Spring 2023

Blockchain for Online Video Game Integrity

Philip Salire
San Jose State University

BLOCKCHAIN FOR ONLINE VIDEO GAME INTEGRITY

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Philip Salire

May 2023

The Designated Thesis Committee Approves the Thesis Titled

BLOCKCHAIN FOR ONLINE VIDEO GAME INTEGRITY

by

Philip Salire

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

May 2023

Gokay Saldamli, Ph.D.	Department of Computer Engineering
Younghee Park, Ph.D.	Department of Computer Engineering
Wencen Wu, Ph.D.	Department of Computer Engineering

ABSTRACT

BLOCKCHAIN FOR ONLINE VIDEO GAME INTEGRITY

by Philip Salire

With the growing level competition in video games, especially with regards to competitively played video games known as ”e-sports,” many players are searching for methods of gaining competitive advantages. As such, there is growing demand in software exploits of video games that aim to provide players unfair competitive advantages. Colloquially, these software exlpoits are referred to as “cheats” or “hacks.” Video game developers counteract these exploits by implementing “anti-cheat” technologies. Anti-cheats employ a myriad of complex methods across software, network, and hardware to detect and prevent cheats. They can be implemented both client-side and server-side with current research and implementations relying heavily client-side. This is an issue, however, as client-side implementations are open to inspection and alteration by malicious users looking to bypass the anti-cheat, who often succeed.

Integrity of players’ actions in online video games cannot be fully maintained with current client-side anti-cheat technologies. Blockchain, however, by design can ensure that integrity is maintained across an entire network. This project explores using blockchain as the core of a server-side anti-cheat implementation. With this method, each player is a member of the anti-cheat blockchain, ensuring integrity of player actions by validating player actions upon consensus.

ACKNOWLEDGMENTS

I would like to thank my advisor Professor Gokay Saldamli for his feedback and assistance throughout the work of this thesis. Additionally, I would like to thank the thesis committee and the whole MSSE faculty for making this thesis possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

The video game industry is rapidly growing, having reached billions of dollars in market cap worth since its inception just a few decades ago. “E-sports,” the competitive playing of video games, has grown heavily in popularity, rivaling traditional sports among younger generations. As with traditional sports, the e-sport industry faces challenges in maintaining competitive integrity. Malicious actors develop software exploits in competitive video games in effort to gain unfair competitive advantages. Often, these exploits are sold to the public for profit.

This has resulted in an active contention between cheat developers and video game developers. Cheat developers aim to reverse engineer video games, identify and develop exploits targeting vulnerable components for the purpose of gaining competitive advantages, i.e. cheats, and sell said exploits to unscrupulous players wanting to cheat. Cheats pose a business threat to the video game industry as they ruin the experience of the majority of players that play video games legitimately. Therefore, leaving cheats unpenalized is not an option for video game developers as it degrades the experience of the majority of the player base who does not cheat, potentially risking them to abandon the game altogether. In addition, a number of professional e-sports players have been caught using cheats, negatively impacting the reputation and overall trust of the e-sports industry. Thus, research in anti-cheat techniques has become an active topic in the e-sports industry. Video game developers are constantly developing new defenses against cheating in hope to stop cheating altogether.

Blockchain is a relatively new technology that provides an immutable, distributed ledger that is accessible to all parties in a network. In a blockchain network a record of transactions is maintained that, due to its inherent immutable design, is guaranteed to be free of tampering and not have any otherwise illegitimate data. It accomplishes this by what is known as consensus. Essentially, before updating the blockchain with new data the

majority of the nodes in the blockchain network must validate and agree on the new data to be stored. Through this, blockchain provides integrity of data across an entire network.

This research explores applying blockchain to online video games as a method to prevent cheating. Blockchain is most known for its application as a ledger of financial transactions, i.e. cryptocurrency. However, as data on a blockchain is constantly validated and immutable, thus guaranteed to not have been tampered with, it also can be applied as a general mechanism of data storage and validation. Particularly, it can be applied to video game data. Video game data can be stored on a blockchain with game specific consensus mechanisms to ensure that game data generated by players is validated for correctness to prevent illegitimate game state, i.e. cheating. This research investigates current methods of cheating in online video games, current mitigations against cheats, and provides a method of applying blockchain for cheat mitigation.

2 BACKGROUND

2.1 Online Video Game Cheats

The architectures for online video games typically follow the client-server model. For example, there exists the game client which players interact with and the game server which receives data from the game clients. From a threat modeling perspective, malicious actors may exploit potential vulnerabilities in any of these components. For example, application-level vulnerabilities in the game client, network vulnerabilities in the client-server network communications, or infrastructure-level vulnerabilities in the back-end server. Therefore, the exploitation of a game's client software, network communications, logic, or even external exploitation such as real-world collusion with the goal of granting a player artificially elevated in-game skill or status are referred to as "cheats." There are various methods of cheating in online games; Webb and Soh [1] proposed categorizing cheats as either game-level, application-level, protocol-level, or infrastructure-level.

Examples of the most popular forms of cheats are "aimbots," "lag switches," and "map hacks" [2], [3]. Respectively, this involves manipulating a player's mouse input to grant them perfect accuracy, tampering network packets to negatively affect network connections of opponents such as introducing latency or disconnecting them from the game, and tampering the game software to allow a player to see opponents through walls. More generally, cheats seek to grant unfair advantages to a player by creating fraudulent, falsified game states through exploitation. Within each cheat category, Webb and Soh [1] propose specific cheat types as summarized in the list below:

- Game level
  - "Bug" exploits in which bugs are leveraged for cheating purposes.
- Application level

  - "Information exposure" cheats in which cheaters leverage sensitive information revealed in game communications. For example, the game client may contain information about other players' positions which a cheater can expose by reading game memory i.e. "map hacks" as discussed in section 2.1.
  - "Invalid command" cheats in which a cheater may tamper a game client to send unexpected commands that allow unintended behavior.
  - "Bots/reflex enhancer" cheats in which a cheater tampers a game client to augment user input for unfair advantages e.g. "aimbots" as discussed in section 2.1.
- Protocol level
  - "Inconsistency cheats" in which cheaters may tamper with network packets for cheating purposes such as dropping packets or introducing delay to cause inconsistent game states.
  - Spoofing or replay cheats in which cheaters respectively masquerade as other players or replay network packets.
- Infrastructure level
  - "Information exposure" cheats in which cheaters leverage sensitive and tamper devices that the game relies. Most notably, a cheater may modify display drivers and augment the display with additional information.
  - "Proxy/reflex enhancer" cheats in which cheaters proxy game communications to augment sent game data to gain unfair advantages.

2.1.1 Game-level Cheats

Game-level cheats are exploits that target flawed game code or logic. For example, certain game input may inadvertently allow a player an unfair competitive advantage over other players whether through a bug or a game logic flaw. A cheater may abuse this bug or logic flaw to consistently gain an unfair advantage. It does not involve tampering with the game as is the case with other cheat types described in this section.

2.1.2 Application-level Cheats

Application-level cheats are software exploits that target flaws and security vulnerabilities in the game client's code and binaries. This is the main method in which cheats are developed.

For native applications, these level of cheats entail dynamically tampering with the game's running process in the operating system (OS) or statically tampering the game's static files that it relies on [4]. Application-level exploits crafted against a video game typically leverage two subsequent methods that are known as function hooking and DLL or shared library injection. Furthermore, binary-level exploitation can also be employed such as memory corruption, heap exploits, return oriented programming (ROP), kernel exploitation, etc.

Function hooking refers to the method in which an attacker patches a function in an executable's binary to jump to a different function upon being called and returning execution to the original function afterwards, or possibly jumping to another area in code to change execution flow altogether. From a software developer's perspective, function hooking is useful for debugging or modifying a program's functionality. However, from a cheat developer's perspective function hooking is useful for overriding game functions with arbitrary cheat functions. Fig. 1 and Fig. 2 illustrate respectively the execution flow of a normal function call and the execution flow of a function call that has been hooked.

