Increasing security of an e-auction smart contracts with Intel SGX trusted hardware

Rohan Deshamudre[1]
Supervisor(s): Kaitai Liang[1]**, Marco Zuniga** [1]
[1]EEMCS, Delft University of Technology, The Netherlands
r.deshamudre , kaitai.liang

Abstract

Smart contracts allow for the collaboration and transaction processes between multiple parties/organisations to be automated and conducted in a neutral environment. In many situations these agreements are confidential and running a smart contract that contains private/sensitive information on a public blockchain network which is transparent and shares data with all nodes is not a sensible method of execution. With the use of hyperledger fabric which is a private and permissioned network merged with Intel SGX trusted execution environments, a feasible solution can be proposed to tackle this problem.

This paper first analyses some of the security issues faced by smart contracts on hyperledger fabric, compares the current solutions for blockchain based e-auction systems and discusses security measures such as encryption, attestation and combination with different trusted hardware modules.

The second part proposes a solution to combine Intel SGX trusted hardware with a e-auctions fabric smart contract, discusses the architecture and step by step implementation of a prototype, explains the security enhancements of this method by comparing just hyperledger vs the new method, and lastly outlines the limitations and future extensions. This solution mitigates many vulnerabilities by isolating execution of chaincode with sensitive data in a TEE and minimizing the trusted computing base to reduce latency and overhead.

1 Introduction

Blockchain technology is a field which has gained immense popularity over the past few years due to the large number of industry use cases that can be derived because of its decentralized structure and security benefits. It is an immutable distributed ledger that is shared among the peer nodes of a network allowing data and information to be communicated easily. Although it is mainly associated with cryptocurrencies, there are numerous other use cases such as cross border payments, secure sharing of data,e-voting and e-auction systems which make it a hot topic for research. Blockchain networks can be public and permissionless or private and permissioned. Public and permissionless means that there are no restrictions as to who joins and interacts in the network whereas private and permissioned is made only for a verified set of participants and not open for everyone. The focus of this report will be on the latter which can be used for transactions which require a secure environment.

Hyperledger fabric is an enterprise-grade permissioned distributed ledger technology platform with a highly modular and configurable architecture vastly used in scenarios where confidential data needs to be stored securely and privately [10]. It is built with the concept of smart contracts in mind. Smart contracts are programs or transaction protocols stored on a blockchain that automatically execute when a set of predetermined conditions are met [11]. It encodes the business logic of the transactions between the peer nodes within the network.

However, smart contracts are still susceptible to attacks and can allow access to data stored in the state DB or ledger. Previous research regarding some vulnerabilities such as rich queries, global variables and pseudorandom number generators is conducted by Catherine Paulsen [15] by deploying some smart contracts on test network and exploiting the issues to find countermeasures. Research regarding enhancing privacy and security of smart contracts using symmetric encryption methods is also conducted by Rado Stefanov [16].

This research explores the concept of merging smart contracts with silicon based trusted hardware. More specifically, it will look at **"How can an e-auctions smart contract be executed within an Intel SGX trusted execution environment in order to strengthen the security concerns?"**

Trusted hardware is a set of equipment protected through resource access control facility in the trusted computing base and includes a trusted processor, trusted printer and trusted console.[4] Intel SGX is a popular trusted hardware module that provides a secure computing base that guarantees confidentiality and integrity of the data shared to it. It helps protect data using application isolation technology as developers can

run confidential smart contracts in an environment where the input and output values are secure from any leaks or modification.[12]. Marcus Brandenburger, Christian Cachin, Alessandro Sorniotti, and Rüdiger Kapitza have published multiple papers paper which go over the challenges of combining trusted hardware with smart contracts [2], discuss problems and pitfalls and highlight the underlying process. [13]
The solutions proposed by the above articles all lack structure and maturity to run and widely deploy general purpose computations combining blockchain with trusted hardware. Issues such as rollback attacks, protocol integration, state continuity etc are still prevalent. Moreover, specifically in the field of e-auctions, most available systems require a strong and trusted third party to mediate the process and ensure the integrity of transactions are withheld [5]. However, with the privacy protection benefits of blockchain technology, encryption techniques, and "tamper proof" design of Intel SGX, aspects like data traceability/verifiability, data transmission privacy and anonymity of participants can be improved.
The analysis will be conducted by answering 5 sub questions:

- What are the security analysis procedures and current vulnerabilities in fabric smart contracts?
- What are the current Blockchain based e-bidding systems?
- How to enhance security and privacy of the smart contract using digital signatures and encryption methods ?
- What is a feasible design to merge Intel SGX trusted hardware attestation with an e-auctions smart contract?

