2021 IEEE International Conference on Blockchain (Blockchain)

SECAUCTEE: Securing Auction Smart Contracts using Trusted Execution Environments

Harsh Desai
The University of Texas at Dallas
Email: hbd140030@utdallas.edu

Murat Kantarcioglu
The University of Texas at Dallas
Email: muratk@utdallas.edu

2021 IEEE International Conference on Blockchain (Blockchain) | 978-1-6654-1760-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/BLOCKCHAIN53845.2021.00069

Abstract—**Smart contracts running on blockchains have emerged as an indispensable mechanism to enhance trust, security, transparency and traceability of data shared among critical distributed applications. Unfortunately, a smart contract deployed on a blockchain by itself is usually inadequate in maintaining data security and privacy because the data are replicated to all the nodes on the network. There has been some recent work that tries to tackle this privacy leakage issue in smart contract execution by integrating blockchains with hardware supported trusted execution environments(TEEs).**

Although TEEs ensure privacy to some extent, the smart contract execution can still be compromised if the developed code does not use the TEEs' capabilities correctly. One important security issue for leveraging TEEs in practice is the memory access pattern disclosure. Even though the TEEs encrypt all the memory content during the program execution, the memory access sequence can be observed by the malicious operating system, and can be used to infer sensitive information such as "who submitted the second highest bid to the auction?". Hence, for enhanced security for TEE based applications, the memory access pattern leakage need to be addressed. Given these observations, an apparent question that comes to light is, how can we use TEEs correctly to enable efficient, privacy enhancing and secure applications? In this work, we address this challenge in the context of digital auctions.

We develop a novel generic and secure framework that allow an auction smart contract to run inside secure enclaves over Intel SGX based TEEs on a blockchain. To our knowledge, this is the first work that provides access pattern leakage free TEE based secure auction smart contract deployment. We achieve this by implementing oblivious execution (i.e., no memory access pattern leakage) of both first price and second price sealed bid auctions as templates. Furthermore, we implement an end-to-end encryption service to keep the bids secure. Our empirical results and privacy analysis show that this architecture does not cause a significant impact to efficiency given the level of security achieved.

1. INTRODUCTION

Popularity of blockchains have grown significantly over the past few years due to the benefits and advantages offered by smart contracts [1]. Smart contracts provided on blockchain based platforms are used for many distributed applications due to their security, immutability and persistence properties.

One such important application of smart contracts is auctions. Auction is a form of distributed application where goods/properties are sold to the highest bidder via public sale. Auctions are considered to be powerful price discovery mechanisms to resolve resource allocation, service pricing and when done online, facilitate many e-commerce solutions. Blockchain based auctions such as [2], [3], [4] and [5] have been proposed in the past for implementing both sealed bid and open bid auctions. These auctions benefit from trustworthiness of agreements in the digital world.

In a distributed application like auction, there are many security and privacy challenges like having to trust the auctioneer with sensitive bidding information over the blockchain. Based on the common pattern of transactions being performed, sensitive data like identity of the stakeholders participating in the auction can be revealed.

Any security and privacy solution for implementing smart contract based auctions need to address the scalability challenges and may potentially need to scale for transactions that may involve large number of users. Although there are cryptographic solutions that leverage zero knowledge proofs (e.g., [6] and [7]), they are not scalable enough for many auction scenarios. Due to these efficiency and scalability concerns, in this work, we focused leveraging Trusted Execution Environment to achieve the desired security and privacy goal while implementing privacy-preserving auction smart contracts.

A Trusted Execution Environment, a.k.a TEE, is an area on the main processor of a TEE enabled device that is isolated from the system's main Operating System (OS). TEEs provide a more secure execution environment that uses both hardware and software to protect the application's data and code. The chief advantage of using a TEE is that it provides assurance that the code and data deployed inside is protected with respect to confidentiality and integrity. TEE has proven useful in protecting sensitive data and guarantee secure information retrieval in public cloud environments (e.g., [8]) as well as ensuring secure data analytic tasks are being performed optimally (e.g., [9]). Moreover, recent frameworks like Ekiden [10] and IBM developed Hyperledger Fabric-SGX [11] showed the feasibility of *running*

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on June 19,2022 at 19:42:21 UTC from IEEE Xplore. Restrictions apply.

smart contracts in secure enclaves inside TEEs.

Still using TEEs directly on top of Blockchains do not solve all the security and privacy issues. For example, the OS can still monitor memory access patterns of the applications running inside a TEE. This memory access pattern leakage has shown to disclose sensitive information even if the entire program, memory and the data are encrypted (e.g., [12]). For example, comparison operators in the smart contract can disclose the operands that are being compared based on memory location access. This is due to the fact that the "IF-Statement" compares two operands keeping the larger operand in the same memory location. A sophisticated attacker can figure out when that memory location content has changed and ultimately figure out bidder details. Hence, in the context of auctions, by observing the memory access patterns, and some known bids, a malicious OS may infer some of the bid amounts. Therefore, at the very least, TEE based solutions need to prevent this access pattern leakage. Unlike *the previous work that leverages TEEs for secure smart contract execution* [10], [11], we address *this security vulnerability for auction smart contracts*. More specifically, we address this challenge by building an auction framework where important auction specific functions are implemented without leaking any memory access patterns (See section 3 for more details).