Consider the following example C code which applies a typical approach for hooking a function in Windows:

Listing 1. Function hooking

```
//Get address of target function
void *targetFunction = (void*)GetProcAddress(
    GetModuleHandle("someDLL.dll"),
    "TargetFunction"
```

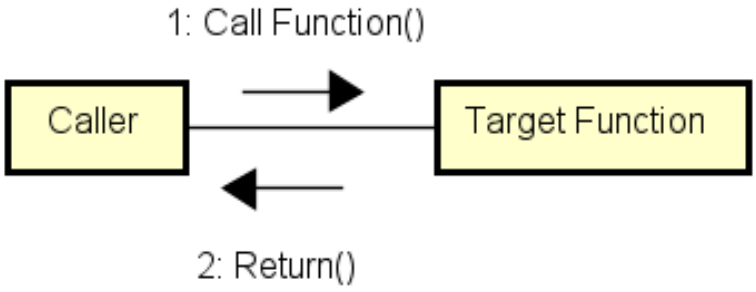

Fig. 1. Normal function call execution flow.

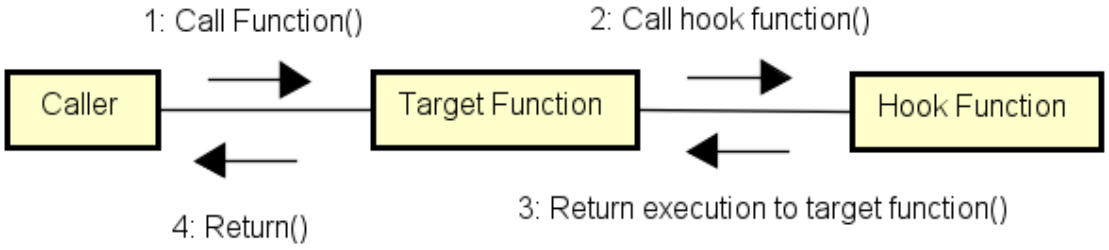

Fig. 2. Hooked function call execution flow.

```
);
//Prepare jump (jmp 0x0; ret)
unsigned char jumpBytes[5] = {
    0xE9,0x00,0x00,0x00,0x00,0xC3};
//Calculate relative address
DWORD relativeAddress = (
    (DWORD)hookFunction-(DWORD)targetFunction-5);
//Write address to jmp instruction
memcpy(&jumpBytes[1],
        &relativeAddress,
        4 //address is 4 bytes);
```

```
//Make sure memory is writable
//and save initial memory protection
DWORD oldProtect;
VirtualProtect(targetFunction,
               6,
               PAGE_EXECUTE_READWRITE,
               &oldProtect);
//Patch target function with jump
memcpy(targetFunction, jumpBytes, 5);
//Restore initial memory protection
VirtualProtect(targetFunction,
               6,
               oldProtect,
               &oldProtect);
```

In this code, a hook is configured to the function called `TargetFunction` located in `someDLL.dll`. The result is that whenever `TargetFunction` is called, the hook will immediately redirect execution to the user-defined hook function `hookFunction` via the `JMP` instruction (byte value `0xE9`) that was patched into `TargetFunction`. Note that this example assumes that `hookFunction` has already been injected into the process's memory space. Furthermore, after the `JMP` instruction a `RET` instruction (byte value `0xC3`) is also patched to the target function. This allows the program to return execution to the original `TargetFunction` after the hook function is called. Overall, this is a generalized approach to function hooking; Other methods exist that all accomplish the same goal which is to modify execution flow of function calls to redirect to user-defined hooked functions.

This allows a cheater to write arbitrary instructions to be executed via the hook function, making this a powerful method of cheat development. For example, a cheater may identify a function in a game's binary that allows a player to walk in the game and hook this function to alter its intended behavior such as causing the player to teleport, fly, etc. instead.

Function hooking is often paired with DLL or shared library injection in which the hooking mechanisms and hook functions are compiled into a DLL or shared library and then injected into the game's running process. Injection involves the following generalized steps:

1) Allocate memory in the target process
2) Load cheat DLL into the target process memory

If done successfully, the functions in the DLL or shared library will be loaded into the process and accessible by the rest of the program. In addition, it is often the case where code is executed upon the DLL or shared library being loaded, such as for patching the target functions to hook to the injected functions.

Furthermore, another client-side cheat approach may involve static files that game uses may be altered, most notably a cheater may target DLL files containing game functions and replace the DLL files with his or her own that contains cheats. This is often referred to as "DLL hijacking." Another method involves hijacking threads and modifying them with threads that the cheater controls.

Cheats may target hardware to accomplish cheating. An example of this is Direct Memory Access (DMA) cheats in which an external hardware device is connected to the cheater's PC that is capable of reading and writing arbitrary memory to the PC, i.e. for writing directly to a game process's memory [5], [6].

Similar methods are employed for web browser based games. In this case, rather than modifying executable binaries, client-side cheat development entails modifying client-side JavaScript.

2.1.3 Protocol-level Cheats

Protocol-level cheats are network exploits that target security vulnerabilities in the game's underlying network communication implementation. This may include exploiting fundamental network vulnerabilities such as replaying packets to replay in-game actions, dropping packets to avoid making certain in-game actions, or spoofing packets to masquerade as other players.

Games also often implement certain network algorithms to improve playability that may introduce additional vulnerabilities. One example is the "dead-reckoning" algorithm in which in the event of packets containing a player's in-game position is lost, the game may attempt to extrapolate position information from previous packets to calculate the expected position had the packet loss not occurred and update the player's position appropriately. A cheater may purposefully drop packets to trigger and abuse a vulnerable dead-reckoning implementation [1].

2.1.4 Infrastructure-level Cheats

Infrastructure-level cheats are exploits that target components that the game relies on. For example, an attacker may monitor network packets, and modify his or her display drivers to display extra information from the monitored network packets with information that is not intended to be visible. These cheats leave little to no trace as no actual game resources are tampered with. Instead, only third-party components that the game uses are tampered with. This type of cheat is an exploit of information exposure of unnecessary information.

2.2 Blockchain

Blockchain refers to the decentralized network technologies which provide users an immutable ledger of all transactions on the network. Due to its maintaining of an immutable ledger, it has seen widespread popularity in financial uses. Specifically it has seen success in its use in digital currencies that are referred to as "cryptocurrency." Bitcoin and Ethereum are examples of successful blockchains used as cryptocurrency.

2.2.1 Decentralized Applications

Outside of cryptocurrency, blockchain technologies have increasingly been used in decentralized applications or commonly referred to as "dApps." This is made possible by using what are known as "smart contracts" on the blockchain. Smart contracts are code compiled to bytecode and stored on the blockchain, which users can interact with by calling functions on the smart contract. DApps entail hosting applications on a blockchain rather than a centralized server as is the case in the traditional client-server network model. In the dApp architecture model, logic and data that is conventionally maintained by servers and databases is instead stored on and served from smart contracts.

3 RELATED WORKS

3.1 Cheat Detection

Han, Kwak, and Kim [7] categorize cheat detection as being either client-side, network-side, or server-side defenses. Each has strengths and weaknesses. As a result, game developers typically will implement cheat detection in each category, reaping the benefits of each. Table 1 illustrates different cheat types and the effectiveness of different anti-cheat architectures. Specifically, three categories of anti-cheats architectures were evaluated: client-side focused, server-side focused, or peer-to-peer (P2P) focused.

Table 1
Cheat categories and the effectiveness of different anti-cheat architectures [1].

Cheat	C/S	PB/VAC2	AS	SEA	RACS	P2P RC
Game Level						
Bug	o	x	o	o	o	o
Application Level						
Information Exposure	o	x	x	x	o	o
Invalid Commands	o	x	x	x	o	o
Bots/Reflex Enchancers	x	o	x	x	x	x
Protocol Level						
Inconsistency	o	x	o	o	o	o
Collusion	x	x	x	x	x	x
Spoofing, Replay	o	x	x	o	o	o
Infrastructure Level						
Information Exposure	o	o	x	x	o	o
Proxy/Reflex Enhancers	x	x	x	x	x	x

3.1.1 Client-side

Client-side cheats rely on tampering with the game software, the game's running process, or network communications. As such, client-side cheat detection relies on monitoring on the player's personal computer the game's software, running processes, and network communications for integrity and actively taking measures in preventing tampering. To prevent cheaters from inspecting static files and binaries, obfuscation of all distributed files is performed. This may include obfuscators that restructure code and rename variables to hinder code readability, using custom file and network packet formats

to prevent tampering and reading of files and network packets, or encrypting files to maintain integrity and confidentiality.

