A Blockchain-Based Solution to High-Volume Web Scraping With Smart Contracts on Ethereum

DANIEL KASTENSSON FAN
Abstract

Since it is difficult to protect servers from high-volume scraping, a new way to reduce excessive requests is needed. Using rougher methods such as rate limit or IP control mechanisms are not sufficient. In this report we propose a new solution to counter high-volume web scraping with blockchain technology. We create a cryptographic algorithm and use it on a mobile device to communicate with an Ethereum network with the purpose to control server access. Our studies seem to indicate that blockchain technology on mobile devices has potential to limit the way information is accessed. Furthermore, blockchains have potential to act as an additional security layer rather than simply a network solution. To determine the practical effectiveness of this solution, more studies are needed.
**Abstrakt**

Acknowledgements

I wish to thank Mathias Lövehagen Ekstedt for being my supervisor and guidance throughout the thesis. Many thanks go to Gabriel Aguilar-Svensk at The Mobile Life AB for his invaluable advice and help throughout this master thesis project. Thanks also go to The Mobile Life AB for allowing me to use their office in both Stockholm and Singapore. Alexander Papageorgiou has been helpful with both Ethereum and Solidity and for that I would like to thank him. Finally, I wish to thank my parents for never pushing me to do something specific and allowing me to develop my own interest.

Stockholm, 2018-07-20
Contents

1 Introduction 1

2 Web Scraping 3
  2.1 Limitations ......................................................... 6
  2.2 Requirements ....................................................... 6
  2.3 Current methods .................................................... 8
    2.3.1 ShieldSquare .................................................. 9
    2.3.2 Distil Networks ............................................... 10
    2.3.3 Alibaba Cloud Web Application Firewall ...................... 11
  2.4 Blockchains ....................................................... 13
  2.5 Comparison ........................................................ 15
    2.5.1 ShieldSquare and Distil .................................... 15
    2.5.2 Alibaba WC .................................................. 16
    2.5.3 Ethereum ..................................................... 17

3 Theory of Blockchains and Smart Contracts 19
  3.1 Fundamentals of the Ethereum network ......................... 21
  3.2 Properties of a blockchain network ................................ 23
  3.3 Blockchain relevance to the project ............................. 31
  3.4 Smart contracts and autonomous processes ...................... 34
  3.5 Hash algorithms and cryptography ................................ 38

4 A New Blockchain-Based Approach to Web Scraping 41
  4.1 Blockchain disposition .......................................... 44
    4.1.1 Deploy and use a contract .................................. 46
  4.2 Scraper mitigation algorithm ..................................... 48
  4.3 Smart contract development ...................................... 52
  4.4 Client side development ......................................... 54
    4.4.1 Blockchain connection setup ................................. 54
    4.4.2 Wallet setup ................................................ 56
    4.4.3 Read states with contract calls ............................ 57
    4.4.4 Write states with contract calls ........................... 59
    4.4.5 Generate dynamic solutions ................................ 60
  4.5 Server side development ......................................... 63
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABI</td>
<td>Application Binary Interface</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DAO</td>
<td>Decentralised Autonomous Organization</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>ETH</td>
<td>Ether (cryptocurrency)</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ID</td>
<td>Identity</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>POC</td>
<td>Proof of Concept</td>
</tr>
<tr>
<td>PoE</td>
<td>Proof of Effort</td>
</tr>
<tr>
<td>PoW</td>
<td>Proof of Work</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>TML</td>
<td>The Mobile Life (company)</td>
</tr>
<tr>
<td>TTP</td>
<td>Trusted Third Party</td>
</tr>
</tbody>
</table>
1 Introduction

Today, it is difficult to imagine a world without the internet. The openness of the internet makes it possible for anyone to access and use the information available. To facilitate information access, Google is using crawling and indexing to systematically present web pages in its search results [1]. However, information that is open to the general public through third-party services can be exploited. Web scraping is when an entity or organization, sometimes illegally, targets a web site to copy large amount of valuable information. A serious issue is when the goal is to request millions of flight prices every day through scraping. Scrapers do not only entail additional economical costs for booking systems but the scraped information can be used by competitors for pricing. Some actors even take advantage of large computer networks for the purpose of web scraping. Unfortunately, the current internet protocol and firewall methods are not sufficient. New solutions and strategies are needed to counter web scraping.

For The Mobile Life (TML) and its customers, scraping is an issue which is difficult to entirely prevent. Not only is the act of scraping accessible to perform, but it can in some regards be seen as an abuse of current internet protocols. For TML, the problem is when there are high-volume requests to servers (with e.g. flight data) in a short period of time. The consequences are persistent and pervasive: unwanted traffic and additional costs. The costs are both direct, since another party has to provide a booking service for a fee, but also indirect when malicious parties extract flight prices on a continuous basis. The flight data can then be used by competitors to undercut prices, if only by a small margin. The way the data is being scraped is by using the Hypertext Transfer Protocol (HTTP) to send GET requests which retrieves information from a server. In normal scenarios, it is simply a tool to fetch data from a server. In this case, these are not normal interactions but large scale operations with a goal of harvesting as much data as possible, with as little effort as possible. Since the scrapers perform their attacks from computers and the flight services are mainly accessed from smart devices, the solution will have to be mobile-based while mitigating attacks from scrapers.

We wish to evaluate the feasibility of using blockchain technology to make the interactions between a client and server more restrictive. Optimally, this
should be done without affecting regular user interactions, i.e. non-scraper clients. We do the following: setup a blockchain, create a verifying smart contract, and develop a cryptographic algorithm to be used by smart devices. In theory, it is supposed to work such that when a client is requesting data, requests should go through the blockchain before it is passed on to the server. Data is then unidirectionally passed on from the server to the client, i.e. the client is never supposed to directly communicate with the server. The blockchain can then act as a filter against scraper attacks since it will drop requests that does not contain a solution to a cryptographic puzzle. Due to the inherent property of hashing, there is a lower limit of how fast such solutions can be solved and generated. This is the basis of how high-volume requests are to be mitigated.

The purpose of this study is to develop a new method to limit high-volume scraping using blockchain technology. The research question is whether it is feasible at all for mobile devices to use blockchain technology to limit high-volume scrapers originating from computer networks. The security mechanism is instructed and enabled by a smart contract, which will be created and programmed in this study. In Chapter 2 we present web scraping and its limitations and requirements in this project. Chapter 3 presents theory of blockchains and smart contracts and how it can contribute to solve our specific problem. The new blockchain-based approach and the proof of concept (POC) for mobile devices is presented in Chapter 4. In Chapter 5 we test and analyze our solution. Lastly, the conclusions and summary of the results are found in Chapter 6.
2 Web Scraping

The principles of web scraping is to extract unstructured data and save it in a structured format. A simple example is HTTP method requests of the type GET. Excessive and malicious traffic from computer networks can create significant costs of operating mobile services and systems for a third party. Scrapers are in particular interested in information that can be sold to third parties, in this case flight prices. In this project, we denote the request method which returns flight prices as $GetAvailableFlightPrice$. Since we have data logs of a server containing flight prices, we can fetch the request history by using a log parser. The logs are then extracted and sorted to get a better picture of scraper behaviour. This means that we can structure the data by type of request and request size. By understanding such, we can become better informed when designing a countermeasure. An example of a raw and unstructured log can be seen in Table 1, where we have different IP addresses and request types, which originally are ordered by their timestamp.

Table 1: Example of unsorted server requests, likely genuine

<table>
<thead>
<tr>
<th>IP address</th>
<th>Request type</th>
<th>Size (bytes)</th>
<th>Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>132.2.123.1</td>
<td>GetAvailability</td>
<td>4661</td>
<td>(iPhone/iOS)</td>
</tr>
<tr>
<td>132.2.123.2</td>
<td>UpdatePassengers</td>
<td>513</td>
<td>(iPhone/iOS)</td>
</tr>
<tr>
<td>132.2.123.3</td>
<td>GetAvailableFlightPrice</td>
<td>3203</td>
<td>(iPhone/iOS)</td>
</tr>
</tbody>
</table>

A good indicator of a genuine user, albeit neither a guarantee nor exhaustive, is if the requests are somewhat varying in type and size, and not over an excessive amount of time. The latter is of course subjective and difficult to build a solution around. An even better indicator is the agent identity, which is supposed to be a smart device. This is spoofable by scrapers. If the field is empty, this is potentially a scraper, since the service is normally accessed through mobile devices. This is due to the fact that the specific service we are looking to protect is access through mobile applications of the flight companies. As a result, it is very suspicious when a field clearly belongs to a computer such as Mac OS X Version 10.12.6 (Build 16G1212). In Table 2, we have pulled information from a specific IP address which exclusively gets the current flight price to a destination.
Note that this request involves many different settings, including both time
dates and flight routes, which explain the varying size of this request. An-
other indicator, which cannot be seen in these tables, is the timestamp.
Scrapers usually persist over a longer period of time, although this is not
always the case since they are instructed by people. It is important to em-
phasize that a naïve solution to counter high-volume requests of Table 2 is
to restrict certain Agent fields. In practice, this is very easy for the scrapers
to circumvent through spoofing and consequently not a sustainable solution.
To get a better perspective of requests in general, we could sort by total
amount of requests by each IP address, as in Table 3.

<table>
<thead>
<tr>
<th>IP address</th>
<th>Request type</th>
<th>Size (bytes)</th>
<th>Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>154.3.123.1</td>
<td>GetAvailableFlightPrice</td>
<td>837</td>
<td>(null)/(null)</td>
</tr>
<tr>
<td>154.3.123.1</td>
<td>GetAvailableFlightPrice</td>
<td>11218</td>
<td>(null)/(null)</td>
</tr>
<tr>
<td>154.3.123.1</td>
<td>GetAvailableFlightPrice</td>
<td>5437</td>
<td>(null)/(null)</td>
</tr>
</tbody>
</table>

It is not definite that all agents with a higher request count are scrapers
but one could question the legitimacy of an IP address, if the one with most
amount of requests and highest average request size exclusively fetches data
composed of price information. Another naïve solution could be to block
that IP address, but this is once more easily circumvented by the scraper
network by changing the IP address. Ultimately, we want to counter high-
volume scraping in real-time and not let it continue over a longer period of
time. A previous and temporary countermeasure can be seen in Fig. 1. This
countermeasure, based on a commercial service, suffered from a great many of requests in relation to time.

Even though the result of the protection mechanism in the short term was significant, it took a few hours before it had an impact. Furthermore, sometimes a proven mechanism stops to be sufficient, as can be seen in Fig. 2, which once again is a commercial service used to counter scrapers.

Figure 1: An existing scraper protection service which results in a clear drop in amount of requests.

Figure 2: An existing solution which only temporarily managed to filter against high-volume requests.
It should also be noted that both Fig. 1 and Fig. 2 are two distinct servers and companies, which means that they are targeted by different kinds of scrapers. Due to many uncertain parameters, including lack of information, it is not fair to compare these with each other on a one-to-one basis. Since the scope of this report is not to mainly evaluate existing scraper protection mechanisms, we will just conclude that current systems have a varying form of success and speed.

2.1 Limitations

There are factors which need to be taken into consideration when deciding on a practical and viable approach. The limitations are derived from commercial demand from TML and from previous observations by looking at the request behaviour through old logs. They are as following:

1. Bot requests, by themselves, cannot be separated from legitimate users, i.e. thorough authorization of users is not the focus in this project but instead try to prevent high-volume requests from individual clients

2. Bot owners have access to multiple networks, computers and personnel to administrate attacks, which means that a mitigation strategy must not be easy to circumvent from the client side (e.g. IP change)

3. Bot behaviour is easy to spot due to a comparatively large amount of traffic in relation to the average user; but both difficult and cumbersome to shut down entirely

4. Bot behaviour that is not easy to spot, i.e. not excessive, is not within our scope since the focus is high-volume requests which entails additional costs

2.2 Requirements

To work as a POC with minimal capability of turning into an industrial and commercial solution, there exists some fundamental requirements that must be satisfied. The requirements can be seen in Fig. 3.
Figure 3: The requirements for the scraper protection module.

