i || b$, i.e., $\mathbf{p}_{\mathbf{x}_i} \mathbf{K}_{i+1,b} = \mathbf{p}_{\mathbf{x}_i || b}$. According to each bit of the inputs $\mathbf{x}_L \in \{0,1\}^L$, the evaluator multiplies matrices $\mathbf{K}_{1,x_1} \cdots \mathbf{K}_{L,x_L}$ by the initial GGH15 encoding provided by the obfuscator, which is called input-insertion process. After inserting all input bits, the evaluator multiplies the final GGH15 encoding by another special matrix (a lattice preimage) $\mathbf{K}_{\mathsf{att}}$ included in the obfuscated circuit that converts the GGH15 encoding of $\mathbf{x}_L \in \{0,1\}^L$ into a BGG+ encoding of the same inputs. During these processes, because each error $\mathbf{e}_{\mathbf{x}_i}$ grows multiplicatively by each matrix multiplication, the noise in the resulting BGG+ encoding increases exponentially in the input size $L$. This is why the error $\mathbf{e}_{\mathbf{x}_i}$ must be periodically refreshed to a small value to make supported input size scalable.

For example, we consider the case where the GGH15 encodings are refreshed every 8 input bits. For each $i$ that is a multiple of 8, the evaluator evaluates the following circuit $C_{\mathsf{ref}}$ over the BGG+ encodings of $\mathbf{x}_i$, the FHE encryption of the PRF key $K_{\mathsf{PRF}}$, and the FHE secret key $\mathbf{k}$:

1. Evaluate the homomorphic PRF on input an encryption of the PRF key $K_{\mathsf{PRF}}$ with the public seed $(\mathsf{secret}, i, \mathbf{x}_i)$, obtaining an encryption of a fresh secret $\mathbf{s}'_{\mathbf{x}_i} \in \{0, \pm 1\}^n$ that is indistinguishable from a uniformly random vector in $\{0, \pm 1\}^n$ [18].

2. Evaluate another homomorphic PRF on input an encryption of the PRF key $K_{\mathsf{PRF}}$ with the public seed $(\mathsf{error}, i, \mathbf{x}_i)$, obtaining an encryption of a fresh error $\mathbf{e}'_{\mathbf{x}_i} \in \mathbb{Z}^{m_B}$ that is indistinguishable from a random vector sampled from an appropriate centered binomial distribution (CBD). Such a PRF can be constructed as a linear combination of the outputs of a

[16]Hence, the bit-length $\log_2 q$ grows only polylogarithmically in $L$

[17]The actual construction in [SBP25] additionally encodes the FHE secret key and the FHE encryption of the circuit being obfuscated along with the PRF key $K_{\mathsf{PRF}}$, but they are omitted here for simplicity.

[18]We need to fresh not only the error but also the secret because the size of the secret also grows multiplicatively with each matrix multiplication, and this secret size in turn affects the magnitude of the error.

PRF whose output is computationally indistinguishable from the uniform distribution over $\{0,1\}$ [SAB+22].

3. By linear homomorphic computation, compute an encryption of $\mathbf{p}'_{\mathbf{x}_i} := \Big((1, \mathbf{x}_i) \otimes \mathbf{s}'_{\mathbf{x}_i}\Big)\mathbf{B}_i + \mathbf{e}'_{\mathbf{x}_i}$.

4. Assume that the modulus $q$ is a product of $h$ co-prime odd integers, i.e., $q = \prod_{j=1}^{h} q_j$. For every $j \in \{1, \dots, h\}$,

   (a) Evaluate the homomorphic PRF on input an encryption of the PRF key $K_{\mathsf{PRF}}$ with the public seed $(\mathsf{mask}, i, \mathbf{x}_i, j)$, obtaining an encryption of a fresh randomness $\mathbf{r}^{(j)}_{\mathbf{x}_i} \in \mathbb{Z}^{m_B}_{q/q_j}$ that is indistinguishable from a uniformly random vector in $\mathbb{Z}^{m_B}_{q/q_j}$[19].

   (b) By linear homomorphic computation, compute an encryption of $\frac{q}{q_j}\mathbf{p}'_{\mathbf{x}_i} + \mathbf{r}^{(j)}_{\mathbf{x}_i}$.