Detecting cheats resulting from tampered game software requires client-side anti-cheat that monitors running processes, memory, and the filesystem for modified states. This may even go as far as monitoring kernel-level information. However, this approach of monitoring player's personal computers has received criticism due to privacy concerns [2].

3.1.2 Network-side

Network packets can be monitored and analyzed for anomalous behavior i.e., anomaly detection. This is a type of statistical analysis that leverages the data generated by network packets. Specifically, normal execution of a game will fall within some range of behavior, which becomes evident as game data is collected over time. If some user's behavior falls out of this expected range of normal behavior, that user can be flagged as suspicious of cheating. Similarly, machine-learning techniques can be leveraged with the same data are often trained to classify whether cheating is occurring [8].

3.1.3 Server-side

Game servers collect logs regarding game state. This can include all information regarding players' actions in the game including raw user input like mouse movement or key presses. Like network-side cheat detection, all the game data can be analyzed by methods such as data mining, machine learning, or statistics to detect cheating. This typically relies on anomaly detection. That is, normal game playing is expected to fall within certain patterns and ranges; if a player consistently plays outside of expected patterns and ranges, it may indicate cheating. Other methods include validating user actions against expected actions i.e., client-side emulation on the server [4]. This entails calculating normal game execution with the given parameters and comparing it with the received execution. Strong server-side cheat detection is important as it is the only method that is not open to tampering or inspection by cheaters.

3.1.4 Combined Methods

Client-side cheat detection is effective as it allows detection at the cheating source but cannot be considered a complete defense because since it is deployed on the client, cheaters have full control to inspect and develop bypasses to detection measures. Server-side cheat detection addresses this limitation as cheaters cannot inspect or tamper with server code. However, server-side cheat detections have drawbacks in that it there is a risk that it may impact the business critical server's performance for processing actual gameplay by consuming excessive resources for cheat detection, it introduces more components that developers must maintain and thus more expenses, and they are generally limited to analyzing player data and logged action. Therefore, combining all types of anti-cheat — server-side, network-side, and client-side — can reap the best aspects of all and mitigate the weaknesses of each [9].

For example, Kaiser, Feng, and Schluessler [4] proposed anomaly-based cheat detection that works by validating game execution against expected game execution obtained via emulation of the game on the server. First, the game is loaded and analyzed on the server for dynamic and static properties regarding files, memory usage patterns, function usage patterns, among other factors. As a result, the server learns how to efficiently emulate dynamic game data. The server receives dynamic, untrusted game execution data from the client, which it then validates against the emulated, trusted game execution. This is an effective approach as it encapsulates both client-side and server-side cheat detection methods collaboratively.

3.1.5 Peer-to-peer

Most games have relied on the client-server network model, which has inherent limitations in both performance and its effectiveness in cheat detection. Performance is limited as having a central server serves as a single point of failure, as well as making it susceptible to denial-of-service attacks. In addition, cheat detection is adversely affected

as clients have full reign over the data communicated to the server. This can be addressed with P2P network architectures.

Webb and Soh [1] propose how P2P game architectures can allow effective cheat detection. With P2P games, peers can work together to validate game state. That is, since the majority of players can be trusted to be non-cheaters barring collusion, all players i.e. peers can perform individual validation of game state and vote to reach an authoritative majority consensus on the actual game state. Another method includes having peers take turns to make actions, which can prevent many network-side cheats such as replay attacks. Additionally, P2P can also be combined with client-server architectures. For example, dividing a game into many smaller P2P networks, each with a "referee" that validates game state, similar to how a server validates game state in the client-server model.

3.2 Blockchain in Games

Blockchain is a relatively new technology that has seen wide success in the digital currency market. Recently, blockchain has been expanding into other industries. This includes the videogame industry where games are recently seeing development with blockchain technology rather than traditional client-server technologies. Blockchain has attracted developers to push to integrate them into their applications, or replace existing components altogether, as it can effectively replace the client-server architecture that has inherent issues in performance, maintainability, and user data privacy [10]–[14]. Rather than a server, a blockchain-based game leverages smart contracts to implement game logic and stores game data and state on the blockchain [15], [16].

Blockchain games began to emerge in popularity in 2018 and have shown continued growth over time [17]. Being a relatively new technology, however, consequently means novel security vulnerabilities are introduced and have been identified in a number of blockchain games [18], [19].

Patel et. al [20] use blockchain alongside the client-server model by using Ethereum smart contracts to maintain game state among players, IPFS to maintain player data, and servers to handle user authentication and pushing game updates.

3.2.1 Blockchain Cheat Detection

Inherently, blockchain games have strong security due to its design of guaranteeing data integrity and immutability. This and blockchain's decentralized design, which removes the necessity of a central server, have been catalysts for its increased usage in applications.

Blockchain being used solely for cheat detection and prevention has also seen some adoption. Kalra, Sanghi, and Dhawan [21] propose an architecture for using blockchain for real-time cheat prevention in online games. In this method an existing game client is minimally modified to send and receive game data respectively to and from a Hyperledger Fabric blockchain. This modification involves automatic instantiation of blockchains for each session in the game, which includes enabling the client to perform peer discovery and generating genesis blocks and smart contracts. This approach requires the developer to predefine a set of rules which the application will read and automatically generate and deploy smart contracts from. Kalra proposes anti-cheat via peer consensus; game state is validated via peer consensus which consequently preventing cheats, considering that cheats are essentially anomalies in individual nodes that cannot possibly reach consensus outside of collusion. This approach does not require an entire game to be developed on a blockchain as is the case with blockchain games but is implemented independently to be integrated into existing games.

4 SYSTEM DESIGN

4.1 Design Considerations

4.1.1 Blockchain

Blockchain is essentially a distributed network that stores data on a ledger and by design guarantees its data to be valid and immutable. This is particularly useful in applications that require data storage, data validation, data immutability, and data logging. Similarly, the responsibilities of a server-side anti-cheat can be summarized as follows:

- Validate player input for expected values and behavior.
- Validate game state for expected values and behavior.
- Validate player data for expected values.
- Log player and game data for future reference.

All player inputs and data are validated for expected values and behavior. In the most simple case, this may involve validating that a player does not hold any unexpected data which might easily be checked by conditional if-statements. In more complex cases as discussed in section 3, analyzing player behavior can be done such as with statistical or machine-learning models.

It is also important to log all relevant game data for future reference. For example, if a cheat is not detected by the anti-cheat, game administrators may be able to reference game logs to manually determine if a player is cheating, or data may be aggregated and used to train a machine-learning model for cheat detection.

Blockchain fulfills all these use cases with its data validation mechanisms and maintaining of a ledger of all past records. The only requirement from the specific blockchain technology to be used is that it must support handling complex logic that an anti-cheat entails. To this end, any blockchain that supports smart contracts fulfills this requirement. This research explores the feasibility of applying blockchain in a server-side anti-cheat implementation.

4.1.2 Design principles

Despite the advances in decentralized blockchain games as discussed in section 3.2, the majority of online video games still rely on the centralized client-server model. For an anti-cheat to be feasible for wide spread adoption in modern online video games, it must be modular and act as an extension to and not a replacement to existing client-server game infrastructure to allow seamless adoption. If an anti-cheat implementation requires significant changes to be implemented into a game's existing infrastructure, the trade-off between the cost to integrate and overall benefit may not be worthwhile. Integrating the decentralized architecture of a blockchain with the centralized architecture of the client-server model poses a challenge as each rely on very different technologies.

