Demo: CLOAK: A Framework For Development of Confidential Blockchain Smart Contracts

Qian Ren*   Han Liu*   Yue Li*   Hong Lei*†

*Oxford-Hainan Blockchain Research Institute
Hainan, China
†School of Computer Science and Cyberspace Security, Hainan University
Hainan, China

Abstract—In recent years, as blockchain adoption has been expanding across a wide range of domains, *e.g.*, digital asset, supply chain finance, *etc.*, the confidentiality of smart contracts is now a fundamental demand for practical applications. However, while new privacy protection techniques keep coming out, how existing ones can best fit development settings is little studied. Suffering from limited architectural support in terms of programming interfaces, state-of-the-art solutions can hardly reach general developers.

In this paper, we proposed the CLOAK framework for developing confidential smart contracts. The key capability of CLOAK is allowing developers to implement and deploy practical solutions to *multi-party transaction* (MPT) problems, *i.e.*, transact with secret inputs and states owned by different parties by simply *specifying* it. To this end, CLOAK introduced a domain-specific annotation language for declaring privacy specifications and further automatically generating confidential smart contracts to be deployed with trusted execution environment (TEE) on blockchain. In our evaluation on both simple and real-world applications, developers managed to deploy business services on blockchain in a concise manner by only developing CLOAK smart contracts whose size is less than 30% of the deployed ones.

Index Terms—Blockchain, Smart contract privacy, Trusted execution environment

I. INTRODUCTION

With the rapid development of both permissionless and permissioned blockchains, privacy issues have now become one of the top concerns for smart contracts, *i.e.*, keep transaction input and contract states as secrets to non-relevant participants. In many of the practical applications, privacy is an essential property to achieve, *e.g.*, avoid malicious arbitrage on cryptocurrency, protect sensitive information in a cooperative business *etc.*. Unfortunately, despite the importance of smart contract privacy, most of the existing blockchains are designed *without privacy* by nature [10]. For example, miners of Ethereum verify transactions in a block by re-executing them with the exact input and states. Consequently, private data is shared in the entire network.

Confidential Smart Contract. To address the aforementioned problem, researchers have proposed a variety of solutions in recent years to the design of *confidential smart contract*. In general, these approaches fall into two categories based on cryptography techniques and trusted hardware, respectively. For the former class of approaches, techniques including ring signature, homomorphic encryption and zero-knowledge proof
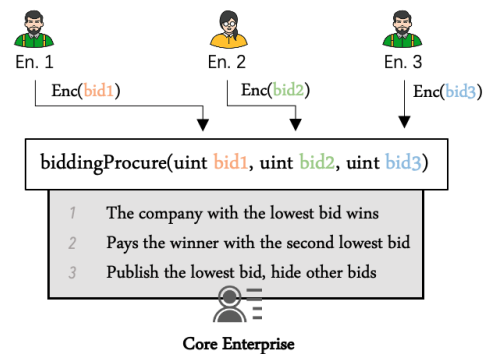

Fig. 1: A simplified multi-party transaction scenario

(ZKP) are adopted to achieve anonymity and privacy [1]–[3], [7]. For the latter, Trusted Execution Environment (TEE), *e.g.*, Intel SGX, is commonly used to provide confidentiality and trustworthiness [4], [8]. More specifically, TEE is able to reveal sealed transactions and execute them in enclaves to hide input and contract states with a verifiable endorsement from the hardware manufacture.

