TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

Uku Markus Tammet 142684IAPB

IMPLEMENTING METHODS FOR PREVENTING CHEATING IN PEER TO PEER ONLINE GAMES

Bachelor's thesis

Supervisor: Jaagup Irve
Master's degree

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Uku Markus Tammet 142684IAPB

SOHIVASTASTE MEETODITE IMPLEMENTEERIMINE VÕRDÕIGUSVÕRKUDEL PÕHINEVATES VÕRGUMÄNGUDES

bakalaureusetöö

Juhendaja: Jaagup Irve
Magistrikraad

Tallinn 2017

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Uku Markus Tammet

22.05.2017

Abstract

This bachelor's thesis explores problems with online video game architectures, more specifically, cheating in peer to peer games. We show the key unsolved problems of avoiding cheating in these games, and propose multiple solutions to solving some of them. We find that blockchain could become a viable method of cheat-proofing certain kinds of online games, but how the basing the functionality off proof-of-work is not feasible for most games. We show how to generate random numbers in distributed consensus while not allowing for the manipulation of said numbers. Finally, we bring together a subset of these methods and implement interaction resolution and random number generation in distributed consensus for a peer to peer game using said methods, uncovering a scalability problem.

This thesis is written in English and is 24 pages long, including 5 chapters and 2 figures.

Annotatsioon

Sohivastaste meetodite implementeerimine võrdõigusvõrkudel põhinevates võrgumängudes

Käesolevas töös uuritakse probleeme erinevate võrgumängude arhitektuuridega, spetsiifiliselt, võrdõigusvõrkudel põhinevates võrgumängudes sohi tegemist. Me näitame põhilisi lahendamata probleeme sellel alal ja pakume nendele välja potensiaalseid lahendusi. Me leiame, et *blockchain*il baseeruv tehnoloogia võib tulevikus muutuda reaalseks abivahendiks selle juures, kuid *proof-of-work*i kasutamine võrgussisese üksmeele loomiseks ei ole sobilik enamuste mängude jaoks ning et peaks uurima alternatiivsete jagatud üksmeele loomise meetoditeid. Me näitame, kuidas saab genereerida suvalisi arve nii, et võrdõigusvõrgu liikmetest on valdav enamus rahul ja kirjeldame, kuidas nende loomist ei saa manipuleerida. Lõpuks, me toome kokku alamhulga kirjeldatud lahendustest ja loome vordõigusvõrgul baseeruvale näidismängule hajutatud suvalise arvu genereerimise ja kodeerime mängu reeglid pearaamatusüsteemi, leides sealjuures probleemi skaleeruvusega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 24 leheküljel, 6 peatükki ja 2 joonist.

List of abbreviations and terms

P2P	Peer-to-peer, a type of network application architecture
SOI	Sphere of influence
TCP	Transmission control protocol
TLS	Transport layer security, a secure communication protocol
MITM	Man-in-the-middle, a type of attack
JSON	JavaScript Object Notation, a lightweight data-interchange format
JavaScript	A high-level, dynamic, untyped and interpreted run-time programming language, standardized in the ECMAScript language specification
Node.js	An open-source, cross-platform JavaScript run-time environment
Blockchain	A distributed database system that prevents tampering using cryptography
ECMAScript	A scripting language specification standardized by Ecma International
HTTP	Hypertext transfer protocol, an application protocol, used in the World Wide Web

Table of contents

List of figures

1 Introduction

Video gaming has become a favourite pastime for large amounts of people, reaching up to 49% of people in the U.S. in 2015 [1]. When these games are played with other people over the internet, various architectural challenges arise for the creators of the game to ensure that their customers are happy. Online games use a multitude of different architectures to enable players to play together online [2]. In this paper, we explore one of those challenges, namely, cheating, and how to implement effective methods to stop different variants of it in the context of online games with P2P (peer-to-peer) architecture by using different methods of verification.