With this in consideration, the overall design decisions of the anti-cheat implementation in both the code and system design were chosen to follow the object-oriented design principles known as "SOLID," which is an acronym that is defined as follows:

- Single-responsibility principle which states that each component or class in a system should have only one responsibility.
- Open-closed principle which states that components or classes should be extendable but not modifiable.
- Liskov substitution principle which states that sub-types of a component or class should be substitutable for each other.
- Interface segregation principle which states that interfaces of components or classes should be only include methods that are used by the implementing component or class.
- Dependency inversion principle which states that components or classes should depend on abstractions not concretions.

The “SOLID” principles ensure that software code and systems are developed in a modular manner as to allow efficient maintainability in development. As a result, it is followed by many software developers and likewise was chosen to be followed for this anti-cheat implementation. This research provides an anti-cheat implementation based on blockchain which is able to run alongside any existing game infrastructure by acting as an extension rather than replacement of existing infrastructure.

4.1.3 Latency

Online video games that respond to player input in real-time must have low latency for optimal player experience. It is necessary that any back-end software and architecture respond quickly as to not introduce excessive latency. For instance, if a player presses a button in the game it is expected that the game respond immediately to that button press, not several seconds later. Therefore, an anti-cheat implementation must not interfere with or negatively impact actual game communications. Blockchain, however, is not optimized for fast response times. In fact, response times can be sporadic depending on the overall usage of the blockchain due to its consensus mechanisms.

To address this, the blockchain anti-cheat implementation was chosen to operate asynchronously to game communications. This asynchronous design enables the anti-cheat to consume relevant game data with minimal to no impact on the time to respond to player requests. The time for the anti-cheat to generate results will not impede other game communications as it will be done asynchronously; the game is not required to wait for a response from the anti-cheat as is the case in a synchronous design. Rather, once a transaction is sent to the anti-cheat a response will be sent once the transaction is complete, allowing the game to continue execution while the anti-cheat operates asynchronously. This, however, has a drawback in that cheat detection is not performed in real-time; There inherently will be a delay from when a player cheats and when it is detected. The time for a cheat to be detected, however, is not an important metric; the

important metric to consider the accuracy of cheat detection. As long as a cheater is eventually detected with high accuracy, then the anti-cheat can be deemed effective. In addition, it is important that both false negatives and false positives be minimized. Legitimate players should not be wrongly penalized.

4.1.4 Blockchain facade

In the anti-cheat blockchain implementation of Kalra et. al [21], it is required to introduce new dependencies that allow communication with the blockchain. However, this approach has several drawbacks from a development perspecive. Blockchain technology is relatively new compared to client-server technologies and thus may lack mature libraries that ease integration to existing code infrastructure. For example, perhaps the most widely adopted blockchain library for clients is the "web3.js" [22]. However, "web3.js" is specfic only to Ethereum and only offered in JavaScript. This means that if the anti-cheat implementation were to use "web3.js," only Ethereum and JavaScript would be supported. In addition, it requires additional dependencies to the code that introduce more complexity. Requiring a specific library and blockchain technology to implement the anti-cheat would be a violation of the dependency inversion principle of "SOLID," which states that components should rely on abstractions and not on concretions. To alleviate this and provide seamless integration into existing game infrastructure, a facade to the blockchain can be provided in the form of an API.

4.1.5 Network protocol

The network protocol to be used for this anti-cheat API must likewise be widely supported to allow seamless integration. For this reason, WebSockets and HTTP were chosen for the network protocols used by the API. HTTP and WebSockets are virtually universally supported protocols as both are IETF standardized protocols that the world wide web rely heavily on. It is important, however, to consider that HTTP traffic is not optimized for real-time communications so latency will be expected to be high.

WebSockets in contrast are light-weight and consequently more optimal for real-time communications and extensively used in web browser based online video games. Still, the WebSockets protocol is not completely optimal for real-time online games as it runs on top of the lower level TCP protocol. TCP is inherently less performant due to its implementation requiring ordered, loss correcting, and error correcting communication. The other network protocol to consider is UDP which is also widely used in game development due it to being a light-weight protocol that is optimal for real-time, low latency communications. Unlike TCP, UDP has no requirements in guaranteeing order, loss correction, or error correction and is consequently a much faster protocol. Therefore, UDP is the optimal protocol for online games and is widely used in the real world. However, UDP is not supported in web browsers and can only be utilized in native application based games, eliminating support for web browser based games. The WebSockets protocol in contrast is widely supported both inside and outside of the web browser and thus chosen for this research implementation to allow widespread support.

4.1.6 Smart contracts

Complex logic can be implemented in a blockchain with smart contracts as discussed in section 2.2.1. From a high-level overview, an online game can be divided into two components: players and sessions. Therefore at a minimum, the anti-cheat's blockchain must have smart contracts defined for players and sessions.

There are two options to consider for the method of deployment of the smart contracts:

- On the public blockchain. This provides the benefit in that game developers do not have to maintain the anti-cheat infrastructure; it is fully decentralized on the public blockchain. Most public blockchains are reliable and can be trusted for resiliency against availabliity issues. In addition, a public blockchain allows full transparency of game data, which may or may not be desired dependening the desired confidentiality of the data. The major drawbacks of a public blockhain is that it will be very costly

as it requires real cryptocurrency to make transactions and that transaction times may be slow due to necessitating competition with all other transactions.

- On a private blockchain. This provides the benefit in that transaction times will be minimized as the game is the only blockchain user and no actual cryptocurrency is required which saves on costs. The drawback of a private blockchain is that the game developers will have to host and maintain it.

4.1.7 Benefits of this implementation

An additional benefit of using an API as a facade to the blockchain, rather than direct communication from the game server to the blockchain such was done by Kalra et al. [21], is increased modularity of the blockchain component. That is, any blockchain technology can be used behind the API facade if implemented correctly. In terms of the "SOLID" principles, this satisfies the Liskov substitution principle which states that sub-types of a component should be substitutable of each other and the dependency and the dependency inversion principle which states that components should rely on abstractions and not on concretions. Furthermore, a conventional server-side anti-cheat implementation may require an intertwining of code that handles game logic and code that performs cheat detection. This is a violation of the "SOLID" single-responsibility principle, which states that each component should have only one responsibility, and thus not ideal as it hinders the efficiency of development due to the decrease in modularity. Instead, the code that handles game logic and the code that performs cheat detection should be separated into different components to separate responsibilities. Implementing an anti-cheat as its own component in the game's back-end infrastructure, rather than as a part of the game server code, effectively accomplishes this.

4.1.8 Components

With these factors considered, the following components and their responsibilities were determined:

- “Players” which send inputs to the game server.
- “Game server” which processes player input and maintains game state.
- “Anti-cheat server” which provides the anti-cheat facade API to maintain anti-cheat rules and communicate information on cheat detections.
- “Blockchain” which validates game state according to the anti-cheat rules.

4.2 System Architecture

The resulting architecture of the blockchain anti-cheat implementation is illustrated in the component diagram in Fig. 3. Here, the multitude of players communicate with the game server to update and receive game state. The game server communicates with the anti-cheat API server to forward the game state and receive information about any cheat detections. Lastly, the anti-cheat API communicates with the blockchain to update its data and receive updates on players that were detected to be cheating via smart contract events.

Within the blockchain, several smart contracts are involved as illustrated in the class diagram in in Fig. 4. There exists a single “Anticheat” smart contract, which owns many “Player” and “Session” smart contracts. The “Anticheat” smart contract handles all of the creation of the “Session” and “Player” contracts. The “Session” smart contracts contain game data on individual game sessions and maintain game session specific data on its associated “Player” contracts. The “Player” smart contracts contain game data on individual players. Both the “Session” and “Player” contracts extend the “DataHandler” contract which is responsible for storing and updating game data.