In our framework, we implement both second price and first price sealed bid auctions [13] over private blockchains by enabling oblivious smart contract execution (e.g., no memory access pattern leakage) inside TEE based secure. To the best of our knowledge, *this is the first work that overcomes the security and privacy challenges presented by running an auction smart contract inside a TEE*. Our framework is general in the sense that, any type of auction can leverage our oblivious building blocks such as "oblivious max" operation to develop secure TEE based auctions thereby preserving privacy of the participants/bidders.

1.1. Overview of Our Contributions

The fundamental contributions of our research is as follows:

- We provide a generic framework that allows any auction smart contract run securely inside an Intel SGX (TEE) enclave while performing oblivious execution to declare the winner over a private blockchain.
- We create an end-to-end encryption service with programming abstraction for oblivious execution of auction logic for bid privacy protection.
- We implemented a prototype and ran comprehensive empirical evaluation under different deployment settings.
- We provide a detailed privacy and security analysis of our framework.

The remainder of our paper is organized as follows: In Section 2, we lay out the essential background to the reader. In section 3, we discuss the system architecture. In section 4, we discuss how the auction smart contracts are designed to prevent memory access leakage. In section 5, we dive into the implementation details of our system. Section 6 illustrates the performance of our system via a thorough experimental evaluation. We subsequently analyze the security and privacy offered by our system in Section 7. Section 8 compares our system with other blockchain based applications that take advantage of TEEs. Finally, we conclude the paper with section 9 while providing some scope for the future work we intend to accomplish.

2. BACKGROUND

2.1. Hyperledger Fabric

Blockchains are segregated broadly into three types, namely: 1) permissioned or private blockchains (e.g., Hyperledger Fabric [14]); 2) permissionless or public blockchains (e.g., Ethereum [15]); 3) consortium or hybrid blockchains (e.g., Quorum [16]).

Hyperledger Fabric [14] is a private / permissioned blockchain wherein participants are validated and known to each other. It offers pluggable consensus protocols to allow owners of the permissioned blockchain custom fit specific use-cases and trust models. Hyperledger Fabric, owing to its permissioned nature and consensus algorithm, does not employ any economic incentive in the form of cryptocurrencies to achieve transaction validation. Hyperledger is particularly used for employing smart contracts over the private chain. Smart contracts are programs running over a blockchain architecture that is intended to automatically execute, control or document events and actions in compliance with the terms and agreement of the contract. Smart contracts over the Hyperledger Fabric framework are termed as "chaincode".

2.2. Intel SGX

Intel Software Guard Extensions (SGX) is a Trusted Execution Environment which is a secure area of the main processor. It provides the isolation of execution of code to maintain the integrity and confidentiality of running applications. In Intel SGX, execution can be partitioned into enclaves which are areas of execution in memory with security protection. Furthermore, Intel SGX provides remote attestation of the enclave identity. A third party can verify any enclave's identity and securely provision keys, credentials and additional sensitive data into the enclave. This is accomplished by the service that Intel provides called the Intel Attestation Service. The infrastructure uses an Enhanced Privacy ID (EPID) for hardware based attestation. Additionally, Intel provides an Enclave Definition Language for untrusted application components of an Intel SGX enabled application to interface with the trusted enclave. The enclave uses both ECalls ("Enclave Call") and OCalls ("Out Call") to communicate within the enclave and outside the enclave to an untrusted application respectively. We use Intel SGX for enabling privacy and security for transaction processing, consensus, smart contracts and key storage for blockchains.

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on June 19,2022 at 19:42:21 UTC from IEEE Xplore. Restrictions apply.

2.3. Auctions

Auctions are popular examples of distributed applications implemented via smart contracts. In this work, we predominantly look into two main types of auctions based on the winner declaration strategy: 1) First price auction 2) Second price auction.

A first price auction is a type of auction where the highest bidder/winner pays the bid made by him/her on auction completion. A second price auction is a slightly different kind of auction where the highest bidder/winner pays the bid made by the second highest bidder of the auction. In many areas second price auction is favored compared to first price auction because second price auctions are designed to give bidders confidence to bid their best price without overpaying [17].

Auctions are also divided based on the bidding technique. The division renders the following two types of auctions: 1) Sealed bid auction; 2) Open Bid auction.

Sealed bid auctions are auctions where the bid is sealed amongst bidders when submitting to auctioneer. Here the auctioneer is a trusted party and only the auctioneer can open it and calculate the winner. Open bid auctions are the exact opposite where bids are known to everyone. For our case, to maintain privacy and security, we focus on the sealed bid auctions.

