Cryptographic Directions in Cheating Detection and Prevention for Strategy Based Games

Joshua Lee jlee2022@mit.edu
Kevin Jiang kev2018@mit.edu
Suraj Srinivasan surajs@mit.edu
Lilian Wang lilianw@mit.edu

Abstract

User privacy and digital civil liberties have always been a relevant topic when it comes to companies collecting user data, and there has been ongoing debate as to whether this should be more firmly regulated. This paper prevents focuses on user privacy issues regarding cheat-detection in online strategy-based games. We first outline existing cheating methodologies in these games and some detection measures against cheating. We then investigate efforts to mitigate cheating through preventive measures rather than detection, and explore ways to incorporate recent findings in cryptography to implement these preventive measures, including the release Intel SGX and breakthroughs on indistinguishability obfuscation. Finally, we suggest directions for future research to develop both cheating prevention and detection measures that ensures both security and user privacy.

1 Introduction

In recent years, the online gaming industry has experienced rapid growth especially with the mainstream adoption of *e-sports*. To this end, the increased popularity of game streaming services (e.g. Twitch, YouTube Live, Facebook Gaming, etc.) and the recent introduction of financial incentives has driven this increased interest. In fact, e-sports is expected to become a 1.62 billion dollar market by 2024 [Gou21]. As a result, maintaining fairness and preserving the integrity of these games is of utmost importance especially as players' stakes in these games grow along with the online gaming ecosystem. On the other hand, increased stakes motivates the advancement of novel cheating strategies, leading to an ongoing cycle of cheating versus cheat-detection, racing to combat each other. Detection is further complicated by players creating new accounts in the event of a ban for cheating. As the cheat-detection methods evolve, the question of its incisiveness to user privacy then becomes increasingly relevant. Specifically, the dilemma becomes an effective way to determine a balance between methodologies to prevent cheating without violating users' privacy and digital civil liberties.

In this paper, we begin by exploring current cheat-detection mechanisms and in doing so, consider both sides of the problem (that is, we consider how a given individual may cheat and how this cheat may be mitigated). We perform this analysis in the context of chess, poker, and the class of first-person shooter games. This longitudinal analysis allows us to develop an understanding of the various techniques used by stakeholders. Subsequently, we proceed to looking at a historical and contemporary examples of anti-cheating software that infringed on users' right to data privacy. Using these examples as a cautionary tale, we determine guiding principles for the development of future cheat detection software such that the aforementioned balance may be achieved. Finally, under the premise of these best practices, we look to employ existing and emerging cryptographic primitives to achieve effective cheating detection.

2 Existing Methodologies

Currently, with games such as chess and poker, players have cheated not only for the prestige, but also when money is involved. In this section, we will first discuss current cheating tactics as well as

anti-cheating methodologies used to counter those tactics for some of these popular strategy-based games.

2.1 Chess

Players might use a range of tactics to cheat in the game of chess. A simple way is to start a second game on another tab against chess engine. Then, the player can copy all the moves of the engine against them over to the original game. This strategy essentially uses a chess engine to enter every move of the opponent to see what the software will suggest. An example of a popular chess engine is Stockfish. Another way of cheating is using chess bots. These chess bots can play moves for the cheater, and its almost instantaneous moves make it good for blitz or fast-paced games. The cheating rate for one of the most popular chess sites, Chess.com, just about doubled from 2019 to 2020, highlighting just how big the cheating problem has become.

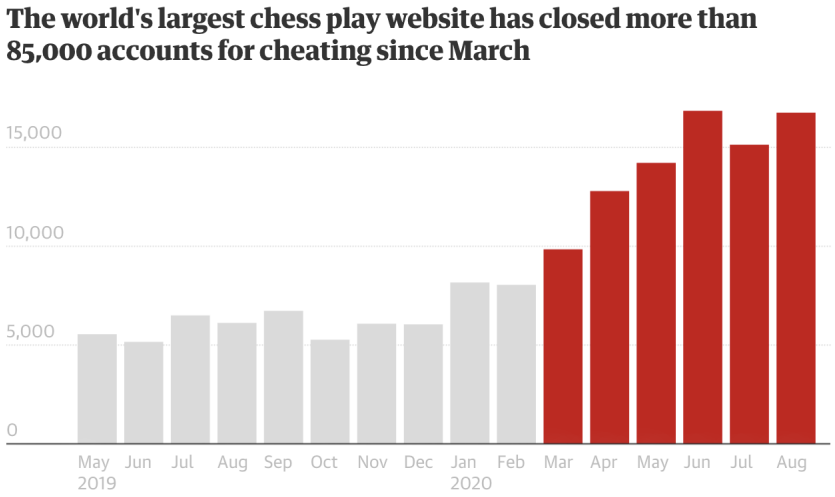

Figure 1: Users Caught Cheating on Chess.com (2019-2020), [Bla20]

