Universiteit Leiden
Computer Science

Efficient Secure Regression Protocols
to predict growth curves
of children

Name:	L.A.J. van der Beek
Date:	14/11/2017
1st supervisor:	Dr. C.J. Veenman
2nd supervisor:	Prof. dr. ir. W. Kraaij
ext. supervisor:	Dr. ir. P.J.M. Veugen (TNO)

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Predicting growth curves of children is usually done by child consultation clinics, who base their prediction on the whole population, to provide parents with a general prediction of how their child will develop. However, when only taking into account data of similar children when predicting the growth curve of a specific child, a more accurate prediction can be given. Since growth data of children is considered privacy-sensitive, we obtain such a prediction using secure regression.

We looked at four different protocols for obtaining such predictions, three of which we implemented ourselves. Some protocols make use of garbled circuits and some protocols make use of homomorphic encryption. We compared them based on their speed, accuracy and degree of security, so we could find the most efficient protocol that can predict growth curves based on growth data of children. In order to improve the accuracy of the predictions, we also considered four different regression methods in our protocols.

After performing some experiments, we came to the conclusion that a garbled circuits method with the point-and-permute technique, free XOR-gate and garbled row reduction adaptations, and with an adaptation that all multiplexer gates are evaluated through use of a boolean circuit, is the best protocol to use in this setting.

Contents

1 Introduction 1

1.1 Secure Regression . . . 1

1.1.1 Homomorphic Encryption . . . 3

1.1.2 Garbled Circuits . . . 3

1.2 The SMOCC dataset . . . 4

1.3 Problem Description . . . 4

1.4 Objective . . . 5

1.5 Overview . . . 6

2 Previous Work 7

2.1 General Work . . . 7

2.2 Methods Used . . . 8

2.2.1 Curve-Matching . . . 9

2.2.2 Garbled Circuits General Method . . . 9

2.2.3 Garbled Circuits Adaptations . . . 10

2.2.4 Oblivious Transfer . . . 12

2.2.5 Homomorphic Cryptosystems . . . 13

2.2.6 The ABY Framework . . . 15

2.3 Contributions of this Work . . . 16

3 Protocol Description 18

3.1 General Setting . . . 18

3.2 Original Protocol . . . 20

3.3 Garbled Circuits Protocol . . . 24

3.4 Alternative Homomorphic Encryption Protocol . . . 32

3.5 Modified Garbled Circuits Protocol . . . 35

4 Results 38

4.1 Comparison of Speed . . . 38

4.2 Comparison of Accuracy . . . 40

4.3 Comparison of Security . . . 44

5 Conclusions 49

5.1 Preferred Protocol . . . 49

5.2 Future Work . . . 50

6 References 52

7 Appendix 55

7.1 Regression . . . 55

7.1.1 General terms 55
7.1.2 Linear Regression 56
7.1.3 Piecewise Linear Regression 57
7.1.4 Slope-based regression 58
7.2 Arithmetic shares 58
7.3 Simple Garbled Circuits Protocol 59
7.4 Slope-based Garbled Circuits Protocol 66
7.5 Linear Garbled Circuits Protocol 67
7.6 Modified Simple Garbled Circuits Protocol 69
7.7 Modified Slope-based Garbled Circuits Protocol 70
7.8 Modified Linear Garbled Circuits Protocol 70
7.9 Alternative Simple Homomorphic Encryption Protocol 71
7.10 Alternative Linear Homomorphic Encryption Protocol 76
7.11 Alternative Slope-based Homomorphic Encryption Protocol . . 78
7.12 Specifications Hard- and Software 78

1 Introduction

Security keeps becoming more and more important in the current age, considering the amounts of privacy-sensitive data that are generated and stored. In order for these data to be more useful, institutions that possess such data want to perform analyses, such as regression, on it. Ideally, they would like to perform those analyses on as much data as possible, not just on their own data. However, since these data are privacy-sensitive, they don't want to share their data with different institutions, and vice-versa. An example of such a situation is determining the influence of a certain drug on the development of a disease, when the drug is used by multiple hospitals (especially for rare diseases). Some other examples of situations where this is the case are given in [28].

Another example, at which we will look extensively in this thesis, is the prediction of growth curves of young children. There, parents of young children want to know how their child is expected to grow, without sharing the growth data of their children.

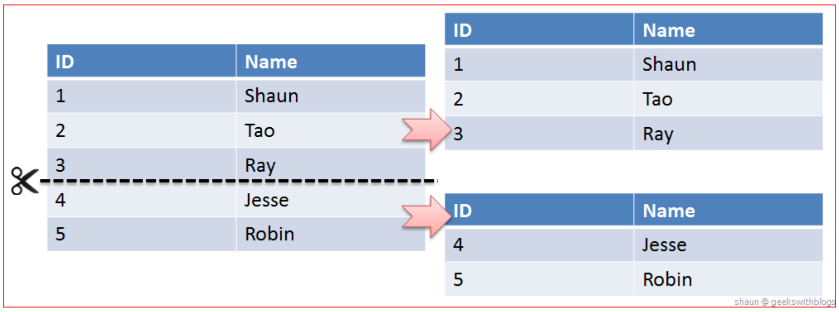
In general, to enable institutions to perform analyses, such as the ones we mentioned, on large amounts of privacy-sensitive data, which come from different owners, a secure protocol is needed, which does not leak any data from one institute to another, but does provide each institute with the result (for example, a predicted growth curve for the child) of the analysis in the end. There exists a collection of such secure protocols for regression analysis, which is known as "secure regression" [15]. In the next sections, we will look briefly at some basic concepts of secure protocols and at what we aim to achieve in this thesis.

1.1 Secure Regression

There are a few different settings in which secure regression is discussed. Firstly, we can look at different kinds of regression (for example, linear regression, which is discussed along with the other types of regression that we use in Section 7.1, ridge regression [14] and LASSO [26]), which each can be expressed differently in a secure protocol. In our setting, we will only look at (variants of) linear regression.

Secondly, the partitioning of the data can differ: we either have data that is horizontally partitioned [24], which means that different parties have data about different entities with the same attributes (for example, patient records from different hospitals), or we have data that is vertically partitioned [24], which means that different parties have data about the same entities with different attributes (for example, different government agencies that have different kinds of information, such as employment information, education information or health information, about the same individuals). We give an example of a horizontally partitioned database in Figure 1 and an example of a vertically partitioned database in Figure 2. The partitioning of the database influences the way in which a secure protocol is designed.

Finally, there are two main methods that are commonly used to achieve the security in these secure regression protocols, namely homomorphic encryption [12] and garbled circuits, also known as Yao's circuits [32].

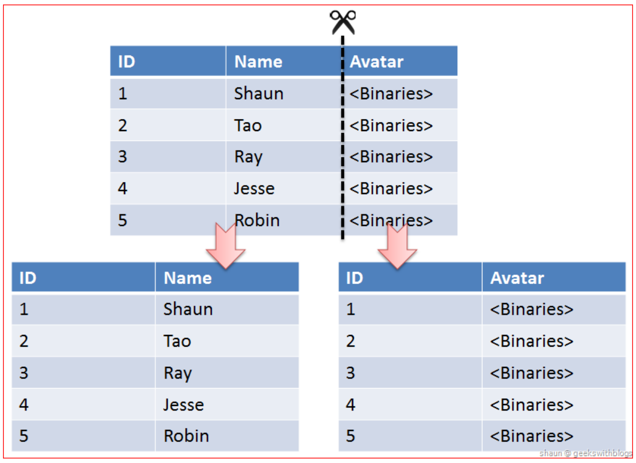


ID	Name
1	Shaun
2	Tao
3	Ray
4	Jesse
5	Robin

ID	Name
1	Shaun
2	Tao
3	Ray

ID	Name
4	Jesse
5	Robin

Figure 1: An example of a horizontally partitioned database



ID	Name	Avatar
1	Shaun	<Binaries>
2	Tao	<Binaries>
3	Ray	<Binaries>
4	Jesse	<Binaries>
5	Robin	<Binaries>

ID	Name
1	Shaun
2	Tao
3	Ray
4	Jesse
5	Robin

ID	Avatar
1	<Binaries>
2	<Binaries>
3	<Binaries>
4	<Binaries>
5	<Binaries>

Figure 2: An example of a vertically partitioned database

1.1.1 Homomorphic Encryption

Homomorphic encryption is a special kind of encryption, which enables the execution of certain mathematical operations on encrypted data in such a way that when the result is decrypted, it is the same as when the mathematical operation would have been performed on the unencrypted data. For example, in additively homomorphic encryption, two original data samples can be added in encrypted form, and when the result is decrypted, the result will be equal to the sum of the original data samples. In addition to additively homomorphic encryption, there also exists multiplicatively homomorphic encryption, and fully homomorphic encryption [3], the latter allows for arbitrary computations on encrypted data (that will match the result of the same computations performed on the unencrypted data when decrypted). Fully homomorphic encryption does cost a significant amount of time complexity to execute, which causes it to be too inefficient to apply to larger datasets (at the moment). Additively homomorphic encryption and multiplicatively homomorphic encryption are called “partially homomorphic cryptosystems”. There exist different kinds of partially homomorphic cryptosystems, for example, the Paillier cryptosystem [21] and the ELGamal cryptosystem [8], and different kinds of fully homomorphic cryptosystems, for example, Gentry’s cryptosystem [10] and Brakerski’s scale-invariant cryptosystem [5]. These cryptosystems differ in the specific encryption that is used and in the types of computations they support on encrypted data (which depends on the type of cryptosystem that is implemented).

1.1.2 Garbled Circuits

Garbled circuits are a cryptographic protocol, which mainly makes use of permuted logic gates, and which is often used to perform secure computations between multiple parties, such as secure regression. We will describe how they work later, in Section 2.2.2. Garbled circuits are not usable as a kind of homomorphic encryption, since it is not possible to perform a mathematical operation on the encrypted data within the garbled circuits, decrypt the data, and then have the same result as when you would have performed the mathematical operation on the unencrypted data. Instead, garbled circuits enable one to convert a boolean circuit, which performs some computation, to a garbled version of that same circuit, which performs the same computation within the encrypted domain given encrypted inputs. The types of computations that can be performed are limited by what types of computations can be represented in a boolean circuit, and by the types of boolean gates that a certain garbled circuits method can convert.

1.2 The SMOCC dataset

In order to be able to compare the performance of different secure regression protocols, we used the Social Medical survey Of children attending Child health Clinics (SMOCC) [13] dataset to test these protocols. The SMOCC dataset consists of health data of children, such as gender, weight at birth and weight after a certain number of months, from their pre- and perinatal period up to their second birthday. Some data of the mothers of these children is also taken into account, such as their age at delivery and their level of education. These data were collected by 21 child health clinics in the Netherlands.

1.3 Problem Description

In this thesis, we want to predict growth patterns for children, of whom we have no data in the SMOCC dataset, based on the data within this dataset. In general, the parents of these children don't want to reveal the data of their children, therefore a secure protocol is used to predict the growth patterns for the children. A global outline of the scenario within this kind of protocol is given in Figure 3. In most cases, the prediction of the growth patterns will be done with a secure regression protocol, which makes sure that the party comparing the data can't access these data. The goal is to find the best protocol out of some existing and some newly introduced secure protocols.

We can see that we are looking at a horizontally partitioned problem, since we have different parties (the parents and the party managing the SMOCC dataset) that each have their own set of data on disjoint individuals, with the same attributes. We will assume that the parties are both semi-honest, which means that they do follow the protocol correctly, but may try to derive additional information from the messages they receive. Since the used data is longitudinal, which means that there are repeated measurements over time of the same subjects (in this case, children), we have to take this into account in our analysis. Therefore, we average the measurements for the considered subjects out of the SMOCC dataset at each time step in all our analyses, in order to get an accurate prediction. A complete description of our analyses is included in Chapter 3.

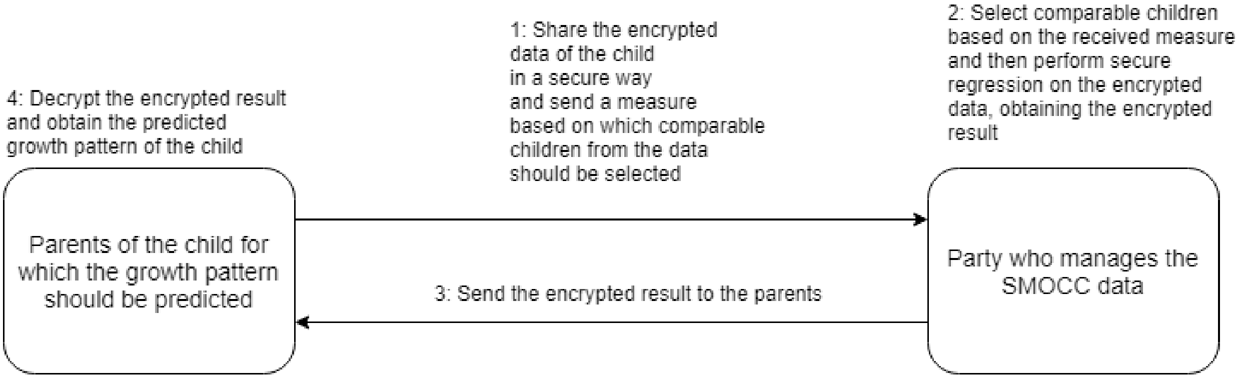

4: Decrypt the encrypted result and obtain the predicted growth pattern of the child
1: Share the encrypted data of the child in a secure way and send a measure based on which comparable children from the data should be selected
2: Select comparable children based on the received measure and then perform secure regression on the encrypted data, obtaining the encrypted result
Parents of the child for which the growth pattern should be predicted
Party who manages the SMOCC data
3: Send the encrypted result to the parents

Figure 3: A global outline of the scenario within the protocols

To reach our goal, we do the following: first, we will have to look for protocols with which we can predict the growth curve of an individual in a secure way (Chapter 3), and compare them based on their degree of security and speed (Chapter 4). Thereafter, we will have to look at the problem of how to get the most accurate prediction of a growth curve of a child outside of the SMOCC dataset with the help of data from the SMOCC dataset (Chapter 3).

1.4 Objective

Given the problem sketched in the previous section, we want to compare some protocols based on garbled circuits to some (partly existing) protocols based on homomorphic encryption. We expect that the protocols based on garbled circuits will yield a faster protocol, which makes use of fewer external semi-trusted parties than the protocols based on homomorphic encryption. This conjecture stems from the fact that the protocol based on homomorphic encryption makes use of two external semi-trusted parties, and from the fact that the largest part of the time complexity of a garbled circuits protocol is in the preprocessing, while for a homomorphic encryption protocol this lies within the communication and computation of complex operations [20]. In this thesis, we test experimentally whether this conjecture is true, in addition to trying to find the most accurate model for the protocols to use in the prediction of the growth curve.

1.5 Overview

In the next Chapter, we will look at previous work that was done in the area of secure regression, and at what this thesis will add to that. In Chapter 3, the new protocols we developed will be described in detail, and some existing protocols, to which we will compare the new protocols later, will be described as well. Chapter 4 will describe the results of the experiments we performed on the SMOCC dataset, and in Chapter 5 we will draw our conclusions. Chapter 6 will consist of the references, and finally Chapter 7 contains the appendices, such as the source code of our new protocols.

2 Previous Work

In this chapter, we will present some work that has been done on the topic of secure regression, and some work that has been done in relation to the SMOCC dataset. Thereafter, we will mention the aspects in which this thesis is innovative.

2.1 General Work

First and foremost, we will mention a study on which the protocols in this thesis are mainly based, namely the PRANA-DATA project [29], in which the setting was (almost) the same as in this project: based on the data from the SMOCC dataset, growth patterns of individuals outside of the dataset were predicted. The largest difference with respect to this project is that in the PRANA-DATA project, no (secure) regression was performed in order to predict the growth patterns, but a recently developed method called "curve matching" [27]. This method was specifically developed to help with the interpretation and prediction of growth curves of individuals. We provide a short example of curve matching here: given some features of the first 6 months of growth of a child outside of the dataset (for example, length and weight over the first 6 months), search in a secure way for a fixed number of children within the dataset that have similar features in their first 6 months of growth. Then, we can predict the growth after 6 months of the child outside of the dataset based on the data of the children with similar features, whose information about their growth after 6 months is included in the dataset.

Considering work that has been done in the area of secure regression, we give a short overview of the development in this area, referring to appropriate papers in the process. One of the first occurrences of a technique related to secure regression (although this was not published in the context of secure regression), was in [31]. In this article, the goal is to solve the so called "millionaires' problem": there are two millionaires who want to know who is richer without revealing additional information about either person's wealth to the other. A protocol is presented in order to achieve this, which is one of the first secure protocols to compute sums and products, and this lies in the foundation of garbled circuits. Somewhat later, the first version of what later would be called the garbled circuits protocol was presented in [32].

Regarding homomorphic encryption, one of the first articles mentioning something related to this was [23]. In this article, a problem is presented for which a few different solutions are given, each having their own downside. As a final solution, so-called ”privacy homomorphisms” are presented, which are the first forms of homomorphic encryption. The application of the privacy homomorphisms presented in this article was limited, but over time, more and more homomorphic cryptosystems were presented, which also became applicable in more cases.

For a detailed overview of the development of the (fully) homomorphic encryption schemes, we refer to [2].

An interesting method that makes use of both garbled circuits and homomorphic encryption in different parts of the protocol, in order to achieve more speed, is presented in [20].

The basic idea is that the protocol is divided in two phases: in the first phase, only additively homomorphic encryption is used (in this article Paillier is used) in order to encrypt the data in such a way that the amount of encrypted data is independent of the number of users, and to do the linear operations required for the regression on the encrypted data. This is done with additively homomorphic encryption because it can handle linear operations, such as addition and multiplication, on large datasets faster than garbled circuits.

In the second phase, garbled circuits are applied to do some heavy non-linear computations that are required for the regression, since this would be a lot slower, if homomorphic encryption had been used. The resulting protocol is quite fast, but this speed comes at a price, since two external parties (parties that are not providing any data in the protocol and don’t obtain a plain result in the protocol), which are never allowed to cooperate according to the protocol, are needed (if they would cooperate, the protocol wouldn’t be secure). This is a questionable assumption, since as it stands it already is hard to find one semi-trusted party in practice, let alone two. Therefore, we will take into account when evaluating a protocol whether it uses an external semi-trusted party or not. Preferably, we want to avoid using such parties in the protocols we use.

2.2 Methods Used

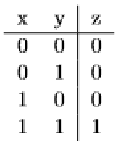
We will now highlight a few articles that present methods on which we base the protocols that we performed in this thesis. We have already discussed a few of these methods to some degree, namely the curve-matching, the garbled circuits and the homomorphic encryption. We will look at some of them in more detail, in order to clarify the specific implementation of those methods in this thesis. We will also mention the framework we used to implement our protocols.