2 Common architectures of online games

Online games are often complex pieces of software, utilizing a multitude of different systems to make the game run smoothly. However, their general approaches can be roughly divided into two categories: client/server and P2P architectures.

2.1 Client/server architectures

The client/server architecture for online games outlines that there is a central source of truth. That source of truth is a server that all clients connect to. All communication of game state changes happens between a single client and the server. This communication causes the server to propagate game state changes to all clients. Every player runs a separate client that talks to said server, where the client is responsible for communicating game changes coming from the server to the player, and facilitating transport of the player's actions back to the server [2].

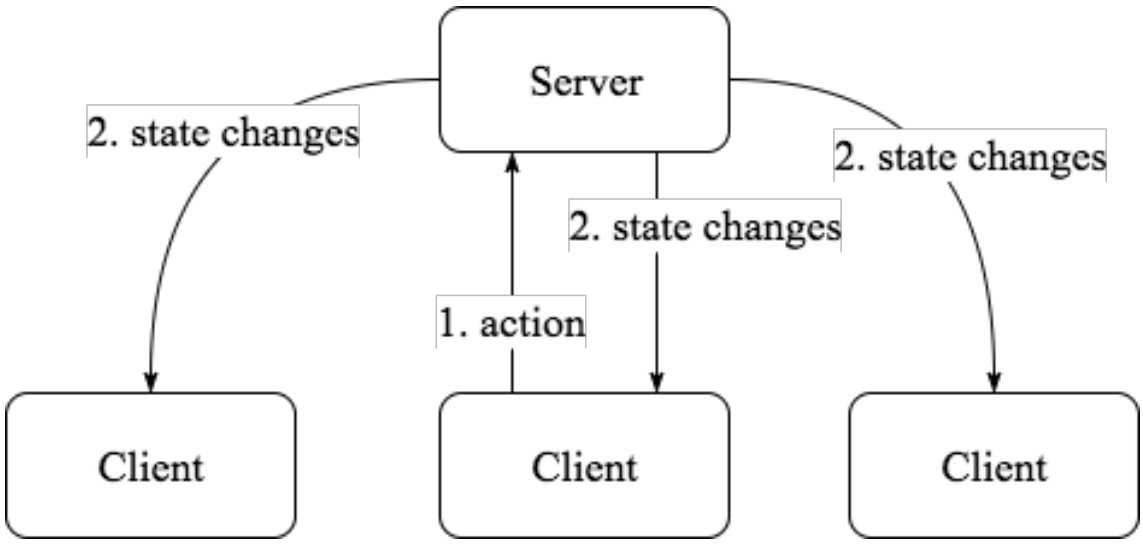

Figure 1. An example of client/server architecture.

Online games architectured in this way can rely on the fact that the server is considered trustworthy, while the clients cannot be. This central source of truth makes the server convenient to be the place where all cheat prevention and discovery takes place [3].

2.2 Peer-to-peer architectures

P2P architectures generally have no central source of truth. All players control a peer, which maintains its own game state. Player actions are turned into state changes and broadcast via some mechanism to other peers. These other peers react to the changes and update their game state accordingly, which, in turn, updates what's shown to the player. There are of course exceptions to this, such as maintaining a centralised listing of peers. Using either of the two given architectures does not mean that parts of the system cannot be architectured in the other way.