3. SYSTEM ARCHITECTURE

In this paper, we propose a pragmatic system named SECAUCTEE that allows any auction protocol to run securely using TEEs. There are primarily four components of our SECAUCTEE system, each playing a crucial role to achieve the overall goal. In the remainder of this section, we discuss these components.

3.1. On-Chain oblivious Execution

Trivial execution of the programs usually change the memory access pattern based on the input data. For example, "IF-Statements" are one of the main culprits in revealing access patterns. On the other hand, oblivious execution, in essence, hides access patterns of an algorithm. In other words, a data oblivious program is a program in which with any data access, the program executes the exact same code path. In other words, whose memory access pattern is independent of input values. [1]

Here, we are assuming that an attacker can always observe Enclave calls (ECalls) and Outside Calls (OCalls) of the Intel SGX processor but not the internal CPU register operations. In addition, the attacker can also record the time to execute and any resource access as a result of the above calls.

1. There are other ways to achieve data obliviousness, e.g., using ORAM constructions [18] but in this work, we focus on much more efficient implementation tailored for auction use case. Such tailored oblivious solutions have been developed for data processing (e.g., [9]) but not auctions

In the case of auctions, once the data is accepted by every bidder and the bidding phase comes to an end, the smart contract computes the winner. The algorithm used by the smart contract intrinsically is a comparison based algorithm and with many "IF-Statements" to determine the winner of the auction during declaration phase, and naive implementation leaks memory access. For example, in the case of a sealed-bid auction there are O(n) comparisons where n is the number of bids made by the bidders. If for example, Bidder 1 is the first bidder and has bid the highest, the first comparison will save Bidder 1 as the winner and all the other comparisons will result in no change in the memory location for the winner. As a result, the attacker now has the knowledge of the bidder identities and their bids which violates the privacy of the bidders in a sealed bid auction.

To address the above scenario, we implement assembly language programming based comparison which manipulates *CPU registers*. Assembly language is a low-level language that's purpose is to interface directly with a computer's hardware. In the case of auctions, we replace the comparison "IF-Statement" with an "asm" block that is an assembly code that assigns the values to be compared to separate registers and then based on the condition we want to test (here being a comparison), we set a flag that allows us to determine the maximum bid. We exit out of the assembly code with the binary value stored in the flag and then execute a simple arithmetic toggle that retains the value if the flag is unset (i.e. value = 0) indicating that the new value is not greater than the current maximum or changes the value to the new bid if the flag is set (i.e. value = 1) indicating that the new value is indeed greater than the current maximum.

We perform this for both first price and second price sealed bid auctions. Further details regarding execution of the contract is described in section 4.

3.2. Blockchain-TEE Integration

As shown in Figure 1, we deploy our private blockchain over Intel SGX machines. Our system is general in the sense that any private blockchain can be integrated with TEE. We have the orderer and the certificate authority as separate containers.

The peer primarily composes of 4 components: the chaincode enclave, the enclave registry, the enclave endorsement validation (not shown) and the chaincode package bundler (not shown). The chaincode enclave is described more in section 3.4. The enclave registry is a chaincode of itself that runs outside the SGX enclave to maintain a list of all prevailing chaincode enclaves in the network.

For the Blockchain to take complete advantage of the Intel SGX TEE environment, the chaincode is run in the Hardware Mode (secure mode) that accommodates for Intel SGX attestation support. The blockchain uses Intel Attestation Service to verify and attest the hardware. Enhanced Privacy ID (EPID) based attestation is performed using a Service Provider ID and the primary key provided by

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on June 19,2022 at 19:42:21 UTC from IEEE Xplore. Restrictions apply.

the Intel Attestation Service thereby, confirming that the Hardware is indeed an Intel approved TEE.

3.3. End-to-end encryption with Auction Flow

The chaincode enclave creates a public private key/pair of its own and makes the public key accessible to everyone by allowing it to be accessed by a GET method on the chaincode. Once the chaincode keys are created, the chaincode starts the bidding phase. Herein, each participating bidder encrypts the bids using their chaincode public key (as shown in Step 2 Bidding Phase) in Figure 1. They use the encryption service to create Bidder Public/Private key pair for signing purposes (Step 3) and subsequently, send the Bidder public key to the auction chaincode (Step 4). The bidder then signs the encrypted bid and send the signed bid to the contract (Step 5).

Once the chaincode ends the bidding phase, the declare winner phase begins which involves running the on-chain oblivious execution described in Section 3.1. When the winner is determined by the contract, the contract signs the result with it's private key and then notifies the bidders (Step 1 Declare Phase) so that the result can be accessible by a GET method (getSignedWinner). The bidder then uses the encryption service to verify whether the result is really coming from the chaincode by using the encryption service as shown in Step 2 of the Declare Phase. For this step, the encryption service uses the chaincode public key to verify the source.

