FACULTY OF ENGINEERING AND ARCHITECTURE

Computational integrity for outsourced execution of SPARQL queries

Serge Morel
Student number: 01407289

Supervisors: Prof. dr. ir. Ruben Verborgh, Dr. ir. Miel Vander Sande
Counsellors: Ir. Ruben Taelman, Joachim Van Herwegen

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2018-2019

GHENT UNIVERSITY

FACULTY OF ENGINEERING
AND ARCHITECTURE

Computational integrity for outsourced execution of SPARQL queries

Serge Morel
Student number: 01407289

Supervisors: Prof. dr. ir. Ruben Verborgh, Dr. ir. Miel Vander Sande
Counsellors: Ir. Ruben Taelman, Joachim Van Herwegen

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2018-2019

GHENT
UNIVERSITY

© Ghent University

The author(s) gives (give) permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In the case of any other use, the copyright terms have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation.

August 16, 2019

Acknowledgements

The topic of this thesis concerns a rather novel and academic concept. Its research area has incredibly talented people working on a very promising technology. Understanding the core principles behind proof systems proved to be quite difficult, but I am strongly convinced that it is a thing of the future. Just like the often highly-praised artificial intelligence technology, I feel that verifiable computation will become very useful in the future.
I would like to thank Joachim Van Herwegen and Ruben Taelman for steering me in the right direction and reviewing my work quickly and intensively. I would also like to thank Ruben Verborgh for allowing me to work on this novel concept.
It would be awful of me not to mention my family and friends for their support. Researching, implementing and writing about something that is entirely new to you requires perseverance and I would like to thank them profusely for giving me this.

Abstract

The Big Four tech companies are notorious for using personal data as a major source of revenue. Recent initiatives such as Solid aim to re-decentralize the web by using Linked Data to bring the data back to the user. Using this technology, any kind of data can be connected together, but this universality raises query scalability issues. This thesis focuses on verifiable outsourcing of BGP queries whilst offering proofs of completeness and soundness of the results. In the solution, outsourcing is done by computationally strong and willing third parties and effectively takes the load away from both the client and the data source. A solution using the concept of TPF and non-interactive proof systems is introduced. Verifying the results obtained from outsourcing maintains the same credibility as local execution, but outperforms it in terms of time, bandwidth and memory usage. Additionally, some use cases are introduced, along with a novel blockchain model using this solution as Proof of Work (PoW).

Index terms: SPARQL, Triple Pattern Fragments, Verifiable Outsourcing, Decentralization, Blockchain

Computational Integrity for Outsourced Execution of SPARQL Queries

Serge Morel
Ghent University

Supervisors: Prof dr. ir. Ruben Verborgh, Dr. ir. Miel Vander Sande
Counsellors: Ir. Ruben Taelman, Joachim Van Herwegen

Abstract

The Big Four tech companies are notorious for using personal data as a major source of revenue. Recent initiatives such as Solid aim to re-decentralize the web by using Linked Data to bring the data back to the user. Using this technology, any kind of data can be connected together, but this universality raises query scalability issues. This paper focuses on verifiable outsourcing of BGP queries whilst offering proofs of completeness and soundness of the results. In the solution, outsourcing is done by computationally strong and willing third parties and effectively takes the load away from both the client and the data source. A solution using the concept of TPF and non-interactive proof systems is introduced. Verifying the results obtained from outsourcing maintains the same credibility as local execution, but outperforms it in terms of time, bandwidth and memory usage. Additionally, some use cases are introduced, along with a novel blockchain model using this solution as Proof of Work (PoW).

1. Introduction

”Data is the new oil”, a quote that has become very familiar to a lot of people and is widely credited to Clive Humby, all the way back in 2006. However, data on its own often does not have any significant meaning. In order to produce meaningful results, careful aggregations need to be made and projected onto meaningful visualizations. For social media giants such as Facebook or Twitter, this has become their major source of revenue. This data mining has raised many concerns among the public, who want to know where, when and how their data is used.

Tim Berners-Lee launched Solid[1] with the purpose of bringing the data back in the hands of the user. It is based on the Semantic Web, an extension of the World Wide Web and uses Linked Data as a backbone. However, Linked Data comes with interoperability issues, causing undesirable availability rates at the servers performing data aggregations [1]. This general unavailability has led to the introduction of TPFs, a concept that moves query solving partially to the client-side [2]. While the technology is promising, it is not suited for hardware-constrained devices, omnipresent in the IoT world. These devices do not have the resources and connectivity to perform these aggregations.

Outsourcing to the data server is not an option and thus this paper focuses on a solution by outsourcing to computationally strong and willing third parties using verifiable outsourcing. The third party executes a Basic Graph Pattern (BGP) query in a verifiable environment, offering a proof of completeness and soundness for the obtained results. The requesting party verifies this using less resources compared to executing locally. It effectively takes the load away from both the data server and the client.

In what comes next, a short introduction to the concepts used in this paper is given in Section 2. The problem is more formally defined in Section 3, while the implementation of a solution is detailed in Section 4. As a means of an empirical evaluation, two use cases are discussed in Section 5. A more informed evaluation and the conclusions are given in Sections 6 and 7 respectively.

2. Related work

2.1. Linked Data

Linked Data is the backbone of the Semantic Web and re-introduces the original vision Tim Berners-Lee had of the web: structured and machine-readable information, all interlinked. The Resource Description Framework (RDF) is a knowledge-representation sys-

[1]`https://solid.inrupt.com/`

tem of Linked Data and uses triples to define relationships. A triple consists of a subject, predicate and object and are all represented by URIs. The query language most used with RDF data is SPARQL and is similar to SQL. It uses Triple Patterns (TPs) to define connected graphs, which are essentially triples containing variables. A collection of triple patterns is called a BGP.

However, SPARQL queries can be quite resource-intensive to execute. This has lead to availability issues and out of a survey of 427 SPARQL endpoints, half did not reach an availability rate of 95% [1]. The concept of Triple Pattern Fragments (TPFs) introduced by Verborgh et al. [2] aims to mitigate this issue by moving part of the complexity to the client-side. TPF servers are the new data servers and offer results to only one single triple pattern, which has low complexity. The client then uses a dynamic iterator pipeline algorithm to aggregate these TPFs locally. Apart from lowering the cost for the server, the concept of TPF also improves caching at the server-side, as result sets can more easily be reused.

2.2. Blockchain

Blockchain has received a lot of attention in the last couple of years. Essentially, its purpose is to offer an immutable, decentralized and verifiable data store. It finds its use in many applications, such as financial ones where privacy and integrity of data is key, but also non-financial ones like a decentralized storage service [3] and media authenticity purposes [4].

Its structure consists of a chain of blocks, ordered by creation time and containing application-specific data. The participants of the network are responsible for creating and validating incoming blocks, often rewarded by means of a digital currency. Changing a block that is already added to the chain, causes subsequent blocks to become invalid. Adding to this that producing a valid block requires a certain amount of time (mining), the immutability property is easily derived. Bitcoin, the most popular example, uses blockchain as a public ledger for digital currency transactions [5].

2.3. Zero-knowledge proofs

A zero-knowledge proof involves two parties: a prover $\mathscr{P}$ and a verifier $\mathscr{V}$. The prover's goal is to convince $\mathscr{V}$ that he knows a value x, without revealing x. They exhibit three properties:

1. **Completeness** If both parties follow the protocol and the statement is true, an honest prover should be able to convince the verifier of this fact.
2. **Soundness** If the statement is false, the probability that a dishonest prover can convince the verifier that it is true, is negligible.
3. **Zero-knowledge** The verifier cannot know anything except for the statement, no additional information is exchanged.

Zero-knowledge proofs can be either interactive or non-interactive. In the first case, $\mathscr{P}$ and $\mathscr{V}$ have a conversation that eventually convinces $\mathscr{V}$ about the truth value of a statement. Non-interactive ones do not require this communication phase, instead a one-time proof is made available by $\mathscr{P}$ and can be verified by anyone.

zk-SNARKs [6] are a popular example of zero-knowledge proofs and are used in the digital currency Zcash[2] to introduce shielded transactions. The recently introduced zk-STARKs [7] are a promising potential successor of zk-SNARKs due to the absence of a trusted setup phase and better scalability.

2.4. Computational integrity

Computational integrity comprises the common problem where an unobserved and distrusted party $\mathscr{P}$ executes a computation $\mathscr{C}$ on input x, resulting in output $y = \mathscr{C}(x)$. Said party might have the incentive to provide the more favorable output $y' \neq y$ and so computational integrity systems ensure that $\mathscr{P}$'s work can be efficiently verified.

A naive approach of ensuring the correctness of $\mathscr{C}$'s execution would be to re-execute the computation in a trusted environment. However, this does not scale and ideally $O(log(T_{\mathscr{C}}))$ verification time is obtained, where $T_{\mathscr{C}}$ indicates the time to compute y.

Proof systems for computational integrity usually compose of three algorithms:

- $Gen(1^{\lambda}, \mathscr{C}) \rightarrow (pk, vk)$
- $Prove(pk, \mathbb{X}, \mathbf{w}) \rightarrow \pi \colon (\mathbb{X}, \mathbf{w}) \in \mathscr{R}$
- $Verify(vk, \mathbb{X}, \pi) \rightarrow accept/deny$

Here, $\mathbb{X}$ comprises both x and y, while $\mathbf{w}$ is a transcript of the nondeterministic steps made during execution. π is a proof that confirms $\mathscr{P}$'s knowledge of $\mathbf{w}$. The two keys pk and vk are used for proving and verifying respectively and λ is the security parameter.

Computational integrity proofs usually boil down to generating an arithmetic circuit that represents the computation. It is then $\mathscr{P}$'s job to find a valid assignment for this circuit and proof to $\mathscr{V}$ that he knows this assignment using zero-knowledge proofs.

[2]https://z.cash/

3. Problem statement

In the previous section, the low availability of SPARQL endpoints was mentioned and how TPFs were introduced to mitigate this issue. However, clients do not always have the hardware, network connectivity or time to run the TPF algorithm. Outsourcing back to the data servers would reintroduce availability issues and is thus not an option. Outsourcing to computationally strong and willing third parties would be ideal and would take the load away from both the data source and the, possible resource constrained, client.

With outsourcing, however, also comes distrust. People do not trust social media giants with their data and privacy is getting progressively more media attention. So how can trust be introduced into what are essentially two strangers communicating about possibly sensitive data? This is a very broad question and this paper focuses on a specific context for this question. Two research questions are defined and define the scope for this paper:

> $\mathbb{Q}_1$: What are the advantages of verifying outsourced BGP query executions instead of executing them locally using TPF.
>
> $\mathbb{Q}_2$: How can non-interactive proof systems be used in verifying the computational integrity of the dynamic iterator pipeline algorithm?

In light of the first research question, the following hypotheses were defined:

> $\mathbb{H}_{1.1}$: Fetching and verifying the result from outsourced BGP queries is at least 25% faster than local execution using TPF.
>
> $\mathbb{H}_{1.2}$: Fetching and verifying the result from outsourced BGP queries uses at least 25% less memory than local execution using TPF.
>
> $\mathbb{H}_{1.3}$: Fetching and verifying the result from outsourced BGP queries uses at least 25% less bandwidth than local execution using TPF.

The second research questions introduces two additional hypotheses:

> $\mathbb{H}_{2.1}$: Using non-interactive proof systems, the time required to verify the completeness and soundness of the results of a BGP query, is independent of result set size.
>
> $\mathbb{H}_{2.2}$: Providing a computational integrity proof for execution of BGP queries requires time and memory that is 2 orders of magnitude larger compared to query execution using TPF.

4. Implementation

4.1. Framework

This paper will make use of the *Pequin* end-to-end toolchain introduced in the Pepper project, an initiative of NYU and UT Austin. The pepper project is a research initiative for making verifiable computations more practical [8]. The main workflow to generate a verifiable program and a proof of correct execution for some input is:

1. Writing a program $\mathscr{C}$ using an extensive subset of the C language
2. Compiling the verifier and prover which consists of translating the C source code to an arithmetic circuit. It produces the prover ($\mathscr{P}$), verifier ($\mathscr{V}$), prover key (pk) and verifier key (vk).
3. Generating input, I.
4. Running $\mathscr{P}$, which uses the input I and prover key pk to produce the output O and proof of correct execution π.

Any party can then verify $\mathscr{P}$'s work by running the verifier executable $\mathscr{V}$, taking as input vk, I, π and O.

Pequin allows for private prover input, which is hidden from the verifier. This is done using exogenous computations by calling the `exo_compute()` function. An auxiliary program then receives input from the prover and gets back arbitrary, nondeterministic data.

4.2. Considerations

During development of a program using the *Pequin* toolchain, some considerations have to be made in what parts of the C programming language can be used.

First of all, commonly used libraries are not available. The most important one, *stdint.h* with all of its constants, however, was re-implemented by the people behind the Pepper project. Unfortunately, strings are not available which poses a large restriction. String manipulation, common practice in any programming language, is not an option and even defining a static string is not possible. An array of `char` types is thus nothing more than an array of 8-bit integers.

Second comes the absence of data-dependent loops. Arithmetic circuits are always of fixed size and data-dependent loops would require a size that depends on the input, which is not possible using zk-SNARKs at the time of writing. Loops where the number of iterations is known at compile-time are possible. These get flattened beforehand, but `break` and `continue` statements are not possible as these are data-dependent.

4.3. Details

The system consists of four parties:

- **R**: A party requesting the solution to a BGP query.
- **DS**: A trusted data source (e.g. DBpedia).
- **AUX**: A group of auxiliary executables written in Python, potentially malicious and not trusted by **R**.
- **P**: A prover $\mathscr{P}$ and the entirely verifiable program $\mathscr{C}$ written and compiled using the *Pequin* toolchain. This party is not trusted by **R**.

The requesting party can be anyone desiring the solution to a BGP query. He provides the system with a BGP query and a random nonce. It receives the solution to the query from **P** and verifies the computational integrity of the whole system.

The data source in this solution is a TPF server with added functionality:

1. *Response digests*: A cryptographic hash for each TPF page.
2. *Nonce*: Each request accepts a nonce, provided by **R**. Response digests should be stored, along with their corresponding nonce, allowing for fast retrieval in the future.
3. *Digital signature scheme*: Upon query completion, the data source receives a hash of the concatenation of all the previous TPF page hashes. Using the nonce, it can check if this hash is indeed correct and if it does, it sends back a digital signature.

The auxiliaries were written in Python 2.7, specifically for this solution. They serve as a bridge for any network communications that have to be made from $\mathscr{C}$ by using the `exo_compute()` function. There are three Python scripts involved: a counter, a fetcher and a signer. All three accept input from $\mathscr{C}$, communicate with **DS** for information and provide output back into $\mathscr{C}$.

1. *Counter*: Returns the number of triples matching a given triple pattern.
2. *Fetcher*: Sends the nonce and triple pattern to **DS** and returns the received triples and page hash.
3. *Signer*: Accepts as input a digest of the concatenation of previous hashes and verifies it with **DS**. If all is correct, it returns the digital signature.

The auxiliary executables are not trusted by **R** and are thus a potential source for any malpractice. The digital signature scheme and cryptographic hashes introduced at the data source is what allows for the system to still provide verifiable computation of a BGP query.

P primarily consists of the computationally verifiable program $\mathscr{C}$. It combines all the logic from the previous parties and implements the TPF algorithm. It receives its input from **R** and relies on **AUX** to do the required communication with **DS**. By reading out the query, sending the proper requests to the counter and fetcher scripts and combining its results, the query solution is constructed. To ensure correctness of the auxiliaries, the TPF page hashes are concatenated and hashes using the `SHA-1` implementation in $\mathscr{C}$. By sending this to signer script, any malicious modifications made by the auxiliary scripts is either noticed by **DS** or afterwards by **R**.

4.4. Limitations

The considerations from Section 4.2 introduce some limitations to the universality of the system.

First of all, the dynamic iterator pipeline algorithm is recursive in nature and as recursive functions are data-dependent, this is not possible using the *Pequin* framework. The algorithm has to be converted to a linear one by inlining and has to be done separately for every different number of triple patterns. Possible workarounds are (i) using a non-recursive TPF algorithm, (ii) implementing separate functions for each amount of triple patterns or (iii) manually rewrite the generated circuit to re-use some parts. However, no non-recursive algorithm was found that outperformed the current one. Furthermore, implementing separate functions would bloat the amount of constraints to be met. Manually rewriting the circuit was out of scope for this paper.

The static nature of circuits also implies that every array definition must have a predefined length. This introduces limits in, for example, the length of the input query and the number of total results obtainable. Additionally, to keep the length of the input low, the concept of URIs and prefixes in SPARQL was removed. The aforementioned limits can, however, all be increased but at an increased cost for the prover.

5. Use cases

The back-end relying on non-interactive proof systems is still in a research phase and only practical in certain scenarios. The limitations introduced in the previous section illustrate this and Section 6 will further clarify this. In this section, a general blockchain model and two use cases that work well within these limitations will be given.

5.1. Proof systems inside blockchain

The immutability of Bitcoin relies on Proof of Work (PoW), a useless computation that proves some work was done and which can quickly be verified, all the while using an immense amount of energy. To mitigate this issue, research is being done in finding more useful PoWs and the solution of this paper is a perfect example.

Instead of mining cryptographic hash functions to proof some work was done, computing SPARQL queries using verifiable computation can used. A valid block then consists of the usual metadata, along with the prover's input and output and a proof of correct execution (PoW). This kind of structure for blocks allows for a decentralized, immutable and trusted data source for all kinds of Linked Data. Querying the chain for results is straight-forward and quick, while updating requires publishing a new block with an updated timestamp.

5.2. Public health care

Rural Africa is without a doubt in need of proper health care services. Start-ups and government supported initiatives trying to better the situation are omnipresent, but lack universality. They often focus on very specific problems and are only available in certain areas. The information is spread over multiple companies and thus a centralized source of information is still missing. With the blockchain principle from the previous section, a more general and scalable solution can be set up. Instead of multiple startups and government-aided initiatives covering a variety of subjects, one decentralized, immutable and reliable source of information is used. This information is delivered quickly to community health workers and its integrity is easily verified, which is an undeniable necessity when it comes to health care services.

5.3. Privacy sensitive computations

Verifiable computation delegation of SPARQL queries can also be used for private data processing purposes. Suppose the secret service needs to publish data concerning an international dispute, but does not want to expose their internal database. For this there exists no existing solution, but using Linked Data and the solution of this paper, they could assign a trusted worker node to compute the necessary results, without exposing any confidential information. Another scenario is the "download your data" tools offered by tech giants. These companies might have an incentive to exclude key data points for privacy reasons or profit. Verifiable SPARQL queries are an indisputable way of reporting all available user data to a client, without providing any information about other users on the platform.

6. Evaluation

6.1. Setup

The solution of this paper will be compared with Comunica v1.7.0 [9], which implements the TPF algorithm. It supports all of SPARQL, so only the limitations of this paper's solution are to be kept in mind. The absence of recursiveness in *Pequin* resulted in an implementation supporting exactly two triple patterns. Additionally, to keep circuit sizes low, the number of triples matching the minimum triple pattern in the BGP has to be kept low (20). With these constraints, four queries were set up, having 13, 6, 2 and 0 results respectively.

A network setup was not done in this paper and thus a simulation of network delays has to be made. Using the solution of this paper, getting the results to a BGP query requires fetching both the actual result set and a proof of correct execution. This was simulated by setting up a simple HTTP server on commodity hardware, serving these files.