1. The solution to unwanted bot behaviour should not introduce additional complexity for the user, i.e. all security schemes will happen in the back-end (data access layer) and not rely on any challenges or tests which can affect the user experience in any way such as with a challenge-response test (e.g. CAPTCHA); the solution has to be self-reliant and self-serving, isolated from any user input.

2. The solution has to follow the principle of least authority, i.e. the client should not have access to more information than needed to retrieve the resource; we would like to conceal and obscure as much data as possible.

3. The solution has to be controlled centrally, on the server side, and decompilation of the mobile application should not reveal the entire logic of the security mechanism; the filter and logic, however, should not be present on the server side (i.e. should be on the blockchain).

4. The solution has to be scalable and permutable, with the ability to in real-time be able to revise the permissions and logic structure on the network to efficiently mitigate high-volume scrapers.

5. The solution has to be compatible with mobile devices and should not affect the service level to users in any way, e.g. slower access to the service.

Requirement 1 is based on the principle of no additional and significant latency of back-end solutions to the server, which will have to be evaluated. The basis of requirements 2 and 3 is derived from our proposed solution.
where the less a user knows about the logic, the more difficult it will be to circumvent a solution. Requirement 4 is an important factor of TML, since new back-end solutions should preferably have the ability to deploy and scale for new customers as times goes on, and especially be able to be replicated for new business. The last requirement 5 is the basis of the company since a significant share of their business is catered to large airlines in need of mobile solutions and server architecture. These requirements are not based on any previous solution or research, only potential components of a POC. Furthermore, we do not know if the requirements are complete or sufficient.

2.3 Current methods

It exists some feasible network and higher-layer techniques to make it more difficult for scrapers. These techniques are to be described below. For customers that suffer scraper attacks there exist commercial solutions such as ShieldSquare [2], Distil Networks [3], and Alibaba Cloud Web Application Firewall [4]. We will evaluate these solutions and discuss their capabilities in relation to the requirements for our solution. The techniques will be presented first and then each service will be discussed. The problem of each technique is for countering scraping in general, many of which will violate the requirements of our POC. Thus, the following techniques are insufficient for the proposed solution requirements.

1. **Rate limit** – controls the rate of the traffic which helps to prevent DoS (Denial of Service) attacks and/or limit the rate in which information can be fetched
   **Problem:** affects every user and may cause service delay if used excessively

2. **Block IP address** – drop clients that abuse the service
   **Problem:** does not work when large computer networks can change and spoof IP addresses; it is defeated by proxies

3. **CAPTCHA or email signup** – forces verification of all users to confirm their identity, at least temporarily
   **Problem:** allows for a less smooth user experience, may decrease the
conversion rate of some user-centric consumer applications and ultimately cannot stop scraping

4. **Client puzzle protocol** – deters abuse by forcing clients to solve a computational problem and return a solution
   **Problem:** mainly designed against connection denial attacks and makes retrieving information more difficult for every user, and not specifically users with unwanted behaviour

5. **Detention based on cookies** – a longer delay on new cookies and a cap of the amount of requests each are able to do could control the data access
   **Problem:** this and any other client-centric solution suffers from the inherent weakness that it can be easily modified and accessed by the client: specifically, alter or even delete a cookie such that the restriction is circumvented completely, which means that a solution cannot allow clients to have control of both the logic and ticket used to access the resource

### 2.3.1 ShieldSquare

Their approach to bot protection is to build signatures for each unique visitor [2]. This should not have an impact on genuine users and performance. In our case, this violates project limitation 1 since the assumption of differentiating between users is not a priority. Their solution is both integrated with the web page being protected, and this is then evaluated on a cloud engine and if it is evaluated as friendly (user or search engine crawler), then the API response code allows the request. For our proposed solution, this violates requirement 3 since we are not interested in protection mechanism entirely on the server side. The user analysis is done in several layers

- **IP tracking:** network forensics based on data such as geo location, ISP information, and connection type, but also whether it comes from a proxy or not
- **Behaviour analysis:** bot behaviours are significantly different from a genuine user; typical users have certain behavioral characteristics in terms of page views per session, time spent on each page, and frequency of repeat visits
• Collective intelligence: data that is gathered across sites are shared with other websites to be fully utilized to identify bots; data from a third party fraud intelligence can also be used to keep track of flagged IPs and devices to counter attacks.

This solution is a lower-layer approach (relative to ours) with a clear goal of authenticating users to let certain requests bypass their bot filter, as can be seen in Fig. 4. The proposed solution in this project is also partly based on behaviour analysis, albeit not as extensive with the amount of parameters as theirs. The main issue, which we are trying to explore, is the opportunity to build a solution which does not need to be dependent on protecting the server. It is very difficult to judge the success or efficiency of ShieldSquare from a mere look on their approach. But the proposed solution should not be dependent on classifying users nor should it have a server-sided dependency. Furthermore, we have no interest in challenging a potential bot with CAPTCHA.

Figure 4: Real-time bot protection (www.shieldsquare.com/how-it-works)

2.3.2 Distil Networks

Validating a browser and its JavaScript engine is once again the way to detect scrapers and this is in direct violation of requirement 3. Similarly, when a device is roaming the website, it collects and analyzes data to identify malicious behaviour. It is difficult to tell what this kind of analysis actually
means but an educated guess could be that it partly tries to look for certain behaviours, like any other solutions. Furthermore, their main approach is different kinds of CAPTCHA responses which they have designed. Even if it is assumed these have zero false positives, it is not our intention to base a protection mechanism on challenges, which violates requirement 1. We are looking to build a reliable and robust filter, which is open to anyone but still puts a dynamical limit on the amount of requests within a certain time frame.

When it comes to identification of threat agents, in Fig. 5, Distil includes some of our suspected sources, requirement 5. Specifically, this means that the proposed solution has to resist attacks from not only zombie farms of PCs and mobile devices but also emulators. This means that if the solution solely relies on the computational and hashing capacity of mobile devices, or lack thereof, then a personal computer could very well be equipped to crack such protection mechanism.

Figure 5: Blocking bots from a website (www.distilnetworks.com/superior-technology)

2.3.3 Alibaba Cloud Web Application Firewall

This web application security firewall is not entirely focused on scrapers and bots but web attacks in general [4], such as injections and exploits of other vulnerabilities. For this reason, they have similar strategies of detecting il-
legal requests and their solution is partly based on modifying the website’s DNS record. In comparison to ShieldSquare, this solution seems more focused on the application layer instead of the network layer. When it comes to efficiency for protection against scrapers, it is difficult to say whether a more lower-layer solution is more suitable or not. What can be seen from this approach is that it tries to defend against flooding, pinging attacks, and other types of intrusion. A main concern is that it heavily relies on firewall techniques such as DoS protection and not specifically scrapers.

The proposed solution in this study has neither requirements nor any intentions to protect against DoS attacks. Like previous alternatives, it is almost impossible to evaluate their solution in practice without using it and understanding the approach in-depth. We can only state that it may work, but it is not apparent that it is tailored to our end specifically. Observing Fig. 6 we can notice that it has very similar characteristics to current methods, already discussed in Section 2.3. This may not come as a surprise considering the service being offered has the word firewall in it, something which is already deemed insufficient.

Figure 6: Scenario diagram (www.alibabacloud.com/product/waf#scenarios)
2.4 Blockchains

Blockchains as a whole possess certain attractive properties that makes it suitable for the project and with strong potential to satisfy the requirements. Generally speaking, a blockchain has the following network behaviours which addresses the requirements in Section 2.2, respectively.

1. Interaction with the blockchain can be completely autonomous in line with a specified network protocol; it is possible to in advance create every component necessary to connect and communicate with a blockchain as well as execute all of this in the back-end.

2. While anyone is free to participate in public blockchains, agents are also allowed to both read and write to the network [5]. In contrast, a private blockchain can enforce eligibility, i.e. which are to be allowed to participate and to what extent information should be public; it is feasible to conceal all but the bare minimum for a device to interact with a blockchain.

3. A corollary of a blockchain being private is that any logic that specifies whether a user is eligible to perform certain operations or not, can be concealed entirely from public display.

4. A blockchain is by definition scalable, the issue is to what extent with respect to throughput and latency [6]; furthermore, permissions and contract requirements can be revised on smart-contract based networks [7] to allow access for genuine users.

5. The support for mobile blockchain use is limited but evolving and in principle, this project could treat mobile devices as a light version of edge computing in a decentralised framework [8]; consequently, it must be emphasized that mobile support is even less developed than many blockchain networks, and for that reason there are neither guarantees nor support, apart from the open source community, that our proposed solution will work as intended or even at all.

Blockchains may be well-equipped to achieve its purpose of defeating high-volume scrapers. For the project specifically, this could prove to be a challenge since blockchain technology available to the public is still not very es-
tablished. Furthermore, the mobile client is not a prioritized issue within the biggest smart-contract network Ethereum. In this report the mobile client is sometimes referred to as a light client. The opposite is a full client which is run from a computer.

In January 2018, when this project was initiated, basing a blockchain on Ethereum was the only viable option with a light client. Since it was the only publicly available library which contained a basic and functional POC, there were no other options. In this project the blockchain act as an additional security layer between the client (user) and the server, while it is ideally accessible from both enterprise and public networks, see Fig. 7 as a concept. It is of interest to investigate whether or not a blockchain layer has potential as a layer of security to reduce the amount of unwanted high-volume requests.

Figure 7: Blockchain as a separate network, on top of the traditional security layer, such that information retrieved by the cloud network has to initially pass through the blockchain [9].
2.5 Comparison

In Section 2.3 we treated some of the more practical and direct countermeasures to scraping. Methods that can be seen as rougher and not precise enough. Here, we will compare a few commercial services and how they satisfy our requirements. Their approach is, at least in theory, significantly more refined than e.g. simply blocking IP addresses. Each alternative and how they comply with the requirements can be seen in Table 4.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>ShieldSquare</th>
<th>Distil</th>
<th>Alibaba Web</th>
<th>Ethereum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Complexity</td>
<td>No</td>
<td>No</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2) Obscurity</td>
<td>✓</td>
<td>✓</td>
<td>No</td>
<td>✓</td>
</tr>
<tr>
<td>3) Robustness</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>✓</td>
</tr>
<tr>
<td>4) Dynamicity</td>
<td>Questionable</td>
<td>Questionable</td>
<td>No</td>
<td>✓</td>
</tr>
<tr>
<td>5) Compatibility</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Possibly</td>
</tr>
</tbody>
</table>

There are three things we can distinguish from our analysis:

1. ShieldSquare and Distil have a similar adherence to our required solution
2. Alibaba Cloud Web Firewall is the least useful for this project
3. Blockchains, more specifically Ethereum, has certain properties that may satisfy all criteria

These will be discussed accordingly.

2.5.1 ShieldSquare and Distil

Their properties are deemed to be similar to each other, with respect to the requirements. Firstly, the complexity (requirement 1) is similar in the sense that applications can be challenged through actions such as CAPTCHA. Even if these are very well tuned without any zero positives, this is not the approach we are looking to take since it requires application input and hence is not aligned with the requirements. Consequently, both solutions are completely on the server side which means that it is difficult for a malicious user
to access the logic or algorithms being used. On the other hand, we are not looking for a solution that is present on the server side as both of these are since they intercept the communication between client and server.

For dynamicity (requirement 4), it is not entirely clear if it would have the intended effect on rapid response notice. To mention an example, Shield-Square has an active mode protection [10], which means that it is possible to choose one of two methods. The first one is called Real-Time Protection and lets one make synchronous calls to their API to take action against bots of different categories. The other is called Feed-Based Protection which allows for asynchronous calls to be made. Similarly with Distil, they have different protection settings that lets you configure what actions to take in response of bot requests. What these two solutions have in common is that neither are revising the permissions of an absolute amount of requests, but merely responds in different ways depending on the classification of the bot. In the solution, the sole goal is to eliminate the possibility of making excessive queries and to this end neither is a particularly good fit.

