Confidentiality Support over Financial Grade Consortium Blockchain

Ying Yan, Changzheng Wei, Xuepeng Guo, Xuming Lu, Xiaofu Zheng, Qi Liu, Chenhui Zhou, Xuyang Song, Boran Zhao, Hui Zhang, Guofei Jiang
Blockchain Platform Division, Ant Financial Services Group

ABSTRACT

Confidentiality is an indispensable requirement in financial applications of blockchain technology, and supporting it along with high performance and friendly programmability is technically challenging. In this paper, we present a system design called CONFIDE to support on-chain confidentiality by leveraging Trust Execution Environment (TEE). CONFIDE's secure data transmission protocol and data encryption protocol, together with a highly efficient virtual machine run in TEE, guarantee the confidentiality in the life cycle of a transaction from end to end. CONFIDE proposes a secure data model along with an application-driven secure protocol to guarantee data confidentiality and integrity. Its smart contract language extension offers users the flexibility to define complex confidentiality models. CONFIDE is implemented as a plugin module to Antfin Blockchain's proprietary platform, and can be plugged into other blockchain platforms as well with its universal interface design. Nowadays, CONFIDE is supporting millions of commercial transactions daily on consortium blockchain running financial applications including supply chain finance [18], ABS, commodity provenance, and cold-chain logistics.

CCS CONCEPTS

• **Information systems** → **Distributed database transactions**; • **Security and privacy** → *Access control.*

KEYWORDS

Blockchain; Confidentiality; Trusted Execution Environment

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3386127

ACM Reference Format:
Ying Yan, Changzheng Wei, Xuepeng Guo, Xuming Lu, Xiaofu Zheng, Qi Liu, Chenhui Zhou, Xuyang Song, Boran Zhao, Hui Zhang, Guofei Jiang. 2020. Confidentiality Support over Financial Grade Consortium Blockchain. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3318464.3386127

1 INTRODUCTION

Blockchain is gaining technology adoption in financial services. Its transparency, traceability, immutability and verifiability features are valuable in multi-party business collaborations, such as Supply Chain Finance (SCF) [18]. In SCF, financial institutions make money lending conservatively towards small and medium suppliers because of challenges in risk control. On the other hand, majority of small and medium-sized firms' suppliers are in need of affordable financing. With blockchain system as shown in Figure 1, each pivotal step in the business process, such as creditworthiness, purchase orders and invoices, payment history, is timely consensused and cannot be tampered with. As the blockchain records collectively verifable history, it enables trading record assessment based on trustable data, hence reduces counterparty risks that banks concern about.

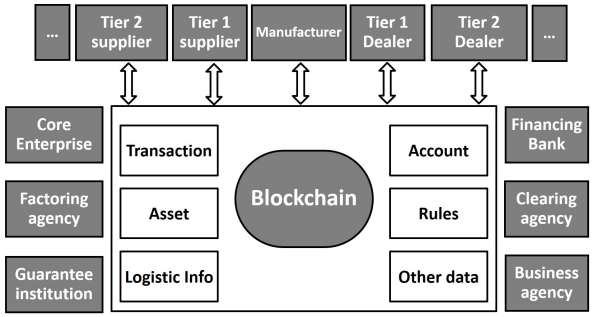

Figure 1: Supply Chain Finance on Blockchain

Transaction confidentiality is indispensable in blockchain financial applications on a shared ledger. In the example of SCF, transactions from one bank should be strictly kept confidential and inaccessible without permission to other banks.

A financial transaction often involves execution of multiple smart contracts, in the mean time requires low processing latency. A flexible and configurable confidentiality model is also necessary, for instance, a transaction can be configured as public or private, or part of a transaction output is confidential.

Many efforts have been taken to address the confidentiality problem on blockchains. However, solving the confidentiality problem in financial scenarios remains a challenge. Solutions introduced in [3, 14, 24, 32] are based on cryptographic primitive zero knowledge proof. However, these solutions suffer from low performance and limited generality. Trusted Execution Environment (TEE) such as Intel SGX has been introduced in some blockchain systems to provide data confidentiality[5, 7, 10]. Microsoft CCF [10] proposes a blockchain architecture to solve the performance problem by loading all the key components into TEE. However, consortium blockchain is usually designed with modularization to adapt for different scenarios, for example, storage module may be loosely coupled with computation and consensus modules to allow users choose their own KV stores. Loading all modules into TEE is often a hard choice in many blockchains. Ekiden [7] is another related work. It offloads smart contract computation from blockchain, and decouples consensus module with computation module. The whole contract states have to be loaded into TEE to guarantee the data integrity before transaction execution. This works well for simple and small smart contracts in public blockchains. However, in our scenario, financial service smart contracts are complicated and have large bytesize with the state data, for example, the total size of an SCF smart contract for one-month execution can be larger than the SGX physical memory limit.

In this paper, we introduce a system to support on-chain confidentiality in financial grade blockchain (CONFIDE) scenarios by leveraging TEE technology. CONFIDE has been implemented as a plugin module to Antfin Blockchain[1]. With its open interface design, CONFIDE design can be easily adopted by other blockchain platforms as well.

Our contributions are as follows:

- We identify the blockchain confidentiality requirements in financial blockchain applications and propose a solution - CONFIDE, which provides the full life cycle privacy protection of complex transactions with high performance and easy-to-configure confidentiality models.
- We introduce a confidential primitive to support fine-grained confidentiality specifications in smart contract programming.
- We implement CONFIDE with multiple performance optimizations. The production system is online and hosting multiple financial applications such as supply chain finance, ABS issuing, commodity provenance, and cold-chain logistics. Comprehensive evaluations are conducted with real workloads and demonstrated its effectiveness and efficiency.

The rest of the paper is organized as follows: In Section 2, we present the preliminary introduction to blockchain, trusted computing technology and the definition of blockchain confidentiality. Next, we propose our solution –CONFIDE together with its architecture description in Section 3. The confidential smart contract language extension is proposed in Section 4 which enables fine-grained confidential model. Section 5 introduces CONFIDE's implementation and optimization details followed by the evaluations with real workloads in Section 6. Other related works are discussed in Section 7. Finally, our work is concluded in Section 8.

2 PRELIMINARY

2.1 Transaction and Smart Contract

Traditionally, a contract exists as printed paper or a digital copy of scan. Transaction, a business action based on contract is often initiated and fulfilled with human involvement. With blockchain technology, the contract can be composed in a certain computer programming language and executed automatically on blockchain nodes without human involvement, e.g. depositing the money to an account once the predefined condition is met. Such computer-based contract is also known as smart contract. To some extent, the execution of smart contract involves handling the input information with programmed business logic, reaching consensus with nodes and then uplifting the chain to a new state, a transaction is fulfilled during this process. In practice, the smart contract is programmed in high level language and then compiled to native code or some sort of bytecode, resulting in docker-based execution or virtual-machine-based execution, respectively. The reason that avoids running smart contract natively is mainly for the sake of the safety of the whole blockchain platform. The pitfalls in smart contract should be properly handled to avoid breaking the belonged platform. We give out an informal definition about smart contract and its related terminologies in blockchain context.

- **Smart contract.** Smart contract is a piece of computer code which can be executed in a specific environment on blockchain system. A smart contract is composited of method and data like a "Class" in object-oriented programming language. The data of a contract is usually named as "state" in a smart contract's context.
- **Transaction.** Transactions are initiated from a client to blockchain node, which are mainly composed of

[1] Antfin Blockchain is a proprietary consortium blockchain designed by Ant Financial Blockchain Platform Division.

account information and transaction input information. Account information indicates the initiator address and a contract address on blockchain. Transaction information contains smart contract method along with values for its arguments. Note that different from public blockchain, all the on-chain business activities are encoded into smart contracts in a consortium blockchain. The transactions we considered are smart contract transactions which take actions on smart contracts. Other transactions without accessing a smart contract are not our focus.