Both systems under test were run on the High Performance Computing (HPC) infrastructure of Ghent University. In particular, the nodes used had a *2 x 12-core Intel E5-2680v3 (Haswell-EP @ 2.5 GHz)* processor architecture and 512 *GB* in physical memory.

6.2. Time

Table 1 shows that the running time of verification has no correlation to result set size and therefore confirms hypothesis $\mathbb{H}_{2.1}$. The time for Comunica only increases with the amount of results. Therefore, with the exception of BGPs with only one or an empty triple pattern, the speedup only increases for more complex queries and thus hypothesis $\mathbb{H}_{1.1}$ can be accepted as well.

6.3. Memory

Table 2 lists the maximum total resident size for both solutions. For this paper, this is reached by the Python script that verifies the digital signature and thus

# Results	Comunica	Paper	Speedup
13	$1.09s$	$0.16s$	$6.81X$
6	$1.03s$	$0.14s$	$7.36X$
2	$0.98s$	$0.14s$	$7.00X$
0	$0.96s$	$0.15s$	$6.40X$

Table 1. Timing comparison of Comunica vs. fetching and verifying for each query

includes some overhead due to the Python environment. It confirms hypothesis $\mathbb{H}_{1.2}$ and shows that memory usage is independent of result set size.

# Results	Comunica	Paper	Gain
13	$65.80MB$	$11.51MB$	83%
6	$62.65MB$	$11.51MB$	82%
2	$61.78MB$	$11.51MB$	81%
0	$61.64MB$	$11.51MB$	81%

Table 2. Memory comparison of Comunica vs. fetching and verifying for each query

6.4. Bandwidth

Due to the nature of the TPF algorithm, the amount of network requests that Comunica makes scales directly with the number of results. The solution introduced in this paper requires only 2 requests: fetching the proof and the results. Therefore, if the requesting party has to fetch the results with limited bandwidth, using Comunica will be slower unless the number of requests made is smaller than 2. Ignoring these very simple queries, hypothesis $\mathbb{H}_{1.3}$ can be accepted.

6.5. Prover

The solution introduced in Section 4 outperforms Comunica when it comes to simply verifying. However, for the proving party, it comes at a greater cost. Proving the integrity of the previous queries took on average 42 minutes and $20.68\,GB$. Based on these results and the previous tables, hypothesis $\mathbb{H}_{2.2}$ has to be rejected as the prover requires time and memory that is 3 orders of magnitude larger.

7. Conclusions

This paper introduced verification of BGP queries executed by untrusted third parties. By moving the computational load to a computationally strong and willing third party, both the data server and the client can relax their resource requirements. Using non-interactive proof systems and the TPF algorithm, the computational integrity of the query execution can be efficiently verified.

BGP queries with two triple patterns were compared with naive local re-execution and resulted in a verification system that is 7 times faster and uses 60% less memory. Additionally, the amount of network requests that need to done is static and does not depend on the query complexity. It was shown that the speedup and bandwidth usage will only become more favorable as query complexity grows.

However, to make the introduced solution practical, additional research needs to be done. This includes making the solution fully SPARQL-compliant by reintroducing URIs and flexible data sources. Additionally, improving prover scalability is left open for future work. Approaches include decreasing the size of the generated circuit and implementing other promising proof systems.

References

[1] C. Buil-Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche, “SPARQL Web-Querying Infrastructure: Ready for Action?” in *The Semantic Web – ISWC 2013*. Springer, Berlin, Heidelberg, oct 2013, pp. 277–293.

[2] R. Verborgh, M. Vander, S. P. Colpaert, S. Coppens, E. Mannens, and R. Van De Walle, “Web-Scale Querying through Linked Data Fragments,” 2014.

[3] Protocol Labs, “Filecoin: A Decentralized Storage Network,” 2017.

[4] D. Bhowmik and T. Feng, “The multimedia blockchain: A distributed and tamper-proof media transaction framework,” in *2017 22nd International Conference on Digital Signal Processing (DSP)*. IEEE, aug 2017, pp. 1–5.

[5] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.

[6] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, “From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again,” in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS ’12*. ACM Press, 2012, pp. 326–349.

[7] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, “Scalable, transparent, and post-quantum secure computational integrity,” 2018.

[8] NYU and UT Austin. (2017) Pepper: toward practical verifiable computation. https://www.pepper-project.org/.

[9] R. Taelman, J. Van Herwegen, M. Vander Sande, and R. Verborgh, “Comunica: A Modular SPARQL Query Engine for the Web.” Springer, Cham, oct 2018, pp. 239–255.

Contents

List of Figures

List of Tables

Listings

List of acronyms

ADS	Authenticated Data Structure
ALU	Arithmetic Logic Unit
ASCII	American Standard Code for Information Interchange
ASIC	Application-Specific Integrated Circuit
AWS	Amazon Web Services
BGP	Basic Graph Pattern
CA	Certificate Authority
CLI	Command-line interface
DPC	Data Protection Commission
DPKI	Decentralized Public Key Infrastructure
ECDSA	Ellipitic Curve Digital Signature Algorithm
ECRH	Extractable Collision-Resistant Hash functions
GDPR	European General Data Protection Regulation
HPC	High Performance Computing
HR	Human Resources
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
MLaaS	Machine Learning as a Service

NIZK	Non-Interactive Zero-Knowledge
PCP	Probabilistically Checkable Proof
PoS	Proof of Stake
PoW	Proof of Work
QAP	Quadratic Arithmetic Program
R1CS	Rank-1 Constraint System
RAM	Random Access Memory
RDF	Resource Description Framework
SHA-1	Secure Hash Algorithm 1
SPARQL	SPARQL Protocol And RDF Query Language
SQL	Structured Query Language
TP	Triple Pattern
TPF	Triple Pattern Fragment
URI	Uniform Resource Identifier
UX	User Experience
WWW	World Wide Web
zk-SNARK	Zero-Knowledge Succinct Non-interactive ARgument of Knowledge
zk-STARK	Zero-Knowledge Scalable Transparent ARgument of Knowledge

Chapter 1

Introduction

1.1 Background

”Data is the new oil”, a quote that has become very familiar to a lot of people and is widely credited to Clive Humby, all the way back in 2006. Whether this quote is a suitable analogy or not, it’s undebatable that rapid data growth is a thing. This is confirmed in Figure 1.1, which gives a forecast of the Big Data market size up until 2027.

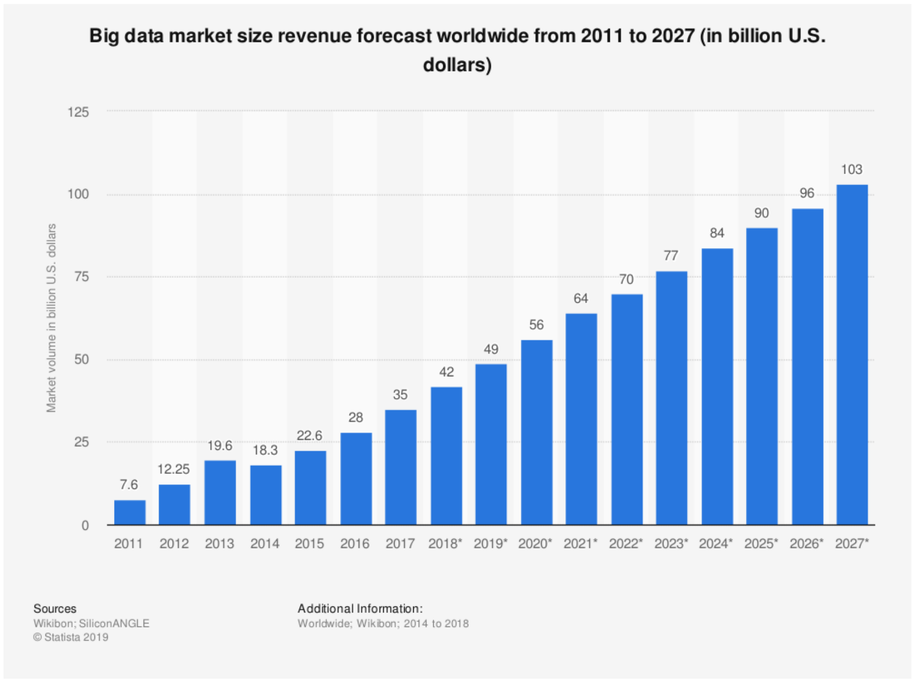


Figure 1.1: Forecast of Big Data market size based on revenue (in USD) [1]

Data on its own often does not have any significant meaning. In order to produce meaningful results, careful aggregations need to be made and projected onto meaningful visualizations. For social media giants such as Facebook or Twitter, this has become a major focus and is in fact their major source of revenue. They are happy to collect any data that is available as this gives them a better understanding of their user base, after which they can then sell this knowledge at a higher price.

In 2009, Tim Berners-Lee said ”The web as I envisaged it we have not seen yet” [3], meaning that the web was not yet used as a big web of data but rather a document store. 10 years later and this has changed dramatically and as previously mentioned, is the main source of revenue for a lot of internet companies. This enormous growth of data over the last years, has come with some serious concerns. Major privacy breaches such as the Facebook-Cambridge Analytica scandal or the Equifax security failure has awakened the public about the importance of their privacy. New regulations such as the European General Data Protection Regulation (GDPR) are an attempt to ease these concerns and provide a way to control the data that these companies possess about you. However, even these laws do not stop the social media giants from doing everything within their power to keep your data [4].

For these reasons, Tim Berners-Lee launched Solid[1]. The environment in which it operates is fundamentally different from what is the case for a company such as Facebook. It is based on the Semantic Web, an extension of the World Wide Web and re-defines the purpose of the Web. Currently, we exchange our data in order to be part of a bigger network where we can watch friend’s or other people’s activity. This data is thus stored by some internet giant, which has already been shown to not be very privacy-sensitive. To add to this, users are not in control of their data. They cannot change their data at will and this makes a big part of the web essentially read-only.
In Solid, the data is not in hands of the internet giants, but rather stored by the users themselves. The revenue model of the media giants is thus useless in a scenario like this. Data is both readable and writable and in complete control of the users. Applications using Solid could provide exactly the same perceived value as currently done by Facebook, but instead of fetching data from their own data centers, they aggregate it from all their user’s data centers (called ”data pods”).

The Solid project also comes with some technical challenges. For example, the data is not stored in one data center but instead each user has one or more pods, spread around the world. Fur-

[1]`https://solid.inrupt.com/`

thermore, these pods might use different representations of their data, making interoperability a major task in the context of the Semantic Web. Taking the former two difficulties into account, one can easily understand that the query engine for this kind of data quickly becomes very complex. In fact, availability of the servers that provide this kind of aggregation is a known problem [5]. This availability issue led to the concept of offloading a part of the complexity from the server to the client [6], allowing the servers to maintain higher availability rates.

However, moving this complexity to the client does not come easily. Today's applications provide meaningful information such as likes on a photo or number of retweets by just one API call, this is quick and low-effort for the client. Delivering the same information using the aforementioned decentralized data storage model, however, is not and involves querying possibly hundreds of data pods. Taking network delays into account, this can quickly take several seconds to even minutes, which is clearly not ideal User Experience (UX). Additionally, these data aggregation tasks on the client-side require resources that might not be available on the user's device. With the recent popularity concerning the Internet of Things (IoT), power consumption has become a major design factor and thus not every connected device might have the resources to do the necessary computations.

This limited computational power is clearly a delaying factor in the adoption of new systems like Solid. However, massive amounts of computational power are readily available in big data centers, provided by third parties. These heavy-duty devices could easily accept requests from user devices to aggregate the information for them and then send back just the results. While this sounds promising, this also tends to go back to what already exists: internet giants that have all our data. So how can we trust these third parties to firstly, not misuse our data and secondly, provide us with the correct data aggregations? This last concern is the main focus of this thesis. An approach to verify that others executed these data aggregations as expected will be presented and discussed in detail.

1.2 Scope

The Semantic Web works on the principle of Linked Data [7], in which the Resource Description Framework (RDF) is often used to represent this data. To pull relationships from this data, a query language named SPARQL was standardized [8]. Solving queries in this language is exactly

the kind of work that could be outsourced to third parties. In this thesis, a method to verify the computational integrity of SPARQL query execution will be analyzed.
Basic Graph Patterns (BGPs) are the foundation of SPARQL and is therefore the main focus of this thesis. During the introduction in the previous section, transferring load from the server to the client was briefly mentioned. In this thesis, we will use the concept of Triple Pattern Fragments (TPFs) to allow for this [2]. It is a new approach in which servers offer a simple query interface to the client, which then solves the query locally. More specifically, the query solver proposed in the work by Verborgh et al. called a dynamic iterator pipeline will be used [2]. This algorithm will be implemented in a manner that can be verified by someone else, without re-executing it from scratch. The exact meaning of verification and correct execution of this algorithm will be defined further on.

1.3 Outline

Chapter 2 will give a general introduction to the technological concepts used throughout this thesis while also giving some insight into related work concerning verifying data queries. Chapter 3 will formalize the problem at hand while Chapter 4 covers the implementation of a solution to the aforementioned problem, including the considerations made and limitations discovered throughout the process. Following through, Chapter 5 discusses two possible use cases for this solution together with a novel blockchain model. Chapter 6 evaluates the solution in terms of memory consumption, execution time and bandwidth requirements with relation to executing the query yourself. Lastly, possible future work, the relevance and conclusion of this thesis are discussed in Section 7.

Chapter 2

Related work

2.1 The Semantic Web

2.1.1 Linked Data

The Web today is enormous, there's an estimated size of at least 5.5 billion pages and that's only the indexed pages by search engines [9]. When looking at the Web today, pages are built to ease human-readability, not to mention JavaScript and the CSS3 additions. These kind of web pages are hard for a machine to interpret reliably. For example, for a human reader, finding the family tree connections on a Wikipedia page is easy. For a machine, on the other hand, it is a lot harder as there are many ways to describe these connections and differentiating between them quickly becomes a cumbersome task to write code for.
Tim Berners-Lee, the inventor of the World Wide Web (WWW) (along with Robert Cailliau) and the HyperText Transfer Protocol (HTTP) envisioned the Web in a different way. He wanted to bring structure to the Web, a web of data through which software agents could scroll and extract complicated information relationships [10].
This vision of the web lead to the introduction of the term "Linked Data" [7]. This is data as used in the Semantic Web. Data in which each piece is identified by an HTTP URI and in which relationships are made through these URIs.

2.1.2 Resource Description Framework (RDF)

In the context of the semantic web, there is of course also the need for a way to represent the data. In a connected web of knowledge, representing data using a predetermined schema is unmanageable as too much information of a different nature is to be included. For example, imagine a traditional Structured Query Language (SQL) server with a schema including a user profile along with his work history. If we represent a user with his gender, name and age, we could answer questions along the lines of "What is the gender distribution of people working in HR?". With this fixed schema, however, we cannot derive information that goes beyond the imposed relationships (e.g. "Where do most people in the farming industry live?"). For this, versatile knowledge-representation frameworks are needed. For the Semantic Web, this is the Resource Description Framework (RDF) [11].

With all this versatility also comes a downside though. If one database uses `lastName` as its identifier for a person's family name and another uses the semantically equivalent `surname`, how can a computer interpret both as the same? This why ontologies are essential. An ontology is a vocabulary, defining the meaning and relationships for certain keywords. It categorizes data into classes and sub-classes, forming a hierarchy that can be interpreted. For example, the DBpedia Ontology defines classes such as `Person` and `Place` [1]. Digging deeper, entities such as `birthPlace` are defined as well, which are then potentially considered subclasses of `Place`.

RDF is a knowledge-representation system that allow dynamic schema definitions and incorporates features to ease data merging coming from different schemas. It exists of structures called "triples". A triple describes the relation between two entities and consists of a subject, predicate and object. The predicate is what defines the relationship between the subject and the object and is defined by one of the classes of an ontology. Using these triples, you can build any relationship possible. For example, saying that Barack Obama was born in Honolulu (without any ontologies or URIs) would be done as illustrated in Table 2.1:

Subject	"Barack Obama"
Predicate	"birthPlace"
Object	"Honolulu"

Table 2.1: An example of how a relationship using a triple is defined in plain text

As previously mentioned, Linked Data (and thus also RDF) uses HTTP URIs to represent everything. This means that triples are also defined using HTTP URIs, even the relationship itself (the predicate). The example from before, using the DBpedia ontology, is thus changed accordingly and depicted in Table 2.2. Note that the predicate is now an object defined in

[1]`http://dbpedia.org/ontology/`

the DBpedia ontology, while Barack Obama and Honolulu are no longer plain strings but an HTTP URI. This is the main idea surrounding Linked Data: everything is an HTTP URI. You can follow any one of them and you will get back extra data concerning that object, even the relationships can be looked up and will explain what exactly the relationship represents.

Subject	http://dbpedia.org/page/Barack_Obama
Predicate	http://dbpedia.org/ontology/birthPlace
Object	http://dbpedia.org/page/Honolulu

Table 2.2: An example of how a relationship using a triple is defined using HTTP URIs

2.1.3 SPARQL

SPARQL is an often used language to query RDF data [8]. It is somewhat similar to normal SQL, in that it has a `SELECT` form to define the return variables. It also has other query forms such as `CONSTRUCT` or `DESCRIBE`, but those are out of scope for this thesis. The `WHERE` clause of a query consists of triples, but each component of the triple can be a variable which, in SPARQL, starts with a question mark. Triples that are part of a query and thus contain variables are also called Triple Patterns (TPs). Variables get bounded during query execution and can be used in the `SELECT` statement.

The query language is too extensive to explore in this short introduction, but for the purpose of this thesis, we do need the concept of Basic Graph Patterns (BGPs). A BGP is a set of triple patterns that are all interlinked. Consider the example in Listing 2.1. This query is a simple `SELECT` query containing only one BGP in its `WHERE` clause. In the beginning of the query, prefixes are defined. These are meant to make the query more readable and easier to write. In the example, `dbpedia-owl:Agent` actually means `http://dbpedia.org/ontology/Agent` by simple substitution. Listing 2.1 would return all entities of type `Agent` (defined in the DBpedia ontology) that have their `ground` in the country of Fiji. Here, `ground` might not be intuitively clear what it means, but due to the design of the Semantic Web, you can simply follow its link and get an explanation (`http://dbpedia.org/ontology/ground`). By doing this, it becomes clear that it means the home ground of a soccer club.