   (c) Decrypt the above ciphertext using the technique in [GVW15], thereby obtaining $\mathbf{y}_{i,j} := \frac{q}{q_j}\mathbf{p}'_{\mathbf{x}_i} + \mathbf{r}^{(j)}_{\mathbf{x}_i} + \mathbf{e}_{\mathsf{fhe},i,j}$, where $\mathbf{e}_{\mathsf{fhe},i,j}$ denotes FHE decryption errors.

   (d) Output $\mathbf{y}_{i,j}$.

When the input public key is $\mathbf{A}$, the obfuscator computes $\mathbf{A}_{C_{\mathsf{ref}}} \leftarrow \mathsf{EvalPK}(C_{\mathsf{ref}}, \mathbf{A})$ and releases a special matrix (lattice preimage) that allows the evaluator to obtain the mask $\underline{\mathbf{s}_{\mathbf{x}_i}\mathbf{A}_{C_{\mathsf{ref}}}}$. As described above, this mask enables the evaluator to recover $\mathbf{y}$ from the output encoding $\mathsf{Encode}_{\mathbf{s}_{\mathbf{x}_i}}(\mathbf{A}_{C_{\mathsf{ref}}}, \mathbf{y})$. The evaluator finally computes $\lceil \frac{q_j}{q}\mathbf{y} \rfloor = \mathbf{p}'_{\mathbf{x}_i} \mod q_j$ for every $q_j$, recovering $\mathbf{p}'_{\mathbf{x}_i}$ by Chinese remainder theorem (CRT). The noise in the output GGH15 encoding is refreshed because the noise in the original GGH15 encoding is discarded by the final rounding operation. In this manner, if we can practically implement homomorphic evaluation of a PRF over BGG+ encodings—whose output only needs to be indistinguishable from a uniformly random vector—then we can easily realize noise refreshing.

**SNARK verification over encodings and witness encryption**: This component requires evaluating a circuit over BGG+ encodings that on input a SNARK proof simulates the SNARK verification algorithm. If we can additionally use noise refreshing and feed an input of sufficient size to accommodate a SNARK proof into the obfuscated circuit, the evaluator can obtain a BGG+ encoding of their SNARK proof non-interactively. Therefore, by having the evaluator run a circuit over encodings that outputs a hardcoded private message only when the provided proof is valid, we obtain a construction of witness encryption [GGSW13], also known as null-iO [WZ17, GKW17].

[19]More precisely, $\mathbf{r}^{(j)}_{\mathbf{x}_i}$ must be sampled from a range slightly smaller than $[0, q/q_j)$ so that, after adding the FHE decryption error introduced in the next step, the resulting value is still less than $q/q_j$.

**Root iO**: Recall that, as described in Section 3, root iO is the obfuscation of a circuit that takes as input FHE ciphertexts together with a SNARK proof, and outputs their decryption only if the proof certifies that the ciphertexts were obtained by an appropriate homomorphic evaluation and are authorized for decryption under the relevant decryption policies[20]. Verifiable FHE [KVMGH24] enables the generation of a SNARK proof certifying the correctness of such ciphertexts. Noise refreshing allows the evaluator to input the proof and ciphertext to the obfuscated circuit, namely obtaining their BGG+ encodings non-interactively, without the exponential growth of the modulus $q$. SNARK verification over encodings makes it possible to evaluate the required circuit over the encodings of the proof and the ciphertext, and reveal only its output to the evaluator. Therefore, practical implementations of these three components would suffice to make root iO practical.

## References

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219. Springer, Berlin, Heidelberg, December 2016.

[AHKM14] Daniel Apon, Yan Huang, Jonathan Katz, and Alex J. Malozemoff. Implementing cryptographic program obfuscation. Cryptology ePrint Archive, Report 2014/779, 2014.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 308–326. Springer, Berlin, Heidelberg, August 2015.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer, Berlin, Heidelberg, April 2012.

[20]More precisely, the fact that a ciphertext was obtained by the appropriate homomorphic evaluation is verified within the smart contract program, and root iO in turn verifies that, after passing this check, the ciphertext has been written into the storage of the gateway contract. Nevertheless, albeit in an indirect way, verifiable FHE is still required in order to construct a private world computer based on root iO.

[AJL$^+$19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 284–332. Springer, Cham, August 2019.

[AKY24a] Shweta Agrawal, Simran Kumari, and Shota Yamada. Compact pseudorandom functional encryption from evasive LWE. Cryptology ePrint Archive, Report 2024/1719, 2024.