2.2 Blockchain Virtual Machine

Based on the supported instruction set format, the virtual machines can be classified into two kinds: the register-based and the stack-based. The former is assuming the instruction operands are either in register or in memory and hence more hardware-related and can yield a better performance with a sophisticated fine-tuning toolchain, while the latter is assuming the instruction operands are in a virtual stack and hence hardware agnostic and very easy to implement(the corresponding toolchain is simple as well). As for executing smart contract in blockchain, it is much more important to maintain the result consistency across the nodes in the network. Thus the portable stack-based virtual machine is the first choice for the most cases. The Ethereum Virtual Machine (EVM)[16] is a typically stack based virtual machine released by Ethereum project, using Solidity[34] as its primary programming language. It is known for its simplicity and easy to use. One with few programming experience can quickly come up with a DApp[39] in Solidity. Furthermore, the EVM and Solidity are quite popular in academia because it is good enough to support realistic application while easy enough to be studied for eye-catching topics like security, formal verification and etc. In general, the EVM has better ecosystem for scenarios with less care on performance.

On the contrary, the WebAssembly (Wasm)[35] is targeting for a better performance. Its binary instruction format is carefully designed to be hardware-agnostic yet hardware-friend. This enables a Wasm virtual machine executes Wasm program in a decent speed via interpreting or in a near-native speed via JIT. More and more high level languages like C++, Rust and Go can be compiled into Wasm. Compared to Solidity language used in EVM, those high level languages allow users to program complicated business logic in smart contract in a much more convenient way. The strength of programming languages and the Wasm instruction format make the Wasm-based virtual machine an ideal solution for complicated performance-demanding applications.

2.3 Trusted Execution Environment

A Trusted Execution Environment (TEE)[31] is a secure area in the processor and it can guarantee the confidentiality and integrity of the code and data inside. Intel Software Guard Extensions (SGX)[9] is a widely used TEE implementation based on Intel CPU. Intel SGX provides application level code and data isolation, and guarantees data confidentiality and integrity in memory. The TEE of Intel SGX is an embedded "enclave" inside Intel CPU. In the enclave, the code and data are measured at the startup stage and the measurement is signed into an attestation report based on a hardware-based root of trust. The report can be verified to show the unmodified enclave code logical, by which users can confirm the security of enclave and provision the secret into the enclave at runtime, which is called remote attestation protocol.

2.4 Blockchain Confidentiality

Blockchain confidentiality refers to sensitive information that is broadcasted, verified and stored in a blockchain. Confidentiality on a blockchain is a mechanism to provide the control of permissions to the parties hosting the blockchain and users who can access the information on the blockchain. The information on the blockchain which needs confidentiality support includes:

- **Raw transaction**. The content of a raw transaction contains most of the information of a business action. The confidentiality of a transaction from it entering the network to its storage should be supported.
- **Smart contract**. The smart contract code sometimes is also sensitive. E.g. m parties have a smart contract deployed on a blockchain hosted by n parties ($m < n$) . The smart contract code should be kept private to m parties only. We should have the ability to support the confidentiality (configurable) of smart contract code, runtime information and results.
- **Storage**. The information on a blockchain's storage includes the raw transaction, smart contract states, indexes, block headers and so on.

Based on the discussion in the above sections, to solve the challenges of confidentiality in financial consortium blockchain scenarios, the design of our proposed solution follows the following four principles:

- **Minimizing TCB in the enclave**. The importance of minimizing the trusted computing base (TCB) comes from two folds. One is for small attack interface, and the other for the limited physical memory space of SGX which currently is commercial available. In order to minimize the TCB, we should minimize the components to be put into the TEE.
- **Loosely coupling with blockchain platform**. To be optimized for different applications, consortium

blockchains are usually designed with modularization. For example, there are different consensus algorithms for different blockchain sizes and network conditions. There are different blockchain VMs for different smart contract languages. Users can even choose their only KV storage when hosting a node. Thus, the confidentiality solution with TEE should also consider modularization and loosely coupling with the platform.

- **Maximizing the efficiency**. The introduction of confidentiality will bring additional overhead to the performance even with TEE. Optimization to IO, transaction execution and so on should be developed.
- **Guarantee the security**. The last one which is the most important one is to guarantee the integrity and confidentiality of all the information both inside and outside of TEE.

In the following section, we start introducing the details of our proposed solution - CONFIDE.

3 ARCHITECTURE

3.1 Overview

CONFIDE consists of a core Confidential Smart Contract Execution Engine (*Confidential-Engine*) and three key protocols as shown in Figure 2.

Confidential-Engine (Figure 2-①) protects transactions at run-time by leveraging TEE. *Confidential-Engine* is composed of a virtual machine, a secure data module and a key management module. All these modules are implemented into the enclave.

The three protocols are: (1) Secure Data Transmission Protocol (*T-Protocol*) for the security of confidential transaction (Figure 2-②), (2) Data Encryption Protocol (*D-Protocol*) for confidentiality and integrity of transaction execution results in persistent database (Figure 2-③), and (3) Key Management Protocol (*K-Protocol*) for secret key agreement among blockchain nodes. *T-Protocol* and *D-Protocol* guarantee a full life cycle protection to transactions with confidentiality and integrity.

Transactions (both public and confidential) sent from clients are broadcasted into the network. Each node processes the transactions independently. In the order consensus phase, public and confidential transactions are processed together. In the execution phase, public transactions are handled in Public Smart Contract Execution Engine (*Public-Engine*), while confidential transactions in *Confidential-Engine*. According to the confidentiality model defined in the smart contract, plain-text and cipher-text states are generated and stored into blockchain storage (KV database).

Finally, raw transactions, execution results are kept in the format (plain or cipher) according to the confidential model. Even though, the data in database can be accessed through

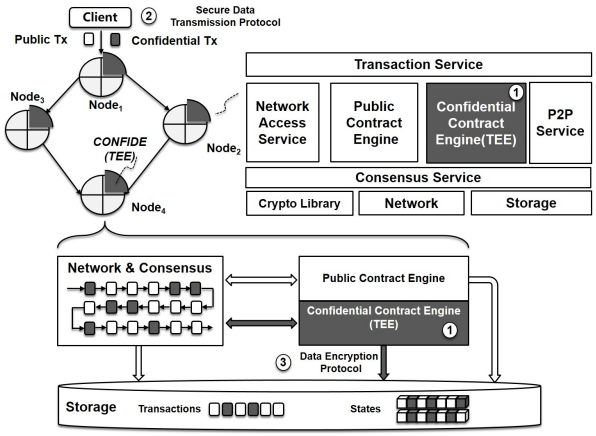

Figure 2: The architecture of CONFIDE

database API directly on each node. Thanks to the well-designed *T-Protocol* and *D-Protocol*, only the cipher-format data can be accessed if it is confidential. No one can access the content without permission.

3.2 Building Blocks

3.2.1 Confidential contract execution engine (Confidential-Engine). A *Confidential-Engine* contains four key components: a Transaction Pre-processor (Pre-processor), a Smart Contract Virtual Machine (VM), a Secure Data Module (SDM) and a Key Management Module (KMM).