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/resource/>
SELECT ?p WHERE {
  ?p a dbpedia-owl:Agent.
  ?p dbpedia-owl:ground dbpedia:Fiji.
}
```

Listing 2.1: A SPARQL query

2.1.4 Triple Pattern Fragments (TPF)

2.1.4.1 SPARQL endpoint availability

As mentioned during the introduction, executing these queries can be expensive and time-consuming as a lot of network requests have to be made if the data lies beyond the information contained in the internal database of the SPARQL endpoint. Ideally, an endpoint should accept any kind of query (up to the most complex ones) and do this for an unknown number of concurrent requests at any time. These characteristics lead to a lot of SPARQL endpoints being down and not available. In fact, out of a survey of 427 endpoints, only a third have an availability rate above 99% and half of them do not even reach 95% [5]. This is a very bad availability rate, 95% comes down to more than 18 days of non-availability each year. This is an enormous difference compared with the availability rate of S3 from Amazon Web Services (AWS), which is 99.99% and translates to 53 minutes each year [12].

2.1.4.2 Load balancing

The concept of Triple Pattern Fragments (TPFs) is to reduce the load on the servers [2]. A TPF is, in essence, the result of a SPARQL query with only 1 triple pattern, paginated. A server responding to TPF queries thus returns paginated results with each result being a triple matching the one in the query. A sample TPF server is hosted at `http://fragments.dbpedia.org/`, it allows requesting data using exactly one triple pattern (potentially leaving some fields out, indicating a variable that can be bound). For example, the first triple pattern in Listing 2.1 could be the input on this website, when doing this, it returns more than 1.8 million results with 100 triples per page.

An added benefit of using Triple Pattern Fragments is caching. SPARQL query results are specific to the exact query, while the triples from the TPF server are quite generic. In the example from Listing 2.1, the first triple pattern `?p a dbpedia-owl:Agent` might be a very common triple pattern in queries in very different contexts. The results can thus be cached very efficiently.

Figure 2.1 shows parts of the evaluation done in the original TPF paper [2]. Here it can clearly be seen that TPF reduces the load on the server, as all other full SPARQL endpoints run at 100% CPU usage with around 50 concurrent clients. Another concept shown is the cache network traffic, which is clearly more effective using TPF. This means the main load of executing SPARQL queries can be taken away from the critical data servers. With TPF, they merely serve as a data source, while solving the query remains up to the client. How exactly this is done is explained in the following section.

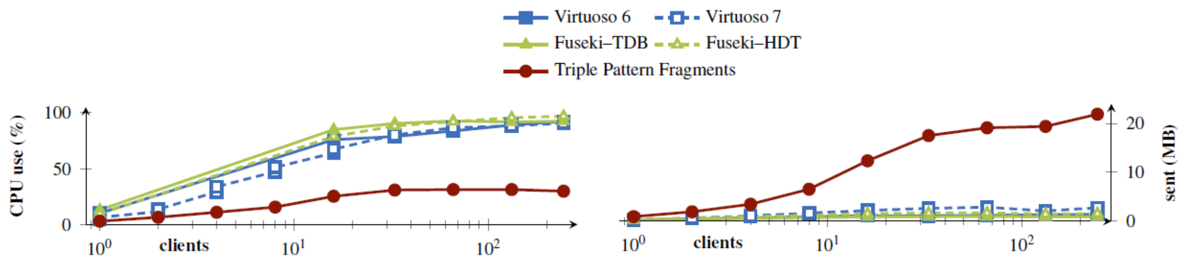


Figure 2.1: Server processor usage per core (left) and cache network traffic (right) [2]

2.1.4.3 Dynamic iterator pipelines

The client has access to a data source that provides triples in response to a triple pattern query. It can thus access any data that the data source contains. In fact, it can get all the data from the data source by submitting the triple pattern `?s ?p ?o`. An iterator arranged in a pipeline is a common query evaluation technique [13] and can also be applied for SPARQL queries. The dynamic iterator pipeline algorithm is an algorithm developed to evaluate BGPs [14]. The details of it will not be discussed here, but a simple example shows the general concept.

Consider the query in Listing 2.2, which finds all artists born in cities named ”York”. First, the first page of each triple pattern is requested from a TPF server (e.g. `http://fragments.dbpedia.org/2015/en`). Using the response, the total count of triples matching this pattern is read from the metadata. This results in 145, 878 for *tp*1 and 1, 137, 061 for *tp*2, whereas only 32

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://dbpedia.org/foaf/0.1/name>
SELECT ?p ?c WHERE {
  ?p a dbpedia-owl:Artist.                        # tp1
  ?p dbpedia-owl:birthPlace ?c.                   # tp2
  ?c foaf:name "York"@en.                         # tp3
}
```

