Privacy-Preserving Fast and Exact Linear Equations Solver with Fully Homomorphic Encryption

Keita Arimitsu[1,2] and Kazuki Otsuka[1]

[1] *ThinkX,Inc – Minato, Tokyo*
[2] *Theoretics Physik, Eidgenössische Technische Hochschule, 8093 Zürich, Switzerland*
arimitsu.keita@thinkxinc.com, kaz@thinkxinc.com

February 27, 2020

Abstract

Privacy and machine learning are difficult to coexist due to their nature: parivacy should be kept from others while machine learning requires large amount of data. Among several possible solutions to this problem, *Fully Homomorphic Encryption* (FHE)[1, 2, 3, 4] has been a center of intensive researches in this field. FHE enables linear operations of ciphertext. To take advantage of this property, many protocols to achieve statistical operaions have been proposed. On the other hand, many of them are impractical. Some of the approaches introduce cryptosystems that are not familiar. Moreover, most of their protocols are approximation which might sensitively depend on our choice of parameters. In this paper, we propose fast, simple, and exact privacy-preserving linear equation solver using FHE. Our two-party protocol is secure against at least semi-honest model, and we can exactly calculate the model even without the bootstrapping.

1 Introduction

Machine learning plays a momentous role in our society today. It is being applied to almost all fields of not only science such as information science, biology, and physics, but also of industries, or even everyday comodity. Among technologies where machine learning is used, cloud computing is one of the most intensive field of study and application. While we can never doubt usefulness of the machine learning, it sometimes requires reveal of sensitive information. For example, credit scoring is a growing trend in finance: in order to properly score the user's credibility, private data such as payment history, diploma, and career would be necessary. Moreover, sometimes sharing of data is even prohibited by law. For example, patients' information is not allowed to share between two different hospitals. Therefore, to fully utilize the ability of machine learning, we should develop the way to keep privacy preserved when these are proccessed. To tackle this problem, there has been many protocols proposed. One of the most common choice is *Homomorphic Encryption*[1, 2, 3, 4], which enables calculations of ciphertext without decrypting. The other possible choice is *Yao's Garbled Circuit*. One can evaluate any circuit with this garbled circuit without revealing its input. In out paper, we propose a simple and exact way to train the given data for *Linear Regression*, which is one of the most popular statistical analysis. Our paper is organized as follows: In Sec.3 we introduce so-called *Fully Homomorphic Encryption*, and also we review the linear regression and how to do this job in the privacy-preserved way in Sec.4. Finally we introduce our protocol for the training phase.

2 Our Contribution

We propose a simple and exact way to implement privacy-preserving linear regression. There has been many protocols which achieves the privacy-preserving linear regression[5, 6, 7]. On the other hand, they are either complicated and even introduce an encryption scheme which is not used by majority of the community. Our protocol is within the standard protocol and introduces no uncommon idea. Moreover, in many cases those schemes are iterative such as gradient descent and Cholesky decomposition. This means their results are approximation. However, our protocol is exact and can be performed in the reasonable time. Our experiment shows that linear regression can be carried exactly even without bootstrapping.

3 Homomorphic Encryption

In this section homomorphic encryption, mainly focusing on so-called fully homomorphic encryption (FHE) is introduced briefly. First its history and basic philosophy are explained, then one particular schemes by Brakerski/Fan-Vercauteren (BFV or FV) scheme[9] is discussed in a more detail.

3.1 Brief Summary of (Fully) Homomorphic Encryption

History of homomorphic encryption can be traced back to Rivest in 1978 where he termed "privacy homomorphism" for homomorphic encryption[10]. In that paper he proposed the existence of some encryption scheme with which ciphertext can be operated before decrypting it. Since then, the study of homomorphic encryption has been one of the big area of cryptography. Until now, there has been many homomorphic encryption schemes proposed such as RSA[11], ElGamal[12], Goldwasser-Micali[3], Paillier[4], and many more. Those schemes are called *partially homomorphic*: if m_1 and m_2 are plaintext in a plaintext space $\mathcal{M}$ and c_1 and c_2 are corresponding ciphertext generated by an ecryption scheme Enc in a ciphertext space $\mathcal{C}$, then operations $\cdot_m, \cdot_c$ are defined both in $\mathcal{M}$ and $\mathcal{C}$ respectively, and satisfy the following relation

$$\mathrm{Enc}(m_1 \cdot_m m_2) = \mathrm{Enc}(m_1) \cdot_c \mathrm{Enc}(m_2) = c_1 \cdot_c c_2. \tag{1}$$

The operation $\cdot$ can be either addition or multiplication, but not both in *partially homomorphic encryption* (PHE) schemes.

