Smart Contracts Obfuscation from Blockchain-based One-time Program

Sora Suegami

Department of Information and Communication Engineering, The University of Tokyo
suegamisora@g.ecc.u-tokyo.ac.jp

April 29, 2022

Abstract

We propose a cryptographic obfuscation scheme for smart contracts from one-time programs using a blockchain, a garbled circuit, and witness encryption. The proposed scheme protects not only the privacy of its input data and states but also the privacy of its algorithm and hardcoded secrets. Its security depends on existing secure blockchains and does not require the honest majority of secure multiparty computation and trusted hardware. This scheme is more efficient than obfuscating an entire program with indistinguishability obfuscation. In addition, it needs a trusted setup, but its security is protected unless all participants of the setup process are malicious.

Keywords: privacy-preserving smart contract, blockchain, cryptographic obfuscation, garbled circuit, witness encryption

1 Introduction

In smart contracts, which are user-defined programs supported by blockchain protocols, there is inadequate privacy protection because a blockchain is a ledger shared among an unspecified number of anonymous computers in a peer-to-peer network, and inputs to the smart contract and the bytecode representing its functions are recorded on the blockchain.

We can categorize the privacies that smart contracts should protect into data and function privacies. The former is the privacy for input and state data of the smart contract, whereas the latter is the privacy for the smart contract algorithm and secrets hardcoded in its bytecode. Although function privacy is not as essential as data privacy in traditional applications using smart contracts, e.g., decentralized finance, it is crucial to help a company implement a program for its service as a smart contract because there would be a business problem if that program were available to competitors. Besides, by hardcoding a secret key in a bytecode of the smart contract, we can improve existing scaling solutions for blockchains, e.g., zero-knowledge (ZK) rollups [1]. For these reasons, we propose a solution that protects both data and function privacies.

Existing solutions for privacy-preserving smart contracts use either a noninteractive ZK (NIZK) proof system [2, 3, 4], secure multiparty computation (SMPC) [5, 6, 7, 8], or trusted hardware called trusted execution environment (TEE) [9, 10, 11]. The NIZK proof system cryptographically protects the data privacy of a limited class of smart contracts where only a single user's secrets are handled simultaneously but does not protect function privacy. Meanwhile, SMPC and TEE can protect data and function privacies for a smart contract that accesses multiuser secrets; however, their security is not solely based on cryptographic assumptions. An SMPC-based smart contract requires computations and communications by participants of SMPC, and its privacy protection is lost if its majority colludes. Besides, a TEE manufacturer can plant a backdoor in the TEE, which can break the privacy protection of a smart contract. Thus, we need a new solution whose security does not depend on such third parties. Cryptographic obfuscation could yield such a new solution.

1.1 Cryptographic Obfuscation

Obfuscation is a technique to hide the contents of a program while preserving its functionality. As such, anyone can execute an obfuscated program and obtain the same output as the source program, but they cannot know the algorithms or secrets included in the program. Cryptographic obfuscation is an obfuscation realized by cryptographic techniques, and its security is based on cryptographic assumptions. It can be used, for example, by software developers to distribute demo versions with limited functionalities to prevent reverse engineering of their software.

A smart contract that protects data and function privacies is constructed by hardcoding the secret key in its bytecode and obfuscating it with cryptographic obfuscation [12]. All of its input and state data are encrypted under the public key, and they are decrypted inside the obfuscated program. Its function privacy is directly guaranteed by the obfuscation security. Its data privacy is also guaranteed because an adversary cannot extract the hardcoded secret key from the obfuscated program. Therefore, cryptographic obfuscation enables a smart contract to cryptographically protect data and function privacies without requiring an honest majority of the SMPC participants or TEE.

However, there is no cryptographic obfuscation that makes an arbitrary program a black box. Particularly, it proved impossible to obfuscate an arbitrary program under "virtual black box security" [13]. Instead, cryptographic obfuscation that guarantees weak indistinguishability, called indistinguishability obfuscation (iO), would be feasible. The security of iO guarantees that if two programs of the same size and functionality (input-output relationship) are obfuscated, they will be indistinguishable. Recently, many iO candidates have been proposed, particularly in [14, 15], which showed that iO can be constructed based solely on already established cryptographic assumptions. However, even in the latest research [14, 16, 17, 18], the execution time is unrealistic. In summary, although the secure construction of iO exists under established assumptions, its efficiency remains impractical.

In addition to its inefficiency, iO is unsuitable to construct an obfuscated smart contract for two reasons[1]. First, its privacy protection is limited because iO only

[1]As far as our knowledge, there is no concrete proposal to realize a privacy-preserving

guarantees the indistinguishability of functionality-equivalent functions. That is, it only protects the function privacy for a smart contract that can theoretically realize the same functionality without no secret algorithm and values. Second, a smart contract can preserve a state, whereas iO cannot do that as it always returns the same output for the same input. Assuming that a smart contract obfuscated with iO takes as input a previous state and outputs a new state, we can provide any state as input to the obfuscated smart contract. As such, already provided states cannot be rejected. However, a smart contract, by specification, cannot use any state other than the latest one, so we have to resolve this difference.

1.2 Our Contribution

We propose a smart contract obfuscation scheme that solves the above problems. Our scheme is characterized by the following points.

1. It protects data and function privacies for wider classes of functions than iO.
2. Its security depends on a secure blockchain. Although it is not based solely on cryptographic assumptions, we can use existing proof-of-stake (POS) blockchain mechanisms without modifying their protocols.
3. It can preserve the state of an obfuscated smart contract, that is, only the latest state can be provided as the input.
4. It needs multiparty computation (MPC) for only the setup process; its security is maintained if at least one of the MPC participants is honest, and it does not require the participants to work after the setup process.

1.3 Literature Review

We categorize the existing studies for privacy-preserving smart contracts into NIZK, SMPC, and TEE-based solutions. Our scheme does not belong to any category as described in Subsection 1.1. We compare them in terms of the following features.

- Data Privacy: Privacy for the input and state data of the smart contract.
- Function Privacy: Privacy for the algorithm and hardcoded secrets of the smart contract.
- Trust Model: The strength of the trust model.

Their differences are summarized in Table 1.

Table 1: Comparison between previous works and this work.

	Data Privacy	Function Privacy	Trust Model
NIZK-based	Limited	Not protected	Cryptographic assumption
SMPC-based	Protected	Protected	Honest majority of participants
TEE-based	Protected	Protected	Trusted hardware
Our scheme	Protected	Protected	Existing secure blockchain

NIZK-based solutions

smart contract using iO except [12].

In the NIZK-based solutions, a user who holds some secrets generates a proof using the NIZK proof system, and that proof is verified on-chain, i.e., the verification process is performed by all computers participating in a blockchain protocol. The NIZK proof system guarantees that the proof leaks no secret under established cryptographic assumptions [19, 20, 21, 22]. The NIZK proof system was first adopted for specific applications. Zerocash uses this system to hide the remittance information of its crypto assets [23]. Aztec [24] and Tornado cash [25] also use it to improve the confidentiality and anonymity in transferring crypto assets on Ethereum blockchain. For general-purpose smart contracts, Steffen et al. [4] proposed a programing language named "zkay" that helps a smart contract developer implement a privacy-preserving smart contract with the NIZK proof system. This language allows the developer to specify data ownership of variable-kept secrets, whose encryption is recorded on a blockchain by the data owner. It automatically checks that the program is feasible using the NIZK proof system, e.g., its caller can access all secrets necessary for the proof generation, and prevents a vulnerability that implicitly leaks confidential information. Zether [2] is a privacy-preserving smart contract that has interoperability with arbitrary smart contracts. Although its core feature focuses on the confidential transfer of a crypto asset called "ZTH", by allowing a user to lock the ownership of his/her assets to other smart contracts, it can realize relevant applications, e.g., sealed-bid auctions without overcollateralization, confidential payment channel, and stake-based voting. Besides, unlike a smart contract written by zkay [4], Zether can be extended for protecting user anonymity. Zexe [26] is an exception in the NIZK-based solutions because it partially protects function privacy. It enables a user to execute offline computations and uses the NIZK proof system to prove its validity without revealing secrets. Although it only handles stateless computations, it can protect the privacy of not only confidential data but also its supported functions. KACHINA [3] is one of the most recent NIZK-based solutions. It proposes a generalized model for privacy-preserving smart contracts using the NIZK proof system. Particularly, previous NIZK-based solutions [2, 4, 26] are expressed as smart contracts in the KACHINA protocol. It also enables these contracts to interact without compromising their privacy protections.

SMPC-based solutions

The idea of combining SMPC and a blockchain was first considered in [5] for improving the security of SMPC. An incentive system that confiscates a deposit of maliciously behaving participants of the SMPC was introduced [5]. For protecting the data privacy of smart contracts, the SMPC has been used with a secret sharing scheme [6]. The user's secret inputs are divided into secret shares and then distributed to participants of the SMPC. To construct the secret a sufficient proportion of the secret shares is necessary, implying that an adversary who collects a sufficient number of secret shares can break the privacy protection for the input data. This security property has been improved by integrating the SMPC with homomorphic encryption (HE) [7], wherein the input data are first encrypted under HE and then divided into secret shares. Its decryption key is held only by a coordinator who coordinates SMPC participants, so the adversary cannot recover the input data from the collected shares. However, it is still insecure if that coordinator is malicious and colludes with a dishonest majority of participants.[2] Notably, function privacy has not been considered in the existing SMPC-based solutions, but we can generally upgrade them to protect the

[2]The authors of [8] also used HE with SMPC, where the input data were encrypted under the user's public key. Its privacy protection does not depend on any third parties, but it cannot handle a smart contract that requires multiuser secret inputs.

function privacy, using a universal circuit as a public function and a private function as a secret input [27, 28, 29].

TEE-based solutions

TEE, e.g., Intel SGX [30], is an isolated region inside a CPU. Even the owner of the hardware cannot access data and programs in this region, and this feature plays a significant role in the TEE-based solutions. Hawk [9] is one of the first frameworks for building a privacy-preserving smart contract. To protect data privacy it requires a trusted manager that can see user inputs, which can be instantiated with TEE. FASTKITTEN [10] proposed a practical scheme to protect privacy for smart contracts on existing public blockchains such as Bitcoin [31]. It executes a contract inside the isolated region so that an adversary cannot learn the information during its execution. Besides, it is highly efficient because an execution with TEE requires fewer overheads than cryptographic schemes. Secret Network [11] is a public blockchain that achieves privacy-preserving smart contracts in a similar to FASTKITTEN's scheme [10]. However, although a single TEE could execute smart contracts in [10], multiple parties called validators executed it with their TEE and made a consensus for its result in [11]. TEE is also useful for function privacy [32], which protects it by evaluating a universal circuit inside the isolated region. Despite the advantages of the TEE-based solutions, they have severe security issues. That is, a malicious TEE manufacturer can plant a backdoor that allows monitoring and tampering with the isolated region. Moreover, even if such a backdoor is not planted, an adversary that can physically access the hardware can access the isolated region by physical attacks, e.g., side-channel attacks [33].