This research report will be split into multiple sections. Section 2 will go over the methodology followed to complete the research and answer the questions stated above. Section 3 will give background about all the technology building blocks that are involved in the process. Section 4 discusses the system design and the architecture of the approach to use Intel SGX to execute the private chaincode of blind e-auctions.

2 Methodology

At the beginning of the project, in order to get a good understanding of the blockchain platform, the first step was to do a comprehensive study about hyperledger fabric, and smart contracts. By reviewing the documentation and reading research papers/articles related to the topic with a time frame from 2018-2022 in mind, a clear understanding of the core components and construction and execution of fabric smart contracts was gained. Scientific papers on the current smart contract vulnerabilities were taken into consideration to understand areas of improvement. Some of the biggest vulnerabilities mentioned in [15] such as updates using rich queries, pseudorandom number generator, global variables and their exploits were understood and the suggested countermeasures were analyzed.

The next step in the process was to gain a better understanding of trusted hardware and the different modules available. Research about the pros and cons and technical differences between modules such as Intel SGX, TPM, and DICE was conducted to pick which module to use in the implementation. After assessing the technology on aspects like, access to resources, ease of implementation and best security benefits, Intel SGX was the chosen trusted hardware module. Furthermore, to determine a feasible and safe way to merge Intel SGX, open source github repositories, scientific papers and cyber security blogs were reviewed leading to the use of the fabric private chaincode architecture.
After gaining knowledge on the platforms, tools and systems required to merge trusted hardware and smart contracts, different applications that require a secure environment for data storage and transactions were brainstormed. Use cases such as e-voting, supply chain management, financial data recording, trading activity etc were taken into consideration and a blind auctions smart contract was picked to be implemented. Based on this, a main research question and sub-questions to shape the paper were formulated leading this research towards merging an blind e-auctions fabric smart contract with Intel SGX attestation.

Now that there is a clear research question formulated, the next step was to get familiar with programming in the platforms and frameworks and begin the implementation of the smart contract. The first step, was to install hyperledger fabric, examine how to build a network and set up a channel to deploy a sample smart contract. During this process, more information about the different languages in which the smart contracts can be authored and their pros and cons were discovered. The next step was to run a hello world smart contract with fabric private chain code to understand the merger with Intel SGX. Once sample code was run on both hyperledger and Intel SGX, the entire chaincode and application code for the blind auctions was implemented. The auction part of the chaincode is written in Go and executed within the untrusted part of the peer whereas chaincode regarding the blind bids is written in C++ and conducted from within the SGX enclave.

In order to further improve the security apart from running chaincode with sensitive information in the trusted execution environment is to have an encryption for the bids to ensure the privacy. Different encryption methods were compared and contrasted to find a suitable one to include in the implementation of the blind smart contract. Along with this, a digital signature or peer endorsement was required to make sure that the transactions are being committed by the authorized parties.The final step to complete the research was to analyze the security enhancements by considering overhead, size of trusted computing base, endorsement throughput and latency.

3 Background Information

This section provides more background information regarding some of the major decision points of this research. Section 3.1 dives into the details of current smart contract vulnerabilities, section 3.2 compares Intel SGX with some other trusted hardware modules and section 3.3 goes over some of the current blockchain based e-auction solutions available.

3.1 Current smart contract vulnerabilities

Smart contracts executed in a blockchain can be vulnerable to a variety of malicious attacks. These attacks can lead to several issues such as leakage of confidential information, massive financial losses, disruption in the supply chain management for businesses, network compromise and more. Some of the specific and inherent problems include issues like updates using rich queries, and global variables. Using rich queries, a malicious attacker could access the world state of a peer and make changes to it without creating any transactions or other records of this update being made. This illegal value propagation is done using couchDB as state database on assets stored in the ledger[15]. The second issue mentioned above is global variables which are useful in many cases but cause a lot of trouble in smart contracts as they need to be deterministic. Global variables can be used to make the smart contract behave unexpectedly as if a client invokes a function, this changes the global variable value for everyone without requiring endorsement from the other peers. Countermeasures or alternative solutions to mitigate these issues other than using analysis tools are not widely discussed which gives the combination of smart contracts and trusted hardware great scope to be a feasible solution for the industry.