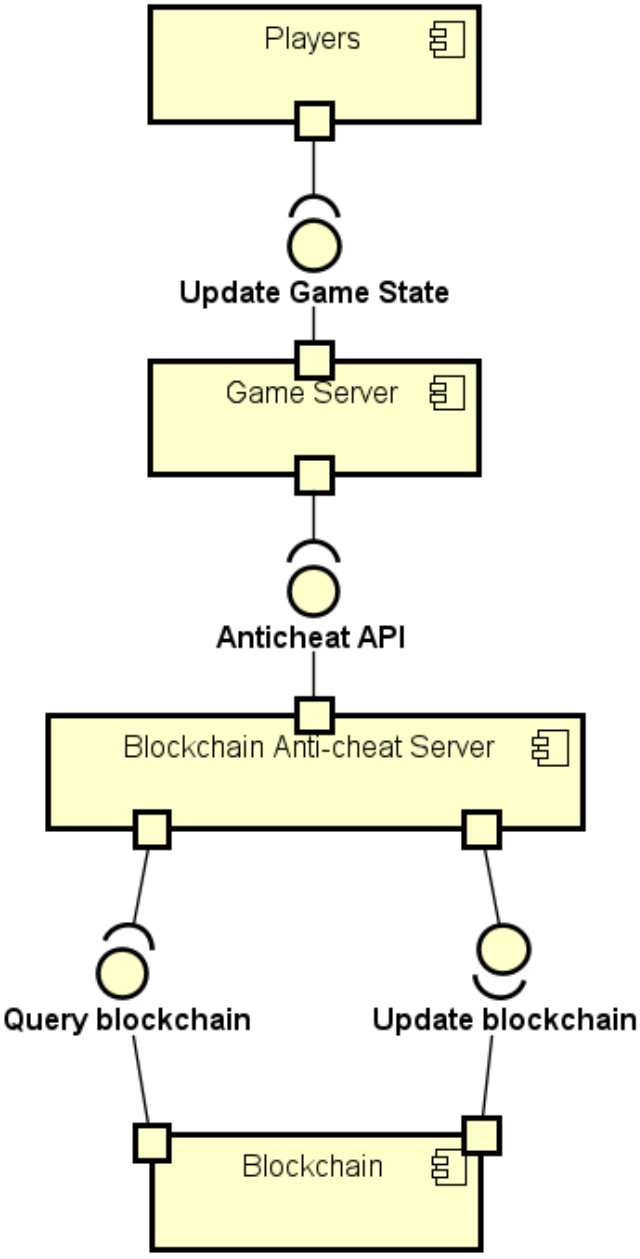

Fig. 3. Blockchain anti-cheat component diagram.

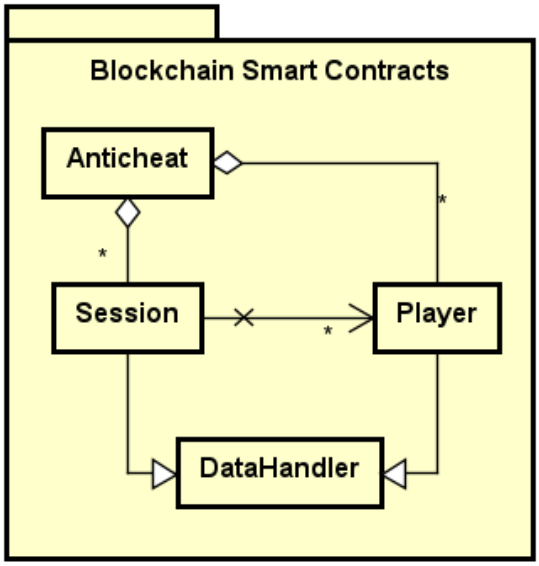

Fig. 4. Blockchain smart contract class diagram.

5 IMPLEMENTATION AND EVALUATION

5.1 Blockchain

Ethereum was chosen as the blockchain technology due to its widespread usage in blockchain applications and its support for smart contracts. To create a private Ethereum blockchain, an Ethereum emulator called “Ganache” was used as it provides a fast, easy way to locally host a private Ethereum blockchain. Following this, the anti-cheat was tested on the public Ethereum Sepolia testnet.

5.2 Anti-cheat API

The anti-cheat API was developed and hosted in Python using the following libraries:

- “FastAPI” which is a framework for developing web APIs.
- “Pydantic” which is a data validator library.
- “web3.py” which is a library for interacting with the Ethereum blockchain.
- “uvicorn” which is an Asynchronous Server Gateway Interface (ASGI) webserver.

The Python programming language with the FastAPI library were both chosen for their widespread support and ease of development. The API was developed with a RESTful architecture. In the API, it is important that user input be validated for expected formats and values to prevent undefined and potentially insecure behavior resulting from untrusted user input. Therefore, the Pydantic library was used for defining and enforcing API schemas, which acts as an input validation mechanism. The API is hosted with the uvicorn ASGI webserver, mainly due to FastAPI’s built-in support for it.

5.2.1 Rule-based system

As discussed in Section 4 the anti-cheat API acts as a facade to the blockchain to facilitate widespread support for existing server architectures. Furthermore, it is also necessary to allow the anti-cheat to be easily customizable so that it can be used to defend any possible game. To allow this, a rule based system was developed in which the server

must define specific conditions that game data is expected to conform to. The intention of this is that if any of these rules or conditions are not met, it can be categorized as cheating. This allows the added benefit in there are no restrictions as to when anti-cheat rules can be defined; they can be defined before, during, or after a game session is running.

As illustrated in Fig. 4 and Section 4, the anti-cheat blockchain contains a single "Anticheat" smart contract which handles the logic of maintaining a multitude of "Session" and "Player" smart contracts. The server must interact with the "Anticheat" smart contract to create new "Session" and "Player" smart contracts. These contracts maintain session and player data and as such are expected to have one-to-one mappings to their server-side equivalents. The "DataHandler" smart contract that both "Session" and "Player" extend contain functions on maintaining data and implementing anti-cheat rules. Therefore, it is possible set anti-cheat rules both session-wide and per individual player.

5.2.2 API Specification

The anti-cheat API supports "CRUD" operations (creating, reading, updating, and deleting) of rules to enforce by the anti-cheat. For the request and response schemas, the JSON format is used. Both HTTP and WebSocket network protocols are supported.

5.2.2.1 HTTP: The following HTTP API endpoints and their purposes were defined as follows:

- `POST /session/{session_id}` - create a game session.
- `POST /session` - create a game session with a random ID.
- `GET /session/{session_id}` - retrieve the game session address.
- `POST /session/{session_id}/player/{player_id}` - create or update a player in a session.
- `POST /session/{session_id}/player` - create or update a player in a session with a random ID.

- `GET /session/{session_id}/player/{player_id}` - retrieve the player address in a session.
- `PUT /session/{session_id}/data/{data_type}/{key}` - create or update player data in a session.
- `GET /session/{session_id}/data/{data_type}/{key}` - retrieve session data in a session.
- `PUT /session/{session_id}/rule/{data_type}/{key}` - create or update an anti-cheat rule in a session.
- `GET /session/{session_id}/rule/{data_type}/{key}` - retrieve an anti-cheat rule in a session.
- `PUT /session/{session_id}/data/{data_type}/{key}/validate` - update session data in a session while validating it against the anti-cheat rules.
- `PUT /session/{session_id}/player/{player_id}/data/{data_type}/{key}/validate` - update player data in a session while validating it against the anti-cheat rules.
- `GET /session/{session_id}/player/{player_id}/data/{data_type}/{key}` - retrieve player data in a session.

Session and player data is sent with the following schema:

Listing 2. Session and player data schema

```
{
    ''key'': ''key'',
    ''data'': [0,1,2,...]
}
```

The “key” field contains a unique string to associate “data” with, i.e. a key-value mapping. The “data” field is an integer value. Integers were chosen to represent all possible data types as they can also indirectly represent booleans (with 1 or 0) or floats (when rounded). They can also represent strings if the string values are expected such as the case in an Enum type.

5.2.2.2 Anti-cheat rule schema: The anti-cheat rule system follows the following schema:

Listing 3. Anti-cheat rule schema

```
{
    ''key'': ''key'',
    ''data'': 0,
    ''operand'': ''eq''|''ne''|''lt''|''gt''|''lte''|''gte''
}
```

The “key” field contains the same “key” value of the data that the rule is intended to be attached to. The “data” field contains the value that input data will be compared to by the “operand” value. If any input data does causes the condition that is defined by the “data” and “operand” values to be fulfilled, that input will be flagged as cheating. The possible “operand” values are defined as follows:

- “eq” - equals
- “ne” - not equals
- “lt” - less than
- “gt” - greater than
- “lte” - less than or equal to
- “gte” - greater than or equal to

As mentioned, rules are defined either for a session or a single player, and within a session or a player, rules are mapped to key values. Each key can have an indefinite

amount of rules attached to it. Therefore, multiple rules can be attached to a single key to create complex rules.