3.4. Chaincode in an enclave

Each chaincode spawned from the peer running on Intel SGX runs in its own separate enclave so that it can isolate itself from the peer node and the other chaincodes. The enclave is initiated with a blockchain shim that allows it to be called by the peer. In order to interact with the chaincode, a bidder must send the call to the peer. The peer verifies the identity of the bidder by using a certificate authority and it then forwards the request to the enclave. The enclave's shim is used as an interface between the peer and the chaincode. The chaincode responds to the request with the desired result and the peer returns back the result in a Base64 format with the public key and signature. The chaincode execution does not leave its trace in any system logs or peer logs thereby keeping the execution secure.

4. SMART CONTRACTS

Code Block 1 portrays a simple comparison between bid1 and bid2 to return a true/false flag in the form of an integer. This Code Block 1, in our implementation is replaced by Algorithm 2 which calculates the true/false flag is replaced by the code in Algorithm 2.

```
int maximum(int bid1, int bid2) {
  if (bid1 > bid2) {
    return 0;
  } else {
    return 1;
  }
}
```

Code Block 1: Non oblivious Maximum

Algorithm 2 replaces the "IF-Statement" with an Assembly Language Block that directly manipulates the CPU registers to get the final answer (please note attacker cannot observe the CPU registers). To expound on the Assembly Block, clc on line 4 clears the carry flag. In other words, the EFlags register in the Intel x86 architecture which holds the CF flag is set to 0. Line 5 and Line 6 of Algorithm 2 moves first bid and second bid to the EAX and EBX registers respectively. ECX and EDX registers are initially set to 0 as done in line 7 and line 8. "movl" basically moves a 32 bit integer either between registers or from a value to a register. In algorithm 2, we only use movl to move 32 bit integers to registers. So for example, "movl x, y" will copy the value of x into y where x is the 32 bit integer and y is the register.

```
int maximum(int bid1, int bid2) {
  int returnvalue = 0;
  asm(
  "clc \n"
  "movl %1, %%eax \n"
  "movl %2, %%ebx \n"
  "movl $0, %%ecx \n"
  "movl $0, %%edx \n"
  "cmp %%ebx, %%eax \n"
  "adc %%ecx, %%edx \n"
  "movl %0, %%edx"
  : "=d"(returnvalue)
  : "g"(bid1), "g"(bid2)
  );
  return returnvalue;
}
```

Algorithm 2: Oblivious Maximum

"cmp" is the compare operation that compares values of 2 registers. "cmp %%ebx, %%eax" resembles "cmp destination, source". Consequently on line 10, an "adc" operation is executed which stands for Add With Carry. The result of the addition performed is stored in the EDX register. Finally, to read the result / contents of the EDX register a movl operation is performed which returns the result (either 0 or 1, depending on the comparison result) and stores it inside "returnvalue".

The "maximum" function described in Algorithm 2 is the central recipe to make our auctions oblivious. We use the function in both Second Price Auction and First Price Auction as shown in Algorithm 3 and Algorithm 4 respectively. The maximimum value is stored in the "max" variable and the ID of the bidder that bid the maximum value is stored in the "maxID" variable. On similar lines, "secMax" and "secMaxID" variables store the second maximum bid

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on June 19,2022 at 19:42:21 UTC from IEEE Xplore. Restrictions apply.

