Rheinisch-Westfälische Technische Hochschule Aachen
Informatik 5, Information Systems
Prof. Dr. Stefan Decker

Bachelor Thesis Proposal

An application of verifiable computation to an anonymisation algorithm

Daniel Pujiula Buhl

January 15, 2019

Advisor Benjamin Heitmann, Ph.D.
Supervisor Prof. Dr. Stefan Decker

Contents

1 Introduction

1.1 Motivation

Following new developments like electronic health records, which are being introduced in Germany as of the start of 2019 [Ehe], the step from analogue paperwork to electronic records in health care now becomes reality and will continue to spread. This enables to systematically use always bigger datasets for research purposes with comparably small effort, let it be in public or private research facilities. At the same time, this development requires the most careful and sensible handling of the privacy protection of the health care data. Such data is of exceptional sensitivity and the handling of the data must fulfill the highest requirements of data protection. Rightly, the transfer and reuse of health care data for new purposes or in new contexts is very restricted. For statistical evaluation of such data, it has to be ensured that the data is properly anonymized.

Imagine a producer of pharmaceuticals wants to evaluate the effectiveness of an existing medication. Instead of conducting a targeted study for this, the company wants to use existing data of treatments with this medication from several hospitals. The company buys the aggregated and anonymized data from a data provider. This provider gets the data from hospitals. Medical data is sensitive personal data. Therefore, the hospitals may not give away data which could be associated with individuals. A thorough anonymisation is required. To increase the value of the data, it should be first aggregated and then anonymised. This is the job of the provider. The hospitals have an interest to be able to prove that the combined data sold from the provider does comply to data protection law requirements of anonymisation. At the same time, the pharmaceuticals company has an interest in receiving real data from hospitals and in a proof of the sample size to determine the weight of the aggregated, anonymised dataset from the provider. Both the hospital and the provider cannot check these requirements themselves without access to all input data. Due to the sensitive content, this input is private and cannot be published to other data providers (hospitals), nor to the buyer (pharmaceutical company).

The provider needs a way to prove to their clients that the aggregation and anonymisation of the data has been executed correctly, without exposing the private data.

1.2 Thesis Goal

The goal of this thesis is to apply verifiable computation in form of non-interactive zero-knowledge proofs (NIZKs) on an anonymization algorithm. This shall add verifiability to anonymization of data without a need to potentially compromise sensitive data in the process of verification. It shall demonstrate a possible application of zk-SNARKs outside of the area of cryptocurrencies.

Zero-Knowledge Proofs are around since the 90's [GMR85; BFM88]. With a zero-knowledge proof, it is possible to prove the knowledge of any NP-Instance without revealing anything about that instance itself. In other words, one can prove *that* one knows a certain information, without revealing anything about the information itself.

A traditional zero-knowledge proof would be interactive; thus require both parties (the prover and the verifier) to be able to communicate with each other. With *non-interactive zero-knowledge proofs* (NIZKs), the proof may be verified independently from its construction. One particular form of such NIZKs are zk-SNARKs (zero-knowledge succinct non-interactive arguments of knowledge). They are succinct, meaning *short and easy to verify*. These properties make them suitable for use in Blockchains, where space is very limited and every transaction has to be verifiable anytime and independently from the prover. Furthermore, verification has to be fast, as all participants in the blockchain are required to verify all entries of the blockchain.

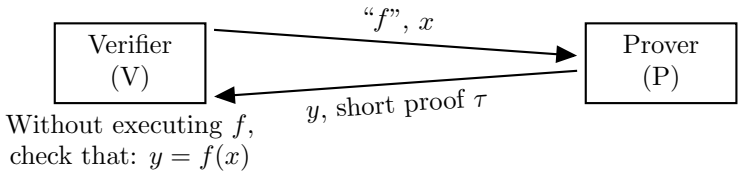

Figure 1: Concept of verifiable computation [Wal]

Currently, zk-SNARKs are prominently applied in the cryptocurrencies zCash and Ethereum. While in theory promising to be applicable to any NP-Instance, the current application of zk-SNARKs in Blockchains only require the verification of some basic constraints - stripped down, the verification of the correct concatenation of a few hashes [ref zCash].

The thesis shall demonstrate the application of zk-SNARKs in another domain - anonymization - on a more complex problem, the verification of the k-Anonymity-property. The focus will be to provide a proof-of-concept.

1.3 Outline

This proposal starts with a background chapter covering verifiable computation in Section 2.1, zero knowledge proofs in Section 2.2, homomorphic encryption in Section 2.3, zk-SNARKs in Section 2.4 and a short introduction of anonymization algorithms in Section 2.5. In Section 3, the use case and requirements for the thesis are explored and in the following sections the conceptual approach and the plan for realisation and implementation are presented. The proposal finishes with the evaluation plan in Section 6 and a project plan in Section 7.

2 Background and Related Work

2.1 Verifiable Computation

This section provides an overview of the concept of verifiable computation in general. It formally introduces verifiable computation schemes and their properties in Section 2.1. In Section 2.1, the special case of non-interactive verifiable computation is presented and an extended setting for verifiable computation is introduced.

Verifiable Computation in general

There are a number of different approaches to the general problem of verifiable computation. BOINC and similar projects use so-called audit-based solutions. These typically require the client or a random number of workers to recalculate some portion of the work [GGP10].

In the general setting, there is a client $\mathcal{C}$ who wants a server $\mathcal{S}$ to evaluate a function f on some input x. Therefore $\mathcal{C}$ transmits (encodings of) f and x to $\mathcal{S}$. $\mathcal{S}$ will do the computation and get a result $y = f(x)$. $\mathcal{S}$ will then transmit (an encoding of) y back to $\mathcal{C}$. To proof the correctness of the result, a verifiable computation scheme can be used, which typically involves sending back an additional proof τ to $\mathcal{C}$ [Buc+16].

This setting is also depicted in Fig. 1, where $\mathcal{C}$ is the verifier and $\mathcal{S}$ is the prover. Gennaro, Gentry and Parno [GGP10] define a verifiable computation scheme $\mathcal{VC}$ as follows (quoted and slightly adapted from [Buc+16]):

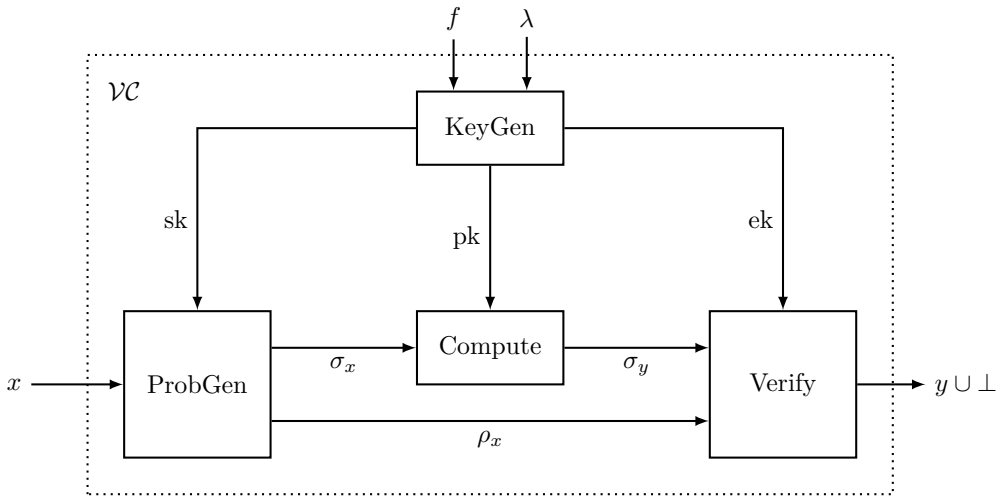

Figure 2: Illustration of a verifiable computation scheme $\mathcal{VC}$

Definition 1 (Verifiable Computing Scheme)**.** *A* Verifiable Computing Scheme $\mathcal{VC}$ *is a tuple of the following probabilistic polynomial-time (PPT) algorithms:*