2 Basic Idea

2.1 Construction from BOTP

Our scheme is built from one-time programs (OTPs) that use the POS blockchain. An OTP is a program that can be evaluated only on a single input specified at the evaluation time [34]. In other words, once an OTP has been evaluated, it cannot be evaluated on any other inputs. The first OTP construction proposed in [34] relied on trusted hardware called "one-time memory (OTM)". Goyal [35] replaced the OTM with POS blockchain and extractable witness encryption and proposed a software-based OTP, which we call blockchain-based OTP (BOTP). In the BOTP scheme, a program generator compiles a program, and its evaluator records a single input onto the blockchain [35]. The valid blockchain data, including its record, allows the evaluator to evaluate the compiled program on that input. Its output is equivalent to that of the source program, but the evaluator cannot obtain any information other than the input and output of the program, provided the evaluator cannot remove the record from the blockchain. Thus, this scheme obfuscates the smart contract for only a single input.

If a program compiled under the BOTP scheme can be evaluated on multiple inputs, such a scheme implies smart contract obfuscation. However, if its generator newly compiles a program for each evaluation, the scheme must continuously rely on that generator. In other words, if the generator terminates the compiling process, the evaluator cannot evaluate obfuscated smart contracts anymore. To solve this problem, we develop an updatable OTP scheme, which does not require a trusted generator except for the setup process, by compiling a program that outputs the newly compiled program. In the following description, the program is assumed to be represented as a Boolean circuit. In general, the circuit compiled in our scheme (defined as a recursive circuit) takes as input a bit string of an arbitrary circuit, compiles the given circuit, and outputs the compiled circuit. If the input is equivalent to a recursive circuit, the output is a compiled recursive circuit. By repeating the operation of inputting a recursive circuit to a recursive circuit, the evaluator can continue to generate new recursive circuits from a single recursive circuit. Therefore, although the evaluator can evaluate the circuit on multiple inputs, the only process that requires a trusted generator in our scheme is the setup process of compiling the first recursive circuit.

2.2 Security Definition

The security definition of our scheme is stronger than that of iO; although iO guarantees indistinguishability only for two circuits whose output is always the same for the same input, our scheme guarantees it, provided the two outputs are indistinguishable. For example, if two circuits encrypt different outputs under public key encryption (PKE) using internally generated random coins, an adversary who does not know the secret key required to decrypt them cannot distinguish their outputs. Therefore, our scheme can make them indistinguishable, although iO can not do that.

Specifically, we define data and function privacies as input and function indistinguishabilities, respectively. They are described as a game between an adversary and a challenger. In the game of the former definition, the adversary selects a source program and two inputs whose outputs are indistinguishable, sending them to the challenger. The challenger randomly chooses one of the received inputs and returns its encryptions. If the adversary cannot guess which input is encrypted with nontrivial probability, our scheme satisfies input indistinguishability. The game of the latter definition is described in the same manner, except that the adversary selects two source programs whose outputs are indistinguishable for the same input, and the challenger returns the obfuscation of the randomly chosen program. Our scheme protects both of them as shown in Subsection 4.5.

2.3 State Preservation

The modified construction of the original BOTP scheme allows obfuscated smart contracts to preserve states. We first review the construction proposed in [35]. In addition to the blockchain, it uses a garbled circuit and extractable witness encryption as its underlying scheme. The garbled circuit scheme is a cryptographic technique for encoding a circuit and its inputs, whose encoded circuit, called a garbled circuit, reveals nothing except its output [36]. Its encoded inputs, called wire keys, are generated for each bit of the input, and the corresponding wire keys are necessary to evaluate a garbled circuit at a specified input. The witness encryption scheme is an encryption scheme to encrypt a message to a particular problem instance in the NP language [37]. In the BOTP scheme, a circuit is encoded to a garbled circuit, and its wire keys are encrypted under the witness encryption scheme. We cannot directly release the wire keys for all inputs because the security of the garbled circuit is maintained only for a single input. That is, an adversary who holds wire keys for two different inputs can learn the information on the circuit. These encrypted wire keys are decrypted if the blockchain data that includes a record of only one input are provided. Therefore, unless removing the record from the blockchain, the adversary cannot obtain wire keys for more than two inputs.

In the origin construction, all wire keys were encrypted under the witness encryption scheme. However, if some bits of the input are fixed when the circuit is compiled, we can securely reveal the wire keys corresponding to the fixed bits. In our modified construction, such fixed bits are defined as constant inputs. The others are called variable inputs, which are provided during evaluation. The wire keys for the variable inputs are encrypted as with the original ones.

We realize the state preservation by fixing the next state as a constant input in the recursive circuit process. Specifically, the recursive circuit outputs wire keys corresponding to the bits of the next state encryption as a part of the compiled circuit. Because the wire keys corresponding to the opposite bits are not revealed, no other state can be provided to the compiled circuit.

2.4 Efficiency analysis

We discuss the efficiency of our scheme by analyzing the computational complexity of the process that depends on an impractical cryptographic scheme. In the BOTP scheme, the computational complexity of the process using the garbled circuit scheme is proportional to the circuit size, whereas that of the witness encryption scheme is described by the circuit input size. Recently, the garbled circuit scheme has been optimized by various scholars [38, 39, 40, 41, 42, 43], thus we can regard it as a practical scheme. However, the witness encryption scheme is still impractical because most of its candidates depend on multilinear maps [37, 44, 45] or iO [46, 47, 48, 49]. In this way, although the BOTP scheme still depends on the impractical cryptographic scheme, its computational complexity is $\mathcal{O}(n)$, where n is the input size. Because our scheme is an iterative application of the BOTP scheme, this efficiency analysis also applies to our scheme.

2.5 MPC for Decentralized Trusted Setup

A generator of an obfuscated smart contract can break its privacy protection because it knows all secrets hardcoded in the recursive circuit. Therefore, instead of relying on a single trusted generator, we employ generator decentralization using MPC. In the MPC, the hardcoded secrets are generated by a pseudorandom number generator (PRG) with a seed unknown to any MPC participant. Specifically, each participant submits a fresh PRG seed continuously, and the unknown seed is derived as the XOR of all submitted seeds. To compute the XOR with only revealing the output, we use an input-hiding technique of the BOTP scheme in [35]. Using the above mechanisms, if at least one of the MPC participants is honest, no participant can learn the hardcoded secrets and break the privacy protection of the obfuscated smart contract.

3 Preliminaries

3.1 Notations

Let $\mathbb{N}$ be the set of natural number and $\mathbb{R}$ be the set of real number. For $n \in \mathbb{N}$, $[n]$ represents a set of $\{1, \ldots, n\}$, and $[0]$ is defined as an empty set $\emptyset$. $x \overset{U}{\longleftarrow} \mathcal{X}$ is uniformly sampled from a distribution $\mathcal{X}$, where $\mathcal{X} \approx_c \mathcal{Y}$, $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable.

We will denote the security parameter by λ. A function $\mathrm{negl}(\lambda) : \mathbb{N} \to \mathbb{R}$ is said to be negligible, if there exists $n \in \mathbb{N}$ for any $c > 0$ such that $\mathrm{negl}(\lambda) < \lambda^{-c}$ holds for all $\lambda > n$.

$\mathcal{C}_{n,m}$ denotes a class of circuits that takes n bit inputs and returns m bit outputs. If a circuit C takes as input random coins r, C is a probabilistic circuit. When $C(x)$ outputs multiple types of values, we denote its i-th output value by $[C(x)]_i$. Let $U(\cdot, \cdot)$ be a universal circuit, which satisfies $U(C, x) = C(x)$ for all circuit $C \in \mathcal{C}_{n,m}$ and its input value $x \in \{0,1\}^n$.

We adopts definitions of $\mathrm{EXEC}^{\Gamma^V}(\mathcal{A}(x), \mathcal{Z}, 1^\lambda)$ and $\mathrm{view}_{\mathcal{A}}(\mathrm{EXEC}^{\Gamma^V}(\mathcal{A}(x), \mathcal{Z}, 1^\lambda))$ from [35].

3.2 Basic Cryptographic Schemes

We use the following functions of the basic cryptographic schemes. PKE.Enc and PKE.Dec are, respectively, encryption and decryption algorithms of IND-CPA secure public key encryption (PKE). Pseudorandom generator (PRG) is secure if $\mathrm{PRG}(s) \in \{0,1\}^{n_{PRG}} \approx_c r \overset{U}{\longleftarrow} \{0,1\}^{n_{PRG}}$ holds for all seed s. PRF is pseudorandom function. We assume that secure PRF with a PRF key K satisfies $\mathrm{PRF}(K, s) \in \{0,1\}^{n_{PRF}} \approx_c r \overset{U}{\longleftarrow} \{0,1\}^{n_{PRF}}$ for all seed s. KeyGen generates a key pair (sk, pk) of the PKE and a PRF key K. Commit and Open are, respectively, committing and opening algorithms of the commitment scheme.

3.3 BOTP

A BOTP scheme was first proposed in [35]. We modify its definition regarding the following points:

1. Our compiling algorithm takes as input random coins $r \overset{U}{\longleftarrow} \{0,1\}^*$ as a source of the randomness; if the same r is used, this algorithm always returns the equivalent output for the same input. Besides, this algorithm also takes as input $2n$ instances $\{x_i^b\}_{i\in[n],b\in\{0,1\}}$ in the NP language $\mathcal{L}$, for which the BOTP scheme is defined.
2. We divide an input $y \in \{0,1\}^n$ of the circuit $C \in \mathcal{C}_{n,m}$ into a constant input $y_c \in \{0,1\}^{n_c}$ and a variable input $y_v \in \{0,1\}^{n_v}$, where $n = n_c + n_v$. The constant input is determined in the compiling process, and the variable input is provided by the evaluator. Our compiling algorithm takes as input y_c, and outputs a compiled circuit CC whose constant input is fixed.
3. Our definition separates the algorithm of recording a variable input onto the blockchain from the evaluation algorithm. The recoding algorithm outputs wire keys for the variable inputs.
4. Our evaluation algorithm returns as output multiple bits, whereas that in [35] outputs a single bit.

In summary, our BOTP scheme comprises the polynomial algorithms Compile, Record, and Eval.