2.2.1 Curve-Matching

Regarding the curve-matching, we referred to Stef van Buuren's article on the matter [27] for a description of the standard curve-matching method. However, in the protocol we will use in this thesis, as presented in [29], a slightly different assumption regarding the obtaining of matches is used. In the description of the standard curve-matching method, the goal is to predict a single value to use as a comparison metric for all children (the length of a child that currently is three months old, is predicted for when that child has reached the age of fourteen months), based on multiple values (the length of the child at birth and at the age of one, two and three months respectively, and some covariates, such as the sex of the child and the height of the parents). This is then achieved by making a linear regression model out of this information, and predicting the length of every child in the dataset at the age of fourteen months based on this regression model. In the description of the protocol that we will use in this thesis, we assume that this comparison metric is already known for all children and is also included in the database. This saves computation time when trying to obtain matches in the protocol.

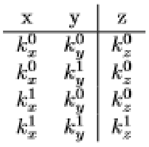
2.2.2 Garbled Circuits General Method

Within the garbled circuits protocol, we have two different parties: the garbler, who can generate a garbled gate [25] and send it to the other party, which is the evaluator, who can evaluate this garbled gate. By a garbled gate we mean a certain logic gate (for example, an AND-gate), from which the inputs of the truth table are obscured and the outputs of the truth table are encrypted. For example, for an AND-gate, we can see the normal representation of its truth table in Figure 4, and the garbled representation of its truth table in Figure 5. In the garbled representation, the subscripts of k represent the wire (also known as input variable) it corresponds to, and the superscript describes the semantic value of the wire, corresponding to the original AND-gate. Usually, the rows of a truth table of a garbled gate are randomly permuted before they are sent to the evaluator, in order to prevent the evaluator from learning too much about the gate from the order of the cipher texts. Also, we assume that the evaluator only knows two of the input keys, so he can only decrypt one output of the garbled table, as intended. Using these garbled gates, a garbled circuit can be constructed by combining gates, using the output of some gates as the input of other gates. Depending on the functionality that you want the circuit to have, a combination of gates is selected, so that you get the desired result when executing the garbled circuit. For a more detailed description of garbled circuits along with some more advanced techniques and applications of garbled circuits, the reader is encouraged to take a look at [30].



x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Figure 4: The truth table of a normal AND-gate



x	y	z
k_x^0	k_y^0	k_z^0
k_x^0	k_y^1	k_z^0
k_x^1	k_y^0	k_z^0
k_x^1	k_y^1	k_z^1

Figure 5: The truth table of a garbled AND-gate

2.2.3 Garbled Circuits Adaptations

Considering the garbled circuits method we used in this thesis, in addition to the standard garbled circuits method as described in Section 2.2.2, we mention a few adaptations that improve the efficiency of the garbled circuits that we will use.

Firstly, we use the point-and-permute technique, which was proposed in [4]. This method enables the evaluator to only have to decrypt one of the cipher texts instead of all four of them, despite the fact that the cipher texts are garbled. It achieves this by generating two select bits for each wire, k^0 and k^1. For $i \in \{0,1\}$, k^i represents $i \bigoplus r$, where $r \in \{0,1\}$ is a bit chosen at random by the garbler. In this way, k^i is different for each i and does not uncover any information about i to the evaluator. The select bit k for every wire is obtained together with the wire label, through oblivious transfer if the wire is an input wire, or through decryption otherwise. Then, the evaluator is able to use the input bits corresponding to the input wires when he evaluates a gate, to find out which cipher text he should decrypt. Therefore, using this method reduces the number of decryptions you have to try by a factor four.

Secondly, we apply the free-XOR technique as described in [17], which causes the computation of XOR-gates to be free (in terms of computation and communication required for creation, transfer and evaluation of the garbled tables). This is achieved by making a restrictive relationship between the input wires of a XOR gate: let the input wires be W_a and W_b, and let the output wire be W_c. Then the wire values are garbled as follows: the garbler chooses a random $R \in \{0, 1\}$, and picks each label w_i^0 individually at random. Thereafter, he combines both to obtain $w_i^1 = w_i^0 \bigoplus R$. Finally, we set $w_a^0 \bigoplus w_b^0 = w_c^0$. This results in a system where the garbled gate outputs can be obtained by XOR-ing the garbled gate inputs:

$w_a^0 \bigoplus w_b^0 = w_c^0$,
$w_a^1 \bigoplus w_b^0 = (w_a^0 \bigoplus R) \bigoplus w_b^0 = w_c^0 \bigoplus R = w_c^1$,
$w_a^0 \bigoplus w_b^1 = w_a^0 \bigoplus (w_b^0 \bigoplus R) = w_c^0 \bigoplus R = w_c^1$ and
$w_a^1 \bigoplus w_b^1 = (w_a^0 \bigoplus R) \bigoplus (w_b^0 \bigoplus R) = w_a^0 \bigoplus w_b^0 = w_c^0$.

Since this principle works for all XOR gates, their computation becomes free.

Finally, we incorporate the concept of garbled row reduction, as described in [19]. This should not be confused with the garbled row reduction introduced in [22], which reduces more rows, but is incompatible with the free-XOR technique. The garbled row reduction we use eliminates one cipher text for each gate, which results in gates with three cipher texts instead of four. This is achieved by picking a wire label in such a way that its corresponding cipher text is 0. Because of the select bits, this will always be the top cipher text (the select bits are the least significant bits of the cipher text).

The garbled row reduction may sound conflicting with the point-and-permute technique, however, since we know that the garbled row reduction method still includes a wire label for the eliminated cipher text, we can still obtain a wire label and a select bit for every wire for all four cipher texts corresponding to a gate. The only difference to when we only use the point-and-permute technique is that for one cipher text, the mapping of the select bits is changed such that they map to zero instead of to the original cipher text. Since we know which cipher text corresponds to zero because of the wire label, we can combine the garbled row reduction and the point-and-permute technique to reduce the amount of computation that has to be performed in a garbled circuits protocol.

2.2.4 Oblivious Transfer

In addition to these techniques which improve the performance of the garbled circuits, we discuss a protocol which is essential to be able to implement garbled circuits: Oblivious Transfer (OT), which is mentioned in [9]. This protocol consists of two parties, the sender (A) and the receiver (B), where the sender wants to send one out of two values to the receiver, which the receiver can pick beforehand. However, the sender should not be able to learn which value was learned by the receiver, and the receiver should not be able to learn anything about the other value that the sender has. Intuitively, this might seem strange, but this can be achieved by using the protocol that is described below in Protocol 2.1.

Protocol 2.1 The Oblivous Transfer protocol

Party	**Input**	**Output**
A	m_0, m_1, d, e, x_0, x_1	-
B	b, k	m_b

1. A has two secret messages, m_0, m_1, she generates two random messages x_0, x_1, and she generates a RSA key pair $(d, (e, N))$ in the following way: she chooses two distinct large prime numbers, p and q, multiplying them to obtain $N = p * q$, then chooses an integer e with $1 < e < \phi(N)$, where e is relatively prime to $\phi(N) = (p-1) * (q-1)$. Finally, she calculates d such that $e * d \equiv 1 \bmod (p-1) * (q-1)$. Then, she sends N, e, x_0 and x_1 to B.
2. B then chooses a $b \in \{0, 1\}$ and generates a random message k. Using these values in addition to the received values from A, B computes $v = (x_b + k^e) \bmod N$, where k^e represents the encryption of k, and sends v to A.
3. A then decrypts the following values using d: $k_0 = (v - x_0)^d$ and $k_1 = (v - x_1)^d$. One of these is equal to k, but A does not know which one. A then constructs two messages: $m'_0 = m_0 + k_0$ and $m'_1 = m_1 + k_1$, and sends them to B.
4. B now computes $m_b = m'_b - k$, and thus obtained a message without A knowing which one he received, and without learning anything about the other message, since B doesn't know k_{1-b}.

2.2.5 Homomorphic Cryptosystems

Looking at the homomorphic encryption systems we employed, we only use partially homomorphic encryption in this thesis. We discuss two different cryptosystems here: the Paillier cryptosystem and the Damgård, Geisler and Krøigaard (DGK) cryptosystem. It should be noted that some knowledge of group theory is needed to completely understand the descriptions of these cryptosystems, however, since this knowledge is not needed to understand the work we've done, we encourage the interested reader to read [16] for a description of the basics of group theory.

The Paillier cryptosystem, as described in [21], is an additively homomorphic cryptosystem which makes use of multiplication modulo the square of the product of two large prime numbers. If we set the product of the two prime numbers (public key) to be n and the base to g, then a message m with $0 \leq m < n$ is encrypted with this system as $E(m) = g^m r^n \pmod{n^2}$, with r a uniformly random number in $\mathbb{Z}_{n^2}^*$ (this denotes the multiplicative group of integers modulo n^2). The homomorphic property of this system is the following: $E(m_1) \cdot E(m_2) = (g^{m_1} r_1^n)(g^{m_2} r_2^n) \pmod{n^2} = g^{m_1+m_2}(r_1 r_2)^n \pmod{n^2} = E(m_1 + m_2)$. So by multiplying encrypted messages, we obtain the encryption of the sum of the original messages.

The DGK cryptosystem, which was proposed in [6], is also an additively homomorphic cryptosystem. We describe this cryptosystem following the explanation found in [18]. First and foremost, the DGK cryptosystem is based on the Strong RSA Subgroup Assumption, which is the following assumption:

Let K be a key generation algorithm that produces a "RSA subgroup pair" (N, g). The Strong RSA Subgroup Assumption for this key algorithm is that it is infeasible to find $u, w \in \mathbb{Z}_N^*$ and $d, e > 1$ such that $g \equiv uw^e \pmod{N}$ and $u^d \equiv 1 \pmod{N}$

The DGK cryptosystem uses two of these subgroups, with one of the subgroups being contained in the other one. This causes the message space to become smaller; in the original paper the message space is reduced to only 16 bits. In order to generate keys, the DGK cryptosystem needs three parameters, a, b, c with $a > b > c$. Here, a is the size of the RSA modulus, N, in bits, b is the size in bits of two small primes s_p and s_q, and c represents the size of the message space in bits. Key generation can then be done within the DGK cryptosystem as follows: We construct two b-bit primes s_p and s_q, and two distinct primes p and q of the same bit length such that $s_p | p-1$ and $s_q | q-1$. Then, we choose a c-bit prime u and an element $g \in \mathbb{Z}_N^*$ with order $us_p s_q$ and choose h to have order $s_p s_q$. We find these g and h by making use of a subgroup of hidden order of $\mathbb{Z}_N^*$ [11]. The public key is then (N, g, h, u) and the private key is (p, q, s_p, s_q). Also, an auxiliary table is generated in which tuples $(g^{s_p s_q})^i$ for $0 \leq i \leq u$ along with i itself are stored.

The encryption is then performed as follows: a random value r is generated (for each encryption a new value is generated), given the message m, the ciphertext will then be $ciph = g^m h^r \pmod{N}$.

Finally, the decryption can be done using the auxiliary table: $ciph^{s_p s_q}$ is looked up in the auxiliary table, the corresponding index number i is the message m.

This system works because of the following: we choose a message m, with

$m < u$ and a uniformly random value $r \in \mathbb{Z}_N$. The decryption of $ciph$ is $ciph^{s_p s_q} \equiv (g^m h^r)^{s_p s_q} \pmod{N} \equiv (g^m)^{s_p s_q}(h^r)^{s_p s_q} \pmod{N}$. Since h by definition has order $s_p s_q$, $(h^r)^{s_p s_q} = 1$. This leaves $(g^{s_p s_q})^m \pmod{N}$, and since the auxiliary table contains tuples of $\{(g^{s_p s_q})^i, i\}$ for $0 \leq i \leq u$, we are able to find the corresponding message of $g^{s_p s_q m}$.

2.2.6 The ABY Framework

In order to implement the garbled circuits protocols which we will present in the next chapter, we had to use a framework which provided ways to implement garbled circuits, as well as some additional secure computations. After trying some frameworks which turned out not to be sufficient for our protocols, we ultimately found a framework which did suit our needs: the Arithmetic sharing, Boolean sharing and Yao's garbled circuits framework (ABY), which is described in [7].

We will now briefly take a look at the Arithmetic sharing and the Boolean sharing, which we have not yet discussed. The Yao sharing boils down to a garbled circuits implementation similar to the one we discussed earlier in the previous chapter, also making use of the optimizations discussed in this chapter. The descriptions we provide are based on [7]. Since we don't make use of the Arithmetic sharing in our protocols, we refer to Section 7.2 in the Appendix for its description.

In Boolean sharing, a XOR-based secret sharing scheme is used to share a variable x. This is always done bit by bit, so shares of a bit x are defined in such a way that $\langle x \rangle_1 \oplus \langle x \rangle_2 = x$, here a value between $\langle\rangle$ denotes a Boolean share. In order to implement a multiplexer gate, we make use of R-OT (random oblivious transfer). For reference, a multiplexer gate takes two values and a boolean as input. When the boolean is true, the multiplexer returns the first value and when it is false, the multiplexer returns the second value. In a random oblivious transfer, the sender has no input messages and obtains two random messages (s_0, s_1). Then the protocol proceeds like the original OT protocol, which is described earlier in this section. The multiplexer gate can then be implemented using two parallel R-OT's on l-bit strings, which is easier than implementing the multiplexer gate using l AND gates, for which we would need $2l$ parallel R-OT's on 1-bit strings. Computationally speaking, the latter is more expensive, which causes us to prefer to use Boolean sharing for multiplexer gates when trying to find the fastest protocol.

In the ABY framework for garbled circuits, the three mentioned techniques, which improve the performance of garbled circuits, are all implemented. In addition to that, an optimization based on the type of gates that we use can be performed: MUL (multiplication) gates can be executed most efficiently within Arithmetic shares, which are optimized for such computations, MUX (multiplexer) gates can be executed most efficiently within Boolean shares, since the size and evaluation of those gates is constant in such shares, and CMP (compare) gates can be executed most efficiently within Yao shares, since the Boolean shares require a logarithmically scaling time to evaluate such gates, while in Yao shares, this can be done in constant time (Arithmetic shares can only perform arithmetic operations barring division in this framework, thus no CMP or MUX gates can be evaluated with them). The conversion between these types of shares is almost free within this framework, therefore we expect that converting between the types of shares will yield a faster protocol.

Since the ABY framework is implemented in C++, we expect it to be faster than similar protocols in Python. For the sake of a fair comparison, we also implemented a homomorphic encryption protocol in C++. Based on that implementation, we reason whether we expect the protocol that is implemented in Python to do better than the garbled circuits protocol, if it were implemented in C++.

We have now seen a large part of the existing methods that are used in some way in this thesis. Before moving on to the more formal descriptions of the protocols, we first look briefly at what this thesis adds to the existing literature.

2.3 Contributions of this Work

First and foremost, we present a new secure regression protocol, which has a better performance regarding speed, and a similar performance regarding security in comparison to the existing methods. In this protocol, we will make use of garbled circuits with some optimizations applied to it, which we will present in the next chapter, in addition to boolean circuits, which can be more efficient for certain operations within the secure domain. After defining this protocol theoretically, we will implement it.

We will also test this protocol against some other, partly existing protocols, by doing some experiments on the SMOCC dataset for each protocol, and comparing the results based on speed and degree of security. There are very few articles that perform such comparisons for secure regression methods, therefore we think that this will be a welcome addition to the existing literature.

Furthermore, we will try to change the way in which the regression is done in comparison to the way it was done in the PRANA-DATA project [29], in order for our predictions to become more accurate in this regard. Some attention to the difference in the results of those methods will be given in the fourth chapter.

3 Protocol Description

Our research setting is mainly based on the PRANA-DATA project [29]. Our baseline is the protocol included in that project, which we will describe here, along with the other protocols we used. These other protocols are all adaptations of the baseline protocol, changing only the security related steps in most cases. However, this can still lead to fundamentally different protocols, as we will see below.

3.1 General Setting

Since the setting will be the same for each protocol, we describe this once in this chapter. In addition, we look at the inputs for the protocols, which should also be very similar, if not equal, in order to have a fair comparison.

In these protocols, we have two parties: the individual (IND) and the institute managing the dataset (DAT). DAT has access to a large collection of historic child growth data of their first two years of life: in the dataset it manages, there are 17329 records of 2151 unique children, with each record being a measurement of a certain child at a certain age. Such a measurement involves among its 295 attributes the length and weight of the child, but also the background of the child, with things like the level of education of the mother, and the age of the mother at the delivery of the child. An example of (part of) such a record is given in Table 1. In this table, the terms that consist of a k followed by a number represent certain attributes, for example the first three k terms represent respectively the birth year, month and day of the child, while the terms named “year”, “month” and “day” right before that represent the date on which the measurement was performed. Out of the 295 attributes however, we will only use 8 in these protocols: the identifying number of a child, its date of birth, the dates on which a measurement was performed on this child, and the length of this child. The other attributes that we didn’t consider here represent the outcomes of certain tests that the child participated in, such as whether the child performs certain actions while playing a game, or represent complications that the child has experienced, such as certain diseases.

IND only has access to the growth data of its own child (we assume an individual has only one child), who is not included in DAT’s dataset and who is only one year old. There are four to seven measurements for such a

pnr	year	month	day	k101617	k101415	k101213	k1018	k10257	k102832	k103336	k103739	k1040	k1041
10001	88	5	2	88	4	2	Meisje	100	4050	556	378	Geen	Wel

Table 1: Part of a record out of DAT's database

child, depending on how often a check up was performed for the child. The goal of the protocols is to predict the growth data of the child of IND in its second year (for example, the length by month), using the growth data of all children in the first two years. The result of the protocols does not necessarily have to be made available to DAT, but does need to be learned by IND, since the goal of the protocols is for the parents of a child to learn its predicted growth pattern.

We employ the semi-honest adversary model, which assumes that each adversary will not deviate from the protocol, but will try to derive as much as possible from the information they do receive while following the protocol. We use this model because in this situation, both parties are discouraged to break the protocol in order to learn more about the data of the other party because of the risk this represents: if DAT would break the protocol and this would be noticed, its position as an institute which manages datasets containing privacy-sensitive data would become unsustainable, and if IND would break the protocol and this would be noticed, they would have to face charges based on invasion of privacy. Under this model, the protocols we present will be proven secure later in this chapter. This is a weaker notion of security than the malicious adversary model, in which adversaries may arbitrarily deviate from the protocol, but the semi-honest model is considered sufficient in a number of practical settings, including the one presented here, because of the risks of deviating from the protocol as mentioned above.

The inputs for the protocol are the so-called "comparison metrics" (CM) of all children: these are a single element out of the growth data of each individual child, including the child of IND, for example their length at twelve months, which is the CM that will be used in most of our protocols. These comparison metrics could in principle be any data or attribute, which is known for most children, but they are chosen in such a way that they are not considered sensitive information, so that the growth data of the children can not be derived from them. If they were chosen to be sensitive information, we would need to process them in the encrypted domain, which would cause our protocols to be computationally more expensive and thus slower (we expect this would cause an increase in processing time of 20-30%), therefore we chose them in such a way that they are considered non-sensitive information. The

CM's are used to determine what children are the most similar to the child of IND, so that an accurate prediction of the growth curve of the child of IND can be made using data from children out of DAT. In our case only a general distribution of the lengths of children at a certain age can be derived from the comparison metrics, which is already known based on DAT. Therefore we can precompute these comparison metrics. It should be noted that in the actual protocols, generally not all of the 2150 children whose data is in DAT are considered: this is because for some children, too few measurements are available to be able to compute a comparison metric reliably.