Listing 2.2: A SPARQL query illustrating the dynamic iterator pipeline, part 1

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
SELECT ?p ?c WHERE {
  ?p a dbpedia-owl:Artist.                        # tp1
  ?p dbpedia-owl:birthPlace dbpedia:York.         # tp2
}
```

Listing 2.3: A SPARQL query illustrating the dynamic iterator pipeline, part 2

for *tp*3. The algorithm continues to fetch the remaining pages for the triple pattern that had the least results. In this case there are no other pages to fetch as *tp*3 only has 32 results and one page contains 100 triples. It then continues recursively for each triple in the result set, but with a BGP where *tp*3 is removed and where the variables from *tp*3 have been bound to the other triple patterns. The resulting remaining query if we consider the city York in North Yorkshire, England can be seen in Listing 2.3.

*tp*1 now has 236 results, while *tp*1 still has the same number of triples. The algorithm therefore chooses *tp*2 to continue the same as before, now resulting in a BGP with only one triple pattern and finding the final results. Note that if anywhere in the algorithm, a triple pattern fragment returns zero results, the query has no results in its solution.

2.2 Blockchain

2.2.1 Purpose

The purpose of blockchain technology is primarily a distributed database in which the data cannot be tampered with. It serves as a public ledger and its contents are agreed upon by the members of the network. This data is permanent once added to the database and it cannot be changed as long as the network maintaining it, is stable. It finds its use in many applications, such as financial ones where privacy and integrity of data is key, but also non-financial ones like a decentralized storage service [15, 16] and media authenticity purposes [17, 18].

2.2.2 Cryptographic hash functions

The working principles of blockchain cannot be explained without first explaining cryptographic hash functions. They have been around for a long time and are used in for example message integrity verification and storage of passwords. Cryptographic hash functions are a subcategory of normal hash functions. A hash function is a function that produces a deterministic, fixed sized output for given input (often called the ”digest” or ”hash”) and are used in for example the implementation of caches and hash tables. The cryptographic variant of these functions are those that possess non-continuity in their output and have the collision resistance property. Both these properties and other important properties in the context of blockchain are explained below.

- **Deterministic** Given x, $y = h(x)$ is always the same.
- **Collision resistance** It is computationally infeasible to find any x_1 and x_2 for which $h(x_1) = h(x_2)$.
- **Non-continuity** Given x_1 and x_2, $h(x_1)$ and $h(x_2)$ should appear uncorrelated.
- **Pre-image resistance** Given y, it is computationally infeasible to find x from its hash value $y = h(x)$, also known as non-invertability.
- **Second pre-image resistance** Given x_1, it is computationally infeasible to find x_2 for which $h(x_1) = h(x_2)$.

2.2.3 Structure

The basic structure was first proposed by Stuart Haber and W. Scott Stornetta to digitally time-stamp documents that later could not be falsified [19]. It consists, as the name depicts, of a chain of blocks, distributed in a network (either public or private). New data arrives in blocks, which are published to the entire network and upon which each member checks the validity of this new block. If the majority agrees, the new block is then part of the chain. A general structure is drawn in Figure 2.2. The two branches that emerge are a temporary disagreement in the network, where not everyone agrees on which block is the next valid one. This disagreement, however, is quickly resolved by the part of the network that advances the quickest: the branch gets abandoned as a newer, longer chain gets created. The first block of a blockchain (the green node in Figure 2.2) is called the genesis block.

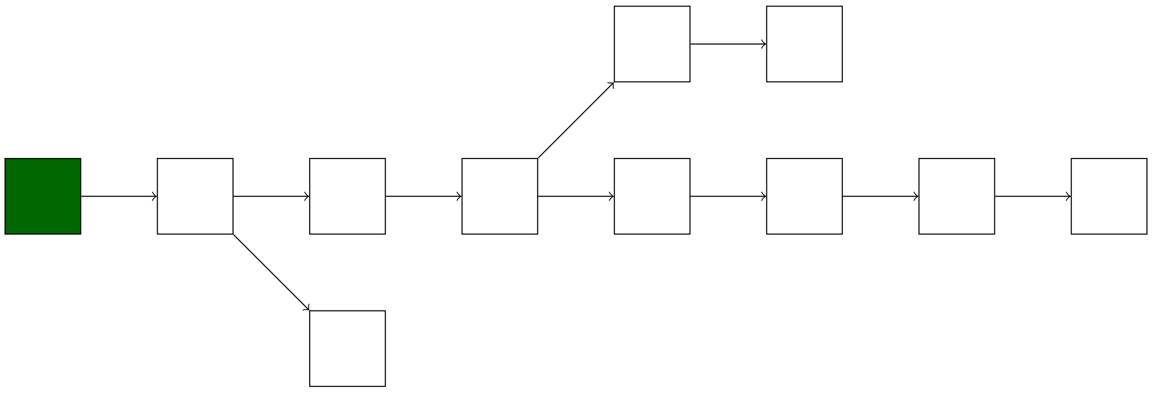


Figure 2.2: Illustration of a blockchain structure with the genesis block in green

The immutability of this structure is due to the concept of hashing and the need for a solution to a difficult puzzle. Figure 2.3 is a very general diagram indicating how blocks are defined and linked together. First of all, the block of course contains the actual data that is to be stored. Secondly, the block also contains metadata which, depending on the application, can be all kinds of things. This mainly consists of versioning information, a timestamp and the solution to the puzzle mentioned before. As previously stated, adding a new block requires approval of the network. This is done by validating the data within the block, but also by checking if the solution to a predetermined puzzle is correct. These puzzles are computationally hard for modern computers and are directly related to the block that is to be added to the chain. This means the creator of new data should first compute the solution to a puzzle in order for his block to be submitted to the network. Lastly, each block also contains the cryptographic hash of the previous block's contents.

Given these contents, the immutability of a blockchain is easily derived by considering an example on Figure 2.3. Two properties need to be taken into account. First, if changes are made to block $N-1$, the hash of this block changes. However, the hash of block $N-1$ is referenced in block N and thus the contents of block N also change, along with its hash. This further propagates to block $N+1$ and its successors until the end of the chain has been reached. Secondly, the solutions to the puzzle required for submitting a block are stored within the block's metadata and are directly linked to the content within that block. Changing the content of a block therefore invalidates the solution to the puzzle. In the example, this means that all blocks from block $N-1$ onward are now invalid. Changing the data in one block already submitted in the chain thus invalidates all subsequent blocks. In order for the network to accept this change, all solutions to the puzzles have to be recomputed. Recomputing the solutions is computationally infeasible and requires at least 50% of the network's computational capacity. This is exactly what causes a blockchain to be immutable and unless the network nodes cooperate, data is stored forever.

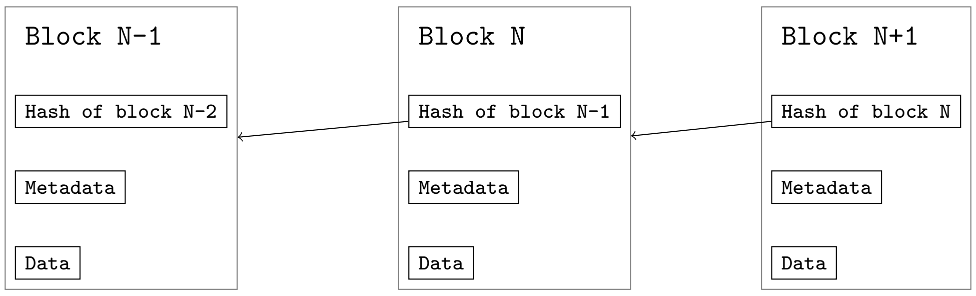


Figure 2.3: Figure showing how subsequent blocks are linked together

2.2.4 Bitcoin

Bitcoin is perhaps the most popular application of blockchain, as it remains the most dominant cryptocurrency on the market[2]. It was the first practical proposal for a crypto currency that found a solution for the double spending problem [20]. It is the first of many digital currencies and part of what's known as "Blockchain 1.0".

The data stored in Bitcoin's distributed ledger is transactions made in its cryptocurrency (con-

[2]`https://coinmarketcap.com/`

veniently also called bitcoins and denoted by the symbol ₿). Each block contains multiple transactions of money and are stored in a Merkle tree for efficient verification strategies [21]. Upon gathering enough transactions to fill one block, the block can be published to the network, but first the solution to the puzzle has to be found. The puzzle in Bitcoin is called its "Proof of Work" while the act of solving it is what is known as "mining".

The puzzle for Bitcoin consists of finding a cryptographic hash for a block that is conform a certain difficulty. The difficulty is set as the number of zeroes that the hash starts with. In order to support this scheme, a block's metadata contains a nonce. This nonce can be set arbitrarily in order to find the correct hash. For example, finding the solution to the puzzle if the difficulty is set to 5 is equal to finding a nonce that results in a block's hash starting with 5 zeroes. Formally this means finding x for which $h(x|z) < C$, in which z is the content of the block, x is the nonce to be found and C is a constant depending on the difficulty and hash function.
Due to the pre-image resistance property of cryptographic hashes, this is a computationally hard problem and is only possible by brute-force (i.e. trying all the values until a solution is found). Validating a published block is quick and consists of checking the validity of the transactions within the block and checking if $h(B) < C$.

The structure of a blockchain and its accompanying requirements for a valid block make for a rigid and permanent public ledger. As mentioned earlier on, for adversaries to publish false data, he would have to be in control of more than 50% of the network's computational power.

There are many other aspects to Bitcoin and their derivatives (called "altcoins"), including attacks and their defenses but these are out of scope for this thesis.

2.2.5 Blockchain 2.0 and onwards

Blockchain 1.0 is the generation of digital currencies where Bitcoin is the most popular, but certainly not the only one (Litecoin[3], Ripple[4], ...). They built the fundamentals of blockchain technology and for the first time, enabled online money. New concepts, however, have arisen from this technology.
Blockchain 2.0 is the term used to describe applications of blockchain that apply the concept of "Smart Contracts". Smart contracts are small programs that reside on a blockchain in much

[3]`https://litecoin.com`
[4]`https://www.ripple.com/`

the same way a Bitcoin transaction does. The most prominent example of a blockchain built for Smart Contracts is Ethereum[5]. They find their use in a lot of markets: real estate, healthcare, copyright protection, ...
Next steps include better scalability, privacy and user-friendliness to build applications as know them in a distributed fashion. There's a plethora of proposals and actual applications out there that each have their own advantages and disadvantages compared to Bitcoin. Some examples will be given in Section 2.3.

2.3 Zero-knowledge proofs

2.3.1 The principle

A zero-knowledge proof consists of two parties: the prover $\mathcal{P}$ and the verifier $\mathcal{V}$. The prover's goal is to convince $\mathcal{V}$ that he knows a value x, without revealing x. They are quite a mathematical concept and is most-easily explained by means of a simple example. Quisquater et al. provided an excellent one called the Ali Baba cave, but a shorter one will do as well [22].
Consider you, the prover, have two balls (red and green) and a red-green color-blind friend, the verifier. Suppose you wish to prove to him that you can tell the difference between the two balls, which he cannot distinguish. The proof goes like this: your friend hides the balls behind his back and optionally shuffles them. He then shows you one of the balls, on which you answer with its color. This procedure is repeated many times and each time he shows a ball, you tell him the color. Your friend can easily tell if you are lying as he knows if he showed the same ball as the round before. Each round, the chance of you getting the color right by chance lowers by 50%. This is called soundness of the system. After sufficient rounds have been carried out, your friend should become convinced that you can, in fact, distinguish the color of the balls. This is referred to as the completeness property of the system.

Properties Zero-knowledge proofs exhibit three properties:

1. **Completeness** If both parties follow the protocol and the statement is true, an honest prover should be able to convince the verifier of this fact.

2. **Soundness** If the statement is false, the probability that a dishonest prover can convince the verifier that it is true, is negligible.

[5]`https://www.ethereum.org/`

3. **Zero-knowledge** The verifier cannot know anything except for the statement, no additional information is exchanged.

2.3.2 Interactivity

The example touched upon in Section 2.3.1 is an example of an interactive proof. The prover and verifier repeatedly exchange information and in the end, the verifier is convinced that the prover's claim is correct.
Zero-knowledge proofs can be divided into interactive and non-interactive ones.

Interactive Interactive proofs are the most common type. They require a series of communication between the prover and the verifier. Anyone outside of these two parties should not be able to participate and if this third party also wanted to verify the statement, he would have to start communicating with the prover too. This scheme does not require any prior knowledge between the prover and the verifier. However, it does require a lot of effort from the prover, as he has to do the same process over and over again if multiple people want to verify his statement. The first interactive zero-knowledge proofs date back to 1988 and rely on typical cryptographic concepts such as the complexity of integer factorization [23, 24].

Non-interactive Non-interactive proofs require no such communication phase and thus the proving party can publish a single "transcript" that serves as prove. Any party can then verify the claim without interacting with the prover. Non-interactive zero-knowledge proofs often require a setup phase in which a common reference string needs to be defined but on which no party has influence, nor does it know the private parameters to create this reference string. If some party would know these parameters, he would be able to create false proofs. This makes non-interactive proofs harder to accomplish and careful considerations need to be made. The "transcript" is usually a hash and in general, non-interactive proofs are constructed using interactive ones by means of the Fiat-Shamir heuristic [25].
The most popular example are zk-SNARKs and will be discussed in the next section [26]. More recently, zk-STARKs were introduced [27]. They offer transparency by requiring no "trusted" setup phase and will be touched upon in Section 2.3.4.

2.3.3 zk-SNARKs

A zk-SNARK is an acronym for Zero-Knowledge Succinct Non-interactive ARgument of Knowledge. Here, *succinct* indicates that the size of the proofs are small and can be verified in a matter of milliseconds, making them suitable for use in a blockchain scenario. *Arguments of Knowledge* is best understood as a computationally sound proof of knowledge.

A zk-SNARK is a non-interactive proof system that has the usual computational soundness and completeness along with being zero-knowledge. What differs zk-SNARKs from other proof systems is that it also requires *extractability.* This property, informally, means that there exists an extractor algorithm that takes the same inputs as the prover and produces a witness. Traditional proof systems merely require to proof that such a witness "exists". zk-SNARKs on the other hand, require that such a witness can be extracted from a prover, eliminating the possibility that the prover generated a proof in a non-standard way without knowing the witness [26]. Cryptographic hash functions that exert this property are called Extractable Collision-Resistant Hash functionss (ECRHs).
The construction relies on the newly introduced ECRHs, introduced by Di Crescenzo and Lipmaa [28]. It builds further on the combination of Probabilistically Checkable Proof (PCP) [29] and Merkle Trees [21]. The zero-knowledge property is achieved by applying their construction on top of Non-Interactive Zero-Knowledge (NIZK) arguments, introduced by Groth in 2010 [30]. The individual parts in the construction of zk-SNARKs find their home in the cryptographic and complexity theory community. Some parts required to understand the backbone of this thesis will be explained in Section 2.4.2.

2.3.4 zk-STARKs

zk-STARKs, also known as Zero-Knowledge Scalable Transparant ARguments of Knowledge is the more recent argument of knowledge and possible successor of zk-SNARKs [27].
One of the main issues with zk-SNARKs is the trusted setup phase. The randomness used in the scheme is not publicly known and the parameters for it should ideally be destroyed and known by nobody. If an adversary knows the parameters used to generate the common reference string (the randomness), he can forge proofs at will and nobody would know. zk-STARKs are *transparent*, meaning that randomness used in the protocol is public and thus no complicated setup phase is necessary.
Furthermore, the scalability of zk-SNARKs can be improved, especially the prover's. zk-STARKs

introduce *full scalability*: scalability for both the verifier and the prover. They achieve 10x faster prover running time and better verifier scalability if the one-time setup time of a zk-SNARK is accounted for.
Lastly, zk-SNARKs rely on elliptic curve pairings, which can be broken if quantum computers ever really do come through. zk-STARKs do not rely on any cryptographic assumptions that can be broken using quantum algorithms and is therefore (currently) post-quantum secure.

2.3.5 Applications

2.3.5.1 Zcash

Zcash[6] is inarguably the most known applicaton of zero-knowledge proofs. Zcash uses the zk-SNARK technology to offer a privacy-preserving digital currency. The core purpose is exactly the same as Bitcoin and it uses blockchain as a public ledger. Transactions in Bitcoin, however, are public and although addresses are kind of private in the sense that they do not directly link back to actual people, methods exists to infringe on this privacy [31]. Examples are taint analysis and monitoring network node ip addresses.
Zcash is different in that the elements stored on the blockchain are zero-knowledge proofs instead of money transactions. The amount of money that is transferred from and to who is thus not known. Each money expenditure is done by providing a zero-knowledge proof, stating that: (i) the money has not been spend yet, (ii) the creator has authority to spend the money and (iii) that the quantity and type of input and output of the transaction are the same.
Note that Zcash also has a transparent mode of operation in which transactions are not hidden. In that case Zcash behaves exactly like Bitcoin.

2.3.5.2 Others

Zero-knowledge proofs find their use in many other applications. Ethereum[7], for example, has already implemented the math primitives required for using zk-SNARKs and ZoKrates is a toolbox for zk-SNARKs on Ethereum [32]. Loopring[8] is a decentralized crypto exchange platform, using zk-SNARKs on top of Ethereum. Coda uses zk-SNARKs in a recursive way to design a cryptocurrency that has better throughput than Bitcoin while remaining efficient for new users to verify the chain state [33]. Bulletproofs are another zero-knowledge proof scheme

[6]`https://z.cash/`
[7]`https://www.ethereum.org/`
[8]`https://loopring.org/`

and do not require a trusted setup phase [34]. It uses range proofs to validate transactions and is used in blockchains such as Grin[9] and Monero[10].

2.4 Computational Integrity and Privacy

2.4.1 Definition

2.4.1.1 Computational Integrity

The Cambridge Dictionary defines "integrity" as "the quality of being whole and complete"[11]. In the same way that buildings can have structural integrity and that foreign keys in a database are enforced using referential integrity, unobserved parties can operate with computational integrity. Computational integrity comprises the common problem where an unobserved and distrusted party $\mathcal{P}$ executes a computation $\mathcal{C}$ on input x, resulting in output $y = \mathcal{C}(x)$. Said party might have the incentive to provide the more favorable output $y' \neq y$ and so computational integrity systems ensure that $\mathcal{P}$ undoubtedly reports $\mathcal{C}(x)$ [35].
Computational integrity is in close relation with computation delegation, especially in cloud environments. To give one rather recent example: Machine Learning as a Service (MLaaS) using AWS. Delegating the, often expensive, steps in designing a machine learning model is now commonplace. However, the data that is necessary for this might be confidential and thus we need to ensure that AWS keeps this confidential. In other scenarios, or even in the case of AWS, the delegatee might profit of misreporting the output, for example to increase ad-income. Computational integrity can provide a solution for these scenarios.
If the data on which $\mathcal{P}$ computes y is public, a first and naive solution would be for anyone interested in verifying $\mathcal{C}$ to redo the same computation. This party is often called the verifier ($\mathcal{V}$) and this scenario is present in a lot of places. For example, re-calculating your grocery shopping list or the validation strategy in the Bitcoin network in which all nodes verify a new block separately [20]. This naive way of verifying, however, requires the verifying party to spend the same amount of time ($\mathcal{T}_{\mathcal{V}}$) in verifying as he would have spent executing $\mathcal{C}$ himself ($\mathcal{T}_{\mathcal{C}}$) and thus this solution does not scale. The time to verify should ideally scale like $log(\mathcal{T}_{\mathcal{C}})$.

A formal definition of computational integrity is given by Ben-Sasson et al. [27] as:

[9] `https://grin-tech.org/`
[10] `https://www.getmonero.org/`
[11] `https://dictionary.cambridge.org/dictionary/english/integrity`

Definition 2.4.1. Computational Integrity The binary relation $\mathcal{R}$ is the set of pairs $(\mathbb{X}, \mathbf{w})$ where

- $\mathbb{X} = (TM, x, y, T)$ with TM a Turing machine, x the input and y the output of TM after T steps.
- $\mathbf{w}$ a description of nondeterministic steps made by TM on input x to form output y.

The computational integrity language $\mathcal{L}$ is the projection of the binary relation $\mathcal{R}$ onto its first coordinate or alternatively, $\mathcal{L} \triangleq \{(TM, x, y, T) | \exists \mathbf{w} \colon (\mathbb{X}, \mathbf{w}) \in \mathcal{R}\}$.

If $\mathcal{V}$ wants to enforce $\mathcal{P}$ to operate with computation integrity, he should require a an irrefutable proof that dictates $(\mathbb{X}, \mathbf{w}) \in \mathcal{R}$. The prover therefore has to keep track of and provide a proof of possession for this transcription. This, however, means that the execution time of the prover $(\mathcal{T}_{\mathcal{P}})$ will be bigger than $\mathcal{T}_{\mathcal{C}}$. Existing solutions aim to find quasilinear scalability for $(\mathcal{T}_{\mathcal{P}})$.

Properties For a computational integrity solution to be useful, it should at least possess the completeness and soundness properties as explained in Section 2.3.1. Ideally, the solution should have the universality property as well, meaning that any computation $\mathcal{C}$ can efficiently be verified. All solutions offer soundness and completeness, but not all offer universality. Different solutions are mostly measured by their prover and verifier efficiency and complexity.

2.4.1.2 Privacy

Privacy and one's right thereto has been around for a long time and was already a problem when the first newspapers started invading the privacy of individuals [36]. With the invention of the internet, this has taken another turn. It's not just "newspaper-worthy" data anymore, it's everything and about everyone. People are beginning to take an interest in where, how and when their data is used [37].
GDPR is a recent regulation in Europe's law in an attempt to protect the privacy of European citizens on online platforms. The Data Protection Commission (DPC)[12] is the Irish supervisory authority for GDPR and as many social media giants have their home in Ireland, the DPC has already opened up inquiries into the data-processing activities of companies such as Facebook, Twitter and LinkedIn[13].

[12]`https://www.dataprotection.ie/`

[13]See the 2018 annual report of the DPC

The recent abundance of data has found the attention of a lot of researchers in order to find methods to publish data in a privacy-preserving way. A popular example is κ-anonymity, which makes sure that each record in a published dataset is indistinguishable from $\kappa - 1$ other records [38]. Another is using differential privacy [39]. This method requires probability distributions derived from the dataset to be "essentially the same", independent of the inclusion or exclusion of any individual in the dataset.

Privacy is also an important manner in the case of computational integrity. Consider the case in which an intelligence service has to let the public know that they found a match in their database for a certain criminal case. It is clear that they cannot release their full database on which the public can than run the same algorithm. The concept of privacy in computational integrity problems strongly relates to Zero-Knowledge Proofs, discussed in the previous section.

2.4.2 Approach

2.4.2.1 Outline

Proof systems have a very general outline. They consist of a one-time parameter generation function *Gen*, that takes as input a security parameter 1^λ and $\mathcal{C}$. It outputs a proving key *pk* and verification key *vk*. These two keys are then used as input in a proving and a verifying function. The proving function constructs a proof π that confirms his knowledge of $\mathbf{w}$, while the verifying function takes this proof as input and either accepts or denies $\mathcal{P}$'s claim.
Using the notations from Definition 2.4.1:

- $Gen(1^\lambda, \mathcal{C}) \rightarrow (pk, vk)$
- $Prove(pk, \mathbb{X}, \mathbf{w}) \rightarrow \pi$: $(\mathbb{X}, \mathbf{w}) \in \mathcal{R}$
- $Verify(vk, \mathbb{X}, \pi) \rightarrow accept/deny$

The author of the zk-SNARK paper already mentioned a possible application of his work to be delegation of computation [40]. Indeed, the most prominent recent approaches to achieve computational integrity when outsourcing the execution of a program, are based on zk-SNARKs. There are multiple approaches to verifiable computation and it would be too extensive a survey to list all of them in this thesis. The bedrock of the approach used in this thesis is zk-SNARKs, which will be explained in this section.

2.4.2.2 Details

The earlier defined computation $\mathcal{C}$ is just a conventional computer program written in some programming language. In order to provide verification for the execution of $\mathcal{C}$, a universal representation of the execution path is needed. This universal representation is an arithmetic circuit, which is a circuit consisting of inputs, outputs and mathematical operations (usually only $+$ and $*$). An example is given in Figure 2.4. The first step towards computational integrity is thus translating a high-level programming language such as C to these circuits.

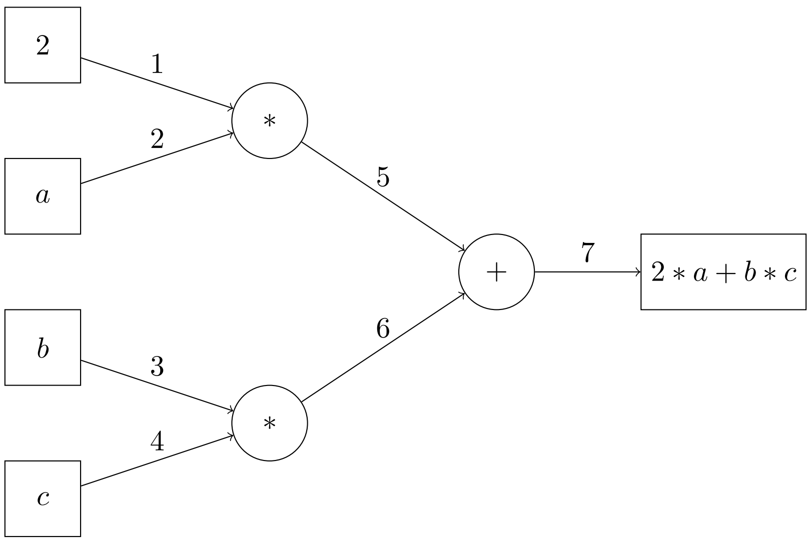


Figure 2.4: An example arithmetic circuit computing $2 * a + b * c$

The next step is to convert this circuit to a Rank-1 Constraint System (R1CS), which checks if the values traveling along the edges are correct. For instance, that the value of edge 5 in Figure 2.4 equals $2 * a$. An R1CS is set up for each gate in the circuit, which is done by finding a vector s of valid intermediate values so that $(s.A) * (s.B) - (s.C) = 0$. The vector s maps to a vector with length equal to the number of variables in the circuit (7 in Figure 2.4). A, B and C are vectors selecting the proper variables from s by using the dot product. It is the prover's task to provide a valid assignment for the vectors s, A, B and C.

For a simple circuit, providing these constraints is simple. However, for larger circuits this quickly becomes too cumbersome. To overcome this, the set of constraints is transformed to a Quadratic Arithmetic Program (QAP). Whereas R1CS uses vectors and their dot products, QAP uses Lagrange interpolation to form polynomials which, if evaluated at a point gives the corresponding sets of vectors from the R1CS representation. A polynomial is built for each intermediate value in the circuit and combined into one big polynomial using a linear combination, resulting in three polynomials: $A(x)$, $B(x)$ and $C(x)$. These three polynomials can then be combined, similar to the R1CS form as $A(x)*B(x)-C(x) = H(x)*Z(x)$. Checking if a valid assignment is provided is then equal to checking if this polynomial evaluates to zero at each point corresponding to an intermediate value. If at least one point is found that does not evaluate to zero, then no satisfying assignment was given. Checking if all points evaluate to zero is done by diving $A(x) * B(x) - C(x)$ by $Z(x)$, where the remainder should be zero and $Z(x)$ is a polynomial having zeros at all the necessary points.

For circuits that contain thousands of gates (intermediate variables), the polynomials equally have thousands of terms. The Pinocchio protocol is an algorithm to check if the prover has the correct linear combination of the polynomials without sending over that linear combination [41]. It relies on repeatedly asking for evaluations at a certain point and checking if the equation defined earlier holds. It is very similar (at least conceptually) to the principle in the example given in Section 2.3.1. This verification is done using elliptic curve pairings and homomorphic encryption of the evaluation points. Explaining these would go beyond the scope of this thesis.

This section is heavily based on the work done by Vitalik Buterin (Ethereum) and Ariel Gabizon (Zcash) and I strongly recommend reading their series of blog posts for a more in depth explanation of verifiable computation using zk-SNARKs in Zcash [42, 43]

2.5 Verifiable SQL

This thesis introduces a method for verifiable execution of SPARQL queries. As there is currently no prior work on this, an overview of research concerning verifiable SQL query execution will be given in this section.

The two specific solutions covered below make use of Authenticated Data Structures (ADSs) [44]. These structure's purpose is very similar to that of verifiable computation. It allows for an interested party to fetch query results over some data structure from an untrusted worker. The data source in this scenario is trusted and provides the untrusted worker with additional authentication information in order to provide the necessary proof to the interested party. There are several implementations of ADSs, mostly inspired by *tree structures*, *hash functions* or *signature schemes.*

Solutions using ADSs suffer from a lack of generality and expressiveness, which will become clear in the this section.

IntegriDB IntegriDB is a first implementation for verifiable outsourcing of SQL queries [45]. It supports multidimensional range queries and the query operators $JOIN$, SUM, MAX, MIN, $COUNT$, AVG AND $LIKE$. Their solution relies on building a new ADS scheme called an authenticated interval tree for each table and each pair of columns. Verification is done using principles similar to Merkle trees [21].

vSQL vSQL is a later work, not relying on ADSs but instead on earlier mentioned interactive proof systems [46]. More specific, it relies on the CMT protocol introduced by Cormode et al. [47], a very foundational work on which a lot of previously discussed research is based. As the CMT protocol boils down to the arithmetic circuits from the previous section, vSQL also generates these circuits for their queries along with a novel scheme for verifiable polynomial delegation.

Whereas IntegriDB has a rather limited set of supported queries (12 out of 22 queries from the TPC-H[14] benchmark), vSQL enjoys a wider range of possible queries (all TPC-H queries). The prover and verifier for both IntegriDB and vSQL operate in the same orders of magnitude. The main difference between the two implementations is the difference in setup time: vSQL is about 2 orders of magnitude faster than IntegriDB.

The vSQL authors also built a zk-SNARK circuit which incorporates a query and the database itself and thus requires separate design for each query and database. This setup cost is high and has significantly worse server performance compared to IntegriDB and vSQL. However, the verification is two orders of magnitude faster than the former two.

Others Other approaches include signature-based schemes such as the one introduced by Zheng et al. [48]. Another approach worth nothing is CorrectDB and relies on trusted hardware on the query executor side instead of authentication schemes or proof systems [49].

2.6 Pepper project

The Pepper project is an initiative of NYU and UT Austin in an attempt to make computational integrity for outsourced executions more practical [50]. Their works consists of intensive research

[14]`http://www.tpc.org/tpch/`

of the field and producing more practical approaches, including open-source implementations. A brief summary of their work will be listed here, for a full overview including the publications and presentations, see their website[15].

Their work started with *Pepper*, back in 2012, by improving existing work on PCPs by 20 orders of magnitude. Later that year, *Ginger* improved *Pepper* by adding several features such as conditional control flow and inequality comparisons. *Zaatar* is the first system to use the aforementioned QAPs in an attempt to reduce the substantial overhead at the prover's side. It introduces a prover's complexity that is roughly linear in the running time of the computation. The next big research paper introduced by the Pepper project is *Pantry*. It allows for verifiable computations with state: programs that interact with untrusted storage such as disk space or Random Access Memory (RAM). In 2015, they released another important improvement called *Buffet*, which allows for data-dependent control flow.

Other systems introduced by the Pepper project, but less relevant for this thesis are:

- *Allspice*: A hybrid architecture that improves non-cryptographic approaches such as the CMT protocol and uses static program analysis to determine the best solution for verifiable computation.
- *Zebra*: A system to build verifiable Application-Specific Integrated Circuits (ASICs) so that computations can be outsourced to chips produced by untrusted manufacturers. It is based on the interactive CMT proof protocol.
- *Giraffe*: An optimization of *Zebra*.

This thesis makes heavy use of the Pepper project and more specifically of their end-to-end toolchain *Pequin*. It composes of a series of tools to transform a program written in a high-level programming language such as C to two executables: the prover and the verifier. The system consists of a front-end which translates an extensive subset of C to arithmetic circuits. These arithmetic circuits are the input to *Pequin*'s back-end, which relies on zk-SNARKs. The toolchain brings all the research developed by the Pepper project together in an easier to use framework.

[15]`https://www.pepper-project.org`

Chapter 3

Problem statement

During the introduction, some challenges along the evolution of the Semantic Web were given. One of them is the computational complexity of solving a query. Outsourcing the execution of data aggregation tasks to computationally powerful third parties is an option, but comes with trust issues. Solving these trust issues will be the focus of this thesis. The exact specifications and technologies involved will be discussed in this section.

Linked Data is the backbone of the Semantic Web [7] and with its corresponding data model RDF, the web can be made more accessible for non-human analysis [11]. Together with a query language such as SPARQL to query this machine-friendly data format, the possibilities for knowledge extraction are endless [8]. New initiatives to decentralize data and give back control to the user, such as Solid, have promising operational models in a world where privacy is getting more and more important. New technologies always come with new challenges and a shift of data possession is no different.
Firstly, querying data that is decentralized and spread over multiple pods is a significantly computationally harder task and requires more resources than querying existing centralized databases. These have been around for a long time and optimized to offer millisecond response time with minimal work.
A second challenge is that in a decentralized model such as Solid, it can be a common to have multiple pods for just one user (e.g. a separation of someone's tax statements and grocery bills). If these pods are not stored by the same provider, even a single-user query can take several seconds if network conditions are not favorable. If quick response times are of significant importance, the Semantic Web is thus currently not an ideal solution.

In Section 2.1.4, the low SPARQL endpoint availability was discussed and how the concept of TPFs was introduced to mitigate this issue. The dynamic iterator pipeline algorithm is a major component in the whole TPF setup. However, many clients do not have the hardware, network connectivity or time to run this algorithm. Outsourcing the execution of this client-side piece of software would be ideal for these scenarios and would solve the first challenge from the previous paragraph.
The second challenge is conveniently solved by outsourcing as well. Queries that require quick response time can be calculated in advance and stored until necessary. These computations can be done when the query load is low and inactive resources can be assigned to perform these tasks ahead of time. Outsourcing comes with many possibilities and advantages, which will be discussed in Section 7.2. Two use cases will be covered in Chapter 5

With outsourcing, however, also comes distrust. People do not trust social media giants with their data and privacy is getting progressively more media attention. So how can trust be introduced into what are essentially two strangers communicating about possibly sensitive data? This is a very broad question and this thesis focuses on a specific context for this question. Two research questions are defined and define the scope for this thesis:

> $\mathbb{Q}_1$: What are the advantages of verifying outsourced BGP query executions instead of executing them locally using TPF.
>
> $\mathbb{Q}_2$: How can non-interactive proof systems be used in verifying the computational integrity of the dynamic iterator pipeline algorithm?

First of all, it is important to find an incentive to verify outsourced BGP queries. If it offers no reasonable benefit over executing it locally, there is no use in doing it.
Secondly, whilst recalculating a grocery shopping list might be a common thing to do, re-executing is less obvious for computationally harder scenarios where resources and time are scarce. The second research question thus focuses on how non-interactive proof systems can be used to verify without re-executing.

Verification in the context of this thesis means assuring computational integrity of the program executed by the third party. Computational integrity boils down to soundness and completeness of the results and both were defined more formally in Section 2.3.1. The third party might have the incentive to obscure some of the results (completeness) or include falsified information (soundness).

In light of the first research question, some hypotheses are defined:

$\mathbb{H}_{1.1}$: Fetching and verifying the result from outsourced BGP queries is at least 25% faster than local execution using TPF.

$\mathbb{H}_{1.2}$: Fetching and verifying the result from outsourced BGP queries uses at least 25% less memory than local execution using TPF.

$\mathbb{H}_{1.3}$: Fetching and verifying the result from outsourced BGP queries uses at least 25% less bandwidth than local execution using TPF.

The second research questions introduces two additional hypotheses:

$\mathbb{H}_{2.1}$: Using non-interactive proof systems, the time required to verify the completeness and soundness of the results of a BGP query, is independent of result set size.

$\mathbb{H}_{2.2}$: Providing a computational integrity proof for execution of BGP queries requires time and memory that is 2 orders of magnitude larger compared to query execution using TPF.

Chapter 4

Implementation

The previous chapter introduced the general problem statement, the research questions and the hypotheses. This chapter focuses on finding a solution to the problem statement, providing us with an answer to the research questions.
Re-executing is a very naive way of verifying a distrusted party's work and only works in very specific scenarios. Recently, there has been a trend towards research in verifiable computation, which aims to verify every single machine instruction done by an unobserved party. In this thesis, a step towards more practical verifiable computation will be made by implementing the dynamic iterator pipeline algorithm using verifiable arithmetic circuits.
Section 4.1 covers the framework that will be used to do this, while Section 4.2 discusses some of the considerations that have to be made by using this framework. Section 4.3 goes on to list the actual details of the implementation of which the practical limitations are mentioned in Section 4.4.
The code for the implementation is available at `https://github.ugent.be/sbmorel/thesis`.

4.1 Framework

As mentioned during the introduction, the Pepper project is an attempt to make computational integrity proofs more practical [50]. This thesis will make use of their *Pequin* end-to-end toolchain, of which no official releases have been made. Its source is publicly available, but not actively maintained. It consists of a front-end that translates a subset of the C language to arithmetic circuits and a back-end relying on *libsnark*[1], a C++ library for zk-SNARKs, to

[1]`https://github.com/scipr-lab/libsnark`

provide a PCP for the circuit generated by the front-end.
The main workflow to generate a verifiable program and a proof of correct execution for some input is:

1. Writing a program $\mathcal{C}$ using an extensive subset of the C language

2. Compiling the verifier and prover using `pepper_compile_and_setup_{V,P}.sh`. This calls the front-end and translates the C source code to an arithmetic circuit and its constraints. It also includes the setup phase of the non-interactive proof system used by *Pequin*'s backend. As mentioned in Section 2.3.2, this is an important part and should be done with care not to expose the secret parameters, at least for production environments. For example, Zcash designed a whole ceremony to create these parameters and afterwards, destroyed the toxic waste (private keys)[2]. This phase produces a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ along with their prover key (pk) and verifier keys (vk).

3. Generating input, I, that is according to a default format provided by *Pequin* and accepted by $\mathcal{C}$.

4. Running $\mathcal{P}$ using `pepper_prover_{program_name} prove`, which takes as input the prover key pk and the input I and outputs O. It executes $\mathcal{C}$, providing the output along with a proof π that shows knowledge of a valid assignment to the arithmetic circuit.

Any party can then verify $\mathcal{P}$'s work by running the verifier executable $\mathcal{V}$, taking as input the verifier key vk, the input I, the proof generated by the prover π and the output O. The verifier will run until all constraints are verified or quit as soon as an inconsistency is found.

Pequin allows for private prover input, which is hidden from the verifier. This is done by either commitment schemes introduced by the *Pantry* system of the Pepper project or by exogenous computations introduced in *Buffet* [51, 52]. For now, it suffices to know that *Pequin* allows to call a function `exo_compute()` which receives input from the prover and gets back arbitrary, nondeterministic data. Details on how this is used will be given later in this chapter.

4.2 Considerations

During development of the program used in the *Pequin* toolchain, some considerations have to be made in what parts of the C programming language can be used. Each line of code has to be

[2]`https://electriccoin.co/blog/the-design-of-the-ceremony/`

translated to a constraint in one way or another and thus only the logic that is present in the *Pequin* front-end compiler can be used.

First of all, libraries often used without much questioning are not available. However, some header files have been re-implemented from scratch by the people behind the Pepper project. This includes *stdint.h* with all of its constants, but also some header files offering basic maps, lists and tree functionality. The most restricting part for this thesis is the absence of strings. String manipulation, common practice in any programming language, is not an option and even defining a static string is not possible. This does not mean that it is entirely impossible, but rather that a library such as *string.h* needs to be re-implemented for this particular subset of C. An array of `char` types (defined in *stdint.h*) is thus nothing more than an array of 8-bit integers.

The majority of the C programming language syntax is supported, including: functions, structs, if-else statements, explicit type conversion , integer and bit-wise operations and preprocessor definitions. One major constraint in the syntax is the absence of data-dependent loops. Arithmetic circuits are always of fixed size and data-dependent loops would require a circuit of variable size, depending on the input, which is not possible using zk-SNARKs at the time of writing. Statically-bound loops (i.e. bound at compile-time) can be implemented by flattening the loop into sequential execution of the same loop body several times. This also means that the `break` and `continue` statements are not available, as these are data-dependent. The *Buffet* is a C-to-C compiler of the Pepper project that does allow data-dependent loops and the `break` and `continue` statements. This is only possible for cases where a static maximum number of iterations is known, resulting in a rewritten loop using a simple state machine. An example of this rewriting using a data-dependent loop and a `break` statement is given in Listing 4.1. The rewritten loop in Listing 4.1 can then easily be flattened and converted to constraints by simply replicating the loop body `MAX_ITER` times.

A last consideration that has to be made is that input/output is not supported. This includes all standard streams (*stdin*, *stdout*, *stderr*), but also any network communication. Verifying the computational integrity of network communications would mean writing an arithmetic circuit for the OSI model, which is not feasible.
Nonetheless, as mentioned in Section 4.1, there are possibilities to provide the prover with external input using exogenous computations. How exactly these are used, is explained in the next section.