- **Compile**$(1^\lambda, C \in \mathcal{C}_{n,m}, \{x_i^b\}_{i\in[n],b\in\{0,1\}}, y_c \in \{0,1\}^{n_c}; r)$: Takes as input a security parameter λ, a circuit C, $2n$ instances $\{x_i^b\}_{i\in[n],b\in\{0,1\}}$, a constant input y_c, and random coins $r \overset{U}{\longleftarrow} \{0,1\}^*$. It outputs a compiled circuit CC.

- **Record**$(CC, y_v \in \{0,1\}^{n_v})$: Takes as input a compiled circuit CC and a variable input y_v. It outputs wire keys w for the variable input or the symbol $\perp$.
- **Eval**(CC, w): Takes as inputs a compiled circuit CC and wire keys w for the variable input. It outputs $z \in \{0,1\}^m$ or the symbol $\perp$.

The BOTP scheme defined for an NP language $\mathcal{L}$ and a class of circuits $\mathcal{C}_{n,m}$ satisfies the correctness if for all $\lambda, n, m, C \in \mathcal{C}_{n,m}, r \xleftarrow{U} \{0,1\}^*, y_c \in \{0,1\}^{n_c}, y_v \in \{0,1\}^{n_v}$, and $\{x_i^b \in \mathcal{L}\}_{i\in[n],b\in\{0,1\}}$,

$$\Pr[\text{Eval}(CC, \text{Record}(CC, y_v)) = C(y_v, y_c)] \geq 1 - \text{negl}(\lambda)$$

where $CC \leftarrow \text{Compile}(1^\lambda, C, \{x_i^b\}_{i\in[n],b\in\{0,1\}}, y_c; r)$.

The BOTP scheme is secure if it satisfies one-time security [35]. Informally, it guarantees that the information that an adversary can extract from a compiled circuit CC can be simulated by one-time access to the source circuit C, provided the number of forked blocks that the adversary can generate is bounded. Goyal [35] defined adaptive one-time secrecy (Definition 3.9 in [35]) and selective one-time secrecy (Definition 3.10 in [35]); the former allows an adversary to adaptively choose an input value after seeing the compiled circuit, whereas the latter is a weaker security notion because the adversary has to select an input value beforehand. The construction of the BOTP scheme in [35] only satisfies the selective one-time secrecy. Formally, they are defined as follows.

Definition 3.1 (Adaptive one-time secrecy)**.** A compiler of the BOTP scheme defined for an NP language $\mathcal{L}$ and a class of circuits $\mathcal{C}_{n,m}$ is a B/C-secure one-time compiler, if and only if (iff) for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists a PPT simulator Sim such that for all $\lambda, n, m, C \in \mathcal{C}_{n,m}, r \xleftarrow{U} \{0,1\}^*, \{x_i^b\}_{i\in[n],b\in\{0,1\}}, y_c \in \{0,1\}^{n_c}$, the following holds.

$$\begin{aligned}
&\{\text{view}_{\text{Sim}}(\text{EXEC}^{\Gamma^V}(\text{Sim}^{\mathcal{O}_C(\cdot,y_c)}(1^n, 1^{|C|}), \mathcal{Z}, 1^\lambda))\}\\
&\approx_c\\
&\{\text{view}_{\mathcal{A}}(\text{EXEC}^{\Gamma^V}(\mathcal{A}(CC), \mathcal{Z}, 1^\lambda)) : CC \leftarrow \text{Compile}(1^\lambda, C, \{x_i^b\}_{i\in[n],b\in\{0,1\}}, y_c; r)\}
\end{aligned}$$

where $\mathcal{O}_C(\cdot, y_c)$ provides one-time access to the circuit C with a constant input y_c.

Definition 3.2 (Selective one-time secrecy)**.** A compiler of the BOTP scheme defined for an NP language $\mathcal{L}$ and a class of circuits $\mathcal{C}_{n,m}$ is a B/C-selectively-secure one-time compiler, iff for every PPT adversary $\mathcal{A}$, there exists a PPT simulator Sim such that for all $\lambda, n, m, C \in \mathcal{C}_{n,m}, r \xleftarrow{U} \{0,1\}^*, \{x_i^b\}_{i\in[n],b\in\{0,1\}}, y_c \in \{0,1\}^{n_c}$, and $y_v \in \{0,1\}^{n_v}$, the following holds.

$$\begin{aligned}
&\{\text{view}_{\text{Sim}}(\text{EXEC}^{\Gamma^V}(\text{Sim}(1^n, 1^{|C|}, y_c, y_v, C(y_v, y_c)), \mathcal{Z}, 1^\lambda))\}\\
&\approx_c\\
&\{\text{view}_{\mathcal{A}}(\text{EXEC}^{\Gamma^V}(\mathcal{A}(CC), \mathcal{Z}, 1^\lambda)) : CC \leftarrow \text{Compile}(1^\lambda, C, \{x_i^b\}_{i\in[n],b\in\{0,1\}}, y_c; r)\}
\end{aligned}$$

where adversary $\mathcal{A}$ is admissible if it evaluates the OTP CC on (y_v, y_c) before evaluating on any other input.

In the proof of the selective one-time secrecy, the simulator Sim simulates a compiled circuit indistinguishable from the compiled circuit obtained from the Compile algorithm. (See Proof of Theorem 7.1 in [35]).

Lemma 3.1. If a compiler of the BOTP scheme defined for an NP language $\mathcal{L}$ and a class of circuits $\mathcal{C}_{n,m}$ is B/C-selectively-secure one-time compiler, for every PPT adversary $\mathcal{A}$, there exists a PPT simulator Sim_{CC} such that for all $\lambda, n, m, C \in \mathcal{C}_{n,m}, r \xleftarrow{U} \{0,1\}^*, \{x_i^b\}_{i\in[n],b\in\{0,1\}}, y_c \in \{0,1\}^{n_c}$, and $y_v \in \{0,1\}^{n_v}$, the following holds.

$$|\Pr[\mathcal{A}(CC) = 1 : CC \leftarrow \mathrm{Compile}(1^\lambda, C, \{x_i^b\}_{i\in[n],b\in\{0,1\}}, y_c; r)] -$$
$$\Pr[\mathcal{A}(\tilde{CC}) = 1 : \tilde{CC} \leftarrow \mathrm{Sim}_{CC}(1^n, 1^{|C|}, y_c, y_v, C(y_v, y_c))]| < \mathrm{negl}(\lambda)$$

Remark (Efficiency of the BOTP scheme). In the Compile algorithm, a circuit is encoded to a garbled circuit and its wire keys corresponding to the variable input are encrypted under the witness encryption scheme [35]. Therefore, the number of ciphertexts of the witness encryption scheme, which we call impractical, depends only on the input size of the source circuit C.

3.4 Smart Contract Obfuscation

There are four roles for the smart contract obfuscation participants: a generator, an obfuscator, a user, and an evaluator (Figure 1). The generator generates a common reference string, which includes a compiled circuit of the recursive circuit. The obfuscator publishes an obfuscated circuit using the common reference string. The user encrypts the user's input and sends it to the evaluator. The evaluator evaluates the obfuscated circuit on the encrypted input. The obfuscator obfuscates a probabilistic circuit $C_{sc} \in \mathcal{C}_{n,m}$ that represents functions of the smart contract (defined as a contract circuit).

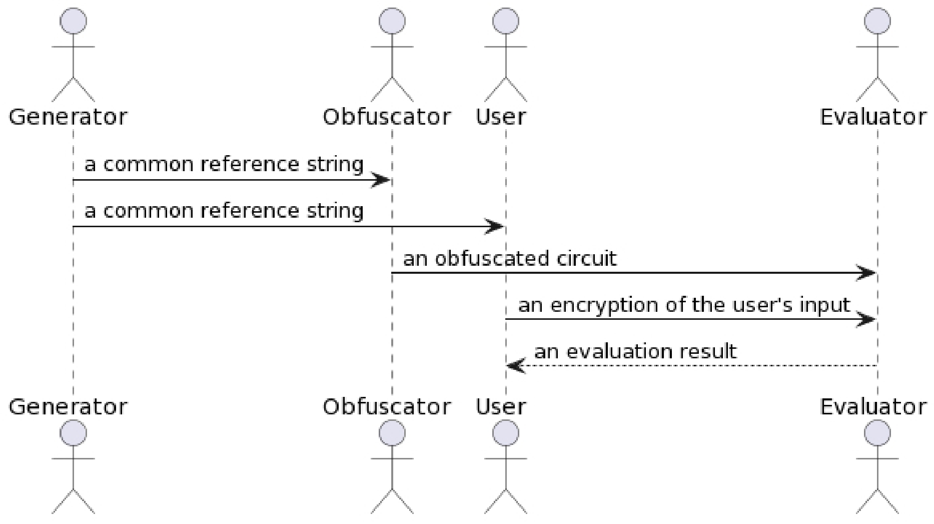

Figure 1: Relationships among smart contract obfuscation participants.

The contract circuit handles a state and a nonce, which is a number that indicates how many times the circuit has been evaluated. Specifically, it takes as inputs a value $y \in \{0,1\}^{n_y}$, a state $state_p \in \{0,1\}^{n_{s1}}$, a nonce $nc \in \{0,1\}^{n_{nc}}$ and random coins $r \overset{U}{\longleftarrow} \{0,1\}^{n_r}$, and returns an output $z \in \{0,1\}^{m_z}$, an updated state $state_n \in \{0,1\}^{m_{s2}}$, and an increased nonce $nc+1 \in \{0,1\}^{m_{nc}}$. (Notably, $n_{nc} = m_{nc}, n = n_y + n_{s1} + n_{nc} + n_r$, and $m = m_z + m_{s2} + m_{nc}$ holds.) When C_{sc} is obfuscated, its encrypted state and nonce are included in the obfuscated circuit, and updated for each evaluation.

The smart contract obfuscation scheme provides polynomial algorithms as follows.

- **Setup**(1^λ): Takes as input a security parameter λ, and outputs a common reference string crs.
- **Obfuscate**$(\mathrm{crs}, C_{sc} \in \mathcal{C}_{n,m}, state_0)$: Takes as input a common reference string crs, a contract circuit C_{sc}, and an initial state $state_0$ and outputs an obfuscated circuit C'.
- **Enc**(crs, y): Takes as input a common reference string crs and an input y and outputs an encrypted input ct_y.
- **Eval&Update**(C', ct_y): Takes as input an obfuscated circuit C' and an encrypted input ct_y and returns an output z and an updated circuit C'' or the symbol $\perp$. The updated circuit includes the encryption of the updated state and nonce, and it is used as C' in the next execution of this algorithm.