3.2 Comparing trusted hardware modules

There are many trusted hardware modules that can be used for attestation. Some examples include Intel Software Guard Extensions (SGX), Trusted Platform Module (TPM) and Device Identifier Composite Engine (DICE). Each module has its own positive and negative aspects.

- Intel SGX verifies the code before executing it and the attestation can be done using two mechanisms, Enhanced Privacy ID (EPID) and Data Center Attestation Primitives (DCAP). It checks to ensure that the code is genuine as well as to ensure that the enclave is genuine using hashes. The positive aspects of Intel SGX is that it is easy to generate and verify attestation quotes, it is highly scalable, the code and data in an enclave is measured while loading an enclave and it can also be seamlessly integrated by third party applications as long as prerequisites are met. [9]
- TPM follows a process called chain of trust for attestation. This is the when each stage of execution verifies the next stage which allows for the trusted computing base to be minimized. However, if there are any data integrity violations detected, TPM is passive and cannot do anything other than seal itself. Similar to SGX, TPM also contains an endorsement key to verify the authenticity of the enclave.[8]
- DICE allows for cryptographic device identity, attestation and data encryption just like the other two modules discussed. It incorporates a mechanism called latching to disable read access to the data before passing verification. It also has the feature to utilize one way cryptographic functions to transform device identity.

After comparing the different available modules and assessing the key pros and cons related to the situation of e-auctions smart contract, as well as due to the variety of available resources, Intel SGX was picked as the trusted hardware to utilize in this research.

3.3 Blockchain based e-auctions

A blind e-auctions system is an electronic auctions platform which allows bidders to place bids on a product that is being sold in the auction privately. The bids are hashed and are kept secret until the auction is closed and then later revealed to announce the winner. Blockchain technology offers great benefits such as privacy protection and immutability to enable efficient e-auctions applications. However, all current systems require a neutral mediator and uses tradiational databases. With the use of blockchain, these databases can be replaced and business logic is replaced by smart contracts. Additional security features such as encryption and trusted hardware attestation improves anonymity, traceability and verifiability of data.

4 Technologies

This section provides an overview of the main technical components that are relevant towards the creation of a Blind Auctions smart contract within Hyperledger Fabric. Section 3.1 introduces Hyperledger fabric and the general structure a fabric application follows whereas Section 3.2 is more about the integration of Intel SGX trusted hardware.

4.1 Hyperledger Fabric

Hyperledger Fabric was created to allow creation of private networks which can be accessed by only a set verified of participants. All the nodes part of the private network know each other rather than being anonymous, operate under the same governance model and maintain a copy of the data. A membership service provider defines rules to authenticate, validate and identify members who can access the network. The transactions within this network are private and channels can be created to share private information with only a subset of the nodes. The three main aspects of a fabric network are clients, peers and an ordering service.The transaction process has a execute-order-validate flow. Chaincodes or transaction requests are submitted by the client/user to the peers which then executes the chaincode and replies back to the client with a response. All responses are then collected and submitted to the ordering service where it gets ordered and submitted to the blockchain.

Figure 1 below shows the general structure of channel in a simple private network and this section will outline the key components and features that enable this enterprise grade blockchain solution. A channel is a sub network within the full private network with only some selected peers that are part of a process.

- Organizations: A network is formed by the collaboration of multiple organisations and each organisation represents a collection of peer nodes and the application developed by that organisation. For a network to be useful, there need to be atleast 2 organizations and these organizations are not a physical part of the network. They

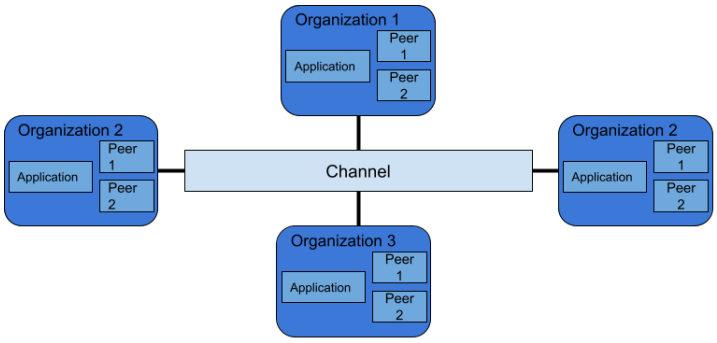

Figure 1: Structure of a network

communicate with each other through a channel ensuring that the information is not shared with any other participants of the network.