We define CM_0 as the comparison metric of the child of IND, and CM_1 to CM_n as the comparison metrics of the children in DAT, where n is the number of children whose growth data is in DAT. The growth data as a whole is represented for the parties by $DATA_{IND}$ and $DATA_{DAT}$ respectively. Using these definitions, we can say that IND uses $DATA_{IND}$ and CM_0 as input, and DAT has $DATA_{DAT}$ and $\{CM_1, ..., CM_n\}$ as input.

3.2 Original Protocol

We will describe the protocol which was used in the PRANA-DATA project, based on [29]. Before providing this description, we present a global outline of the protocol below. In this protocol, q denotes the query sent by IND (i.e. "What will the growth of my child look like in the next year?"), k_p denotes the public key, res denotes the result of the regression which is performed by ANA, and k_d denotes the decryption key. In the protocol, we will denote that values are encrypted by enveloping them within squared braces: i.e. $[b]$ indicates that the variable b is encrypted.

Protocol 3.1 A protocol similar to the one presented in PRANA-DATA, which makes use of homomorphic encryption

Party	**Input**	**Output**
IND	CM_0, q	res
DAT	$\{CM_1, ..., CM_n\}, k_p$ and $[DATA_{DAT}]$	-
ANA	-	-
DEC	k_d	-

1. $DATA_{DAT}$ is encrypted by DAT using k_p, so the data can be used in the protocol.
2. IND sends CM_0 and q to ANA, who then sends CM_0 to DAT. DAT then finds the five children in $[DATA_{DAT}]$ with closest CM to CM_0 and sends their records, $[R_{i,j}]$, back to ANA.
3. Based on q, ANA performs regression on the $[R_{i,j}]$ and sends $[res]$ to IND.
4. IND computes $[res^*] = [res]+[r]$, where r is an uniformly random number between 0 and 100000, and sends it to DEC, who decrypts $[res^*]$ using k_d, and sends the decrypted res^* back to IND, who computes $res^* - r = res$ to obtain the result.

We will now give a more detailed description of the protocol we just presented: in part (a) we will look at the assumptions that we made and at the roles of the different parties that participate in the protocol, step 1 of Protocol 3.1 is part of what is described there. In part (b), we describe the communication that precedes the analysis in order to get the data in the right place to analyze it, this corresponds to step 2. Then in part (c), the actual analysis is described, including the methods we use to deal with having to compute an average, step 3 is highlighted in this part. Finally, in part (d), the result is decrypted by DEC and obtained by IND, corresponding to what happens in step 4.

(a) **Setup:** In this protocol, in addition to IND and DAT, we have two other parties: the Analysis server (ANA) and the Decryption server (DEC). These parties are named quite intuitively, and later in the protocol we will see what roles they perform exactly. ANA is introduced in order to keep the query, which IND wants to do, hidden from DAT (in our case this is not so relevant, since we perform the same query each time the protocol is run, but in the original paper, different kinds of queries are suggested). DEC is introduced so no party can decrypt the encrypted data and obtain the plain data of another party, DEC itself can not obtain the plain data of any party either, since it only obtains the encrypted results of the analysis performed by ANA (with some added noise). We also assume that all data in DAT is homomorphically encrypted using Paillier [21] by a known public key k_p, which cannot be used to decrypt any data. The decryption key k_d is only known by the decryption server. Finally, we consider whether there is a record for a certain child in a certain month to be public information.

(b) **Communication:** First, IND sends CM_0 to ANA, in addition to the query q he wants to be answered by the protocol. ANA will then request all records describing the length from the five children from DAT with comparison metric closest to CM_0, by sending CM_0 to DAT (with "closest" we mean $min(|CM_0 - CM_i|, \forall i \in [1, n])$. We define these records as $R_{i,j}$, $i \in [1, 5]$, $j \in [1, 24]$, where i represents the child to which a record belongs, and where j represents the month of life of the child in which the record was measured. If there is no record for child i in month j, $R_{i,j} = 0$. Whether $R_{i,j}$ is equal to zero or not will be considered public information. This reveals some information about the child data, but this enables us to compute the average length of the children for each month in a swift way.

(c) **Analysis:** Having received the homomorphically encrypted full (containing all attributes) records, ANA performs the analysis that was requested by IND, which in this case consists of regression-like analysis on the records, obtaining an encrypted prediction for the development of the length of the child of IND. The analysis can be described as follows (everything what follows happens within the homomorphically encrypted domain): first, piecewise linear regression is performed for each child, obtaining curves predicting their length in the first two years of life. This regression method is described in its regular form in Section 7.1, the calculations that are performed in the homomorphically encrypted domain are described in Section 3.4. Then, using these curves, curve matching [27] is performed. This imputes the previously missing $R_{i,j}$ values, causing every child to contribute to every month for the average that we compute in order to predict the growth of the child of IND.

Then, the following is done: first, the records which are measured in the same month of life are aggregated and averaged, so for month j: $SUM_j = \sum_{i=1}^{5} R_{i,j}$ and $AVG_j = \frac{1}{\sum_{i=1}^{5}(R_{i,j} \neq 0)} SUM_j$, if $\sum_{i=1}^{5}(R_{i,j} \neq 0) \neq 0$, otherwise $AVG_j = 0$, which means that there are no measurements for month j. Since we know that in this case we have a measurement for each child in each month, we can compute the average multiplied by five for each month in the secure domain (by adding the measurements for each child in each month), and then when decrypted divide the plain value by five again to obtain the average. Every party knows that the averages will be computed in this way, so when IND receives the final AVG_j, he can obtain the real averages by dividing the decrypted AVG_j by five. When $AVG_j = 0$, the prediction for LEN_j is based on AVG_{j-1} and AVG_{j+1}, since there are no measurements for month j. After computing the SUM_j and the AVG_j, regression is performed on the AVG_j: $LEN_j = \beta_j AVG_j + \epsilon$, where LEN_j represents the predicted length of the child of IND in month j, β_j represents the regression coefficient for the child in month j and ϵ represents the intercept. The β_j can be constructed based on the measurement in month j and in month $j - 1$, for $j \in [1, 24]$, and ϵ is defined as the average measurement for month 0.

The regression coefficient changes each month since we try to model the growth of the child of IND using linear regression, which is an acceptable approximation when we do a separate regression for each period of one month. We can see this from the existing graphs (for example, Figure 6) modeling the growth of children in the first two years of life, and also see that this is likely not an appropriate approximation when looking at the whole period of two years at once. The encrypted prediction that is obtained by following this protocol is an encryption of the result of the regression, i.e. the β_j, LEN_j and AVG_j, $\forall j \in [1, 24]$.

(d) **Decryption:** The encrypted prediction $[res]$ is sent back to IND, who can add some noise $[r]$ to the prediction before sending it to DEC, obtaining $[res^*] = [res] + [r]$, in order to prevent DEC from learning the plain prediction. This noise can be some encrypted random number (known by IND), a new random number should be generated each time the protocol is executed though, to prevent DEC from learning the random number in some way, which would allow DEC to also obtain the plain predictions. DEC then is able to decrypt $[res^*]$, but because of the noise that was added by IND, nothing about the prediction is learned by DEC.

Finally, the decrypted result res^* is sent back to IND, and IND removes the noise r (since the data was homomorphically encrypted, it is possible to add encrypted noise to the encrypted data and remove the same noise in decrypted form from the decrypted data), to obtain

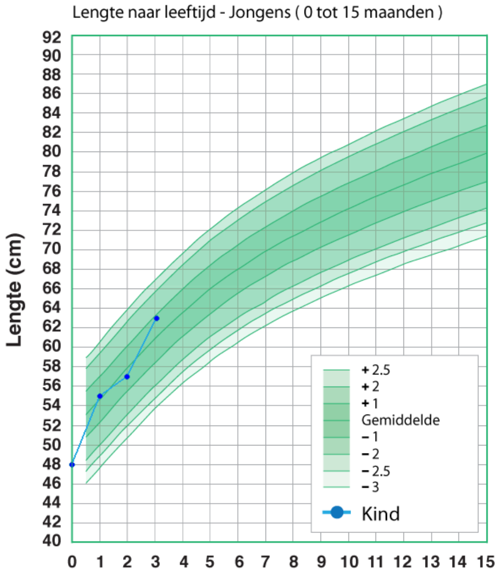


Figure 6: An example of a graph which represents a prediction on the growth of boys in their first 15 months of life, the blue points represent the actual length of an individual.

$res^* - r = res$. Then he divides the averages by five (and multiplies the βs by five), obtaining the result of the analysis done by ANA. This result contains the prediction of the growth data for the child of IND in its first two years of life, and since the child is one year old, the prediction of the growth data for the first year can be compared to the actual growth of the child in this period to measure how accurate the obtained prediction will be. We call this part of the prediction the "control group".

3.3 Garbled Circuits Protocol

In the previous section, we described the original protocol which was used in [29], changing only the curve-matching, which was performed there to predict the growth data of the child of IND, for regression. The main goal of this thesis is to develop a novel protocol to predict the growth data of the child of IND based on garbled circuits, and to compare this protocol to a few other protocols considering speed and degree of security. In order to build towards this goal, we present a basic garbled circuits protocol which predicts the growth data of the child of IND in the given setting. Since no garbled circuit protocol existed for this setting, this protocol is a contribution to the field. Just like the previous protocol, we will describe this protocol step by step, providing as much details as possible, but before that, we once again provide a general outline of the protocol. In this outline, res denotes the result of the regression which is performed, and k_p and k_d denote the public key and the decryption key respectively. Also, we will denote that values are garbled by enveloping them within double squared braces: i.e. $[[b]]$ indicates that the variable b is garbled.

Protocol 3.2 A basic Garbled Circuits protocol in the given setting

Party	Input	Output
IND (Garbler)	CM_0, k_p, k_d and $DATA_{IND}$	res
DAT (Evaluator)	$\{CM_1, ..., CM_n\}$ and $DATA_{DAT}$	-

1. IND sends CM_0 to DAT, who determines the five comparison metrics closest to CM_0 among the ones he possesses (CM_i). Then, DAT selects the corresponding children ($CHILD_i$) out of $DATA_{DAT}$, so their data can be used in the protocol.
2. IND prepares an encrypted version of the Boolean Circuit (BC) that the parties want to evaluate, creating a Garbled Circuit (GC) which performs regression, taking the garbled versions of $[[CHILD_i]]$ and $[[CHILD_0]]$ as inputs ($CHILD_0$ is the child corresponding to CM_0, whose data is in $DATA_{IND}$). IND garbles $DATA_{IND}$ to obtain $[[GARB_{IND}]]$. The GC is then sent to DAT, together with the $[[GARB_{IND}]]$.
3. DAT now still needs his own garbled input data to evaluate the GC. He obtains $[[GARB_{DAT}]]$ by means of an oblivious transfer protocol with IND involving the $CHILD_i$, and then evaluates the GC, obtaining $[[res]]$.
4. DAT sends $[[res]]$ to IND, who decrypts it, obtaining res.

We will now give a more detailed description of the protocol we just presented: in part (a), we discuss the roles of both parties, and give a general outline of how a boolean circuit that represents a regression should be constructed. In part (b), we describe the communication that is done before the analysis to get the data to the right party, so we can analyze it, this corresponds to what is done in step 1 of Protocol 3.2. In part (c), we describe the way in which the boolean circuit is garbled, corresponding to step 2. Then in part (d), the evaluation of the circuit is shown, which corresponds to steps 3 and 4 of Protocol 3.2. Part (e) provides details on the way in which the regression is performed within the garbled circuits, shedding light on some decisions we made to avoid having to do divisions within the garbled circuits. In part (f), we argue that our protocol indeed is secure in the semi-honest sense. Part (g) mentions an alteration that could be made to the protocol to allow both parties to learn the results of the protocol, while remaining secure. Finally, part (h) describes how our protocol could be used to predict growth curves with a different regression method, which makes use of a different comparison metric than we used until then.

(a) **Setup:** Since we employ garbled circuits in this protocol, one of the parties needs to be the Garbler, who generates and encrypts the garbled circuits, and one of the parties needs to be the Evaluator, who evaluates the garbled circuits and obtains the results. In this protocol we don't need two additional parties, because garbled circuits work in a whole different way than homomorphic encryption. Even without those parties, we can still guarantee the same security aspects that were achieved in the homomorphic encryption protocol by using the additional parties: by having IND construct the circuit, DAT learns nothing about the query, he can only see the types of the gates that are being used, and because IND has the decryption key and DAT evaluates the circuit, there is no encrypted data available to IND for which he has the decryption key that he isn't allowed to see in plain text.

Since we only need IND to learn the results, we choose IND to be the Garbler, and DAT to be the Evaluator, because the Evaluator obtains encrypted results, which have to be sent to the Garbler to be decrypted into the final results. Therefore, if we pick IND to be the Garbler, this saves one messaging step in which the decrypted results would be sent to the Evaluator from the Garbler. Furthermore, a Boolean Circuit (BC), which represents the linear regression function that the parties want to be computed securely, should be constructed by IND beforehand, so DAT learns no relevant information about the query.

Such a boolean circuit can be constructed in the following way: first, we look at a description of the regression for which we want to construct a boolean circuit. Then, we do the following: for each addition in the regression, we represent it by one or more circuits that each add two integers together, for each comparison in the regression, we represent it by one or more circuits that each compare two integers, and for each scalar multiplication in the regression, we represent it by one or more circuits that multiply an integer with a scalar. Furthermore, the inputs of the boolean circuit are the length measurements, including the month in which they were performed, of all six children on which the regression is performed. The outputs of the boolean circuit are the predicted regression coefficients for the growth of the child of IND, and the party who receives this output is IND. Some implementations of boolean circuits of this type can be found in Sections 7.3 and 7.5 of the Appendix.

(b) **Communication:** Before we get to the secure part of the protocol, we first determine the five closest comparison metrics to CM_0 in DAT, in order to get a prediction which is focused on $DATA_{IND}$ instead of just looking at the general curve obtained by considering all data in $DATA_{DAT}$. As noted before, the comparison metrics are not considered sensitive information, so we can do the comparison of the CM_i, $i \in \{1, n\}$ to CM_0 in plaintext. We do this by sending CM_0 to DAT, who can then locally determine the children that have the closest comparison metric to CM_0. These children can be identified by DAT by the identifier of their corresponding comparison metric (which is the same as their identifier in the measurement database), their data will be the input of DAT in the garbled circuits part of the protocol. We denote each of these children by $CHILD_i$, where i is the subscript of the corresponding comparison metric of the child. Only DAT knows the data of these children, since they are privacy-sensitive data.

(c) **Garbling the Boolean Circuits:** Now we have the input of each party for the garbled circuits, we can move on to the secure part of the protocol. First, IND prepares an encrypted version of the BC in the following way: for each wire j of the circuit, IND associates two random cryptographic symmetric keys w_j^0 and w_j^1 with it, where w_j^0 encodes a 0-bit and w_j^1 encodes a 1-bit. These encodings do not reveal any information about its plain value, since they both are randomly generated. Then, for each binary gate g_m in BC, helper information in the form of garbled tables T_m is created by IND, which allows the decryption of the output key (and nothing more) given a gates' input keys. A garbled table is the garbled version of the binary table of a gate. The garbled tables of all gates together form the garbled circuit, $GC = \{T_1, ..., T_p\}$ (where p is the number of gates), which is sent to DAT.

(d) **Evaluating the Garbled Circuits:** In order to evaluate this circuit, DAT needs the garbled input values of himself and IND ($GARB_{IND}$

and $GARB_{DAT}$). The $GARB_{IND}$ are by definition encrypted (because they are garbled), so they can be sent to DAT. However, DAT does not know how to garble his own input values, and does not want IND to learn them. Therefore, an oblivious transfer protocol [9] is used to obtain $GARB_{DAT}$, bit by bit. For a description of such a protocol, we refer to Section 2.2. After this, DAT has obtained $GARB_{DAT}$ as well as $GARB_{IND}$, and evaluates GC on these values. DAT then obtains the garbled output values ($GARB_{OUT}$), which he sends to IND, who then decrypts the output.

(e) **Regression:** The output consists of the result of a type of regression, which depends on the test setting and which is performed on the five children from DAT who have the closest comparison metrics to the child from IND and the child of IND itself. This regression is performed as follows: first, we take a look at the exact input of the circuit. For each child on which the regression is performed, all length measurements are included, including the month in which they were performed. For IND this is data of one child ($CHILD_0$), for DAT this is data of five children ($CHILD_i$, with $i \in [1, 5]$).

Then, using this data, we perform a moderately different regression protocol than in the original protocol: first, the records which are measured in the same month of life are aggregated and averaged, over all 6 children (the child of IND is also taken into account). So, for month j we have: $SUM_j = \sum_{i=0}^{5} R_{i,j}$ and $AVG_j = \frac{60}{\sum_{i=0}^{5}(R_{i,j} \neq 0)} SUM_j$ if $\sum_{i=0}^{5}(R_{i,j} \neq 0) \neq 0$, otherwise $AVG_j = 0$, where the variables have the same meaning as they have in the original protocol, the only difference is that $i \in [0, 5]$.

If we would want to compute the average this way in the secure domain, we would need to be able to do arbitrary divisions on data within garbled circuits. Since this is not possible, we present a method which corrects for the number of children who contributed to the average value of length in a certain month by multiplication with integers: we use the least common multiple (LCM) of $\{1, 2, 3, 4, 5, 6\}$, which is 60. So, we multiply each sum by $\frac{60}{\sum_{i=0}^{5}(R_{i,j} \neq 0)}$ (this is always an integer when $\sum_{i=0}^{5}(R_{i,j} \neq 0) \neq 0$, since $\sum_{i=0}^{5}(R_{i,j} \neq 0) \in [0, 6]$, and $LCM(1, 2, 3, 4, 5, 6) = 60$) to obtain the average multiplied by 60: $AVG_j = \frac{60}{\sum_{i=0}^{5}(R_{i,j} \neq 0)} SUM_j$ if $\sum_{i=0}^{5}(R_{i,j} \neq 0) \neq 0$, otherwise $AVG_j = 0$. Because the data is within the garbled circuits here, these additions and scalar multiplications are done using the appropriate garbled boolean circuits.

After computing the AVG_j, the regression is performed as follows: with the help of these averages and the corresponding months, we do some computations corresponding to the type of regression, and obtain an approximation for the length of the child of IND in the first two years of life. For the 'simple' regression, we do piecewise linear regression, which comes down to taking each two points that are closest to each other regarding in which month they were measured, and then constructing a line between those two points.