With increasing ability to cheat, people have been searching for ways to detect and penalize players for cheating. To start, a lot of chess websites use an ELO System, where players gain or lose points based on how well they play relative to other players, and have ratings corresponding to their performance. As such, we can use tournament performance and their rating to gauge likelihood of cheating. A tournament performance significantly higher than expected based on their rating would be suspicious. Another measure we can look at is decision speed. If player is in a difficult position, but makes a strong move or sacrifices something without hesitation, or they're in an easy position, but has a delay for a move that should be trivial, this would also lead to reasonable suspicion.

Since a lot of cheaters might use the popular chess engine Stockfish, then checking the moves of a suspicious player using the chess engine is a good way to tell, especially if a lot of the moves are identical to the engine. More formally, some researchers have used the following metrics:

- Move Matching Percentage (MM): % moves matching those preferred by engine
- Coincidence Value (CV): proportion of non-book moves chosen by a player with the same evaluation (one of the top 5 moves) as the engine
- Average Error (AE): A way to quantify the difference between the best move and the move that was actually played

For move matching percentage, 100% would be considered evident cheating, while 80-90% remains ambiguous. Coincidence value was used by Barnes et al. [BHC15] to account for cheaters who might try to pick the second/third ranked (but equally good) moves to bypass any rank-based cheating detection. As we note, CV will always be greater than or equal to MM. Typically anticheat detection algorithms will look for high MM/CV percentages and low average error plays, which are likely candidates for cheating.

Some other factors to account for include human-probabilistic fatigue, where as the length of the game increases, the CV decreases naturally. We also have to be careful to look for false positives, since there are historic examples of games with high CV/low AE plays that did not involve any cheating, even if those metrics are a clear indication of cheating.

Barnes also discusses limitations of using engines for cheat-detection . In addition to false positives, the decision to use single-threaded or multi-threaded mode when running game analysis can yield different outcomes. Single-threaded implementations are deterministic, yielding identical evaluations over multiple runs, even on different machines, assuming all other configurations (such as search depth) were identical. On the other hand, multi-threaded implementations naturally introduced some non-determinism into the exhaustive search process during analysis due to the way threads are scheduled/managed (each thread receives some share of available computing resources) [BHC15]. These variations can be abused to falsely incriminate players by running it over different machines or depths until one gets desired results. Thus, these current metrics for cheat-detection must be used with extreme care and largely as supporting evidence to an already suspect player, rather than solely as the incriminating evidence.

2.2 Poker

With real money at stake, there is a lot of incentive for cheating in the world of online poker. Online casino platforms that provide poker play such as PokerStars, GGPoker, and partypoker have to constantly battle the threat of cheating to make their sites a safe platform for players to use. Current popular strategies used to gain an unfair advantage over other poker players include fully automated poker bots, bot networks/rings, assisted play, and player collusion.

A vast majority of fully automated poker bots are mediocre at best at the time this paper was written. Poker bots are fully automated when no human player needs to interfere with the program (i.e. the poker bot can both decide what the next move is and execute it on the platform). With the current advancements in technology, extremely good bots like CMU's Libratus and Facebook's Pluribus that can beat professional poker players are only achievable with the extreme use of resources like supercomputers. Most bots are only a threat to novice poker players, in which case the stakes are usually not high. Nonetheless, usage of poker bots in game formats such as cash games can give new players a bad experience.

When multiple poker bots are utilized at a single poker table, then that is called a bot network or bot ring. As mentioned earlier, a single poker bot at a poker table is usually not a huge threat except to the most novice of poker players. However, when there is a bot network at the same table, they can share card information between themselves, unbeknownst to the human player. This setup creates an extremely unfair advantage and can beat even very good poker players.