```
limit = get_limit(input)
for i in [0, limit]:
  # <BODY 1>
  if condition(i):
    # <BODY 2>
    break
  # <BODY 3>
```

```
i = j = state = 0
limit = get_limit(input)
while j < MAX_ITER:
  if state == 0:
    if i < limit:
      # <BODY 1>
      if condition(i):
        # <BODY 2>
        state = 1
      else:
        # <BODY 3>
        i += 1
    else:
      state = 1
  j += 1
```

Listing 4.1: Loop rewriting example showing the original while loop (left) and rewritten using a state machine (right).

4.3 Details

The complete solution architecture consists of four parties:

- **R**: A party requesting the solution to a BGP query.
- **DS**: The data source (e.g. DBpedia). This source is trusted by **R**.
- **AUX**: A group of auxiliary executables written in Python. These are potentially malicious and are not trusted by **R**.
- **P**: The prover $\mathcal{P}$ and the entirely verifiable program $\mathcal{C}$ written and compiled using the *Pequin* toolchain. This party is not trusted by **R** and is subject to the considerations mentioned in the previous section.

Their purpose and working will be explained in this section. A sequence diagram illustrating the proving protocol is given in Appendix A and can serve as a guideline for this chapter. Appendix B is an overview of the syntax used for the input and output both to the program and to exogenous computations.

4.3.1 Requesting party

The requesting party can be anyone desiring the solution to a BGP query. He provides the system with a BGP query and a random nonce, of which the purpose will become clear later on in this chapter.
Note that in order to query from one or more dynamic data source(s), the requesting party should also provide these as input. However, in this thesis, the data source in this thesis will always be DBpedia.

This party receives the solution to the query from **P** and verifies the computational integrity of the whole system. It does this by running the verifier executable $\mathcal{V}$, produced during the setup phase. $\mathcal{V}$ checks if the proof $\mathbf{w}$ shows knowledge of a valid assignment of the arithmetic circuit, given the input and output. However, this is not the only verification that needs te be done. **R** also needs to check the digital signature provided by **DS**, of which the details are explained in the following sections.

4.3.2 Data source

The data source in this solution is a TPF server and thus offers TPFs upon receiving a simple triple pattern query (e.g. the data portal at *linkeddatafragments.org*[3]). While in theory the data source can be anything, in this thesis the data source is fixed and in particular will be DBpedia version $2016 - 04$.

This data source, however, needs to be extended in functionality which will be explained in the next paragraphs.

Response digests First of all, the data source should be able to provide a cryptographic hash (digest) of each result page it provides. This thesis focuses on the cryptographic hash function $\mathcal{H}_{\texttt{SHA1}}$, but this is easily converted to more secure relatives.

Nonce Secondly, the data source should be able to accept a nonce with each request, which is chosen at random by $\mathbf{R}$. This enables for subsequent requests with the same nonce to be linked together. Response digests should be stored, along with their corresponding nonce, for a predetermined amount of time. This requirement should allow the data source to perform a simple query along the lines of (in SQL):