- Peers: Peers are physical participants and each peer stricly belongs to one organization in the network. They represent an individual within an organization on the blockchain network and hosts the ledgers and smart contracts (chaincode) within them. Their job is to update the world state and execute the chaincode when the application requests it. The structure of a peer node can be seen in Figure 2 below.
- Application: using Fabric SDKs and FPC client SDK, the application interacts with the blockchain network to perform 6 steps: select an identity from a wallet, connect to a gateway, access network, construct smart contract transaction request, submit transaction, and process the response. This can be implemented in many ways such as a web application, mobile application, etc.
- Ordering service: does the transaction ordering, broadcasting and packaging into blocks, maintains a list of organizations allowed to create channels and enforces basic access control.

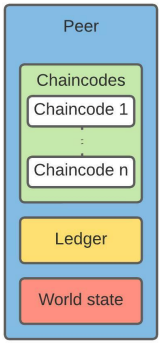

Figure 2: Components of a peer node

Figure 2 represents the internal components of a peer node. Within a peer node there are 3 main aspects:

- Chaincode: executable logic for transactions that accesses the ledgers to get/put/delete records. With regards to to a blind auction, transactions such as creating the auction, registering users, creating/submitting bids, evaluating the winner, and ending the auction are all done here.
- Ledger and world state: records all the state-updates of transactions between parties generated by chaincodes immutably. Consists of two parts, the blockchain and the world state database. The blockchain records the transaction history of chaincodes whereas the world-state keeps track of the current values of the business objects in key-value pairs. Every peer node in a fabric network has a copy of the ledger.

4.2 Intel SGX

Intel SGX stands for software guard extensions and is a trusted hardware module built into the CPU's which allow operating system and user level code to define private regions of memory called enclaves [3]. It adds hardware enforced security to commodity platforms such as hyperledger fabric and enables peer to execute chaincode in a trusted execution environment. All the data within this enclave is protected and only parties within the enclave can perform any CRUD operations. In instances where a secure environment to execute private code and store sensitive data is necessary, SGX plays an important role to ensure confidentiality and integrity. Especially in a blockchain based solution where there is complete transparency and all nodes within a network see and store a copy of the transactions, the lack of privacy is highlighted. The service is also highly scalable making it attractive in the situation of smart contracts which may involve large organisations and thousands of peers. [14]

Confidential code and data can be partitioned and isolated into trusted execution modules which establishes a secure container, and the remote computation service user uploads the desired computation and data into the secure container. It encrypts sections of memory using security instructions native the CPU. [6]. It provides features such as attestation to ensure that the applications being executed within the enclave have not been tampered with and perform what is intended by the developer . This is done by comparing two hashes, one which is generated while loading data into the enclave called mrenclave and the other produced by the application. If the hashes match, the enclave is started and code can be executed.

Another feature that further ensures data integrity even in the case of crashes where state is lost is data sealing. This allows for data to encrypted and stored externally ensuring that even in an event of a crash or a restart, the sealed encrypted data can be loaded and decrypted.

5 Implementation

The aim of the implementation is to develop a blind e-auction smart contract that is safe from many vulnerabilities such as data leaks, roll back attacks, updates using rich queries etc. In order to ensure the safe execution of confidential chaincodes and protect private data from untrusted peers, a trusted execution environment is introduced. The private chaincode is then run within this enclave and called upon from the application

by the clients. Section 4.1 goes over the system design of the solution whereas 4.2 goes over the prototype.

5.1 System design

The following framework allows the development of application and chaincode with encrypted data and attestation which can be accessed only by trusted parties. As a result, there are additions to the structure of a peer compared to what is seen in Figure 2.

A high level rundown of the process is that some chaincodes which contain sensitive information are isolated from the rest of the program and executed within the trusted execution environment which is the Intel SGX enclave. [1] This chaincode is stateless and uses getState and putState to perform read and update operations on the ledger. Other chaincodes and the application within an organization which is the untrusted part of the peer can communicate and proxy this chaincode through the SHIM functionality. In addition, there is an enclave registry and enclave endorsement validator which are responsible for maintaining the identities of all chaincode enclaves and validating transactions performed by a chaincode enclave respectively. [17]

The new components listed below and outlined in Figure 3 describe their roles in the design of the blind Auctions start contract merged with Intel SGX trusted hardware implementation.