Another form of cheating is the use of assisted play, which usually takes the form of semi-automated poker bots. While fully automated bots both make decisions and execute them on a poker platform, semi-automated poker bots tell human players what decision to make. Since it is up to the human player to execute the decision via clicking buttons and dragging a slider for bet sizing, the human player receives an unfair advantage from the assistance of a bot.

Cheating on online poker platforms can exist in forms other than the use of bots. Player collusion takes on a similar flavor to poker networks/rings described earlier. Multiple players can enter the same cash game table or tournament and provide illegal information to each other in order to gain an unfair advantage.

Since there are many different ways of cheating in online casinos, large gambling platforms have to implement sophisticated ways of detecting cheating. Many popular online casinos like PokerStars do not reveal their secrets since real money is at stake. In a black box, dedicated anti-cheat teams use machine learning for pattern and anomaly detection to catch bots. PokerStars has access to approximately 200 billion hands, which allows them to track relationships between different players. Another method of detecting bots is the use of CAPTCHA, which can appear at any time during a poker game. Other heuristics such as mouse pointer tracking to see if players click the same pixel for actions, bet sizing patterns, and player statistics are used to catch bots in action. One potentially invasive anti-cheat method is monitoring programs. Some online platforms will look at what programs are being run in the background and ban accounts that use blacklisted programs.

2.3 First-Person Shooter Games

First-Person Shooters (hereon, FPS) are one of the more prevalent game genres played today. In particular, examples of these games include Counter Strike: Global Offensive (CS:GO) or Valorant, two of the most popular games in the field. The objective of these games is to find and eliminate opponents using a firearm. The basic mechanic involved is aiming at opponents with the mouse, which has a high ceiling for skill.

Players can give themselves an unfair advantage by using aimbots, which is software that automatically aims at opponents for the player. Another cheating software that a player might employ is the wallhack, which outlines the location of players on the opposing team that are not within the vision field of the player, as shown in Figure 2.

Figure 2: Wallhacking in CS:GO. An enemy player is outlined in red, and is normally not visible due to the wall between the players.

Wallhacks rely on being able to read game data that was intended to be hidden from the user. The standard procedure to achieve this is to uncover the location in game memory that, thereby retrieving the coordinates of the opponent players.

Another commonly used method of cheating is the usage of scripts which are also often used in other types of games such as *League of Legends*. Scripts attach themselves to game memory and code, retrieving access to some of the functionalities that would otherwise be impossible. An attacker could use a debugger to find the necessary functions in the code, as explained in [Van18]. This method is called “hooking", and one possible way this attack could be used in CS:GO is to change user speed parameters and input timing to perfectly execute a difficult maneuver known as bunny-hopping.

Detection of these cheats primarily relies on player reports and reviews. If one player suspects another of cheating, there is a built-in report system that would send game footage to other members of the community, who would then review the replay of the game to pick out any suspicious behavior. An example would be when the cursor of the player abruptly locks into the desired aim location, which is a strong signal that the player may be assisted with an aimbot.

There are clear limitations to this method, since there is a lot of room for human error. In addition, players using the hacks could be more careful about their usage to avoid detection, such as only using the programs every few rounds to mitigate any suspicion. This motivates more automated methods of cheating detection, which are outlined below.

The first method is to detect signatures of known cheats. Cheating programs modify the computer’s memory in unique ways known as signatures, which could be detected by an anti-cheat software through a scan, as described in [Leh20]. While effective, it is relatively simple to circumvent this strategy by compromising the channel through which data about the cheat program is sent to

the identifier. A sophisticated attack would obfuscate its signature in the user memory to avoid identification. An example of anti-cheating software which looks to detect cheat signatures is given in 3.1.

Aimbots are often passive programs that do not leave some of these recognizable signatures. It is also often very difficult to pick out by hand, since an aimbot may appear to simply be a very skilled player. To address this issue, Liu et al. devised a scheme where aimbots could be differentiated from human players [LGZ+17]. By placing “bait-targets" on the map that resemble opponent players to a third-party software scanning for where to aim, the authors were able to trick the aimbots to aim accurately at objects that would not appear as enemy targets to the human eye.

An effective way to mitigate scripts and wallhacks is to encrypt the game code to make hooking using a standard disassembler more difficult. In addition, the memory locations of game data could be obfuscated to prevent procedures that try to determine the memory addresses. The packets that are sent from the game server to the client can be encrypted as well, thereby preventing packet-sniffing to obtain the game data. Specifics for these encryption techniques are outlined in Section 4.1.