1. **KeyGen**$(1^\lambda, f) \rightarrow (\text{sk}, \text{vk}, \text{ek})$*: The probabilistic key generation algorithm takes a security parameter* λ *and the description of a function* f*. It generates a secret key* sk*, a corresponding verification key* vk *and a public evaluation key* ek *(that encodes the target function* f*) and returns all these keys.*

2. **ProbGen**$(\text{sk}, x) \rightarrow (\sigma_x, \rho_x)$*: The* problem generation *algorithm takes a secret key* sk *and data* x*. It outputs a decoding value* ρ_x *and a public value* σ_x *which encodes the data* x*.*

3. **Compute**$(\text{pk}, \sigma_x) \rightarrow \sigma_y$*: The* computation algorithm *takes the evaluation key* ek *and the encoded input* σ_x*. It outputs an encoded version* σ_y *of the function's output* $y = f(x)$*.*

4. **Verify**$(\text{vk}, \rho_x, \sigma_y) \rightarrow y \cup \perp$*: The* verification algorithm *obtains a verification key* vk *and the decoding value* ρ_x*. It converts the encoded output* σ_y *into the output of the function. If* $y = f(x)$ *holds, it returns* y *or outputs* $\perp$ *indicating that* σ_y *does not represent a valid output of* f *on* x*.*

A verifiable computing scheme must satisfy some constraints, namely *correctness*, *security* and *efficiency*. Furthermore, it is desirable to incorporate a level of *privacy* to the scheme.

Correctness

A verifiable computing scheme is correct if the algorithms fulfill their intended function. This is defined as follows:

Definition 2 (Correctness [Buc+16])**.** *A verifiable computing scheme* $\mathcal{VC}$ *is correct if for any choice of* f *and output* $(\text{sk}, \text{vk}, \text{ek}) \leftarrow \text{KeyGen}(1^\lambda, f)$ *of the key generation algorithm it holds that* $\forall x \in \text{Domain}(f)$*, if* $(\sigma_x, \rho_x) \leftarrow \text{ProbGen}(\text{sk}, x)$ *and* $y \leftarrow \text{Compute}(\text{ek}, \sigma_x)$*, then* $y = f(x) \leftarrow \text{Verify}(\text{vk}, \rho_x, \sigma_y)$*.*

Security

A verifiable computing scheme is *secure* if the probability of a fraud (i.e. convincing the verifier of a correct computation without having executed such a computation) is negligible [GGP10], [Buc+16].

Publicly and Privately Verifiable Computation

Verifiable computing schemes can be either privately verifiable or publicly verifiable [Buc+16]. Privately verifiable computing schemes only allow the Client $\mathcal{C}$ to verify the computation of $\mathcal{S}$ using the verification algorithm. The client has to keep the verification key secret; a revelation of the client's verification key to the server would break the verifiable computing scheme.

A publicly verifiable computing scheme allows anybody to verify the computation. For that, the verifiable computing scheme must allow the publication of the verification key, without compromising the scheme.

These properties are defined as follows:

Definition 3 (Privately Verifiable Computation Scheme [Buc+16])**.** *If $sk = vk$ and $\mathcal{C}$ needs to keep ρ_x private, $\mathcal{VC}$ is called a privately verifiable computation scheme.*

Definition 4 (Publicly Verifiable Computation Scheme [Buc+16])**.** *If $sk \neq vk$ and $\mathcal{C}$ needs to keep ρ_x private, $\mathcal{VC}$ is called a privately verifiable computation scheme.*

A publicly verifiable computing scheme allows anyone to check the correctness of the performed computation, not only the client. This property is used in the cryptocurrency Zcash to enable everyone to check the correctness and integrity of transactions in the blockchain without performing the validation by themselves and even without knowing all inputs required to be able to perform that validation. This leads us to another property of verifiable computing schemes: *Input and Output Privacy.*

Input and Output Privacy

The general setting, as summarized at the beginning of Sect. 2.1, does protect the integrity of the computation. The value of the client's input parameter x might be revealed to the server. In addition to the integrity of the computation, the protection of the input value(s) is a useful property [GGP10].

A formal definition of *input privacy* is given in [Buc+16]. It follows the idea that for a private input when there are two results of the computation and two private inputs, it is not possible to determine which of the inputs corresponds to which result. This is expressed in form of an experiment and a constraint on the probability that the right input can be guessed.

Not stated in [Buc+16], but important to keep in mind: Depending on the function f, the output may already leak information on the input. Consider the function that maps the input to the output:

$$f : x \mapsto x$$

It is not possible to provide input privacy in such a function without additionally providing *output privacy.* Output privacy refers to the output of the computation and is analogous to *input privacy.* Without output privacy, input privacy can only hide all the information of the input that is not revealed by the output of the computation itself.

On the Efficiency of Verifiable Computing Schemes

The whole point of a verifiable computation scheme, as introduced in [GGP10], is to enable outsourcing of computation to untrusted workers. This is only true if the client can save some work by outsourcing the computation, meaning that "the time to encode the input and verify the output must be smaller than the time to compute the function from scratch." [GGP10]

Definition 5 (Outsourceable [GGP10])**.** *A $\mathcal{VC}$ can be outsourced if it permits efficient generation and efficient verification. This implies that for any x and any σ_y, the time required for* $\text{ProbGen}_{SK}(x)$ *plus the time required for* $\text{Verify}(\sigma_y)$ *is $\mathcal{O}(T)$, where T is the fastest known time required to compute $f(x)$.*

However, when considering input privacy, metrics of what cannot be considered efficient anymore, change. Verifiable computing in the form of non-interactive zero-knowledge proofs, as introduced and explained in the following sections, the key motivation of performing such a compututation might not be the ability to outsource the computation, but the ability to convince one (or even multiple) verifiers of the correctness of results *without revealing the input of the computation.* At this point, the notion of *outsourceability* does not apply anymore.

Non-interactive verifiable computing

The thesis will focus on one approach to *non-interactive* verifiable computing, called zk-SNARKs (Sect. 2.4). In non-interactive verifiable computation, there is no back-and-forth communication required between the working party and the verifying party; after an initial setup phase, the proof may be constructed anytime without connecting back to the client, and the verification of the proof can then be executed anytime later (and in case of publicly verifiable computation, also by anyone) without any extra communication to the prover being required.

The client-server analogy does not work very well for this setting. In the following, the party that performs the computational work and produces a proof of this work is also referred to as the *prover* or the *worker*. The party that wants to verify the proof is also referred to as the *verifier*. These terms come from the zero-knowledge terminology. A short introduction to zero-knowledge proofs is given in Sect. 2.2.