[AKY24b] Shweta Agrawal, Simran Kumari, and Shota Yamada. Pseudorandom multi-input functional encryption and applications. Cryptology ePrint Archive, Report 2024/1720, 2024.

[AMYY25] Shweta Agrawal, Anuja Modi, Anshu Yadav, and Shota Yamada. Evasive LWE: Attacks, variants & obfustopia. Cryptology ePrint Archive, Report 2025/375, 2025.

[ARS$^+$15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, Berlin, Heidelberg, April 2015.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018.

[BDGM22] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for IO: Circular-secure LWE suffices. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP 2022: 49th International Colloquium on Automata, Languages and Programming*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:20. Schloss Dagstuhl, July 2022.

[BDGM23] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. *Journal of Cryptology*, 36(3):27, July 2023.

[BDJ$^+$25] Pedro Branco, Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Pseudorandom obfuscation and applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part V*, volume 16004 of *Lecture Notes in Computer Science*, pages 663–698. Springer, Cham, August 2025.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14: 9th International Conference on Security in Communication Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196. Springer, Cham, September 2014.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556. Springer, Berlin, Heidelberg, May 2014.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Berlin, Heidelberg, August 2001.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, Berlin, Heidelberg, March 2011.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 264–302. Springer, Cham, November 2017.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 171–190. IEEE Computer Society Press, October 2015.

[CCH$^+$19] Jung Hee Cheon, Wonhee Cho, Minki Hhan, Jiseung Kim, and Changmin Lee. Statistical zeroizing attack: Cryptanalysis of candidates of BP obfuscation over GGH15 multilinear map. In

Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 253–283. Springer, Cham, August 2019.

[CCMR24] Ran Canetti, Claudio Chamon, Eduardo Mucciolo, and Andrei Ruckenstein. Towards general-purpose program obfuscation via local mixing. Cryptology ePrint Archive, Report 2024/006, 2024.

[CDG$^+$18] David Bruce Cousins, Giovanni Di Crescenzo, Kamil Doruk Gür, Kevin King, Yuriy Polyakov, Kurt Rohloff, Gerard W. Ryan, and Erkay Savas. Implementing conjunction obfuscation under entropic ring LWE. In *2018 IEEE Symposium on Security and Privacy*, pages 354–371. IEEE Computer Society Press, May 2018.

[CFL$^+$16] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new CLT multilinear map over the integers. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 509–536. Springer, Berlin, Heidelberg, May 2016.

[CGH$^+$15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 247–266. Springer, Berlin, Heidelberg, August 2015.

[CGH17] Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 278–307. Springer, Cham, April / May 2017.

[CHL$^+$14] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. Cryptology ePrint Archive, Report 2014/906, 2014.

[CHVW19] Yilei Chen, Minki Hhan, Vinod Vaikuntanathan, and Hoeteck Wee. Matrix PRFs: Constructions, attacks, and applications to obfuscation. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 55–80. Springer, Cham, December 2019.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. Cryptology ePrint Archive, Report 2021/091, 2021.

[CLLT16] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 607–628. Springer, Berlin, Heidelberg, August 2016.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493. Springer, Berlin, Heidelberg, August 2013.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 577–607. Springer, Cham, August 2018.

[CZ21] Hongrui Cui and Kaiyi Zhang. A simple post-quantum non-interactive zero-knowledge proof from garbled circuits. In Yu Yu and Moti Yung, editors, *Information Security and Cryptology*, pages 269–280, Cham, 2021. Springer International Publishing.

[Dai22] Wei Dai. PESCA: A privacy-enhancing smart-contract architecture. Cryptology ePrint Archive, Report 2022/1119, 2022.

[DCBC$^+$16] Giovanni Di Crescenzo, Lisa Bahler, Brian Coan, Yuriy Polyakov, Kurt Rohloff, and David B Cousins. Practical implementations of program obfuscators for point functions. In *2016 International Conference on High Performance Computing & Simulation (HPCS)*, pages 460–467. IEEE, 2016.

[DEG$^+$18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low ANDdepth and few ANDs per bit. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 662–692. Springer, Cham, August 2018.

[DGH$^+$23] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Pasta:

A case for hybrid homomorphic encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):30–73, 2023.