### 2.5.2 Alibaba Web

The first requirement this solution satisfies is that none of the logic happens in the front-end and will not challenge the user in any visual way, in the very same a way a router would drop a packet. Also, it should not be an issue with mobile applications, which the other established solutions most definitely supports. What it does not support, however, is the following requirements

1. Obscurity (requirement 2): the client may be very well exposed to the constraints of this solution, as well as how it works with respect to scraping; the product website [1] is very clear with what kind of measures are taken and for that reason we will not reach sufficient obscurity when protecting the data

2. Robustness (requirement 3): following the previous requirement, the logic is mainly based on interaction with the server. Furthermore, common network rules may very well be circumvented through trial-and-error or traditional brute force attacks, which violates the requirement
3. Dynamicity (requirement 5): even if firewall rules are changed in real-time, it will not be precise enough or sufficiently effective for dealing with high-volume scrapers; the approach results in weaknesses discussed in Section 2.3 and 2.5

The combination of the inherent properties of a firewall, and points 1 to 3 above, makes this service the least suitable for this project against high-volume scraping.

2.5.3 Ethereum

As discussed in Section 2.4, blockchains are susceptible to be shaped in such a way that requirements 1 to 4 are satisfied. The issue at hand is from a practical point since many technical benchmarks are neither transparent nor known for blockchains in general. Factors such as throughput, latency, reliability and security, to mention a few, are still yet to be discovered and tested on a broader scale. A reason for this could be that blockchain is still an emerging technology and not properly established yet. It is constantly changing and evolving.

It must be stressed that the biggest uncertainty is requirement 5. This is somewhat of an unusual circumstance since most, if not all, commercially-orientated IT solutions has support for mobile web communication and applications since it is more or less ubiquitous. In our case, requirement 5 and mobile support is not only the most fundamental and important part for our project, it is paradoxically the very requirement that has the highest uncertainty to succeed. To base our project on a blockchain solution could very well be rewarding if it turns out to have industrial potential but it could just as likely not work at all due to its technological immaturity. We could say that we base our POC on technology that in itself is a POC, which means that it is more likely to contain certain flaws.

We base our developed solution on the official Go Ethereum library, Go-Ethereum [11] and specifically use their Light Ethereum Subprotocol [12] for mobile devices, to communicate with the private blockchain. The assumption to investigate is that data retrieved through a centrally controlled private blockchain, with the help of smart contracts, can be accessed in an improved
manner compared to today. The significance of this research is that if it works, it is not limited to protecting server APIs and other semi-public information but to any industry that is dependent on data being accessible by consumers from interactive on-demand services.
3 Theory of Blockchains and Smart Contracts

In the past years we have seen a huge increase of global use of mobile devices compared with desktop computers. Likewise, for many developing countries opportunities to access the internet has increased due to lower costs and better access. For many people, the first interaction with the World Wide Web is through a handheld device. In the case of internet, the availability and openness has a backside when it comes to vulnerability and new attack vectors. Before digitization, companies relied on paper and physical records containing information. An example is ledgers with credit and debit cards. Since the internet is decentralised, it is also difficult for information to be stopped. This is where blockchain can improve both openness and data confidentiality through mathematical protocols. A blockchain protocol is, ideally, defined in such a way that agents are inclined to take the most beneficial decisions. Not only for themselves but for the network as a whole. Moreover, a blockchain protocol can be used for more than its network properties. An investigation of its potential as an additional security layer between users and servers is both interesting and promising. This is something which could help web services to defend against unwanted scrapers. Because of this, the company The Mobile Life (TML) in Sweden has initiated an investigation of blockchain technology to counter scraper attacks. Specifically, the solution is intended for applications accessed through smart devices.

With an increased blockchain technology adoption, the very same internet revolution may happen once again. But now it is not about enabling communication but distribution of data in such a way that no central node is assigned sole responsibility over both data and computations. In one way, every node can both be a client and server at the same time. In other words: there is no longer a single point of failure. It could be argued that blockchain technology in its very essence is a decentralised and growing, distributed ledger. Protocols which operate on it, like the Ethereum Protocol, define read and write operations of actors on the network. It is suggested that the blockchain network and the protocol in this project is treated as an additional security layer. Due to the scope of the project it is easier to see it as an extension of the current OSI model rather than a, potentially, global network. The main blockchain advantage to exploit is its ability to store immutable programs and processes, in the form of smart contracts. The contracts are
interacted with through the blockchain and differ from traditional requests on the internet which makes scraping possible. Since the blockchain in this report is private, a local network only for the company, it is also controlled by a single authority. This may contradict the philosophy of blockchains but its importance as a potential security layer against scrapers must be stressed.

For blockchain technology as a whole, it exists research in everything from cryptocurrencies to security mechanisms. As the basis of and currently largest cryptocurrency by market capitalization, the Bitcoin protocol is mainly based on a peer-to-peer electronic payment system where third-party trust is replaced by digital signatures of cryptographic proof [13]. Instead of trust in unreliable identifiers such as IP addresses, this type of peer-to-peer network is maintained through CPU power and calculations of cryptographic puzzles. Trust in this case is replaced by asymmetric cryptography and proof-of-work which is a mathematical way to verify valid transactions. For a computer, this is sufficiently difficult to calculate but easily verifiable. In the domain of distributed ledger technology both Bitcoin and Ethereum are the two biggest protocols based on market cap as of today. There are also other types of networks whose goal may differ from each other. Since blockchain technology has a diverse set of applications, there are different reference models and actors in such networks, and one could divide the different types of technologies into two groups: a) transaction-based networks, and b) transaction-based networks which include smart contracts, and consequently support operations on the blockchain [7]. This is something which may change in the future, though, since e.g. Bitcoin is planning to implement such as well [14]. An example of the latter type is the Ethereum network. It attempts to build a generalized technology on which all transaction-based state machine concepts may be built and at the same time provide end developers with the end-to-end system for building software on a trustful object messaging compute framework [15]. Like most public blockchains, Ethereum can guarantee high integrity and availability. Despite that, this comes with a price of privacy since calculations and transactions are broadcast, with opportunities to intercept and analyze the data [16]. There are both benefits and consequences of this. An important corollary is that the contract code (and all logic therein) is public and even if it is in the form of byte code there are tools dedicated to the analysis of such. In practice, few people would trust a contract on the blockchain without seeing its source code.
Ethereum is a blockchain network which is different from Bitcoin. There is some overlap between the networks, such as cryptographic and communication principles, while other things are unique such as gas. Despite this project not being based on Bitcoin, the way its network acts and agents operate is sometimes applicable to Ethereum. In such case, Bitcoin will be used as a reference for practical examples.

3.1 Fundamentals of the Ethereum network

Before proceeding with discussions of the theory behind blockchain technology, it would be suitable to clarify terms that can be considered some of the fundamental building blocks of a blockchain network.

- **Block**: a data structure that contains transactions and other identifiers of a network.
- **Miner**: an entity, e.g. a computer controlled by a person or a cloud service, which verifies and processes unconfirmed transactions on the blockchain.
- **Genesis block**: the first block in a blockchain, which among other things, includes network information such as mining difficulty and other identifiers.
- **Network protocol**: the rules in which the network operates, e.g. how does miners verify a valid transaction and prevent double spending. In a private Ethereum blockchain, the protocol Proof of Work (PoW) is used.
- **Blockchain**: a transparent, distributed, append-only ledger which consists of blocks and with infrastructural and operational rules according to a network protocol.
- **Cryptocurrency**: a digital asset on a blockchain network, not necessarily monetary as e.g. bitcoin on the Bitcoin network. On the Ethereum network, ether may be used as a means of payment but also perform calculations and storage operations through interaction with smart contracts.
• **Cryptocurrency wallet**: each user has to possess a wallet, an Ethereum address, to receive and send a cryptocurrency, and this is based on a key pair consisting of a public (64 bytes) and private (64 bytes) key. The last 20 bytes of a hashed public key is referred to as an Ethereum address.

• **Public key encryption**: the principle of asymmetric cryptography with a pair of keys can be seen in Fig.[8]. In the sense of cryptocurrencies, a public key can be seen as the address to a wallet. In traditional banking, this would be the bank account number. If funds are transferred to a wallet address, a transaction is valid if it is correctly signed using the corresponding private key.

• **Ether**: on the Ethereum network, the unit of value used to access computational resources is called ether. Apart from simply a currency, this is the unit of value often considered the ‘digital oil’ of cryptocurrencies due to its ability to invoke computations and change the stored information on the blockchain.

• **Smart contract**: a piece of code written as instructions to the Ethereum Virtual Machine. A contract also works as an account and can both receive and send ether, according to its instructions. The smart-contract oriented language used for development is called Solidity.

• **Gas**: a smart contract which performs operations and storage on the blockchain comes with a cost at both deployment and execution of the contract. Instead of estimating such costs in ether, which can fluctuate significantly against other FIAT currencies (USD, EUR, JPY etc.), it is rather denominated in gas. For example, a transaction to another wallet costs 21,000 gas just for performing the operation.
3.2 Properties of a blockchain network

**Blocks in a blockchain.** According to [18], a blockchain is a digital information storage method capable of recording data through a logbook approach, composed of a cryptographically linked chain of blocks of data. The essential characteristics are: (1) ordered, (2) incremental, (3) sound and verifiable, and (4) digital. An important distinction is made between intrinsic characteristics of a blockchain, and other properties that arise through distribution, communication and agreement protocols. Some of these additional characteristics are the way a blockchain is distributed and mutable by PoW. As briefly mentioned in Section 3.1, a block contains several important identifiers. The three major components are [18]

1. **Block-Data:** messages or transactions
2. **Chaining-Hash:** copy of the hash value of the immediately preceding block
3. **Block-Hash:** the hashed value of the result when adding the previous two components with each other

A very simple example of the block-hash is when we let block-data and the
chaining-hash be the strings \textit{KTH} and \textit{ElectricalEngineering}, respectively. In practice, the chaining-hash will itself be another hash while block-data could contain several transactions (hashes). In Fig. 9 we see the Keccak-256 hash result of our example. Since hash functions are one-way functions, it is easy to verify that the chaining-hash is made up of the two strings appended. It must be noted that in this example we only hash individual strings, while in other cases it can be beneficial to hash other simple or composite hash values for verification, privacy or security reasons.

Figure 9: An example of when a block-hash is a Keccak-256 composite hash value of the strings ‘KTH’ and ‘ElectricalEngineering’.

\textbf{Agents on a blockchain.} Each actor on a blockchain may have one or several roles when it participates in network activity. Nothing stops ordinary users to utilize the network and mine at the same time. It is also assumed that miners do not follow a network protocol and consensus out of ideology but instead tries to maximize its financial gain \cite{10}. From a game theoretical perspective, it could be argued that an important component in a successful blockchain network aligns the interest of the individual miner with the network as a whole. In this project, a user is simply interacting with the network through a smart contract by sending a transaction that triggers certain execution on the blockchain. The miners process all transactions and change of states, and finally incorporates all these in a block.

\textbf{Blockchain overview.} The basic flow of a blockchain transaction can be seen in Fig. 10. For the project, some of the steps 1 to 6 are slightly modified and therefore each step is commented.
1. A smart contract is deployed on the blockchain and we trigger a function of that contract. If we do not use a private blockchain, this would cost ether. In the case of this project, all ethers gained through mining is only usable on the private blockchain, limited for company use.

2. Transactions will be written to a block, which contains information of users on the network and other identifiers that describe their behaviour.

3. Blocks are broadcast but since we only rely on one full client (computer) this property is not utilized to its fullest potential.

4. Transactions that occur on the proposed protocol and domain network will be validated, if performed properly, by a miner.

5. On a public and decentralised network, the chain is treated and assumed indelible. In the case of the project it is easy to restart the blockchain from scratch due to locally stored chain history and data.

6. A successful transaction, accepted by all network participants, will in
the project result in a token which can be used to access a resource.

**Openness of blockchains.** Blockchains can be either permissioned, where users allowed to participate in the network is limited by an authority, or permissionless like Bitcoin or Ethereum which are free for anyone to participate in. The two types of blockchains can be divided into the following groups:

- **Public:** since there is no central entity which manages the membership of the network, any peer can join and leave the network as a reader and writer at any time [21]. Openness here implies that written content is readable by any peer.

- **Private:** when parties trust each other, it could be sufficient to have a private blockchain in the same way one has a local area network. If the assumption is that writers mutually trust each other, then a database with shared write access is probably the preferred solution [21].