A more invasive procedure would be to use kernel-based anti-cheat drivers, which enters the user kernel to detect any suspicous processes. These methods are effective and rising in popularity, and are outline in Section 3.2.

3 Implications on User Privacy and Guiding Principles

Broadly speaking, there are implications of anti-cheating methodologies with respect to user privacy and digital rights management. Here, we consider both a historical and contemporary example of the invasive nature of anti-cheating methodologies and subsequently define generalized guiding principles on how these policies should be designed.

3.1 Warden Anti-Cheat

Developed and deployed by Blizzard Entertainment in late 2005, the software known as *Warden* looked to prevent cheating across the companies popular titles including Diablo, World of Warcraft, and Starcraft. The software purportedly operated by scanning the code segments of currently running processes on a user’s system such that third-party programs could be detected. Specifically, Warden would hash the code segments of running processes and compare this hash to known values of cheating software banned by Blizzard. Thereby, Warden aimed to prevent cheating across Blizzard’s titles albeit being highly invasive [war].

To this end, a brief investigation of the terms of service users agreed to when signing up for World of Warcraft reveals the following: "When running, the program [Warden] may monitor your computer’s random access memory (RAM) and/or CPU processes...Blizzard may obtain certain identification information about your computer and its operating system, including without limitation your hard drives, central processing unit, IP address(es) and operating system(s)..." [wow08]. This contract language demonstrates a pathway leading to blatant violations of privacy and such an avenue was later found by cyber-security researcher Greg Hoglund, who thoroughly analyzed the runtime operations of Warden. Hoglund found that the anti-cheating software not only scanned the title bars of all currently open applications, but also sniffed the URLs of webpages he currently had opened and the email addresses of individuals with whom he was communicating [hog05].

This analysis prompted the Electronic Frontier Foundation, a nonprofit focused on defending digital civil liberties, to declare Warden a *spyware*. In response these concerns and the outrage of millions of players, Blizzard asserted that: 1) the Warden software did not actually collect nor utilize personal information, 2) several other prominent programs and softwares have some implementation of hack-scanning, and 3) users should have been aware of Warden’s capabilities from the terms of service. We maintain that none of these arguments adequately explain the serious invasion of user privacy detailed above and look to provide guiding principles to balance anti-cheating and user privacy [McS05].

3.2 Kernel-Based Methods

Before proceeding, we briefly note that invasive approaches to anti-cheating continue to be utilized in the industry including most prominently kernel-based methods.

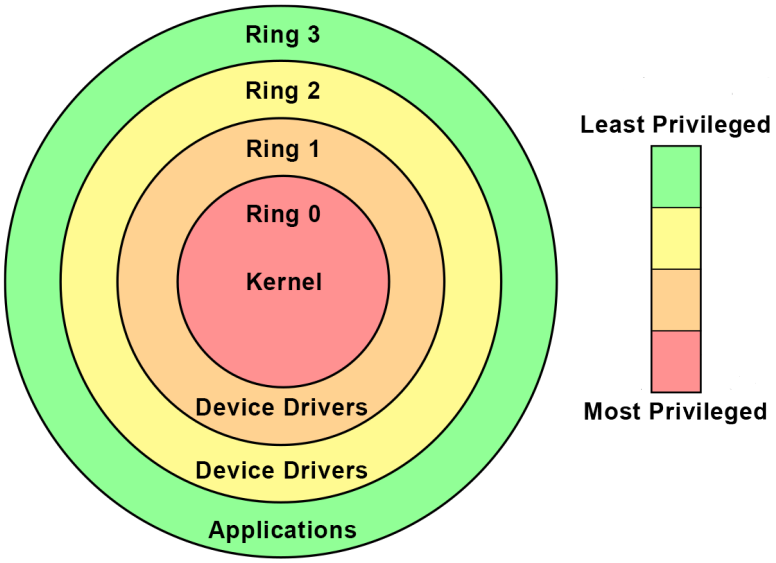

Figure 3: Privilege level in operating system, [lea]

Riot Games, a company popular for creating some of the most played video games today such as *League of Legends* and *Valorant*, implements such an invasive anti-cheat systems. Most anti-cheat software operates on user-mode, which is when an application only has direct access to the memory of its own process. In order to read or write memory outside of its process, or "see outside" of itself, the program has to rely on the operating system [lea]. Figure 3 depicts the privilege levels of various applications. Most cheating software operates on user-level, which is at ring 3, but Riot has implemented anti-cheat software at ring 0, which is the kernel mode. In the kernel mode, the application can see exactly what programs are being run on the operating system. Furthermore, methods operating in the kernel space generally have access to the systems memory, hardware, all CPU instructions, and critical operating system data structures. Riot's reasoning for creating anti-cheat software on the kernel level is to make the strategy more sustainable for the future.