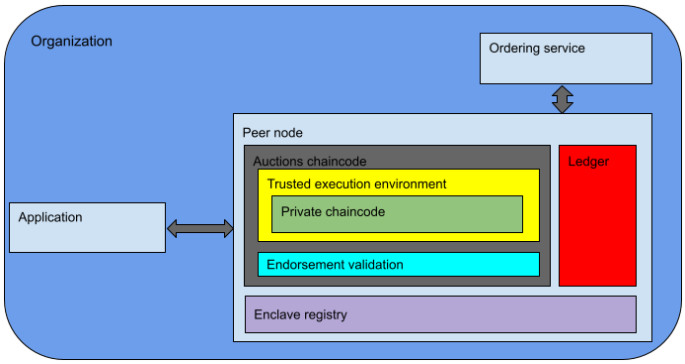

Figure 3: Components of an organization

- Trusted execution environment (chaincode enclave): isolates and executes chaincode that contains confidential transactions and data. The enclave extends fabric private chaincode with attestation (digital signature), state encryption (data sealing), and many more advanced capabilities. Creating bids, submitting bids, quering bids and revealing bids is done in these chaincode enclaves.
- Enclave registry: maintains a list of all existing chaincode enclaves, performs attestation to ensure genuine applications are executed and stores the result on the blockchain. Attestation proves that specific chaincode executes in an actual enclave and enables peers and clients to inspect before committing or invoking operations.
- Endorsement validation: validates transactions outputted by the TEE enclave and checks for a valid signature issued by a registered enclave. If validation is successful, the transaction is submitted to the ledger.

5.2 Prototype

The process starts by deploying the auction network and auction channel which consists of 2 organizations for this prototype implementation which can be increased according to requirements. Buyers and sellers of the auction are deployed onto the network using certificate authorities which register the identities. When a new peer is added to the network, the mrenclave hash of the trusted execution environment and the hash produced by the application need to match in order to validate that the SGX enclave is trusted and genuine. This enclave is identified by the public and private key pair that it generates. The smart contract is then deployed onto this auction network to be executed. Any interactions by the client with this smart contract is done through a set of Node.js applications. These applications call the auction chaincodes in order to communicate with the ledger and make the state updates. Before even starting the blind-auction there are some steps that need to be fulfilled to setup the different users of the system and they are as follows:

1. Enrol the certificate authority administrators of the two organizations, Org 1 and Org 2. These are then added to the admin wallet on your local file system.
2. These admins can then register the seller who wants to sell an item in the auction and the bidders who want to bid on the product being sold. The seller is registered and will create the auction and can belong to either of the organisations.
3. Bidders are then registered for each organization and added to the user wallet. The number of bidders can be increased or decreased depending on the organization and the based on the situation.

Once all the stakeholders of the auction are created and enrolled, the next step is to create the actual auction called by the initAuction function in the application. The client specifies which organisation, who the seller is, name of the auction and the item being sold in order to invoke the chaincode that creates the auction. Figure 4 below shows the structure of a blind auction. The chaincode written in the untrusted part of the peer then performs peer endorsement to verify who is creating the contract and submits the transaction to update the state in the ledger using the put-State function. This auction can then be queried based on the auction name. The status of the auction is now set to open. This can be seen in Appendix A.

When the auction is open, bidders from the different organizations can submit their bid to try and buy the product for sale. This is done by invoking the createBid function in the application which takes in the organisation of the bidder, bidder name, auction name and the bid price they are submitting as parameters. All chaincode related to the bid is executed within the TEE and stored in the private data collection f the bidders organisation. At this step, attestation is performed to ensure that the enclave being called exists in the list of trusted

```
type BlindAuction struct {
    EventType          string                 `json:"eventType"`
    SellingItem        string                 `json:"sellingItem"`
    Seller             string                 `json:"seller"`
    Price              int                    `json:"price"`
    Organisations      []string               `json:"organisations"`
    HiddenBids         map[string]HiddenBid   `json:"hiddenBids"`
    RevealedBids       map[string]RevealedBids `json:"revealedBid"`
    WinningBidder      string                 `json:"winningBidder"`
    AuctionState       string                 `json:"auctionState"`
}
```