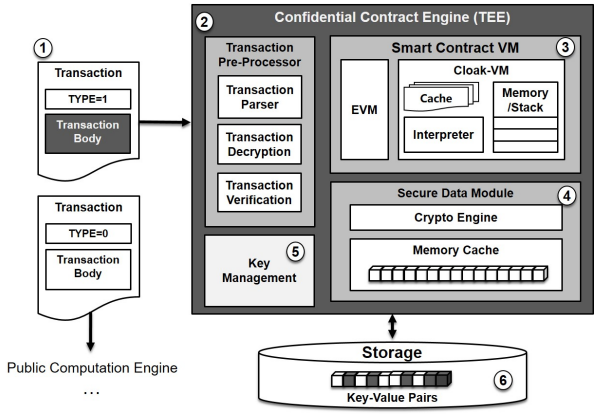

Figure 3: Confidential-Engine

As illustrated in Figure 3-①, a confidential transaction should be configured with TYPE=1. After it is forwarded into *Confidential-Engine*, it is parsed by the pre-processor first as shown in Figure 3-②. In the pre-processor, the transaction body is decrypted and verified. After the verification, the VM will be initiated. The VM then loads the corresponding

contract code via SDM from database and executes the contract as shown in Figure 3-③. Meanwhile, any interaction between *Confidential-Engine* and blockchain is handled by SDM.

SDM (Figure 3-④) contains a crypto engine for input/output data decryption/encryption according to *D-Protocol*, and a memory cache for I/O efficiency. The states of the contract as well as the information of the transaction are encrypted according to the confidentiality model and stored into the database outside. The security considerations about SDM is discussed in section 3.3.

VM. CONFIDE enables EVM for a traditional smart contract ecosystem using Solidity. For more complicated application requirements, EVM is not efficient enough. CONFIDE introduces a WASM-derived smart contract virtual machine CONFIDE-VM with multiple optimizations. CONFIDE-VM is composed of a bytecode interpreter, a code cache and a fixed size linear memory&stack which is used to execute Wasm bytecode. It inherits Wasm's advantages on multi-language support and efficient execution.

KMM manages secret keys in the enclave. In the initiation (when a new node joins the network) phase, KMM initials the key agreement among nodes.

3.2.2 Key management protocol (K-Protocol). According to the consensus protocol, confidential transactions are executed by *Confidential-Engine* on each node independently. That means, each *Confidential-Engine* is able to decrypt confidential transactions and generates the same encrypted contract state. *K-Protocol* is designed for secret key agreement among blockchain nodes. Two types of keys are participated in the *K-Protocol*.

- **Asymmetric private key (sk_{tx})** is used to decrypt the crypto digital envelope (confidential transaction) from clients. Its corresponding public key pk_{tx} is published to end users. Moreover the digital fingerprint of pk_{tx} is locked into the attestation report to immune the man-in-the-middle attack.
- **Symmetric states root key (k_{states})** is used for smart contract states encryption and decryption between *Confidential-Engine* and platform storage service.

These secret keys should be negotiated among the *Confidential-Engine* on all the nodes in advance. CONFIDE introduces *K-Protocol* for key agreement. *K-Protocol* supports two key agreement protocols:

- **Centralized key management service** is a low-cost and highly efficient service, which is acceptable especially for the ones equipped Hardware security module (HSM) with high security level in a consortium blockchain scenario.
- **Decentralized key management service** is based on a Mutual Authenticated Protocol (MAP) among *Confidential-Engine* on blockchain nodes. A random node in the network starts up and generates secret keys mentioned above. Any following node joining the blockchain network needs to be authenticated via the MAP. MAP in CONFIDE is implemented based on Intel SGX Remote Attestation [9] protocol.

3.2.3 Secure data transmission protocol (T-Protocol). To guarantee transactions' end-to-end security, the raw transaction should be encrypted when transmitted in the network. The secure data transmission protocol (*T-Protocol*) is introduced to serve for data encryption between clients (the user of the smart contract) and *Confidential-Engine*. Crypto keys used by *T-Protocol* are the public key pk_{tx} and a random generated transaction key k_{tx}. Transactions and transaction receipts are covered by *T-Protocol*. There are three important design principles for *T-Protocol*:

- **Security.** To maximize the entropy of cipher-text to countermeasure plain-text chosen attack and cipher-text chosen attack, we should use one-time encryption key for each individual transaction.
- **Efficiency.** An efficient key agreement protocol should minimize interaction round trips between clients and *Confidential-Engine*.
- **Performance.** Data encryption and decryption between clients and *Confidential-Engine* are usually frequent, so the cryptography overhead should not be neglected.

Based on above design principles, we propose a secure, non-interactive and high-performance *T-Protocol* for a full life cycle transaction data confidentiality including the raw transaction initiated from clients and the corresponding execution results which are referred as transaction receipts in blockchain context.

Confidential transaction generation. The confidential transaction Tx_{conf} generated under *T-Protocol* is shown in formula 1

$$Tx_{conf} = Enc(pk_{tx}, k_{tx}) | Enc(k_{tx}, Tx_{raw}) \quad (1)$$

where pk_{tx} is the public key of *Confidential-Engine* whose corresponding private key is provisioned into *Confidential-Engine* securely at the startup stage of CONFIDE, k_{tx} is one-time symmetric key of each transaction which is derived from a user root key and the transaction hash, Tx_{raw} is raw transaction.

Transaction receipts. The execution receipt Rpt_{raw} is encrypted using k_{tx} which is the one-time key of the confidential transaction as shown in formula 2. The Rpt_{raw} contains the addresses of transaction sender and receiver, transaction hash and so on.

$$Rpt_{conf} = Enc(k_{tx}, Rpt_{raw}) \quad (2)$$

The essential rule is that only the transaction owner has the permission to check the execution receipt. However, it is easy to implement a delegation or authorization protocol on the transaction. Since k_{tx} is a one-time key for each individual transaction, transaction owner can distribute k_{tx} to anyone who is authorized offline. Meanwhile, CONFIDE provides a more elegant way to realize the authorization not only for transaction receipt, but also including raw transaction information. CONFIDE built a pre-defined chain code to handle the pending request on the transaction receipts or raw transactions. The request will be parsed and forwarded to the related user smart contract, where user can define accessing rules for such requests.

3.2.4 Data encryption protocol (D-Protocol). *D-Protocol* is designed to protect contract states persistently stored as k-v pair in database. Crypto key used by *D-Protocol* is only k_{states}. *D-Protocol* covers confidentiality of smart contract states and smart contract code.

Contract states and code. Contract states are defined as public or confidential by smart contract through confidential smart contract language extension. Confidential states should only be visible to *Confidential-Engine* while the contract is executing. Any request to access the confidential states must be authorized via the smart contract logic with an intended design based on a privacy model. Consequently, the confidential states are encrypted by k_{states} and only decrypted and authenticated by the *Confidential-Engine*. The k_{states} is a secret key provisioned via *K-Protocol* introduced in section 3.2.2.

$$Data_{auth} = Enc(k_{states}, Data) \quad (3)$$

where $Data_{auth}$ is encrypted states or contract code by authenticated encryption with associated data. The additional authentication data is related to on-chain run-time information such as contract identity, contract owner and security version related to contract code. k_{states} is the root state key, $Data$ could be the raw state data or contract code.

3.3 Security Considerations

The integrity of code and data inside the enclave can be guaranteed, while those from outside can not. A malicious host may trigger enclave's computation with incorrect or stale data to break the application's confidentiality. TEE + blockchain is a typical example. A blockchain is composed of several distributed nodes. Smart contracts are replicated to each individual node's enclave for execution. An adversary can probe the information from the TEE boundary between trusted and untrusted world. For example, as a malicious owner of a node, s/he can reorder the transactions to observe execution results. Or s/he can also discard some transactions or even roll back the data in local database to replace the new data with the stale ones. The correctness and confidentiality are affected. Therefore, the applications in the enclave must have the ability to verify the integrity of data loaded from outside.