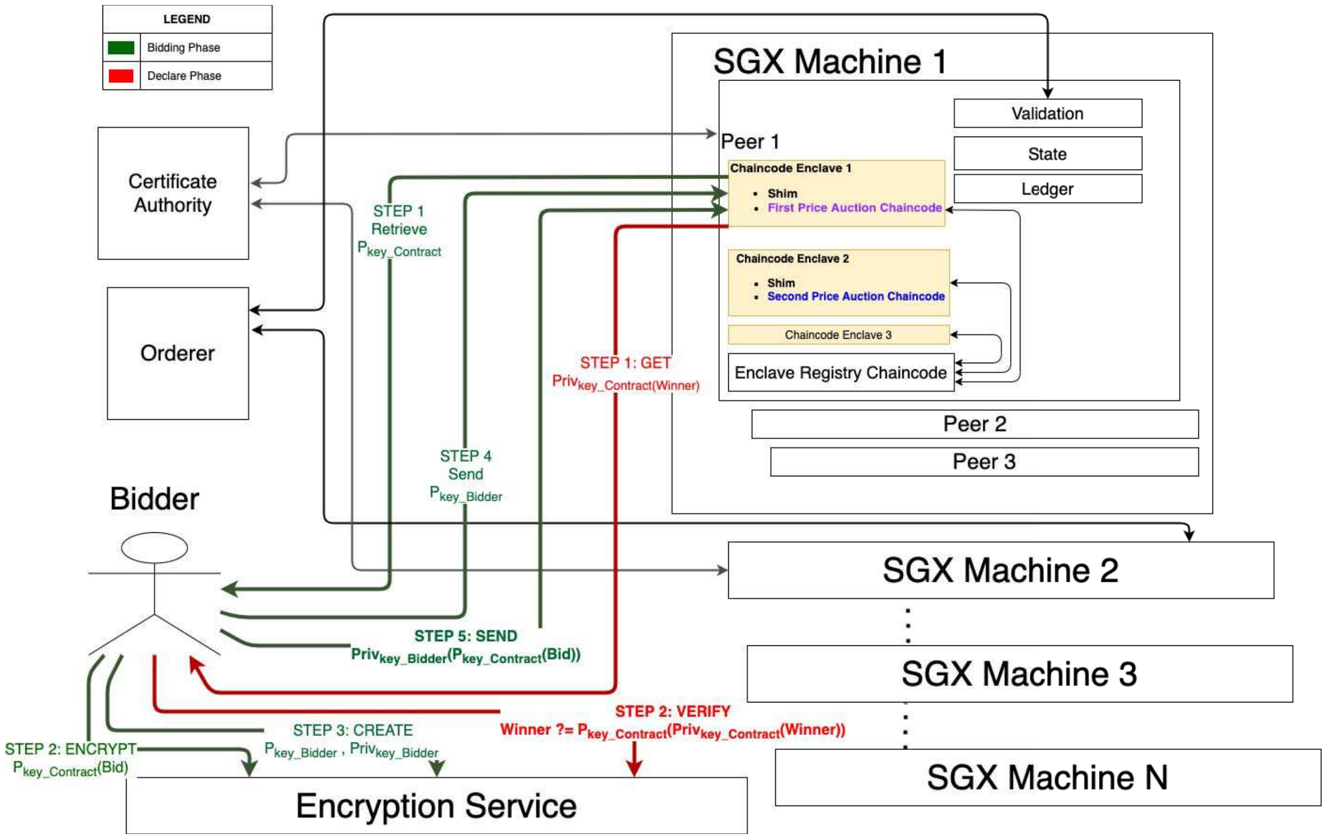

LEGEND
🟩 Bidding Phase
🟥 Declare Phase
SGX Machine 1
Peer 1
Validation
State
Ledger
Chaincode Enclave 1
Shim
First Price Auction Chaincode
Chaincode Enclave 2
Shim
Second Price Auction Chaincode
Chaincode Enclave 3
Enclave Registry Chaincode
Peer 2
Peer 3
Certificate Authority
Orderer
STEP 1 Retrieve P_key_Contract
STEP 1: GET Priv_key_Contract(Winner)
STEP 4 Send P_key_Bidder
Bidder
SGX Machine 2
SGX Machine 3
SGX Machine N
STEP 5: SEND Priv_key_Bidder(P_key_Contract(Bid))
STEP 2: VERIFY Winner ?= P_key_Contract(Priv_key_Contract(Winner))
STEP 3: CREATE P_key_Bidder, Priv_key_Bidder
STEP 2: ENCRYPT P_key_Contract(Bid)
Encryption Service

Figure 1: SecAucTee: Architecture Diagram

made and the bidder ID that made the second maximum bid respectively.

```
int max = 0;
int maxID= 0;
int secMaxID = 0;
int b = 0; // b is the selector
for (int i = 0; i < user_count; i++) {
  bid = getBid(i);
  bidderID = getBidderID(i);

  b = maximum(secondMax, bid);
  secMax = (1-b)*secMax + b*bid;
  secMaxID = (1-b)*secMaxID + b*bidderID;

  b = maximum(max, secMax);
  int tempMax = max;
  int tempMaxID = maxID;
  max = (1-b)*max + b*secMax;
  maxID = (1-b)*maxID + b*secMaxID;

  secMax = b*tempMax + (1-b)*secMax;
  secMaxID = b*tempMaxID + (1-b)secMaxID;
}
```

Algorithm 3: Oblivious Second Price Auction

```
int max = 0;
int maxID = 0;
int b = 0;
for (int i = 0; i < user_count; i++) {
  bid = getBid(i);
  bidderID = getBidderID(i);

  b = maximum(max, bid);
  max = (1-b)*max + b*bid;
  maxID = (1-b)*maxID + b*bidderID;
}
```

Algorithm 4: Oblivious First Price Auction

5. IMPLEMENTATION

5.1. Platform details

Considering hyperledger and Intel Software Guard Extensions (SGX) as the choice of Private blockchain and Trusted Execution Environment respectively running on Linux (Ubuntu). The chaincode is primarily written in C++. We used docker containers to deploy the orderer, certificate authority and peer nodes over the architecture of SGX machines. The containers were deployed as separate microservices over all the bare metal machines. Network connectivity

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on June 19,2022 at 19:42:21 UTC from IEEE Xplore. Restrictions apply.

was set up amongst all machines. We used Fabric Private Chaincode concept-release 2.0 to deploy the hyperledger fabric blockchain over Intel SGX. We used SGX Software Development Kit (SDK) v2.6 to interface the chaincode with the enclave. SGX SSL based off of OpenSSL 1.1.0j was used to create the certificates for both the bidders and the chaincode. We used openssl RSA algorithm (Asymmetric Encryption) with a minimum key length of 3072 bits to generate bidder public/private keys.