5.2.2.3 WebSockets: HTTP, however, is not optimal for real-time communications. To this end, a WebSocket endpoint is also provided that offers the same operations. The WebSocket requests use JSON with the following schema:

Listing 4. WebSocket Request Schema

```
{
    ''action'': ''action_string'',
    ''msg'': { ... }
}
```

The fields of the request schema are defined as follows:

- The `action` field determines which operation to perform.
- The `msg` field contains the same request data that is used in the relevant HTTP request.

5.2.2.4 API Responses: All JSON responses follow the JSend response specification:

Listing 5. JSend Response Specification

```
{
    ''status'': ''success''|''fail'',
    ''error'': ''error_message''|null,
    ''data'': { ... }
}
```

The fields of the JSend response schema are defined as follows:

- The `status` field designates whether the request succeeded or failed.

- The `error` field contains the error message string if the request failed, or null or fully omitted if the request was successful.
- The `data` field contains the same relevant response data used in the HTTP responses.

5.3 Implementation

5.3.1 Online Game

Open-source online games were used to test the effectiveness of the anti-cheat. First, it was necessary to develop cheats for these games. Then, the games were extended to communicate with the anti-cheat API to detect cheats and take action when detected.

"example-.io-game" [23] is a simple game that includes only two in-game objects: players and bullets. The game is played on a two-dimensional plane with players controlling movement through their mouse. As players move, bullets are emitted from the players, which if collide with other players decreases the collided players' health. The game relies on the "Socket.io" library, which is built on top of the WebSocket protocol, to handle client-server communications. The server maintains game state, which is then repeatedly broadcasted to all players over "Socket.io".

In its original state the "example-.io-game" game had no apparent vulnerabilities that could be exploited for cheating. Therefore, the game was extended to introduce several exploitable vulnerabilities for anti-cheat testing purposes. The following features were added to the game:

- Ammo with a limited capacity of thirty rounds
- Speed boosts with a limited usage of three times
- Walls were modified to end a player's game upon collision

All three features were intentionally designed to be vulnerable to client-side only validation. That is, the input validation for each new feature is not implemented on the server but only on the client. As a result, a cheater may exploit these vulnerabilities by

Fig. 5. Screenshot of “example-.io-game.”

modifying the game client to remove the client-side validation to gain infinite ammo, speed boosts, or bypass the loss condition that occurs upon wall collisions.

5.3.2 Execution Sequences

To have the game interact with the anti-cheat it was necessary to extend the code to communicate with the anti-cheat API. The sequence used to configure the game with the anti-cheat is illustrated in Fig. 6. To summarize:

1) Game server makes an API request to create a game session.
2) API server makes a transaction and calls the smart contract function to create the game session.
3) Game server polls the API until the session creation transaction is completed.
4) Game server makes API requests to create anti-cheat rules.
5) API sever makes a transaction and calls the smart contract function to create an anti-cheat rule for each rule.

6) Game server polls the API until the rules creation transactions are completed.
7) Game server is ready.

The sequence used to connect a player to the game and anti-cheat is illustrated in Fig. 7. To summarize:

1) Player connects to the game server.
2) Game server makes an API request to add a player to a session.
3) API server makes a transaction and calls the smart contract function to add a player to a session.
4) Game server polls API until the player creation transaction is completed.
5) Game server adds player to memory.

The sequence used to update player data to the anti-cheat is illustrated in Fig. 8. To summarize:

1) Player sends game data to the game server over WebSockets.
2) Game server makes an API request to validate the player data.
3) API server makes a transaction and calls the smart contract function to valdidate the player data.
4) Smart contract function checks the player data against each rule it has.
5) If a validation rule fails:
    a) Smart contract removes the player from the smart contract and revert the transaction.
    b) API server upon receiving a revert sends a response over WebSockets that a player was flagged for cheating.
    c) Game server receives player that was flagged and removes the player from the game.

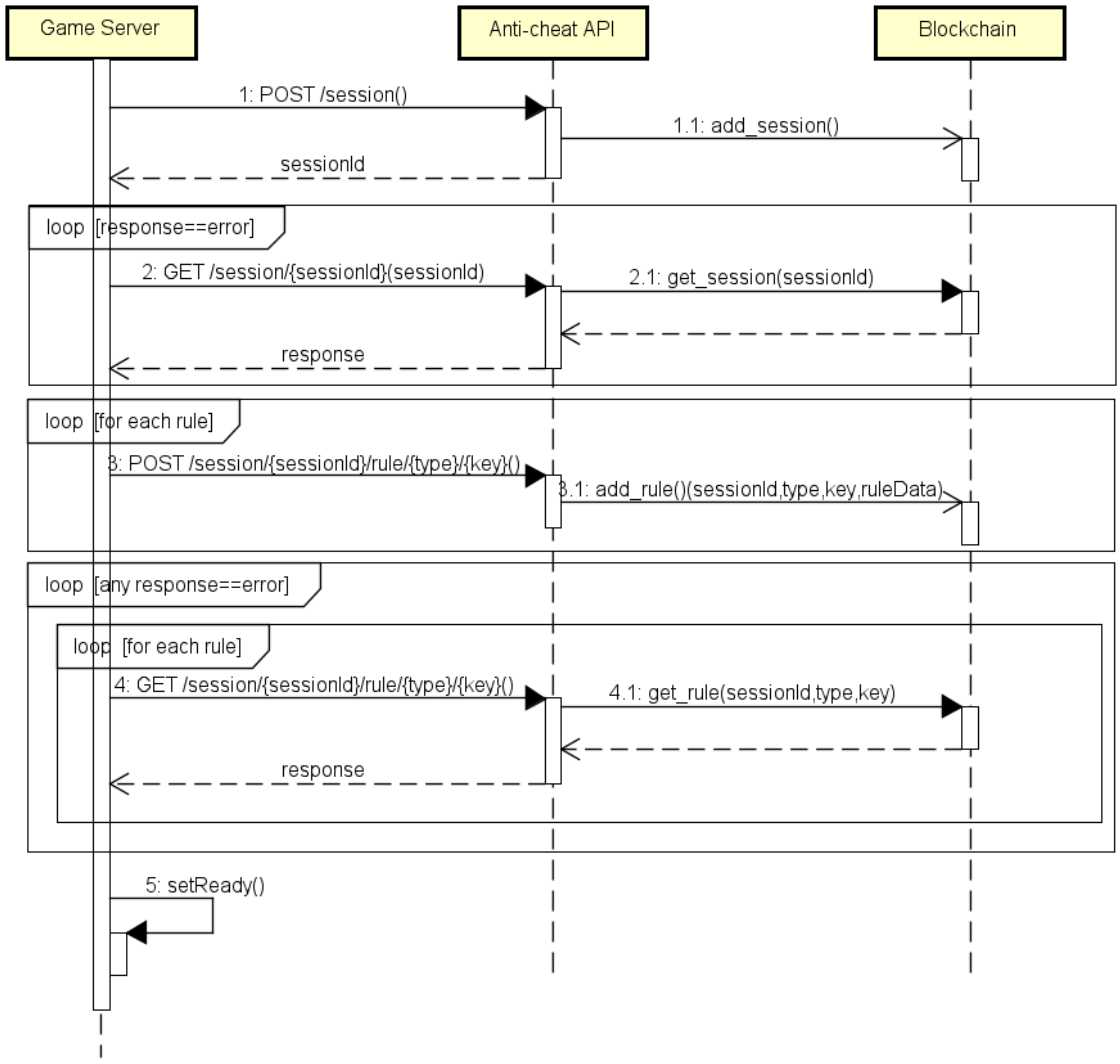

Fig. 6. Sequence diagram of setting up an example-game-io game session.