Limitations. However, while both classes of solutions provide architectural capabilities to enforce confidential lifecycles of transactions, they are non-sufficient for the *development* of practical applications. Figure 1 describes a scenario of procurement bidding among multiple enterprises in the setting of supply chain applications. Specifically, each participant submits its secret bid and a core enterprise decides a winner with the lowest bid. The core enterprise pays the second-lowest bid to the winner instead of the lowest one through updates on its balance. For cryptography-based solutions, developers are required to implement a set of off-chain multi-party computation programs and on-chain verification smart contracts, as indicated by [3]. On the other hand, TEE-based solutions allow developers to implement general smart contracts with secrets owned by only one side in a single transaction. Consequently, the implementation needs to process one source of confidential bid input at a time, cache intermediate bids and generate final states when bidding completes. To sum up, confidential smart contracts in the literature can hardly fit in the practical development of multi-party applications.
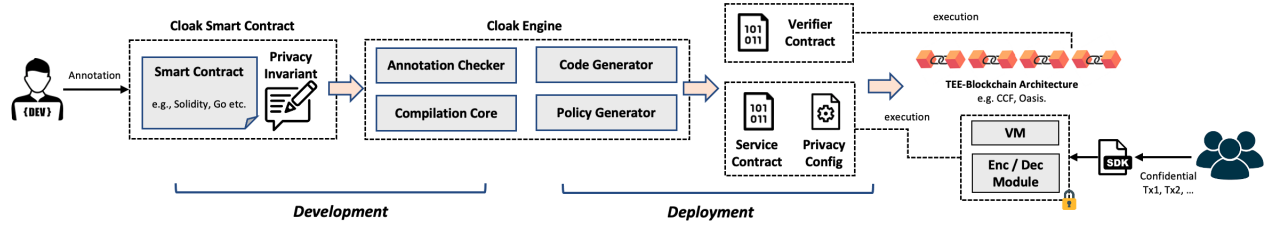
Fig. 2: The overall workflow of CLOAK

Multi-party Transaction. In this demo paper, we formalized the Multi-party Transaction (MPT) problem on blockchain for the first time and designed the CLOAK framework as a practical solution to it. CLOAK enables developers to develop and deploy confidential smart contracts in an MPT application by simply specifying it. Specifically, CLOAK allows developers to annotate privacy invariants as annotations in contract source code. It checks the privacy specification consistency and then generates deployable smart contract on blockchain. The main contributions of this work are as follows:

- We formalized the *Multi-party Transaction* (MPT) problem on blockchain for the first time, which transacts with secret inputs and states owned by different parties.
- We developed a framework, CLOAK, which allows developers to annotate privacy invariants, including MPT, as annotations in smart contracts, and generate privacy-compliant deployable code.
- We conducted a preliminary evaluation on real-world contracts with different privacy scenarios. The result shows the easy to use, high efficiency, and low cost of CLOAK.

Demonstration Plan. Our demonstration will showcase the capabilities of CLOAK in handling real-world data privacy issues. The detailed plan includes i) an automatic type checking, compilation, and deployment process, ii) in-depth explanation of the domain-specific annotation language, generated deployable code, and debug skills when developing with CLOAK iii) more comparative tests on representative contracts and privacy scenarios.

II. MULTI-PARTY TRANSACTION

We propose a new smart contract data privacy problem in blockchain called Multi-party Transaction (MPT): For $n(n \in \mathbf{Z}^* \wedge n > 1)$ parties, an MPT takes input x_i from each party i and $C(s)$, which is the cryptography commitment of contract old state s, *e.g.*, the hash or encryption result of s, *etc.*. Then, it runs the specified function f, publishes committed output r_i, a proof p and committed contract new state $C(s')$.

$$f(x_1, x_2, ..., x_n, C(s)) \Rightarrow C(r_1), C(r_2), ..., C(r_n), C(s'), p$$

An MPT should satisfy following two attributes:

- *Confidentiality*: Each party i knows r_i without knowing $\{x_j, r_j | i \neq j\}$ except what can be derived from x_i, r_i itself.

i should also know state s or s' only when it's owned and provided by him.
- *Verifiability*: With p, all nodes could verify that the commitment of new state $C(s')$ and return value $C(r_i)$ is the correct result of a function f, which takes unknown $\{x_j | j = 1..n \wedge j \neq i\}$ from $n(n \in \mathbf{Z}^* \wedge n > 1)$ parties and old state s, which is committed by on-chain $C(s)$.

MPT is different from Multi-party Computation (MPC). In MPC, even though all MPC participants acknowledge the transaction and record the result on a blockchain, it is hard for other nodes to verify it. Consequently, other nodes regard it as normal immutable data, making the MPC results lose widespread trust. In contrast, MPT achieves the same level of security and final consistency as smart contracts with proofs.