5.2. Private Chain Specifications

Sgxtop is used to validate that the chaincode is indeed running inside an SGX enclave. Since maximum payload size submitted by a bidder in a transaction is upper bounded by 3072 bits, the peer node must support this limitation. Therefore, the enclave registry chaincode provided by Fabric Private Chaincode is modified to accommodate for larger transactions in golang. A configuration is provided for the peer to run the chaincode in an enclave or outside the enclave. The chaincode running inside the enclave lives on one and only one machine. Since the enclave cannot be split into two, execution of the chaincode remains in one peer.

5.3. Transaction Commitment

Analogous to the hyperledger fabric that validates transactions using the validation system chaincode (VSCC), the enclave transaction validator explained in Section 3, performs the same validation. The orderer service on receipt of the transaction assigns it to a block and subsequently broadcasts the block to all the peers thereby creating consensus. The ordering service options to evaluate was Raft vs. Kafka and we use Raft simply because the success rate and throughput to commit transactions is much more higher than Kafka.

6. EXPERIMENTAL EVALUATION

We primarily evaluate the execution time while maintaining stability and security of the system. We examine effects of altering number of bidders, number of bids and number of bids per bidder taking part in the auction process.

6.1. Experiment Setup

We use hyperledger fabric provided docker containers to deploy our private blockchain over bare metal machines hosted locally.We use fabric-private-chaincode developed by IBM as a means to integrate the peer containers with the SGX architecture to deploy chaincode in separate enclaves. Bash scripts are used to control the deployment of chaincodes.

Since our framework is not deployed on the public network, neither does it use any public chain, we do not report any monetary costs. This would be an issue on public blockchain such as Ethereum where smart contrats need to pay for the gas.

Our experimentation setup involves first fixing the number of bids per user and then changing the number of users per round. We evaluate the time taken to complete the auction when the number of bids per user are 5 bids, 10 bids and 100 bids. We then fix the number of users to say 1000 and then change the number of bids per user. That way, we have a comprehensive insight into the performance of our system.

6.2. Results

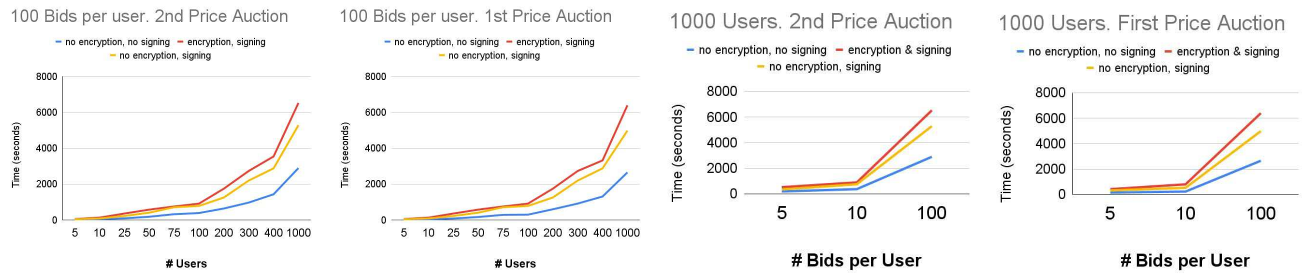
Our results are pertinent to the Time Analysis under 3 different scenarios. Here we measure the time it takes for both the bidding and declare phase cumulatively in a sealed bid second price auction since we wanted to measure the performance for one whole round of auction.

6.2.1. Encryption and Signing. In this scenario, once the public key is available on the chaincode for retrieval, each bidder uses the Encryption service to download encryption key to encrypt the bid with the chaincode public key and signs it with bidder's own private key. The chaincode handles the verification and decryption on receipt of the bid. As shown in Figure 2a and Figure 2b, the red line indicates that the time taken to complete the auction when bids are fixed to 100 per user and users are varied from 5 to 1000. Time taken is longest compared to the other two scenarios because the overhead to encrypt, sign, verify and decrypt takes the most time. Moreover, since the number of bids for 1000 users is the highest where the total number of bids become 1000*100 = 100,000 bids as shown by the red line in Figure 2c and 2d, the declaration phase is also the longest. This validates the general rule of thumb being security always comes at a cost to efficiency.

6.2.2. No encryption, Only signing. This setting involves the case where the bids merely being signed for integrity purposes and no encryption is involved (i.e., no privacy protection for the bids). Here, we are able to gain a little bit of efficiency due to the removal of encryption and decryption operations. However, total time required still does not drop that much due to the fact that since bidding is done randomly, in order to verify the signed bid, the chaincode has to retrieve each distinct bidder public key from the blockchain state. This causes the maximum overhead as depicted by the yellow line in Figure 2.