5.4 Evaluation

The blockchain anti-cheat implementation was effective in preventing most types of cheats. Webb et. al [1] compared different anti-cheat architectures and their effectiveness. In terms of this categorization, this research implementation performs as expected from an server-side anti-cheat. That is, cheats that can be encapsulated in player data and network communications are mitigated, while cheats that occur client-side only such as

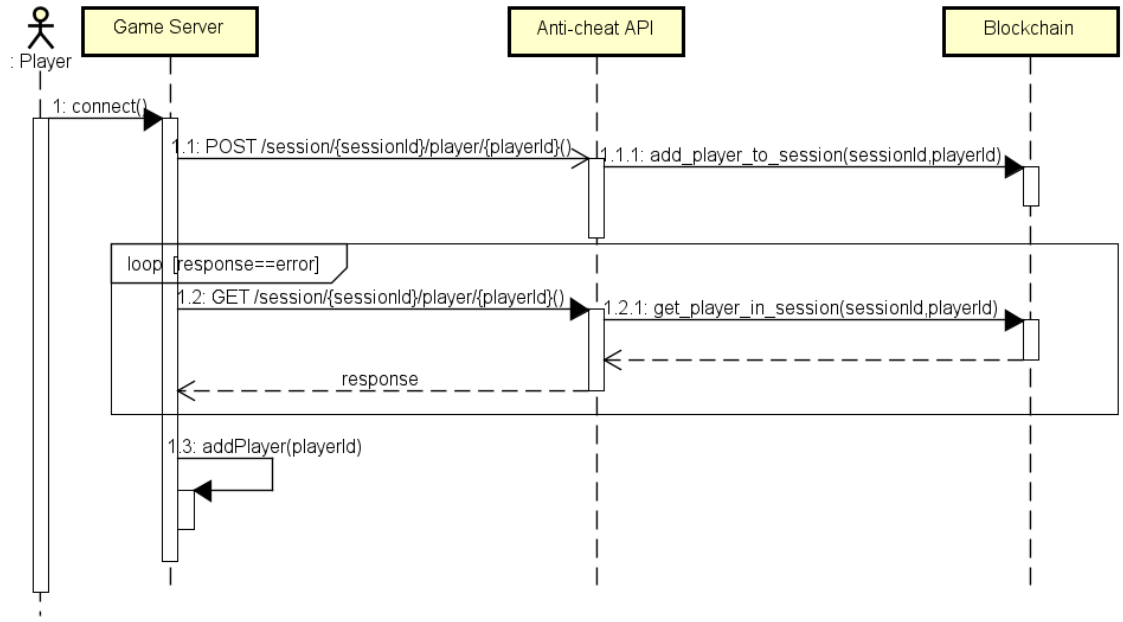

Fig. 7. Sequence diagram of a player connecting to an example-game-io game session.

reflex enhancers are not mitigated. Client-side only cheats require client-side anti-cheats such as PunkBuster (PB) or Valve Anti-cheat (VAC2) [1]. Table 1 was extended with this research anti-cheat implementation to illustrate its effectiveness compared to other past implementations in Table 2.

With regards to the performance, as discussed in Section 4.1.3 it is worth considering the time that the anti-cheat takes to detect a cheat. Due to blockchain transaction times, there exists an inherent and significant duration of time between the act of cheating and the cheating being detected. Furthermore, it was also evident that introducing more players in a single game session introduced further increased cheat detection time which is expected due to the increased amount of transactions that are needed to be processed. To address this, it was found that batching multiple points of data in a single transaction improves performance. Table 2 and Fig. 9 summarize these evaluations.

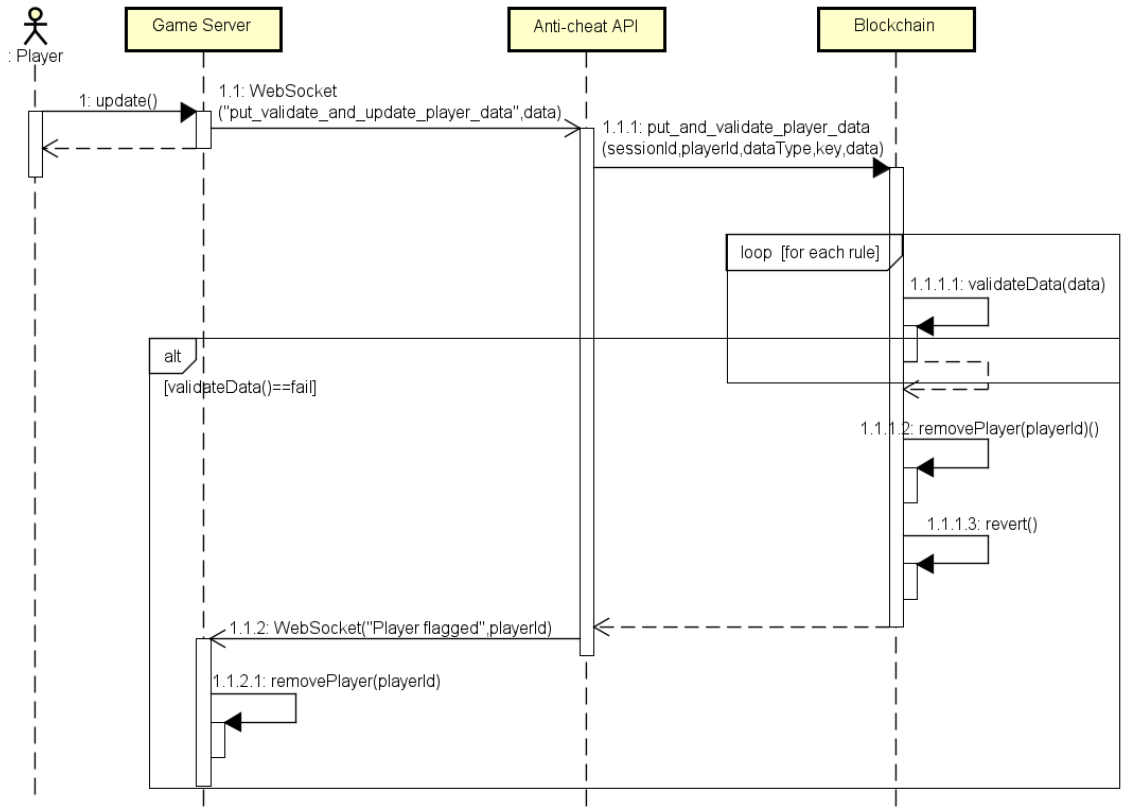

Fig. 8. Sequence diagram of rule-based data validation.

On the public Sepolia Ethereum testnet, transaction times were significantly greater than the private blockchain as expected. To address this, gas prices were experimented with in attempt to prioritize our transactions. However despite increasing gas prices, transaction times were still excessively long for it to be considered feasible. In addition, the transaction expenses needed to operate on on a public Ethereum blockchain is not practical as real cryptocurrency would need to be obtained and spent. Ultimately, using a public blockchain was not considered a viable option after this evaluation.

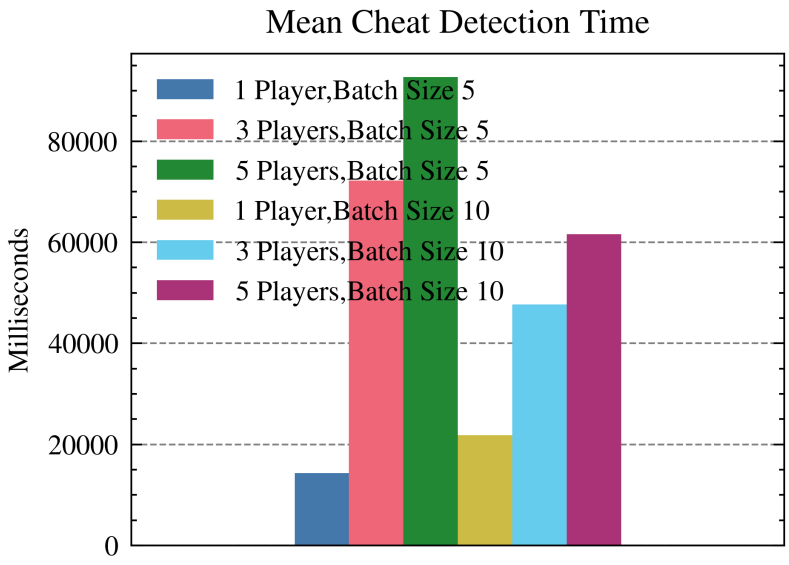

Fig. 9. Mean time differences from cheat occurrence and cheat detection.