**Fault tolerance.** Centralisation in itself is a way to describe the communication flow in a network. An example of a central hub can be seen in Fig. 11 where every interaction within the network has to go through a single node.

![Figure 11: A network with star topology](image)

In contrast, a decentralised network has no such node. For blockchains, there are several types with different levels of privacy and centralisation. Blockchains in general can roughly be divided into following groups
• Decentralised: one major advantage of blockchains compared to other distributed databases is the integration of consistency and security through algorithmically enforced rules, which removes the the human factor from the equation [23].

• Centralised: a blockchain that is centralised has one or a few single points of failure. This means that a trusted party in this case will solely be the company computer itself, running the full node.

**Cryptocurrency and blockchains.** A factor which likely contributed to the emergence of cryptocurrencies, "virtual money", is blockchain technology. One of the fundamental issues which Bitcoin solved was the principle of double spending. How is it possible to make sure that two entities, such in Fig. 10, can transact and receive units of value without malicious interference from both each other and other external actors? According to [24], to understand a monetary system such as Bitcoin, it is necessary to combine knowledge from disciplines such as economics, cryptography, and computer science. Furthermore, a strict monetary policy needs to be regulated through a protocol and it is argued that a transaction needs to satisfy the following three requirements:

1. Transaction capability
2. Transaction legitimacy
3. Transaction consensus

Capability here means that payments can be submitted to the network successfully. In contrast to banks, where a central authority approves or disapproves a transaction, a new order is eventually communicated to the whole network. Legitimacy is the next step since there is always an inherent risk that nodes communicate fraudulent payment orders without any repercussions. Here we have two important issues: (1) determining if a transaction is initiated by the rightful owner, and (2) ensuring that a transaction message is not manipulated before being passed between nodes as in Fig. 12.
To solve this, asymmetric cryptography is used to guarantee legitimacy. The sender uses the recipient’s public key to encrypt a message, while the recipient can use their private key to decipher the message, as in Fig. 8. Furthermore, a user can sign a message using the private key to guarantee ownership of a message. This type of encryption is known as ”signature” [24]. So when Edith in Fig. 12 wants to make sure her message is not manipulated, she can sign each message with her private key. If she is the only owner of it, any other participant can use her public key to verify that it was her message indeed. This way other participants can reject malicious users with erroneous masquerading attempts. The final part is consensus. We assume a scenario where Edith sends two identical transactions within a short time which refers to the same units of bitcoin as in Fig. 13.
Both transactions could be propagated across the network at the same time and both would show a valid origin, i.e., transaction legitimacy. Depending on where each node is in the network, some would receive transaction A first while others would receive transaction B. Since only one of these transactions will be added to the blockchain, it will be the first confirmed block that includes any of these transactions. After such confirmation, the other transaction will be discarded and not processed.

**Proof of work and hash rate.** Both Bitcoin and Ethereum are using the consensus protocol PoW, a principle which originally may have been a proposed method to prevent spam [25]. The enabler of spam is its availability and cheapness to send but with proof of work (PoW) the sender has to perform certain resource-intensive computations, memory intensive operations or in some way post a bond for each message sent. Computations of these kind are quantified by its hash rate, which measures the number of times a hash function can be calculated per second. It is easy for an agent to calculate their own hash rate, but more likely to only be able to estimate the hash power of other actors on the network [26]. Furthermore, this hash rate is dynamical due to its hardware and software nature but also since its operators can quickly change the network to mine. It could be argued that a strong incentive to mine a specific network is the current profitability, i.e., rewards of finding a new block. Since networks are driven by honest miners to process transactions and find new blocks, it is necessary that the hash power of the majority network belong to this group of honest nodes.

**Blockchain vulnerabilities.** Before possible attack vectors and scenarios are discussed, it is important to distinguish between attacks on the network and entities which operate on the network. This distinction should be stressed since when the concept of bitcoin being hacked is mentioned, often in public media, it refers to an exchange or other type of middleman being defrauded in some way and not the network protocol itself. Many exchanges have been both targeted and breached, sometimes multiple times, and are a valuable target due its availability and concentration of wealth. There is also an indication that transaction volume of exchanges is positively correlated with experiencing a breach, in a study of how almost half of exchanges has been closed with customer account balances being wiped out [27]. It could be argued that the exchanges of today, which stand for centralisation and replication of old financial institutions, are not the most robust way to han-
dle and trade cryptocurrencies. Nonetheless, there are no other established alternative for the vast majority of retail cryptocurrency investors, seeking liquidity of these digital currencies.

An occurring and perhaps the most fretted attack today is the so-called 51%-attack, which means that a majority of the network’s hash power belongs to one or several malicious users which may in some way want to diverge from the blockchain protocol, such as legitimacy and consensus. A network which suffers from such attack can accept double-spend transactions as valid and even refuse to process certain transactions. In practice this means it is possible to impose censorship on certain entities. There are ways to combat 51%-attacks, by introduction of second cryptographic challenges such as two phase proof-of-work, or through other consensus mechanisms based on proof of stake which is not dependent on hash power. A version of the latter, Casper, is considered to be implemented on the Ethereum network due to additional security compared to PoW.

Privacy and integrity. From a privacy perspective, the benefits of an open and permissionless blockchain is the transparent and verifiable history of transactions to audit. This applies for global blockchains such as Bitcoin and Ethereum but is not always the case for private and permissioned blockchains. In this case it is not transparent to the whole network but for the party in control, i.e. the central authority. For the global blockchains one could argue that there is a necessity of not being completely anonymous due to anti-money laundering and know-your-customer compliance. On the other hand, the issue of concealing your identity on public blockchains has several potential solutions based on coin mixers, confidential security schemes, ring signatures and tumblers, to mention a few.

Even if integrity breaches could be considered a consequence of human fault, it is in the network engineers and technicians interest to make misuse as difficult as possible. In the case of a hospital misrepresenting information and manipulating sensitive information while not sufficiently concealing it, there is an apparent risk of it becoming public, especially if it is on a global chain. To this end, it should possibly be a weighted combination of off-chain storage (traditional server or database) and on-chain operations to minimize such risks of. On a larger scale, this is something the EU SUNFISH project partly tries to address with its Federation-as-a-Service which supports inter-
operability and cooperation between private cloud systems [35], albeit with a possibly better approach than previously mentioned. The project aims to achieve high data integrity among databases and for that reason blockchains are a suitable option for distribution of databases. As can be seen in Fig. 14, it is suggested to use a combination of a permissioned blockchain in the first layer and a traditional PoW based blockchain protocol in the second layer. A block in the second blockchain will be linked to a certain part of the first blockchain, which means that the immutable transaction hash will act as a forensic evidence that proves and validates the integrity of the data stored in the first blockchain.

Figure 14: A proposal of a blockchain-based database for a cloud federation platform [35].

3.3 Blockchain relevance to the project

A blockchain is used due to its properties and suitability to work as an additional layer of security. Mainly, we try to exploit the way it can keep track and react to external and internal input. In detail, it is the smart contracts that operates on the Ethereum network we want to leverage. Contracts,
which can be treated as autonomous processes in the sense that they react
the same way to each trigger, independently of which entity interacted with
it in the first place. When triggered with certain parameters, i.e. solution to
a puzzle, the contract will reward the sender of the trigger with a token.

It could still be argued by blockchain sceptics whether or not it really is
necessary for most organizations. In [21] it is argued that for most use cases,
a blockchain is in practice not an appropriate technical solution. In Fig. 15
a flowchart can be seen where answers to up to six questions will result in a
recommendation: using a type of blockchain, or none at all. Let us continue
to apply the framework to the project and discuss its relevance and appli-
cability.

The first question to ask is if we need to store states for the solution to be
able to control the medium access, it is necessary with memory to keep track
of each user instance on the blockchain network. In practice, this memory
does not need to be on the blockchain network but it becomes simpler if all
information is on the same layer, to avoid having to relay data.

Next, we ask ourselves if there exists multiple writers. For a successful scraper
module, it must be expected that there are multiple users which operate in
parallel. In the same way, the solution has to be able to deal with multiple
readers and writers of data.

The third question is an interesting one; if we always have the opportunity to
use a trusted third party (TTP). For any successful web service, may it be an
e-commerce company or a video streaming service, it is more or less assumed
that there is availability and trust involved. The authors argue that when a
TTP always is online the write operations can be delegated to it and it can
verify state transitions. While this may be true in general, for this project
it could be argued that the core issue is not consensus and disputes between
entities but taking advantage of verifiable computations on the blockchain
as a memory-capable puzzle protocol filter. Granted that, it could still be
beneficial to have a blockchain despite an always online TTP.

The following question is if the writers are all known. The answer to that
is bluntly speaking no, since we can never know for sure who the mobile
application users are nor the scrapers. Were we to follow the flowchart then
we would end up with a permissionless blockchain such as the main network
of Ethereum. It could be argued that for this specific project that would
be very inappropriate and not make any sense whatsoever. Perhaps it is
because the project has neither pure economic incentive nor a need for trans-
parency for the participants. There would not be feasible at all to run this module on the main network and since this benefits strictly the company, a private blockchain is a sufficient enabler. Further, the company would have to distribute real ethers to consumers also and have to deal with actual cryptocurrencies.

The penultimate question is if writers are trusted, and they are not. Too much trust in users has possibly contributed to scraper behaviour. Finally, do we need public verifiability? Not really, as previously we are not interested in using the blockchain data collected by the users in any way apart from identifying users that sends excessive requests.

![Figure 15: A flowchart from [21] which argues for the necessity of a blockchain for organizations.](image)

Lastly, some of the properties of a permissioned blockchain can be compared to other alternatives as in Fig. [16]. In the project we value centrally managed authorities and assume there are a few untrusted writers such as
3.4 Smart contracts and autonomous processes

A smart contract is executable code that lives on the blockchain to facilitate, execute and enforce the terms of an agreement between untrusted parties [36]. The contract can release digital assets to certain participants when pre-defined rules are met. In its essence, a smart contract is programmed instructions on the blockchain with the opportunity to react differently to the input being received. Input can vary from simple read operations to value transfers to the contract, which may result in computational or write operations on the blockchain. In this project, this would be units of value in the form of tokens, received in exchange of successful proofs verified by the blockchain. States of the smart contract are saved and updated on the blockchain as in Fig. [17].
There is also an immutability feature of both blockchains and smart contracts which can both be a blessing and a curse. Not being able to alter a contract is a trust mechanism that can soothe parties but could also open up to vulnerabilities and exploits. Such exploits can only be fixed by redeployment of the contract and for many cases, such as bigger projects, this is sometimes not feasible when it includes many stakeholders and large funds.

For the project, we are not concerned about security but it should be noted that there is a significant amount of risks associated with smart contracts. Some of these risks can be seen in Fig. 18 and a proposed set of countermeasures, respectively.
The most well-known exploit is known as the *decentralised autonomous organization hack*, or the DAO hack, which resulted in a loss of 60 million US dollar in June 2016 [37] for the organization. The exploit took advantage of a vulnerability in a smart contract, which made it possible to retrieve funds that were supposed to be locked, through a recursive call bug. The result of this was that the Ethereum network performed a so called ”hard fork” and splitted the network into two chains as in Fig. 19.
Users who strongly believed in the philosophy of blockchain immutability stayed with the original network which acknowledged the loss while others, the core team at the time included, sought to recover the funds by all means. The result is that the network we know today as Ethereum is the new one, while the original chain is called Ethereum Classic with its own cryptocurrency.

The motivation of smart contracts for this project could be questioned. One could argue that the corresponding logic on a server or similar machine could work just as well and then a blockchain would not be needed at all. The objection is valid but not entirely justified since a transaction-oriented blockchain with smart contracts is more than simply the logic and rules. Firstly, the smart contracts opens up a layer that can operate outside of HTTP and consequently traditional requests. This study can push the boundaries further to make Web 3.0 become mainstream and an established network standard to solve this issue. Within the scope of scraping, given that a smart-contract solution works it could be distributed across a vast amount of systems and industries with a steady stream of data available to consumers. A logic that is simply stored on a server and distributed to thousands of others is not as distributed as a decentralised and autonomous service. Such service is audited, verified, and used around the world, around the clock. It
could be argued that there are few, if any, alternatives which can match the transparency and integrity of a decentralised application. Provided that it is necessary for an organization which is not always the case.