In a non-interactive setting, the focus moves from a client who wants to outsource a computation, to a prover who wants to prove the correctness of their computation. The prover may have their own private input to the computation about which nothing should be revealed (other than what is inherently revealed by the output of the computation). Therefore, Parno et al. introduce the *extended setting* for verifiable computation [Par+13]. In this setting, the outsourced computation is a function $F(x, w)$ of two inputs - the client's input x and the worker's auxiliary input w. zk-SNARKs are based on this setting.

2.2 Zero-Knowledge Proofs

Consider the scenario where person P knows a secret S and wants to prove to another person V, that she indeed knows S, without revealing S. The solution to this problem is called a Zero-Knowledge Proof. There are two fundamentally different variants of zero-knowledge proofs: *interactive zero-knowledge proofs* and *non-interactive zero-knowledge proves.* Interactive zero-knowledge proofs require the interaction of prover and verifier, while non-interactive zero-knowledge proofs are constructed in such a way that they do not require any interaction when performing the actual proof.

Every zero-knowledge proof needs to satisfy the following properties [Ros]:

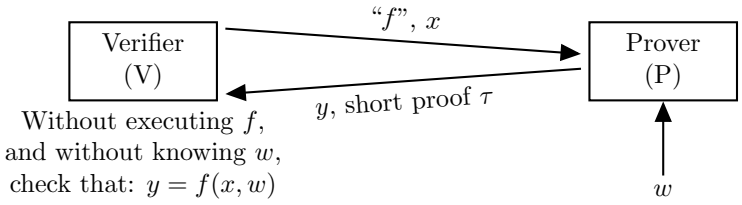

Figure 3: The extended setting for verifiable computation

- **Completeness:** If the statement is true, then an honest verifier can be convinced of it by an honest prover.
- **Soundness:** If the prover is dishonest, the prover can't convince the verifier of the soundness of the statement by lying.
- **Zero-Knowledge:** If the statement is true, the verifier will have no idea what the statement actually is.

2.2.1 Interactive Zero-Knowledge Proofs

A zero-knowledge proof typically is a protocol that consists of the repetition of three steps: *commitment, challenge, and response.* Before that, the prover might additionally publish some public information to the verifier, which is called the prover's public key in the following.

1. **Commitment.** The Prover sends a *lockable box* (or a number of lockable boxes) to the verifier. This box contains information that is encrypted. The prover may open the box at any time by sending the secret encryption key for the box to the verifier. At the same time, it is impossible to retrospectively change the contents of the box, for example by sending a fake key that opens a different content in that same box. Thus, the prover commits himself to the information in the box.

   The information hidden in the box is in a relation to the secret that is only known to the prover.

2. **Challenge.** The verifier then randomly chooses a number $\in \{0, 1\}$ and sends it to the prover.

3. **Response.** Depending on the challenge, the prover know shows one of two mathematical properties of the commitment. This can involve opening the boxes, or only a part of them, and/or showing a relation of the commitment to the prover's public key.

For example, the secret can be the knowledge of a Hamiltonian path in a graph [Ung]. In this case, the prover knows a graph $G = (V, E)$ that contains a Hamiltonian path $H \subset E$. G is the public key of the prover and H is the secret. It is NP-complete, to test if G contains a Hamiltonian path. G has to be chosen large enough that such a test is not feasible. For the commitment, the prover can construct a permutation $G' = \pi(G)$. The prover then locks all edges and all vertices in boxes and sends them to the verifier. The verifier randomly chooses the challenge $r \in \{0, 1\}$ and sends it back to the prover.

Case $r = 0$: The prover has to show the permutation. This is achieved by opening all boxes (by sending the keys to all boxes to the verifier) and additionally sending the permutation to the verifier. The verifier can now verify that $G' = \pi(G)$.

Case $r = 1$**:** The prover has to show the Hamiltonian path in G'. This is achieved by opening the boxes of the edges $H' = \pi(H)$. The verifier can now verify that these are indeed a Hamiltonian path.

Zero-knowledge proofs are constructed in such a way that there is a chance of typically $\frac{1}{2}$ that the prover cheated in each round: When guessing the challenge right, it is easy to construct a commitment that answers correctly to that challenge but would fail the other challenge. Only with knowledge of the secret, the commitment will always succeed to prove both challenges. By repetition of the commitment, challenge, response protocol n times, the chance of a successful fraud diminishes to $\frac{1}{2^n}$. The parameter n therefore determines the security of the protocol and can be chosen such that it is extremely unlikely that a prover is able to cheat successfully.

The example before was based on the Hamiltonian path. There are also zero-knowledge protocols based on 3-SAT and many other problems. Using the technique of reduction of one problem to another, any NP-Complete-Problem can be used to construct a zero-knowledge-protocol. [Ung]

2.2.2 Non-Interactive Zero-Knowledge Proofs

For an interactive zero-knowledge proof, both parties need to actively communicate. This limits the applicability of interactive zero-knowledge proofs to scenarios where they are able to communicate in both directions and respond to each other. But there are many scenarios where these prerequisites are not fulfilled. For example, in the blockchain: If someone publishes a transaction that contains a statement of zero knowledge, any person needs to be able to verify the integrity of the blockchain at any time, but it cannot be guaranteed that the original publisher is always and forever online available for an interactive protocol.

In 1986, Fiat and Shamir developed the Fiat-Shamir heuristic [FS87]. This was a breakthrough in cryptography; with this protocol, non-interactive zero-knowledge proofs became possible.

The idea of the protocol is simple: The challenge, that has formerly been chosen by the verifier, is now chosen using a hash function on the public parameters and the commitment from the prover. This way, the prover can "simulate" the interactive protocol and determine the challenge by herself. Non-interactive zero-knowledge proofs are not as strong as interactive zero-knowledge proofs; they require an additional constraint on the behaviour of the verifier called "honest-verifier". For further details, the reader is referred to [FS87].

2.3 Homomorphic encryption

Homomorphic encryption enables the use of arithmetic operations on encrypted values.

There are partially homomorphic encryption systems that only support addition and multiplication, and fully homomorphic encryption systems who support both.

Let E be a homomorhpic encryption system. For an additive homomorphic system, there exists an operation $\oplus$ such that

$$E(a) \oplus E(b) = E(a + b).$$

For an multiplicative homomorphic system, there exists an operation $\odot$ such that

$$E(a) \odot E(b) = E(a \cdot b).$$

Example:

Let p prime, $g \in \mathbb{Z}_p^*$, g generator.

$\mathbb{Z}_p^*$ is a cyclic group. All elements of $\mathbb{Z}_p^*$ can be written as $g^a \mod p$ for some $a \in \{0, ..., p-2\}$.

Define the encryption function $E : \mathbb{Z}_p^* \to \mathbb{Z}_p^*, E(x) \mapsto g^{x \mod p-1}$. The following property can be observed:

$$E(a+b) = g^{a+b \mod p-1} = g^{a \mod p-1} \cdot g^{b \mod p-1} = E(a) \cdot E(b)$$

E is a partially homomorphic function. This particular function is one of the basic cryptographic primitives because it is believed that the discrete logarithm problem is hard in $\mathbb{Z}_p^*$. [Ung]

2.4 zk-SNARKs

This section presents zk-SNARKs, a recent form of non-interactive zero-knowledge proofs, as an approach to publicly verifiable non-interactive computing with input privacy. Section 2.4.1 informally introduces the concept. In Sect. 2.4.2, the construction of zk-SNARKs as in the Pinocchio Protocol is explained.