For the linear regression, we determine the slope and intercept of the resulting line using the following formulas:

$$intercept = \frac{\sum_j(AVG_j) * \sum_j(j^2) - \sum_j(j) * \sum_j(j * AVG_j)}{n * \sum_j(j^2) - (\sum_j(j))^2} \tag{1}$$

$$slope = \frac{n * \sum_j(j * AVG_j) - \sum_j(j) * \sum_j(AVG_j)}{n * \sum_j(j^2) - (\sum_j(j))^2} \tag{2}$$

Here, j is the month of life a child is in and n is the total number of different months in which we have measurements. Some more details about how these regression methods work can be found in Section 7.1.

Since performing fractions within our garbled circuits framework would be computationally expensive, hard to implement, and does not hide significant additional information, we compute the nominator and denominator separately within the garbled circuits, and perform the fraction after decryption. This does not leak any data from which any party can derive more than that which can be derived from the result, and thus does not influence the degree of security of our protocol.

After the regression, DAT obtains the garbled output of the circuit, which is the AVG_j and the relevant regression coefficients, which together form the prediction of the growth of the child of IND. Since $DATA_{IND}$ is only available for the first year, we can see that part of the regression as a "control group" (just like was described in the end of the original protocol), which indicates how accurate our prediction of the second year will be, if we compare that part of the regression to $DATA_{IND}$ (which IND can do privately when he has the results).

(f) **Ensuring Privacy:** Since the circuit that will be evaluated in the protocol is known by both parties, we see that DAT shouldn't get the plain results, since he could do the same kind of regression as used in the protocol on just the five children with the closest comparison metric out of his database ($CHILD_i$, with $i \in [1, 5]$), which he knows since the closest comparison metrics were determined publicly, then compare the result of this regression (β'_j, LEN'_j and AVG'_j) to the result which he got from the protocol (β_j, LEN_j and AVG_j), and then find (part of) $DATA_{IND}$ by imputation. This imputation can be done easily by computing $((\sum_{i=1}^{5}(R_{i,j} \neq 0)) + 1)AVG_j - (\sum_{i=1}^{5}(R_{i,j} \neq 0))AVG'_j$: if this is equal to AVG_j, then $AVG_j = AVG'_j$, and thus there either is no value measured in month j for the child of IND, or its value is equal to AVG_j. In the first case, AVG_j is a good approximation of the value of the child of IND and in the second case, $AVG_j = R_{0,j}$, thus AVG_j is equal to the value of the child of IND in month j.

If $((\sum_{i=1}^{5}(R_{i,j} \neq 0)) + 1)AVG_j - (\sum_{i=1}^{5}(R_{i,j} \neq 0))AVG'_j \neq AVG_j$, then there is a value measured in month j for the child of IND, and we have that $R_{0,j} = ((\sum_{i=1}^{5}(R_{i,j} \neq 0)) + 1)AVG_j - (\sum_{i=1}^{5}(R_{i,j} \neq 0))AVG'_j$ since $((\sum_{i=1}^{5}(R_{i,j} \neq 0)) + 1)AVG_j$ is the number of children that were considered in the regression for whom a value was measured in month j, times the average length of all children in that month, which is equal to the sum of the lengths all children in that month, thus $((\sum_{i=1}^{5}(R_{i,j} \neq 0)) + 1)AVG_j = (\sum_{i=0}^{5}(R_{i,j} \neq 0))AVG_j = \sum_{i=0}^{5} R_{i,j}$. The part behind the minus, $(\sum_{i=1}^{5}(R_{i,j} \neq 0))AVG'_j$, is equal to the number of children that were considered in the regression ànd that were in DAT, for whom a value was measured in month j, times the average length of those children in that month, which is equal to the sum of the lengths of those children in that month, thus $(\sum_{i=1}^{5}(R_{i,j} \neq 0))AVG'_j = \sum_{i=1}^{5} R_{i,j}$. Thus, when we substract the second part from the first part, we obtain the measured length of the child of IND in month j, since clearly $\sum_{i=0}^{5} R_{i,j} - \sum_{i=1}^{5} R_{i,j} = R_{0,j}$. All operations we did would be allowed according to the semi-honest adversary model, since DAT does follow the protocol in this case, he just computes some extra information in order to be able to derive $DATA_{IND}$.

(g) **Possible alteration:** An alternative protocol in which DAT can be allowed to obtain the results without being able to find (part of) $DATA_{IND}$ would involve obtaining the five children from DAT who have the closest comparison metrics to the child from IND in an oblivious way (so without DAT knowing which children exactly were selected, and without IND learning anything about the other children from DAT), but since DAT does not have to get the results in our case, we will not go into detail about such a protocol.

(h) **Alternative Regression:** Another regression method that we considered was the 'slope-based' regression, of which a description can be found in Section 7.1. For this method, we use a different comparison metric than for the methods we described before, namely the slope of the curve describing the length of a child at 12 months, instead of the actual length of the child. The reason we chose this comparison metric is rather than the actual length is that intuitively, it makes sense for children that grow with similar speed at the same age to keep growing with similar speed. When we tried to validate this using the data from the SMOCC-dataset, we came to the conclusion that on average, the speed with which the children grow will indeed remain roughly within the same range, with a margin that varies between 1.5 and 3.

    Using this comparison metric, we find the five children out of DAT with the closest comparison metric to the child of IND. We denote the children once again by $CHILD_0$ (the child of IND) and $CHILD_i$ with $i \in [1, 5]$ (the five children out of DAT). Then, for each month, we compute the average slope of the curves of the $CHILD_i$, so IND can later apply that slope in the corresponding month to construct the prediction of the development of the length of $CHILD_0$. Only the slopes from months later than the month in which the final measurement of $CHILD_0$ was measured are applied, since that is the part for which we want to predict the length of $CHILD_0$. Since the slopes are not necessarily integers, we need to scale them before averaging them in the secure domain, because we can only process integers within our garbled circuits.

    Then, within the garbled circuits, we apply the same principle as with the other regression methods and scale with the least common multiple (LCM) of $\{1, 2, 3, 4, 5\}$, which is 60. Note that since a continuous graph has a slope in every month, we get an average slope for each month from the final measurement we have for $CHILD_0$, which was not necessarily the case with the other regression methods, since those looked at the average of the measurements for the closest children out of DAT, and these measurements did not occur in every month. Thus, by evaluating the garbled circuits, DAT obtains the encrypted average slopes for each month, which he sends to IND, who can decrypt those and apply them to find a prediction of the development of the length of $CHILD_0$.

3.4 Alternative Homomorphic Encryption Protocol

In this section, we describe a different homomorphic encryption protocol than the one presented in Section 3.2. Its structure is different, since we built this protocol ourselves rather than using an existing protocol, and the cryptosystem used is also different: in this protocol we use the Damgård, Geisler and Krøigaard (DGK) cryptosystem. We tried to use similar operations to the ones used in the garbled circuits, in order to keep the comparison between the cryptosystems as fair as possible. Without any further ado, we present a general outline of the protocol, before describing the protocol in more detail. In this outline, res denotes the result of the regression which is performed, k_p denotes the public key and k_d denotes the decryption key (note that the keys for this protocol consist of multiple different values, this representation is a simplification for the sake of clarity). Also, we will denote that values are encrypted by enveloping them within squared braces: i.e. $[b]$ indicates that the variable b is encrypted.

Protocol 3.3 A homomorphic encryption protocol which uses the DGK cryptosystem in the given setting

Party	Input	Output
IND	CM_0, k_p, k_d and $DATA_{IND}$	res
DAT	$\{CM_1, ..., CM_n\}, k_p$ and $DATA_{DAT}$	-

1. IND sends CM_0 to DAT, who determines the five comparison metrics closest to CM_0 among the ones he posseses (CM_i). Then, DAT selects the corresponding children ($CHILD_i$) out of $DATA_{DAT}$, so their data can be used in the protocol.
2. IND encrypts $CHILD_0$ (the data of his child) and DAT encrypts $CHILD_i$, both using k_p, and IND sends $[CHILD_0]$ to DAT.
3. DAT then performs the regression, obtaining the correct scale, s_j, to use for each month j by checking in the encrypted domain whether the measurement for $[CHILD_0]$ is zero or not, and adding the result to the number of measurements that the $[CHILD_i]$ contribute to that month. This value is then decrypted, so that the encrypted average for that month can be scaled with the scalar $s_j = 60/n_j$, where n_j is the number of children contributing a measurement in month j.
4. The result of the regression is sent to IND, who then decrypts it to obtain res.

Now, we give a more detailed description of the protocol we just presented: in part (a), we describe the roles of the parties that participate in the protocol. Part (b) discusses the communication that is done before the analysis, which ensures that each party has the correct data to perform its part of the analysis, this corresponds to step 1 and 2 of Protocol 3.3. In part (c), we look at how the actual analysis is performed, and what methods are applied to allow us to perform regression using homomorphic encryption, what we describe here corresponds to step 3 and 4. Finally, in part (d), we shed some light on the way in which we calculate the components of the result of the linear regression for homomorphic encryption, and argue that this indeed is secure in the semi-honest setting.

(a) **Setup:** In this protocol, other than in the original protocol, we don't have two parties in addition to IND and DAT. We pay a small price for this, since the function that IND wants to perform on the data will be known to DAT in this scenario. However, since we always want to do regression in the protocol in our setting, this is not a problem. Despite this, we should still take into consideration that the original protocol does conceal the function in our analysis. Furthermore, DAT gets to know the number of children that contribute to the average measurement each month in this protocol, from which he can derive in which month the child of IND has a measurement. This is necessary because we can't compute the product of two encrypted integers in this framework. Since DAT doesn't get to see the plain results, this knowledge does not reveal anything about the data values of the child of IND, and therefore this does not negatively influence the degree of security of the protocol. Finally, IND has access to the decryption key in this protocol, but since he doesn't get to see any encrypted data that he isn't allowed to see in plain text (since DAT performs the regression), and since IND is semi-honest, the fact that IND holds the decryption key doesn't impair the security of the protocol. We use the DGK cryptosystem here, implemented in C++ through the SeComLib framework [1].

(b) **Precomputation:** In preparation of the secure part of the protocol, we send CM_0 from IND to DAT, so DAT can find the five children with comparison metric closest to CM_0. Since the comparison metrics are not considered privacy-sensitive information, this comparison can be done in plaintext. Continuing, DAT extracts the data of the aforementioned children from $DATA_{DAT}$ in order to encrypt it and use it as input for the protocol. IND does the same with the data for his child, and then the protocol can be initiated.

(c) **Analysis:** First, IND and DAT encrypt the data of their respective children, then IND sends the encrypted data of his child to DAT, so that DAT obtains $R_{i,j}$ for $i \in [0,5]$ and $j \in [0,23]$, where i represents the child of which the length data was measured (0 for the child of IND, 1-5 for the children of DAT), and j represents the month in which a measurement was performed. Then, DAT checks for each month which children have a measurement for that month, and adds these measurements together within the secure domain. This is possible because DGK is an additive homomorphic cryptosystem, so secure additions are possible.

Since not every month has the same number of children with a measurement, the sums obtained by adding the measurements for each month need to be scaled in order to get the average measurement value for each month. However, it is not possible to do division within the encrypted domain with the DGK framework we use. Therefore, we use the least common multiple of the possible numbers of children that can have a measurement in a certain month, this gives us $LCM(1,2,3,4,5,6) = 60$. In order to achieve this, we want to scale the sums of measurements in month j with $60/n_j$, where n_j is the number of children that contributed a measurement in month j. Because we cannot do comparisons between encrypted values in our framework, we need to decrypt n_j, compute $60/n_j$ and then multiply the encrypted average by the obtained scalar.

This is done as follows: DAT checks for each j whether $R_{0,j}$ is equal to 0, this is possible in the encrypted domain in our framework and only reveals whether there is a measurement for $CHILD_0$ in month j, which is exactly the information DAT needs. By adding whether there is a measurement in for $CHILD_0$ in month j to the number of measurements for $CHILD_i$ in month j where $i \in [1,5]$, DAT obtains the total number of measurements in month j, n_j. Then, DAT computes $60/n_j$ and multiplies the average of month j by this scalar in the encrypted domain, obtaining the encrypted average value for month j scaled with 60. After obtaining these values, DAT sends them to IND, who then can decrypt them. This way IND obtains the results of the piecewise linear regression.

(d) **Secure Linear Regression:** If we want to perform linear regression within the homomorphic encryption protocol, we want to determine the slope and the intercept using the same formulas as in the garbled circuits protocol, namely formulas (1) and (2). Some more details about these regression methods can be found in Section 7.1. The calculation of these values in the encrypted domain proved difficult, since no values greater than 2^{17} could be stored in the encrypted domain. Therefore, DAT constructs each sum in the encrypted domain (multiplying is done by taking the encrypted value to the power of the public value, subtraction is done by multiplying one encrypted value with the inverse value of the other encrypted value):

$$intercept_{num} = [\sum_j (AVG_j)]^{\Sigma_j (j^2)} * ([\sum_j (j * AVG_j)]^{\Sigma_j (j)})^{-1}$$

$$slope_{num} = [\sum_j (j * AVG_j)]^n * ([\sum_j (AVG_j)]^{\Sigma_j (j)})^{-1}$$

$$intercept_{den} = slope_{den} = [n]^{\Sigma_j (j^2)} * ([\sum_j (j)]^{\Sigma_j (j)})^{-1}$$

Here, $j \in [0, 23]$ represents the months, and n represents the total amount of months in which there is a measurements for at least one child that was taken into account in the protocol. The sums for the numerators were split into two parts by first considering their formulas for $j \in [0, 11]$, and then for $j \in [12, 23]$. Then, these values are sent to IND who computes the intercept and slope in the plain domain, by decrypting each sum and them combining them according to the formulas given for the slope and the intercept. This does not reveal any additional privacy-sensitive information to IND, since only part of the combined sums of the average measurements and the contributing months are revealed, but without information about the largest part of the individual measurements, no privacy-sensitive information can be derived from this. Having obtained this slope and intercept, IND can create a curve representing the predicted growth of his child.

3.5 Modified Garbled Circuits Protocol

In Section 3.3, we described a basic version of a garbled circuits protocol applied to the PRANA-DATA use case. Here, we will describe a slightly modified version of this protocol, which we expect to be faster than its predecessor. We will describe only the differences of this protocol compared to the previous garbled circuits protocol, since they are very similar. Before that, we once again provide a general outline of the protocol. In this outline, *res* denotes the result of the regression which is performed. Also, we will denote that values are garbled by enveloping them within squared braces: i.e. $[[b]]$ indicates that the variable b is garbled.

Protocol 3.4 A modified Garbled Circuits protocol in the given setting

Party	Input	Output
IND	CM_0, k_p, k_d and $DATA_{IND}$	res
DAT	$\{CM_1, ..., CM_n\}, k_p$ and $DATA_{DAT}$	-

1. IND sends CM_0 to DAT, who determines the five comparison metrics closest to CM_0 among the ones he possesses (CM_i). Then, DAT selects the corresponding children ($CHILD_i$) out of $DATA_{DAT}$, so their data can be used in the protocol.
2. IND prepares an encrypted version of the Boolean Circuit (BC) that the parties want to evaluate, creating a garbled circuit. Also, a separate boolean circuit is created, which is used together with the garbled circuit to perform regression. The garbled circuit takes the garbled versions of the children's data, $[[CHILD_i]]$ and $[[CHILD_0]]$ as inputs ($CHILD_0$ is the child corresponding to CM_0, whose data is in $DATA_{IND}$), and the boolean circuit converts intermediate values to calculate MUX gates more efficiently. IND garbles $DATA_{IND}$ to obtain $[[GARB_{IND}]]$. The [[GC]] is then sent to DAT, together with the $[[GARB_{IND}]]$.
3. DAT now still needs his own garbled input data to evaluate the [[GC]]. He obtains $[[GARB_{DAT}]]$ by means of an oblivious transfer protocol with IND involving the $CHILD_i$, and then evaluates the [[GC]], obtaining $[[res]]$.
4. DAT sends $[[res]]$ to IND, who decrypts it, obtaining res.

We will now look at the differences of the protocol we used here compared to the one we presented in Section 3.3. The functionality of both protocols is the same, however we expect this protocol to be faster, since we perform the MUX (multiplexer) gates in a boolean circuit. A description of these gates can be found in Section 2.2 This is done by converting the input values (shares) of the MUX gate to boolean shares, then performing the MUX gate, and then converting the output values back to garbled shares. We do this several times throughout the protocol, but the general structure of the protocol remains unchanged compared to the description of the other garbled circuits protocol.

This conversion also has little negative impact on the security of our protocol, since the conversion is done in the secure domain and from one secure representation to another secure representation, thus the disadvantages of this are that a possible adversary now could try to break either of the two secure representations or that the conversion between two secure representations possibly isn't secure. However, since both representations are secure in the semi-honest sense, this doesn't impact the degree of security of the protocol. Regarding the conversions, we refer to the article in which the ABY framework was presented [7] for proof that this indeed does not compromise the security of the protocol. Another possible optimization would be to use an Arithmetic circuit to perform MUL gates, but the conversion of garbled circuits to Arithmetic circuits proved to be not yet possible without failure in the ABY framework (at least for our application), thus applying these circuits could be a future improvement.

4 Results

In this chapter we discuss the experiments we performed on the protocols, which were described in the previous chapter, and their results. There are three main aspects that we look at when comparing these protocols: their speed when we vary the number of children that are considered out of DAT and the type of regression we perform, their accuracy, which depends on the type of regression we perform, and their security, specifying which variables are kept secret by the protocol, and whether the security is computational (breakable by a sufficient amount of computational power) or unconditional (secure independent of computational power of an adversary).

4.1 Comparison of Speed

We compared the speed of the different methods with each other, also considering which regression method was used since this influences the amount of computation that has to be done. We looked at different numbers of children that should be extracted from DAT using the CM's: any number between one and twenty children. We expected that considering more children would lead to a slower method. Looking at Table 2 and Figure 7, we see that on average this is indeed the case for each method. We also see that the protocols which perform a full linear regression, indicated in the table by '(linear)', rather than a piecewise linear regression, indicated in the table by '(simple)', are a bit slower. This can be explained by the fact that piecewise linear regression is a simplification of full linear regression, in the garbled circuits protocols the circuit which does the linear regression is even built on the circuit which does the piecewise linear regression.

We also see that the protocols which perform a 'slope-based' regression, indicated in the table by '(slope)', rather than a piecewise linear regression, are a bit faster, this can also clearly be seen in Figure 8. This can be explained by looking at the source code for both methods, which can be found in Section 7.3 and Section 7.4: there we see that the protocol that performs the 'slope-based' regression is almost identical to the protocol that performs the piecewise linear regression. The difference in speed between the two is caused by an additional required sanity check, as is explained in Section 7.4. Therefore, it makes sense for the former protocol to be a bit faster than the latter protocol.