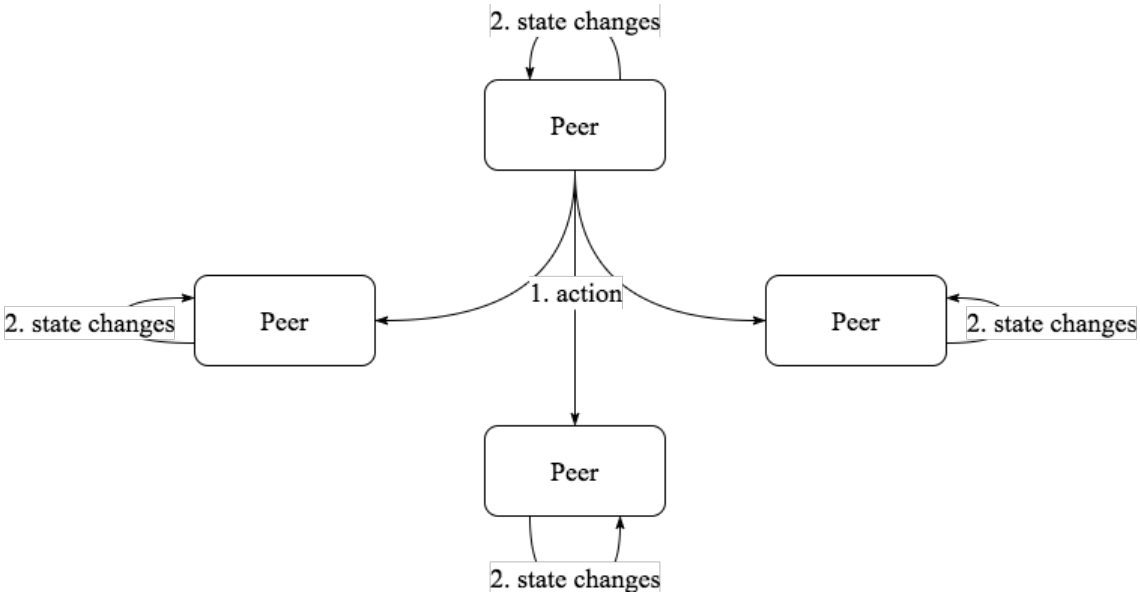

Figure 2. An example of P2P architecture.

Since peer-to-peer online games cannot rely on a central trustworthy element, every peer has to implement cheat prevention on their own, which turns out to be a lot more difficult than just relying on a single source of truth in the system. Every peer has to assume that other peers are not trustworthy.

Even though cheat prevention is a challenge, there are large upsides to using this type of architecture. First, the proprietors of the game do not have to run physical deployments of central servers, as every peer is essentially also a server. Because of that, the game is also more easily scalable, as every peer brings their own computing power to the game. Lastly, since there is no single point of failure, downtime in peer-to-peer games is rare, as long as the game is tolerable of single peers stopping service [3].

Since peer-to-peer architectures can be useful, but hard to use in a system where cheat prevention is relevant, we would like to explore and implement some novel approaches to cheat prevention in peer-to-peer games.

3 Preventing cheating in P2P games

J. Yan and B. Randell have classified online game cheating into 12 broad categories [4]. In this paper we will be focusing on cheating methods more problematic to solve in P2P games as opposed to client/server games, such as cheating by exploiting misplaced trust, cheating by modifying client infrastructure, cheating by exploiting lack of authentication, timing cheating and cheating by exploiting lack of secrecy while keeping in mind exploits caused by implementation errors, such as cheating by exploiting a bug or loophole.

Online games have a multitude of attack vectors. In these games, one player, given the state of the game, will perform some kind of action that will change the state of the game, which will have to be propagated to other players. The action taken by the player must be valid in the context of the rules placed on the game. Thus, games need to be able to validate an action based on the state that the action was played upon. However, the possible actions might not always be deterministic, as often random numbers come into play, for example, when drawing cards from a deck. In fact, in this example, it is essential to the rules of the game that the card drawn from the deck is not deterministic, or at least not knowingly deterministic to the players. In this case, a random factor must be brought to the game. In these games, a player is given the previous state of the game and a random factor prior to making their move. Thus, not only do we need to validate that actions taken are valid considering the previous state of the game, we also need to validate that a random factor is truly non-deterministic in the context of the game, or at least that it was not possible for players to know about the random factor prior to making their move.