Having software that can have access to core level of an operating system is very dangerous if left unchecked. By using a kernel driver to detect cheats, Riot is essentially "[moving] the cat-and-mouse game between game developers and cheat developers from user mode to the kernel", "[encouraging] cheat developers to implement cheats in ring 0", which makes "it even more difficult for Riot to detect cheating in the future" [val]. Moving the game to the kernel level also produces many privacy concerns that Warden also raises and invites cheaters to attack the game at the kernel level. In fact, Riot Games released a game in 2020 called *Project A*, or *Valorant*, that employs an anti-cheat software called Vanguard, which uses a kernel driver.

3.3 Guiding Principles for Cheating Detection

Here, we look to develop guiding principles for the future of cheat detection methodologies based on historical examples and considerations for user privacy.

First and foremost, we maintain that the ability of a software to access user data is equivalent to the utilization of that data and to this end, a given software and associated cheat detection methodology should be self encapsulated. That is, the program should not have any access beyond the user level (ring 3 in Figure 3) let alone be able to access user data. Such a principle invalidates the aforementioned response from Blizzard which maintained that while Warden had access to user data, it was not necessarily using said data.

We also broadly prescribe that cheat detection methods should remain on the server-side as much as possible. By doing so, any invasions of user privacy are prevented and the integrity of the game is

maintained.However, we acknowledge this does not necessarily protect against all hacks (such as wall hacking).

Currently, kernel-based drivers, or similar invasive cheat detection schemes, appear necessary to detect such active cheating software on the client side. Being able to access all of the system memory, CPU instructions, and system data is almost a prerequisite for effective detection of cheating. Improvements can be made, however, to protect such drivers from side-channel attacks by a third party. This is accomplished in Section 4.2 through the usage of Intel SGX. Other potential improvements could involve increasing transparency of the kernel-based drivers themselves to the user to ensure that no malicious activity is occurring, although this would prove useful only for users that are familiar with computer systems. In addition, an increased insight into the driver would likely open up other vectors of attack that can circumvent it.

We note that there do exist some anti-cheating methodologies which achieve the balance between security and privacy; however, these methods suffer in other aspects. For example, Valve utilizes VACnet, deep-learning based cheat detection for CS:GO. The model is trained on examples of cheating while observing game play passively. It then flags any suspicious activity that is passed on to be reviewed by a board of experienced players who then adjudicate the gameplay and reach a final verdict. This process remains purely on the server-side, making it inherently non-invasive and no user data is ever exposed to Valve nor VACnet. However, this method is lacking from a functionality and resource standpoint in that it can only detect players who are cheating very obviously. It is also not easily scalable due to its need for an experienced review board, and has high overhead in terms of computational and physical cost [hbs]. Further research into these server-side methodologies can uncover methods that achieve the security provided by kernel-based drivers while maintaining the privacy provided by current server-side methods.

4 Cryptographic Directions

Considering the above guidelines, we suggest the following methods stemming from cryptographic fundamentals and new directions in the field.

4.1 Packet Encryption and Memory Obfuscation

In Section 2.3, we briefly discuss methods of cheating through reading packets that are sent from the server to the client, or by finding the memory addresses of vital game data and game code. These attacks can be thwarted by carefully encrypting the packets so they are unintelligible to an eavesdropper, and by obfuscating the memory to confuse any programs that try to determine the memory locations. [Van18] outlines a method of memory obfuscation that moves the addresses of important variables after they are modified to limit memory-searching. This does create a lot of computational overhead, as analyzed in [Leh20]. Another method of memory obfuscation that has less overhead is to selectively encrypt heap variables, which may allow an attacker to be able to access some variables but not all.