The above definition satisfies the correctness if for all $\lambda, n, m, C_{sc} \in \mathcal{C}_{n,m}, state$, and $y \in \{0,1\}^n$, the following holds.

$$\Pr[[\mathrm{Eval\&Update}(C', \mathrm{Enc}(\mathrm{crs}, y))]_1 = \mathrm{Enc}(\mathrm{crs}, [C_{sc}(y, state, nc; r)]_1]) \geq 1 - \mathrm{negl}(\lambda)$$

where $\mathrm{crs} \leftarrow \mathrm{Setup}(1^\lambda)$, $C' \leftarrow \mathrm{Obfuscate}(\mathrm{crs}, C_{sc}, state_0)$, $state$ and nc are the latest state and nonce when C' is evaluated for the nc times, and $r \overset{U}{\longleftarrow} \{0,1\}^{n_r}$ is random coins generated in the Eval&Update algorithm.

We present two definitions of game-based security for the smart contract obfuscation scheme: input and function indistinguishabilities. The former requires that the two encrypted inputs ct_{y_0}, ct_{y_1} to the obfuscated circuit of C_{sc} are indistinguishable, provided $C_{sc}(y_0, state, nc; r)$ and $C_{sc}(y_1, state, nc; r)$ are indistinguishable. The latter is satisfied iff obfuscated C^0_{sc} and C^1_{sc} are indistinguishable, for two contract circuits $C^0_{sc}, C^1_{sc} \in \mathcal{C}_{n,m}$ whose sizes are equivalent and outputs are indistinguishable (i.e., $C^0_{sc}(y, state, nc; r) \approx_c C^1_{sc}(y, state, nc; r)$ for all $y \in \{0,1\}^n$ and $r \overset{U}{\longleftarrow} \{0,1\}^{n_r}$). Notably, both definitions assume a selective adversary, i.e., the adversary chooses two inputs or circuits for the $(n+1)$-th evaluation before seeing the n-th updated circuit.

Definition 3.3 (Selective input indistinguishability)**.** An obfuscator of the smart contract obfuscation scheme satisfies the selective input indistinguishability iff for all PPT adversary $\mathcal{A}, \lambda, n, m, nc > 1$ and $\mathcal{C}_{n,m}$, the following holds.

$$\Pr[\mathrm{IND}^{\mathrm{input,sel}}_{\mathcal{A},\mathcal{C}_{n,m},nc}(1^\lambda) = 1] < \frac{1}{2} + \mathrm{negl}(\lambda)$$

where the game $\mathrm{IND}^{\mathrm{input,sel}}_{\mathcal{A},\mathcal{C}_{n,m},nc}(1^\lambda)$ is defined as follows:

- $\mathrm{IND}^{\mathrm{input,sel}}_{\mathcal{A},\mathcal{C}_{n,m}}(1^\lambda)$:
  1. The adversary chooses the first input y_0, an initial state $state_0$, the second input y_1, and a contract circuit $C_{sc} \in \mathcal{C}_{n,m}$: $(y_0, state_0, y_1, C_{sc}) \leftarrow \mathcal{A}(1^\lambda)$. The adversary sends $(C_{sc}, state_0)$ to the challenger.

2. The challenger setups a common reference string crs $\leftarrow$ Setup(1^λ), and provides crs to the adversary.
3. The adversary obfuscates C_{sc}: $C'_0 \leftarrow$ Obfuscate(crs, C_{sc}, $state_0$).
4. For $n \in [nc-2]$,
    (a) The adversary chooses the $(n{+}1)$-th input: $y_{n+1} \leftarrow \mathcal{A}(\mathrm{crs}, state_0, \{y_i\}_{i\in\{0\ldots n\}}, C_{sc})$, and sends y_{n+1} to the challenger.
    (b) The adversary evaluates C'_n on the input y_n and updates the circuit: $(z_n, C'_{n+1}) \leftarrow$ Eval&Update$(C'_n, \mathrm{Enc}(\mathrm{crs}, y_n))$.
5. The adversary chooses two nc-th inputs (y^0_{nc}, y^1_{nc}) that satisfies $|y^0_{nc}| = |y^1_{nc}|$ and $C_{sc}(y^0_{nc}, state_{nc-1}, nc-1; r) \approx_c C_{sc}(y^1_{nc}, state_{nc-1}, nc-1; r)$ for random coins $r \overset{U}{\leftarrow} \{0,1\}^{n_r}$: $(y^0_{nc}, y^1_{nc}) \leftarrow \mathcal{A}(\mathrm{crs}, state_0, \{y_i\}_{i\in\{0\ldots nc-1\}}, C_{sc})$. These inputs (y^0_{nc}, y^1_{nc}) are provided to the challenger.
6. The challenger selects a bit $b \in \{0,1\}$ randomly, and returns $ct_b \leftarrow$ Enc(crs, y^b_{nc}) to the adversary.
7. The adversary guesses b: $b' \leftarrow \mathcal{A}(\mathrm{crs}, state_0, \{y_i\}_{i\in\{0\ldots nc\}}, C_{sc}, ct_b)$.
8. The adversary outputs $b = b'$.

Definition 3.4 (Selective function indistinguishability)**.** An obfuscator of the smart contract obfuscation scheme satisfies the selective function indistinguishability iff for all PPT adversary $\mathcal{A}, \lambda, n, m, nc$ and $\mathcal{C}_{n,m}$, the following holds.

$$\Pr[\mathrm{IND}^{\mathrm{func,sel}}_{\mathcal{A},\mathcal{C}_{n,m}}(1^\lambda) = 1] < \frac{1}{2} + \mathrm{negl}(\lambda)$$

where the game $\mathrm{IND}^{\mathrm{func,sel}}_{\mathcal{A},\mathcal{C}_{n,m}}(1^\lambda)$ is defined as follows:

- $\mathrm{IND}^{\mathrm{func,sel}}_{\mathcal{A},\mathcal{C}_{n,m}}(1^\lambda)$:
    1. The adversary chooses the first input y_0, an initial state $state_0$, the second input y_1, and two contract circuits $C^0_{sc}, C^1_{sc} \in \mathcal{C}_{n,m}$ that satisfies $|C^0_{sc}| = |C^1_{sc}|$ and $C^0_{sc}(y, state, nc; r) \approx_c C^1_{sc}(y, state, nc; r)$ for all $state, nc, r \overset{U}{\leftarrow} \{0,1\}^{n_r}$: $(y_0, state_0, y_1 C^0_{sc}, C^1_{sc}) \leftarrow \mathcal{A}(1^\lambda)$. The adversary sends $(y_0, state_0, y_1, C^0_{sc}, C^1_{sc})$ to the challenger.
    2. The challenger setups a common reference string crs $\leftarrow$ Setup(1^λ) and provides crs to the adversary.
    3. The challenger selects a bit $b \in \{0,1\}$ randomly, and returns $C'_b \leftarrow$ Obfuscate(crs, C^b_{sc}, $state_0$).
    4. For $n \in [nc-1]$,
        (a) The adversary chooses the $(n+1)$-th input: $y_{n+1} \leftarrow \mathcal{A}(\mathrm{crs}, state_0, \{y_i\}_{i\in\{0\ldots n\}}, C^0_{sc}, C^1_{sc})$.
        (b) The adversary evaluates C'_b on the input y_n and updates the circuit: $(z_n, C'_b) \leftarrow$ Eval&Update$(C'_b, \mathrm{Enc}(\mathrm{crs}, y_n))$.
    5. The adversary guesses b: $b' \leftarrow \mathcal{A}(\mathrm{crs}, state_0, \{y_i\}_{i\in\{0\ldots nc\}}, C^0_{sc}, C^1_{sc}, C'_b)$.
    6. The adversary outputs $b = b'$.

4 Smart Contract Obfuscation from BOTP

4.1 System model

In our scheme, a smart contract comprises an obfuscated program and an on-chain program. An obfuscated program is an obfuscated contract circuit, whose inputs and states are encrypted. It is executed by an evaluator of the smart contract obfuscation scheme. An on-chain program is a bytecode deployed on the blockchain. Its input and state are available to participants of the blockchain network and executed by the associated transactions.

4.2 Setup process

To describe the definition of the setup process, we first specify how to construct a recursive circuit. The recursive circuit includes a secret key sk, its corresponding public key pk, and a PRF key K as hardcoded keys. It uses sk to decrypt its input, pk to encrypt its output state, and K to generate random coins, thereby inducing the randomness used for probabilistic algorithms.

In the recursive circuit, an NP language $\mathcal{L}$ that defines the BOTP scheme is also fixed. Our definition of $\mathcal{L}$ is equivalent to that defined in Subsection 7.1 of [35], except that ours includes a nonce nc. Let $\boldsymbol{B_1}, \boldsymbol{B_2}$ be blocks of the POS blockchain, n be the input size of the recursive circuit, and cId be an id of the contract (e.g., contract address). A pair of the instance $x = (\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, i, b \in \{0,1\}, \mathrm{cId}, nc)$ and witness $w = \boldsymbol{B_2}$ satisfies an NP relation R_L corresponding to $\mathcal{L}$, iff the following properties are satisfied [35].

1. Both $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ are valid blockchains.
2. $\boldsymbol{B_1}$ is an ancestor chain of $\boldsymbol{B_2}$. That is, the first different block between $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ follows the latest block in $\boldsymbol{B_1}$.
3. There is a unique block B^* in the different blocks between $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ such that
   - In a transaction that is associated with the contract of cId and included in B^*, an input value y is recorded, where the i-th bit of $y \in \{0,1\}^n$ is equivalent to b.
   - The nonce of the contract is increased in B^* from nc to $nc + 1$.
   - When ℓ' denotes the index of block B^* in the blockchain $\boldsymbol{B_2}$, $\ell' \geq \ell_1 + \ell_2$ holds and the fraction of unique stakes for the latest $\ell' - \ell_1$ blocks is larger than β.

In general, this definition indicates that an honest blockchain including the unique input value is necessary as a witness to decrypt the ciphertext of the witness encryption used in our scheme.