[DJL$^+$24] Amit Deo, Marc Joye, Benoit Libert, Benjamin R. Curtis, and Mayeul de Bellabre. Fast homomorphic evaluation of lwr-based prfs. Cryptology ePrint Archive, Report 2024/665, 2024.

[DJM$^+$25] Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, and Vinod Vaikuntanathan. Simple and general counterexamples for private-coin evasive LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part VII*, volume 16006 of *Lecture Notes in Computer Science*, pages 73–92. Springer, Cham, August 2025.

[DQV$^+$21] Lalita Devadas, Willy Quach, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Succinct LWE sampling, random polynomials, and obfuscation. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 256–287. Springer, Cham, November 2021.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, Berlin, Heidelberg, May 2013.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 498–527. Springer, Berlin, Heidelberg, March 2015.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476. ACM Press, June 2013.

[GKR$^+$21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 519–535. USENIX Association, August 2021.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th Annual Symposium on Foundations of Computer Science*, pages 612–621. IEEE Computer Society Press, October 2017.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. Cryptology ePrint Archive, Report 2000/063, 2000.

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM Symposium on Theory of Computing*, pages 736–749. ACM Press, June 2021.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, Berlin, Heidelberg, August 2015.

[HHSS17] Shai Halevi, Tzipora Halevi, Victor Shoup, and Noah Stephens-Davidowitz. Implementing BP-obfuscation using graph-induced encoding. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 783–798. ACM Press, October / November 2017.

[HJL21] Samuel B. Hopkins, Aayush Jain, and Huijia Lin. Counterexamples to new circular security assumptions underlying iO. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 673–700, Virtual Event, August 2021. Springer, Cham.

[HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia Lin. Lattice-based post-quantum iO from circular security with random opening assumption. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part VII*, volume 16006 of *Lecture Notes in Computer Science*, pages 3–38. Springer, Cham, August 2025.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th Annual*

*Symposium on Foundations of Computer Science*, pages 415–434. IEEE Computer Society Press, November 2023.

[JLLS23] Aayush Jain, Huijia Lin, Paul Lou, and Amit Sahai. Polynomial-time cryptanalysis of the subspace flooding assumption for post-quantum $i\mathcal{O}$. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 205–235. Springer, Cham, April 2023.

[JLLW23] Aayush Jain, Huijia Lin, Ji Luo, and Daniel Wichs. The pseudorandom oracle model and ideal obfuscation. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 233–262. Springer, Cham, August 2023.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM Symposium on Theory of Computing*, pages 60–73. ACM Press, June 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in $NC^0$. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 670–699. Springer, Cham, May / June 2022.

[KMSV21] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 98–127. Springer, Cham, December 2021.

[KVMGH24] Christian Knabenhans, Alexander Viand, Antonio Merino-Gallardo, and Anwar Hithnawi. vfhe: Verifiable fully homomorphic encryption. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '24, pages 11–22, New York, NY, USA, 2024. Association for Computing Machinery.

[LMA$^+$16] Kevin Lewi, Alex J. Malozemoff, Daniel Apon, Brent Carmer, Adam Foltzer, Daniel Wagner, David W. Archer, Dan Boneh, Jonathan Katz, and Mariana Raykova. 5Gen: A framework for prototyping applications using multilinear maps and matrix branching programs. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 981–992. ACM Press, October 2016.

[LMS24] Dongwon Lee, Seonhong Min, and Yongsoo Song. Functional bootstrapping for packed ciphertexts via homomorphic LUT evaluation. Cryptology ePrint Archive, Report 2024/181, 2024.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, Berlin, Heidelberg, May 2007.

[LWSH23] Fucai Luo, Haiyan Wang, Al-Kuwari Saif, and Weihong Han. Multi-key fully homomorphic encryption without crs from rlwe. *Computer Standards & Interfaces*, 86:103742, 2023.

[Mi25] Machina-iO. Hello, world: The first signs of practical io. `https://machina-io.com/posts/hello_world_first.html`, 2025. (Accessed on 15/11/2025).

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, Berlin, Heidelberg, April 2012.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 735–763. Springer, Berlin, Heidelberg, May 2016.

[NRBB24] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. In Christina Pöpper and Lejla Batina, editors, *ACNS 2024: 22nd International Conference on Applied Cryptography and Network Security, Part III*, volume 14585 of *Lecture Notes in Computer Science*, pages 105–134. Springer, Cham, March 2024.