```
SELECT hash FROM hashes WHERE nonce=$nonce
```

The result set from this query will be referred to as $\mathbf{H}$ and its elements as $\mathbf{H}_k$.

This requirement is relatively low cost for the server and can be implemented easily. This data is temporary and can be deleted once the query is finished. Note that the TPF server now receives notion of BGP queries. While before it merely answered to triple pattern queries, it can now link together triple pattern queries, infer the requested query and tell when a party started/stopped resolving a query.

Digital signature scheme A last requirement is that the data source should possess over a public and private key-pair for use in a digital signature scheme such as the Ellipitic Curve Digital Signature Algorithm (ECDSA). Upon query completion, the data source will receive a new hash $\mathcal{H}_{\texttt{SHA1}}(\mathbf{H}_1|\mathbf{H}_2|\ldots|\mathbf{H}_k)$, the digest of the concatenation of the elements in $\mathbf{H}$. This is again easily converted to another, possibly stronger, one-way function.

The data source can easily check, using the previous requirements, if this is the correct hash for the given nonce. If it is, a digitally signed version is sent back. Of course, the key-pair used here

[3]`http://data.linkeddatafragments.org/`

undergoes the same security considerations for many other asymmetric cryptographic protocols. Decentralized Public Key Infrastructure (DPKI) is an interesting approach for this in light of the general decentralization idea behind this thesis. Nevertheless, any Certificate Authority (CA) will do as well.

As a prototype for this thesis, a dummy data source was implemented that directly integrates with the auxiliaries explained in the next section. Its data is conform with that from DBpedia and which data was used will be given during the evaluation in Chapter 6.

4.3.3 Auxiliaries

The third group are the Python scripts. These are the source of private prover input mentioned earlier and were written for Python 2.7, specifically for this thesis. They serve as a bridge for any network communications that have to be made from $\mathcal{C}$. There are three Python scripts involved: a counter, a fetcher and a signer. All three accept input from $\mathcal{C}$ in a specific format imposed by the *Pequin* toolchain (see Appendix B) and provide output back into $\mathcal{C}$ in response.

Counter The counter is a script that has as input a triple pattern and thus consists of a subject, predicate and object. In case of a dynamic data source, an extra input should be added for this as well. The script returns the number of triples matching the triple pattern in **DS**.

Fetcher The fetcher script takes the same input as the counter, along with a page number and the nonce given by **R**. It outputs data in a format that is readable by $\mathcal{C}$. The result consists of the actual triples for the given page (if valid) and the digest $\mathbf{H}_k$ given by **DS**.

Signer The signer is the last script that is consulted by $\mathcal{C}$ and accepts as input the hash $\mathcal{H}_{\texttt{SHA1}}(\mathbf{H}_1|\mathbf{H}_2|\dots|\mathbf{H}_k)$. It verifies this hash with **DS** and, if all is correct, sends back the digitally signed message to $\mathcal{C}$.

As already mentioned, these auxiliary executables are not trusted by **R** and are thus a potential source for any malpractice. The digital signature scheme and cryptographic hashes introduced at the data source is what allows for the system to still provide verifiable computation of a BGP query. Details will be given in the next section explaining the prover's work.

4.3.4 Prover

Finally, the prover $\mathcal{P}$ and its computationally verifiable program $\mathcal{C}$. This program is written in a subset of C and compiled using the *Pequin* toolchain. It implements the dynamic iterator pipeline algorithm described in Section 2.1.4.3, receives its input from **R** and relies on **AUX** to do the required communication with **DS**.
The program $\mathcal{C}$ consists of different parts, each serving their own purpose and the most important ones will be explained in this section.

Secure Hash Algorithm 1 (SHA-1) The SHA-1 algorithm has been implemented from the ground up based on the original specifications [53]. As no data-dependent loops are possible, a maximum message size has to be set. Furthermore, as there is only the notion of separate integers in this subset of C, the message length has to be known up front and fed to the function. A length-independent version was implemented as well, but relies on the presence of a null-byte to indicate the end of the message and is not used in this thesis.
This function, for example, calculates $\mathcal{H}_{\texttt{SHA1}}(\mathbf{H}_1|\mathbf{H}_2|\dots|\mathbf{H}_k)$ which later gets send to $\mathbf{AUX}_{Signer}$.

SPARQL parsing The input to $\mathcal{C}$ consists, in part, of the BGP query. This query has to be read and the triple patterns have to be extracted. The main difficulty with this is the absence of strings and data-dependent loops.
The BGP query is given to $\mathcal{P}$ as an array of integers, representing the ordinals of the characters in the ASCII table. The query is subject to some basic rules:

- The query should be minified and thus should not contain any unnecessary characters
- The query should not make use of shared subjects/predicates/objects, all triple patterns should be written in full.
- There should only be 1 BGP present in the query

A state machine is set up to find the different parts of the query and saves them to intermediary variables. It does this by reading the input array sequentially, changing the state accordingly.

Finding a minimum triple pattern The main idea behind the dynamic iterator pipeline algorithm is to always start with the triple pattern having the least results. A dedicated function, set up to call $\mathbf{AUX}_{Counter}$ for each triple pattern present in the query, returns the triple pattern with the least results according to **DS**.

TPF hashes As mentioned before, the **AUX** scripts act as a bridge between $\mathcal{C}$ and **DS**. It is a possible bottleneck in the trust of the whole system. Hashes and a digital signature scheme (explained in the next paragraph) are an approach to check if the auxiliaries are not malicious. The auxiliary $\mathbf{AUX}_{Fetcher}$ forwards the hashes provided by **DS** for each TPF page to $\mathcal{C}$. In order to check if $\mathbf{AUX}_{Fetcher}$ made any unwanted changes to the results, $\mathcal{C}$ can recompute the hash and throw an error if they don't match. As $\mathcal{C}$ runs in a verifiable environment, there is no way for the auxiliary to circumvent this integrity check.

Digital signature Later on, $\mathcal{C}$ calculates a digest that represents the combination of all the previously received hashes, $\mathcal{H}_{\mathtt{SHA1}}(\mathbf{H}_1|\mathbf{H}_2|\ldots|\mathbf{H}_k)$. Again, these digest calculations are done inside the *Pequin* framework and are thus tamper-proof.
This produced digest is sent to **DS** (through $\mathbf{AUX}_{Signer}$). If this corresponds with his locally-stored version, he produces a signed version and sends it back to $\mathcal{C}$. This signed digest is included in $\mathcal{C}$'s output and can be verified to be valid by any party having **DS**'s public key.
This digital signature scheme will protect against any modifications made by the auxiliaries. For example, if $\mathbf{AUX}_{Fetcher}$ modified a TPF together with its hash, it would go unnoticed at first. However, due to the digital signature scheme in the end, this change will be detected by **DS** and he will refuse to sign. Unless **DS**'s private key is exposed, $\mathbf{AUX}_{Signer}$ cannot fake this signature and is forced to sent back either an invalid one or an error status to $\mathcal{C}$. In both cases, an invalid proof is produced.

4.4 Limitations

The majority of the system was successfully implemented according to the details in the previous section. However, the considerations listed in Section 4.2 introduce some limitations to the universality of this proposed solution. This sections offers an overview of these limitations, together with possible workarounds.

Recursiveness Remember that the dynamic iterator pipeline algorithm explained in Section 2.1.4.3 is recursive in nature. As recursive functions are data-dependent, this is not possible using the *Pequin* framework. The recursive depth is equal to the number of triple patterns included in the BGP query. The algorithm has to be converted to a linear one by inlining. This has to be done separately for every different number of triple patterns. For example, if three

triple patterns were detected during query parsing, a program dedicated to solving three triple patterns should be called.
This restriction can somewhat be circumvented by implementing a function that detects the number of triple patterns present and then call the corresponding function able to solve it. However, this comes with a lot of replication of the same code, all of which will be converted to separate constraints. Manually rewriting parts of the arithmetic circuit to be more modular and re-use existing constraints is an option, but out of scope for this thesis.
Another possible workaround would be an efficient non-recursive algorithm that can aggregate data from Triple Pattern Fragments. However, no algorithm more efficient than the dynamic iterator pipeline was found. Implementing a naive one that fetches all results for every triple pattern and then combines them would require too many constraints to be practical.

Input length The statically bounded loop requirement of the framework is a deal breaker for most of the implementation details. For example, the input SPARQL query needs to be read character by character and thus this needs a loop. This means a first limitation is the maximum input length, which needs to be predefined. This is also the reason for the previous requirement of minified input.

Maximum variable length Because machinery such as `malloc` or `calloc` do not exist inside the framework, variables are all static and have a predefined length. Therefore, if the triple patterns or SPARQL variables have to be stored, this needs to be done in `structs` that have predefined lengths. This causes a limitation in the length of variables and the maximum length of a triple's subject, predicate and object fields.

URIs As the input length is bounded by a maximum value, the notion of URIs in SPARQL queries was removed. URIs quickly make a resource's identifier two to three times as long in length and are thus removed to keep the input length low. This, however, comes with the disadvantage of having to incorporate the ontology somewhere else in order to provide meaning to the triples. In the implementation, this was done by writing a dummy data source.
Note that URIs can easily be re-introduced by simply increasing the maximum variable and input length, but this introduces more constraints to be checked and would scale linearly in the number of input triple patterns. Why exactly this is a problem, is explained at the end of this section.

A less naive workaround would be to disallow URIs anywhere except for in prefixes. By writing the SPARQL query using prefixes only, $\mathcal{C}$ could be programmed to read out these prefixes and sent them along to the auxiliaries. This way, the maximum variable length can stay the same (as long as it supports a short prefix) and only the input length has to be increased. This solution scales sub-linearly with the number of triple patterns and is thus more desirable.

Maximum results Again, static loops and variables require a definition for the length of the results. This roughly translates to a `LIMIT` query, with the limit being the maximum length defined during compilation.

Maximum triple patterns A BGP query can have any number of triple patterns included in its `WHERE` clause. However, the same limitations again pose a restriction here. As the dynamic iterator pipeline algorithm requires looping over these triple patterns, a fixed amount is required. Having a fixed number of triple patterns has the benefit of immediately offering the maximum number of variables as well, which is required to define the output. Usually, a BGP query represents a graph and thus all triple patterns are connected through a variable. This is not a strict requirement, but keeping the length of variables low is beneficial for performance. For this reason, a connected graph in the BGP query is made a requirement. This leads to the following: If a BGP query contains x triple patterns, then the maximum number of variables allowed in the query is:

$$2 * (x - 1) + 3$$

Maximum TPF results As mentioned earlier, $\mathbf{AUX}_{Fetcher}$ provides $\mathcal{C}$ with results from a TPF server (**DS**). This has to be stored inside $\mathcal{C}$ and looped through according to the main algorithm and thus also needs a fixed length. Standard TPF pages have a length of 100 triples and unlimited length in total and both need to be defined upfront. $\mathcal{C}$ imposes a limit on this total length and thus triple patterns that go above this threshold will result in triples missing from the result.

The above limitations are mainly due to the static nature of the arithmetic circuits produced by the *Pequin* framework. While dynamically sized circuits based on the input might be a thing in the future, at the moment each line of a code is converted to its own small sub-circuit. However, a general improvement to all these limitations can be made by simply increasing the constants defined in the program. Compiling with higher predefined lengths, however, also means producing more constraints as more memory read-writes and ALU operations have to be verified. This results in more work for the compiler, but more importantly, results in a bigger proving key and more work for the prover to find a valid assignment for the circuit. Chapter 6 gives an idea of exactly how much work is required for a predefined set of lengths.

This chapter summarized the details of the implementation, together with some considerations that had to be made. The system provides a BGP query resolving method that is verifiable in terms of computational integrity. It serves soundness and completeness of the results, but lacks total universality due to some limitations dictated by the underlying mechanisms, which were listed in the last section. In the next chapter, some use cases for this particular implementation and its limitations are discussed.

Chapter 5

Use cases

In the previous chapter, the full details of the system were discussed and some limitations were mentioned. The constraints due to the nature of the back-end providing the integrity proofs, which is still in research phase, are a limiting factor in the practicality of the solution. In general, however, there are still use cases that work well within these limits. Specific timings for all the parts involved in the system, such as compiling, proving and verifying are given in Chapter 6. These should give more notion to some subjectiveness introduced in this section.

The problem statement and corresponding solution introduced in the previous chapters gives an idea on the desired properties of a use case:

- Query results should be retrievable quickly and reliably
- The data source that provides the actual data is considered a trusted party
- The query executor is an untrusted third party and potentially has malicious intent
- The data source should contain relatively static data
- Queries can be done upfront and plenty resources are available for this

The last two requirements are a result from the currently high prover complexity and thus long execution times. If data were to change faster than a prover can execute the query, the prover would quickly fall behind.

In what follows, a general blockchain implementation and two possible use cases will be elaborated. Note that tweaking the limits from Chapter 4 and future work discussed in Section 7.1

are very likely to introduce many more practical scenarios. A brief discussion of this growth potential is done in Section 7.2.

5.1 Proof systems inside blockchain

In Chapter 2, an introduction to blockchain was given. It is, without a doubt, a promising technology. However, its immutability relies on Proof of Work (PoW), a concept introduced in the original bitcoin paper [20]. It is, in essence, a useless computation that proves some work was done and which can quickly be verified, all the while using an immense amount of energy. To put the energy consumption into perspective, surveys estimated the total energy consumption to be comparable to national energy consumption of countries such as Ireland and Austria [54, 55]. This is considered a major drawback and scalability issue for blockchain applications such as Bitcoin.
To mitigate this, research has proposed several alternatives such as Proof of Useful Work [56]. Here, the energy is not wasted, but instead used to find solutions to computational problems of practical interest. Primecoin[1], for example, is based on finding large prime numbers as their PoW.
Another popular example is Proof of Stake (PoS). In PoW systems, revenue is directly proportional to the hardware setup and therefore also energy consumption. PoS systems aim to solve this by linking revenue to the stake a participant has put forward to become a validator. There is no longer a battle to have the most computational power. The large energy consumption is a major factor in the decision of Ethereum to move towards PoS with their Casper protocol[2].

This thesis introduced a system to delegate verifiable computation of SPARQL query results. In order to provide a proof of correct computation, the query executor (the prover $\mathcal{P}$) needs to invest resources to firstly, calculate the query result and secondly, to produce a valid proof. While producing the proof is resource intensive, verifying the correctness of the results is quick and low-effort. Chapter 6 lists some practical execution times and memory hardness confirming this.
This proof system effectively boils down to providing useful work. It can thus be used in a blockchain architecture as a substitute for the environmentally unfriendly PoW. The immutability of the blockchain comes from linking blocks together and requiring a solution to a

[1]`http://primecoin.io/`
[2]`https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ`

computationally hard puzzle for each block in order to be valid, of which the solution of this thesis is a perfect example of. In what follows, an explanation will be given on how this proof system can be applied to blockchain.

Figure 2.3 was a more general structure to illustrate how blocks are linked together in a chain. It contained only a hash to the previous block, the data and some metadata. Figure 5.1 is a more specific illustration and adaptation to support the novel proof system of this thesis and in what comes next, its contents will be explained in more detail.
First of all, there is a timestamp included in each block. It allows to check which block is newer than another and possibly to ignore new blocks that are too far in the future or past. This is a very general field and is also present in every existing blockchain application.
To offer the same immutability as other blockchains, however, the program $\mathcal{C}$ introduced in the previous chapter needs to accept additional input. Namely, it has to accept the hash from the previous block, which it can simply copy to the output. Summarizing, each block needs three components:

1. The prover's input
    - SPARQL query
    - Nonce
    - Previous block hash
2. The prover's output
    - Query results
    - Resulting hash
    - Digital signature
    - The previous block hash
3. The proof of correct execution (computational integrity proof)

If an adversary would want to change a block's content (e.g. the query result or the query itself), it would invalidate the proof and change the hash of the block. Knowing that blocks are linked by their hashes and that the proofs incorporate these hashes, it would also invalidate the proofs of the subsequent blocks.

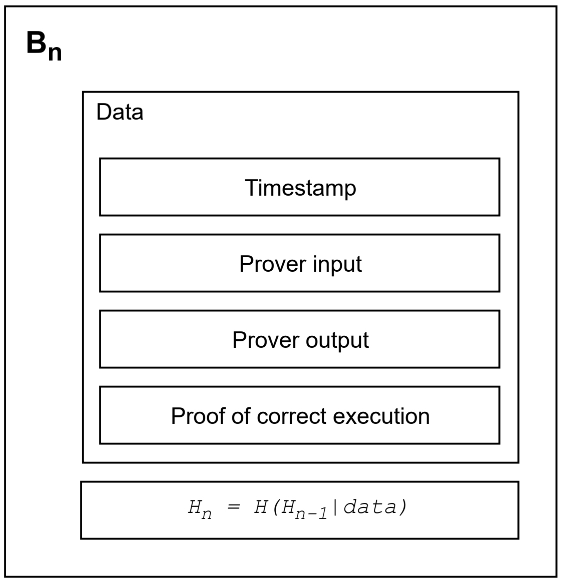


Figure 5.1: Structure of a block supporting proofs of computational integrity for use in a blockchain environment

This kind of structure for blocks allows for a decentralized, immutable and trusted data source for all kinds of Linked Data. Providing new blocks is a computationally intensive task and requires some effort, equal to current PoW schemes. Verifying a block is quick and effortless and consists of verifying the proof of correct execution. Getting the results for some query is as easy as traveling along the chain (backwards), comparing the input with the desired query and if they match, extract the results. Updating information is done by publishing a new block with the same query but an updated result set and increased timestamp.

The structure from Figure 5.1 is not necessarily the one that should be used. For example, the query result and the query itself do not necessarily have to part of the block's data in their whole. Instead, a hash of them can be included in order to keep the block size low. The actual data can then be stored elsewhere. Furthermore, in many current blockchain networks, each block contains multiple "transactions" of digital currency whereas in the structure above, there is only one query per block. This could easily be extended to multiple queries per block to provide more throughput. The next two sections apply this blockchain model to provide a possible practical scenario.

5.2 Public health care

Having listed the properties a system should possess in order to be a suitable use case in the beginning of this chapter, a first use case is in the public health care domain.
Consider a scenario where a doctor and his team diagnose a patient with a disease and they need the latest treatments and procedures from renown health organizations to help the patient. Firstly, this information needs to be gathered quickly as it can potentially be life-threatening. If this scenario were to happen in places lacking modern network throughput (e.g. rural Africa), fetching the data quickly can pose a problem. Furthermore, the doctor and his team need reliable data from a trusted source, such as the World Health Organization (WHO). If anything bad were to happen during a procedure, the cause certainly cannot be misinformation.
Existing databases offering medical information are mostly independent and do not offer a unified information platform. A central hub offering all information that is known on a subject from all institutions around the world is thus still missing. Computational integrity protocols for SPARQL queries can offer a solution here and in particular the blockchain application mentioned in the previous section.

5.2.1 Existing systems

Rural Africa is without a doubt in need of proper health care services. The region accounts for 24% of the global burden of disease, but has only 3% of the world's health workers. Figure 5.2 clearly show how sub-Saharan Africa is struggling.
Startups have a hard time to survive (e.g. the HiDoctor app[3]) due to a plethora of reasons as Africa is not always the easiest region for an entrepreneur. With government support, on the other hand, initiatives have a higher success rate. However, they all focus on very specific problems and are often restricted to certain countries.
For example, mHealth$_4$Africa[4] focuses on storing patient health records and tests results and relaying information to health workers. It's main focus is maternal and newborn healthcare in Malawi, South Africa, Kenya and Ethiopia. The Ugandan ministry of health came up with mTrac[5], a system to track nation-wide stock-outs. DMC-MALVEC[6], supported by the European Commission's Horizon 2020 Framework, aims on gathering and providing up-to-date data on

[3]`https://www.f6s.com/hidoctor`
[4]`http://www.mhealth4afrika.eu`
[5]`http://www.mtrac.ug`
[6]`https://dmc-malvec.eu/`

malaria. They developed a platform (LabDisk) for, among others, monitoring the infection status of mosquitoes and insecticide resistance. In Ghana, the Novartis Foundation[7] is a collection of several solutions, one of which focuses on connecting community health workers to doctors via consultation centers.

It is clear that an effort is being made to balance the ratio of the region's burden of disease and its health workers concentration. However, these initiatives lack universality and often focus on very specific problems and are only available in certain areas. The information is spread over multiple companies and thus a centralized source of information is still missing. Furthermore, these initiatives are often startups and therefore more susceptible to suddenly be discontinued, making adoption rates lower.

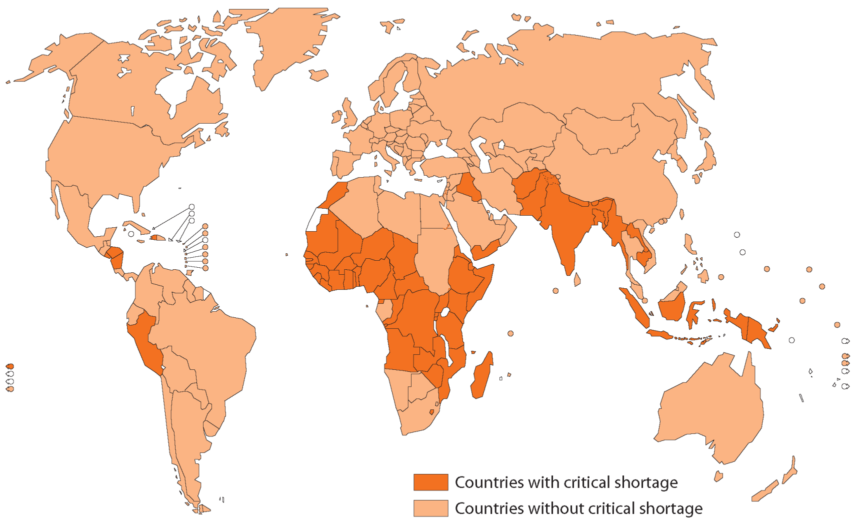

Countries with critical shortage
Countries without critical shortage

Figure 5.2: Countries with a critical shortage of health service providers (doctors, nurses and midwives). Data source: World Health Organization. Global Atlas of the Health Workforce

5.2.2 Decentralized, reliable information source

With the blockchain principle from Section 5.1, a more general and scalable solution can be set up. Instead of multiple startups and government-aided initiatives covering a variety of subjects, one decentralized, immutable and most importantly, reliable source of information is used. Rather than having specific countries use specific solutions, the same network of information can be used throughout all countries that suffer from lacking health care services. There is no more need for a separate solution per country, per problem.

[7]`https://www.novartisfoundation.org`

Health care organizations such as the ones mentioned in the previous section should focus on publishing their important information using Linked Data. They should not have to spend their time in building sophisticated platforms, but only to gather information, for which they have the necessary expertise. Building these accommodating platforms can be transferred to companies having more expertise in the matter. Ideally, there are three sorts of players involved: (i) health care information providers, (ii) the blockchain network discussed earlier and (iii) competing platform providers, offering an intuitive interface for the available data. Health workers can choose which platform to use, based on their personal preference, which introduces a natural competition for companies to develop the best possible platform.

The decentralized model proposed here easily works together with patient profiles. Platforms such as Solid can provide for these profiles and the participants of the blockchain network then have the responsibility of aggregating these with existing medical databases. By looking at the distribution of queries over time, the blockchain network and its participants can predict which queries are most needed at different times.

For this solution to make work, existing health care organizations need to migrate to Linked Data. From the previous section, mTrac is easily converted to an organization maintaining, for example, a TPF server. This TPF server then serves the triples representing the medicine stock in Uganda. Another example is the Novartis Foundation. Instead of having doctors on call, they can publish relevant medical information only once using Linked Data. This data is then always available to community health workers and can easily be updated. mHealth$_4$Africa's patient files and test results can be converted to data pods belonging to the patients. Their maternal and newborn healthcare services can be converted to raw data, explaining any complications and procedures and how community health workers should deal with them.
Some of them do not have an ideal solution and careful considerations should be made on how this data is best published, but this is out of scope for this use case.

5.2.3 Incentive

In the Bitcoin network, the incentive to produce new blocks by spending costly resources is money. When someone produces a new and valid block, called mining, he receives some Bitcoin currency in exchange for his work. In the blockchain scenario from the previous section, however, no such incentive is present. However, in the domain of public health care, many countries and organizations have dedicated resources available for this and thus incentives are of little importance. In fact, in order to encourage people to participate in the network, bonuses could be handed out by certain organizational units by means of a digital currency, which would make this system very similar to existing ones.

5.2.4 Conclusion

Section 5.2.1 clearly listed the problem with current health care services. With Linked Data, health care information can all be made public and linked together. Interlinking all the knowledge surrounding health care services offers one single technology stack to gather information rather than multiple independent ones. Furthermore, it improves further research as more automated tasks can be deployed.
As explained before, data aggregations are not always the most resource friendly, technology-wise, but this is inherently part of using Linked Data. This especially poses a problem in remote and/or rural areas where these resources are often scarce. The system introduced in this use case brings a solution to this. It provides critical medical information in a timely manner and consumes less bandwidth and memory, which might be expensive. It also introduces trust in the information that is spread using this medium, which is an undeniable necessity when it comes to health care services.

5.3 Privacy sensitive computations

A second potential application of verifiable computation delegation of SPARQL queries is privacy sensitive scenarios. An interested party might need some specific data for statistical purposes, but does not necessarily need access to the entire data source. Even stronger, he might not have sufficient permission to access the data. DBpedia is a popular example of a public dataset, but there are many private and potentially confidential ones.

Suppose the secret service needs to publish data concerning an international dispute, but does not want to expose their internal database. They could assign a trusted worker node to compute the necessary public information transcripts, without exposing any confidential information. Currently, there exist no solutions to this, at least not without exposing confidential data. If the intelligence service were to use Linked Data, the system from this thesis could be a very quick and easy solution. A worker node trusted with the whole database is assigned to solve the query, providing the public with a verifiable result.

Another scenario is related to social media giants and their recent privacy concerns among the public. Currently, they offer tools to download, what they consider "all your data". At the moment you trust them to give all the information they have on you and they are certainly not keen on giving this [4]. Additionally, they might have an incentive to prevent certain key data points from being seen. For example, for privacy reasons or for their own profit. Publishing their entire database would solve this distrust, but that would expose other's private information.
Verifiable SPARQL queries are an indisputable way of reporting all available user data to a client, without providing any information about other users on the platform.
Note that in Chapter 4, the data source was assumed to be a trusted party. In the first example of this section, trusting the data source would eliminate the problem. However, the secret service might have a very strong motive to misreport the output and thus the data source is not trusted. The same reasoning is true for the second example.
To overcome this issue, cryptographic commitments can be used. In both examples, the data source can publicly publish a commitment of their database to, for example, a blockchain. It is then up to the trusted worker node, having access to the database, to verify this commitment along with producing the necessary results and the proof.

Chapter 6

Evaluation

Chapter 4 detailed the solution to verifying BGP query results without re-executing. However, the rather new technology of verifiable computation delegation imposed some limitations on the practicality of the solution, as covered in Section 4.4. This chapter will evaluate how the implementation performs under certain conditions. The verification and fetching of outsourced BGP queries of this solution will be compared with naive re-execution locally using the dynamic iterator pipeline algorithm implemented in the Comunica engine [57]. A discussion of future work and applicability to production systems is covered in the next, and also last, chapter.

6.1 Setup

To properly evaluate between both systems, queries supported by both implementations are constructed. Comunica version 1.7.0 was tested and supports all of SPARQL, so only the limitations of this thesis' solution are to be kept in mind.

Due to the absence of recursiveness in the implementation framework, the depth of the recursiveness had to be inlined and hard coded. To keep the number of constraints low and thus also the compilation and prover times, the depth was hardcoded to support exactly two triple patterns, which is a first constraint. This has as a side-effect that any evaluations and conclusions made in this chapter are only valid for BGPs with two triple patterns. However, as soon will become clear, conclusions can easily be extended to any amount of triple patterns. There are only two edge-cases in which the results do not generalize: (i) BGPs with only one triple pattern and (ii) BGPs containing a triple pattern with zero results.

Secondly, in Section 4.4, the variables that are limited in length were given. To keep compilation and proving times within reasonable limits, these were kept rather low. This poses a constraint in which kind of queries can be used. The major factor to take into account is the maximum number of results a TP has. For example, if $\mathcal{C}$ defines this constant to be 20, the query's smallest triple pattern cannot represent more than 20 triples or an overflow would happen, causing undefined behavior.

In light of these constraints, the following first query was handpicked:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT ?p WHERE {
  ?p a dbo:Boxer.                # 6634 triples
  ?p dbp:wins 66.                # 17 triples
}
```

Listing 6.1: The SPARQL query used to evaluate the solution against the Comunica engine

It finds all the boxers who have had exactly 66 wins in their career and has 13 results. Using the comments next to the triple patterns, it is easily derived that the dynamic iterator pipeline algorithm first fetches all the triples representing `?p dbp:wins 66`. For each found triple, it then checks if it corresponds to a boxer and includes it in the result set accordingly.
The other queries are essentially the same as the one in Listing 6.1, but the number of wins differ. In total, four queries are set up, each having a different number of results. The number of wins, its corresponding TPF size and the total number of results are given in Table 6.1.