When comparing the speed of the different types of protocols, we see in Figure 7 that the protocols that are based on garbled circuits outperform the protocols that are based on homomorphic encryption by a notable difference. For example, when comparing the linear regression protocol of the regular garbled circuits to the original protocol, there is an average difference in speed of more than fifteen seconds (which corresponds to a factor of seven to eight), regardless of the number of children that should be selected out of DAT. This can partially be explained by the fact that the original protocol is implemented in Python and the regular garbled circuits protocol in C++, since the latter is faster in general. This is illustrated by the fact that the average difference in speed between the regular garbled circuits protocol and the alternative homomorphic encryption protocol, which is implemented in C++, is between six and seven seconds (which corresponds to a factor of three to four). Since the operations that are performed in the original protocol and the alternative homomorphic encryption protocol are similar, we estimate the influence of the fact that the original protocol is implemented in Python rather than in C++ on the speed to be eight to nine seconds, which corresponds to a factor two if we compare the speed of the two protocols which make use of homomorphic encryption.

The final difference between the two protocols which could explain the difference in speed is the fact that one protocol makes use of garbled circuits, while the other one makes use of homomorphic encryption. The main difference between these two protocols regarding their speed lies in the point that for garbled circuits, most time is needed in order to build the circuits (thus the preprocessing takes up more time), while for homomorphic encryption, most time is needed in order to perform operations and data transfers in runtime. Since we work with preprocessed data here, the large difference in speed could be explained.

Regarding the modified garbled circuits protocol, which is depicted in Table 2 under the term ‘modGC’, we see that, compared to the regular garbled circuits protocol, the methods are a bit faster in each case, which was the purpose of the modification. This can clearly be seen in Figure 8. The increase in speed lies between the 10% and 20% in each case, which is reasonable considering the modifications we made: the modifications were performed on a small part of the protocol and therefore we did not expect a major increase in speed.

Summarizing, we found that the garbled circuits protocols were faster than the homomorphic encryption protocols, that the modified garbled circuits protocol was the fastest protocol, and that the slope-based method was the fastest method.

# of children	Original Protocol (Python)	GC Protocol (simple) (C++)	GC Protocol (linear) (C++)	GC Protocol (slope) (C++)	modGC Protocol (simple) (C++)	modGC protocol (linear) (C++)	modGC Protocol (slope) (C++)	Alt HE Protocol(simple) (C++)	Alt HE Protocol (linear) (C++)	Alt HE Protocol (slope) (C++)
5	17,75	2,16	2,71	2,04	1,93	2,29	1,84	8,74	8,85	8,72
10	18,65	2,28	2,92	2,11	2,00	2,47	1,92	9,16	9,31	9,10
15	19,14	2,63	3,05	2,40	2,35	2,59	2,13	9,33	9,42	9,20

Table 2: The speed in seconds with which a protocol is performed, averaged over 10 runs.

4.2 Comparison of Accuracy

In order to compare the accuracy of the different protocols, we removed one third of all children from the SMOCC dataset one by one to use as test individuals. We then performed every protocol on each of these test individuals, and compared the results in order to draw conclusions about the accuracy of the methods. In Figure 10, we see the results of the protocols for one test individual, and in Figure 11, we see the average results of the protocols for 634 different test individuals. What should be kept in mind when looking at this comparison is that the original method, which is based on homomorphic encryption, does not take the measurements of the length of the child of IND other than its comparison metric into account, while the methods, that make use of Garbled Circuits and the alternative homomorphic encryption protocol, do take these measurements into account up until the first year of life of that child. This is the case because that information was not taken into account in the original protocol, while we thought it could lead to better predictions. Also note that we make no distinction between the modified garbled circuit protocols, the alternative homomorphic encryption protocols and the garbled circuits protocols in Figure 10 and Figure 11. This is because the methods we developed all perform the same calculations, changing only the efficiency with which some calculations are processed and the exact methods that are used to perform some calculations. We include the predictions of the original protocol separately in our graph, since the calculations performed in the original protocol are slightly different than the calculations that are performed in the other protocols, therefore we expect its prediction to be different.

Since we want our methods to predict the length of the child after one year, we value the predictions of our protocols in that period higher than good

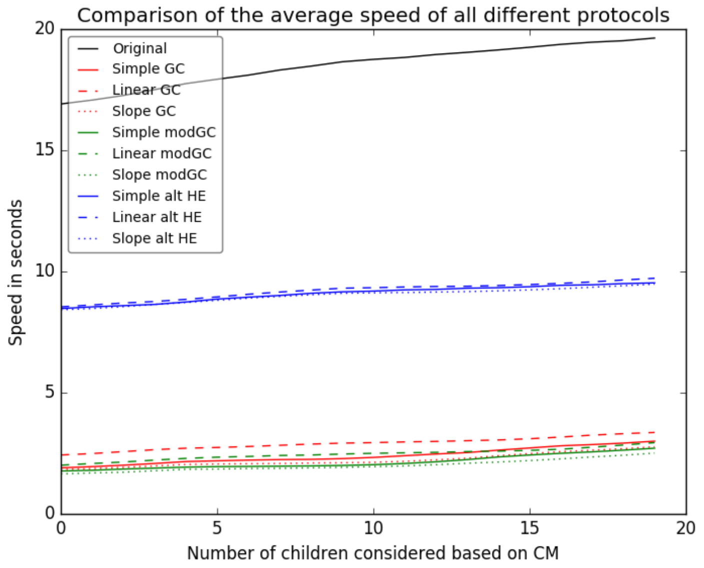


Figure 7: A graph depicting the average speed of all different protocols, considering the number of children that were taken into account based on the CM.

predictions before that point in time (the predictions before that point in time only serve as verification that our methods work correctly, since we can compare them to the actual measurements of the child). Looking at Figure 11, we see that both the 'slope-based' and the 'simple' method give an accurate prediction of the actual development of the length of the child in this case. We see that for the slope-based method, we only have a prediction from twelve months and onwards. This is the case because we pick the closest children to the child of IND based on the slope of the curve describing their length at twelve months, thus we only predict the slope here for each month from twelve months and onwards. We therefore suspect that the accurate results for the slope-based method can be explained by the theory that children that grow with similar speed at twelve months will on average continue to grow with similar speed. Regarding the 'simple' method, we see that up to twelve months, it approximates the child growth accurately. However, after twelve months, this prediction becomes less accurate (but still more accurate than the other methods). We suspect this is influenced by the fact that the records of the child of IND up to twelve months are taken into account for this method, and the records afterwards are not.

In Table 3, we give the squared residuals of the average prediction using the

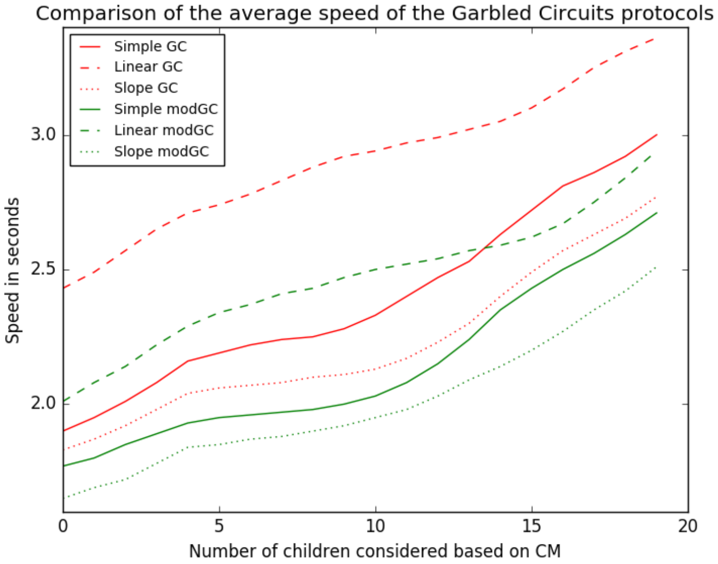


Figure 8: A graph depicting the average speed of the garbled circuits protocols, considering the number of children that were taken into account based on the CM.

same individuals as used to create Figure 11. We tried varying the number of children extracted from DAT using the CM's, this can be seen in Figure 9. There we can see that the error tends to go down until we have ten selected children, from then on it stays roughly the same. We expect the error to increase again when we select a large part of the dataset, since we would then resemble the method from which we are trying to improve, where for the child of the individual all children in the database are considered when predicting its growth curve, instead of only the ones with a comparison metric close to the comparison metric of the child of the individual. For efficiency reasons (mainly because of the speed of the protocols), we decided to stick with the original number of children that are selected based on the comparison metrics, namely five. For a setting where accuracy is the primary objective however, we strongly encourage that ten children would be selected based on the comparison metric. We see in Figure 9 that when considering five or more children, on average the 'simple' method gives the most accurate predictions, and when considering less than five children, both the 'simple' and 'slope-based' method on average give the most accurate predictions.

Looking at the linear method in Figure 11, we see that the prediction is worse than that of the simple and slope-based method, this corresponds to what we

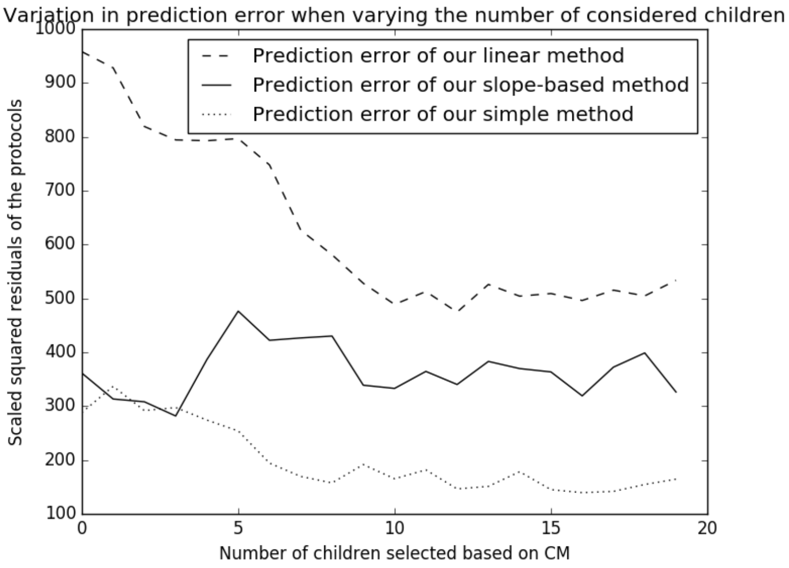


Figure 9: A graph showing the difference in scaled average squared residuals for the three methods we constructed, when varying the number of selected children.

Original	Simple	Linear	Slope-based
4063	297	3152	869

Table 3: The average squared residuals of the different methods in their predictions of the lengths of different children after the age of one year.

saw in Table 3. Since the growth curve of a child in its first two years is not linear (but can be close to linear), it is not surprising that the linear method does not yield the best prediction. Also, we saw in some individual cases that their was a strong non-linear correlation between their measurements, which is hard to approximate with the linear method.

Finally, we looked at the original method in Figure 11. We see that it gives a prediction that is too conservative from 6 months and onward. This could be explained by the difference in input values, but also by the difference in method, since the original method makes use of curve matching [27] to predict the growth of the child.

When we looked at other test individuals, we saw a similar trend to the one illustrated in Figure 11: the predictions of the 'simple' and ' slope-based' methods we developed are fairly accurate, while the prediction of the original protocol is somewhat conservative from the sixth to ninth month and onwards, and the prediction of the 'linear' method also is less accurate than that of the other two methods we developed. This leads us to strongly preferring the 'simple' and 'slope-based' methods we implemented accuracy wise over the original method, between these two methods there is a slight preference for the 'simple' method, since in general its predictions are somewhat closer to the actual measurements.

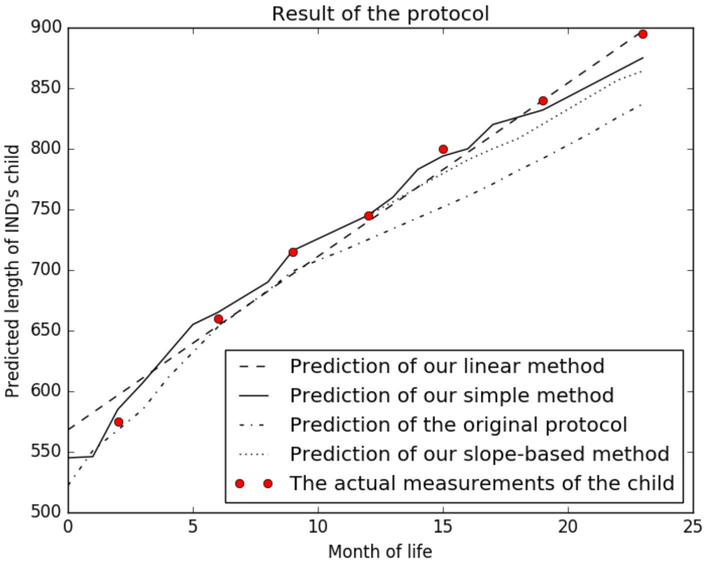


Figure 10: An example graph depicting the accuracy of predictions of a specific child's length generated by different protocols.

4.3 Comparison of Security

We compare the security of our protocols by comparing what variables are kept secure by the respective protocols (we will only mention variables that are not kept secure by a protocol, thus if a value is not mentioned here it means that it is secure in each protocol), and by stating whether a protocol has computational or unconditional security.

Firstly, we consider the original protocol. The most notable variables that are available in plaintext here are the comparison metrics, but as discussed

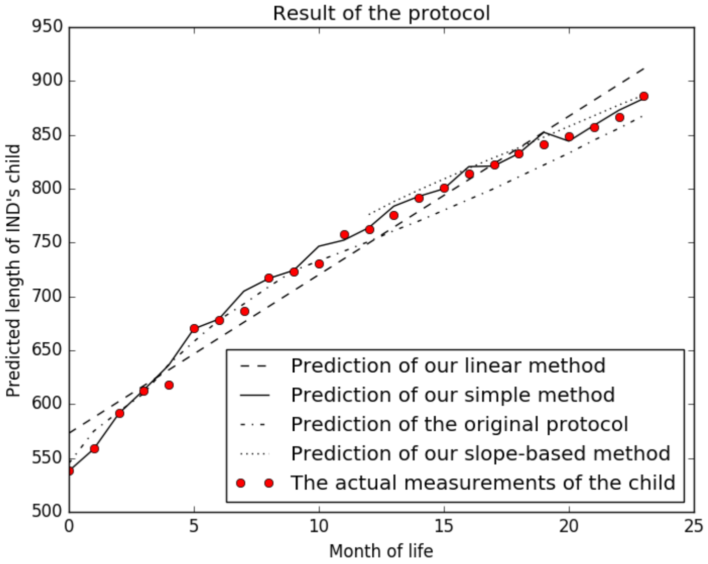


Figure 11: A graph depicting the average accuracy of predictions of a child's length generated by different protocols.

in Section 3.1, these variables reveal no additional privacy-sensitive information about the data, and thus do not impair the security of the protocol. Other protocols that we use do also make use of unencrypted comparison metrics. This is the only variable that is available to all parties unencrypted in this protocol. Furthermore, the decryption server obtains the noisy result, but since only IND knows what the noise is, this reveals nothing about the data. Finally, the security achieved by using Paillier is computational, if an adversary would be able to find the prime factorization of the public key, he would be able to derive the private key. There are no more variables in this protocol that are available in plaintext, therefore we move on to the next protocol, the 'simple' garbled circuits protocol.

In the 'simple' garbled circuits protocol, all 'auxiliary' variables that are used as input for the protocol are available in plaintext to both parties. This includes the least common multiple of $\{1, 2, ..., n\}$, where n is the number of children considered in the protocol. Since the number of children that is selected from DAT is decided upon beforehand by both parties, this number can be derived by both parties, and therefore isn't considered privacy-sensitive information. For the rest, general information that is needed in garbled form for the protocol is available in plaintext to both parties, such as a number for each month and the least common multiple divided by $1, 2, ..., n$ respectively. The result multiplied by the least common multiple of $\{1, 2, ..., n\}$ is only plainly available to IND, which is the intention of the protocol. If the result would also be available to DAT, he could derive the values of the child of IND using the measurements of the children that were taken into the account in the protocol out of DAT. To prevent this, the plain result is not made available to DAT. Regarding the security that is achieved by using the "simple" garbled circuits protocol, we can say that it is computational, since the semi-honest security model for our garbled circuits method imposes a computational bound on adversaries, without which our protocols wouldn't be secure in the semi-honest sense.

For the 'slope-based' garbled circuits protocol, the same variables are used as in the 'simple' garbled circuits protocol, thus its security is also the same.

Looking at the 'linear' garbled circuits protocol, we see that the same variables as in the 'simple' garbled circuits protocol are available in plaintext to both parties. As mentioned above, this is not a problem. Furthermore, for this method, IND obtains both the numerator and denominator of the slope and the intercept of the line that predicts the length of the child of IND in plaintext, since we chose to not do division within the garbled circuits because of its computational cost, and because it doesn't hide significant additional information. From these values, he then constructs the slope and intercept himself by dividing the numerators by the corresponding denominators. Since IND does not know the measurements of the children out of DAT that are used in the protocol, he cannot derive any additional information out of the numerators and denominators of the slope and intercept that he couldn't have derived from the slope and intercept themselves. Since the garbled circuits method that is used is the same as in the 'simple' garbled circuits protocol, the security that is achieved by using this method is also computational.

Considering the modified versions of the 'simple', 'slope-based' and the 'linear' garbled circuits protocols, we see that the same variables are available in plaintext to both parties as in the 'simple' and 'linear' garbled circuits protocols respectively. In addition to that, the number of children that have a measurement in a certain month, and information on whether any child has a measurement in a certain month or not, are converted to another secure representation, boolean circuits. Since this secure representation is also secure in the semi-honest sense, the protocols as a whole stays secure in the semi-honest sense. The security we achieve using this protocol is computational, since we still make use of the same garbled circuits method as in the two previous protocols in the most (vital) parts of our protocols.

It should be noted that theoretically, it would be possible to design garbled circuits for these methods that are secure against malicious adversaries, however this would cost a considerable amount of time complexitywise, and would also make the circuits much harder to construct. Furthermore, since this is not supported by the framework we currently use to implement the garbled circuits, we did not attempt to construct such circuits, but this could be an improvement in future work.

Finally, we look at the alternative homomorphic encryption protocols. In the 'simple' protocol, the most notable variables that are available in plaintext are the least common multiple of $\{1, 2, ..., n\}$ and n_j, where n_j is the amount of children in month j that contributed a measurement to the average length. both are available to DAT. As we discussed before, the least common multiple of $\{1, 2, ..., n\}$ can be derived by both parties because the number of children that is selected by DAT is decided upon beforehand. Therefore, this isn't considered privacy-sensitive information. Regarding the n_j, we see that DAT can derive in which month the child of IND has a measurement. However, since DAT doesn't get access to the plain results of the protocol, he cannot derive anything about the actual values of the measurements of IND from this, and thus this does not reveal any privacy-sensitive information to DAT.

In the 'linear' protocol, we have access to the same variables in plaintext as in the 'simple' protocol, in addition to the sums of which the denominator and the numerator for the slope and intercept are built: the sum of the averages, the sum of the contributing months, the sum of the squared contributing months, the sum of the contributing months multiplied by their corresponding average and the number of contributing months. These sums are only available in plaintext to IND, who uses them to construct the numerator and denominator for the slope and the intercept. IND can't derive any additional privacy-sensitive information from these sums, since he only knows the data of his own child. Therefore, he can derive something about in which months there are measurements for some of the children of DAT, but he can't derive the values of those measurements, and thus the security of this protocol is not impaired by this information being available in plaintext to IND.