4.2 Intel SGX

In 2015, Intel introduced an extension dubbed Software Guard Extensions (SGX) to the Intel architecture that aimed to provide security guarantees for processes that were run using trusted hardware but potentially malicious software. [CD16] provides an extensive overview of the extension, and we highlight the components that are relevant for our uses.

Intel SGX utilizes software attestation to ensure security. SGX provides a secure enclave, or a space separate from the main kernel of the hardware, in which a program could run. This program would have a unique attestation key that the remote user, in our case the game company, can use to verify that they are communicating with only the software contained within the enclave. The enclave would then contain private data and private game code, and any external programs run by the same piece of hardware would be unable to access this information.

4.2.1 Usage of Intel SGX in Cheat-Detection

As noted above, one major problem with an invasive kernel-based anti-cheat driver is that if one successfully can attack the driver, they can potentially achieve full access to the memory space and processes of the entire kernel, compromising the security of the entire machine. This also provides problems for the game developers, as their entire game code may be available to the attacker.

The authors of [BRM20] provide a potential anti-cheat mechanism that addresses this issue using Intel SGX. The enclave needs to be able to access game memory regions and its dependencies, and this information is copied into the enclave from the game file. The enclave then stores the necessary information to monitor the regions of the memory that is responsible for gameplay for any unexpected or unwanted changes. If an attacker attempts to directly modify game code or abuse game functionality, the enclave can contain an integrity checker of the game. Because the checker is secured by the enclave and therefore cannot be modified, any attacker that tampers with the game will fail the integrity checker.

4.2.2 Limitations

The implementation outlined by the authors of [BRM20] still assumes that the game company themselves are a trusted party, since they have full control over what is included in the enclave. The anti-cheat detection program still makes use of the kernel module, and the protection is only against third party attackers who intend to attack the kernel through the use of the game's invasive driver. In fact, since SGX provides no protection guarantees to the user's own kernel, it may be difficult to devise a cheat-detection driver that would protect the user from any other party based on Intel SGX alone. Usually, the game companies outline the invasive nature of some of these drivers in the terms of service documents, which the players have to agree to in order to play the game. It is in the best interest of the consumers to understand the risks associated with providing the game company access to parts of their kernel.

There are additional technical difficulties with securing a license to use SGX for cheat-detection driver development. During the initial release, Intel added a feature that required any party desiring to use the enclave in SGX to obtain approval from Intel. However, we note that the fundamental principles underlying Intel SGX and the idea of using a secure enclave to prevent cheating is sound.

4.3 Indistinguishability Obfuscation ($i\mathcal{O}$)

The cryptographic idea of indistinguishability obfuscation ($i\mathcal{O}$) introduced by Barak et. al. may be utilized to achieve the necessary balance between user privacy and the prevention of cheating. Specifically, the premise $i\mathcal{O}$ involves the use of an obfuscator $\mathcal{O}$ which acts as a compiler with respect to an input program or circuit P and produces $\mathcal{O}(P)$. Subsequently, we have that the compiled version of the program $\mathcal{O}(P)$ has the same functionality as P but is *unintelligible* from a user standpoint. The authors note that an obfuscated program effectively operates as a black box whereby the input-output behavior of the program may be efficiently ascertained. However, no information about the execution of the program is revealed to the user and the underlying codebase is effective incomprehensible. We note that an principle of $i\mathcal{O}$ includes computational indistinguishability whereby two obfuscated programs $\mathcal{O}(P_0)$ and $\mathcal{O}(P_1)$ may not be discriminated by some probabilistic polynomial time adversary with non-negligible probability. [BGI$^+$12]

Fundamentally, applying $i\mathcal{O}$ to the problem of cheating prevention will operate in a fashion such that the game code for some game G is never revealed to a given player (hereon, referred to as the adversary). Thus, the only interaction the adversary has with $\mathcal{O}(G)$ are the inputs they provide (i.e. keyboard and mouse) and the output from the obfuscated game. Thereby, a cheat may not be effectively deployed as nothing may be learned about the game state and thus, cheats may be prevented. [JLS20]

We note that the premise of indistinguishability obfuscation is relatively theoretical at this point and is a primitive that does not quite work in practice at this time. A wide body of literature details various constructions of $i\mathcal{O}$; however, these methods require bespoke hardness assumptions designed specifically in the context of the construction. These hardness assumptions have either been demonstrated to be infeasible or are purely theoretical in nature. [JLS20]