# Wins	**# TP results**	**# Total query results**
66	17	13
67	11	6
91	3	2
182	1	0

Table 6.1: The individual statistics for the queries used during evaluation

In Section 4.3, it was mentioned that the concept of URIs was removed to keep the circuit small in size. This means that the input to the system is not entirely SPARQL-compliant. The same is true for the output (see Appendix B). This means an evaluation is done between Comunica,

a system fully SPARQL-compliant, and a system that drops certain requirements in order to be feasible. However, the solution of this thesis is a prototype and can be extended in functionality to support the dropped features. Even simpler, an intermediate, trusted party could be placed between the requesting party (**R**) and the prover ($\mathcal{P}$), acting as a translator of both input and output and would have little influence on the evaluation.
To accommodate for these modifications, a dummy data source was implemented that impersonates DBpedia. This dummy data source provides the same amount of results for each query and each triple pattern, but does not return the actual results. For example, one of the results from the query above is `http://dbpedia.org/resource/Veeraphol_Sahaprom`. Including this in the result would require the program to have its inner constant set to at least 46 characters. Knowing that having 2 triple patterns gives a maximum of 5 output variables, this would quickly bloat the size of the generated circuit. It gets even worse when results from DBpedia contain non-ASCII characters, which cannot be represented by an `uint8_t`. To circumvent this, dummy values are chosen, but reflect the actual data from DBpedia. Nevertheless, this non-compliance is easily reintroduced but comes at a cost for the prover.
Transforming the query to be supported by the SPARQL parser in the program $\mathcal{C}$ results in:

```
SELECT ?p WHERE {?p a Boxer.?p wins 66.}
```

Listing 6.2: The minified SPARQL query, without URIs

This query becomes 41 ASCII characters in length when the number of wins is set to 182 and with a safety margin of 1 character, gives the maximum input length constraint. `Boxer` is the longest string present in the query and is thus set to be te maximum length possible for subjects, predicates and objects. This has implications on the dummy data source as well. Namely, any output given cannot exceed 5 characters.
Based on these observations, the limits for the constants in the program were chosen, see Table 6.2. Section 6.3 will give an indication in how these parameters influence scaling of the solution.

All aspects of this evaluation were run on the High Performance Computing (HPC) infrastructure of Ghent University. In particular, the nodes used had a *2 x 12-core Intel E5-2680v3 (Haswell-EP @ 2.5 GHz)* processor architecture and 512 *GB* in physical memory.
Compiling with the above constants took approximately 1 hour and 39 minutes and resulted in an arithmetic circuit with just short of 21.5 million constraints to be met by the prover. Constructing this circuit required at least 28.70 *GB* of physical memory to be available. The

Variable	**Value**
Input length	42 ASCII characters
Number of results	15
Subject/object/predicate length	5 ASCII characters
Variable length	1 ASCII character
TPF page size	10 triples
Total TPF results	20 triples
Triple patterns in the input	2

Table 6.2: Constant set 1, the maximum values set during evaluation

compiler failed mid-progress when not enough physical memory was present. The generated prover key pk is 6.38 GB in size while the verifying key vk is only 50.82 kB. Note that the prover key is never needed by the, potentially resource constrained, requesting party **R**.

6.2 Performance comparison

In order to evaluate the hypotheses introduced in Chapter 3, the timing, memory usage and bandwidth consumption need to be measured. This is done using Linux' `time(1)` command[1]. To measure the total up-link and down-link traffic, the `nethogs(8)` tool[2] is consulted.
For the Comunica engine, this is straightforward as a Command-line interface (CLI) version is readily available[3]. For the solution of this thesis, verification time and memory usage are easily derived as well. However, no network infrastructure was set up for this thesis and thus a simulation of the network traffic is needed.

6.2.1 Network simulation

The solution of this thesis needs 5 components in order to successfully fetch and verify BGP query results:

1. The actual output produced by the prover $\mathcal{P}$ (including the query results).

2. A proof of computational integrity produced by the prover $\mathcal{P}$.

[1]`https://linux.die.net/man/1/time`
[2]`https://linux.die.net/man/8/nethogs`
[3]`https://github.com/comunica/comunica/tree/master/packages/actor-init-sparql`

3. The public key of the data source **DS**.
4. The verification key produced during compilation.
5. The verifier executable

The last three components can be cached or hard-coded in the device and will not be included in the comparison. Nevertheless, the public key size is 154 bytes, while the verifier executable 11.07 MB. As mentioned earlier, the verification key is 50.82 kB.
In order to provide fair comparison with the Comunica engine, this data has to be simulated as being sent over HTTP. To do this, a Python `SimpleHTTPServer` was set up on commodity hardware which serves the output and proof files for the evaluation queries and of course, is not part of the HPC network. Averaging over 100 iterations gives an approximation of the timing, memory and bandwidth required to fetching these files in a real-life scenario.
The proof size is constant for each query and is equal to 134 bytes. The prover output lengths per query is given in Table 6.3, but can vary according to the value of the digital signature.

# Wins	Output size (B)
66	1,009
67	986
91	980
182	980

Table 6.3: The output size associated with each query

Bounds on output size Using Table 6.2, the formula from Section 4.4 and Listing B.1, an upper and lower bound of the output size produced by the prover can be calculated. Assuming ASCII representation of the output file, this yields 1.882 kB and 0.857 kB, respectively. The upper bound is only reached if all integers reach their maximum representational value. For example, a 32-bit integer bigger than $999,999,999$ is represented by 10 ASCII characters, but anything below 10 is only represented by one. This maximum size, however, is very unlikely to ever be reached. This is a result of the static length of the variables, resulting in a lot of values being equal to zero.

6.2.2 Time

Using the network simulation, a time comparison can be made between Comunica and the solution of this thesis. A party needing to fetch and verify BGP results needs to do four things: (i) fetch the result, (ii) fetch the proof, (iii) verify the computational integrity using the verifier executable and (iv) verify the digital signature. Fetching the results and proof is done using cURL[4], while verifying the digital signature is done using a small Python script. An overview of the timings is given in Figure 6.1 and are a result of averaging over 100 iterations.

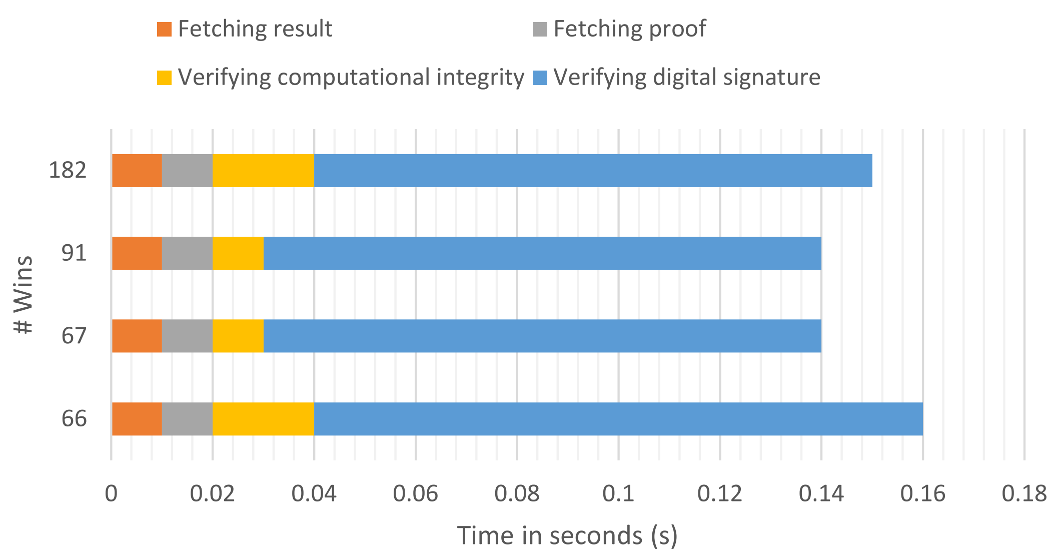


Figure 6.1: Overview of timings for each part of fetching and verifying BGP results

As mentioned before, both protocols under evaluation here are being run on the HPC, an extremely well performing computing infrastructure. For example, the small timings for fetching in Figure 6.1 are explained by the down-link speed being close to 100MBps. This performance is not very common, but as long as both protocols are evaluated on the same infrastructure, this does not matter.

Figure 6.1 also indicates that verifying the digital signature requires the dominant factor in verification. However, it is important to note that verification of the digital signature is done using a Python script, whereas the verifier executable was compiled from pure C. Part of the larger execution time is thus due to the Python environment.

Table 6.4 compares the total timing between Comunica and the solution of this thesis. It is

[4]`https://curl.haxx.se/`

clearly visible that verifying and fetching the results from outsourcing is quicker than local execution using TPF. With these results, hypothesis $\mathbb{H}_{1.1}$ can be accepted with a constraint that is a lot stronger than 25%. Note that this evaluation is done using BGPs with two triple patterns and thus does not necessarily generalize for all BGP queries. However, it is intuitively clear that increasing the number of triple patterns will only increase the speedup. The two exceptions mentioned in the beginning of this chapter are not considered.

Furthermore, hypothesis $\mathbb{H}_{2.1}$ can be accepted as well. Figure 6.1 and Table 6.1 indicate that there is no correlation between the verification time and the result set size. Although slight fluctuation is present, it is not in direct correlation with the result size, but rather with system call delays and network fluctuations. This is an important result because for Comunica, the total time required to solve a query is in positive correlation with the amount of results. It, again, indicates that the speedup will only increase for more complex queries.
Although at first something unexpected, some reasoning quickly explains why verification time is independent of result size. First of all, the proof size is only correlated with the circuit size and therefore the same for all queries (and result sizes). Verifying the proof thus requires the same amount of checks for each query. Secondly, each digital signature is exactly 384 bits in length and therefore verifying the digital signature is also independent of the actual inputs and outputs of the system.

# Wins	**Comunica**	**Thesis**	**Speedup**
66	1.09 s	0.16 s	6.81X
67	1.03 s	0.14 s	7.36X
91	0.98 s	0.14 s	7.00X
182	0.96 s	0.15 s	6.40X

Table 6.4: Timing comparison of Comunica vs. fetching and verifying for each query

6.2.3 Memory

To confirm or reject hypothesis $\mathbb{H}_{1.2}$, a comparison in memory is needed. As said in the beginning of this section, `time(1)` is used for this. In particular, its reported "Maximum resident size". The total memory used for verifying and fetching the results from a query solved by $\mathcal{P}$ versus the memory used by Comunica is given in Table 6.5. As mentioned in the previous section, part of the verification exists of a python script that verifies the digital signature. The maximum resident size encountered during evaluation was due to this part of the verification. Additional gain is thus possible by eliminating the Python environment overhead. This confirms hypothesis $\mathbb{H}_{1.2}$ and shows how, as an added benefit, memory usage is independent of result size.

# Wins	Comunica	Thesis	Gain
66	65.80 MB	11.51 MB	83%
67	62.65 MB	11.51 MB	82%
91	61.78 MB	11.51 MB	81%
182	61.64 MB	11.51 MB	81%

Table 6.5: Memory comparison of Comunica vs. fetching and verifying for each query

6.2.4 Bandwidth

Due to the nature of the TPF algorithm, it is immediately clear that Comunica will make many more network requests than the solution to this thesis. While the prover ($\mathcal{P}$) does need to make all the same network requests, this is of no importance to the requesting party (**R**). He only has to make a request for the results and for the proof of computational integrity, optionally even bundled in only one HTTP request.

Table 6.6 lists the incoming traffic for each solution, along with how many HTTP requests were made. This comparison is not entirely justifiable because the output of this thesis does not use the same syntax as the one returned by Comunica. However, it is intuitively clear that even for low-result queries like the ones used in this evaluation, the solution of this thesis requires a lot less network requests. In fact, apart from possible pagination, the number of HTTP requests that need to be made to fetch and verify outsourced BGP query execution is static and equal to 2 (or 1 if bundled into one request). Comunica, on the other hand, is subject to a positive correlation between the query complexity and total incoming traffic. Therefore, if the requesting party (**R**) has to fetch the results with limited bandwidth, using Comunica will be slower unless the number of requests made is smaller than 2. This is only the case for BGP queries that have only 1 triple pattern and less than 201 results. With these conclusions, hypothesis $\mathbb{H}_{1.3}$ can be accepted.

# Wins	Comunica		Thesis	
	IN traffic	# Requests	IN traffic	# Requests
66	109.00 kB	19	2.17 kB	2
67	81.43 kB	13	2.15 kB	2
91	33.54 kB	5	2.14 kB	2
182	33.15 kB	3	2.14 kB	2

Table 6.6: Incoming network traffic comparison of Comunica vs. fetching the proof and output of this thesis

6.2.5 The prover

In light of the second research question, a hypothesis was made about how much more work is needed by the party executing the BGP query. To evaluate this, the dynamic iterator pipeline algorithm needs to be compared when operating with and without computational integrity. Comunica gives us the tools to test the algorithm without integrity and compare with the amount of work done by the prover ($\mathcal{P}$) in this thesis. As hypothesis $\mathbb{H}_{2.2}$ only includes an order of magnitude, it suffices to have an average for the time required and memory used by both solution.

Comunica took on average $1.02\,s$ to complete while using $61.49\,MB$ of memory. For the prover this was a lot more: nearly 42 minutes and $20.68\,GB$ on average. Both time and memory required for executing a BGP query with computational integrity is 3 orders of magnitude larger compared to executing the TPF algorithm locally using Comunica. With these results, hypothesis $\mathbb{H}_{2.2}$ has to be rejected. Furthermore, Section 6.3 will show how this comparison is even worse for larger compile-time constants.

The main reason that the prover needs so much memory is due to memory-consistency. In order to provide verifiable computation, the prover needs a snapshot of all memory locations at any moment during the execution of the program. This is a known bottleneck of current proving systems.

6.3 Scaling

To better illustrate how the constants defined during compile-time influence verifier and prover scalability, a new set of constants was set up. Only the most important parameters were increased, see Table 6.7 for a comparison of both sets of values. The verifiable program $\mathcal{C}$ was recompiled using these constants and the query with the largest result set was used to evaluate the verification. Table 6.8 compares some key metrics for both sets and shows how the requirements for verification do not change or increase only slightly. The solution to this thesis thus gives excellent verifier scalability. However, the compiler and prover do not yet scale very well. Doubling the number of results and allowed number of TPF results causes more than tripling the compiler and prover resource requirements.

Variable	**Original**	**New**
Input length	42 ASCII characters	50 ASCII characters
Number of results	15	30
Total TPF results	20 triples	40 triples

Table 6.7: Constant set 2, only changed constants are shown, together with their original value.

Metric	**Constant set #1**	**Constant set #2**
Compile time	1 hour and 39 minutes	6 hours and 16 minutes
Compile memory	28.70 GB	96.53 GB
Prover time	42 minutes	3 hours
Prover memory	20.68 GB	84.50 GB
Prover key size	6.38 GB	26.38 GB
Verifier time	0.16s	0.14s
Verifier memory	11.51 MB	11.53 MB
Verification key size	50.82 kB	65.08 kB
Proof size	134 B	134 B

Table 6.8: Comparison of timing, memory usage and component sizes for constant set 1 vs. 2.

This chapter evaluated the solution of this thesis, which offers verifiable delegation of BGP query execution, to local execution using the TPF algorithm. Fetching and verifying the results from a pre-computed query proved to be faster and use less memory and bandwidth than executing the query locally. Verification time and memory was shown to be independent of the result size. The biggest downside of the solution introduced in Chapter 4 is the increased complexity for the party actually executing the query. To prove soundness and completeness of the results, he has to perform work that is 3 orders of magnitude larger compared to executing without these reassures. The next chapter finishes up the work done in this thesis by discussing its relevance and possible future work.

Chapter 7

Conclusions

7.1 Future work

The limitations from Section 4.4 and its implications on compilation and prover resources is clearly something that hinders the applicability of this thesis. This section will discuss possible future work to mitigate these limitations and make the solution more practical.

Circuit size The most limiting factor is the produced circuit size for relatively low predefined constants. This results in exceptionally large resource requirements for both the compiler and the prover, which is certainly something to take in mind when designing use cases. The usage of arithmetic circuits is inherit to the non-interactive proof system used in the background. The current state of the art still requires a lot of resources to provide a valid assignment to the constraints of the circuit. For this reason, possible future work includes lowering the circuit size. A first method would be to optimize the current implementation. At the moment, all triples representing the minimum triple pattern are fetched at once. This imposes a large constraint on the maximum amount of triples that can match the minimum triple pattern because they all have to be stored concurrently. However, using proper pagination, a fixed and low amount of triples can be processed at a time and would reduce the circuit size considerably at relatively low implementation cost.
A second method could be re-using parts of the circuit. At the moment, the *Pequin* toolchain translates high-level C code to a set of constraints to be used in an arithmetic circuit. For loops, this essentially means duplicating a lot of constraints for every loop iteration. Future optimizations could thus include lower circuit sizes by using smarter translation of high-level C

code.
Another way to lower the circuit size is to find a more optimal TPF algorithm that is not recursive. For example, for some queries, it might be easier to first fetch all triples for every triple pattern and then aggregate internally. However, this needs additional research in whether or not this would improve the performance.

Proof systems Closely related to the previous paragraph, is the current state of proof systems. Currently, a naive way of assuring memory-consistency is being used. That is, a snapshot of memory at every single point during execution is stored and verified after each operation. It explains the significant memory usage of the prover and is thus not yet very practical. Re-using part of the circuit helps with this and a more "blocked" structure was already proposed by Ben Sasson et al. to provide better memory scalability [58], but goes out of scope for this thesis.
zk-STARKs are a promising new technology, which do not require a critical setup phase, offer post-quantum security and have better prover scalability. Using zk-STARKs would certainly improve the feasibility of this thesis by eliminating the setup phase and decreasing prover runtime. However, no equivalent to *Pequin* was available as zk-STARKs were only introduced in 2018. Furthermore, while zk-STARKs have better scalability, memory-consistency still requires a good amount of memory to be available. Migrating to a back-end using zk-STARKs instead of zk-SNARKs is left for future work.
Another recently introduced system, Spice, is built on top of *Pequin* and improves it significantly by introducing concurrent computation models [59]. The authors were contacted, but unfortunately, their code was not publicly available yet and therefore also remains up to future work.

SPARQL-compliance The solution to this thesis is not fully SPARQL-compliant. The removal of URIs and the I/O characteristics are another factor in why the solution is not practical yet. However, as mentioned before, the concept of URIs are rather simple to re-introduce but requires extending the verifiable program $\mathcal{C}$. One of the possible solutions was already mentioned in Section 4.4 and proposes to use prefixes only. Reading them out once, at the start of the program, and sending them along the the auxiliary variables would then make $\mathcal{C}$'s input fully SPARQL-compliant. This, however, would increase the circuit size and again more work for the prover. Nevertheless, it would not cause exponential growth, contrary to increasing some

constants. Converting the output of the system to a proper serialization format is rather low effort and is not of primary interest to make this solution more practical.
Additionally, a SPARQL query goes hand in hand with one or more data sources. During the explanation of the architecture of this system, it was mentioned that the data source was always DBpedia. If queries need to be made over other data sources, separate auxiliary scripts would have to be consulted. A more general solution can be constructed in which $\mathcal{C}$ accepts a list of data sources, similar to how the Comunica web engine operates[1]. The counter, fetcher and signer auxiliaries would then have to be updated to accept these as well and act accordingly. Note that using multiple data sources also implies multiple digital signatures and therefore additional extensions to $\mathcal{C}$ are required.

Blockchain Lastly, the proposed blockchain structure introduced in Chapter 5 is left open for future work. The description assumed one SPARQL query per block. However, this is certainly not required. Depending on the actual network requirements, multiple queries could be included in one block, similar to the concept of transactions in Bitcoin.