In CONFIDE, we guarantee the correctness of the whole blockchain and confidentiality of data through achieving the following security goals:

- **Correctness on chain.** Transactions are replicated to and verified independently on each node. The consensus protocol guarantees the state continuity. Only the transactions whose results are computed based on the latest states can pass the consensus phase and be written into blockchain successfully.
- **Confidentiality of the data on each node.** Confidentiality is guaranteed by a data protection mechanism and a fine-grained access control rule as introduced in section 2.4. Data protection is achieved by encryption and authentication protocol. Access control are declared by applications in smart contract. Data can only be accessed by the parties who are authorized. Access control rules (a white list for example) are defined as part of contract code. Updating the rules should be done through upgrading the contract. This guarantees the integrity of contract.

Note that, similar to other blockchain platforms, the correctness of a query from a single node is not guaranteed since a malicious host can hack the storage or the code of the platform (the code outside of TEE of CONFIDE). Therefore, to query blockchain data from other nodes, a consensus read (e.g. SPV) should be performed. For the malicious host, no matter how s/he attack the local node, the correctness of the whole ledger and data confidentiality can be guaranteed in CONFIDE.

4 CONFIDENTIAL SMART CONTRACT LANGUAGE EXTENSION (CCLe)

The smart contract deployed via a confidential transaction is confidential. It is fully encrypted, and executed after decryption inside TEE. This whole-contract-encrypted approach will bring maximum privacy protection for smart contracts. However in the financial sector, the third party audit is often required to conduct statistics on assets. If asset-related information stored in smart contract is in an encrypted status, third-party audit company must obtain the corresponding

keys to get the information. Sharing keys to third parties is clearly inappropriate and dangerous. Besides, the full encryption will bring much computation overhead and reduce the throughput. Therefore, users need to be able to specify which information in the smart contract is confidential and the remaining information can be exposed to other parties. But a programming language usually has no semantics of confidentiality, so a solution to support this behavior is needed.

Hence, the Confidential Smart Contract Language Extension (CCLe) is introduced. CCLe is designed based on the concept of IDL, which describes the data model in a language-independent way. So it can bring cross-platform and cross-language advantages. With CCLe, users can define the data model once and for all. Two attributes are provided: *confidential* and *map*. The *confidential* attribute is used to mark the confidentiality of data. It can support both primitive data types and composite data types. CCLe parser will track the confidential data of primitive types and automatically add the corresponding encryption and decryption process for it. The composite data types will be parsed recursively, and all the primitive data in it will be set *confidential* attribute. A *map* attribute is used to declare a composite data type which contains *key:value* pairs. It is often used in the financial sector, such as the *account:asset* model. It allows users to define their data model more easily.

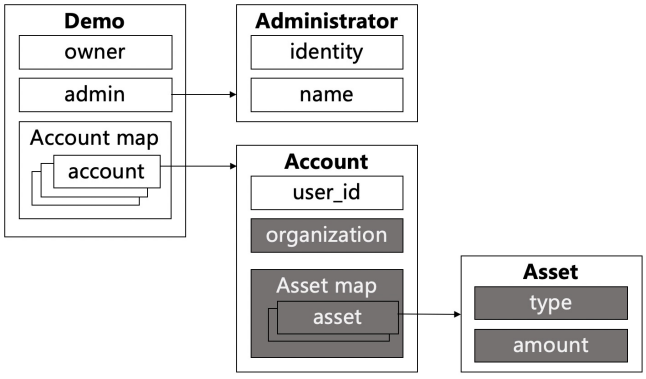

Figure 4: Example contract with confidential state

Figure 4 shows an example of a complex data model which has three hierarchies. The top level is "Demo", which has three members. The "owner" is a primitive data type, and "admin" is a composite data type. The two member are all public to other parties. The "Account map" is a *map* data type. The "Account" member in it is also a *map* data type and inserted in the runtime. The members in gray such as "organization" and "Asset map" are confidential. The code to describe the data model is showed in Listing 1. The *confidential* attribute is attached to the data definitions for both primitive data types and composite data types. The *map* attribute is also applied to related data, such as "asset_map" combined with *confidential* attribute.

Listing 1: Confidential Contract

```
attribute "map";
attribute "confidential";

table Demo {
  owner: string;
  admin: [Administrator];
  account_map: [Account](map);
}

table Administrator {
  identity: string;
  name: string;
}

table Account {
  user_id: string;
  organization: string(confidential);
  asset_map: [Asset](map, confidential);
}

table Asset {
  type: ubyte;
  amount: ulong;
}

root_type Demo;
```

CCLe implements an extension of Flatbuffers[20] IDL. Programmers can write the code in CCLe's schema file to define the basic data model. A codegen tool is provided to compile CCLe's schema file and translate it into low-level data structures. It is portable on different specific programming languages. The contract development workflow is shown in Figure 5.

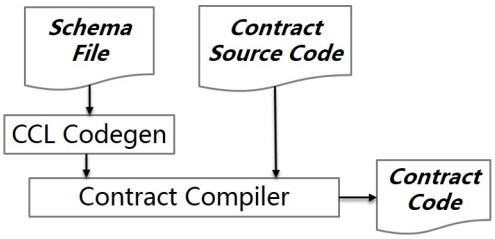

Figure 5: Development flow

CCLe is not limited to Flatbuffers IDL, and any other IDL can also be applied such as Protocol Buffers[21] and Apache

Thrift[38]. CCLe provides a fine-grained confidential model with high flexibility. At the same time, instead of encrypting the whole contract states, only sensitive ones are encrypted/decrypted with additional authentication metadata, which greatly saves computation cost. Besides *confidential* attribute, CCLe can be further extended to support more attributes easily, such as data access control.

5 IMPLEMENTATION AND OPTIMIZATION

CONFIDE is implemented as a plugin module to existing blockchain platforms, such as Antfin Blockchain. Smart contract can be programmed with Solidity, C/C++ and Go. CONFIDE-compatible blockchain explorer and smart contract IDE are available on Ant Finance official website [19] for developers.

5.1 Enclave Implementation

As illustrated in Figure 6, two enclaves are implemented separately to support contract execution and key management, respectively. This enclave decoupling implementation provides the flexibility for service upgrading in production environment. Following are implementation details.

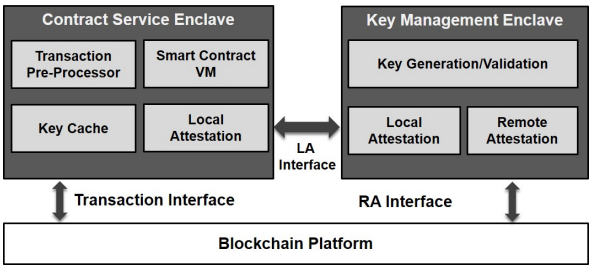

Figure 6: Enclaves in Confidential-Engine

Key management enclave (KM Enclave) implements Remote Attestation module to achieve a peer-to-peer mutual authentication. Key Generation module generates secret keys locally, while Key Validation module validates secret keys provisioned from other peers. Local Attestation module works as a challenger to other service enclave on the same platform.

Contract Service Enclave (CS Enclave), which supports contract execution, implements a pre-processor and a VM to process confidential transactions. Local attestation is used to establish secure channel with *KM Enclave*, reporting technical attestation to *KM Enclave*. Secret keys for confidential transactions processing are provisioned from *KM Enclave* via the secure channel, and then loaded into the Key Cache module.