2.4.1 Definition

A relatively new approach to non-interactive zero-knowledge proofs ([Bit+12]) are the so-called *zk-SNARKs*. zk-SNARK is an acronym that stands for *zero knowledge Succinct Non-interactive Argument of Knowledge*. zk-SNARKs basically are an elaborated form of a non-interactive zero-knowledge argument. They are commonly referred to as *succinct non-interactive zero-knowledge proofs*. With the usage of zk-SNARKs in zCash, transactions in the blockchain can be fully encrypted and yet be verified using zk-SNARKs [Zca].

This implies the following properties:

argument of knowledge The (computationally bounded) prover can convince any verifier of their knowledge of some statement.

non-interactive The argument does not require any interaction between the verifier and the prover.

succinct The proof is short and easy to verify.

zero-knowledge The proof does not give away any information other than the fact of the prover's knowledge.

A protocol or construction that satisfies these properties may be called a zk-SNARK. With the *Pinocchio Protocol* [Par+13], Parno et al. present a construction of zk-SNARKs based on Quadratic Arithmetic Programs. zk-SNARKs enable publicly verifiable computation with input privacy for the auxiliary input from the worker, as defined in the extended setting (Sect. 2.1).

zk-SNARKs can be used to prove the knowledge of an instance of an NP language without revealing this instance. For example, one can prove the knowledge of a Hamiltonian path in a specific graph without revealing that Hamiltonian path itself, or the knowledge of the input for that a hash function returns a specific value, without revealing that input. This can be used for example in cryptocurrencies to prove ownership of coins, where the input of the hash function is a secret that proves ownership of a coin and thus cannot be revealed.

The NP Language is encoded in form of a decision circuit. The circuit outputs whether the input is an instance of the NP Language or not. Thus, formally, the zk-SNARK is not directly used to compute a result for a problem, but it can be used instead to prove that a given result is indeed the solution to the problem. As this can also involve calculating the result, it depends on the implementation to use the proof generation for the computation as well.

Parno et al. define zk-SNARKs as follows:

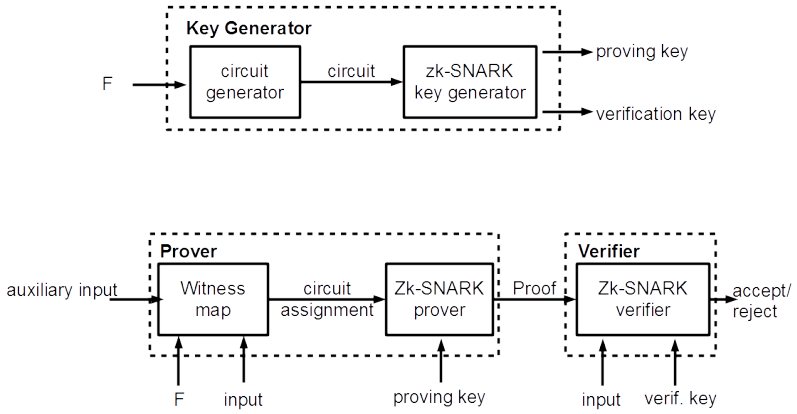

Figure 4: zk-SNARKs in Pinocchio

Definition 6 (zk-SNARK [Par+13])**.** *Let $\mathcal{L}$ be an NP language, and let C be a non-deterministic decision circuit for $\mathcal{L}$ on a given instance size n. A zk-SNARK is a 3-tuple (KeyGen, Compute, Verify) of algorithms that can be used to prove and verify membership in $\mathcal{L}$ for instances of size n.*

KeyGen$(C, \lambda) \to (pk, vk)$ *A* trusted party *takes C and the security parameter λ as input and uses* KeyGen *to create a proving key* pk *and a verification key* vk*.*

Compute$(pk, u, w) \to (y, \pi_y)$ *The worker algorithm takes the proving key* pk*, the public input u and the secret input w and outputs $y = F(u, w)$ and a proof π_y of y's correctness.*

Verify$(vk, u, y, \pi_y) \to \{0, 1\}$ *Given the verification key* vk*, the algorithm outputs* 1 *if the worker knows w such that $F(u, w) = y$.*

In comparison to $\mathcal{VC}$ as defined in Section 2.1, the zk-SNARK consists of only three algorithms. *KeyGen* is the common setup phase that needs to be executed only once for a given function/algorithm. After this, anybody can produce proofs using the proving key and anybody can verify these proofs using the verification key. No direct interaction between the prover and the verifier is required after the setup.

The proofs are very short and can be verified within a few milliseconds; even for statements about programs that are very large [BS+14]. This is a huge advantage over interactive zero-knowledge protocols which typically require lots of data exchange over several rounds.

Figure 4 shows how zk-SNARKs work in Pinocchio.

2.4.2 Construction of zk-SNARKs in Pinocchio

The following part of this proposal goes into some detail on how zk-SNARKs are constructed in Zcash. It is geared to an explanation of Zcash function principles of the Zcash project [Zca].

Under the hood, Zcash uses the Pinocchio Protocol [Par+13] to construct the required zk-SNARKs. The required process can be split up into the following steps: NP Statement → Program (Computation) → Arithmetic Circuit → R1CS → QAP → zk-SNARK.

The pipeline starts with an NP Statement that should be proven and a program (computation) that can decide on an input whether the input is an instance of the NP Statement. This program is then transformed into an Arithmetic Circuit. From the circuit, a Rank 1 Constraint System is built, whose purpose is to check that the values travel correctly through the circuit. The constraints can be bundled into one constraint, using a method shown in

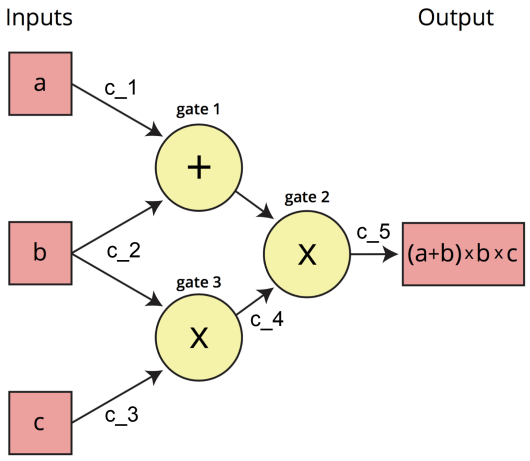

Figure 5: A simple arithmetic circuit with assigned labels [Zca]

[BS+14]. In the next step, the circuit is converted to a representation called a Quadratic Arithmetic program (QAP). This representation uses polynomials; for checking the constraint(s), it is now only required to check that two polynomials match at one randomly chosen point. Now, the polynomials have to be evaluated blindly. For this, zk-SNARKs are used, utilizing homomorphic encryption and pairings of elliptic curves. Here, *blind evaluation* means that the verifier evaluates the polynomial without knowing which point is being evaluated.

2.4.3 From the Computation to Arithmetic Circuits

An arithmetic circuit is a directed acyclic graph, where a fixed number of inputs are connected to any number of arithmetic gates. Each gate performs an arithmetic operation (like addition or multiplication) on its inputs and has an output that can itself serve as input to other gates until finally the circuit has one output. The connections of the gates are called wires.

For example, the operation

$$(a + b) \cdot b \cdot c$$

can be represented by the arithmetic circuit in Fig. 5. The red boxes on the left are the inputs, the wires from the inputs to the gates are the input wires. The right wire is the output wire, connected to the output (the red box on the right). The output contains the result of the computation represented by the circuit.