Regarding the security of these two protocols, we can say that it is computational, since the security of the DGK cryptosystem is based on the strong RSA subgroup assumption, as described in Section 2.2. Therefore, given a RSA subgroup pair (N, g), if it would be possible to find $u, w \in \mathbb{Z}_N^*$ and

$d, e > 1$ such that $g \equiv uw^e \pmod{N}$ and $u^d \equiv 1 \pmod{N}$. Currently, this is infeasible, but this is based on the amount of computational power that currently can be generated in a reasonable time, thus this security is computational.

5 Conclusions

In this chapter, we present the conclusions we obtained from our research as presented in the previous two chapters. First, we will look at which protocol is the preferred protocol in our setting and what implementation is its preferred implementation, based on the comparisons we performed in the previous chapter. Then, we present a few directions in which future research in this area could be done.

5.1 Preferred Protocol

Considering the comparisons we performed in the previous chapter, we have seen that the security is similar for the different protocols: the way in which the security is achieved differs from protocol to protocol, but its strength is computational in each case. There is a difference in what information exactly is available to each party in plaintext, but since there is no sensitive information available in plaintext in every case, and there also is no sensitive information that can be derived from the information that is available in plaintext, this is only a small factor when deciding between protocols and implementations.

Regarding the accuracy of the methods, we saw that the simple regression protocols provided the most accurate results on average, therefore this would lead us to the conclusion that this protocol should be used if only regarding accuracy. However, since other factors are also taken into account, this is not necessarily the case. It should be noted though that the accuracy of the protocol is a very important factor in determining the preferred protocol, since the preferred protocol should provide us with good predictions for the growth curve that we want to predict, and this is dependent on the accuracy of the protocol.

Finally, we take the speed of the protocols into account. We saw that when using the same implementation, the 'slope' protocols were always the fastest, then the 'simple' protocols, then the 'linear' protocols and finally the 'original' protocol. However, the relative difference in speed between the first three methods was small for each implementation, as can be seen in Table 2.

When considering all these factors, we came to the conclusion that for our setting, the 'simple' regression protocols were optimal: they provided the best accuracy of the methods that we looked at, and given the difference in accuracy and the difference in speed between the two types of protocols that were the best candidates (the ‘simple’ regression method and the ‘slope’ regression method), we concluded that the relatively larger difference in accuracy outweighs the difference in speed between the two methods. However, it should be noted that in a setting where we would value speed much higher than accuracy, the ‘slope’ regression method could be preferred over the ‘simple’ regression method.

When determining what implementation of the ‘simple’ regression protocol would be preferred, we once again take a look at the comparisons that are performed in the previous chapter: regarding the security, we see that for the alternative homomorphic encryption protocols, more variables are available in plaintext than for the garbled circuit protocols. However, since the data in these variables is not privacy-sensitive, this has limited influence on what method we prefer.

Considering the accuracy of the protocols, we implemented each protocol in the same way regarding the calculations that are performed, only the way in which they are done within the secure domain differs. Therefore, each protocol which implements the ‘simple’ regression has the same accuracy.

Looking at the speed of the protocols, we see in Table 2 that the modified garbled circuits provide the fastest implementation of the ‘simple’ regression. Since the other two factors are similar for all implementations, we prefer the modified garbled circuits implementation over the other implementations of ‘simple’ regression.

Thus, the preferred protocol in our setting is the ‘simple’ regression protocol we presented, implemented using modified garbled circuits.

5.2 Future Work

Throughout this thesis, we have mentioned some possibilities that we did consider but didn’t implement or didn’t look at in detail, we summarize those possibilities here along with some suggestions that we didn’t mention before, in order to guide future research within this subject.

When we created the modified garbled circuits, which are designed to be faster than the regular garbled circuits, there was another optimization that we couldn’t get to work, but which according to the ABY-framework we used should be possible to do, namely the usage of arithmetic circuits to compute the MUL-gates. We expect that this would improve the speed of our modified garbled circuit protocols by about ten percent, but applying and testing this is left as a future improvement.

It would also be possible to construct the garbled circuits in such a way that they would be secure against malicious adversaries rather than just semi-honest adversaries. Since our main concern was the speed of the protocols while preserving a certain level of security, and since this modification would certainly slow our protocols, we didn't consider it here, but it is certainly an interesting direction for future work.

Another direction that could be explored further is the different regression protocols that are considered. Here, we considered four different types of protocols, but there exist many more: a future direction could be to explore whether other promising regression protocols for our setting yield a better performance than our current best performance. It is important to note that in order to get comparable results, the setup for the experiments should be similar. To that end, we provide the specifications of the hard- and software that we used in Section 7.12.

It could also be interesting to try and optimize the methods while putting a higher importance on the accuracy of the obtained predictions instead of on the speed of the protocols. If this would be done, we suggest based on Figure 9 that ten children should be selected based on the comparison metrics, instead of the five children that we select here, because the accuracy we gain would in that setting be worth the speed of the protocols that we lose.

Finally, as mentioned in Section 3.3, a protocol could be constructed in which DAT also could be allowed to receive the results without revealing any privacy-sensitive information. This could be done by sharing the data of the children from DAT in an oblivious way, so DAT doesn't know what children were selected and thus can't reproduce the experiment without the data of IND's child and then compare the result to the result of the experiment with all the data. However, this is by no means the only way to achieve this. For our setting it was not necessary to share the result with both parties, but in a different setting this could be desirable, therefore it could be an interesting direction for future work.

6 References

[1] Secomlib. `https://mihaitodor.github.io/SeComLib/`.

[2] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *arXiv preprint arXiv:1704.03578*, 2017.

[3] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A Reuter, and Martin Strand. A guide to fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2015:1192, 2015.

[4] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM, 1990.

[5] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, volume 7417, pages 868–886. Springer, 2012.

[6] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian Conference on Information Security and Privacy*, pages 416–430. Springer, 2007.

[7] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[8] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[9] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[10] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[11] Jens Groth. Cryptography in subgroups of $\mathbb{Z}_N^*$. In *Theory of Cryptography Conference*, pages 50–65. Springer, 2005.

[12] Rob Hall, Stephen E Fienberg, and Yuval Nardi. Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics*, 27(4):669, 2011.

[13] WP Herngreen, JD Reerink, BM van Noord-Zaadstra, SP Verloover-Vanhorick, and JH Ruys. Smocc: Design of a representative cohort-study of live-born infants in the netherlands. *The European Journal of Public Health*, 2(2):117–122, 1992.

[14] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[15] Alan F Karr, Xiaodong Lin, Ashish P Sanil, and Jerome P Reiter. Secure regression on distributed databases. *Journal of Computational and Graphical Statistics*, 14(2):263–279, 2005.

[16] Sam Kennerly. A crash course in group theory, part 1: Finite groups.

[17] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[18] M.X. Makkes. Efficient implementations of homomorphic cryptosystems. 2010.

[19] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[20] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 334–348. IEEE, 2013.

[21] Pascal Paillier et al. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, volume 99, pages 223–238. Springer, 1999.

[22] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. Secure two-party computation is practical. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 250–267. Springer, 2009.

[23] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[24] Aleksandra B Slavkovic, Yuval Nardi, and Matthew M Tibbits. " secure" logistic regression of horizontally and vertically partitioned distributed

databases. In *Data Mining Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on*, pages 723–728. IEEE, 2007.

[25] Ben Terner. A survey of garbled circuit techniques. 2014.

[26] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[27] Stef Van Buuren. Curve matching: a data-driven technique to improve individual prediction of childhood growth. *Annals of Nutrition and Metabolism*, 65(2-3):227–233, 2014.

[28] Thijs Veugen. Gevoelige informatie delen, zonder de voordeur open te zetten. `https://time.tno.nl/nl/artikelen/gevoelige-informatie-delen-zonder-de-voordeur-open-te-zetten/`, 2018.

[29] Thymen Wabeke and Wessel Kraaij. Da.3 proof of principle of an approach based on bringing the algorithms to the data – patient setting. 2017.

[30] Sophia Yakoubov. A gentle introduction to Yao's garbled circuits. 2017.

[31] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[32] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

7 Appendix

In this chapter, we provide part of the source code that we used to implement the garbled circuit protocols, and present some theory that is either considered preliminary or extra. We also provide the specifications of the setup we used to do our experiments.

7.1 Regression

In this section, we shed some light on the types of regression and the related terms we use in our work.

7.1.1 General terms

Throughout this thesis, we talk about a few different variations of regression. In this subsection we define what we mean by regression, and shed some light on a few regression related terms we used.

We define regression in the following way: it is a method to fit a curve through a set of points (measurements) using some criterion to measure how well a curve fits to the points. The criterion we use depends on the type of regression we do, and the end result of a regression function is the curve that fits best to the points following the criterion we use. The criterion is chosen based on what form we expect the relation to take: if we expect it to be linear, we do some kind of linear regression, if we expect it to be quadratic, we do some kind of quadratic regression, and so forth.

Within this thesis we use some other regression related terms that might be unknown to the reader, we discuss the specific regression methods we mention in their own subsections, and explain two general terms here using Figure 12. Looking at this figure, we see a line, of which the equation is given to be $y = ax + b$. We see that b is the intersection between the line and the y-axis, this is defined to be the intercept. We also see in the figure that when the x-coordinate of the line increases by one, the line goes up in the y-direction by a. The difference in the y-direction that the line goes through when it traverses a distance of one in the x-direction, a, is defined to be the slope of the line. Given the slope and the intercept of a line, we can plot it like we did in Figure 12.

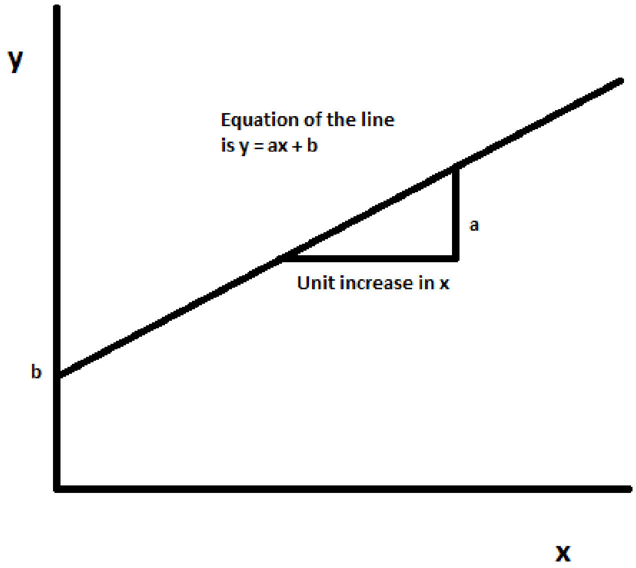


Figure 12: An example of a graph with its equation included, to introduce the terms 'slope' and 'intercept'

7.1.2 Linear Regression

Now that we have defined some basic terms, we can take a look at what linear regression consists of. In general, we have a number of measurements spread through time, and want to model the relationship between a dependent variable and one or more explanatory variables. This model will take the form of a line in a n-dimensional space, where n is the amount of variables in the model, which predicts future measurements that are done in the same setting. Since we want to find the best fitting line based on the measurements that we have, we want to minimize the distance between each measurement and the line which predicts the future measurements. This can be done by a few different estimators, which try to minimize the sum of the squares of the distance of the measurements to the line in some way. In our protocol, we do this by using the following formulas:

$$b = intercept = \frac{\sum_j(AVG_j) * \sum_j(j^2) - \sum_j(j) * \sum_j(j * AVG_j)}{n * \sum_j(j^2) - (\sum_j(j))^2} \tag{3}$$

$$a = slope = \frac{n * \sum_j(j * AVG_j) - \sum_j(j) * \sum_j(AVG_j)}{n * \sum_j(j^2) - (\sum_j(j))^2} \tag{4}$$

Here, j is the month of life a measurement is made in and n is the total number of different months in which we have measurements.

7.1.3 Piecewise Linear Regression

Another regression method we perform that is similar to linear regression is piecewise linear regression, we do a specific variant of this method which we call 'pairwise' linear regression. This method consists of doing multiple instances of linear regression, between each two following measurements regarding x-coordinate (assuming we only have one explanatory variable and one dependent variable). An example of applying this method can be found in Figure 13. We see that between every two closest points, linear regression is done. Since linear regression between two points always generates a line between those two points (since that is the line that minimizes the least squared errors between the measurements and the line to 0), we get a graph where each two following points are connected by a straight line. Since we perform this method using average measurements in our protocols, this yields a reasonable prediction, which we talk about in more detail in Section 4.2.

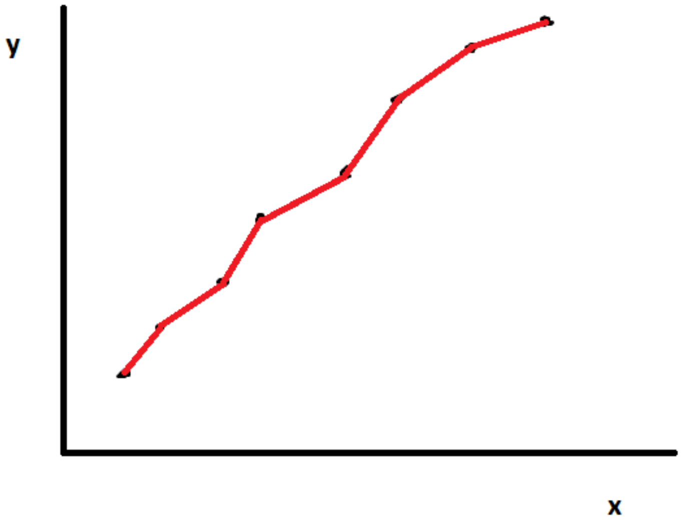


Figure 13: An example of the piecewise (pairwise) linear regression we perform in our 'simple' protocol

7.1.4 Slope-based regression

Here, we look at a regression method which is based on the availability of curves about similar objects to the one for which we are trying to construct a curve here. Since it is based on the slopes of those curves, we call this type of regression 'slope-based' regression. It can be done as follows: we select curves of similar objects out of a large set of such curves using a criterion determining how close the development of an object is to the development of the object we are trying to construct a curve for. Then, we take small steps, averaging the slope of all selected curves in each step, and applying the obtained slope in the same time step where it was obtained while constructing the curve. Our starting point is the latest measurement which we have for the object we are trying to construct a curve for. This is illustrated in Figure 14: there we see green points, which represent the measurements we have for the object for which we are trying to construct a curve here. The black curve and the brown curve represent curves of similar objects that were selected to base the slope of the curve we want to construct on. This curve is then constructed by averaging the slopes of the black and brown curve in every time step, this is depicted as the green curve. Thus, we see we can construct a curve in this way, given that we have curves of similar object available, and given that we have a way to determine whether an object is similar to the object for which we want to construct a curve.

7.2 Arithmetic shares

Here, we provide a description of the Arithmetic sharing out of the ABY framework. In Arithmetic sharing, a k-bit value x can be shared additively in the ring of integers modulo 2^l as the sum of two values: $x = \langle x \rangle_1 + \langle x \rangle_2$, where a value between $\langle\rangle$ denotes an arithmetic share. Assume now that we want to compute $\langle z \rangle = \langle x \rangle *^A \langle y \rangle$, where $\langle x \rangle, \langle y \rangle$ are shared between party 1 and 2 (they have $\langle x \rangle_1, \langle y \rangle_1$ and $\langle x \rangle_2, \langle y \rangle_2$ respectively), and $*^A$ denotes the multiplication of two arithmetic shares. This multiplication is then computed with the help of a precomputed Arithmetic multiplication triple: $\langle c \rangle = \langle a \rangle *^A \langle b \rangle$. This is done as follows: each party P_i sets $\langle e \rangle_i = \langle x \rangle_i - \langle a \rangle_i$ and $\langle f \rangle_i = \langle y \rangle_i - \langle b \rangle_i$. Both parties then perform reconstruction for both $\langle e \rangle$ and $\langle f \rangle$, which consists of them sending their part of $\langle e \rangle$ and $\langle f \rangle$ to the other party, so they can construct e and f. Then, each P_i sets $\langle z \rangle_i = (i-1) * e * f + f * \langle a \rangle_i + e * \langle b \rangle_i + \langle c \rangle_i$, which can be mutually reconstructed by both parties to obtain $\langle z \rangle$.