### 3.5 Hash algorithms and cryptography

A hash function and its stablemate cryptography are two of the fundamental aspects of blockchain technology when it comes to trust and integrity. The output to a hash function is a message digest, or simply digest; a digital fingerprint of characters and numbers. Cryptography provides a mechanism to encode the rules of a cryptocurrency system in the system itself [39]. It depends on deep academic research and advanced mathematical techniques which can be difficult to understand. To understand a hash function better, three basic properties are as presented in Definition 1.

<table>
<thead>
<tr>
<th>Definition 1 (Basic properties of a hash).</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Accepts a string input of any size</td>
</tr>
<tr>
<td>• Produces a fixed sized output; in this project assumed to be a 256-bit output size due to the use of either Keccak-256 or SHA-256 in Solidity</td>
</tr>
<tr>
<td>• Efficiently computable, i.e. given an input it is possible to determine its output within a reasonable amount of time; a hash of an n-bit string should have a running time of $O(n)$</td>
</tr>
</tbody>
</table>

The properties in Definition 1 could be used to build a data structure such as a hash table. Since we focus exclusively on cryptographic hash functions, we continue to define cryptographically secure hash functions [39]. These additional three properties are presented in Definition 2 below.
Definition 2 (Cryptographically secure properties).

- **Collision-resistance.** A hash function $H$ is collision resistant if it is infeasible to find two values, $x$ and $y$, such that $x \neq y$ and $H(x) = H(y)$.

- **Hiding.** A hash function $H$ is hiding if: when a secret value $r$ is chosen from a probability distribution function with high min-entropy, then given $H(r||x)$ it is infeasible to find $x$. Concatenation is denoted by the operator $||$.

- **Puzzle friendliness.** A hash function $H$ is called puzzle-friendly if for every possible $n$-bit output value $y$ it is infeasible to find $x$ such that $H(k+x) = y$ in time significantly less than $2^n$, where $k$ is chosen from a distribution with high min-entropy.

For collision-resistance, it is worth to note that infeasability is not the same as it being impossible to do. It is known that collisions do exists and it can be proven by a simple counting argument \[39\]. Consider hash functions of 256-bit output size. Let us simply pick $2^{256} + 1$ distinct values and check if any two outputs are equal. Since we picked more inputs than possible outputs, some pair of hash digests must collide. In fact, it is sufficient to pick $2^{130} + 1$ inputs to have a 99.8% probability of at least two collisions.

The hiding property asserts that given the outputs of the hash function $y = H(x)$, there is no feasible way to figure out the input $x$. Consider the following example: an experiment with a coin flip is performed. If the outcome of the flip is heads, we announce the hash of the string $heads$. Similarly, we do the same with tails and the hash of the string $tails$. Were we to ask an adversary, who did not see the flip but only saw the hash output, to figure out the outcome of flip it is easy to verify: just hash the two strings and compare. The adversary could find the input because of the limited set of possible inputs: \{heads, tails\}. It needs to be the case that no value of $x$ is likely to achieve the hiding property and $x$ has to be chosen from a very large, spread out set. If $x$ is sufficiently large, the method of trying a limited but likely amount of values will not work. But it is possible to achieve the hiding property even when the input is limited if it is concatenated with something else such as a secret value $r$. This is known as salting a string to hash in cryptography and takes advantage of the desirable cryptographic avalanche
property of hashes: a small change in the input string of a hash, such as flipping a single bit, will change the output of a hash function significantly.

Lastly, we have the puzzle friendliness property. It means that if someone wants to achieve a particular output $y$ of a hash function and that there is a part of the input chosen in a suitable random way, it is very difficult to find another value that coincides with $y$. Assume there is a puzzle which requires finding an input to a hash function such that the output belongs to a set $\mathcal{Y}$. If this puzzle is puzzle-friendly it implies that there is no better solving strategy than trying values of $x$ such that $H(x) \in \mathcal{Y}$. The size of $\mathcal{Y}$ determines the difficulty of the puzzle. If the cardinality is equal to the set of all $n$-bit strings, $|\mathcal{Y}| = 2^n$, it is a trivial puzzle where all inputs are considered a solution. Contrarily, if there is only one element, $|\mathcal{Y}| = 1$, it is a maximally hard. The adjustable difficulty of the proposed PoW function will be based on the set $\mathcal{Y}$ and hash computations which consist of the properties in Definition 1 and 2.
4 A New Blockchain-Based Approach to Web Scraping

Since we have decided to use the Ethereum blockchain and smart contracts, it is necessary to run such blockchain with appropriate permission control. The blockchain client is based on Geth according to Section 2.6 and since it is required to run a private and permissioned blockchain, this should be the first step in the project disposition. The whole mechanism of protecting against scrapers must be decided before smart contract and prototype development are initiated. It is necessary to develop a smart contract with the primary feature to validate a message digest and confirm that it contains a prefix with certain amount of leading zeros, in the same way the PoW protocol works. This has been done in the contract-oriented programming language Solidity paired with the development and testing environment Truffle.

Subsequently, a corresponding client application needs to be developed for Android which matches the validation function in the smart contract. Both the logic on the blockchain (smart contract) and the client (Android) need to continuously communicate and it is necessary to ensure that they can interact with each other successfully. This part (requirement 5) is critical due to its dependability on the light client protocol and blockchain software. If everything works as intended, we will measure the performance of the solution and whether it is viable to use from a latency perspective in a local area network. In Fig. 20, we show the implementation phases of the project, and the different technologies used can be seen in Fig. 21.
Figure 20: The approach and the sequential steps of the project.
Figure 21: Technology used in the project, mapped to each implementation step.

1. **MAIN BLOCKCHAIN NODE**
   - Terminal-based (MacBook)

2. **SOLUTION PROTOCOL**
   - Rules for the mobile client, blockchain, and server

3. **SMART CONTRACT**
   - Developed on a computer, exists on the blockchain

4. **ANDROID APPLICATION (LIGHT NODE)**
   - Communicates mainly with the blockchain

5. **SERVER**
   - Reads blockchain information, sends the client data
4.1 Blockchain disposition

To run a private blockchain network, we run a blockchain node according to [10]. Before being able to start a blockchain node, it is necessary to configure the genesis block. We use the standard settings apart from the chain identity, which by default connects to the main network. To connect to a private blockchain network, the genesis file and the network identity flag has to conform. The most important custom settings in this case is difficulty and gas limit, two parameters which defines how easy it is to find a solution, which translates into a new block, and the maximum gas cost to accept a transaction, respectively.

- `gasLimit: "800000000000"
- `difficulty: "200000"

The values corresponding to each parameter were not entirely arbitrary. Firstly, it is beneficial to have a very high limit of operational costs since ether is not an issue in a private environment with only one miner. Secondly, we aim at a faster mining rate to process transactions compared to the main Ethereum network with a relatively low value. Since we use a private blockchain on a local area network, our equipment setup is the following:

- MacBook Air (Early 2014), macOS High Sierra 10.13.4, Geth 1.8.10-stable
- Samsung Galaxy S6, Android 7.0

After initiation of the blockchain, through the genesis file which creates the first block, we run the node with the following flags:

```
geth --datadir LightClient --networkid 180128 --port 30303 --maxpeers 5 --rpc --rpcport 8545 --gasprice 0 --targetgaslimit 800000000000 --lightserv 90 --rpcaddr 0.0.0.0 --rpcapi personal,db,eth,net,web3 console
```

The flags will be briefly presented below.
**datadir.** The directory of where we store blockchain information and history, including the genesis file.

**networkid.** To distinguish between different networks.

**port.** The network listening port.

**maxpeers.** The maximum amount of users on the network, in the case of the project it is simply three: two full nodes (computer and server) and a light node (device).

**rpc.** Allowing remote procedure call, i.e. mobile devices are allowed to send commands to the blockchain for execution.

**rpcport.** The port for devices to connect to, makes it possible to exploit RPC calls.

**gasprice.** The minimum gas price to accept for processing a transaction (i.e. mining).

**targetgaslimit.** The gas limit to not exceed when making a transaction. Operational calls to the blockchain which exceeds this amount will revert and not be broadcast on the network.

**lightserv.** Percentage of the time allowed to serve light nodes, maximum 90%.

**rpcaddr.** HTTP-RPC listening interface and in this case we allow any IP address to connect.

**rpcapi.** APIs offered over the HTTP-RPC interface.

The next steps are to create an account, unlock the account, and make it a default account to be able to mine; process transactions and get rewarded in ether. We have used the quick way to create and unlock an account without any time restriction (0), and set the password to a simple string. This is done by the following commands:
personal.newAccount("daniel")
personal.unlockAccount(eth.accounts[0], "daniel", 0)
eth.defaultAccount = eth.coinbase

In practice, using this way to create and unlock accounts is not to be recom-
manded with respect to eavesdropping in the network. For this project such
vulnerabilities are not considered but it is important to emphasis.
After mining at least one block, with a reward of 5 ETH, it is now pos-
sible to deploy one or several smart contracts on the blockchain network.
Deploying a contract comes with a gas cost, which can be translated to a
corresponding value in ether. Furthermore, on the main network, ethers have
a somewhat volatile price but gas costs are always the same for execution of
certain instructions.

4.1.1 Deploy and use a contract

To enable deployment and calls of a simple contract through the terminal, it
is assumed that a coinbase account exists with a non-nil balance. To deploy
a simple contract, we use the one in Fig. 22 to demonstrate the approach of
putting it on the blockchain and calling its function.

```solidity
pragma solidity ^0.4.18;

contract Greeter
{
    function Hello(string _str) public pure returns (string)
    {
        return _str;
    }
}
```

Figure 22: A complete contract in Solidity (Greeter.sol) which contains a
simple function Hello.

First, we check the current balance of our account

> web3.fromWei(eth.getBalance(eth.accounts[0]), "ether");
5

46
To deploy the contract on the blockchain, known as migration, we use the development framework Truffle and connect it to our blockchain through the open `rpcport` flag used earlier. With a finished contract, the next step is to migrate it through

```
truffle(development)> migrate
```

Which shows a new transaction posted to the blockchain

```
INFO [07-11|15:19:40] Submitted contract creation
fullhash=0x792abb766955f5e2fb9eb1f12b3b83f[..]
```

To make it appear on the network, it needs to be processed through mining. After a few blocks, the contract will exist on the private blockchain permanently.

```
INFO [07-11|15:22:07] Submitted contract creation fullhash=0x50[..]
contract=0x5745c388D6f145e3f4517377301C1035Ea8C13BF
```

The contract address can be seen directly on the chain, after the full hash of the block, or through Truffle. Despite a contract existing on the network, it is not possible to directly communicate with it. The application binary interface (ABI) of the contract is needed to interact with it on the blockchain.

```
truffle(development)> JSON.stringify(Greeter.abi)
[..]
```

After saving the output, we have both necessary components to fully interact with a smart contract: (1) its contract address, and (2) its contract ABI. The last steps can be seen in Fig. 23 where we specify a contract address and ABI before we use its function. The very same steps shown are performed in the following sections when we interact with a contract through the terminal.
4.2 Scraper mitigation algorithm

The basis of the proposed solution is that access to a resource requires invested CPU power and computational effort. Furthermore, the effort should be dynamic and not static. More importantly, the resource should not simply be accessible through traditional HTTP requests which becomes an easy target for scrapers. Using the requirements from Section 2.2 we can conclude the following:

1. The client must perform certain calculations that are not easily spoofable or in some way circumvented without investing time and computational power. A current and unique solution, specifically for a user, changes after each reward.

2. The contract must have a way to validate current solutions and invalidate solutions already verified. In addition, a mechanism which prevents a user from sending multiple identical solutions. This is a transaction consensus property of blockchains, discussed in Section 3.2.

3. The server must be able to check whether a user has sufficient tokens to exchange for a resource and after providing the resource be able to subtract a fee from the user.