The recursive circuit $C_{rec} \in \mathcal{C}_{n,m}$ takes as inputs an encrypted input $ct_y \in \{0,1\}^{n_1}$, an encrypted state $ct_{state_p} \in \{0,1\}^{n_2}$, an encrypted circuit that will be compiled $ct_{C_1} \in \{0,1\}^{n_3}$, and an encryption $ct_{C_2} \in \{0,1\}^{n_4}$ of the zero-padded contract circuit $C_{sc} \in \mathcal{C}_{n_{sc},m_{sc}}$ whose size is $2n_{sc}$, and the nonce $nc \in \{0,1\}^{n_5}$. Therefore, its input size is $n = n_1 + n_2 + n_3 + n_4 + n_5$. It outputs the first output of the contract circuit $z \in \{0,1\}^{m_1}$, an encryption of the updated state $ct_{state_n} \in \{0,1\}^{m_2}$, and a compiled circuit $CC \in \{0,1\}^{m_3}$, so that its output size is $m = m_1 + m_2 + m_3$. Let n_y be a length of input y. We assume that ct_y is an encryption of y that is zero-padded so that its length is double of n_y, i.e., $ct_y = \mathrm{Enc}(\mathrm{crs}, pk, (y \in \{0,1\}^{n_y}, 0^{n_y}))$. The contract circuit $C_{sc} \in \{0,1\}^{n_{sc}}$ in ct_{C_2} is also zero-padded similarly. This padding is significant to achieve the input indistinguishability as shown in the proof of Lemma 4.2.

In summary, a recursive circuit C_{rec} is constructed as follows:

- $C_{rec}[1^\lambda, \boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, \mathrm{cId}, sk, pk, K](ct_y, ct_{state_p}, ct_{C_1}, ct_{C_2}, nc)$:
  1. $(y_{pad} \in \{0,1\}^{2n_y}, state_p, C_1, C_{pad} \in \{0,1\}^{2n_{nc}}) \leftarrow \mathrm{PKE.Dec}(sk, (ct_y, ct_{state_p}, ct_{C_1}, ct_{C_2}))$.
  2. Cut the left half of y_{pad} as $y \in \{0,1\}^{n_y}$ and ignore the right half.
  3. Cut the left half of C_{pad} as $C_2 \in \{0,1\}^{sc}$ and ignore the right half.
  4. $r \leftarrow \mathrm{PRF}(K, (\mathrm{cId}, ct_{C_1}, ct_{C_2}, nc))$.
  5. For each $i \in [n]$ and $b \in \{0,1\}$,
     $x_i^b = (\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, i, b, \mathrm{cId}, nc+1) \in \mathcal{L}$.
  6. $(z, state_n, nc+1) \leftarrow U(C_2, (y, state_p, nc; r))$.
  7. $ct_{state_n} \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, state_n; r)$.
  8. $CC \leftarrow \mathrm{BOTP.Compile}(1^\lambda, C_1, \{x_i^b\}_{i\in[n], b\in\{0,1\}}, (ct_{state_n}, ct_{C_1}, ct_{C_2}, nc{+}1); r)$.
  9. Output (z, ct_{state_n}, CC).

Next, we describe the specifications of the on-chain program. It stores in its storage a nonce nc and a mapping map_{y_v} from nc to the encrypted input ct_y. Besides, it must implement the RecordInputOnChain function. It is called in the Record algorithm of the BOTP scheme.

- **RecordInputOnChain**(ct_y):
  1. Store an encrypted input ct_y in map_y at nc.
  2. Increase nc to $nc + 1$.

Using the above recursive circuit and on-chain program, we formally define a Setup algorithm. In this algorithm, a generator generates sk, pk, and K and deploys a new on-chain program whose contract id is cId. A recursive circuit is constructed with the hardcoded values $(\mathrm{cId}, sk, pk, K)$. Then, the generator compiles and encrypts the recursive circuit, as well as outputs a common reference string crs. The crs comprises pk, cId, the compiled circuit CC_{rec}, and the encrypted circuit ct_{rec}.

- **Setup**(1^λ):
  1. Select apposite public parameters $(\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, \beta)$ for the security parameter 1^λ.
  2. Sample random coins $r \overset{U}{\leftarrow} \{0,1\}^*$.
  3. $(sk, pk, K) \leftarrow \mathrm{KeyGen}(1^\lambda; r)$.
  4. Deploy an on-chain program and obtain its contract id cId.
  5. Construct a recursive circuit with hardcoding the parameters and generated keys: $C_{rec}[1^\lambda, \boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, \mathrm{cId}, sk, pk, K]$.
  6. $ct_{rec} \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, C_{rec}; r)$.
  7. For each $i \in [n]$ and $b \in \{0,1\}$,
     $x_i^b = (\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, i, b, \mathrm{cId}, 0) \in \mathcal{L}$.
  8. $CC_{rec} \leftarrow \mathrm{BOTP.Compile}(1^\lambda, C_{rec}, \{x_i^b\}_{i\in[n], b\in\{0,1\}}, (ct_{C_1} = ct_{rec}, nc = 0); r)$.
  9. Output $C' \leftarrow (pk, \mathrm{cId}, CC_{rec}, ct_{rec})$.

4.3 Encryption and Obfuscation process

Our encryption and obfuscation processes are almost identical: a user encrypts a zero-padded input y under pk in the encryption process, whereas an obfuscator encrypts a zero-padded contract circuit C_{sc} and its initial state $state_0$ under pk. Afterward, the obfuscator evaluates the obfuscated circuit on a dummy input that does not modify the initial state. Because the first compiled circuit returned by the Setup algorithm takes the encryption of $(C_{sc}, 0^{|C_{sc}|})$ (i.e., ct_{C_2}) as variable input, the size of the input encrypted under the witness encryption schemes increases proportionally with the size of the contract circuit $|C_{sc}|$. Therefore, the obfuscator needs to perform the initial evaluation. Formally, an Enc algorithm and an Obfuscate algorithm are defined as follows.

- **Enc**(crs, y):
  1. Parse crs as $(pk, \text{cId}, CC_{rec}, ct_{rec})$.
  2. Sample random coins $r \xleftarrow{U} \{0,1\}^*$.
  3. $ct_y \leftarrow \text{PKE.Enc}(1^\lambda, pk, (y, 0^{|y|}); r)$.
  4. Output ct_y.
- **Obfuscate**$(\text{crs}, C_{sc} \in \mathcal{C}_{n,m}, state_0)$:
  1. Parse crs as $(pk, \text{cId}, CC_0, ct_{rec})$.
  2. Sample random coins $r \xleftarrow{U} \{0,1\}^*$.
  3. $ct_{C_{sc}} \leftarrow \text{PKE.Enc}(1^\lambda, pk, (C_{sc}, 0^{|C_{sc}|}); r)$.
  4. $ct_{state_0} \leftarrow \text{PKE.Enc}(1^\lambda, pk, state_0; r)$.
  5. Select a dummy input y_0.
  6. $ct_{y_0} \leftarrow \text{Enc}(\text{crs}, y_0)$.
  7. $w \leftarrow \text{BOTP.Record}(CC_0, (ct_y = ct_{y_0}, ct_{state_p} = ct_{state_0}, ct_{C_2} = ct_{C_{sc}}))$. In this process, a transaction to call $\text{RecordInputOnChain}((ct_{y_0}, ct_{state_0}, ct_{C_{sc}}))$ is broadcast, and a blockchain $\boldsymbol{B_2}$ including that transaction is used as a witness for the decryption of encrypted wire keys.
  8. $(z, ct_{state_1}, CC_1) \leftarrow \text{BOTP.Eval}(CC_0, w)$.
  9. Output $C' \leftarrow (pk, \text{cId}, CC_1, ct_{state_1}, ct_{rec}, ct_{C_{sc}}, 1)$.

4.4 Evaluation and Update process

The evaluation and update processes are executed simultaneously. That is, an evaluator can obtain the output of the circuit and the updated circuit simultaneously. The updated circuit is used in the next evaluation as the obfuscated circuit.

- **Eval&Update**(C', ct_y):
  1. Parse C' as $(pk, \text{cId}, CC_p, ct_{state_p}, ct_{rec}, ct_{C_{sc}}, nc)$.
  2. Sample random coins $r \xleftarrow{U} \{0,1\}^*$.
  3. $w \leftarrow \text{BOTP.Record}(CC_p, (ct_y = ct_{y_0}))$. In this process, a transaction to call $\text{RecordInputOnChain}(ct_{y_0})$ is broadcast, and a blockchain $\boldsymbol{B_2}$ including that transaction is used as a witness for the decryption of encrypted wire keys.

4. $(z, ct_{state_n}, CC_n) \leftarrow$ BOTP.Eval(CC_p, w).
5. $C'' \leftarrow (pk, \mathrm{cId}, CC_n, ct_{state_n}, ct_{rec}, ct_{C_{sc}}, nc+1)$
6. Output (z, C'').

We can prove the correctness of the above algorithms straightforwardly:

$$\begin{aligned}
&[\mathrm{Eval\&Update}(C', \mathrm{Enc}(\mathrm{crs}, y))]_1 \\
&= [\mathrm{Eval\&Update}((pk, \mathrm{cId}, CC_p, ct_{state_p}, ct_{rec}, ct_{C_{sc}}, nc), \mathrm{Enc}(\mathrm{crs}, y))]_1 \\
&= [\mathrm{BOTP.Eval}(CC_p, \mathrm{BOTP.Record}(CC_p, (\mathrm{Enc}(\mathrm{crs}, y))))]_1 \\
&= [C_{rec}(\mathrm{Enc}(\mathrm{crs}, y), (\mathrm{Enc}(\mathrm{crs}, state_p), \mathrm{Enc}(\mathrm{crs}, C_{rec}), \mathrm{Enc}(\mathrm{crs}, C_{sc}), nc))]_1 \\
&= [U(C_{sc}, (y, state_p, nc; r))]_1 \\
&= [C_{sc}(y, state_p, nc; r)]_1
\end{aligned}$$

4.5 Security Proof

Before proving the security of our scheme, we show input indistinguishability of the compiled recursive circuit CC_{rec}.

Claim 4.1. For two variable inputs y_0, y_1 and a contract circuit C_{sc}, if $C_{sc}(y_0, state_p, nc; r) \approx_c C_{sc}(y_1, state_p, nc; r)$ holds for random coins $r \overset{U}{\leftarrow} \{0,1\}^*$, then $(ct_{y_0}, CC_{rec}, C_{sc}(y_0, state_p, nc; r)) \approx_c (ct_{y_1}, CC_{rec}, C_{sc}(y_1, state_p, nc; r))$ also holds.