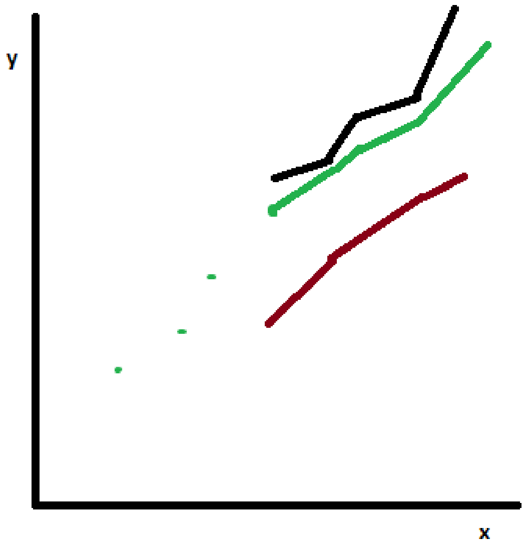


Figure 14: An example of the slope-based regression we perform in our ‘slope’ protocol

7.3 Simple Garbled Circuits Protocol

Here we provide the source code of the ‘simple’ garbled circuits, since all other protocols use the same structure in their test_circuit function, we will only include the functions in which they differ in their respective sections.

```
/**
 \file          GC_test.cpp
 \author        Laurens van der Beek
 \copyright     ABY − A Framework for Efficient Mixed−protocol Secure Two−party
        Computation Copyright (C) 2015 Engineering Cryptographic Protocols
        Group, TU Darmstadt. This program is free software: you can
        redistribute it and/or modify it under the terms of the GNU
        Affero General Public License as published by the Free Software
        Foundation, either version 3 of the License, or (at your option)
        any later version. This program is distributed in the hope that
        it will be useful, but WITHOUT ANY WARRANTY; without even the
        implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
        PURPOSE. See the GNU Affero General Public License for more
        details. You should have received a copy of the GNU Affero
        General Public License along with this program. If not, see
        <http://www.gnu.org/licenses/>.
 \brief         Implementation of the child length prediction protocol, which
        uses garbled circuits, using ABY Framework.
 */

#include "GC_test.h"

int32_t test_GC_test_circuit(e_role role, char* address, uint16_t port,
seclvl seclvl, uint32_t nvals, uint32_t bitlen, uint32_t nthreads,
e_mt_gen_alg mt_alg, e_sharing sharing) {

        /**
                Step 1: Create the ABYParty object which defines the basis of all
        the     operations which are happening. Operations performed are on the
        basis of the role played by this object.
        */
        // the amount of children considered in the protocol
        int n_c = 6;
        ABYParty* party = new ABYParty(role, address, port, seclvl, bitlen,
        nthreads, mt_alg);


        /**
        Step 2: Get to know all the sharing types available in the program.
        */

        vector<Sharing*>& sharings = party->GetSharings();

        /**
        Step 3: Create the circuit object on the basis of the sharing type
        being inputed.
        */
        Circuit* circ = sharings[sharing]->GetCircuitBuildRoutine();


        /**
        Step 4: Creating the share objects - s_IND_l, s_IND_m, s_DAT_i_l and
        s_DAT_i_m which are used as input to the computation function,
        s_0 to s_6 which store the values with which measurements should be
        scaled depending on how many measurements there are for a certain
        month. Also s_out which stores the output.
        */
        uint32_t binsize = 24;
        share *s_IND_l, *s_DAT_1_l, *s_DAT_2_l, *s_DAT_3_l, *s_DAT_4_l,
        *s_DAT_5_l, *s_0, *s_1, *s_2, *s_3, *s_4, *s_5, *s_6, *scl_0, *scl_1,
        *scl_2, *scl_3, *scl_4, *scl_5, *scl_6;
        share** s_out = (share**) malloc(sizeof(share*) * binsize);
        /**
        Step 5: Initialize the input of IND and DAT.
        */

        int a;
	uint32_t b = 0, c = 0, d = 0;
	uint32_t output[24] = {};
	/** The months in which a measurement is performed is stored in m, the
	length measurements are stored in l. We have 6 rows since there are 6
	children to be considered. The first child is the child of IND.
	*/
	uint32_t l[n_c][24] = {};

	std::fstream myfile(”GC/common/RelevantChildren.txt”, std::ios_base::in);
	std::map<int,uint8_t> childID;
	while (myfile >> a)
	{
		/** We identify the meaning of a number from the file by its
		position and process it accordingly. Each line has 3 values, the
		first of which is the childID.
		*/
		if(b % 3 == 0){
			// If the childID has not been mapped yet
			if(childID.find(a) == childID.end()){
				childID.insert(std::pair<int,uint8_t>(a,c));
				c += 1;
			}
		}
		/** We know the second entry is the month in which the measurement
		was performed, thus we store it in an array corresponding to
		the child the measurement is from.
		*/
		if(b % 3 == 1){
			/** We know from preprocessing that the maximum month
			value is 24 and the minimum value is 0, therefore we
			map the 25th month to the 24th month, as is intended.
			*/
			if(a!=24){
				d = a;
			}
			else{
				d = 23;
			}
		}
		/** We know the third entry is the measurement of the length
		of the child with childID c-1 performed in month d. In
		the rare case that we have two measurements in the same
		month for one child, we average them to obtain the
		contribution for that child to the prediction of the value
		for the child of IND in that month.
		*/
		if(b % 3 == 2){
			if(l[c-1][d] != 0){
				l[c-1][d] = (l[c-1][d] + a)/2;
```

```
            }
            else{
                l[c-1][d] = a;
            }
        }
        b += 1;
    }
    /**
    Step 6: Copy the month and length of the measurements into the
    respective share objects using the circuit object method
    PutSIMDINGate(), while mentioning who is sharing the object.
    */
    s_IND_l = circ->PutSIMDINGate(24, l[0], bitlen, SERVER);
    s_DAT_1_l = circ->PutSIMDINGate(24, l[1], bitlen, CLIENT);
    s_DAT_2_l = circ->PutSIMDINGate(24, l[2], bitlen, CLIENT);
    s_DAT_3_l = circ->PutSIMDINGate(24, l[3], bitlen, CLIENT);
    s_DAT_4_l = circ->PutSIMDINGate(24, l[4], bitlen, CLIENT);
    s_DAT_5_l = circ->PutSIMDINGate(24, l[5], bitlen, CLIENT);
    /** Technically speaking these values are not privacy sensitive and
    known to both parties, so they are shared input, however we
    define that the server inputs them into the protocol.
    */
    uint32_t s0 = 0, s1= 1, s2 = 2, s3 = 3, s4 = 4, s5 = 5, s6 = 6, scl0 = 0,
    scl1 = 60, scl2 = 30, scl3 = 20, scl4 = 15, scl5 = 12, scl6 = 10;
    s_0 = circ->PutINGate(s0, bitlen, SERVER);
    s_1 = circ->PutINGate(s1, bitlen, SERVER);
    s_2 = circ->PutINGate(s2, bitlen, SERVER);
    s_3 = circ->PutINGate(s3, bitlen, SERVER);
    s_4 = circ->PutINGate(s4, bitlen, SERVER);
    s_5 = circ->PutINGate(s5, bitlen, SERVER);
    s_6 = circ->PutINGate(s6, bitlen, SERVER);
    scl_0 = circ->PutINGate(scl0, bitlen, SERVER);
    scl_1 = circ->PutINGate(scl1, bitlen, SERVER);
    scl_2 = circ->PutINGate(scl2, bitlen, SERVER);
    scl_3 = circ->PutINGate(scl3, bitlen, SERVER);
    scl_4 = circ->PutINGate(scl4, bitlen, SERVER);
    scl_5 = circ->PutINGate(scl5, bitlen, SERVER);
    scl_6 = circ->PutINGate(scl6, bitlen, SERVER);

    /**
    Step 7: Call the build method for building the circuit for the
    problem by passing the shared objects and circuit object.
    Don't forget to type cast the circuit object to type of share
    Since we know that we have to perform the same operation for
    each month,      we programmed one circuit which we built for each
    month.
    */
    uint32_t val[24][1] = {{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},
    {12},{13},{14},{15},{16},{17},{18},{19},{20},{21},{22},{23}};
```

```
for(int i=0; i < 24; i++){
        s_out[i] = BuildRegressionCircuit(s_IND_l, s_DAT_1_l, s_DAT_2_l,
        s_DAT_3_l, s_DAT_4_l, s_DAT_5_l, s_0, s_1, s_2, s_3, s_4, s_5,
        s_6, scl_0, scl_1, scl_2, scl_3, scl_4, scl_5, scl_6,
        (BooleanCircuit*) circ, val[i]);
}

/**
Step 8: Modify the output receiver based on the role played by
the server and the client. This step writes the output to the
shared output object based on the role.
*/
for(int i = 0; i < 24; i++){
        s_out[i] = circ->PutOUTGate(s_out[i], SERVER);
}

/**
Step 9: Executing the circuit using the ABYParty object evaluate the
problem.
*/
party->ExecCircuit();

/**
Step 10:Only IND has acces to the output, and can do something with
that output.
*/
if(role==SERVER){
        ofstream file;
        /** Each time we run the program, we want to start with an empty
        file.
        */
        file.open("GC/common/resultsGC.txt", std::ofstream::out |
        std::ofstream::trunc);
        for(int i = 0; i < 24; i++){
                output[i] = (s_out[i]->get_clear_value<uint32_t>())/60;
                if(output[i] != 0){
                        /** A child can only grow, thus if we would obtain
                        a month in which a decrease in length compared to
                        a previous month occurs, for example because of th
                        averaging between multiple different children, we
                        don't take the length in that month into account.
                        */
                        if(i >= 1){
                                for(int j = 0; j < i; j++){
                                if(output[i] > output[j]){
                                if(j == i-1){
                                cout <<
                                "The␣predicted␣length␣of␣the␣child␣in␣mont
```

```
                    << i << "␣is:\t" << output[i] << "\n";
                    file << i << "\t" << output[i] << "\n";
                    }
                    }
                    else{
                        continue;
                    }
                    }
                }
                else{
                cout << "The␣predicted␣length␣of␣the␣child␣in␣mont
                << i << "␣is:\t" << output[i] << "\n";
                file << i << "\t" << output[i] << "\n";
                }
            }
        }
            file.close();
    }
    free(s_out);
    delete party;
    return 0;
}

share* BuildRegressionCircuit(share *s_IND_l, share *s_DAT_1_l,
share *s_DAT_2_l, share *s_DAT_3_l, share *s_DAT_4_l, share *s_DAT_5_l,
share *s_0, share *s_1, share *s_2, share *s_3, share *s_4, share *s_5,
share *s_6, share *scl_0, share *scl_1, share *scl_2, share *scl_3,
share *scl_4, share *scl_5, share *scl_6, BooleanCircuit *bc,
uint32_t* month) {

    share* out, *c_0, *c_1, *c_2, *c_3, *c_4, *c_5, *gtcheck_0, *gtcheck_1,
    *gtcheck_2, *gtcheck_3, *gtcheck_4, *gtcheck_5, *sum, *mult;
    /** We need to store the arrayindices for the SubsetGate in the manner
    below, to comply with the expected input of PutSubsetGate.
    */
    uint32_t vals_out = 1;
    /** checking whether there is a measurement for child i in the current
    month, if so, it is stored in c_i, otherwise 0 is stored in m_i_j.
    */
    c_0 = bc->PutSubsetGate(s_IND_l, month, vals_out);
    gtcheck_0 = bc->PutGTGate(c_0, s_0);
    c_1 = bc->PutSubsetGate(s_DAT_1_l, month, vals_out);
    gtcheck_1 = bc->PutGTGate(c_1, s_0);
    c_2 = bc->PutSubsetGate(s_DAT_2_l, month, vals_out);
    gtcheck_2 = bc->PutGTGate(c_2, s_0);
    c_3 = bc->PutSubsetGate(s_DAT_3_l, month, vals_out);
    gtcheck_3 = bc->PutGTGate(c_3, s_0);
    c_4 = bc->PutSubsetGate(s_DAT_4_l, month, vals_out);
    gtcheck_4 = bc->PutGTGate(c_4, s_0);
```

```
        c_5 = bc->PutSubsetGate(s_DAT_5_l, month, vals_out);
        gtcheck_5 = bc->PutGTGate(c_5, s_0);
        // sum is the amount of measurements performed in this month
        sum = bc->PutADDGate(gtcheck_0, gtcheck_1);
        sum = bc->PutADDGate(sum, gtcheck_2);
        sum = bc->PutADDGate(sum, gtcheck_3);
        sum = bc->PutADDGate(sum, gtcheck_4);
        sum = bc->PutADDGate(sum, gtcheck_5);
        /** we now check how large sum_0 is, and scale the values from month 0
        based on it.
        */
        gtcheck_0 = bc->PutGTGate(sum, s_0);
        mult = bc->PutMUXGate(scl_1, scl_0, gtcheck_0);
        gtcheck_1 = bc->PutGTGate(sum, s_1);
        /** if sum_0 was larger than s_1, we know that there were at least 2
        measurements in the current month, thus we set mult to the
        corresponding scale. Whenever sum_0 is equal to s_i, the
        corresponding GTGate will output 0, and thus the MUXGate will set
        mult equal to itself, therefore the correct scale is obtained by
        following this procedure 6 times.
        */
        mult = bc->PutMUXGate(scl_2, mult, gtcheck_1);
        gtcheck_2 = bc->PutGTGate(sum, s_2);
        mult = bc->PutMUXGate(scl_3, mult, gtcheck_2);
        gtcheck_3 = bc->PutGTGate(sum, s_3);
        mult = bc->PutMUXGate(scl_4, mult, gtcheck_3);
        gtcheck_4 = bc->PutGTGate(sum, s_4);
        mult = bc->PutMUXGate(scl_5, mult, gtcheck_4);
        gtcheck_5 = bc->PutGTGate(sum, s_5);
        mult = bc->PutMUXGate(scl_6, mult, gtcheck_5);
        /** now we have the scaling factor that should be applied to this month,
        thus we apply it.
        */
        c_0 = bc->PutMULGate(c_0, mult);
        c_1 = bc->PutMULGate(c_1, mult);
        c_2 = bc->PutMULGate(c_2, mult);
        c_3 = bc->PutMULGate(c_3, mult);
        c_4 = bc->PutMULGate(c_4, mult);
        c_5 = bc->PutMULGate(c_5, mult);
        /** we have now obtained the scaled measurements for this month (test
        output). When we divide this by 60, we will obtain the actual mean
        value independent of the amount of measurements we had for this month,
        because of the way we securely scale our values.
        */
        sum = bc->PutSUBGate(sum, sum);
        sum = bc->PutADDGate(c_0, c_1);
        sum = bc->PutADDGate(sum, c_2);
        sum = bc->PutADDGate(sum, c_3);
        sum = bc->PutADDGate(sum, c_4);
        out = bc->PutADDGate(sum, c_5);
        return out;
}
```

7.4 Slope-based Garbled Circuits Protocol

The source code of the ‘slope-based’ garbled circuits is almost the same as the source code of the ‘simple’ garbled circuits which we saw in the previous section. The only difference between the two (apart from the fact that different comparison metrics are used, but this only affects the preprocessing and does not impact the code for the garbled circuits), is in step ten of the main method (test_GC_test_circuit), where we don’t have to perform a sanity check which ensures that each month, the prediction for the length of a child can only rise compared to the previous month, since children can only grow, not shrink. Since we look at how rapid a child grows each month here, and not at the value of the length of the child in a certain month, this sanity check is not needed (and even unwanted). Below, we include only step ten of the main method, since the rest of the code is the same as in the previous section.

```
/**
        Step 10:Only IND has acces to the output, and can do something with
        that output.
*/
if(role==SERVER){
        ofstream file;
        // Each time we run the program, we want to start with an empty file.
        file.open("/home/lvdbeek/ABY/src/examples/rcGC/common/resultsGC.txt",
        std::ofstream::out | std::ofstream::trunc);
        for(int i = 0; i < 24; i++){
                output[i] = (s_out[i]->get_clear_value<uint32_t>())/60;
                if(output[i] != 0){
                        /* A child can only grow, thus if we would obtain a month
                        in which a decrease in length compared to a previous
                        month occurs, for example because of the averaging between
                        multiple different children, we don't take the length in
                        that month into account.*/
                        cout << "The␣predicted␣length␣of␣the␣child␣in␣month␣" << i
                        << "␣is:\t" << float(output[i])/12 << "\n";
                        file << i << "\t" << float(output[i])/12 << "\n";
                }
        }
                file.close();
}
free(s_out);
```

```
delete party;
return 0;
```

7.5 Linear Garbled Circuits Protocol

For the 'linear' Garbled Circuits Protocol, we only provide the source code of the function which constructs the linear circuit, all the code from Section 7.3 is also used in the protocol (since the linear circuit is built upon the simple circuit). We cannot use a for-loop here because each secure variable needs to be declared explicitly as such.

```
share* BuildLinearRegressionCircuit(share *m_0, share *m_1, share *m_2,
share *m_3, share *m_4, share *m_5, share *m_6, share *m_7, share *m_8,
share *m_9, share *m_10, share *m_11, share *m_12, share *m_13, share *m_14,
share *m_15, share *m_16, share *m_17, share *m_18, share *m_19, share *m_20,
share *m_21, share *m_22, share *m_23, share *s_0, share *s_1, share *s_2,
share *s_3, share *s_4, share *s_5, share *s_6, share *s_7, share *s_8,
share *s_9, share *s_10, share *s_11, share *s_12, share *s_13, share *s_14,
share *s_15, share *s_16, share *s_17, share *s_18, share *s_19, share *s_20,
share *s_21, share *s_22, share *s_23, BooleanCircuit *bc, uint32_t* value,
share *s_012){

        share* out, *sum_y, *sum_x, *xy, *sum_xy, *x2, *x2contrib, *x2sum,
        *gtcheck, *contrib, *n, *a_t, *ab_n, *b_t, *s012;
        uint32_t vals_out = 1;
        // First we compute the sum of the average lengths in each month
        sum_y = bc->PutADDGate(m_0, m_1);
        sum_y = bc->PutADDGate(sum_y, m_2);
        sum_y = bc->PutADDGate(sum_y, m_3);
        sum_y = bc->PutADDGate(sum_y, m_4);
        sum_y = bc->PutADDGate(sum_y, m_5);
        sum_y = bc->PutADDGate(sum_y, m_6);
        sum_y = bc->PutADDGate(sum_y, m_7);
        sum_y = bc->PutADDGate(sum_y, m_8);
        sum_y = bc->PutADDGate(sum_y, m_9);
        sum_y = bc->PutADDGate(sum_y, m_10);
        sum_y = bc->PutADDGate(sum_y, m_11);
        sum_y = bc->PutADDGate(sum_y, m_12);
        sum_y = bc->PutADDGate(sum_y, m_13);
        sum_y = bc->PutADDGate(sum_y, m_14);
        sum_y = bc->PutADDGate(sum_y, m_15);
        sum_y = bc->PutADDGate(sum_y, m_16);
        sum_y = bc->PutADDGate(sum_y, m_17);
        sum_y = bc->PutADDGate(sum_y, m_18);
        sum_y = bc->PutADDGate(sum_y, m_19);
        sum_y = bc->PutADDGate(sum_y, m_20);
        sum_y = bc->PutADDGate(sum_y, m_21);
        sum_y = bc->PutADDGate(sum_y, m_22);
        sum_y = bc->PutADDGate(sum_y, m_23);
        /** Then, we compute the amount of months which contribute a measurement,
        the sum of the months, the sum of the months multiplied with the length
        for each month, and the squared sum of the months. Gtcheck determines
        whether there is a contribution from month i to the result, if so we
        take its values into account using contrib and x2contrib.
        */
        gtcheck = bc->PutGTGate(m_0, s_0);
        n = bc->PutADDGate(s_0, gtcheck);
        gtcheck = bc->PutGTGate(m_1, s_0);
        contrib = bc->PutMUXGate(s_1, s_0, gtcheck);
        sum_x = bc->PutADDGate(s_0, contrib);
        xy = bc->PutMULGate(contrib, m_1);
        sum_xy = bc->PutADDGate(s_0, xy);
        x2 = bc->PutMULGate(s_1, s_1);
        x2contrib = bc->PutMUXGate(x2, s_0, gtcheck);
        x2sum = bc->PutADDGate(s_0, x2contrib);
        n = bc->PutADDGate(n, gtcheck);
        gtcheck = bc->PutGTGate(m_2, s_0);
        contrib = bc->PutMUXGate(s_2, s_0, gtcheck);
        sum_x = bc->PutADDGate(sum_x, contrib);
        xy = bc->PutMULGate(contrib, m_2);
        sum_xy = bc->PutADDGate(sum_xy, xy);
        x2 = bc->PutMULGate(s_2, s_2);
        x2contrib = bc->PutMUXGate(x2, s_0, gtcheck);
        x2sum = bc->PutADDGate(x2sum, x2contrib);
        n = bc->PutADDGate(n, gtcheck);
        // Similar operations are done for s_3 - s_22
        ...
        ...
        ...
        ...
        ...
        gtcheck = bc->PutGTGate(m_23, s_0);
        contrib = bc->PutMUXGate(s_23, s_0, gtcheck);
        sum_x = bc->PutADDGate(sum_x, contrib);
        xy = bc->PutMULGate(contrib, m_23);
        sum_xy = bc->PutADDGate(sum_xy, xy);
        x2 = bc->PutMULGate(s_23, s_23);
        x2contrib = bc->PutMUXGate(x2, s_0, gtcheck);
        x2sum = bc->PutADDGate(x2sum, x2contrib);
        n = bc->PutADDGate(n, gtcheck);

        // numerator for the intercept
        a_t = bc->PutMULGate(sum_y, x2sum);
        contrib = bc->PutMULGate(sum_x, sum_xy);
        a_t = bc->PutSUBGate(a_t, contrib);
        // denominator for both the intercept and the slope
        ab_n = bc->PutMULGate(n, x2sum);
        contrib = bc->PutMULGate(sum_x, sum_x);
        ab_n = bc->PutSUBGate(ab_n, contrib);
        // numerator for the slope
        b_t = bc->PutMULGate(n, sum_xy);
        contrib = bc->PutMULGate(sum_x, sum_y);
        b_t = bc->PutSUBGate(b_t, contrib);
        /** Since we can't output more than one value at a time, we output the one
        corresponding with a value: for 0 it is a_t, for 1 it is ab_n and for 2
        it is b_t.
        */
        s012 = bc->PutSubsetGate(s_012, value, vals_out);
        gtcheck = bc->PutGTGate(s012, s_0);
        out = bc->PutMUXGate(s_0, a_t, gtcheck);
        gtcheck = bc->PutGTGate(out, s_0);
        out = bc->PutMUXGate(out, ab_n, gtcheck);
        gtcheck = bc->PutGTGate(s012, s_1);
        out = bc->PutMUXGate(b_t, out, gtcheck);
        return out;
}
```