3.1 Keeping track of state

Since the drivers of change in any multiplayer game are the actions performed by the players, it is important to be able to quickly validate that an action received from a peer is allowed within the rules of the game for the current state of the game. In multiplayer games, the sequence of actions that determines the current state can be complex and

outlast a single play session. Every peer keeping a trusted ledger of actions that determines the state is a simple solution, but in that case, the missed actions that other peers have taken while the current peer is not interacting with the game have to be replayed somehow to the current peer. Here, we encounter another problem, namely, we now need to validate that not only is a received action valid for the current state, we also need to catch up on actions and verify that the information we received when catching up is valid. Essentially, what we're looking for is a distributed ledger [5].

3.1.1 Blockchain for games

Thankfully, distributed ledgers are a topic of popular research. Namely, Satoshi Nakamoto proposes a system of hashed blocks of transactions pointing to the previous block, where consensus of this kind of ledger is guaranteed to be accepted by the majority using a proof of work system [6]. This system, blockchain, can potentially work for elements not strictly classified as monetary transactions. Thus, we propose to use a blockchain system for keeping track of the actions of players.

In a blockchain type distributed ledger, every entry (block) has a field pointing to the previous entry. These chains can diverge, but eventually, they have to converge so the state of the distributed ledger would be consistent. In bitcoin, this problem is solved using a proof of work system. This means that adding blocks to the chain requires some amount of resources, CPU time in the case of bitcoin. When a conflicting state is detected by a peer, the peer selects the longest chain, which happens to be the most computationally intensive chain to generate, as the correct state. This guarantees that state is infeasible to modify by a malicious peer, as it would have to control more computational power than the rest of the network. This means that whomever controls the majority of computational power controls the state of the chain.

In bitcoin, there is an incentive to spend resources mining these blocks. Namely, every mined block generates bitcoins that are awarded to the miner. If, however, our game blockchain system is not strictly based on monetary transactions, we need to find an incentive to make participants in the network work on mining in order to guarantee that the state is controlled by the majority of the computational power.

Participants play the game to have fun. The main path to this is to perform actions yourself and see them influence the state of everyone else. In order to influence the state

of others, peers must have their actions be entered into the ledger of everyone else. Since mining directly makes influencing others using actions faster, we postulate that decreasing delay will provide some incentive to peers to at least mine their own actions. Furthermore, in a turn-based game, an action is not valid if it is not taken during the correct peer's turn. This means that actions cannot be taken until other peers have finished their actions. This provides incentive to mine everyone's actions to get to perform actions faster.

A fundamental problem with using proof of work for game blockchains, however, is that by definition, it takes time to add blocks to the ledger. This is highly problematic for games, especially real-time ones, as players do not want to wait. This could be potentially usable for turn-based games, however. In addition, if some games can sacrifice being verifiably up-to-date with the common ledger, they can utilize unverified actions in gameplay. The scalability of this system has potential, as consistency only has to be achieved eventually, so all peers do not have to wait for all other peers. Furthermore, every peer does not have to send messages to every other peer, only receive events. This makes blockchain a viable candidate for a niche set of games. Further research in other viable alternatives to proof-of-work, such as proof-of-stake [7], in the domain of multiplayer games could potentially yield a more appropriate solution to games outside of this niche.

3.1.2 Interaction resolution

A simpler but more rudimentary method to guarantee correctness of actions is a type of *stop-and-wait* protocol where every action must essentially *clear* with every peer before it can be used. N. E. Baughman, M. Liberatore and B. N. Levine have outlined some problems with the naïve implementation of this, where players gain knowledge of sent actions before they are applied to every peer’s game state [8]. They solved this by developing their Lockstep and Asynchronous Synchronization protocols. In these systems, peers synchronize on cryptographic hashes of actions, having them clear before sending the actions themselves. This way, commitment to making an action must be made before any of the other peers’ actions are visible. In addition, with Asynchronous Synchronization, the time to resolve interactions is reduced compared to both the naïve implementation and Lockstep.

Asynchronous Synchronization assumes that localization, that is, confining actions and state changes to certain SOIs (spheres of influence), is possible. However, this might not always be so. For instance, in large strategy games, where players might be interacting with each other at all times. In these cases, Lockstep does not get the benefits of Asynchronous Synchronization, and every peer still has to wait for every peer's cryptographic hashes to clear. In addition, these protocols do not cover catching up on actions that might have happened while a player was missing from the game, so by themselves they are not sufficient to run a long-running game.