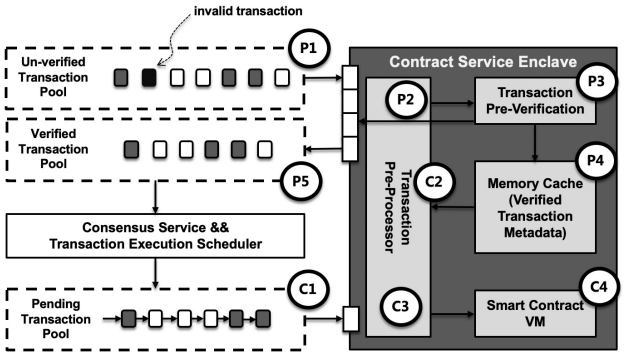

Figure 7: Transaction process

5.2 Transaction Pre-verification

Typically in a blockchain platform, a pre-verification phase is usually introduced to filter invalid transactions in advance which can be done in parallel. Discarding invalid transactions in advance can improve execution efficiency and throughput. As mentioned in Section 3, public and confidential transactions are mixed and processed together in transaction ordering phase. The public transactions can be verified easily, while for confidential transactions to be decrypted, their verification must be done in *Confidential-Engine*, in which two expensive operations exist: one is private key decryption and the other is public key signature verification. The two operations can be done in parallel among transactions, and the temporary information, e.g. signature verification, can be re-used in the later smart contract phase which greatly helps improve the platform's end-to-end throughput.

Pre-verification. As shown in Figure 7, there are two transaction pools in pre-verification phase. Transactions received from nodes, either valid or invalid, are first collected into the un-verified transaction pool, then after verification phase, valid transactions are put into verified transaction pool. The whole process for confidential transactions is as follows:

(1) P_1: Batch un-verified confidential transactions and push them into *CS Enclave*.
(2) P_2: Decrypt the transactions in pre-processor with sk_{tx} as in *T-Protocol*. Meanwhile, transaction key k_{tx} and raw transaction body Tx_{raw} are recovered from the crypto digital envelope.
(3) P_3: Signature of Tx_{raw} is verified, with the output result $f_{verified}$ of verification.
(4) P_4: Aggregate the transaction hash, k_{tx} and $f_{verified}$ as metadata and cache them into internal memory of *CS Enclave*.
(5) P_5: The verified transactions (in cipher-text) are put into the verified transaction pool.

Contract execution. After consensus, a batch of transactions are scheduled for execution as follows:

(1) C_1: Confidential transactions are forwarded into *CS Enclave*.
(2) C_2: Transaction pre-processor tries to fetch information from cache, which may be cached in pre-verification phase, according to incoming confidential transaction's hash. If the hash is missed in cache, the confidential transaction will be processed as normal, i.e. be decrypted and verified. Otherwise, the corresponding k_{tx} and $f_{verified}$ are retrieved from cache.
(3) C_3: With k_{tx}, transaction pre-processor can save the decryption cost. Tx_{raw} is recovered from confidential transaction through symmetric crypto decryption which is much more efficient than the private key decryption.
(4) C_4: If Tx_{raw} is parsed, the corresponding smart contract can be triggered for execution.

5.3 Optimizations for TEE

In this section, experiences on engineering with TEE (Intel SGX) will be shared.

TEE incurs performance overhead, including (1)*Software overhead* with enclave transition, in-enclave synchronization and enclave memory paging[36], and (2)*Hardware overhead* with memory security and integrity check by Memory Encryption Engine (MEE)[11]. In the implementation of CONFIDE, several optimizations are applied for Intel SGX to minimize these overhead.

Optimized data structure. Intel SGX provides Enclave Definition Language (EDL)[9] to describe the interface between the enclave and untrusted application. The Edger8r tool [9] generates proxy/bridge functions according to the user-defined EDL file for each ecall and ocall. In the proxy/bridge functions with [in] or [out] flags, which describe the data transfer direction, memory referenced by a pointer will be copied as there is no memory occupied across the boundary. It is negligible for a small memory check. However, for large memory buffer, the copy-and-check process will have a significant impact on the performance. Fortunately, EDL also provides a "user_check" flag, which can be attached to a pointer in the parameter list of the interface ecall/ocall. Setting this flag indicates that the copy-and-check process can be skipped and it becomes the programmer's responsibility to ensure the memory safety. So to fully benefit from "user_check" flag, the memory layout of key data structures needs to be optimized to facilitate memory boundary check and extra memory copy. Also, some key data structures, which are relative large or transferred across the enclave frequently, are flattened.

According to SGX developer reference[9], enclave interface functions are limited to C only, but the enclave functions can be written in C/C++. Besides, the libraries in enclave, for example STL, are specifically designed to be used inside enclaves. It may cause data compatibility issues when across the boundary. So a serialization and de-serialization protocol is necessary, such as the RLP code[17] in blockchain. But even with the light protocol like RLP, the cost of a complex class serialization is non-negligible sometimes. Observing that not all of the sub-field of class are necessary, or most of the sub-fields are read only, one-time ocall is split to multi-times ocalls. Rather than using the one-time ocall to get all the complex data structures to only obtain some of its sub-field, it is better to use multi-times ocalls, which only take some sub-fields needed at a time. The reason is based on the following calculation. Enclave transition of ocall typically introduces about 8,314 to 14,160 cycles cost, depending on cache hit or miss[37]. A typical platform supporting Intel SGX has a CPU frequency of 3.7GHz, which means it takes roughly 3-4μs for an ocall. So with proper design, balance between the cost of one-time ocall and multi-times ocall can be achieved.

Efficient memory management. Intel SGX v1 has only 128MB physical memory available (actually only 93.5MB is available for user enclaves)[1, 2, 30]. The physical memory is maintained by SGX driver and split into pages as Enclave Cache Pages (EPC). Memory allocation inside enclave will be redirected to SGX driver to allocate EPC memory. A page swapping mechanism is provided by Intel SGX to handle EPC allocation. When there are no enough EPC pages inside physical memory of Intel SGX, some selected EPC pages will encrypt their contents and store them into plain, normal memory pages hosted by the OS to free the space in the enclave. When the evicted page contents are needed, the SGX driver will load the contents back and decrypt them into enclave. This operation is transparent to users, but will cause significant performance degradation. To minimize page swapping overhead, it needs to efficiently manage the memory in enclave.

(1) *Configurable function module.* Configurable function module is used to minimize memory occupied by code and data. Different VMs are available for different scenarios. CONFIDE provides compile switch to disable EVM or CONFIDE-VM when one of them is not used to save memory usage. *KM Enclave* takes the responsibility for key management which is a quite low frequency operation, so it will be destroyed as soon as possible to release EPC memory.
(2) *Memory pool.* Memory pool is used to reduce fragmentation and improve memory utilization.

Improved enclave's monitor system. Financial-grade service has a strict requirement on high availability. So a monitor system is needed for developers to locate and solve problems in time when problems occur in the system. However, the status of confidential transactions executed inside enclave can not be detected directly by the OS due to the memory protection of Intel SGX and the lack of system I/O supported inside Intel SGX. In order to get status information inside enclave, ecall/ocall must be used. If ecall is used, it needs to actively query status from the untrusted application, but it can only get the inter-function status inside enclave, not the intra-function status. Therefore, ocall must be used as it can be inserted in the middle of the function execution. But the frequent status output can bring significant enclave transition overhead, which is unacceptable. Considering the status information is one-way stream out of the enclave, a simplified exit-less SGX call like [30] and a lock-free ring buffer outside enclave are implemented for monitor system. The exit-less call dumps the status information out and a polling thread pulls them on the ring buffer asynchronously into monitor system. The status information contains only error messages which are not related to any application data.