Suppose the statement that should be proved is the knowledge of $a, b, c \in \mathbb{F}_p$ such that $(a+b) \cdot b \cdot c = 9$. This statement can be transformed to the arithmetic circuit. Parno, Gentry, Howell, and Raykova have built a compiler that can compile a subset of C to an equivalent arithmetic circuit [Par+13].

2.4.4 Universal Arithmetic Circuits

At this stage, Ben-Sasson, Chiesa, Tromer, and Virza have developed a way to reduce basically any QAP to a constant size [BS+14].

This reduction step follows an amazing idea: They constructed a *universal* circuit generator.

This is achieved by a layer of indirection - in some respect similar to the idea of a Universal Turing Machine. They constructed a circuit generator for arbitrary programs in

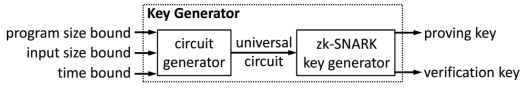

Figure 6: Offline phase (once). The key generator outputs a proving key and verification key, for proving and verifying correctness of any program execution meeting the given bounds. [BS+14]

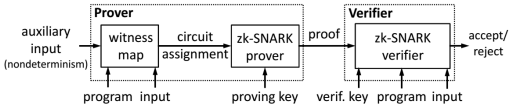

Figure 7: Online phase (any number of times). The prover sends a short and easy-to-verify proof to a verifier. This can be repeated any number of times, each time for a different program and input. [BS+14]

a von-Neumann architecture. This allows for almost any possible program - in the von-Neumann architecture, program space and data space are shared. Thus, programs are capable of changing themselves and of using techniques like in-time compilation.

Instead of generating a circuit for the given program, the universal circuit can check any program with a given input size constraint and a time constraint. This way, the actual program is transferred to be part of the input of the circuit. This changes the process slightly, as shown in Fig. 6 and Fig. 7. One key feature that is enabled by this new setup is the possibility of a common Key Generation phase, which is independent of the NP Statements that are to be proven later; these can be passed into the system as part of the public input.

This paper does not go into more details on universal arithmetic circuits; it is a fascinating topic and worth its own seminar paper. The interested reader is referred to [BS+14] for further information.

2.4.5 From arithmetic circuits to QAPs

The next step is to convert the circuit to Quadratic arithmetic programs (QAPs), or simplified speaking, to a set of polynomials.

QAPs are defined as follows:

Definition 7 (Quadratic Arithmetic Program (QAP) [Par+13])**.** *A QAP Q over field $\mathbb{F}$ contains three sets of $m+1$ polynomials $\mathcal{V} = \{v_k(x)\}, \mathcal{W} = \{w_k(x)\}, \mathcal{Y} = \{y_k\}$, for $k \in \{0...m\}$, and a target polynomial $t(x)$. Suppose F is a function that takes as input n elements of $\mathbb{F}$ and outputs n' elements, for a total of $N = n + n'$ I/O elements. Then we say that Q computes F if: $(c_1, ..., c_N) \in \mathbb{F}^N$ is a valid assignment of F's inputs and outputs,* if and only if *there exist coefficients $(c_{N+1}, ..., c_m)$ such that $t(x)$ divides $p(x)$, where:*

$$p(x) = \left(v_0(x) + \sum_{k=1}^{m} c_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} c_k \cdot w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} c_k \cdot y_k(x)\right).$$

In other words, there must exist some polynomial $h(x)$ such that $h(x) \cdot t(x) = p(x)$. The size of Q is m, and the degree is the degree of $t(x)$.

In a QAP, each multiplication gate is assigned a root r_g. The polynomial $p(x)$, with

$$p(r_g) = (\text{sum of left inputs of g} + \text{sum of right inputs of g}) - \text{output of g},$$

wire	gate 2			gate 3		
k	$v_k(r_2)$	$w_k(r_2)$	$y_k(r_2)$	$v_k(r_3)$	$w_k(r_3)$	$y_k(r_3)$
1	1	0	0	0	0	0
2	1	0	0	1	0	0
3	0	0	0	0	1	0
4	0	1	0	0	0	1
5	0	0	1	0	0	0

Figure 8: Parameters v, w, y at the roots of the multiplication gates for the polynomials in the example

.

is constructed such that it vanishes at the roots that represent the multiplication gates.

The building of a QAP from an arithmetic circuit (along with Section 2.2.1 of [Par+13]) starts with assigning an arbitrary root $r_g \in \mathbb{F}$ for each multiplication gate g in the circuit. Also, each input of the circuit and each output of a multiplication gate is assigned an index $k \in [m] = \{1...m\}$. There is no need to assign such an index to the addition gates, as they will be compressed into their contributions to the multiplication gates.

In the example from above, there are three gates; one addition gate and two multiplication gates. The multiplication gates will now be assigned arbitrary roots r_{g2} and r_{g3}. The inputs are assigned the indexes 1 through 3 (from top to bottom) and the outputs of the multiplication gates indices 4 and 5 (from left to right).

When a gate has multiple outputs, like the input b, these wires count as only one.

In the example, this result is the assignment $(c_1, c_2, c_3, c_4, c_5)$ where $c_5 = (c_1 + c_2) \cdot c_4$ and $c_4 = c_2 \cdot c_3$. This is illustrated in Fig. 5.

The target polynomial is defined as $t(x) = \prod_g (x - r_g)$, where g are the multiplication gates.

Finally, the polynomials $\mathcal{V}$, $\mathcal{W}$, $\mathcal{Y}$ are defined by letting the polynomials in $\mathcal{V}$ encode the left (or top) input into each gate, the polynomials in $\mathcal{W}$ the right (or bottom) input into each gate and the polynomials in $\mathcal{Y}$ encode the output of each gate.

For example, $v_k(r_g) = 1$ if the k-th wire in a left input to gate g, and $v_k(r_g) = 0$ otherwise.

Similarly for w_k and y_k: $w_k(r_g) = 1$ if the k-th wire in a right input to gate g, and $w_k(r_g) = 0$ otherwise, and $y_k(r_g) = 1$ if the k-th wire in a output to gate g, and $y_k(r_g) = 0$ otherwise.

As above, this only applies to the multiplication gates. This results in the parameters in the table Fig. 8 for the example.

This way, the polynomials encode simply that the output value of any gate is equal to the product of its inputs, which is exactly the definition of a multiplication gate.

Notably, the polynomials of this construction are extremely sparse.

2.4.6 From QAPs to zk-SNARKs

The next step is to produce a very short proof, that proofs the knowledge of a satisfying assignment of the QAP.

For this step, the Pinocchio Protocol is used [Par+13]. Consider Alice as the prover:

If Alice has a satisfying assignment for the QAP, this means that with the polynomials defined from above, there exists a polynomial h such that $p(x) = h(x) \cdot t(x)$.