3.2 Validating that an action propagated to other peers comes from a certain peer

Since actions need to be sent by peers, to verify their validity, we need to know for certain that an action has been sent by a certain peer. A solution to this can be found using public-private key cryptography, like RSA [9]. Using such cryptography, we can sign every action sent with the peer's private key, while sending the public key along with the signed action. By using public keys as peer identifiers, peers can verify that a certain peer (identified by a public key) has sent some action, as nobody besides that peer could have created a valid action signature for said public key.

3.3 Creating a random factor in a P2P distributed game

In many kinds of online games, the state of the game is not purely deterministic on the actions of the players, but also relies on the game's environmental factors, which often turn out to be based on a pseudorandom number. For example, in a game of cards, the order of the deck (or at least the order of the drawn cards) must be random. In these kinds of cases, peers need to be able to verify and agree that a generated number is sufficiently random, or at least that it's not deterministic to the degree that it could have been forged to further a malicious peer's agenda. For instance, if a player could influence the order of the cards they draw, they could make themselves draw an advantageous card, and make their opponents draw bad cards.

Assuming that every non-malicious peer has access to a cryptographically secure pseudorandom generator with a sufficiently good seeding mechanism and a secure hashing algorithm H, there is a potential solution to this resembling Lockstep. If some

number of peers want to generate a random number from 0 to N, every peer starts by picking and storing a secret number from 0 to N-1. Afterwards, every peer also generates a second random factor, the type of which does not matter. We will call this second factor a salt. Once every peer has a random number and a random salt, they append the salt to the random number, generate a hash out of the appended value using algorithm H and broadcast this hash to every peer in the network. After this is done, every peer must verify that this hash has been received by everyone else. When hash acceptance has been verified, every peer sends their generated random number and salt to every peer. When receiving these random numbers and salts, a peer must verify that when appended together and hashed using algorithm H, they create the same hash that was sent earlier by the same peer that sent the hash. This way, peers can verify that the random numbers sent earlier were generated before knowledge was obtained about other random numbers. Using the salt value lessens the probability of hash collisions, that is, a peer cannot generate numbers from 0 to N-1 and find the corresponding hash to gain knowledge about the random numbers in play. Once everyone has everyone's numbers, they add all the received numbers together and take modulo N of the resulting sum for the final random number [10]. This approach guarantees that if at least one peer is not maliciously colluding with others outside of the system, the resulting number will not be predictable.

If at some point peers detect foul play, for example, a random number and salt that, when added together and hashed using algorithm H, do not match the hash sent earlier, they can collectively decide to start ignoring the actions of the malicious peers sending these incorrect messages, effectively banning them from the game. In this way, the game states of malicious peers and honest peers can separate, leaving only honest players playing together.

4 Implementing a game based on interaction resolution and distributed random

To test out interaction resolution and generating random numbers in distributed consensus, we implemented game systems for an example game using them. The code for the application can be found via this link: https://www.dropbox.com/s/v348q1rms15zw39/application.zip?dl=0.

4.1 Gameplay

As our goal was to test cheat prevention methods, we did not focus on creating a functional game, but instead created underlying systems to prevent cheating. To this end, we decided to forego a user interface and build systems for a coin-gathering game with relatively simple rules. These rules are the following:

1. The game world is an effectively infinite grid of two dimensional squares, each of which has infinite capacity to contain players and coins.

2. The game world can be joined once all peers agree that they are all ready.

3. Upon joining the world, a peer starts on a random grid square with both x and y coordinates being randomly picked from 0 to 10.

4. A player can move at a maximum rate of 1 grid square per second, horizontally or vertically.

5. Every 10 seconds since joining the world, a player may spawn a coin in a random grid square with both x and y coordinates being randomly picked from 0 to 15.