III. THE CLOAK FRAMEWORK

To handle the MPT problem, we designed a CLOAK framework. Figure 2 shows the workflow of CLOAK. It mainly divides into two phase, development and deployment. In the development phase, developers first annotate privacy invariants in Solidity smart contract intuitively to get CLOAK smart contract. `Annotation Checker` checks the annotation to make sure the privacy invariants are correct. The core of the development phase is `Cloak Engine`, in which the `Code Generator`, `Policy Generator`, and `Compilation Core` generate verifier contract, service contract, and privacy config. All generated code will be deployed to blockchains with TEE-Blockchain Architecture, *e.g.*, Oasis [4], CCF [9], *etc.*.

A. Develop Confidential Smart Contract

Annotate Privacy Invariants. Developers could annotate variable owner in the declaration statement to one of the {**all**, **me**, **id**, **tee**}. The **all** means public; **me** means the `msg.sender`; **id** is declared variable in type `address`; **tee** means any registered address of SGX with CLOAK runtime.

With CLOAK, users could intuitively specify the MPT in Figure 1 as a CLOAK smart contract, the *.cloak* file in Listing 1. In line 1, the developer could declare the key of `balances` as a temporary variable **k**, then specifies the corresponding value is owned by the account with address k, *e.g.*, `balances[tenderer]` is only known by the `tenderer` in line 23. In line 2, the developer specifies `mPrice` should be public. In line 6-7, to handle an uncertain number of suppliers, the developer declares owners **p** and specifies the

owners' owned data separately in two dynamic arrays. In line 10, the return value `sPrice` is owned by the `winner`. In line 12-13, the developer **reveal** private data to another owner, which is forced by CLOAK to avoid unconsciously leaking privacy. In line 14-24, it computes the lowest price, the second lowest price, and the winner. The computation is based on the operation between private data from different parties, *e.g.*, `bids[i] < sPrice`, `balances[tenderer] += sPrice`.

```solidity
1  contract SupplyChain {
2      mapping(address !k => uint @k) balances;
3      uint @all mPrice;
4
5      function biddingProcure(
6          address[!p] parties,
7          uint[@p] bids,
8          address tenderer
9      ) public
10     returns (address winner, uint @winner sPrice) {
11         winner = parties[0];
12         uint mPrice = reveal(bids[0], all);
13         sPrice = reveal(bids[0], winner);
14         for (uint i = 1; i < parties.length; i++) {
15             if (bids[i] < mPrice) {
16                 winner = parties[i];
17                 sPrice = mPrice;
18                 mPrice = bids[i];
19             } else if (bids[i] < sPrice) {
20                 sPrice = bids[i];
21             }
22         }
23         balances[tenderer] -= sPrice;
24         balances[winner] += sPrice;
25     }
26 }
```

Listing 1: CLOAK smart contract of bidding procurement

Annotation Checker. Taking a CLOAK smart contract, CLOAK ignores the annotation to checks the Solidity validation first. Then, CLOAK builds an Abstract Syntax Tree (AST) for further analysis. It infers data owner and checks the privacy invariants. It traversals the AST in post-order and updates each parent node's owner $o_p = o_l \cup o_r$. The o_l and o_r is the owner set of the left and right child node respectively. CLOAK recognizes a function as an MPT if $TEE \in o$ or $|o \setminus \{all\}| \geq 2$. The latter means the function will take private data from different parties. Then, CLOAK checks privacy invariants consistency. For example, CLOAK prohibits developers from implicitly assigning their private data to variables owned by others.

B. Deploy Confidential Smart Contract

Policy Generator. With checked AST, `Policy Generator` generates a privacy config P for the contract. P simplifies and characterizes the privacy invariants. Typically, P includes variables with data type and owners. It also includes ABI, a read-write set of each function. Specifically, P records each function's characteristics from four aspects, *inputs*, *read*, *mutate* and *return*. The *inputs* includes its parameters with specified data type and owner; *read* records state variables the function read in execution; *mutate* records the contract states it mutated; *return* records the return variables. Since P has recorded the details of state variables in the head, *e.g.*,

data type and owner, `Policy Generator` leaves the variable identities in *read*, *mutate* and *return*.