Lemma 4.2. If IND-CPA secure PKE, secure PRF, and a B/C-selectively-secure one-time compiler of the BOTP scheme exists, for all PPT adversaries $\mathcal{A}, \lambda, n_{sc}, m_{sc}, C_{sc} \in \mathcal{C}_{n_{sc}, m_{sc}}, state_p, nc, pk$, and two variable inputs y_0, y_1 that satisfies $|y_0| = |y_1|$ and $C_{sc}(y_0, state_p, nc; r) \approx_c C_{sc}(y_1, state_p, nc; r)$ for random coins $r \overset{U}{\leftarrow} \{0,1\}^*$, the following holds.

$$(ct_{y_0}, CC_{rec}, C_{sc}(y_0, state_p, nc; r)) \approx_c (ct_{y_1}, CC_{rec}, C_{sc}(y_1, state_p, nc; r))$$

where $ct_{y_0} \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, (y_0, 0^{|y_0|}); r)$, $ct_{y_1} \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, (y_1, 0^{|y_1|}); r)$, $CC_{rec} \leftarrow$ BOTP.Compile$(1^\lambda, C_{rec}, \{x_i^b\}_{i \in [n], b \in \{0,1\}}, (ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}, nc); r)$, $ct_{state_p} \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, state_p)$, $ct_{C_{rec}} \leftarrow$ PKE.Enc$(1^\lambda, pk, C_{rec})$, $ct_{C_{sc}} \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, C_{sc})$, and $C_{rec}, \{x_i^b\}_{i \in [n], b \in \{0,1\}}, r$ are defined in Subsection 4.2.

Proof. We define a sequence of hybrid experiments.

Hybrid 1: This hybrid corresponds to a compiled recursive circuit CC_{rec} whose input is $ct_y \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, (y_0, 0^{|y_0|}); r)$.

Hybrid 2: This hybrid is the same as Hybrid 1, except that CC_{rec} is generated by a simulator
$\mathrm{Sim}_{CC}(1^\lambda, 1^{|C_{rec}|}, (ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}, nc), C_{rec}(ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}, nc))$.

Hybrid 3: This hybrid is the same as Hybrid 2, except that the input is $ct_y \leftarrow$ PKE.Enc$(1^\lambda, pk, (y_0, y_1); r)$.

Hybrid 4: This hybrid is the same as Hybrid 3, except that the CC_{rec} is generated by a simulator
$\mathrm{Sim}_{CC}(1^\lambda, 1^{|C'_{rec}|}, (ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}, nc), C'_{rec}(ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}, nc))$, where C'_{rec} is defined as follows:

- $C'_{rec}[1^\lambda, \boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, \mathrm{cId}, sk, pk, K](ct_y, ct_{state_p}, ct_{C_1}, ct_{C_2}, nc)$:
    1. $(y_{pad} \in \{0,1\}^{2n_y}, state_p, C_1, C_{pad} \in \{0,1\}^{2n_{sc}}) \leftarrow \mathrm{PKE.Dec}(sk, (ct_y, ct_{state_p}, ct_{C_1}, ct_{C_2}))$.

2. **Cut the right half of y_{pad} as $y \in \{0,1\}^{n_y}$ and ignore the left half.**
3. Cut the left half of C_{pad} as $C_2 \in \{0,1\}^{sc}$ and ignore the right half.
4. $r \leftarrow \mathrm{PRF}(K, (\mathrm{cId}, ct_{C_1}, ct_{C_2}, nc))$.
5. For each $i \in [n]$ and $b \in \{0,1\}$,
   $x_i^b = (\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, i, b, \mathrm{cId}, nc+1) \in \mathcal{L}$.
6. $(z, state_n, nc+1) \leftarrow U(C_2, (y, state_p, nc))$.
7. $ct_{state_n} \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, state_n; r)$.
8. $CC \leftarrow \mathrm{BOTP.Compile}(1^\lambda, C_1, \{x_i^b\}_{i\in[n],b\in\{0,1\}}, (ct_{state_n}, ct_{C_1}, ct_{C_2}, nc{+}1); r)$.
9. Output (z, ct_{state_n}, CC).

Hybrid 5: This hybrid is the same as Hybrid 4, except that the input is $ct_y \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, (y_1, y_1); r)$.

Hybrid 6: This hybrid is the same as Hybrid 5, except that the CC_{rec} is generated by a simulator
$\mathrm{Sim}_{CC}(1^\lambda, 1^{|C_{rec}|}, (ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}, nc), C_{rec}(ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}, nc))$.

Hybrid 7: This hybrid is the same as Hybrid 6, except that the input is $ct_y \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, (y_1, 0^{|y_1|}); r)$.

Hybrid 8: This hybrid corresponds to the compiled recursive circuit CC_{rec} whose input is $ct_y \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, (y_1, 0^{|y_1|}); r)$.

Assuming the IND-CPA security of PKE, PRF security, and a B/C-selectively-secure one-time compiler of the BOTP scheme, no PPT adversary can distinguish any contiguous hybrids with nontrivial advantage.

Indistinguishability between Hybrid 1 and 2: The only difference between Hybrid 1 and 2 is the use of BOTP.Compile algorithm or the simulator Sim_{CC} to generate CC_{rec}. Therefore, adversary $\mathcal{A}$, which cannot break the selective one-time secrecy of the BOTP scheme, cannot distinguish them.

Indistinguishability between Hybrid 2 and 3: The indistinguishability between Hybrid 2 and 3 is directly proved by the IND-CPA security of PKE.

Indistinguishability between Hybrid 3 and 4: The only difference between Hybrid 3 and 4 is the first output, i.e., the difference between $[C_{rec}(ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}]_1 = [C_{sc}(y_0, state_p, nc; r)]_1$ and $[C'_{rec}(ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}]_1 = [C_{sc}(y_1, state_p, nc; r)]_1$. Because $r = \mathrm{PRF}(K, (\mathrm{cId}, ct_{C_1}, ct_{C_2}, nc))$ is indistinguishable from true random number and $C_{sc}(y_0, state_p, nc; r) \approx_c C_{sc}(y_1, state_p, nc; r)$ holds, the adversary $\mathcal{A}$ cannot distinguish Hybrid 3 and 4.

Indistinguishability between Hybrid 4 and 5: The indistinguishability between Hybrid 4 and 5 is directly proved by the IND-CPA security of PKE.

Indistinguishability between Hybrid 5 and 6: There is no difference between Hybrid 5 and 6 because $C'_{rec}(ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}}) = C_{rec}(ct_y, ct_{state_p}, ct_{C_{rec}}, ct_{C_{sc}})$, where $ct_y \leftarrow \mathrm{PKE.Enc}(1^\lambda, pk, (y_1, y_1); r)$.

Indistinguishability between Hybrid 6 and 7: The indistinguishability between Hybrid 6 and 7 is directly proved by the IND-CPA security of PKE.

Indistinguishability between Hybrid 7 and 8: The only difference between Hybrid 7 and 8 is the use of BOTP.Compile algorithm or the simulator Sim_{CC} to generate CC_{rec}. Therefore, adversary $\mathcal{A}$ cannot distinguish them under the security of the B/C-selectively-secure one-time compiler.

Because Hybrid 1 and 8 are indistinguishable as above, this lemma follows. □

From Lemma 4.2, we can prove both selective input and function indistinguishabilities.

Theorem 4.3. If IND-CPA secure PKE, secure PRF, and a B/C-selectively-secure one-time compiler of the BOTP scheme exists, the smart contract obfuscation scheme defined in Section 4 satisfies selective input indistinguishability (Definition 3.3).

Proof. We define a sequence of games.

Game 1: This game corresponds to $\mathrm{IND}^{\text{input,sel}}_{\mathcal{A},\mathcal{C}_{n,m},nc}(1^\lambda)$ in Definition 3.3.

Game 2: This game is equivalent to Game 1, except that the challenger always returns $ct_0 = \mathrm{Enc}(\mathrm{crs}, y^0_{nc})$ in Step 6.

In Game 1, the encrypted input ct_b returned by the challenger depends on $b \in \{0,1\}$, whereas, in Game 2, the encryption corresponding to $b = 0$ is always selected. Because the adversary $\mathcal{A}$ chooses two vairable inputs y^0_{nc}, y^1_{nc} that $|y^0_{nc}| = |y^1_{nc}|$ and $C_{sc}(y^0_{nc}, state_p, nc; r) \approx_c C_{sc}(y^1_{nc}, state_p, nc; r)$ holds for random coins $r \overset{U}{\longleftarrow} \{0,1\}^*$, before seeing n-th CC_{rec}, $\mathcal{A}$ cannot distinguish $(ct_0, CC_{rec}, C_{sc}(y^0_{nc}, state_p, nc; r))$ and $(ct_1, CC_{rec}, C_{sc}(y^1_{nc}, state_p, nc; r))$ from Lemma 4.2. Therefore, even if $b = 1$ is selected in Game 1, i.e., the input chosen by the challenger in Game 1 and Game 2 are different, Game 1 and Game 2 are indistinguishable for $\mathcal{A}$. Game 2 does not depend on b, thus, this theorem follows. □

Theorem 4.4. If IND-CPA secure PKE, secure PRF, and a B/C-selectively-secure one-time compiler of the BOTP scheme exists, the smart contract obfuscation scheme defined in Section 4 satisfies selective function indistinguishability (Definition 3.4).

Proof. We define a sequence of games.

Game 1: This game corresponds to $\mathrm{IND}^{\text{func,sel}}_{\mathcal{A},\mathcal{C}_{n,m},nc}(1^\lambda)$ in Definition 3.4.

Game 2: This game is equivalent to Game 1, except that the challenger always returns $C'_0 = \mathrm{Obfuscate}(\mathrm{crs}, C^0_{sc}, state_0)$ in Step 3.

We prove the indistinguishability between Game 1 and 2 in the same manner as in the proof of Theorem 4.3, by regarding a universal circuit U as a contract circuit C_{sc} and C^b_{sc} as a part of the input of U. In Game 1, the obfuscated circuit C'_b returned by the challenger depends on $b \in \{0,1\}$, whereas the circuit corresponding to $b = 0$ is always selected in Game 2. In Step 8 of the Obfuscate algorithm, the compiled recursive circuit CC_{rec} is evaluated on the variable input $(ct_{y_0}, ct_{state_0}, ct_{C^b_{sc}})$. Because adversary $\mathcal{A}$ chooses two circuits C^0_{sc}, C^1_{sc} that $|C^0_{sc}| = |C^1_{sc}|$ and $U(C^0_{sc}, y_0, state_0, 0; r) = C^0_{sc}(y_0, state_0, 0; r) \approx_c U(C^1_{sc}, y_0, state_0, 0; r) = C^1_{sc}(y_0, state_0, 0; r)$ holds for random coins $r \overset{U}{\longleftarrow} \{0,1\}^*$, before seeing the first CC_{rec}, $\mathcal{A}$ cannot distinguish $(ct_{C^0_{sc}}, CC_{rec}, U(C^0_{sc}, y_0, state_0, 0; r))$ and $(ct_{C^1_{sc}}, CC_{rec}, U(C^1_{sc}, y_0, state_0, 0; r))$ from Lemma 4.2. Therefore, even if $b = 1$ is selected in Game 1, i.e., the circuit chosen by the challenger in Game 1 and 2 are different, Game 1 and 2 are indistinguishable for $\mathcal{A}$. Game 2 does not depend on b, thus this theorem follows. □

5 MPC for Decentralized Trusted Setup

5.1 System model

First, we only consider the case in which MPC participants extracting the maximum secrets from other participants' data, but follow the protocol honestly, i.e., semi-honest adversaries. If participants may publish invalid data deviating from the protocol, they can be verified by the NIZK proof system or detected by a cut-and-choose technique, which is frequently adopted in the garbled circuit scheme against malicious adversaries [50, 51, 52].