Table 2
Mean times to detect cheating

Batch Size	Player count	Mean Time (ms)
5	1	14333.2
5	3	72213.4
5	5	92680.3
10	1	21858.6
10	3	47688.8
10	5	61606.9

Table 3
Different types of cheats mitigated in each architecture, derived from Webb et. al [1].

Cheat	Thesis	C/S	PB/VAC2	AS	SEA	RACS	P2P RC
Game Level							
Bug	o	o	x	o	o	o	o
Application Level							
Information Exposure	o	o	x	x	x	o	o
Invalid Commands	o	o	x	x	x	o	o
Bots/Reflex Enchancers	x	x	o	x	x	x	x
Protocol Level							
Inconsistency	o	o	x	o	o	o	o
Collusion	x	x	x	x	x	x	x
Spoofing, Replay	o	o	x	x	o	o	o
Infrastructure Level							
Information Exposure	o	o	o	x	x	o	o
Proxy/Reflex Enhancers	x	x	x	x	x	x	x

6 FUTURE WORK

The anti-cheat was developed with the Ethereum blockchain due to its support for complex logic with smart contracts. However, Ethereum is not optimal for real-time communication as is necessitated by online games. This is evident in the measured transaction times which limit the time to detect a cheat after it has occurred as discussed in Section 5.4. To this end, it is worth exploring other blockchain technologies especially with regards to comparing the performance differences of each. Solana blockchain may be more feasible than Ethereum as it boasts transaction times of less than a second. Furthermore, due to the inherent delays caused by blockchain transaction times, further evaluation may also include also comparing a blockchain's performance to a typical server and database implementation performance.

Currently, the anti-cheat works using a rule-based system in which conditions that data is expected to conform to must be defined through the API, as discussed in Section 5. More complex rules can be defined by assigning multiple rules to a single key or defining rules on-the-fly. However, it is not possible to define logic-based rules in this system. It is worth exploring methods of developing a system to define logically complex rules.

As discussed in Section 4, a benefit of implementing an anti-cheat with blockchain is that it maintains a ledger of all past transactions. In the perspective of online games, it maintains game logs that can be reviewed and evaluated for cheating. It is worth developing an interface to easily extract game logs from the blockchain. From there, the collected data can be evaluated and possibly used to train a machine-learning model for cheat detection.

Lastly, this implementation used "Ganache," an Ethereum emulator, rather than an actual Ethereum blockchain. While it was found that deploying to the public Etherum testnet Sepolia was not practical due to the relatively high transaction times, it is worth experimenting with an actual private blockchain rather than an emulated one.

7 CONCLUSIONS

Overall, it was found that blockchain is a viable option for an anti-cheat implementation. Blockchain is a very different architecture than the traditional client-server architecture so it is important to facilitate its integration in existing client-server applications or else it may be too difficult and thus not worthwhile to use. To this end, the implementation in this research abstracts the blockchain with an API that is hosted on a conventional server that supports both HTTP and WebSockets. Furthermore, to allow the anti-cheat to be applied to any game, a rule system was developed in which arbitrary conditions i.e. rules that game data must fulfill can be defined. These rules are stored in smart contracts. Then, whenever data is updated it must first satisfy all of the defined rules, or else the responsible player is flagged as cheating. With both the API facade and the rule based system, this implementation is very modular and can be integrated in a wide variety of possible existing games and game infrastructures.

Literature Cited

[1] S. Webb and S. Soh, "A survey on network game cheats and p2p solutions," *Australian Journal of Intelligent Information Processing Systems*, vol. 9, no. 4, pp. 34–43, 2008.

[2] A. Maario, V. K. Shukla, A. Ambikapathy, and P. Sharma, "Redefining the risks of kernel-level anti-cheat in online gaming," in *2021 8th International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 676–680, 2021.

[3] D. Liu, X. Gao, H. Wang, and A. Stavrou, "Detecting passive cheats in online games via performance-skillfulness inconsistency," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 615–626, 2017.

[4] E. Kaiser, W. Feng, and T. Schluessler, "Fides: remote anomaly-based cheat detection using client emulation," in *Proceedings of the 16th ACM conference on computer and communications security*, pp. 269–279, 2009.

[5] "Dma.gg." https://dma.gg/. (Accessed: March 24, 2023).

[6] "Atomicdma." https://www.atomicdma.com/. (Accessed: March 24, 2023).

[7] M. L. Han, B. I. Kwak, and H. K. Kim, "Cheating and detection method in massively multiplayer online role-playing game: Systematic literature review," *IEEE Access*, pp. 49050–49063, 2022.

[8] H. Alayed, F. Frangoudes, and C. Neuman, "Behavioral-based cheating detection in online first person shooters using machine learning techniques," in *2013 IEEE conference on computational intelligence in games (CIG)*, 2013.

[9] T. Izaiku, S. Yamamoto, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito, "Cheat detection for mmorpg on p2p environments," in *In Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames '06)*, pp. 46–es, 2006.

[10] K. B. Muthe, K. Sharma, and K. E. N. Sri, "A blockchain based decentralized computing and nft insfrastructure for game networks," in *2020 Second International Conference on Blockchain Computing and Applications (BCCA)*, pp. 73–77, 2020.

[11] Y. M. Arif, M. N. Firdaus, and H. Nurhayati, "A scoring system for multiplayer game base on blockchain technology," in *2021 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, pp. 200–205, 2021.

[12] V. Sheherba and R. Hussain, "Blockchain and games: a novel middleware for blockchain-based multiplayer games," in *Proceedings of the SIGCOMM '21 Poster and Demo Sessions (SIGCOMM '21)*, pp. 9–11, 2021.

[13] N. Trojanowska, M. Kedziora, M. Hanif, and H. Song, "Secure decentralized application development of blockchain-based games," in *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8, 2020.

[14] L. Besancon, C. F. D. Silva, and P. Ghodous, "Towards blockchain interoperability: Improving video games data exchange," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 81–85, 2019.

[15] C. Yaklai and V. Kotrajaras, "An architecture for game to game data transfer using blockchain," in *2020 17th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 75–79, 2020.

[16] A. Pfeiffer, S. Kriglstein, and T. Wernbacher, "Blockchain technologies and games: A proper match?," in *Proceedings of the 15th International Conference on the Foundations of Digital Games (FDG '20)*, pp. 1–4, 2020.

[17] T. Min, H. Wang, Y. Guo, and W. Cai, "Blockchain games: A survey," in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, 2019.

[18] N. Amiet, "Blockchain vulnerabilities in practice," *Digital Threats 2*, vol. 2, no. 2, p. 7, 2021.

[19] B. Putz and G. Pernul, "Detecting blockchain security threats," in *2020 IEEE International Conference on Blockchain (Blockchain)*, pp. 313–320, 2020.

[20] N. Patel, A. Shukla, S. Tanwar, N. Kumar, and J. J. P. C. Rodrigues, "Gina: A blockchain-based gaming scheme towards ethereum 2.0," in *ICC 2021 - IEEE International Conference on Communications*, pp. 1–6, 2021.

[21] S. Kalra, S. Sanghi, and M. Dhawan, "Blockchain-based real-time cheat prevention and robustness for multi-player online games," in *Proceedings of the 14th*

International Conference on emerging Networking EXperiments and Technologies (CoNEXT '18), pp. 178–190, 2018.

[22] "Web3.js." https://web3js.org/. (Accessed: March 24, 2023).

[23] "example-.io-game." https://github.com/vzhou842/example-.io-game. (Accessed: March 24, 2023).