If she does not have a satisfying assignment, she could still construct the polynomials from above but with an unsatisfying assignment $(c_1, ..., c_m)$. In this case, $t(x)$ is guaranteed to not divide $p(x)$. This means that for any polynomial H of degree at most d, $p(x)$ and

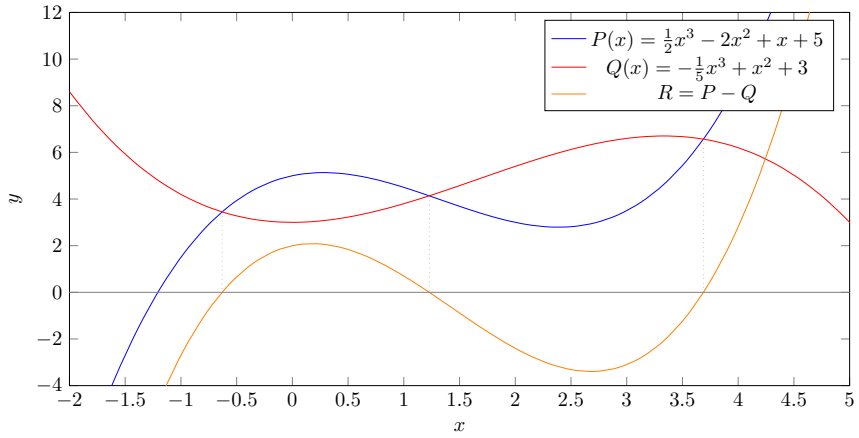

Figure 9: Illustration of the Schwartz-Zippel lemma. P and Q have degree 3. The roots of R are the only points where P and Q agree.

$h(x) \cdot t(x)$ will be different polynomials. Note that $p(x)$ and $h(x) \cdot t(x)$ here are both of degree at most $2d$.

According to the Schwartz-Zippel-Lemma, two different polynomials of degree at most $2d$ can agree on at most $2d$ points $s \in \mathbb{F}_p$. Thus, if p is much larger than $2d$ the probability that $p(s) = h(s) \cdot t(s)$ for a randomly chosen $s \in \mathbb{F}_p$ is very small.

Instead of sending the polynomials directly, they are encoded using a partly homomorphic encryption scheme. For efficiency, and because of some interesting additional properties, elliptic curves are used for this step, but this is beyond the scope of this paper. Further details can be found in [Par+13] and [BS+14].

2.4.7 Implementations

Several implementations of zk-SNARKs exist with Pinocchio [Par+13], libSNARK [Lib], snarklib [Sna] and Bellman [Bel]. They originate as a proof-of-concept of the research in this area, but zk-SNARKs are already put into practical use in cryptocurrencies, as in Zcash and Ethereum. In the next section, this paper looks at the use case of Zcash, which uses libSNARK for its zk-SNARKs.

2.5 Anonymization Algorithms

The previous sections covered verifiable computation in general and Non-interactive zero-knowledge proofs, with a focus on zk-SNARKs. These concepts enable the proof of correctness of a result of computation on *private input data*. In other words, with zk-SNARKs (and the like) it is possible to process sensitive data and later proof the correctness of results *without the need to reveal the sensitive input*.

The goal of the thesis is to apply this concept to an anonymization algorithm. Therefore, this section gives a brief overview of anonymization and introduces the notion of *k-anonymity*, a metric of anonymity which shall be checked with zk-snarks in the implementation part of the thesis.

Anonymization is the process of altering data sets such that the people described by the data sets remain anonymous. In other words, it shall not be possible to relate the data sets to individual people.

This can be achieved by either removing or altering attributes from the data set which may lead to the identification of persons. Anonymity is not the same as pseudonymity, where a unique identifier of the person is maintained without using the person's private data.

Personal Health Attributes are categorized under one of three categories: Direct Identifier attributes (DI), Quasi Identifier attributes (QI) and Sensitive Attributes[EP15]. Direct Identifier attributes can singularly identify an individual in the dataset, like social security numbers. QI attributes, when combined, also behave like direct identifiers, like first name, surname and date of birth. Sensitive attributes are not usually found in publicly available data and therefore present a low risk for re-identification by linking different datasets, but are sensitive if associated with an individual, like diagnosed diseases and drug prescriptions.

2.5.1 *k*-Anonymity

The k-anonymity property of a dataset ensures that each record in the dataset might relate to a number of at least k individuals. To achieve this, information of QI attributes is being compressed such that for each record, there are at least $k-1$ indistinguishable records in the dataset.

Introduced by Samarati and Sweeney[SS][Swe02] in 1998, k-anonymity is the most popular measure of anonymity. But in its simplest form, it is susceptible against a variety of de-anonymization attacks. Refinements like the additional measures of l-diversity and t-closeness[LLV04] improve the anonymization properties of k-anonymity.

In the domain of health data, k-anonymity has further limitations. As the nature of most health data is dynamic and multi-dimensional relational data, the concept cannot be applied[EP15]. But k-anonymity serves well as a simple example of an anonymization property and is suitable for static, non-changing, non-relational data, like for example historical records of completed treatments of patients on specific diseases.

3 Use Case and Requirements

The thesis goal is to demonstrate the applicability of verifiable computation techniques to generate proofs for anonymization algorithms. Considering the scenario from the introduction, there are three types of entities, as depicted in Fig. 10.

- **Data Providers** The hospitals generate sensitive data of their patients and the treatments. Each hospital generates similar type of data, but they cannot interchange the raw data directly due to privacy and data protection requirements. The Data Providers need to ensure compliance to data protection laws and, furthermore, are morally obliged to protect the private information of their clients. At the same time, the aggregation of their data would be benificial for research on diseases and treatments. Such data is valuable for both public and commercial researchers; it might therfore be possible to sell the data if there is a way to adhere to the forementioned restrictions.

  To be able to distribute their data, the data providers are interested in a way to proof that published data meets all requirements on data protection. Traditionally, this would not be possible without access to the original data, thus an aggregation of data from different providers before the anonymization would not be possible. Thus, a middleman is required; but as this middleman has financial incentives on selling data, an independent, strong proof of the correct handling of the data provides legal safety.

- **Third Party (data aggregation & reselling)** This is a middleman who receives raw or pseudonymised data from the data providers and aggregates this data. The

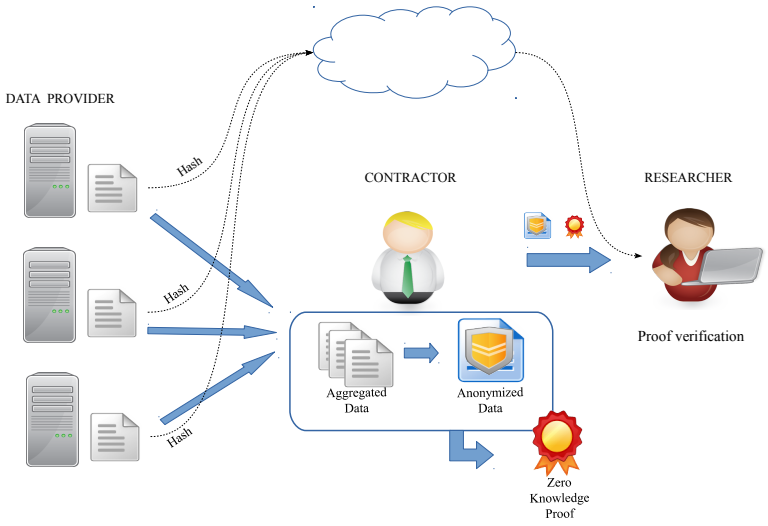

Figure 10: Use Case

middleman has to be obligated by contract to protect the data such that the middleman can work as a service on behalf of the data providers. The middleman can then anonymize the data, using anonymization techniques such as k-anonymity, l-diversity, and so on, to ensure compliance with moral and lawful requirements for data protection. The resulting data may then be distributed (and sold) to researchers in public and private sector.