The proposed protocol can be seen in Fig. 24. At initiation of the mobile application, the client starts to generate a proof and then sends it to the blockchain for validation. After reception of a token, it will send a request to the server through HTTP. The server, which is also connected to the blockchain, can directly read the current state of the blockchain and subsequently send the client a resource. Finally, it must in some way simulate the consumption of a token and it is done with token burn, a technique which subtracts tokens from a user on the network. This means it is not really necessary to transfer tokens from a user to the contract since the tokens

Figure 23: Calling the function Hello from the contract Greeter.
are fungible and without any other practical use or value. The sequence is repeated continuously and in practice we could stop mine after collecting a certain amount of tokens. This amount should preferably let genuine users use the service freely. Scrapers on the other hand would struggle with the dynamic and restrictive resource-granting service if they wish to continue as in Table 2. Note that this scheme does not stop scrapers entirely but aims to limit high-volume scrapers since it is necessary to put in computational effort to generate solutions.

![Resource Access Sequence](image)

Figure 24: Concept of how the scraper protection mechanism should operate.

We continue with the scraper protection mechanism in detail, from the perspectives of a blockchain and user, respectively. The proposed mechanism is called *proof of effort* (PoE), due to its similarity with PoW and emphasis on computational power. In Fig. 24 it is the highest level perspective being displayed, and the different steps between the three systems. The isolated interaction between the client and the blockchain in more detail can be seen in Fig. 25 and here we only take into consideration what the blockchain
knows and expects of the user. Nothing about any off-chain (device) calculation.

Figure 25: PoE protocol principles and expectations of the blockchain. Note that the proof contains three components, where the $||$ operator denotes concatenation.

As can be seen, the blockchain only receives the nonce $N$ and nothing else from the client. This is because the public address is implicit in the message and the token information is already stored and managed by the blockchain. Furthermore, the incrementation will only occur after a token has been sent. The off-chain algorithm and steps taken by the client can be seen in Fig. 26.
Figure 26: PoE protocol principles and expectations of the client in each cycle. The computation is done in the hash step.

Since tokens must be acquired with continuity, both Fig. 25 and Fig. 26 show each cycle (iteration) of the protocol. For the client, the same components as in Fig. 25 have to be taken into consideration. Firstly, by cryptographic methods, an Ethereum address is uniquely generated when participating in a blockchain network. The address is based on the creation of a private key through the Elliptic Curve Digital Signature Algorithm. It is necessary for identification and the ability to transfer funds. The amount of tokens received, $T$, is incremented whenever a user gets rewarded for a valid proof. Since this increases after successful proofs are sent, we will have a unique solution at every new iteration. In fact, the hash output is significantly affected by a small change in $T$ due to the avalanche effect of cryptographic functions (see Section 3.5).
4.3 Smart contract development

With Solidity, we will now implement the proposed protocol in Section 4.3 for the blockchain in Fig. 25. The final function, named **checkPow**, can be seen in Fig. 27.

```solidity
function checkPow(string _nonce) public payable {
    bytes32 hashed = keccak256(msg.sender, totalTokensUserRequest[msg.sender], _nonce);
    if (checkLeadingZeros(hashed, difficulty)) {
        tokenBalance[msg.sender] += 1;
        totalTokensUserRequest[msg.sender] += 1;
    }
}
```

Figure 27: The main smart contract function which checks and validates proofs.

To be able to validate a difficulty which varies, it includes another function **checkLeadingZeros**, which takes two parameters: a hash and a number of zeros required. An example of how such a hash could look like is

```plaintext
hash9 = 000000000 f80ceb128711d5c1e0cd34bc6d588eb9165c1812d396909
```

This hash would satisfy the function call **checkLeadingZeros(hash9, x)** for \( x \leq 9, x \in \mathbb{N} \). For \( x > 9 \) the output to the function will be false. In addition, there are two mapping structures, **tokenBalance** and **totalTokensUserRequest**, to keep track of the tokens in possession and the total tokens received through user requests, respectively. The mappings work like a key-value pair hash table, where the key is an address and the value is an integer. The function that validates the leading zeros of a hash output is seen in Fig. 28.
They way it operates is to iterate over a hash, from left to right, in number of steps equal to the required amount of leading zeros. If any byte is not equal to a zero, the function does not satisfy the difficulty. Furthermore, since both `checkPow` and `setDifficulty` are functions which stores information on the blockchain, they require gas to function. Because of that, they have the payable identifier in the function declaration. The last function used is `setDifficulty`, which can be seen in Fig. 29.

This function changes the amount of zeros necessary to have a valid proof. Since the function should only be used by the contract owner, e.g. TML, this is being checked before the difficulty is adjusted. The variable `owner` is set when the contract is deployed. We let the variable `difficulty` to be global and publicly accessible by all participants through the function in Fig. 30.
Figure 30: The function used to read the current difficulty of the network.

The final two functions display the current token amount and the total amount of tokens received, as in Fig. 31 and Fig. 32 respectively.

Figure 31: The function used to read the amount of tokens of an address.

Figure 32: The function used to read the total received tokens of an address.

4.4 Client side development

4.4.1 Blockchain connection setup

The subsequent step is to implement the protocol of the client in Fig. 26. We aim to generate a solution which can be verified in the same way as on Ethereum. To make this possible, it is vital to take advantage of the light client library for mobile communication with the blockchain. This is the most critical step for a functional POC and part of the compatibility requirement of Ethereum discussed in Section 2.5.3.
If it is assumed we run a blockchain according to the disposition in Section 4.2, the first step is to connect to it with a mobile device, as in Fig. 33.

Figure 33: Establishing a connection to the blockchain and creation of a light node.

This node configuration should match the blockchain. The configuration includes information about the network such as its network ID number, genesis block, and the enode address ID. An example of an enode address could look like the following:

```
enode://2dec2aef5af3b01f1848[...] @ 192.168.0.3:8545
```

In addition, this information is encapsulated and saved in the application of the mobile device. The node is now fully functional and has the ability to listen to a specific port. Since peer discovery has not been functional, the mobile device has to be manually added to the network through its enode address. In contrast to the blockchain, the listener port of the mobile client is neither configurable nor static. This finding is summarized in Observation 1.

**Observation 1 (Mobile).** A new blockchain listener port is dynamically assigned to a free port every time a user opens the mobile application when it is closed. Furthermore, a mobile light node cannot define a listener port beforehand.
4.4.2 Wallet setup

To be able to fully interact with the blockchain network as a participant, an agent needs a public identifier. This is the Ethereum address, which is based on the public address. The public and private key pair is created in Fig. 34.

```
String keyStorePath = getFilesDir() + ".ethereum/keystore/";
final String passphrase = "mobilelife";
final KeyStore keyStore = Geth.newKeyStore(keyStorePath, Geth.Lightscreen, Geth.Lightscreen);
final Account accountFrom = keyStore.newAccount(passphrase);
Account accounts = keyStore.getAccounts();
final Account firstAccount = accounts.get(0);
final String publicKeyAddress = firstAccount.getAddress().getHex();
final Address addressFromHex = Geth.newAddressFromHex(publicKeyAddress);
```

Figure 34: Creation of a public and private key pair, i.e. a blockchain wallet to receive and send transactions.

There is a custom object called KeyStore which manages storage and encryption of the information needed. There is also an Account object that represents a stored key while the Address represents a 20 byte address of an Ethereum account. The only way to fully interact with the blockchain through the blockchain library is to use these objects. Note that it is not recommended to write a password explicitly as in the string variable passphrase but is once again something done for simplicity in this POC.
4.4.3 Read states with contract calls

It is easy to call a contract and read states with the terminal interface. For mobile devices, it has not been as easy to simply send the function call as done in Fig. 23. The code to call a contract which reads the difficulty of the blockchain is in Fig. 35.

Figure 35: Read the current difficulty of the blockchain through the mobile client.

Here there are two main issues to discuss. Firstly, we have encountered an error where it is not possible to simply call a contract without a parameter. To this end, it was necessary to adjust the function such that it receives an unused parameter, e.g. a string. The consequence is that the function `getDifficulty` in Fig. 30 has to be adjusted to include an arbitrary parameter, which does not affect the function itself. Secondly, to retrieve an integer stored on the blockchain incurs some kind of marshalling error. This means that we have only successfully managed to fetch strings and booleans. The new `getDifficultyStr` function which tackles these two shortcomings is found in Fig. 36, where we had to introduce an additional function which converts integers to strings.

Figure 36: The new version which makes it possible to read the difficulty of the blockchain as a string.

Since this, unfortunately, is not a native function in Solidity, the helper
function can be found in Fig. 47 in Appendix A. The same approach is used to retrieve the current token amount and total tokens received. In addition to the same shortcomings as previously, another error occurs when a string simply contains a zero ("0"). To mitigate this problem, we had to introduce a function to replace `requestAmount` in Fig. 32. This function, seen in Fig. 37, returns a minimum value of 1 to circumvent this issue.

```
function requestAmountStr(address _addr) public view returns (string)
{
    return uintToString(totalTokensUserRequest[_addr] + 1);
}
```

Figure 37: The new version which makes it possible to read the total received tokens.

Since this shows one more token than the actual amount, it needs to be taken into consideration when performing off-chain calculations which results in a corresponding subtraction as in Fig. 38.

```
parameterToken.setAddress(Get.newAddressFromHex(publicAddress));
parametersToken.set(0, parameterToken);
boundContract.call(callOptsToken, resultsToken, _"requestAmountStr", parametersToken);
String tempTokenRequested = resultToken.getString(); // (current amount of received tokens) + 1
int realTotalTokensRequested = parseInt(tempTokenRequested) - 1;
```

Figure 38: Read the total received tokens through the blockchain.

The two findings are summarized in Observation 2 and 3 below.

**Observation 2 (Mobile).** When calling a function on the blockchain, it is important that it accepts at least one parameter since the mobile client encounters a problem when sending calls without a parameter.

**Observation 3 (Mobile).** As of now it is certainly possible to read blockchain information of the data types string and boolean. Integers, on the other hand, are not fully developed to be fetched nor passed to a function. In addition, it is not entirely feasible to interpret the number zero as a string on a mobile device.
4.4.4 Write states with contract calls

The last step is to generate and send a solution in accordance to the PoE protocol of the client developed earlier, in Fig. 26. We will continue to present the last step of the mobile communication with the blockchain and in the next subsection finish with how a solution is generated off-chain, a (Java) function we have named generateSolution.

To send a transaction requires more information than simply sending a call since we are not just reading the blockchain. A transaction-based call could manipulate the blockchain and change its state. For this reason, it is necessary to supply such calls with sufficient gas. To fully send a transaction there are two parts. The first part is to set any parameters such as the gas cost, the origin of the transaction (public address), and the actual payload which is the dynamic solution. Every nonce is incremented from the last block (-1) nonce such that each new block has a higher nonce value than the previous one. This can all be seen in Fig. 39. Note that nonce in this setting is the native blockchain protocol nonce and not the proposed nonce N in the PoE protocol.

```java
// Generate a solution with wallet address, total tokens requested, and the difficulty */
String realSolution = GenerateSolution.generateSolution(publicAddress, realTotalTokens, difficultyInt);
// Create transaction object that will work as (from: account, gas: price) */
Transaction transaction = new Transaction();
transaction.setContext(Geth.newContext());
Interfaces parametersPow = Geth.newInterfaces(1);
Interface parameterPow = Geth.newInterface();
parameterPow.setString(realSolution);
parametersPow.set(0, parameterPow);
transaction.setGasLimit(DEFAULT_GAS_LIMIT);
transaction.setGasPrice(DEFAULT_GAS_PRICE);
transaction.setFrom(firstAccount.getAddress());
long nonceTransaction = ec.getTransactionAt(Geth.newContext(), firstAccount.getAddress(), 0 - 1);
transaction.setNonce(nonceTransaction);
```

Figure 39: Setup of a transaction-based call.