6. When a player moves onto a grid square with coins on it, those coins get added to their score.

7. The player with the highest score is considered the current winner.

4.2 Caveats

Because our interest lied in the actual cheat-proof algorithms researched, we decided to forego building multiple systems. For example, we do not work on routing of requests nor peer discovery. Instead of that, we built a simple centralized and trusted catalogue server to give connection information to peers that need it.

In addition, we implemented communication between peers using TCP (transmission control protocol), while in a real-world scenario, this would have to be done using some encrypted protocol, such as TLS (transport layer security) to avoid MITM (man in the middle) attacks, that is, malicious actors intercepting TCP packets and changing them to their advantage. In our example implementation, we consider TCP to be a secure channel, although it is not [11].

We also decided to forego creating a user interface for our game system, as building game user interfaces is outside the scope of this paper.

4.3 Architecture

Our game system consists of two main components, the catalogue server and the peers. The catalogue server is a simple service which serves the IP (internet protocol) addresses and server ports of all currently active peers. This functionality would be included in the peers in a real-world scenario.

The peers connect to the catalogue to both register themselves and receive the addresses and server ports of other peers. These addresses and server ports are used to connect to other peers using a TCP client. Every peer runs a TCP server on their end to receive messages from other peers. To send messages, peers connect to the recipient peer using a TCP client.

4.4 Technologies used

As this application utilizes network communications heavily, it is asynchronous in nature. Because of this, we needed a platform to allow us to easily work with asynchronous programming without creating concurrency-related issues. To this end, we decided to implement both the catalogue and the peer components using Node.js, an

open-source, cross-platform JavaScript run-time environment built on Google Chrome's V8 JavaScript engine [12].

For inter-peer communication, TCP was used to send and receive messages. Facilities for using TCP asynchronously came from the *net* module of the standard library of Node.js [13]. Messages between peers themselves and between peers and the catalogue were serialized using JSON (JavaScript Object Notation), a lightweight data-interchange format [14]. We chose this format as it is based on a subset of the JavaScript programming language, which makes it easy to use in a JavaScript-based application.

For increased productivity, a newer version of the ECMAScript standard JavaScript language was used than the current Node.js version supports. To run it, the source code was transpiled into an older version of JavaScript using Babel, a source-to-source compiler [15].

4.5 Peer discovery

The first thing that a peer does is register with the catalogue so other peers can find their address from said catalogue. To do this, a peer connects to a predefined TCP address and port and sends a command via TCP to register itself. When this peer closes their TCP connection, the catalogue removes them from the list of active peers.

When peers need to message other peers, they require the addresses and ports of other peers. To receive these, they send a message using their already existing TCP connection to the catalogue to request peers. Whenever the catalogue receives this message, it replies with the addresses and ports of other active peers.

4.6 Distributed random number generation

As a game with this ruleset requires random factors to spawn both players and coins, we decided to focus on the system discussed in chapter 3.3, generating random numbers using cryptographic hashing to guarantee ignorance.

We provided an example implementation of this, using the same TCP messaging system of the main game. This system is not resistant to most attacks, as it is not tolerant of

faulty protocol implementations. In a real-world system, blacklisting and fault-tolerance would have to be added. As it stands, when a number is generated, it is guaranteed to have been generated in distributed consensus. However, attackers can disable random number generation all together.

From this example implementation, it turns out that generating random numbers in this way is heavy on network traffic, so the scalability of this system is questionable. As it is implemented, every peer needs to maintain a TCP connection to every other peer, which means that network traffic increases exponentially as peers join.

4.7 Interaction resolution

We created the main ledger system for interaction resolution and verification in the example game. In this system, every peer keeps a ledger of all of the actions they have received. To retrieve the current state of the game, every action is iteratively applied to an initial state, finally yielding the up-to-date state. When an action is received from a peer, it is decorated with that peer's identifier and the current time, as leaving these parameters to a peer to generate can cause cheating vulnerabilities, such as acting as another peer by faking the identifier, or sending a false timestamp to be able to move faster or spawn coins faster.