6 EVALUATIONS

To evaluate the effectiveness of CONFIDE, we report the performance of CONFIDE module as well as the end-to-end performance of a transaction. The blockchain platform we use in the entire evaluation is Ant Blockchain.

We choose 3 representative workloads in the evaluation:

(1) **Production workload from Ant Duo-Chain (SCF-AR)**, a supply chain finance service on "Account Receivable" (SCF-AR), which manages and circulates digitized Receivable certificates. The core business logic is powered by smart contracts that manages the entire life-cycle of assets, from account creation, asset insurance and transfer, financing, to clearing operations. It is a typical multilateral collaboration scenario involving different parties on the consortium blockchain. Preserving data confidentiality is extremely critical in such a business model. Due to its complexity of service model, transitional solutions such as ZKP, often become overly complicated to deal with. Figure 8 demonstrates a hierarchical smart contract design on SCF-AR service. An AR transaction starts at calling a Gateway contract and further to a Manager contract. After initial parameter parsing, the Manager contract dispatches the call to different service contracts, for instance ArAccount to create an account, ArIssue asset to issue asset etc. In the entire processing period, data is protected by CONFIDE to guarantee confidentiality.

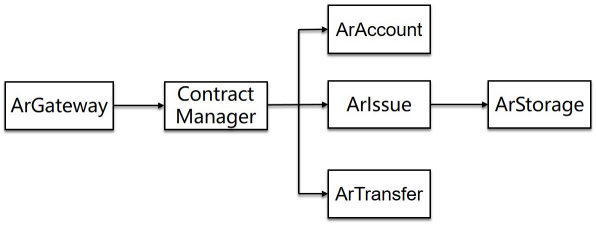

Figure 8: SCF-AR contract workflow

(2) **Production workload from Asset-Backed Securitization (ABS)** service platform, which leverages smart contact with confidentiality, while requiring high performance on asset transfer operations.

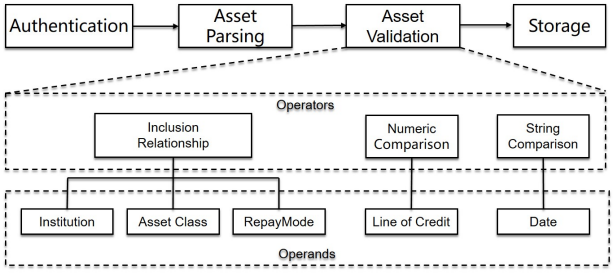

Figure 9: ABS asset transfer workflow

The "*Transfer Asset*" operation of ABS includes four steps, authentication, asset parsing, asset validation and asset storage as shown in Figure 9. Authentication validates the sender of a transaction. Asset information is encoded into a string with Flatbuffers protocol which contains about 10 attributes. The string is decoded in the parsing phase in order to retrieve asset information during the smart contract execution. Asset validation contains three operators, inclusion, numeric comparison and string comparison. Parameters such as institution, repay-mode, asset class and so on, will be sent into the operations in this phase. Finally all valid asset data will be stored. Typical size of the storage is about 1k bytes in our transaction.

(3) **Synthetic workload of smart contract with different complexity (Synthetic):** Besides workload SCF-AR and ABS, there are also other online service using CONFIDE such as warehouse receipt financing with IoT provenance. We summarize several building blocks of smart contract such as string concatenation, JSON parsing, and evaluate the performance on CONFIDE.

Our evaluation consists of two parts. First, to study the effectiveness of CONFIDE, we conduct evaluations on different workloads (Workload ABS and Synthetic) with different parameters and environment settings in section 6.2 and 6.1.

Then, to understand real online traffic pattern and the platform performance, we report performance metrics from our production environments in section 6.3 with Workload SCF-AR and 6.4 with Workload ABS.

Each performance result is reported as the average results of 50 runs. The experiments are evaluated on the Alibaba Cloud ECS Bare Metal Instance (X-Dragon) with Intel(R) Xeon(R) CPU-E3 1240 v6 @ 3.7GHz (4 Physical Cores and 8 Hyper Threading) and 32G bytes memory. The system is CentOS 7.2 and the SGX version is Linux 2.4 Release.

6.1 Performance of CONFIDE

In this group of evaluations, we analyze the performance of CONFIDE module, which includes the overhead of CONFIDE (the cost of confidentiality) comparing to public transactions, as well as the effectiveness of our optimizations. To conduct intensive profiling, we use the Synthetic workload. The experiments are conducted under four nodes in a local network, in which the nodes participate both consensus and computation. The block size is 4KB.

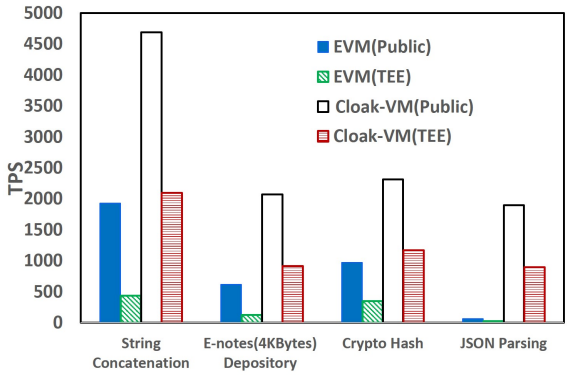

Figure 10: Performance on 4 Synthetic workloads

There are four representative Synthetic workloads with different complexity. (1)*String Concatenation* is a common operation that concatenates several strings into one. The parameters are JSON strings containing 35 key-values and an 10-bytes length ID string, and are joined together for later processing. (2)*E-notes Depository(4KBytes)* is the typical operation for electronic data service (e.g. invoice) and the size of a common payload is 4k bytes. The computation is conducted from receiving a 4k bytes string with an ID, and map the E-notes to this ID. (3)*Crypto Hash* is used in production smart contract. We choose SHA256 and Keccak hash algorithm. The SHA256 and Keccak are being performed 100 times in this experiment. (4)*JSON parsing* is another commonly used operation in smart contracts, e.g. for parameter parsing. In the ABS case, a request is submitted in the format of a JSON string. The platform will parse the JSON string to extract information in the request such as loan info, bank info, and so on. The JSON string is about 60 key-values in the experiment. Processing the whole JSON string is expensive, which introduces un-neglectable overhead, especially in EVM.

First, the efficiency of virtual machines is evaluated by comparing EVM with our optimized CONFIDE-VM for both public and confidential transactions. As mentioned in section 3.2.1 and section 5.3, CONFIDE-VM introduces a lot of optimizations in virtual machine design and implementation. All of these optimizations could be applied to any other public contract execution engine as mentioned in section 3. We port all of the optimizations to the Wasm-based Ant Blockchain's public contract engine. As illustrated in Figure 10, the throughput using EVM (both public and confidential) is much lower than the Wasm-based CONFIDE-VM under all workloads, which demonstrates the effectiveness of our optimizations.

Then, we examine the overhead of confidentiality, through comparing the performance of each VM with both public and confidential transactions. As shown in Figure 10, the performance with confidentiality (marked with TEE) is lower than public. However, the slowdown for CONFIDE-VM is obviously lower than EVM. Below are the reasons behind the results:

The overhead introduced by confidential execution is composed of: *workload independent overhead* from *T-Protocol*, and *workload dependent overhead* from *D-Protocol*:

(1) **Workload independent overhead**. This is also called *fixed-overhead*. Confidential transactions are wrapped with crypto digital envelope and its corresponding receipts are encrypted using *T-Protocol* as describe in section 3.2.3. This type of overhead is neglectable and hard to be further optimized for all VMs.