6.2.3. No encryption, No signing. This is the final scenario to test where there is neither encryption nor signing of the bids is involved. Clearly, in this setting, an attacker not only see all the bids, it can further issue fake bids. Not surprisingly, this is the most efficient scenario since there is no requirement to encrypt, decrypt, sign or verify the bids. The chaincode does not have to retrieve keys from the blockchain state but still have to retrieve the stored bids from the blockchain state. The blue line in Figure 2 indicates that it is the most efficient as compared to the other two scenarios.

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on June 19,2022 at 19:42:21 UTC from IEEE Xplore. Restrictions apply.



(a) **Fixed Bids, Variable Users** (b) **Fixed Bids, Variable Users** (c) **Fixed Users, Variable Bids** (d) **Fixed Users, Variable Bids**

Figure 2: SecAucTee *Experimental Results: Time Analysis for contract execution over Private Blockchain Hyperledger deployed in Intel SGX.*

7. SECURITY AND PRIVACY ANALYSIS

In this section, we give an overview of the oblivious execution guarantees provided by our system. First, we assume that due to SGX capabilities, a malicious attacker cannot observe the register contents. In other words, an attacker can only observe the memory access patterns. Below, we formally define what is leaked during the program execution for an adversary that can observe only memory access patterns. Protection against other type of side channel attacks such as timing, energy consumption is outside the scope of this work, and not modeled in our security analysis.

Let, $Bid = \{Bid_1, .., Bid_\alpha\}$ be the bids submitted by different bidders, $EBid = \{EBid_1, ...EBid_\alpha\}$ be the encrypted version of those bids, O be the output of the auction after the execution of the auction smart contract.

- **Memory Access Pattern ($\mathcal{A}_p$):** Suppose $\mathcal{AUC}$ is the auction executed and during the execution $\mathcal{AUC}$ accessed $\{EBid_1, ..., EBid_z\}$, then, $\mathcal{A}_p$ is the sequence of memory locations that are written and/or accessed during the $\mathcal{AUC}$ execution. The memory access pattern captures all the memory access sequence of during the secure auction execution.
- **Adversary View (v):** The view of an adversary observing the system is $v = \{EBid, \mathcal{O}\}$. View is the information that is accessible to an adversary.

Now, there exists a probabilistic polynomial time simulator $\mathcal{S}$ that can simulate the memory access pattern $\mathcal{A}_p$ using the adversary v.

Theorem 1. *The proposed secure auction execution does not reveal anything other than the view v since all the $\mathcal{A}_p$ can be simulated using the v.*

Proof. We show there exists a polynomial size simulator $\mathcal{S}$ such that the $\mathcal{A}_p$ can be simulated using only the v.

First, the $\mathcal{S}$ generates random values that are the same size as the $EBid$, and put them to the memory locations that will be accessed by $\mathcal{AUC}$. Please note that irrespective of the input values, algorithm 3 and algorithm 4 *access exactly the same memory locations*, and update fix locations irrespective of the input. Then $\mathcal{S}$ will access these locations, and put random values for updated memory locations. Finally, $\mathcal{S}$ will output the $\mathcal{O}$ (i.e., the correct output of the auction). First of all, please note that the $\mathcal{S}$ access exactly the same locations as the real execution. Since, all the updates reflected to memory is encrypted by the TEE, the random values generated by the $\mathcal{S}$ is computationally indistinguishable from the actual values outputted during the execution.[2] Therefore, $\mathcal{S}$ can simulate the $\mathcal{A}_p$ using only the v. $\square$

The above theorem implies that from the memory access pattern point of view, an attacker can only learn the total number of bids, and the auction output and nothing else.

8. RELATED WORK

TEEs and Public Blockchains:. TEEs and smart contracts have supplementing properties that allows us to design a system to utilize the strengths of both technologies and design a distributed, highly available and reliable yet secure and privacy-preserving application. For example, Ekiden [10] has proposed a novel architecture that leverages TEEs to deploy smart contracts in enclaves thereby separating consensus from execution. They have created a privacy-preserving solution for running smart contracts in a TEE environment. Although Ekiden [10] ensures smart contract execution while preserving confidentiality, they do not address the sensitive information leakage when an attacker can observe the memory locations when the smart contract is performing computation over the blockchain deployed in an Intel SGX enclave. Since the computations are performed obliviously in our system, the attacker will not have knowledge of the bidders or the bids being compared even though he/she may have access to system memory.

Auctions and TEEs:. There are multiple proposed systems that claim to preserve privacy by using Intel SGX and a Private Blockchain architecture but fail to cover all attack vectors. Strain [20], for example, proposes a new sealed-bid auction protocol over a permissioned blockchain to guarantee bid confidentiality against malicious parties.

2. This is sometimes referred as semantically secure encryption. It is assumed that semantically secure encrypted values are computationally indistinguishable from random values [19].