Code Generator. `Code Generator` generates a service contract F and a verifier contract V. While leaving the computation logic in F, `Code Generator` generates V to verify the result and update the state. In V, `Code Generator` first imports a pre-deployed CLOAK TEE registration contract, which holds a list of registered SGXs with CLOAK runtime. Then CLOAK transforms each MPT function in *.cloak* into a new function in V, which verifies the MPT proof p and assigns new state $C(s')$ later.

CLOAK Client. With configured nodes IP and ports by developers, CLOAK deploys the confidential smart contract **runtime** to TEE-Blockchain Architecture to get trusted CLOAK executors Es. The **runtime** includes `VM` and a `Enc/Dec Module`. Then, CLOAK deploys a SGX registration contract on the blockchain and registers the Es' certificate. For each CLOAK smart contract, CLOAK will deploys generated P and F to Es and V to the blockchain separately.

When a participant proposes an MPT, each participant i provides the x_i to SDK. x_i will be encrypted and sent to Es. According to deployed P, Es wait for all private inputs, synchronize the `read` state, construct a transaction, and execute it in enclaves. Then, Es encrypt return values and mutated states according to `return` and `mutate`. Finally, Es announce a result transaction on-chain with an MPT proof p. The p is Es' signature, *i.e.*, $p = Sig_E < P, F, C(s), C(r_i), C(s') >$. It means compliant to P, Es confidentially execute F with private inputs x_i and old state s committed by $C(s)$ in enclaves, commit return value r_i and new state s' to get result $C(r_i), C(s')$. Upon receiving the announcement transaction with proof, all nodes and V could believe an MPT real happened and get the result.

IV. PRELIMINARY EVALUATION

We conducted a preliminary evaluation of CLOAK on 9 contracts. The 9 contracts vary from scenarios, privacy invariants and have representative LOC in Ethereum smart contract [6].

TABLE I: The LOC of code before and after using CLOAK. the number of `Functions` (public, private, MPT) ; the LOC of CLOAK smart contract; the whole generated verifier contract V_{all}; generated service contract **F**, verifier contract **V** and privacy config **P** for MPTs

Name	#Functions	#CLOAK	#V_{all}	MPT related		
				#F	#V	#P
PowerGrid	4(1, 1, **2**)	25	146	23	126	72
Bidding	4(0, 2, **2**)	44	148	38	123	102
SupplyChain	6(0, 5, **1**)	68	249	36	145	85
Scores	6(0, 2, **4**)	77	239	57	211	174
Insurance	8(2, 3, **3**)	89	356	52	271	199
ERC20Token	11(4, 4, **3**)	112	347	56	218	173
YunDou	14(10, 0, **4**)	279	501	166	361	345
Oracle	22(19, 0, **3**)	326	413	93	190	196
HTLC	39(31, 0, **8**)	1029	852	429	401	443

Programming Simplicity. Table I shows the LOC of privacy-compliant code before and after using CLOAK. Specifically,

for MPT functions, while developers simply annotate in Solidity contract, CLOAK generates a total of 2.97-9.61X LOC, including 0.73-5.47X verifier contract LOC in Solidity and 1.03-3.13X privacy config in JSON. Therefore, CLOAK significantly reduces the development complexity of privacy in cryptography understanding and code implementation.

Performance. It costs CLOAK less than 1s to compile 8 contracts, while the biggest contract `HTLC` takes 5s. This is completely acceptable.

Gas Cost. In Figure 3, for a total of 27 MPT in 9 contracts, the transformed MPTs cost 0.8X to the original no privacy transactions on average. Specifically, the lowest one is 0.35X. The highest cost in the last 8 is 1.25X. Overall, running transactions on transformed contracts are feasible at a moderate cost.
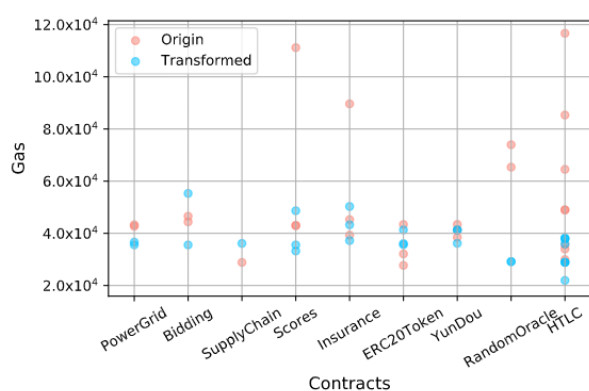