- **Researchers** Pharmaceutical companies and health care researchers require genuine data on the success and failures of treatments, along with as much of possibly related attributes as possible. Due to privacy and data protection, they may only get anonymized data and cannot be allowed to see the original, raw data. Nevertheless, the genuinity and possibly additionally a measure of information loss during the anonymization process are important characteristics to prove the quality of their data, which directly influences the expressiveness and validity of their research results.

  A way to prove the source and quality of their base data is therefore of high interest for Researchers. In the case of commercial researchers, this is also important to fulfill and prove moral and lawful requirements on the accuracy of their studies.

The goal of the thesis is to construct a proof scheme that can be applied by a *data aggregator* with financial interests to prove to both the *data providers* and the *researchers*, that the sold data

- is result of a process of aggregation and anonymization,
- originates in the provided raw data of the *data providers*, and in nothing else,
- is based on a certain sample size,
- relates to a certain utility,
- have not been altered,
- fulfill lawful requirements,

- are correctly anonymized.

To achieve this goal, the use of a verifiable computation scheme with private inputs is required (See Section 2.1):

The aggregated data contains data from several data providers. It may neither be accessed by the researchers, nor by the data providers themselves, as they may only access their own data but not the other provider's data.

That is why the raw data has to be a private input to the verification scheme.

To be able to still tie the anonymized data to the original inputs and eliminating a need of trust in the middleman for this procedure, the hospitals / data providers need to be able to publish a commitment to their data independently from the middleman. This commitment then has to be included as a public parameter of the verifiable computation scheme.

There are at least two forms of this commitment:

- 1: Publish a hash of the data. The hash would also be a public input to the verifiable computation scheme. Pros: This can be easily (and efficiently!) included in zk-SNARKs with libSNARK, because zCash uses lots of hashes. Cons: Requires communication between researcher and publisher every time new datasets are acquired.
- 2: Publish a public key and sign the data with this key. The public key would also be a public input to the verifiable computation scheme. Pros: Public key only needs to be verified once and could even be distributed via a central list of hospital data keys, guarded and signed by a trustet governmental institution. Thus, the researcher would not need to communicate with the individual data providers. Cons: Data providers won't know who is doing research with their data. Probably also a significantly higher implementation effort for this; therefore out of scope for the thesis.

3.1 Verifiable Computing Schemes

Several verifiable computing schemes exists; an overview is given from Buchmann et al. in [Buc+16] from 2016. According to the structure of their work, verifiable computing schemes can be structured into proof-based verifiable computing, verifiable computing using fully homomorphic encryption, and verifiable computing from functional encryption and functional signatures.

The approach of this thesis requires the use of a non-interactive scheme. This is a special case of the first category of schemes, the proof based verifiable computing schemes. Using Quadratic Arithmetic Programs (QAPs), succinct non-interactive arguments of knowledge (SNARKs) can be constructed.

There are several published approaches to SNARKs, including Pinocchio [Par+13], Geppetto, SNARKs for C, SNARKs for a von Neumann Architecture [BS+14], ADSNARKs, Block Programs and zk-STARKs [BS+18]. Most of these are different realisations of the same fundamental concepts and based on QAPs. They have found their way into the implementation of libSNARK.

libSNARK [Lib] is a C++ library for zkSNARK proofs, developed by the SCIPR Lab project. The library is being actively developed and used in production for zCash. With its broad support for different approaches to the creation of SNARKs, it can be considered to be one of the most stable libraries for the handling of zk-SNARKs. I will therefore base my thesis on the usage of zk-SNARKs with libSNARK.

zk-STARKs are a new development of zero knowledge proving systems that aim to provide post-quantum security. In constrast to zk-SNARKs, they do not have a constant verification time, but their verification time scales exponentially faster than database size. This is a promising new system, but due to its recent publishing date, i expect the usage of this scheme to be much more difficult, as there is no library in a state that could compete with

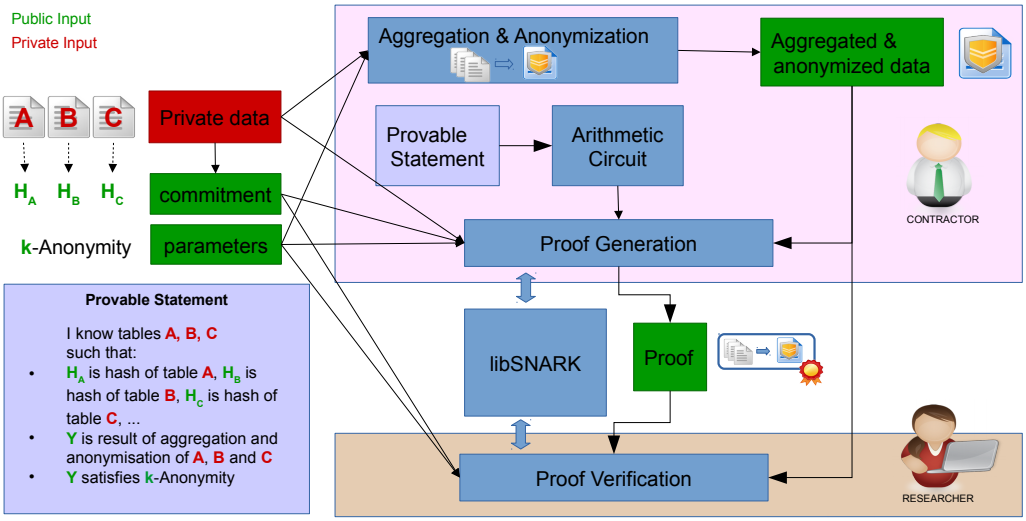

Figure 11: Conceptual approach

the years of existence of libSNARK. The realisation of the use case with zk-STARKs would be another practical use case for verifiable computing in the context of anonymization and can be realised in future work.

4 Conceptual approach

To demonstrate the applicability of zero-knowledge proofs to an anonymisation algorithms, i will construct such a proof for the work of an aggregating and anonymizing third party as explained in the use case above (3).

Data from multiple providers, in 11 depicted as A, B and C, is first aggregated and anonymized by the third party (c̈ontractorïn 11). The contractor additionally computes a proof for a provable statement. This statement must include everything needed to convince both the data providers and the researchers of the correctness of the contractor's work. It therefore has to prove that the result satisfies k-anonymity and is based exactly on the aggregation of the data from the providers.

For the check of the second property, the researchers need to be able to link the proof to the original data. Due to the fundamental privacy constraints, they must not receive the original data, but they can receive a fingerprint of the data from the data providers, e.g. in form of a Hash. Such a fingerprint uniquely identifies the dataset without revealing its content and can therefore be given to the researchers without compromising privacy of the data included.

The provable statement, to be computed by the contractor, then looks like the following:

Let n describe the number of data sources. Let S_1, ..., S_n be the data tables from the data sources. Let Y be the result of aggregation and anonymization.

I know tables $S_1, S_2, ..., S_n$ such that:

- For each i in $1...n$:
  - H_i is hash of table S_i
- Each row in the sources corresponds to exactly one row in Y and vice versa
- Y satisfies k-Anonymity

From the statement, an arithmetic circuit can be constructed. This circuit will then be used to generate a proof of the above statement. This proof can then be distributed along with anonymised data.

The researchers can then receive the fingerprints from the data providers and verify the proof.