5.2 Circuit Construction for Decentralized Setup

In the Setup algorithm in Subsection 4.2, random coins r are sampled to generate the hardcoded keys sk, pk, K, and encrypt/compile a recursive circuit. Our MPC aims to sample r from a PRG seed s that no one knows. If the MPC has two participants, the XOR of the first participant's seed s_1 and the second participant's seed s_2 can be used. To perform the obfuscation algorithm without revealing each other's seed, we use the BOTP scheme: the first participant compiles the setup circuit G_2 which includes s_1, and the second participant evaluates it by providing the new seed s_2 as an input. This setup circuit compiles and encrypts the recursive circuit with random coins generated from $s_1 \oplus s_2$. The second participant cannot record s_2 itself onto the blockchain because s_2 would be revealed. Instead, the second participant records its commitment and includes its actual seed and its opening in the witness for the witness encryption scheme. Notably, this technique has already been proposed in [35] as an input-hiding technique. Let s and r_{open} be an actual seed and an opening of the recorded commitment, respectively, where the seed s is provided by the j-th participant. Then, the NP language $\mathcal{L}'$ used in the setup process is modified as follows: a pair of the instance $x = (\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, i, b \in \{0,1\}, \mathrm{cId}, j)$ and witness $w = (\boldsymbol{B_2}, s, r_{open})$ satisfies an NP relation $R_{L'}$ corresponding to $\mathcal{L}'$, iff the following properties are satisfied.

1. Both $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ are valid blockchains.
2. $\boldsymbol{B_1}$ is an ancestor chain of $\boldsymbol{B_2}$. In other words, the first different block between $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ follows the latest block in $\boldsymbol{B_1}$.
3. There is a unique block B^* in the different blocks between $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ such that
    - In a transaction associated with the contract of cId and included in B^*, an input commitment c is recorded, and $s = \mathrm{Open}(c, r_{open})$ holds, i.e., c is a valid commitment for s.
    - The i-th bit of $s \in \{0,1\}^n$ is equivalent to b.
    - When ℓ' denotes the block height of B^*, $\ell' \geq \ell_1 + \ell_2$ holds and the fraction of unique stakes for the latest $\ell' - \ell_1$ blocks is larger than β.

The above scheme is easily generalized to the case of N participants (Figure 2). The PRG seed s after the setup process will be XOR of each participant's seed: $s = s_1 \oplus s_2 \oplus \cdots \oplus s_N$. In the setup process among $N > 2$ participants, the setup between two parties is performed continuously. The first participant samples the seed s_1, and compiles the setup circuit G_N that includes s_1 as the hardcoded seed. It also takes as input another seed s_2, but instead of the recursive circuit C_{rec}, it compiles G_N whose hardcoded seed is $s_1 \oplus s_2$. Therefore, the third participant can feed a new seed s_3 to the compiled circuit, and generate a compiled circuit that includes $s_1 \oplus s_2 \oplus s_3$.

Formally, the setup circuit G_N evaluated by the j-th participant is defined as follows:

- $G_N[1^\lambda, \boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, \mathrm{cId}, N, j, s_1](s_2, ct_C)$:
    1. $r_{pre} \leftarrow PRG(s_1)$.

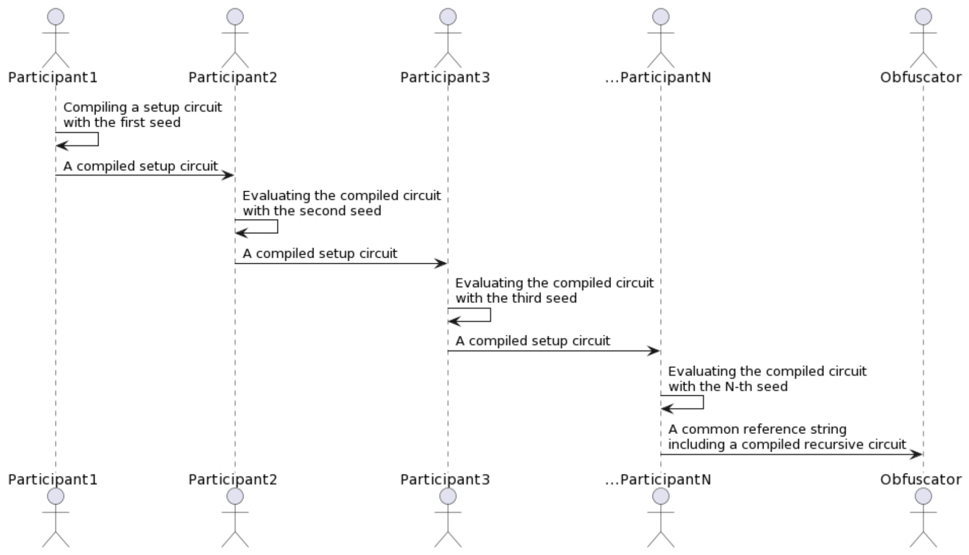
Participant1
Participant2
Participant3
...ParticipantN
Obfuscator
Compiling a setup circuit with the first seed
A compiled setup circuit
Evaluating the compiled circuit with the second seed
A compiled setup circuit
Evaluating the compiled circuit with the third seed
A compiled setup circuit
Evaluating the compiled circuit with the N-th seed
A common reference string including a compiled recursive circuit

Figure 2: Decentralized trusted setup for a smart contract obfuscation scheme.

2. $(sk_{pre}, \cdot, \cdot) \leftarrow \text{KeyGen}(1^\lambda; r_{pre})$.
3. $C \leftarrow \text{PKE.Dec}(sk_{pre}, ct_C)$
4. $s \leftarrow s_1 \oplus s_2$.
5. $r \leftarrow PRG(s)$.
6. If $j < N$,
   (a) Hardcode $[1^\lambda, \boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, \text{cId}, N, j+1, s]$ to C.
   (b) $ct'_C \leftarrow \text{PKE.Enc}(1^\lambda, pk, C; r)$.
   (c) For each $i \in [n]$ and $b \in \{0,1\}$,
   $x_i^b = (\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, i, b, \text{cId}, j+1) \in \mathcal{L}'$.
   (d) $CC \leftarrow \text{BOTP.Compile}(1^\lambda, C, \{x_i^b\}_{i\in[n], b\in\{0,1\}}, (ct_C = ct'_C); r)$.
   (e) Output (CC, ct'_C).

   Otherwise,
   (a) $(sk, pk, K) \leftarrow \text{KeyGen}(1^\lambda; r)$.
   (b) Hardcode $[1^\lambda, \boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, \text{cId}, sk, pk, K]$ to C.
   (c) $ct'_C \leftarrow \text{PKE.Enc}(1^\lambda, pk, C; r)$.
   (d) For each $i \in [n]$ and $b \in \{0,1\}$,
   $x_i^b = (\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, i, b, \text{cId}, 0) \in \mathcal{L}$.
   (e) $CC \leftarrow \text{BOTP.Compile}(1^\lambda, C, \{x_i^b\}_{i\in[n], b\in\{0,1\}}, (ct_{C_1} = ct'_C, nc = 0); r)$.
   (f) Output $(pk, \text{cId}, CC, ct'_C)$.

5.3 Modification of the On-Chain Program

When the setup is processed with MPC, the on-chain program described in Subsection 4.2 additionally stores the latest index j' of the participant who already recorded the commitment, a mapping map_c from j' to the commitment c, and a mapping map_{cc} from j' to the compiled circuit CC. Besides, the on-chain program implements the **RecordCommitOnChain** and **UpdateCircuit** functions:

- **RecordCommitOnChain**(c):
    1. $j' \leftarrow j' + 1$.
    2. Store a commitment c in map_c at j'.
- **UpdateCircuit**(CC):
    1. Store a compiled circuit CC in map_{cc} at j'.

5.4 New Setup Process with MPC

The setup process differs for the first participant ($j = 1$), the latest participant ($j = N$), and the other participants; the first participant directly compiles the setup circuit G_N in which the seed s_1 is hardcoded, and the latest participant inputs the recursive circuit C_{rec} to the compiled setup circuit, which outputs the compiled recursive circuit. Specifically, the j-th participant performs the following algorithm. (We assume that the on-chain program defined in Subsection 5.3 is already deployed before the MPC, and its contract id is cId.)

- **Setup**($1^\lambda, \text{cId}, N, j$):
    1. Select apposite public parameters $(\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, \beta)$ for the security parameter 1^λ.
    2. Sample a random seed $s \xleftarrow{U} \{0,1\}^*$.
    3. $r \leftarrow \text{PRG}(s)$.
    4. If $j = 1$,
        (a) Construct a setup circuit $G_N[1^\lambda, \boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, \text{cId}, N, 2, s]$.
        (b) $(\cdot, pk, \cdot) \leftarrow \text{KeyGen}(1^\lambda; r)$.
        (c) $ct_{G_N} \leftarrow \text{PKE.Enc}(1^\lambda, pk, G_N; r)$.
        (d) For each $i \in [n]$ and $b \in \{0,1\}$,
        $x_i^b = (\boldsymbol{B_1}, 1^{\ell_1}, 1^{\ell_2}, 1^n, \beta, i, b, \text{cId}, 2) \in \mathcal{L}'$.
        (e) $CC_2 \leftarrow \text{BOTP.Compile}(1^\lambda, G_N, \{x_i^b\}_{i\in[n], b\in\{0,1\}}, (ct_C = ct_{G_N}); r)$.

        If $1 < j < N$,

        (a) Obtain the latest index j' from the on-chain program.
        (b) If $j' \neq j - 1$, throw an error.
        (c) Obtain a compiled circuit C_1' stored in map_{cc} at j'.
        (d) $(c, r_{open}) \leftarrow \text{Commit}(1^\lambda, s; r)$.
        (e) $w \leftarrow \text{BOTP.Record}(C_1', c)$. In this process, a transaction to call RecordCommitOnChain(c) is broadcast. A blockchain $\boldsymbol{B_2}$ including that transaction is used as a witness for the decryption of encrypted wire keys.
        (f) $(CC_2, ct_{G_N}) \leftarrow \text{BOTP.Eval}(CC_1, w)$.