(2) **Workload dependent overhead**. The overhead are typically associated with I/Os during smart contract execution including:

- *States encryption/decryption.* According to *D-Protocol* described in section 3.2.4, confidential states will be decrypted (encrypted) when they are loaded (stored) from(to) outside enclave. CONFIDE employs authenticated encryption (AES-GCM[13]) algorithm for confidentiality and integrity. This cost comes along with each state load/store operation. Even the evaluation[8] shows that AES-GCM has quite high performance on current Intel CPU platform. It is still not negligible if there are lots of I/O operations in the smart contract.
- *Enclave transition.* Enclave transition comes when I/O happens along with ocalls. As evaluated by [37], an ocall typically introduces an overhead of 14,000 cycles if there is a cache miss.

6.2 Scalability

In this group of evaluations, we examine the scalability of CONFIDE over Ant Blockchain for confidential transactions only. The workload we use is ABS workload. The number of nodes varies from 4 to 20. Since Ant Blockchain supports smart contract paralleled execution, we also report the numbers with 4-way and 6-way execution in parallel. To further evaluate the effect of different network, we conduct evaluations with the nodes located in different cities (zones).

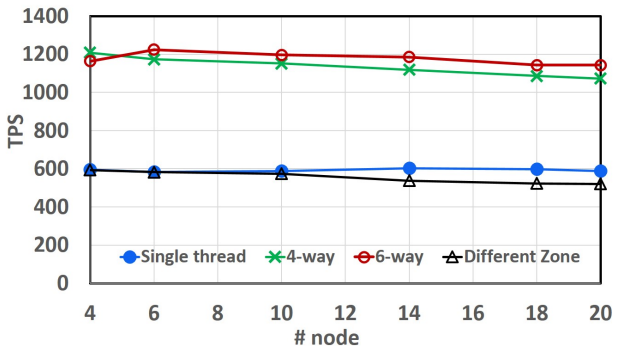

Figure 11: Scalability with ABS workload

Figure 11 shows the evaluation results. First, as the number of nodes increases, the performance under different settings remain stable, which demonstrates the good scalability of our platform with confidentiality. Then, we compare the performance of different threads. With paralleled execution, it shows a 2x improvement when the number of thread is 4. It results no further improvement when the number of thread increases to 6. This is because parallel execution performance is closely related to the pattern of workload. Typically, not all the transactions can be executed in parallel. In this evaluation, a 4-way parallel execution is already good enough for this workload.

Then, we examine the effect of network setting. We separate the nodes into two zones (Shanghai and Beijing) which are connected through public network with relatively less network bandwidth. The ratio of node numbers between the two city groups is 1:2. As shown in Figure 11, the performance decreases when the number of nodes increase. This is due to the relative high latency during consensus.

6.3 Performance of Production SCF-AR

In this group of evaluation, we report the performance of real SCF-AR workload in our experiment. There are 4 nodes in the blockchain network. Each node is Alibaba Cloud ECS Bare Metal Instance (X-Dragon) with Intel(R) Xeon(R) CPU-E3 1240 v6 @ 3.7GHz (4 Physical Cores and 8 Hyper Threading) and 32G bytes memory. The OS is CentOS 7.2 and the SGX version is Linux 2.4 Release. The nodes' p2p network for consensus protocol resides in the same virtual private cloud (VPC), while the transactions are received via public network from user's client independently on each node.

Table 1: Operations of SCF-AR contract

Method	Duration (ms)	Counts	Ratio
Contract Call	32.46	31	86.1%
GetStorage	4.80	151	12.7%
SetStorage	0.55	9	1.5%
Transaction Verify	0.22	1	0.6%
Transaction Decryption	0.10	1	0.3%

As listed in Table 1, there are 31 contract calls during a typical asset transfer flow, directly or indirectly. GetStorage/SetStorage involves contract I/O which will trigger ocalls of enclave. The transaction verification and decryption cost is also negligible.

6.4 Performance of Production ABS

In this group of evaluations, we first demonstrate the effect of CONFIDE's optimizations over ABS workload. Then, report the the performance of ABS from real production environment.

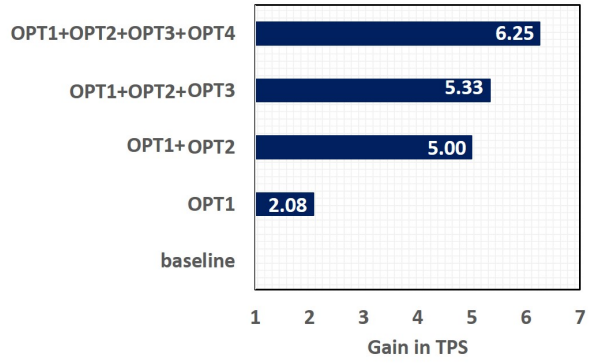

Figure 12: Optimizations on ABS contract

Figure 12 shows the effectiveness of our optimizations:

(1) **OPT1** applies the code cache and memory management optimization. WASM-based contract code has been encoded by LEB128. CONFIDE-VM introduces a code cache mechanism. As mentioned in section 5.3, efficient memory management increases the performance. In our evaluation, 2x gain can be obtained.

(2) **OPT2** is service contract optimization. ABS service contract takes a data structure with amounts of subfields which is encoded in JSON string. In contract,

parsing JSON based on interpreter execution will introduce huge amount of byte code instruction (about 450K instructions). We introduced Flatbuffers protocol instead to improve the performance by another 2.5x.

(3) **OPT3** introduces pre-verification optimization as mentioned in section 5.2 . In this workload, the improvement is another 6%.

(4) **OPT4** is instruction optimization. WASM-based byte code has a large instruction set. We optimize the instruction set for smart contract, reducing about 50% instructions which helps to shrink the jumping table significantly. The jumping table is generated by "switch-case" which is a typical implementation on bytecode interpreter execution. Moreover, by analyzing the byte code generated by service contract, we find a quite hot instruction patterns. By aggregating the instructions into one block, we gain about 17% performance improvement.

In real production system, transactions are submitted in batch by the application into the blockchain network. The time duration of blocks execution is about 30 ms on average. Periodically, empty blocks are generated continuously with about 5ms duration. Cloud SSD disks are mounted as storage system of the blockchain, the typical block write latency is about 6 ms on average.

7 RELATED WORK

Confidential smart contract executions. Ekiden[7] is a smart contract execution platform with confidentiality, integrity and availability by leveraging TEE hardware and blockchain. It decouples smart contract execution from consensus to provide an off-chain computation engine so as to reach a higher performance than Ethereum. Ekiden maintains a single sequence of state transitions to provide data integrity. Similar with Ekiden, Hyperledger Private Data Objects (PDO)[4] from Intel is another smart contract off-chain computation engine. Smart contracts are executed in TEE to guarantee data confidentiality. Different from Ekiden, PDO is stateless. Microsoft Confidential Consortium Framework (CCF)[10] is another work to provide smart contract confidentiality by leveraging TEE. CCF implements a secure memory database and provides secure RPC service inside TEE. With this architecture, CCF uses a TEE-based RAFT consensus protocol and achieves high throughput as a confidential decentralized ledger. The work[6] addresses problems and pitfalls for blockchain and trusted computing, and proposes a solution on Hyperledger Fabric platform.