The second part is when a transaction hash is posted, it is subsequently signed by the sender as in Fig. 40. This is done through the KeyStore object and includes origin of the transaction, the passprase and other block and network information.
4.4.5 Generate dynamic solutions

Since there are issues passing integers to a function, it is necessary to send solutions found off-chain based on strings. For that reason, we need to iterate over nonces N as strings instead of integers and they need to satisfy the difficulty of the solution. It follows that these need translation to the appropriate format for the smart contract. Moreover, Solidity supports tightly packed encoding \[41\] and higher-order (left) side padding of integers \[42\]. To illustrate: an integer, padded to 32 bytes, would be interpreted through the Ethereum interface as

\[
70 \text{ (base 10)} \equiv 46 \text{ (base 16)} \equiv 0x000000...46
\]

The same goes for strings but in contrast to integers they are padded to the right of the 32 bytes while the content is still UTF-8 encoded (ASCII) in hexadecimal notation. Again, we can illustrate how the string \(KTH\) is represented when right padded to 32 bytes

\[
'KTH' \equiv 4b5448 \text{ (base 16)} \equiv 0x4b544800...0
\]

The algorithm to find a solution off-chain, generateSolution, is in Fig. \[11\]
Figure 41: The algorithm to generate dynamic solutions based on a public address, total received tokens, and a nonce. Note that `toLeadingZeros` is used for zero left padding of integers too.

In line with the PoE protocol the nonce N is incremented until the hash output of the total string (public address, amount of received tokens, and the nonce) satisfies the difficulty, i.e. a number of leading zeros of the hash. To simulate the same Solidity (on-chain) Keccak-256 hash algorithm off-chain we use the web3j library and its function `Hash.java` [43]. The steps taken by the algorithm are the following:

1. Create a string `leadingZeros` through the function `toLeadingZeros` (Fig. 42), which is identical to a $d$ zero-lead part of a hash, where $d > 0 \in \mathbb{N}$ is the difficulty.

2. Convert the current nonce N to the corresponding ASCII hexadecimal string with the function `encodeAscii` in Fig. 43.

3. Append the wallet address, the total requests left padded with zeros, and the nonce, to the variable `totalString`.

4. Hash `totalString` and compare the leftmost $d + 2$ characters to the expected `leadingZeros`.

5. If there is a match, decode the hexadecimal string nonce with the function `decodeAscii` in Fig. 44, since the blockchain interface interprets parameters as decimal; see Appendix B for explanation.
6. Else, increment the nonce, \( i = i + 1 \), and go back to step 2.

Figure 42: A helper function which generates a number of leading zeros to be appended in front of a string.

```java
public static String toLeadingZeros(int num) {
    if (num < 1) return "0";
    else {
        return IntStream.range(0, num).mapToObj(n -> "0").collect(Collectors.joining());
    }
}
```

Figure 43: Convert a text string to its corresponding ASCII code in hexadecimal notation.

```java
public static String encodeAscii(String vanillaString) {
    StringBuilder builder = new StringBuilder();
    for (char c : vanillaString.toCharArray()) {
        int decimal = (int) c;
        String hexString = Integer.toHexString(decimal);
        builder.append(hexString);
    }
    return builder.toString();
}
```

Figure 44: Parse a string of ASCII code in hexadecimal notation for conversion to a text string.

```java
public static String decodeAscii(String hexStringAscii) {
    Stream<Character> asciiArray = IntStream.range(0, hexStringAscii.length()/2).mapToObj(x -> {
        int i = x * 2;
        int j = i + 2;
        return hexStringAscii.substring(i, j);
    }).map(c -> Integer.parseInt(c, 16)).flatMap(c -> {
        char[] chars = Character.toChars(c);
        return IntStream.range(0, chars.length).mapToObj(i -> chars[i]);
    });
    return asciiArray.map(c -> Character.toString(c)).collect(Collectors.joining());
}
```

This concludes any communication between the mobile device and smart contract (blockchain). In practice, the prototype of the communication between
blockchain and a smart device worked as expected. There is still more to develop for the blockchain and server communication. Despite that, the last part of how the server communicates with the contract is presented, in line with the PoE protocol.

4.5 Server side development

We use the Geth library to make a simple mock server which receives HTTP headers to verify and then query the blockchain for information about the sender, specifically the token balance. Due to a somewhat flawed mobile client, the server does not contain the full concept of token burn but this is a matter of implementation. It is possible to make a simple function in the contract which subtracts the current `tokenBalance` of a user with an amount. Since the server is not central to the solution and will not be included in any tests, the full server functionality can be found in Appendix C. We summarize the logic of the server as following:

1. Start a local server and connect it to the blockchain, as in Fig. 18.
2. Deploy the smart contract which includes the functions in Section 4.4.
3. Open a listener port and wait for custom headers `X-Wallet-Address`
4. Check the value to see if the header is formatted as a valid wallet address
5. If that is the case, check the current token balance of that address and write a simple string
5 Test Results and Analysis

It is now time to test the PoE protocol in a feasible and local environment. Even though tests will not include any delay to the server, it is still possible to evaluate the communication between the mobile device and the blockchain. Moreover, the majority delay will most likely take place during the computations of the PoE protocol as it should, when fine-tuned, provide the user with a continuous but moderate access to token rewards. It needs to be emphasized that tokens are a mere commodity that can be exchanged for something of value, and the philosophy lies in the adjustable rate of being able to receive these tokens.

5.1 Token generation simulation

Before the main test is presented it is necessary to comment on the appropriate difficulty. As can be expected, the fewer lead-zeros in a hash means a shorter computational time until a successful hash output is found. If a hash output is uniformly distributed among the characters, it should be assumed that the time between a solution is found is stochastic and equally probably albeit the vast majority of outcomes should occur within a certain time frame.

The test setup is simply that at the end of each loop, information about the round is logged in the Android Studio debug console as in Fig. 45.

Figure 45: At every round: save the timestamp, amount of tokens, current difficulty, and balance.

An extract of a log can be seen in Fig. 46, where the first two entries are month/day and timestamp, respectively.
It should be noted that sometimes it does not receive a token, such as at the time incidents 11:57:37 and 12:02:09. Based on the three tests, this is not an outlier and is a repeated phenomenon which occurs throughout the whole session. Indeed, it is not certain to as why this occurs and if the issue is on the client or the blockchain side. Due to the short interval in the first case, less than 4 seconds, it could be suspected that it found a solution in a short time but the solution was outdated due to new states on the blockchain, i.e. a higher total received tokens. On the other hand, in the second case it is noted that between 12:00:46 and 12:01:28 the client has received two tokens within one round.

Since the client software has certain flaws, it does not work to change the difficulty while a client is running. To change the difficulty and make the program work by itself, any adjustment must be done before the client is connected to the blockchain. Thus, it is not possible to change the difficulty while the program is running.

Something to investigate was whether or not the blockchain size proved to be an issue in the long run, i.e. a scalability issue. To this end, three testing sessions in the TML office in Stockholm took place. Sessions lasted for 4 hours, 4.5 hours, and 5.5 hours respectively. Accordingly, when the program is running, it is desired to avoid confounding factors during the test. In other words, for every test session we prepare the test environment equally, follow the same approach, and conduct them during comparable time periods of the day. More specifically, the steps are as following.
1. Clear the whole blockchain of any block data and cache and initialize a new blockchain from a genesis block

2. Run the mobile client which contains the developed client side software and save its public and enode address

3. Create a coinbase account, unlock it, and then add the mobile device to the blockchain network

4. Mine exactly one block for a reward of 5 ether to be able to deploy a contract

5. Deploy a contract through Truffle and mine exactly four blocks such that the contract is processed and exists on the blockchain

6. Fetch the contract address and ABI of the exchange value contract, EvNew, through Truffle and apply it to the blockchain

7. Change the contract difficulty to 4 with setDifficulty which results in a token reward frequency of approximately 20 to 120 seconds

8. Mine five additional blocks such that the blockchain with high certainty has changed state of the difficulty

9. Transfer 10 ether to the light peer and log each round according to Fig.

To analyze these logs, we used the programming language F# to parse and graphically represent the amount of requests in relation to time between a token is received. This was achieved by taking the difference between each time stamp and pair it with each round number (see Appendix F). To increase the confidence in that there would be no scalability issues in a local network, it would be expected that the request time is stochastic and without an increasing trend. In Appendix E the token generation graphs are shown and from the sample size, the blockchain size does not appear to affect the time to receive a token. On the contrary, it looks almost periodic, although the main result observed is that the output (the time between tokens received) is a stable and bounded signal which does not grow indefinitely.
5.2 Outcome

We refer back to Table 4 in Section 2.5 and the comparison between potential solutions. The observed outcome can be seen in Table 5.

Table 5: The assumed potential compared to the outcome of Ethereum.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Pre-project</th>
<th>Post-project</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Complexity</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2) Obscurity</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3) Robustness</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>4) Dynamicity</td>
<td>✓</td>
<td>Likely</td>
</tr>
<tr>
<td>5) Compatibility</td>
<td>Questionable</td>
<td>Unlikely</td>
</tr>
</tbody>
</table>

**Requirement 1:** *the solution should not introduce additional complexity for the user and the security scheme should occur in the back-end.* Our blockchain-based approach does not rely on challenges or tests that are supposed to interfere with the user experience. The complexity has been abstracted away.

**Requirement 2:** *the solution should not grant the client access to more information than necessary.* The purpose of using blockchain has satisfied this criterion. A client only knows a cryptographic scheme and how to solve a certain puzzle. The details of how to circumvent it is not apparent and interaction with the server does not rely on direct HTTP interactions.

**Requirement 3:** *the solution should be controlled centrally, on the server side, and not reveal any logic to the client.* This is the strongest point of the solution due to the non-existent benefits of an eavesdropper and the dynamic structure of how tokens are granted. The property of opaqueness is excelling on private blockchains and have potential to increase the integrity and security of distributed systems and services.

**Requirement 4:** *the solution should be scalable, permutable, and with flexibility to react to the network behaviour.* This was not possible to test due to the static behaviour of the difficulty. On the other hand, there is no reason for as to why it should not work since change of states can occur frequently.
and fast on a blockchain as long as there are miners. This project has not
the ability to evaluate this criteria in depth however.

**Requirement 5:** the solution has to be compatible with mobile devices and
not affect the service level. With respect to mobile support, it has been ob-
served that it may not be the most compatible framework for mobile devices
as of today. One could argue that it is simply the software development
that was flawed but due to lack of cohesive documentation and examples of
contract management on mobile devices, it is at least not a straight-forward
procedure. Our experience consists of struggle and mostly trial-and-error to
make a mobile client work as intended. It must be emphasized that this is
from the perspective of development during January to June 2018 and that,
hopefully, this can result in a completely different experience in a year or two.

Since the project tackled the specific issue of web scraping, the focus was
to investigate the Ethereum blockchain and its feasibility as an additional
security layer with the help of smart contracts. The outcome has not en-
tirely confirmed whether or not it would work in a commercial environment
due to project progress limitations and maturity of software.

### 5.3 Approach to development

The first thing done was to deploy a simple contract on the blockchain and
try to call it from another computer, all through the console (Ethereum in-
face). Subsequently, time was spent on the smart contract and finally the
light client. A better and more time-efficient approach would most definitely
have been to start with calling a simple contract on the mobile device and
learning about its limitations and flaws before the project proceeds to smart
contract development. In retrospect, this would have been faster since a dis-
proportional amount of time were spent on smart contracts and superfluous
helper functions in combination with unresolved issues. Many of which could
have not only been omitted but in the end would need to be rewritten. The
figures we have shown after function improvements in Section 4.5 are simple
getters and not as major as our original `checkPow` which used other ways
to append strings and data types, as well as ways to check for leading zeros.

In like fashion, time was spent on issues that are out of our control and
not vital to the project such as automatic peer discovery and the way a new port is opened at every new session. Similarly, a lot of time was spent on attempts to resolve other light client bugs with ad hoc solutions which after a few weeks would get patched in the subsequent update. At the same time, we tried to fix bugs that still are unresolved as of today and without any shown interest by the community to tackle. It goes without saying that one cannot demand much from the open source community but as for the goal and ambition of the project it was a major headache. Solidity, for instance, does not natively support string concatenation, conversion of data types such as string to integer, or string comparison, to mention a few.
5.4 Server and network

The project was only conducted on a local area network and in a specific office environment. Since we never tried it on a public and decentralised test network, we implicitly assumed the network would and will continue to be ideal. Specifically, there are several network fallacies [44] which have not been taken into consideration at all. These are as following

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. The topology does not change
6. There is one administrator
7. Transport costs are zero
8. The network is homogenous

In retrospect, our project touches the majority of these points and likely all but 3). First and foremost, we assume a reliable network. Although this is not a significant factor since a traditional client-server approach has this assumption also and is fundamentally implicit for any web service. Our latency is not only very low but the lowest possible due to operating a network service on localhost/127.0.0.1 which is the IP address of the local machine. Hence, our tests are outcomes of a very ideal and unrealistically low latency scenario for network services. While this may be true, the bandwidth is most likely less of an important factor since packets include comparatively small message payloads with only characters and no multimedia.