        If $j = N$,

        (a) Obtain the latest index j' from the on-chain program.

(b) If $j' \neq j - 1$, throw an error.

(c) Obtain a compiled circuit CC_1 stored in map_{cc} at j'.

(d) $(c, r_{open}) \leftarrow \text{Commit}(1^\lambda, s; r)$.

(e) $w \leftarrow \text{BOTP.Record}(CC_1, c)$. In this process, a transaction to call RecordCommitOnChain(c) is broadcast. A blockchain $\boldsymbol{B_2}$ including that transaction is used as a witness for the decryption of encrypted wire keys.

(f) $(pk, \text{cId}, CC_2, ct_{G_N}) \leftarrow \text{BOTP.Eval}(CC_1, w)$.

5. Broadcast a transaction to call UpdateCircuit(C_2').
6. If $j = N$, Output $(pk, \text{cId}, CC_2, ct_{G_N})$ as the output of the setup process.

6 Conclusion

We developed smart contract obfuscation based on the BOTP scheme. It protects data privacy and function privacy for a wider class of smart contracts than NIZK-based privacy-preserving smart contracts. Its privacy protection is more secure than the SMPC and TEE-based solutions because it only requires existing secure blockchains. Our construction still depends on an impractical cryptographic scheme, i.e., witness encryption, but its computational complexity depends on only the circuit input size so that it is superior to obfuscating an entire program of the smart contract in terms of efficiency. Although our scheme requires a trusted setup, we can decentralize it via MPC.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work has been supported by the Mohammed bin Salman Center for Future Science and Technology for Saudi-Japan Vision 2030 at The University of Tokyo (MbSC2030).

References

[1] How obfuscation can help ethereum - cryptography - ethereum research, `https://ethresear.ch/t/how-obfuscation-can-help-ethereum/7380`, (Accessed on 03/26/2022).

[2] B. Bünz, S. Agrawal, M. Zamani, D. Boneh, Zether: Towards privacy in a smart contract world, in: International Conference on Financial Cryptography and Data Security, Springer, 2020, pp. 423–443.

[3] T. Kerber, A. Kiayias, M. Kohlweiss, Kachina–foundations of private smart contracts, in: 2021 IEEE 34th Computer Security Foundations Symposium (CSF), IEEE, 2021, pp. 1–16.

[4] S. Steffen, B. Bichsel, M. Gersbach, N. Melchior, P. Tsankov, M. Vechev, zkay: Specifying and enforcing data privacy in smart contracts, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1759–1776.

[5] M. Andrychowicz, S. Dziembowski, D. Malinowski, L. Mazurek, Secure multiparty computations on bitcoin, in: 2014 IEEE Symposium on Security and Privacy, IEEE, 2014, pp. 443–458.

[6] Y. Zhu, X. Song, S. Yang, Y. Qin, Q. Zhou, Secure smart contract system built on smpc over blockchain, in: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2018, pp. 1539–1544.

[7] Z. Li, R. Zhang, P. Li, A secure and efficient smart contract execution scheme, in: International Conference on Web Services, Springer, 2020, pp. 17–32.

[8] X. Pei, L. Sun, X. Li, K. Zheng, X. Wu, Smart contract based multi-party computation with privacy preserving and settlement addressed, in: 2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), IEEE, 2018, pp. 133–139.

[9] A. Kosba, A. Miller, E. Shi, Z. Wen, C. Papamanthou, Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, in: 2016 IEEE symposium on security and privacy (SP), IEEE, 2016, pp. 839–858.

[10] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, A.-R. Sadeghi, {FastKitten}: Practical smart contracts on bitcoin, in: 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 801–818.

[11] Secret network: A privacy-preserving secret contract & dapp platform, `https://scrt.network/graypaper`, (Accessed on 04/26/2022).

[12] Privacy on the blockchain — ethereum foundation blog, `https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain/`, (Accessed on 03/11/2022).

[13] R. Pass, A. Shelat, Impossibility of vbb obfuscation with ideal constant-degree graded encodings, in: Theory of Cryptography Conference, Springer, 2016, pp. 3–17.

[14] A. Jain, H. Lin, A. Sahai, Indistinguishability obfuscation from lpn over f_p, dlin, and prgs in nc^ 0, Cryptology ePrint Archive (2021).

[15] A. Jain, H. Lin, A. Sahai, Indistinguishability obfuscation from well-founded assumptions, in: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, 2021, pp. 60–73.

[16] L. Devadas, W. Quach, V. Vaikuntanathan, H. Wee, D. Wichs, Succinct lwe sampling, random polynomials, and obfuscation, in: Theory of Cryptography Conference, Springer, 2021, pp. 256–287.

[17] R. Gay, R. Pass, Indistinguishability obfuscation from circular security, in: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, 2021, pp. 736–749.

[18] H. Wee, D. Wichs, Candidate obfuscation via oblivious lwe sampling, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2021, pp. 127–156.

[19] J. Groth, On the size of pairing-based non-interactive arguments, in: Annual international conference on the theory and applications of cryptographic techniques, Springer, 2016, pp. 305–326.

[20] A. Gabizon, Z. J. Williamson, O. Ciobotaru, Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge, Cryptology ePrint Archive (2019).

[21] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, G. Maxwell, Bulletproofs: Short proofs for confidential transactions and more, in: 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, pp. 315–334.

[22] E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev, Scalable, transparent, and post-quantum secure computational integrity, Cryptology ePrint Archive (2018).

[23] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: Decentralized anonymous payments from bitcoin, in: 2014 IEEE symposium on security and privacy, IEEE, 2014, pp. 459–474.

[24] AZTEC, Aztecprotocol, `https://github.com/AztecProtocol/AZTEC/blob/master/AZTEC.pdf`, (Accessed on 03/10/2022).

[25] A. Pertsev, R. Semenov, R. Storm, Tornado cash privacy solution version 1.4 (2019).

[26] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, H. Wu, Zexe: Enabling decentralized private computation, in: 2020 IEEE Symposium on Security and Privacy (SP), IEEE, 2020, pp. 947–964.

[27] V. Kolesnikov, T. Schneider, A practical universal circuit construction and secure evaluation of private functions, in: International Conference on Financial Cryptography and Data Security, Springer, 2008, pp. 83–97.

[28] H. Lipmaa, P. Mohassel, S. Sadeghian, Valiant's universal circuit: Improvements, implementation, and applications, Cryptology ePrint Archive (2016).

[29] P. Mohassel, S. Sadeghian, How to hide circuits in mpc an efficient framework for private function evaluation, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2013, pp. 557–574.

[30] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, J. Del Cuvillo, Using innovative instructions to create trustworthy software solutions., HASP@ ISCA 11 (10.1145) (2013) 2487726–2488370.

[31] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Decentralized Business Review (2008) 21260.

[32] S. Felsen, Á. Kiss, T. Schneider, C. Weinert, Secure and private function evaluation with intel sgx, in: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, 2019, pp. 165–181.

[33] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, C. A. Gunter, Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 2421–2434.

[34] S. Goldwasser, Y. T. Kalai, G. N. Rothblum, One-time programs, in: Annual International Cryptology Conference, Springer, 2008, pp. 39–56.

[35] R. Goyal, V. Goyal, Overcoming cryptographic impossibility results using blockchains, in: Theory of Cryptography Conference, Springer, 2017, pp. 529–561.

[36] A. C.-C. Yao, How to generate and exchange secrets, in: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), IEEE, 1986, pp. 162–167.

[37] S. Garg, C. Gentry, A. Sahai, B. Waters, Witness encryption and its applications, in: Proceedings of the forty-fifth annual ACM symposium on Theory of computing, 2013, pp. 467–476.

[38] D. Beaver, S. Micali, P. Rogaway, The round complexity of secure protocols, in: Proceedings of the twenty-second annual ACM symposium on Theory of computing, 1990, pp. 503–513.

[39] M. Naor, B. Pinkas, R. Sumner, Privacy preserving auctions and mechanism design, in: Proceedings of the 1st ACM Conference on Electronic Commerce, 1999, pp. 129–139.

[40] V. Kolesnikov, T. Schneider, Improved garbled circuit: Free xor gates and applications, in: International Colloquium on Automata, Languages, and Programming, Springer, 2008, pp. 486–498.

[41] S. Zahur, M. Rosulek, D. Evans, Two halves make a whole, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2015, pp. 220–250.

[42] V. Kolesnikov, P. Mohassel, M. Rosulek, Flexor: Flexible garbling for xor gates that beats free-xor, in: Annual Cryptology Conference, Springer, 2014, pp. 440–457.

[43] M. Ball, B. Carmer, T. Malkin, M. Rosulek, N. Schimanski, Garbled neural networks are practical, Cryptology ePrint Archive (2019).

[44] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, N. Zeldovich, How to run turing machines on encrypted data, in: Annual Cryptology Conference, Springer, 2013, pp. 536–553.

[45] C. Gentry, A. Lewko, B. Waters, Witness encryption from instance independent assumptions, in: Annual Cryptology Conference, Springer, 2014, pp. 426–443.

[46] H. Abusalah, G. Fuchsbauer, K. Pietrzak, Offline witness encryption, in: International Conference on Applied Cryptography and Network Security, Springer, 2016, pp. 285–303.

[47] D. Pan, B. Liang, H. Li, P. Ni, Witness encryption with (weak) unique decryption and message indistinguishability: constructions and applications, in: Australasian Conference on Information Security and Privacy, Springer, 2019, pp. 609–619.

[48] P. Chvojka, T. Jager, S. A. Kakvi, Offline witness encryption with semi-adaptive security, in: International Conference on Applied Cryptography and Network Security, Springer, 2020, pp. 231–250.

[49] P. Chvojka, Time reveals the truth-more efficient constructions of timed cryptographic primitives, Ph.D. thesis, Universität Wuppertal, Fakultät für Elektrotechnik, Informationstechnik und ⋯ (2021).

[50] Y. Lindell, Fast cut-and-choose-based protocols for malicious and covert adversaries, Journal of Cryptology 29 (2) (2016) 456–490.

[51] V. Kolesnikov, J. B. Nielsen, M. Rosulek, N. Trieu, R. Trifiletti, Duplo: unifying cut-and-choose for garbled circuits, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 3–20.

[52] P. Miao, Cut-and-choose for garbled ram, in: Cryptographers' Track at the RSA Conference, Springer, 2020, pp. 610–637.