Zero knowledge proof is another technology to build blockchain confidentiality. Hawk[24] provides a smart contract system with confidentiality. Smart contracts are executed off-chain and zero knowledge proof of the result is posted on chain. The execution of Hawk contracts are facilitated by a special party called the manager, who is trusted not revealing any secrets. First, all users send the private inputs to the contract. Then, the manager executes the smart contract and outputs the result and associated zero knowledge proofs. Although the manager cannot affect the correctness of result, it may disclose the secrets of users. Zexe[3] executes offline computation and subsequently produces publicly-verifiable transactions that attest to the correctness on-chain system, using zkSNARKs. Solutions based on zero knowledge proofs have limitations on performance or generalization for smart contract.

Blockchain payments and transactions. Bite[27] uses TEEs as full nodes to process requests from lightweight clients, and further implements oblivious algorithms to prevent side-channel attacks. Teechan[26] and Teechain[25] both use TEE to provide off-chain payment channels over Bitcoin. Zcash[22] leverages zk-SNARKs to provide enhanced confidentiality protection and anonymity during transactions. Monero[29] uses ring signatures, ring confidential transactions, and stealth addresses to obfuscate the origins, amounts, and destinations of all transactions. Mimblewimble [23] uses Petersen commitment , bulletproof and specific cryptographical protocol to realize confidentiality and scalability of transactions. All these systems are designed for supporting for payment transactions, not for smart contracts. Sharding is a way to increase blockchain scalability. [12] uses sharding to scale out blockchain platform. In their approach, TEE is served as a trusted random beacon to ensure the flexibility and security of the protocol. The generated random values are then used to partition nodes into shards. In proof of luck protocol[28], TEE is also used to introduce randomness. The protocol chooses the luckiest block based on the generated random number.The luckiest block will be the next block of this blockchain.

TEE-based secure systems. Some works on TEE engineering are also helpful to our optimizations. Opaque[40], ObliDB[15], VC3[33] provide solutions to sensitive data processing and data analytics.

8 CONCLUSION

Confidentiality with efficiency and flexibility plays a crucial role in financial blockchain applications. It is a challenge task to support these requirements all together. In this paper we propose CONFIDE to achieve on-chain confidentiality with high efficiency and flexibility. The data encryption protocol and TEE-based contract engine guarantee a full life cycle data confidentiality. The highly optimized virtual machine with efficient TEE implementations allows for the efficient execution of smart contracts hence obtaining high throughput. The confidential smart contract language extension enables

users to define complex confidentiality models in a simple and flexible way. CONFIDE is already used in production systems supporting several financial applications such as supply chain finance, ABS, IoT provenance, and cold-chain logistics.

REFERENCES

[1] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *OSDI'16*.

[2] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. SPEICHER: Securing LSM-based Key-Value Stores using Shielded Execution. In *FAST'19*.

[3] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2018. Zexe: Enabling Decentralized Private Computation. *IACR Cryptology ePrint Archive* (2018).

[4] Mic Bowman, Andrea Miele, Michael Steiner, and Bruno Vavala. 2018. Private data objects: an overview. *arXiv preprint arXiv:1807.05686* (2018).

[5] Marcus Brandenburger, Christian Cachin, Rüdiger Kapitza, and Alessandro Sorniotti. 2018. Blockchain and Trusted Computing: Problems, Pitfalls, and a Solution for Hyperledger Fabric. *CoRR* (2018).

[6] Marcus Brandenburger, Christian Cachin, Rüdiger Kapitza, and Alessandro Sorniotti. 2018. Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric. *arXiv preprint arXiv:1805.08541* (2018).

[7] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. 2019. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*.

[8] Intel Corp. 2015. *AES-GCM Encryption Performance on Intel® Xeon® E5 v3 Processors.* https://software.intel.com/en-us/articles/aes-gcm-encryption-performance-on-intel-xeon-e5-v3-processors

[9] Intel Corp. 2020. *Intel software guard extensions (intel sgx).* https://software.intel.com/en-us/sgx

[10] Microsoft Corp. 2018. *Confidential Consortium Framework.* https://microsoft.github.io/CCF/

[11] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* (2016).

[12] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards Scaling Blockchain Systems via Sharding. In *Proceedings of the 2019 International Conference on Management of Data*.

[13] Morris J Dworkin. 2007. Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. (2007).

[14] Jacob Eberhardt and Stefan Tai. 2018. ZoKrates-Scalable Privacy-Preserving Off-Chain Computations. 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)*.

[15] Saba Eskandarian and Matei Zaharia. 2017. ObliDB: Oblivious Query Processing using Hardware Enclaves. *arXiv preprint arXiv:1710.00458* (2017).

[16] Ethereum. 2019. *EVM Yellow Paper.* https://github.com/ethereum/yellowpaper

[17] Ethereum. 2020. *RLP wiki.* https://github.com/ethereum/wiki/wiki/RLP

[18] Ant Financial. 2019. *AntDuoChain.* https://www.ledgerinsights.com/alipay-ant-double-chain-blockchain-supply-chain

[19] Ant Financial. 2019. *Smart Contract Cloud IDE of Ant Financial.* https://baas.cloud.alipay.com/

[20] Google. 2019. *Flatbuffers.* https://google.github.io/flatbuffers/

[21] Google. 2019. *Protocol Buffers.* https://developers.google.com/protocol-buffers

[22] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. Zcash protocol specification. *Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep.* (2016).

[23] Tom Elvis Jedusor. 2016. Mimblewimble.

[24] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*.

[25] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter R. Pietzuch, and Emin Gün Sirer. 2019. Teechain: A Secure Payment Network with Asynchronous Blockchain Access. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*.

[26] Joshua Lind, Ittay Eyal, Peter R. Pietzuch, and Emin Gün Sirer. 2016. Teechan: Payment Channels Using Trusted Execution Environments. *CoRR* (2016).

[27] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. 2019. BITE: Bitcoin Lightweight Client Privacy using Trusted Execution. In *28th USENIX Security Symposium (USENIX Security 19)*.

[28] Mitar Milutinovic, Warren He, Howard Wu, and Maxinder Kanwal. 2017. Proof of Luck: an Efficient Blockchain Consensus Protocol. *CoRR* (2017).

[29] Shen Noether. 2015. Ring SIgnature Confidential Transactions for Monero. *IACR Cryptology ePrint Archive* (2015).

[30] Meni Orenbach, Pavel Lifshits, Marina Minkin, and Mark Silberstein. 2017. Eleos: ExitLess OS services for SGX enclaves. In *Proceedings of the Twelfth European Conference on Computer Systems*.

[31] M. Sabt, M. Achemlal, and A. Bouabdallah. 2015. Trusted Execution Environment: What It is, and What It is Not. In *2015 IEEE Trustcom/BigDataSE/ISPA*.

[32] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*.

[33] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *2015 IEEE Symposium on Security and Privacy*.

[34] Solidity. 2019. *Solidity manual.* https://solidity.readthedocs.io/

[35] WebAssembly. 2019. *WebAssembly Official Website.* https://webassembly.org/

[36] Nico Weichbrodt, Pierre-Louis Aublin, and Rüdiger Kapitza. 2018. sgx-perf: A Performance Analysis Tool for Intel SGX Enclaves. In *Proceedings of the 19th International Middleware Conference*.

[37] Ofir Weisse, Valeria Bertacco, and Todd Austin. 2017. Regaining lost cycles with HotCalls: A fast interface for SGX secure enclaves. *ACM SIGARCH Computer Architecture News* (2017).

[38] WikiPedia. 2019. *Apache Thrift.* https://en.wikipedia.org/wiki/Apache_Thrift

[39] Wikipedia. 2019. *Decentralized Application.* https://en.wikipedia.org/wiki/Decentralized_application

[40] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.