Figure 4: Structure of a blind auction

enclaves and that the chaincode being called is genuine. The key generation code snippet can be seen in Appendix C. Once this passes, the enclave invoke function is called and chaincode is executed. This bid information is then encrypted by creating a composite key and storing the hash of the bid. The structure of both the hidden bid and the full bid once revealed can be seen in the figure 5 below. Once the bid has been created, they are submitted to the auction using the submitBid function and can be queried based on the bidId. They key generation and bid creation code can be seen in Appendix B.

```
//Structure of the bids that are yet to be revealed
type HiddenBid struct {
    Organisation    string      `json:"organisation"`
    BidHash         string      `json:"bidHash"`
}

//Structure of the full revealed bid
type RevealedBid struct {
    EventType       string      `json:"eventType"`
    Price           int         `json:"price"`
    Organisation    string      `json:"organisation"`
    Bidder          string      `json:"bidder"`
}
```

Figure 5: Structure of the bids

After a certain amount of time and once all bids are submitted, the seller who is also the creator of the auction will change the status to closed using the closeAuction function after which no further bids will be accepted. The final step then before ending the auction is to reveal the bids and evaluate the winning bidder. Once the auction is closed, the revealBid function can be invoked which performs checks to see if the identity that is trying to reveal the bid is the same one that submitted the bid and if the hash of the revealed bid and the hash of the hidden bid stored in data is the same to ensure the bid is unchanged. This takes the bidders organisation, bidder name, auction name, and their bid ID as input. The chaincode to reveal the bids is also executed from within the Intel SGX enclave to ensure confidentiality and integrity of the data. Post this, the full bid is revealed to the whole auction and all the parties involved. Once all the bids are revealed, they are evaluated to find the highest bid and that bidder is announced as the winner. The auction can then be ended by calling endAuction by the person that started it. **The code repository is available at: https://github.com/Rohan-Deshamudre/blind-auction-rohan.**

5.3 Measurements

The trusted computing base (TCB) consists of the chaincode that runs inside the SGX enclave and is written in C++. This includes about 300 lines of code whereas the chaincode in the untrusted part of the peer written in Go is around 230 lines.

Figure 6 is a graph that showcases the effect of an increasing number of users on the latency of the transactions of the auction. It is seen that initially with just 1 or 2 users, the latency is around 20ms whereas with 4 users the latency decreases down to around 10ms. From that point till the max number of users tested which is 128, the latency increases almost exponentially. The latency when the number of users is 128 is 110ms and looks to keep increasing as the number of users increase even further.

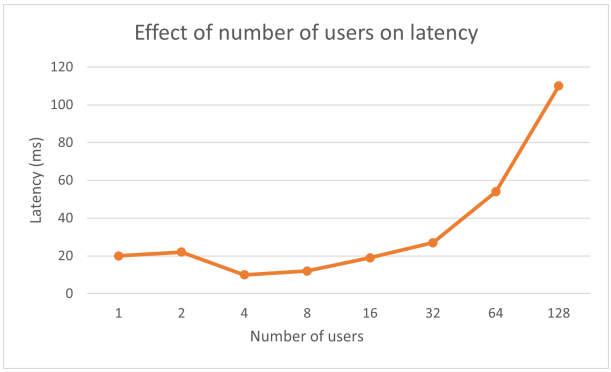

Figure 6: Effect of number of users on latency

Figure 7 is a graph that showcases the effect of an increasing number of bids on the end to end response latency of the blind auction. It is seen that the latency increases as the number of submitted bids goes up. The initial jump from 10 to 100 bids is not significant whereas from 100 to 1000 bids, the latency almost doubles.

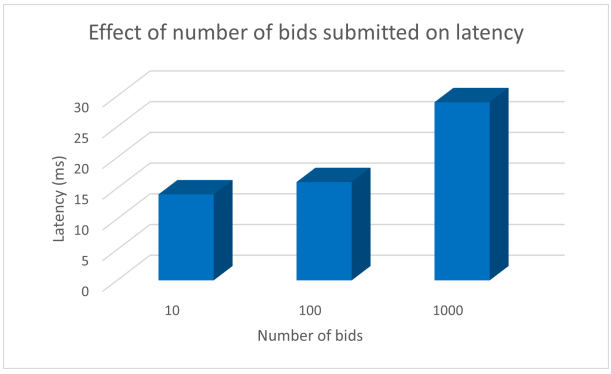

Figure 7: Effect of number of bids on latency

6 Responsible Research

During the process of this writing this paper, ethical aspects both in terms of positive and negative repercussions that could be reflected on society were considered. The second aspect of responsible research is to consider how reproducible these methods of implementation are. Both these topics are discussed in the sections below.