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on June 19,2022 at 19:42:21 UTC from IEEE Xplore. Restrictions apply.

It takes advantage of zero-knowledge proofs, their solution allow participants the ability to verify each outcome based on broadcast strategy. Although Strain's architecture is efficient and latency is asymptotically optimal, it does not use a secure hardware to protect building blocks like cryptographic keys, system logs, transaction details, etc.

Privacy-preserving Computation using TEEs:. TEEs have been used for supporting privacy-preserving data analytics outside the context of blockchains and smartcontracts. For example, SGX-BigMatrix [9] have implemented a practical data analytic framework with trusted processors. Another such implementation of oblivious execution of programs has been carried out by [18]. They attempt to leverage programming language techniques to offer efficient memory-trace oblivious program execution and provide formal security guarantees.

Although, these types of solutions address some data security and privacy challengs, they do not address distributed applications like Blockchains. Their oblivious execution of the data analytics framework serves as a motivation in our effort to implement oblivious execution of smart contracts over a Blockchain architecture.

9. CONCLUSIONS

As Blockchains gain significance in the computer industry with more and more distributed applications being developed over smart contracts to serve out various use-cases, privacy and security becomes a dominant challenge that needs to be addressed. We propose a novel framework to achieve data privacy and security for any auction algorithm as precedent to achieve the same for distributed applications as smart contracts deployed over a private blockchain running inside secure enclaves. Furthermore, we maintain privacy and security by oblivious execution of the auction algorithm inside the smart contracts. We have also implemented an end-to-end encryption service to send and receive signed encrypted messages between the user and the smart contract running inside the enclave.

ACKNOWLEDGEMENTS

The research reported herein was supported in part by NSF awards, CNS-1837627, OAC-1828467, IIS-1939728, DMS-1925346, CNS-2029661, OAC-2115094 and ARO award W911NF-17-1-0356.

References

[1] M. Abdelhamid and G. Hassan, "Blockchain and smart contracts," in *Proceedings of the 2019 8th International Conference on Software and Information Engineering*, ser. ICSIE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 91–95. [Online]. Available: https://doi.org/10.1145/3328833.3328857

[2] H. Galal, "Verifiable sealed-bid auction on the ethereum blockchain," 03 2018.

[3] H. Desai, M. Kantarcioglu, and L. Kagal, "A hybrid blockchain architecture for privacy-enabled and accountable auctions," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 34–43.

[4] E.-O. Blass and F. Kerschbaum, "Borealis: Building block for sealed bid auctions on blockchains," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 558–571. [Online]. Available: https://doi.org/10.1145/3320269.3384752

[5] T. D. T. Nguyen and M. T. Thai, "A blockchain-based iterative double auction protocol using multiparty state channels," *ACM Trans. Internet Technol.*, vol. 21, no. 2, Mar. 2021. [Online]. Available: https://doi.org/10.1145/3389249

[6] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica, "DIZK: A distributed zero knowledge proof system," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 675–692. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/wu

[7] J. Montenegro, M. Fischer, J. Lopez, and R. Peralta, "Secure sealed-bid online auctions using discreet cryptographic proofs," *Mathematical and Computer Modelling - MATH COMPUT MODELLING*, vol. 57, 06 2013.

[8] F. Shaon and M. Kantarcioglu, "Sgx-ir: Secure information retrieval with trusted processors," in *DBSec*, 2020.

[9] F. Shaon, M. Kantarcioglu, Z. Lin, and L. Khan, "Sgx-bigmatrix: A practical encrypted data analytic framework with trusted processors." New York, NY, USA: Association for Computing Machinery, 2017.

[10] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, Jun 2019. [Online]. Available: http://dx.doi.org/10.1109/EuroSP.2019.00023

[11] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti, "Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric," 2018.

[12] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *NDSS*, 2012.

[13] R. P. McAfee and J. McMillan, "Auctions and bidding," *Journal of Economic Literature*, vol. 25, no. 2, pp. 699–738, 1987. [Online]. Available: http://www.jstor.org/stable/2726107

[14] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[15] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[16] P. Robinson, "Requirements for ethereum private sidechains," 2018.

[17] M. Bennett, "How do first price and second price auctions differ?" [Online]. Available: https://oko.uk/blog/first-price-vs-second-price-auctions

[18] C. Liu, M. Hicks, and E. Shi, "Memory trace oblivious program execution," in *2013 IEEE 26th Computer Security Foundations Symposium*, 2013, pp. 51–65.

[19] O. Goldreich, *The Foundations of Cryptography*. Cambridge University Press, 2004, vol. 2, ch. Encryption Schemes. [Online]. Available: http://www.wisdom.weizmann.ac.il/ oded/PSBookFrag/enc.ps

[20] E.-O. Blass and F. Kerschbaum, *Strain: A Secure Auction for Blockchains: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, 08 2018, pp. 87–110.

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on June 19,2022 at 19:42:21 UTC from IEEE Xplore. Restrictions apply.