[Rev25] World Population Review. Internet speeds by country 2025. `https://worldpopulationreview.com/country-rankings/internet-speeds-by-country`, 2025. (Accessed on 21/11/2025).

[RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part IV*, volume 15367 of *Lecture Notes in Computer Science*, pages 3–36. Springer, Cham, December 2024.

[SA21] Ravital Solomon and Ghada Almashaqbeh. smartFHE: Privacy-preserving smart contracts from fully homomorphic encryption. Cryptology ePrint Archive, Report 2021/133, 2021.

[SAB$^+$22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`.

[SB25] Sora Suegami and Enrico Bottazzi. Lookup-table evaluation over key-homomorphic encodings and kp-abe for nonlinear operations. Cryptology ePrint Archive, Report 2025/1870, 2025.

[SBP25] Sora Suegami, Enrico Bottazzi, and Gayeong Park. Diamond iO: A straightforward construction of indistinguishability obfuscation from lattices. Cryptology ePrint Archive, Report 2025/236, 2025.

[Sta25] Lev Stambler. From signature-based witness encryption to ram obfuscation: Achieving blockchain-secured cryptographic primitives. *Cryptology ePrint Archive*, 2025.

[Sue24] Sora Suegami. iomaker - github repository. `https://github.com/SoraSuegami/iOMaker`, 2024. (Accessed on 21/11/2025).

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 535–559. Springer, Cham, August 2022.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 195–221. Springer, Cham, December 2022.

[Woo25] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger - shanghai version efc5f9a. `https://ethereum.github.io/yellowpaper/paper.pdf`, 2025. (Accessed on 19/11/2025).

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021,* *Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 127–156. Springer, Cham, October 2021.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In Chris Umans, editor, *58th Annual Symposium on Foundations of Computer Science*, pages 600–611. IEEE Computer Society Press, October 2017.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, October 1986.

[Zam25] Zama. Fhevm - a cross-chain protocol for confidential smart contracts. `https://github.com/zama-ai/fhevm/blob/main/fhevm-whitepaper.pdf`, 2025. (Accessed on 17/10/2025).

[Zon] Phantom Zone. local_mixing - github repository. `https://github.com/phantomzone-org/local_mixing`. (Accessed on 15/11/2025).

## A Gateway Contract Interface Specification

The orchestrator of the FHE-based private world computer is the Gateway contract[21]. Here is the definition of its interface functions:

- `registerCiphertext(ciphertext, contractAddress, proof)`: Takes as input a `ciphertext` and a smart contract address `contractAddress` along with a `proof` stating that the prover knows the plaintext behind the ciphertext, preventing malicious reuse, and grants permission to the smart contract, identified by its address, to handle the ciphertext. The function performs the verification of the proof and, on success, the ACL of the gateway contract is updated.

- `requestEvaluation(ciphertexts, circuit)`: Takes as input a set of `ciphertexts` and a `circuit` to be evaluated over them. The function checks that the caller has permission to request evaluation on the provided ciphertexts, based on the ACL. On success, it emits an event that triggers the FHE co-processor to perform the FHE evaluation of the provided circuit over the ciphertexts.

- `handleEvaluationResponse(ciphertext, proof)`: Takes as input the resulting `ciphertext` from the FHE evaluation along with a `proof` of correct evaluation. The function verifies the proof and, on success, updates

[21] The interface of the Gateway contract is inspired by Zama's fhevm gateway contract [Zam25] but not identical. As an example, their current design does not support SNARK proof verification for correct homomorphic evaluation.

the ACL to grant permission to the caller smart contract to handle the resulting ciphertext.

- `requestDecryption(ciphertext)`: Takes as input a `ciphertext` to be decrypted. The function checks that the caller has permission to request decryption of the provided ciphertext, based on the ACL. On success, it emits an event that triggers the decryption co-processor to perform the decryption of the ciphertext.

- `handleDecryptionResponse(plaintext, proof)`: Takes as input the resulting `plaintext` from the decryption along with a `proof` of correct decryption. The function verifies the proof and, on success, stores the plaintext in its state.

Users are expected to send their encrypted inputs to the application contract. The application contract must include the logic to interact with the gateway contract for registering the ciphertext, via `registerCiphertext`, performing FHE evaluation via `requestEvaluation` and `handleEvaluationResponse`, and performing decryption via `requestDecryption` and `handleDecryptionResponse`.