7.2 Discussion

This thesis touched on the quite novel concept of verifiable computation delegation. While the principles behind it have long been known, research towards practical applications is only very recent. All in all, it is still a very academical concept. Zcash is one of the only real-world applications and their engineers hand-crafted the circuit, constraint by constraint. It shows just how much work is still necessary to use this technology in applications.
This thesis has, without a doubt, promising results and with the future work discussed earlier, could very well be used to push the re-decentralization of the world wide web. Just like modern machine learning algorithms took their time to be practical, verifiable computation might very well be the standard way to write code in the future. Trusted hardware solutions for verifiable computation already exist, but converting existing setups comes with a cost that is not to be underestimated. With the recent advancements in proof systems, application logic can be run on top of potentially compromised hardware while still offering proofs of computational integrity.
Additionally, proofs for verifiable computation delegation are perfect for blockchain scenarios. Vitalik Buterin, founder of Ethereum and major influencer in the scene, is a big advocate towards

[1]`http://query.linkeddatafragments.org/`

more usage of verifiable computation in existing blockchain applications. Executing SPARQL queries with computational integrity could prove to be a very useful Proof of Work and can eliminate the useless energy-draining of the current mining industry. Using the concept of this thesis and blockchain technology, promising use cases arise for, among other things, public health care and private data processing.

7.3 Conclusion

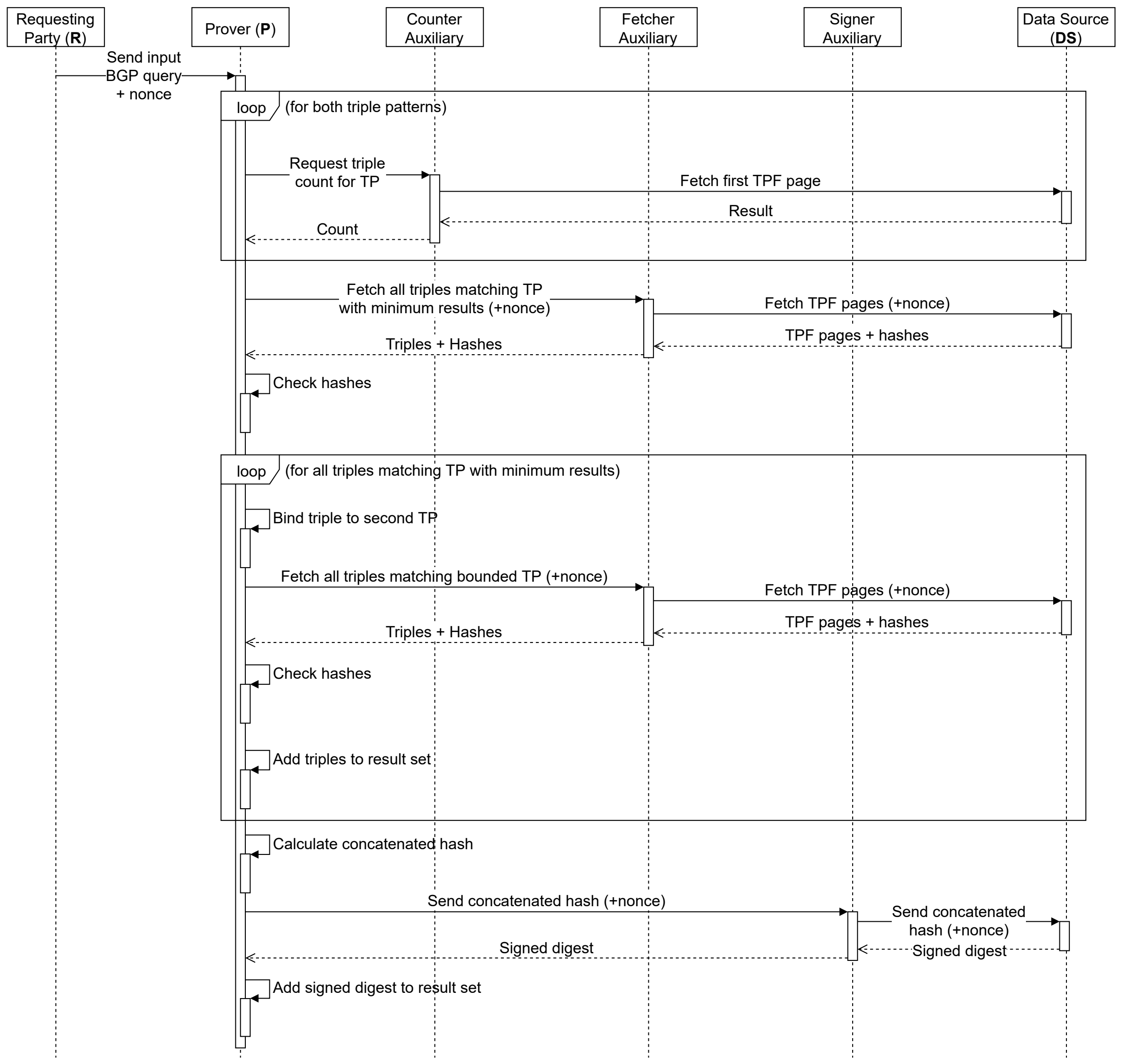
The concept of TPF was introduced to relieve Linked Data servers from their heavy computational load of executing SPARQL queries. By moving essential aggregations to the client-side, data servers profit from better availability rates. However, this shift comes with more work for a potentially resource constrained client device and outsourcing back to the data server is not an option. This thesis introduced verification of BGP queries executed by untrusted third parties. By moving the computational load to a computationally strong and willing third party, both the data server and the client can relax their resource requirements.

Using non-interactive proof systems and the TPF dynamic iterator pipeline algorithm, the computational integrity of the third party can be verified. This resulting systems was evaluated and measure do be up to 7 times faster and uses 60% less memory. Additionally, the amount of network requests that need to done is static and does not depend on the query complexity.

The relevance of the results were illustrated with two use cases and a structure for useful Proof of Work in blockchain applications.

Appendix A

Sequence diagram of proof protocol



Requesting Party (**R**)
Prover (**P**)
Counter Auxiliary
Fetcher Auxiliary
Signer Auxiliary
Data Source (**DS**)
Send input BGP query + nonce
loop (for both triple patterns)
Request triple count for TP
Fetch first TPF page
Result
Count
Fetch all triples matching TP with minimum results (+nonce)
Fetch TPF pages (+nonce)
TPF pages + hashes
Triples + Hashes
Check hashes
loop (for all triples matching TP with minimum results)
Bind triple to second TP
Fetch all triples matching bounded TP (+nonce)
Fetch TPF pages (+nonce)
TPF pages + hashes
Triples + Hashes
Check hashes
Add triples to result set
Calculate concatenated hash
Send concatenated hash (+nonce)
Send concatenated hash (+nonce)
Signed digest
Signed digest
Add signed digest to result set

Appendix B

I/O syntax

Pequin uses its own special syntax for the input and output of both the program and its exogenous computations. Furthermore, just like normal C programs have a `main` function, every program that is to be compiled with *Pequin* requires at least one function (`compute()`) as entry point. This function should have exactly two arguments: the input and output. Listing B.1 gives the entry point used in the implementation of this thesis, including the (required) struct definitions for the input and output. The structs itself can then be defined according to the user's needs. Note that the example in Listing B.1 is not complete and misses additional struct definitions and constants in order to be valid C.
The prover $\mathcal{P}$ requires two files as input: representing the input and output locations of the program. Both files contain flattened projections of the structs used in the entry point of the program. For example, in the solution of this thesis, `MAX_INPUT` was equal to 45, meaning that the input file consisted of 46 lines. The first 45 lines being the ASCII representation of the characters in the SPARQL query. The last line being the nonce chosen by the requesting party. The output of the program uses the same structure and represents the output struct given to `compute()`.
The output received from exogenous computations (**AUX** executables in this thesis) follow a similar structure. For this thesis, this meant making the Python scripts output integers on every line, representing the struct defined in the program $\mathcal{C}$. As an example, the output struct used for the signer auxiliary is given in Listing B.2, again excluding some necessary other definitions. The input to these exogenous computations is a little different. The function `exo_compute()` is the main component in these exogenous computations and requires 4 arguments:

1. An array of input arrays

```c
typedef struct In {
        uint8_t sparql[MAX_INPUT];
        uint32_t nonce;
} In_t;

typedef struct Out {
        int status;
        uint32_t hash[HASH_LENGTH_32];
        uint8_t signature[ECDSA_LENGTH_8];
        field_collection_t results[MAX_RESULTS];
} Out_t;

void compute(In_t *input, Out_t *output) {
        // Core logic
}
```

Listing B.1: *Pequin*'s entry point

2. An array representing the length of each array in the previous argument
3. A variable representing the output of the exogenous computation
4. A number indicating which auxiliary to invoke

The elements from the input arrays then get transformed to the rational notation used in Haskell (e.g. 10%1 is 10; 1%2 is 0.5). All arrays then get surrounded with `[ ]` and passed to to the auxiliary on `stdin`. An example of an exogenous call is given in Listing B.3 and is similar to the call to the signer auxiliary. Depending on the actual values, this call would result in the signer auxiliary receiving: `[ 2156425442%1 2531826028%1 2857901317%1 929717984%1 3810751703%1 ] [ 1337%1 ]`. Here, the first 5 elements represent the hash, split into chunks of 32 bit. The last element is the nonce.

```
typedef struct confirm_exo {
        int status; // 0 = Error, 1 = Success
        uint8_t signature[ECDSA_LENGTH_8]; // 48 bytes for ECDSA
           with NIST192p
} confirm_exo_t;
```

Listing B.2: Output struct for the signer exogenous computation

```
void call_signer(input_nonce, total_hash) {
        uint32_t nonce[1] = { input_nonce };
        uint32_t *exo_inputs[2] = { total_hash, nonce };
        uint32_t exo_lengths[2] = { HASH_LENGTH_32, 1 };
        confirm_exo_t exo_output;
        exo_compute(exo_inputs, exo_lengths, exo_output, 2);
}
```

Listing B.3: Exogenous call to the signer auxiliary

Bibliography

[1] L. Shanhong. (2016) Global big data industry market size 2011-2026. https://www.statista.com/statistics/254266/global-big-data-market-forecast/.

[2] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, and P. Colpaert, "Triple Pattern Fragments: A low-cost knowledge graph interface for the Web," *Journal of Web Semantics*, vol. 37-38, pp. 184–206, 2016.

[3] T. Berners-Lee. (2018) One Small Step for the Web. . . . https://inrupt.com/blog/one-small-step-for-the-web.

[4] R. Verborgh. (2019) Getting my personal data out of Facebook. https://ruben.verborgh.org/facebook/.

[5] C. Buil-Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche, "SPARQL Web-Querying Infrastructure: Ready for Action?" in *The Semantic Web – ISWC 2013*. Springer, Berlin, Heidelberg, oct 2013, pp. 277–293.

[6] R. Verborgh, M. Vander, S. P. Colpaert, S. Coppens, E. Mannens, and R. Van De Walle, "Web-Scale Querying through Linked Data Fragments," 2014.

[7] T. Berners-Lee. (2009) Linked Data - Design Issues. https://www.w3.org/DesignIssues/LinkedData.html.

[8] A. Seaborne and E. Prud'hommeaux, *SPARQL Query Language for RDF*, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/, W3C Std., jan 2008.

[9] WorldWideWebSize. (2019) The size of the World Wide Web (The Internet). https://www.worldwidewebsize.com/.

[10] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, may 2001.

[11] E. Miller, "An Introduction to the Resource Description Framework," *Bulletin of the American Society for Information Science and Technology*, vol. 25, no. 1, pp. 15–19, jan 2005.

[12] Amazon. (2019) Object Storage Classes – Amazon S3. https://aws.amazon.com/s3/storage-classes/.

[13] G. Graefe and Goetz, "Query evaluation techniques for large databases," *ACM Computing Surveys*, vol. 25, no. 2, pp. 73–169, jun 1993.

[14] R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle, "Querying Datasets on the Web with High Availability," in *The Semantic Web – ISWC 2014*. Springer, Cham, oct 2014, pp. 180–196.

[15] S. Wilkinson, "MetaDisk A Blockchain-Based Decentralized File Storage Application," 2014.

[16] Protocol Labs, "Filecoin: A Decentralized Storage Network," 2017.

[17] Prover.io, "Authenticity Verification of User Generated Video Files prover.io," 2018.

[18] D. Bhowmik and T. Feng, "The multimedia blockchain: A distributed and tamper-proof media transaction framework," in *2017 22nd International Conference on Digital Signal Processing (DSP)*. IEEE, aug 2017, pp. 1–5.

[19] S. Haber and W. S. Stornetta, "How to Time-Stamp a Digital Document," in *Advances in Cryptology-CRYPT0' 90*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 437–455.

[20] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[21] R. C. Merkle, "A Certified Digital Signature," in *Advances in Cryptology — CRYPTO' 89 Proceedings*. New York, NY: Springer New York, 1989, pp. 218–238.

[22] J.-J. Quisquater, M. Quisquater, M. Quisquater, M. Quisquater, L. Guillou, M. A. Guillou, G. Guillou, A. Guillou, G. Guillou, and S. Guillou, "How to Explain Zero-Knowledge Protocols to Your Children," in *Advances in Cryptology — CRYPTO' 89 Proceedings*. Springer New York, 1989, pp. 628–631.

[23] U. Feige, A. Fiat, and A. Shamir, “Zero-knowledge proofs of identity,” *Journal of Cryptology*, vol. 1, no. 2, pp. 77–94, jun 1988.

[24] S. Goldwasser, S. Micali, and C. Rackoff, “The Knowledge Complexity of Interactive Proof Systems,” *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, feb 1989.

[25] A. Fiat and A. Shamir, “How To Prove Yourself: Practical Solutions to Identification and Signature Problems,” in *Advances in Cryptology — CRYPTO’ 86.* Springer Berlin Heidelberg, 1986, pp. 186–194.

[26] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, “From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again,” in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS ’12.* ACM Press, 2012, pp. 326–349.

[27] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, “Scalable, transparent, and post-quantum secure computational integrity,” 2018.

[28] G. Di Crescenzo and H. Lipmaa, “Succinct NP Proofs from an Extractability Assumption,” in *Logic and Theory of Algorithms.* Springer, Berlin, Heidelberg, 2008, pp. 175–185.

[29] S. Arora and S. Safra, “Probabilistic checking of proofs: a new characterization of NP,” *Journal of the ACM*, vol. 45, no. 1, pp. 70–122, jan 1998.

[30] J. Groth, “Short Pairing-Based Non-interactive Zero-Knowledge Arguments,” in *Advances in Cryptology - ASIACRYPT 2010.* Springer, Berlin, Heidelberg, 2010, pp. 321–340.

[31] M. Conti, E. Sandeep Kumar, C. Lal, and S. Ruj, “A Survey on Security and Privacy Issues of Bitcoin,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.

[32] J. Eberhardt and S. Tai, “ZoKrates - Scalable Privacy-Preserving Off-Chain Computations,” in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData).* IEEE, jul 2018, pp. 1084–1091.

[33] I. Meckler and E. Shapiro, “Coda: Decentralized cryptocurrency at scale,” 2018.

[34] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. W. ¶3, and G. Maxwell, “Bulletproofs: Efficient Range Proofs for Confidential Transactions,” 2017.

[35] E. Ben-Sasson, I. Bentov, A. Chiesa, A. Gabizon, D. Genkin, M. Hamilis, E. Pergament, M. Riabzev, M. Silberstein, E. Tromer, and M. Virza, “Computational Integrity with a Public Random String from Quasi-Linear PCPs,” in *Advances in Cryptology – EUROCRYPT 2017*. Springer, Cham, 2017, pp. 551–579.

[36] S. D. Warren and L. D. Brandeis, “Right to Privacy,” *Harvard Law Review*, vol. 4, 1890.

[37] P. C. K. Hung and V. S. Y. Cheng, *Privacy*. Springer US, 2009, pp. 2136–2137.

[38] L. Sweeney, “k-Anonymity: A model for protecting privacy,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, oct 2002.

[39] C. Dwork, “Differential Privacy,” in *Encyclopedia of Cryptography and Security*. Springer US, 2011, pp. 338–340.

[40] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, “From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again,” in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS ’12. ACM, 2012, pp. 326–349.

[41] B. Parno, J. Howell, C. Gentry, and M. Raykova, “Pinocchio: Nearly practical verifiable computation,” *Proceedings - IEEE Symposium on Security and Privacy*, pp. 238–252, 2013.

[42] V. Buterin. (2016) Quadratic Arithmetic Programs: from Zero to Hero - Vitalik Buterin - Medium. https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649.

[43] A. Gabizon. (2017) Explaining SNARKs Part I: Homomorphic Hidings - Electric Coin Company. https://electriccoin.co/blog/snark-explain/.

[44] R. Tamassia, “Authenticated Data Structures,” in *Algorithms - ESA 2003*. Springer, Berlin, Heidelberg, 2003, pp. 2–5.

[45] Y. Zhang, J. Katz, and C. Papamanthou, "Integridb: Verifiable sql for outsourced databases," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. ACM, 2015, pp. 1480–1491.

[46] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases," in *2017 IEEE Symposium on Security and Privacy (SP).* IEEE, may 2017, pp. 863–880.

[47] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. ACM, 2012, pp. 90–112.

[48] Q. Zheng, S. Xu, and G. Ateniese, "Efficient query integrity for outsourced dynamic databases," in *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop - CCSW '12.* New York, New York, USA: ACM Press, 2012, p. 71.

[49] S. Bajaj and R. Sion, "CorrectDB: SQL Engine with Practical Query Authentication," 2013.

[50] NYU and UT Austin. (2017) Pepper: toward practical verifiable computation. https://www.pepper-project.org/.

[51] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish, "Verifying computations with state," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. ACM, 2013, pp. 341–357.

[52] R. S. Wahby, S. Setty, Z. Ren, A. J. Blumberg, and M. Walfish, "Efficient RAM and control flow in verifiable outsourced computation," 2015.

[53] Q. H. Dang, "Secure Hash Standard," National Institute of Standards and Technology, Tech. Rep., jul 2015.

[54] D. Malone and K. O'Dwyer, "Bitcoin Mining and its Energy Footprint," in *25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communities Technologies (ISSC 2014/CIICT 2014).* Institution of Engineering and Technology, 2014, pp. 280–285.

[55] A. de Vries, "Bitcoin's Growing Energy Problem," *Joule*, vol. 2, no. 5, pp. 801–805, may 2018.

[56] M. Ball, A. Rosen, M. Sabin, and P. Nalini Vasudevan, "Proofs of Useful Work," 2017.

[57] R. Taelman, J. Van Herwegen, M. Vander Sande, and R. Verborgh, "Comunica: A Modular SPARQL Query Engine for the Web," in *The Semantic Web – ISWC 2018*. Springer, Cham, oct 2018, pp. 239–255.

[58] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Springer Berlin Heidelberg, 2013, pp. 90–108.

[59] S. Setty, S. Angel, T. Gupta, and J. Lee, "Proving the correct execution of concurrent services in zero-knowledge," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18).* USENIX Association, Oct. 2018, pp. 339–356.

FACULTY OF ENGINEERING
AND ARCHITECTURE

Computational integrity for outsourced execution of SPARQL queries

Serge Morel
Student number: 01407289

Supervisors: Prof. dr. ir. Ruben Verborgh, Dr. ir. Miel Vander Sande
Counsellors: Ir. Ruben Taelman, Joachim Van Herwegen

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2018-2019

GHENT
UNIVERSITY