The network is as secure as the local machine and area network; security for the solution is not more important than network precautions and preventive measures in general for web services. The topology and administrator factor are not to be dismissed easily since they are a very fundamental assumption in our project and if this proposed solution would be further developed. Because in the end, it would require an entity to control the blockchain difficulty. Not necessarily a person but could very well be an algorithm which
uses statistical methods and data based on the users and requests, to change token prices and even control individual difficulties. Similarly to the way control systems work with feedback loops in industrial environments. Going back to topology, this would in practice mean that we still have one, or a limited, amount of administrators or central nodes. It could be argued that more middle layer oriented protocols have the opportunity to make certain mass-consumer data communication more opaque with the help of blockchain technology, in a positive way for service providers and genuine users. This could in practice make unwanted scraper behaviour very difficult and strengthen commercial web applications with exposure to clients over the internet, with respect to medium access and infrastructural privacy for any type of agent.

Lastly, a fallacy is to assume zero transport costs and a homogenous network. Since the project has taken place on a private network and with ’free money’, the cryptocurrency component is neither a cost nor an issue. Additional costs can be comparable to when no scraper protection module is active such as when third parties charge for every request to a server. That the network is assumed to be homogenous is something to be investigated further since every agent will have to go through the blockchain to receive a token and finally access to the service.

5.5 PoE protocol and tests

The protocol was designed to make it as difficult as possible for scrapers but it is not without flaws. The most severe one is that it assumes all mobile devices on the network to be equal. In practice, different models have different CPU power and consequently hash power. This is an additional factor that needs to be considered. Likewise, it could also be questioned if HTTP requests between the device and server will work as intended, especially without obstructive delay and congestion. Finally, we have not investigated the power consumption of this type of hashing. The proposed computations really need to be kept on such a level that it is not intense for the device whatsoever. Following that, it will become apparent that using such applications will be more energy consuming and the question is only to what degree.
Regarding the main test, there are several parameters which could have had affected its outcome. Since each component has not been tested in isolation, it is difficult to find a significant cause and effect of the issue with non-credited tokens and double-tokens in a round for instance. Likewise, the experiment may suffer due to possible lack of internal validity and several uncertainties such as

- wireless connection delay
- internet speed
- block congestion on the blockchain
- propagation speed and delay to the blockchain
- device and hardware delays and race conditions
- software specific delays and race conditions
- other stochastic phenomena
6 Conclusion

Scraping is a persistant problem today for services which provide accessible information. There exists several commercial solutions with varying degrees of success. After scraper behaviour analysis, the perpetrators are either blocked or sent a challenge-response test. For persistant high-volume scrapers originating from large bot networks, current solutions are not always sufficient. Based on this study, blockchain technology indicates potential to restrict high-volume scraping and access to a server. Through a proposed PoW algorithm, each smart device can solve unique dynamic solutions and earn its right to access the service. Due to the property of how blocks are appended to the blockchain, it is guaranteed that each solution will only be verified once. Hence, the proposed algorithm on the blockchain will never reward a solution more than once. Tests have indicated that the rate of receiving tokens by the blockchain is stochastic. By changing the contract difficulty, the time between rewards can be adjusted. It has been observed that the proposed protocol, based on mobile devices and a blockchain, has potential to work as an additional security layer. Such information could in practice be freshly generated flight prices, commodity prices, or data from Twitter. Finally, we experienced that the light client for Android may not work as expected with current Ethereum software and network protocol.

Based on studies and discussions presented before, we can summarize the conclusions as follows:

- The blockchain has a network behaviour that stores and computes data in cycles through mining. This can synchronize requests and compare them with requirements programmed on a smart contract. High-volume scrapers can be mitigated through smart contracts on the blockchain but there are trade-offs. There are a few things we did not consider or test: any server communication with the blockchain, energy consumption of the algorithm, and the effectiveness of the algorithm in mediating information between a client and a server. These are all important for an industrial capable solution.

- The benefit of a mobile cryptographic algorithm such as the one proposed is that it forces a client to perform certain calculations. Even if
a malicious user decompiles the program and reverse engineer the algorithm, it can not circumvent the fact that the cryptographic component forces the user to solve puzzles. Because of the nature of hashing and its avalanche effect on the output, the puzzles are solvable only in one way: bruteforce.

- In contrast to the traditional client-server architecture, we have explored a model consisting of client-blockchain and server-blockchain communication. By decreasing communication with a server and mainly communicate with an additional security layer, the blockchain, it is possible to better protect and conceal data. Nonetheless, it is not certain that a public or a private blockchain is the right choice.

6.1 Future work

The principle of mobile mining is an interesting topic to explore further and its requirements. Even if PoW is not the most suitable protocol to this end there could be other consensus algorithms that does not require the same amount of computations and memory. An example is the smartphone-based cryptocurrency Electroneum and its simulated mobile mining. In addition, a current issue is that smart devices cannot read and process a whole blockchain. To further investigate isolated parts of a network is also an interesting point to consider when treating the subject of mobile integration.

Finally, another type of project could be to question whether or not the current blockchains are usable and technologically sustainable for mobile clients. Perhaps a blockchain specifically constructed and customized for mobile devices is more suitable.
Appendix

Appendix A: Solidity

```solidity
function uintToString(uint v) private pure returns (string) {
    uint maxlen = 100;
    bytes memory reversed = new bytes(maxlen);
    uint i = 0;
    while (v > 0) {
        uint remainder = v % 10;
        v = v / 10;
        reversed[i++] = byte(48 + remainder);
    }
    bytes memory s = new bytes(i);
    for (uint j = 0; j < i; j++) {
        s[j] = reversed[i - j - 1];
    }
    string memory str = string(s);
    return str;
}
```

Figure 47: Convert an unsigned integer to a string.
Appendix B: Decode ASCII characters

As can be observed in Fig. 44, the function `decodeAscii` is slightly more complex compared to its counterpart `encodeAscii` and especially the lambda variables `i` and `j`. To explain, this is because of the way ASCII codes are interpreted in comparison to translation of a character to its code value. Thus, if we use the string `dave` and convert it to ASCII code the following is obtained by simply taking each character and appending its code value.

\[
\text{'dave'} \equiv 64617665 \text{ (base 16)}
\]

It is necessary to parse each character from the ASCII code if we would like to go from a sequence of code values to a string. An example is the character `d` represented by the hexadecimal value 64. Since the `substring` operation in Java receives the start and cutoff index as parameters, 0 and 2 is desired. It follows that the next step is to choose indices 2 and 4 for the value 61 representing `a`, and so on until this moving window completes Table 6. The first two steps of the decoding sequence are shown below.

<table>
<thead>
<tr>
<th>Start (i)</th>
<th>End (j)</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>d</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>v</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>e</td>
</tr>
</tbody>
</table>

Table 6: Index intervals and their respective character.
Appendix C: Server

```go
func startGeth(session *EvNewSession) *EvNewSession {
    var key = "c87509a1c065/beda78b3eb793e6fa7653b63b2ac82b41e5e4a9e8a8f44dc8d3"
    privKey, err := crypto.HexToECDSA(key)
    if err != nil {
        log.Fatal("Failed with conversion of private key: %v", err)
    }
    if err != nil {
        log.Fatal("Failed to connect to local blockchain network: %v", err)
    }
    auth := bind.NewKeyedTransactor(privKey)
    if err != nil {
        log.Fatal("Failed to create new keyed Transactor: %v", err)
    }
    addr, _, contract, err := DeployEvNew(auth, conn)
    if err != nil {
        log.Fatal("Failed to deploy contract: %v", err)
    }
    fmt.Println("Contract deployed to address ", addr.String())
    session = &EvNewSession{
        Contract: contract,
        CallOpts: bind.CallOpts{
            Pending: true,
        },
        TransactionOpts: bind.TransactionOpts{
            From: auth.From,
            GasLimit: 50000000,
        },
    }
    log.Println("Current Go account ", auth.From.String())
    diff, err := session.GetDifficulty(pane)
    if err != nil {
        log.Fatal("Failed to fetch the difficulty: %v", err)
    }
    fmt.Println("Current difficulty is: ", diff)
    return session
}
```

Figure 48: Setting up a Geth server locally through RPC and deploying the smart contract.
func main() {
    var firstSession *EwNewSession
    s = startgeth(firstSession)
    http.Handle("/", addressHandler(s))
    log.Println("Server started on localhost:8800")
    http.ListenAndServe(":8800", nil)
}

func addressHandler(sess *EwNewSession) func(w http.ResponseWriter, r *http.Request) {
    return func(w http.ResponseWriter, r *http.Request) {
        address := r.Header.Get("X-Wallet-Address")
        switch {
        case address == "":
            w.WriteHeader(http.StatusBadRequest)
            w.Write([]byte("Missing a wallet address"))
        case len(address) != 42:
            w.WriteHeader(http.StatusBadRequest)
            w.Write([]byte("Invalid length of wallet address"))
        case checkPrefix(address) == false:
            w.WriteHeader(http.StatusBadRequest)
            w.Write([]byte("Invalid hexadecimal prefix: 0x missing"))
        }
        w.WriteHeader(http.StatusOK)
        w.Write([]byte("Address accepted: \n"))
        addr := common.HexToAddress(address)
        userBigInt := big.NewInt(0)
        userBigInt, _ = sess.TokenBalance(addr)
        if userBigInt.Int64() > 8 {
            w.Write([]byte("** Secret Resource **"))
        } else {
            fmt.Println("Not enough tokens!")
        }
    }
}

func checkPrefix(str string) bool {
    runes := []rune(str)
    prefix := string(runes[0:2])
    if prefix == "0x" {
        return true
    }
    return false
}

Figure 49: Starting a Geth server locally to listen for HTTP headers to parse; if correct format and enough tokens, reward with a resource.
Appendix D: Android

```java
final static String GenesisBlock = "\
"{"chainId": 180128, 
"homesteadBlock": 0, 
"eip155Block": 0, 
"eip158Block": 0
"}
"nonce": "\0x0000000000000000180515", 
"timestamp": 0x0, 
"parentHash": "\0x0000000000000000000000000000000000000000000000000000000000000000", 
"extraData": "", 
"gasLimit": "80000000000", 
"difficulty": "\2000000", 
"mixhash": "\0x0000000000000000000000000000000000000000000000000000000000000000", 
"coinbase": "\0x3333333333333333333333333333333333333333333333333333333333333333", 
"alloc": { 
", 
""": 
"}
"};

final static String NodeAddress = 
"enode://2dec2a0ff5a3b3ce67b84eb5bb1b30f01f18e8e064c3a6de4832225d6e3968644d4d26ed1519192.168.0.13:8545";

final static int BlockchainNetworkId = 180128;

final static String ContractAddress = "0x5745c388d6f145e3f4517377301c1035ea8c13bf";

final static String ContractABI = 
"{["constant":true,"inputs":[],"name":"difficulty"},

final static long DefaultGasLimit = 7000000L;

public static BigInt GetDefaultGasPrice()
{
    BigInt DEFAULT_GAS_PRICE = new BigInt(18);
    DEFAULT_GAS_PRICE.setString("589000", 10);
    return DEFAULT_GAS_PRICE;
}
```

Figure 50: Necessary information about the blockchain for the mobile client.
Appendix E: Token generation graphs

Figure 51: 4 hours of token generation: a graph of the time between tokens received, in seconds, and amount of requests.
Figure 52: 4.5 hours of token generation: a graph of the time between tokens received, in seconds, and amount of requests.
Figure 53: 5.5 hours of token generation: a graph of the time between tokens received, in seconds, and amount of requests.
Appendix F: Token generation

Figure 54: Parse raw data in F# of the token generation logs from Android Studio and structure it. (Credits to Gabriel for his contribution.)

Figure 55: Plot the structured data in F#. (Credits to Gabriel for his contribution.)
References