Fig. 3: Gas cost of MPT before and after CLOAK

V. DEMONSTRATION DESCRIPTION

Currently, for simplicity, we implement the CLOAK framework for a TEE-Blockchain Architecture consists of Ethereum and SGX simulators. It is a command-line tool and requires dependencies including ANTLR 3.82, Python 3.8, and solc [5] 0.5.17. A simple command to run CLOAK is:

```
$ cloak -i example.cloak -o out_dir
```

Specifically, `cloak` is a running script to start CLOAK. `-i` specify the input CLOAK smart contract, *.cloak*. `-o` specify the output directory to store generated code. In the demonstration, we use two use cases. *i.e.*, basic development and deployment and annotation debugging. The first helps developers understand the basic capabilities of CLOAK; the second shows fine-grained info of *.cloak* which is critical in debugging.

Use Case 1: Basic Development and Deployment. Developers use CLOAK on a supplied CLOAK smart contract, *i.e.*, a contract with annotated private data and data owner. First, we go through the source code of the specified contract with developers and introduce its metadata. Next, a participant will start the compilation of CLOAK with the basic configuration. We let the participant monitor the runtime logs generated by CLOAK in annotation checking. When it finishes policy and code generation, CLOAK will display an overview of the compilation process, *e.g.*, the private data and its owner, the function type recognized, the time used, *etc.*. Furthermore, we go into details of each function with developers and explain why CLOAK recognizes the function as a public, private, or MPT function. The developers can see the specific privacy statement resulting in the recognized function type and check whether the `Privacy Config` is expected.

Additionally, by using `SDK`, developers could deploy the generated verifier contract on a local Ganache-driven blockchain, the runtime, service contract, and privacy config in an SGX simulator. When it's done, they can send an MPT cooperating with two parties to know how it works and make sure it is privacy compliant.

Use Case 2: Annotation Debugging. This case focuses on debugging privacy invariant annotation when developing CLOAK smart contracts. We provide a group of CLOAK smart contracts with different errors. Developers can learn about debugging by flexibly using different commands to fine-grained control the CLOAK behavior. For example, developers could additionally use `-s/--solc` to ignore the CLOAK annotation and check the Solidity validation. `-t` specify the CLOAK just doing annotation checking and report privacy config. `--debug` specify CLOAK to show more compilation details, *i.e.*, the time used in annotation checking and code generation of each function, the hash of generated verifier contract, service contract, privacy config, and runtime.

REFERENCES

[1] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling Decentralized Private Computation. *2020 IEEE Symposium on Security and Privacy (SP)*, 00:947–964, 2020.

[2] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 423–443, Cham, 2020. Springer International Publishing.

[3] Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, Jonathan Katz, Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. zkay: Specifying and Enforcing Data Privacy in Smart Contracts. pages 1759–1776, 2019.

[4] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 00:185–200, 2019.

[5] Ethereum. Release version 0.5.17. https://github.com/ethereum/solidity/releases/tag/v0.5.17, Dec 2020.

[6] Péter Hegedűs. Towards Analyzing the Complexity Landscape of Solidity Based Ethereum Smart Contracts. *Technologies*, 7(1):6, 2019.

[7] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. *2016 IEEE Symposium on Security and Privacy (SP)*, pages 839–858, 2016.

[8] David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, Hung Q Ngo, Ying Yan, Changzheng Wei, Xuepeng Guo, Xuming Lu, Xiaofu Zheng, Qi Liu, Chenhui Zhou, Xuyang Song, Boran Zhao, Hui Zhang, and Guofei Jiang. Confidentiality Support over Financial Grade Consortium Blockchain. pages 2227–2240, 2020.

[9] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, and et al. Ccf: A framework for building confidential verifiable replicated services. Technical report, Microsoft Research and Microsoft Azure, 2019.

[10] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.