7.6 Modified Simple Garbled Circuits Protocol

Here, we only show the modified part of the source code from the BuildRegressionCircuit function, without the declaration of the boolean circuit, since this happens in the same way as the declaration of the garbled circuit. The modifications consist of the conversions from garbled circuits to boolean circuits and back (the “PutY2BGate” and “PutB2YGate”), and of the execution of MUX-gates within the boolean circuits instead of in the garbled circuits (this is indicated by “boolc→PutMUXGate” replacing “bc→PutMUXGate”).

```
        /** We now check how large sum_0 is, and scale the values from month 0
        based on it.
        */
        gtcheck_0 = bc->PutGTGate(sum, s_0);
        gtcheck_0 = boolc->PutY2BGate(gtcheck_0);
        mult = boolc->PutMUXGate(scl_1, scl_0, gtcheck_0);
        gtcheck_1 = bc->PutGTGate(sum, s_1);
        gtcheck_1 = boolc->PutY2BGate(gtcheck_1);
        /** if sum_0 was larger than s_1, we know that there were at least 2
        measurements in the current month, thus we set mult to the corresponding
        scale. Whenever sum_0 is equal to s_i, the corresponding GTGate will
        output 0, and thus the MUXGate will set mult equal to itself, therefore
        the correct scale is obtained by following this procedure 6 times.
        */
        mult = boolc->PutMUXGate(scl_2 , mult , gtcheck_1);
        gtcheck_2 = bc->PutGTGate(sum, s_2);
        gtcheck_2 = boolc->PutY2BGate(gtcheck_2);
        mult = boolc->PutMUXGate(scl_3 , mult , gtcheck_2);
        gtcheck_3 = bc->PutGTGate(sum, s_3);
        gtcheck_3 = boolc->PutY2BGate(gtcheck_3);
        mult = boolc->PutMUXGate(scl_4 , mult , gtcheck_3);
        gtcheck_4 = bc->PutGTGate(sum, s_4);
        gtcheck_4 = boolc->PutY2BGate(gtcheck_4);
        mult = boolc->PutMUXGate(scl_5 , mult , gtcheck_4);
        gtcheck_5 = bc->PutGTGate(sum, s_5);
        gtcheck_5 = boolc->PutY2BGate(gtcheck_5);
        mult = boolc->PutMUXGate(scl_6 , mult , gtcheck_5);
        mult = bc->PutB2YGate(mult);
```

7.7 Modified Slope-based Garbled Circuits Protocol

The modified ‘slope-based’ garbled circuits have the same source code as the modified ‘simple’ garbled circuits protocol, with the same change as specified in Section 7.4.

7.8 Modified Linear Garbled Circuits Protocol

For this protocol, we made the same modification as in the previous section, in addition to the modifications to the ‘BuildLinearRegressionCircuit’ function that are presented here:

```
        /** then, we compute the amount of months which contribute a measurement,
        the sum of the months, the sum of the months multiplied with the length
        for each month, and the squared sum of the months. Gtcheck determines
        whether there is a contribution from month i to the result, if so we
        take its values into account using contrib and x2contrib.
        */
        gtcheck = bc->PutGTGate(m_0, s_0);
        n = bc->PutADDGate(s_0 , gtcheck);
        gtcheck = bc->PutGTGate(m_1, s_0);
        n = bc->PutADDGate(n, gtcheck);
        gtcheck = boolc->PutY2BGate(gtcheck);
        contrib = boolc->PutMUXGate(sb_1 , sb_0 , gtcheck);
        contrib = bc->PutB2YGate(contrib);
        sum_x = bc->PutADDGate(s_0 , contrib);
        xy = bc->PutMULGate(contrib , m_1);
        sum_xy = bc->PutADDGate(s_0 , xy);
        x2 = bc->PutMULGate(s_1, s_1);
        x2 = boolc->PutY2BGate(x2);
        x2contrib = boolc->PutMUXGate(x2, sb_0, gtcheck);
        x2contrib = bc->PutB2YGate(x2contrib);
        x2sum = bc->PutADDGate(s_0, x2contrib);
        gtcheck = bc->PutGTGate(m_2, s_0);
        n = bc->PutADDGate(n, gtcheck);
        gtcheck = boolc->PutY2BGate(gtcheck);
        contrib = boolc->PutMUXGate(sb_2, sb_0, gtcheck);
        contrib = bc->PutB2YGate(contrib);
        sum_x = bc->PutADDGate(sum_x, contrib);
        xy = bc->PutMULGate(contrib, m_2);
        sum_xy = bc->PutADDGate(sum_xy, xy);
        x2 = bc->PutMULGate(s_2, s_2);
        x2 = boolc->PutY2BGate(x2);
        x2contrib = boolc->PutMUXGate(x2, sb_0, gtcheck);
        x2contrib = bc->PutB2YGate(x2contrib);
        x2sum = bc->PutADDGate(x2sum, x2contrib);
        // Similar operations are done for s_3 - s_22
        ...
        ...
        ...
        ...
        ...
        gtcheck = bc->PutGTGate(m_23, s_0);
        n = bc->PutADDGate(n, gtcheck);
        gtcheck = boolc->PutY2BGate(gtcheck);
        contrib = boolc->PutMUXGate(sb_23, sb_0, gtcheck);
        contrib = bc->PutB2YGate(contrib);
        sum_x = bc->PutADDGate(sum_x, contrib);
        xy = bc->PutMULGate(contrib, m_23);
        sum_xy = bc->PutADDGate(sum_xy, xy);
        x2 = bc->PutMULGate(s_23, s_23);
        x2 = boolc->PutY2BGate(x2);
        x2contrib = boolc->PutMUXGate(x2, sb_0, gtcheck);
        x2contrib = bc->PutB2YGate(x2contrib);
        x2sum = bc->PutADDGate(x2sum, x2contrib);
```

7.9 Alternative Simple Homomorphic Encryption Protocol

Here, we provide the source code for the alternative ‘simple’ homomorphic encryption protocol. We make use of DGK encryption rather than Paillier encryption here, and this protocol is implemented in C++ rather than in Python, where the original homomorphic encryption protocol was implemented. In the source code, we can see similarities to the garbled circuits protocols in the way that the data is preprocessed, however the calculation part is fundamentally different for both methods, as is expected.

```
/*
SeComLib
Copyright 2012-2013 TU Delft, Information Security & Privacy Lab
(http://isplab.tudelft.nl/)

Contributors:
Inald Lagendijk (R.L.Lagendijk@TUDelft.nl)
Mihai Todor (todormihai@gmail.com)
Thijs Veugen (P.J.M.Veugen@tudelft.nl)
Zekeriya Erkin (z.erkin@tudelft.nl)

Licensed under the Apache License, Version 2.0 (the ”License”);
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an ”AS IS” BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/
/**
@file regression/main.cpp
@brief Regression method for the PRANA-DATA setting, using dgk.
@author Laurens van der Beek
*/

#include ”main.h”

gmp_randstate_t state;

/**
Application entry point.

Usage: Accepts one optional parameter: the full path to the
configuration file. Otherwise, it tries to find ”config.xml”
in the current directory.

@param argc number of command line arguments
@param argv array containing the command line arguments
@return The process exit status
*/
int main () {
    try {
        // Number of children that are considered.
        int n_c = 6;
        CpuTimer t;
        int a;
        BigInteger contrib[24];
        DgkCiphertext n, scale;
        Dgk::Ciphertext sumMonth[24] = {};
        uint32_t b = 0, d = 0, e = 0;
        int l[n_c][24] = {};
        BigInteger dec[n_c][24] = {};
        DgkCiphertext c[n_c][24] = {};
        BigInteger output[24] = {};
        // Private key, only available to the key server
        Dgk PrivateCryptoProvider(true);
        PrivateCryptoProvider.GenerateKeys();
        /* Public key, available to all parties and the computation
        server */
        Dgk PublicCryptoProvider(PrivateCryptoProvider.GetPublicKey());
        DgkCiphertext zero = PublicCryptoProvider.EncryptInteger(0);
        DgkCiphertext one = PublicCryptoProvider.EncryptInteger(1);

        std::fstream myfile(
        ”/home/lvdbeek/ABY/src/examples/dupGC/common/RelevantChildren.txt”
        , std::ios_base::in);
        std::map<int,uint8_t> childID;
        while (myfile >> a)
        {
            /* we identify the meaning of a number from the file by
            its position and process it accordingly. Each line has
            3 values, the first of which is the childID */
            if(b % 3 == 0){
                // if the childID has not been mapped yet
                if(childID.find(a) == childID.end()){
                    childID.insert(std::pair<int,uint8_t>(a,e)
                    e += 1;
                }
            }
            /* we know the second entry is the month in which the
            measurement was performed, thus we store it in an array
            corresponding to the child the measurement is from. */
            if(b % 3 == 1){
                /* we know from preprocessing that the maximum
                month value is 24 and the minimum value is 0,
                therefore we map the 25th month to the 24th month,
                as is intended */
                if(a!=24){
                    d = a;
                }
                else{
                    d = 23;
                }
            }
            /* we know the third entry is the measurement of the
            length of the child with childID c-1 performed in
            month d. In the rare case that we have two measurements
            in the same month for one child, we average them to
            obtain the contribution for that child to the prediction
            of the value for the child of IND in that month. */
            if(b % 3 == 2){
                if(l[e-1][d] != 0){
                    l[e-1][d] = (l[e-1][d] + a)/2;
                }
                else{
                    l[e-1][d] = a;
                }
            }
            b += 1;
        }
        for(int i=0; i<n_c; i++)
        {
            for(int j=0; j<24; j++)
            {
            c[i][j] = PublicCryptoProvider.EncryptInteger(l[i][j]);
            }
        }

        for(int j=0; j<24; j++)
        {
            n = zero;
            sumMonth[j] = zero;
            for(int i=0; i<n_c; i++)
            {
            if(PrivateCryptoProvider.IsEncryptedZero(c[i][j]) == 0)
            {
                n = n + one;
                sumMonth[j] = sumMonth[j] + c[i][j];
            }
            }
            /* Scale the sum of a month based on the amount of
            children contributing to the value. */
            contrib[j] = PrivateCryptoProvider.DecryptInteger(n);
            if(contrib[j] != 0)
                sumMonth[j] = sumMonth[j] * (60/contrib[j]);
        }

        std::ofstream file;
        /* Each time we run the program, we want to start with an
        empty file. */
        file.open(
        "/home/lvdbeek/Downloads/SeComLib/regression/resultsRegression.txt"
        , std::ofstream::out | std::ofstream::trunc);
        for(int j = 0; j < 24; j++){
            output[j] = PrivateCryptoProvider.DecryptInteger(
            sumMonth[j])/60;
            if(output[j] != 0){
                /* A child can only grow, thus if we would obtain
                a month in which a decrease in length compared to
                a previous month occurs, for example because of
                the averaging between multiple different children,
                we don't take the length in that month into
                account. */
                if(j >= 1){
                    for(int i = 0; i < j; i++){
                    if(output[j] > output[i]){
                    if(i == j-1){
                    std::cout <<
                    "The␣predicted␣length␣of␣the␣child␣in
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣month␣" << j << "␣is:\t" <<
                    output[j].ToString(10) << "\n";
                    file << j << "\t" <<
                    output[j].ToString(10) << "\n";
                    }
                    }
                    else{
                        continue;
                    }
                    }
                }
                else{
                    std::cout <<
                    "The␣predicted␣length␣of␣the␣child␣in
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣month␣" << j << "␣is:\t" <<
                    output[j].ToString(10) << "\n";
                    file << j << "\t" <<
                    output[j].ToString(10) << "\n";
                }
            }
        }
        file.close();

        std::cout << "Runtime␣of␣the␣protocol␣=␣" << t.ToString()
        << std::endl;

    }
    catch (const std::runtime_error &exception) {
        std::cout << exception.what() << std::endl;
        }
        catch (const std::exception &exception) {
                std::cout << exception.what() << std::endl;
        }
        /*it won't catch low level exceptions, like division by 0,
        produced by GMP... */
        catch (...) {
                std::cout << "Unexpected␣exception␣occured." << std::endl;
        }



        printf("\n␣\n␣press␣ENTER␣to␣exit.\n");
        while(!getchar());
        return 0;
}
```

7.10 Alternative Linear Homomorphic Encryption Protocol

For the alternative ‘linear’ homomorphic encryption protocol, the same goes as for the ‘linear’ garbled circuits protocols: it is built onto the code for the corresponding ‘simple’ protocol, thus in this case the code for the alternative ‘simple’ homomorphic encryption protocol, which was presented in the previous section. It extends this code by adding some variables to store the denominator and numerator for the slope and the intercept, and by adding some operations to compute these variables. These extensions can be found in the code below:

```
DgkCiphertext sum = zero;
DgkCiphertext nMonth = zero;
DgkCiphertext sumNMonth = zero;
DgkCiphertext sumSQMonth = zero;
DgkCiphertext sumMonthLength = zero;
BigInteger num_i, den_is, num_s, reg_a, reg_b;

for(int j = 0; j<24; j++)
{
        if(output[j]!=0)
        {
                /* A child can only grow, thus if we would obtain a month in
                which a decrease in length compared to a previous month
                occurs, for example because of the averaging between multiple
                different children, we don't take the length in that month into
                account.
        Here we build the variables we need in order to do linear
        regression*/
        if(j >= 1){
                for(int i = 0; i < j; i++){
                if(output[j] > output[i]){
                if(i == j-1){
                /* nMonth is the amount of months that contribute an
                average to the linear regression, sumNMonth is the sum
                of the numbers of the contributing months, sumSQMonth
                is the sum of the squared numbers of the contributing
                months, sumMonthLength is the sum of the contributing
                months multiplied by the average measured length in
                that month and sum is the sum of the average measured
                lengths in each month.*/
                nMonth = nMonth + one;
                sumNMonth = sumNMonth +
                PublicCryptoProvider.EncryptInteger(j);
                sumSQMonth = sumSQMonth +
                PublicCryptoProvider.EncryptInteger(j*j);
                /* Scalar multiplication with 0 is not well defined within
                the homomorphically encrypted domain.*/
                if(j!=0){
                        sumMonthLength = sumMonthLength +
                        PublicCryptoProvider.EncryptInteger(output[j]) * j
                }
                sum = sum + PublicCryptoProvider.EncryptInteger(output[j])
                }
                }
                else{
                        continue;
                }
                }
        }
        else{
                nMonth = nMonth + one;
                sumNMonth = sumNMonth +
                PublicCryptoProvider.EncryptInteger(j);
                sumSQMonth = sumSQMonth +
                PublicCryptoProvider.EncryptInteger(j*j);
                /* Scalar multiplication with 0 is not well defined within
                the homomorphically encrypted domain.*/
                if(j!=0){
                        sumMonthLength = sumMonthLength +
                        PublicCryptoProvider.EncryptInteger(output[j]) * j
                }
                sum = sum + PublicCryptoProvider.EncryptInteger(output[j])
        }

}
```

```
}
// Numerator for the intercept:
num_i = PrivateCryptoProvider.DecryptInteger(sum) *
PrivateCryptoProvider.DecryptInteger(sumSQMonth) -
PrivateCryptoProvider.DecryptInteger(sumMonthLength) *
PrivateCryptoProvider.DecryptInteger(sumNMonth);
// Numerator for the slope:
num_s = PrivateCryptoProvider.DecryptInteger(sumMonthLength) *
PrivateCryptoProvider.DecryptInteger(nMonth) -
PrivateCryptoProvider.DecryptInteger(sum) *
PrivateCryptoProvider.DecryptInteger(sumNMonth);
// Denominator for both the intercept and the slope:
den_is = PrivateCryptoProvider.DecryptInteger(sumSQMonth) *
PrivateCryptoProvider.DecryptInteger(nMonth) -
PrivateCryptoProvider.DecryptInteger(sumNMonth) *
PrivateCryptoProvider.DecryptInteger(sumNMonth);
// Result of the regression
reg_a = num_i / den_is;
reg_b = num_s / den_is;
file << reg_a.ToString(10) << "\t" << reg_b.ToString(10) << "\n";
std::cout << "The␣intercept␣is:␣" << reg_a.ToString(10) <<
"␣and␣the␣slope␣is:␣" << reg_b.ToString(10) << "\n";
file.close();

std::cout << "Runtime␣of␣the␣protocol␣=␣" << t.ToString() << std::endl;
```

7.11 Alternative Slope-based Homomorphic Encryption Protocol

Our source code for the alternative ‘slope-based’ homomorphic encryption protocol is the same as the source code for the alternative ‘simple’ homomorphic encryption protocol, with the same relative change as mentioned in Section 7.4.

7.12 Specifications Hard- and Software

Here, we provide the specifications of the machine and programs we used in our experiments, in order to enable future researchers of this topic to do comparable experiments with other protocols or to reproduce our experiments. First, it is important to note that we used a virtual machine which ran on Ubuntu 16.04 and had 2 GB of memory assigned to it to run all our protocols. The host machine we used has as processor an Intel Core i5-6300 CPU with 2.40 GHz and 2 core, and as graphical card an Intel HD graphics 520, has 8 GB of physical RAM, 12,9 GB of virtual RAM and runs on windows 10 enterprise.

Regarding the software we used, a ‘docker container’ structure was used for the original protocol, the ABY-framework for both types of garbled circuit protocols, and SeComLib for the alternative homomorphic encryption protocols. SeComLib has some dependencies, of which we used the following versions: GMP (6.1.2), Boost (1.53.0) and MPIR (2.5.2). The original protocol is implemented in Python, and the rest of the protocols are implemented in C++.