6.1 Ethics

The purpose of this research is to improve the security of smart contracts with the utilisation of Intel SGX trusted hardware attestation. The paper reports vulnerabilities within smart contracts, a system to utilize trusted hardware based attestation to protect integrity and confidentiality of sensitive data, and a prototype implementation. The author of this paper has no personal affiliations with any smart contract technology or Intel SGX trusted hardware that could result in a conflict of interest. The issues and countermeasures implemented were tested in a local environment and did not affect any applications available to be used by public at the moment. None of the data involved was personal to anyone.

6.2 Reproducibility

In order to make the system design and method to utilize Intel SGX to enhance security of the e-auction smart contract more reproducible, code snippets to some relevant sections and the link to the code repository has been added as part of this report. The provided code in the appendix along with the diagrams makes it easier to visualize and understand the process. However, hyperledger fabric and and SGX knowledge is required to undestand the terminology and the architecture explained in this research. In order to get all the technologies involved installed, there are quite a few steps that need to be taken which can be quite time consuming and tedious. However, implementation of smart contracts can be researched and verified by previous literature and cited sources whereas the available resources for functionalities of SGX is a lot more limited.

7 Conclusions and Future Work

The aim of this research was to identify how to enable smart contract execution in combination with trusted hardware for the security enhancements of blockchain applications with high security demands. In particular the question of how to execute an e-auctions smart contract within an Intel SGX trusted execution environment was addressed. In order to gain knowledge of the full picture, sub topics such as current smart contract vulnerabilities, blockchain based e-auction systems, security enhancements using encryption and attestation were also researched. By going over the involved technologies such as hyperledger fabric and Intel SGX in detail and by creating a feasible design to utilize trusted hardware attestation to increase smart contract security, this paper aims to create an example for other similar industry level blockchain use cases.

Although smart contract vulnerabilities such as global variables and updates using rich queries were mentioned, the enclaves are susceptible to issues such as roll back attacks, state continuity and protocol integration which are the limitations of this project. The issue of roll back attacks causes peers to reset and change the order of operation invocation. The trusted computing base was kept to a minimum utilizing the enclave only for parts of the code that are confidential. The architecture and prototype created addresses these issues and ensures integrity of chaincode and data. The end to end latency of the auction transactions increases as the number of clients increase and as the number of bids in the auction increases.

Future improvements would be to explore more features of SGX such as data sealing as well as remote attestation from third party applications. The possibilities of combining multiple trusted execution environments like TPM, SGX and DICE to utilize their strenghts would be a good step. [7]

A Appendix

A.1 A

This section of the appendix contains a code snippet of the chaincode to start the auction and then the response of how the auction structure looks once it is made.

```
func (ac *AuctionContract) initAuction(ctx contractapi.TransactionContextInterface, sellingItem string, auctionName string) error {

    sellerId,err := ac.getSubmitter(ctx)
    if err != nil { return fmt.Errorf("could not get sellerId %v", err) }

    sellerOrgId,err := ctx.GetClientIdentity().GetMSPID()
    if err != nil { return fmt.Errorf("could not get sellerOrgId %v", err) }

    blindAuction := BlindAuction {
        EventType:     "blind-auction",
        SellingItem:   sellingItem,
        Seller:        sellerId,
        Price:         0,
        Organisations: []string{sellerOrgId},
        HiddenBids:    make(map[string]HiddenBid),
        RevealedBids:  make(map[string]RevealedBid),
        WinningBidder: "",
        AuctionState:  "open",
    }

    blindAuctionJSON, err := json.Marshal(blindAuction)
    if err != nil { return err }

    err = ctx.GetStub().PutState(auctionName, blindAuctionJSON)
    if err != nil { return fmt.Errorf("could not put the blind-auction into data: %v", err) }

    err = setOrgValidation(ctx, auctionName, sellerOrgId)
    if err != nil { return fmt.Errorf("could not set endorsement for organisation: %v", err) }

    return nil
}
```

Figure 8: Auction creation chaincode

```
Result: BlindAuction: {
  "eventType": "auction",
  "sellingItem": "ferrari",
  "seller": "x509::CN=seller,OU=client+OU=org1+OU=
  "price": 200000,
  "organisations": [
    "Organisation 1",
    "Organisation 2"
  ],
  "hiddenBids": {},
  "revealedBids": {},
  "winningBidder": "",
  "auctionState": "open"
}
```