Nevertheless, we detail a promising construction presented by Sahai and Waters which presents the notion of *punctured programs*. This methodology works by altering a program by "surgically removing a key elemtno f the program, without which the adversay cannot win the security game it must play". To this end, we envision that 'puncturing' the portion of game software which deals with rendering gameplay to the user and relevant parameters will work effectively in making the game obfuscated. Thereby, we may effectively apply the notion of indistinguishability obfuscation to game software such that the underlying codebase is unintelligible, and thus cheating is inherently prevented. We briefly note that this approach only prevents certain types of cheating (i.e. those which rely on exploiting the game code); however, we believe that this is an effective starting point to anti-cheating methodologies which respect user privacy. [SW13]

5 Conclusion

As the e-sports industry continues to increase in popularity, the incentives for cheating grows. Many popular games such as chess, poker, and first-person shooters have different methods of cheating that we outlined earlier. The platforms that run these aforementioned games create anti-cheat software in order to maintain game integrity as much as possible. Unfortunately, some anti-cheat programs such as Blizzard's Warden and Riot Games' Vanguard come at the cost of user privacy. As we saw from historical examples, it's hard to find the right balance between these two factors: user privacy and game integrity.

Based on current methods, we proposed new guiding principles for designing an anti-cheating system while introducing cryptographic primitives. Both principles focus on maintaining user privacy as much as possible. The first principle is that programs should not have any access beyond the user level, so methods that utilize kernel drivers are looked down upon. If kernel drivers are utilized, then data access should be strictly monitored to prevent user data abuse. The second principle is to make sure that cheat detection methods remain on the server-side as much as possible so that user privacy invasions are prevented.

Based on the user privacy principles, we suggested the deployment of existing and emerging cryptographic methods such as packet encryption, memory obfuscation, the Intel Software Guard Extensions (SGX), and indistinguishability obfuscation for cheat detection. Packet encryption helps with preventing cheating software from intercepting and reading packets sent from servers to the client. Obfuscating the memory also achieves the same goal by confusing programs that try to determine memory locations. The Intel SGX extension described in detail in 4.2 provides an alternative to the invasive kernel-based anti-cheat drivers. Finally, we investigated the use of indistinguishability obfuscation ($i\mathcal{O}$) for cheat prevention. Successfully implementing $i\mathcal{O}$ can essentially hide the game code from players, rendering cheat programs as ineffective since no information can be learned about the game state from an obfuscated game.

In brief, we have suggested new policies for anti-cheat software that better maintain user privacy than existing methods. We believe that our topic is not only useful for strategy-based games but also has applications in anti-cheating in online exams, which also deals with issues of privacy in proctoring.

References

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2), May 2012.

[BHC15] David J. Barnes and Julio Hernandez-Castro. On the limits of engine analysis for cheating detection in chess. *Computers Security*, 48:58–73, 2015.

[Bla20] Archie Bland. Chess's cheating crisis: 'paranoia has become the culture', Oct 2020.

[BRM20] André Brandão, João Resende, and Rolando Martins. Employment of secure enclaves in cheat detection hardening. 09 2020.

[CD16] Victor Costan and S. Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.

[Gou21] Christina Gough. Global esports market revenue 2024, Mar 2021.

[hbs] Valve using machine learning and deep learning to catch cheaters on CS:GO (794 words).

[hog05] Warcraft game maker in spying row. October 2005.

[JLS20] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions, 2020.

[lea] /dev/null: Anti-Cheat Kernel Driver - League of Legends.

[Leh20] Samuli Lehtonen. Comparative study of anti-cheat methods in video games. 2020.

[LGZ+17] Daiping Liu, Xing Gao, Mingwei Zhang, Haining Wang, and Angelos 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)*, pages 615–626, 2017.

[McS05] Corynne McSherry. A New Gaming Feature: Spyware, October 2005.

[SW13] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Cryptology ePrint Archive, Report 2013/454, 2013. `https://eprint.iacr.org/2013/454`.

[val] A closer look at Valorant's always-on anti-cheat system.

[Van18] Michael VanKuipers. Riot's approach to anti-cheat, 2018.

[war] Deceiving Blizzard Warden – HackMag.

[wow08] WoW Terms of Service, May 2008. publisher: Blizzard Entertainment.