In this system, actions are verified by utilizing this ledger. We encode the rules of the game into a function to verify a given action. This function uses previous actions to see if the current action can be considered valid. If this game was to be built fully, these actions would also require integration with a protocol like Lockstep or Asynchronous Synchronization to avoid some of the problems discussed.

This system, if implemented using Lockstep, as it turns out, is also heavy on network traffic, as every peer similarly needs to maintain TCP connections to every other to achieve consensus. Further research in splitting peers into groups and achieving consensus via these subgroups might yield scalability increases in both interaction resolution and distributed random number generation.

5 Summary

In this paper, we explored methods for preventing cheating in online peer-to-peer games. To understand where the challenges are, we dissected P2P architecture. We found that some proposed protocols have pitfalls in certain types of games. We also found that blockchain technology could become a viable method of cheat prevention in games in the future, if proof-of-work is replaced with a faster system. We show how pseudorandom numbers can be generated in a way where every interested party is involved in creating them, while their influence on the outcome is unknown until this number is created. Finally, we implemented random number generation in distributed consensus and encoded the rules of an imaginary game into a ledger system, using both systems to build a P2P game backend.

References

[1] Statista Inc., “Video game industry - statistics and facts,” [Online]. Available: https://www.statista.com/topics/868/video-games/. [Accessed 20 May 2017].

[2] J. Smed, T. Kaukoranta and H. Hakonen, “Aspects of networking in multiplayer computer games,” *The Electronic Library,* vol. 20, no. 2, pp. 87-97, 2002.

[3] C. Neumann, N. Prigent, M. Varvello and K. Suh, “Challenges in peer-to-peer gaming,” *ACM SIGCOMM Computer Communication Review,* vol. 37, no. 1, pp. 79-82, 22 01 2007.

[4] J. Yan and B. Randell, “A systematic classification of cheating in online games,” in *NetGames '05 Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, New York, 2005.

[5] C. Scardovi, Restructuring and Innovation in Banking, Springer International Publishing, 2016, p. 36.

[6] S. Nakamoto (pseudonym), “Bitcoin: A Peer-to-Peer Electronic Cash System,” November 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed 30 March 2017].

[7] I. Bentov, A. Gabizon and A. Mizrahi, “Cryptocurrencies without Proof of Work,” 22 June 2014. [Online]. Available: http://www.cs.technion.ac.il/~idddo/CoA.pdf. [Accessed 16 May 2017].

[8] B. N. Levine, M. Liberatore and N. E. Baughman, “Cheat-proof playout for centralized and peer-to-peer gaming,” *IEEE/ACM Transactions on Networking,* vol. 15, no. 1, pp. 1-13, 1 February 2007.

[9] R. L. Rivest, A. Shamir and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM,* vol. 21, no. 2, pp. 120-126, February 1978.

[10] wnoise (pseudonym) and Menkboy (pseudonym), “Distributed Random Number Generation - Stack Overflow,” 22 October 2008. [Online]. Available: http://stackoverflow.com/a/224067. [Accessed 15 February 2017].

[11] I. S. Dua, “Data Security in Cloud Oriented Application Using SSL/TLS Protocol,” *nternational Journal of Application or Innovation in Engineering & Management,* vol. 2, no. 12, pp. 79-85, December 2013.

[12] Node.js Foundation, “Node.js,” 2017. [Online]. Available: https://nodejs.org/en/. [Accessed 22 May 2017].

[13] Node.js Foundation, “Net - Node.js Documentation,” 2017. [Online]. Available: https://nodejs.org/api/net.html. [Accessed 22 May 2017].

[14] ECMA international, “The JSON Data Interchange Format,” October 2013. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. [Accessed 22 May 2017].

[15] S. McKenzie, “Babel · The compiler for writing next generation JavaScript,” 2017. [Online]. Available: https://babeljs.io/. [Accessed 22 May 2017].