Figure 9: Auction response

A.2 B

This section contains the code snippet of the chaincode written in C++ in SGX to create a bid and the the next figure

showcases the structure of both a bid in its hashed form as well as in its revealed form.

```
std::string create_bid(std::string bidder, int price, std::string organisation, std::string auctionName, shim_ctx_ptr_t ctx) {

    uint32_t len = 0;
    uint8_t bytes[1024];
    get_state(auctionName.c_str(), bytes, sizeof(bytes), &len, ctx);

    if (len == 0)    {
        LOG_DEBUG("Auction does not exist");
        return "AUCTION_DOESNT_EXISTING";
    }

    blind_auction_t blind_auction;
    unmarshal_auction(&blind_auction, (const char *)bytes, len);

    if (!blind_auction == 'open')
    {
        LOG_DEBUG("Auction is closed");
        return "AUCTION CLOSED";
    }

    //create bid hash
    std::string bid_hash = create_composite_key(".PREFIX." + auctionName + "." + bidder + ".");

    full_bid_t full_bid;
    full_bid.type = "bid";
    full_bid.price = price;
    full_bid.organisation = organisation;
    full_bid.bidder = bidder;

    std::string bidJSON = marshal_bid(&full_bid)
    put_state(bid_Key.c_str(), (uint8_t *)bidJSON.c_str(), bidJSON.size(), ctx)

    return "OK";
}
```

Figure 10: Bid chaincode

```
Result:  RevealedBid: {
  "objectType": "bid",
  "price": 200000,
  "org": "Organisation 1",
  "bidder": "x509::CN=bidder1,OU=client+OU=org1+OU=
}

Result: HiddenBid: {
   "organisation": "bid",
   "bidHash":"0b8bbdb96b1d252e71ac1ed71df3580f7a0e3
}
```

Figure 11: Bid response

A.3 C

The code snippet in Figure 12 demonstrates the key generation for enclave attestation. The hash generated by the enclave and the hash generated by the application need to match in order for the private chaincode to be invoked.

```
bool ra_info::mrenclave()
{
    try
    {
        enclave_priv_enc.Generate();
        enclave_pub_enc = enclave_priv_enc.GetPublicKey();
        enclave_priv_dec.Generate();
        enclave_pub_dec = enclave_priv_dec.GetPublicKey();
        chaincode_priv_dec.Generate();
        chaincode_pub_dec = chaincode_priv_dec.GetPublicKey();
        state_key = pdo::crypto::skenc::GenerateKey();

        std::string key;
        key = enclave_pub_enc.Serialize();
        LOG_DEBUG("mrenclave verification key: %s", key.c_str());
        key = get_enclave_id();
        LOG_DEBUG("enclave id: %s", key.c_str());
    }
    catch (...)
    {
        LOG_ERROR("Could not create hash key");
        return false;
    }

    return true;
}
```

Figure 12: Key generation for attestation

References

[1] Saharsh Agrawal. Enabling verifiable execution of distributed secure enclave platforms. 2021.

[2] Marcus Brandenburger and Christian Cachin. Challenges for combining smart contracts with trusted computing. 2018.

[3] Wikipedia contributors. Software guard extensions, 2022.

[4] IBM Corporation. Trusted hardware, 2014.

[5] Chunxiao Mu Dan Wang, Jindong Zhao. Research on blockchain-based e-bidding system. 2021.

[6] Brett Daniel. What is intel sgx?, 2021.

[7] Sven Bugiel Dhiman Chakraborty, Atul Anand Jha. Poster: Tgx: Secure sgx enclave management using tpm. 2019.

[8] Glorfindel. What are the functional similarities and differences between tpm and sgx in trusted computing?, 2021.

[9] Ilhan Gurel. The attestation challenges/gaps and cloud deployment., 2015.

[10] Hyperledger. Hyperledger fabric introduction, 2020.

[11] IBM. What are smart contracts on blockchain, 2020.

[12] Intel. Intel software guard extensions, 2021.

[13] Alessandro Sorniotti Marcus Brandenburger, Christian Cachin and Rudiger Kapitza. Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric. 2018.

[14] Alessandro Parisi. *Securing Blockchain Networks like Ethereum and Hyperledger Fabric*. Packt Publishing Ltd.

[15] Catherine Paulsen. Revisiting smart contract vulnerabilities in hyperledger fabric. 2021.

[16] Rado Stefanov. Enhancing the privacy and security of hyperledger fabric smart contracts using different encryption methods. 2021.

[17] WENBO SHI LEI WANG HONG LEI BANGDAO CHEN ZIJIAN BAO, QINGHAO WANG. When blockchain meets sgx: An overview, challenges, and open issues. 2020.