5 Realisation / Implementation

I want do provide a basic interface to the user. The interface is probably going to be a REST-based interface to a django webserver, but might as well be realised with something simpler. With this interface, it shall be possible to upload data tables as comma separated values (CSV) from different providers, check that their columns match and then trigger the process of aggregation, anonymization, proof generation and verification.

Django is a web development framework in Python. Within django, the actual implementation will be called. The core implementation will use libSNARK, a C/C++ library for zkSNARKs.

If suitable, i will implement the circuit for the proving statement with ZoKrates, a domain specific language for circuit generation for zkSNARKs. If not, this part will be done either in python or in cpp; the first week in the schedule (familiarizing myself with libSNARK and circuit generation) shall answer which of these is the better suitable option.

One part of the process that is important for zkSNARKs but out of scope for the bachelor thesis is the generation of public parameters. For a secure setup, this parameter generation has to be realized with a trusted setup. This is tricky, but already part of other research[BGG17] and can be answered independently from the thesis. The thesis will use the builtin-functions of libSNARK for parameter generation and assume that it is possible to replace this step with a trusted setup.

To facilitate the setup of the runtime-environment of the implementation for evaluation or enhancement purposes for future research, my implementation will be realized within one or several docker containers, such that it is easy to reconstruct my setup anywhere. The docker containers will be based on Ubuntu or Alpine Linux.

6 Evaluation

The thesis goal is reached when a functional proof-of-concept implementation of a system can be shown that enables proof generation and verification, using zkSNARKs, for the aggregation and anonymisation of multiple aggregatable datasets.

If this goal is not reachable, instead an analysis of the reason should be given.

If the goal is reachable, evaluation is simple, as a working implementation will provide the proof-of-concept this thesis aims for.

Additionally, the efficiency of the system will analyzed. There are several factors which might have a significant influence on the runtime of proof generation (proof validation is expected to be in constant time): the number of data providers, the parameter k of the k-anonimity-property, the number of records in the aggregated dataset, and the number of attributes in the dataset.

I plan to include a benchmark of at least two of these parameters in the thesis: the number of data providers and the number of records in the aggregated dataset. The benchmark can then support a statement on whether this kind of proof system is ready for real-world-applications and on possible limitations of my implementation and/or of the concept in general.

For the benchmarking, i am going to use the example dataset (or a subset from it) from the ARX Anonymization Tool [Arx]. In case this data proves to not be sufficient for the evaluation, it is also possible to generate a dataset with the required properties (e.g. number of attributes); there is no need for the evaluation of the proof-of-concept to operate on real datasets. In any case, the datasets used for evaluation will stored together with the implementation such that an evaluation of the work will be possible.

The thesis will not use actual medical datasets, as they are not publicly available in de-anonymised form.

7 Project plan

The first two weeks are dedicated to familiarize myself with libSNARK and circuit implementation and with the implementation and check of k-anonymity. The main implementation phase comes next, followed by the first writing phase. Afterwards, benchmarking and a second writing phase are planned.

- **Create & run simple circuits (1 week):** Familiarize with circuit creation and libSNARK.
- **Christmas holidays (2 weeks)**
- **k-Anonymity property check and software design (1 week):** Implement a check for k-Anonymity and tests, and define interfaces, data flow and interaction model for the software. Decide on a library-provided k-anonymity check.
- **Circuit implementation and basic user interface (4 weeks):** Create a circuit for the k-anonymity-property and an basic user interface that enables experimentation with different datasets
- **Writing (1 week):** First writing phase
- **Benchmarking (2 weeks)**
- **Writing (3 weeks)**
- **Buffer (3 weeks)**

References

[Arx] *ARX - Data Anonymization Tool*. https://arx.deidentifier.org/downloads/. accessed: 2019-01-11.

[Bel] *Bellmann (zk-SNARKs in Rust) on GitHub*. https://github.com/ebfull/bellman. accessed: 2018-04-22.

[BS+18] Eli Ben-Sasson et al. *Scalable, transparent, and post-quantum secure computational integrity*. 2018.

[BS+14] Eli Ben-Sasson et al. "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture." In: *USENIX Security Symposium*. 2014, pp. 781–796.

[Bit+12] Nir Bitansky et al. "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*. ACM. 2012, pp. 326–349.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. "Non-interactive zero-knowledge and its applications". In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. Ed. by J'anos Simon. New York, NY: ACM, 1988, pp. 103–112. ISBN: 0897912640. DOI: 10.1145/62212.62222.

[BGG17] Sean Bowe, Ariel Gabizon, and Matthew D Green. *A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK*. Tech. rep. TR 2017/602, IACR, 2017.

[Buc+16] Johannes Buchmann et al. "Overview of Verifiable Computing Techniques Providing Private and Public Verification". In: (2016).

[Ehe] *E-Health-Gesetz – neue Anwendungen für Ärzte und Versicherte kommen*. https://www.bundesaerztekammer.de/aerzte/telematiktelemedizin/earztausweis/e-health-gesetz/. accessed: 2019-01-11.

[EP15] Benjamin Eze and Liam Peyton. "Systematic Literature Review on the Anonymization of High Dimensional Streaming Datasets for Health Data Sharing". In: *Procedia Computer Science* 63 (2015), pp. 348–355. ISSN: 18770509. DOI: 10.1016/j.procs.2015.08.353.

[FS87] Amos Fiat and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. Ed. by Andrew M. Odlyzko. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194. ISBN: 978-3-540-47721-1.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. "Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers". In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Berlin, Heidelberg: Springer, 2010, pp. 465–482. ISBN: 978-3-642-14623-7.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. "The knowledge complexity of interactive proof-systems". In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. Ed. by Robert Sedgewick. New York, NY: ACM, 1985, pp. 291–304. ISBN: 0897911512. DOI: 10.1145/22145.22178.

[Zca] *How Zcash works. What are zk-SNARKs?* https://z.cash/technology/zksnarks.html. accessed: 2018-04-23.

[LLV04] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity". In: *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 15.04.2006 - 20.04.2007, pp. 106–115. ISBN: 1-4244-0802-4. DOI: 10.1109/ICDE.2007.367856.

[Lib] *libSNARK on Github*. https://github.com/scipr-lab/libsnark. accessed: 2018-04-22.

[Par+13] Bryan Parno et al. "Pinocchio: Nearly practical verifiable computation". In: *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE. 2013, pp. 238–252.

[Ros] Ameer Rosic. *What is zkSNARKs: Spooky Moon Math*. https://blockgeeks.com/guides/what-is-zksnarks/. accessed: 2018-04-23.

[SS] Pierangela Samarati and Latanya Sweeney. "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression". In: *Proc IEEE 1998 Symp Res Secur Priv*, pp. 384–393.

[Sna] *snarklib (a libsnark reimplementation) on GitHub*. https://github.com/jancarlsson/snarklib. accessed: 2018-04-22.

[Swe02] Latanya Sweeney. "k-Anonymity: A Model for Protecting Privacy". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.5 (2002), pp. 557–570. DOI: 10.1142/S0218488502001648. URL: https://doi.org/10.1142/S0218488502001648.

[Ung] Walter Unger. *Algorithmische Kryptographie*. Lecture at RWTH Aachen University in winter term of 2017/18.

[Wal] Michael Walfish. *Introduction and Overview of Verifiable Computation*. https://www.youtube.com/watch?v=qiusq9R8Wws. Accessed: 2018-04-23.