In 2009, Gentry proposed the first fully homomorphic encryption scheme in his PhD thesis[13], which is based on ideal lattice. His construction of FHE is as follows: first *somewhat homomorphic encryption* (SWHE) scheme is introduced. SWHE is an encryption scheme which allows the limited number of addition and multiplication. Here the detail of the construction of SWHE is not discussed. But we should note that the decryption would not work if we add or multiply too many ciphertexts. To tackle this problem, *bootstrapping* is defined on SWHE scheme. Bootstrapping is a special technique which makes a noisy ciphertext less noisy. This is done by applying to a noisy ciphertext a decryption circuit which is homomorphic.

Since Gentry's blueprint, several variations of FHE have been proposed so far, and those schemes are classified into three generations. The first generation include the original Gentry's approach[13], and several sophistications[14, 15, 16, 17].

The second generation includes Brakerski-Gentry-Vaikuntanathan (BGV) scheme[8], Brakerski/Fan - Vaikuntanathan (BFV) scheme[9], and Cheon-Kim-Kim-Song (CKKS) scheme[18]. Theoretical difference between the first generation and the second generation is that the first generation depends on ideal lattice or integer for security while many of the second generation schemes uses Learing With Error (LWE) problem, or its variant Ring LWE (RLWE) problem. Practical difference is that these schemes are much more efficient than those of the first generations and in some cases it suffices only to run SWHE scheme, i.e., efficient so that we do not need the bootstrapping for some problems. These second generation schemes are also practically important since many homomorphic encryption libraries are based on these schemes.

The third scheme is based on Gentry-Sahai-Waters (GSW) cryptosystem[19]. It includes so-called FHEW[20] and so-called TFHE[21].

3.2 Example –BFV scheme–

As mentioned in the previous section, there are many variants of FHE scheme. But in the practical sense, BGV scheme[8] and BFV scheme[9] are mostly used. Therefore here we review BFV implementation. Here only BFV scheme is discussed since we implement FHE using BFV scheme and difference between BGV scheme and BFV scheme is inessential and the library we use in Sec,4.3 is based on BFV.

We first introduce basic notations which follow the original paper[9].

- We consider operations in the polynomial ring $R = \mathbb{Z}[x]/f(x)$ where $\mathbb{Z}[x]$ means a set of polynomial with integer coefficient and $f(x)$ is a monic irreducible polynomial with degree d. Usually $f(x) = x^d + 1$ with $d = 2^n$.

- An element of R is denoted by $\boldsymbol{a} \in R$, where we can view $\boldsymbol{a}$ as a vector in some sense since $\boldsymbol{a} = \sum_{i=0}^{d-1} a_i x^i$. Addition, subtraction, and multiplication are defined as elementwise operations. The norm is defined such that $\|\boldsymbol{a}\| \coloneqq \max_i\{a_i\}$ which is called L^∞ norm. Expansion factor δ_R is also defined as $\delta_R \coloneqq \max\{\|\boldsymbol{a}\cdot\boldsymbol{b}\|/\|\boldsymbol{a}\|\|\boldsymbol{b}\|; \boldsymbol{a}, \boldsymbol{b} \in R\}$

- For an integer q, $\mathbb{Z}_q \coloneqq (-q/2, q/2]$. R_q is a subset of the polynomial ring R whose coefficient is in $\mathbb{Z}_q$. For an integer a, $[a]_q \equiv a(\bmod q)$ and satisfies $[a]_q \in \mathbb{Z}_q$. For $\boldsymbol{a} \in R$, $[\boldsymbol{a}]_q$ is an element of the polynomial ring R with each coefficient being $[a_i]_q$.

- For $x \in \mathbb{R}$, $\lfloor x \rceil$ is rounding to the nearest integer, $\lfloor x \rfloor$ is rounding up and $\lceil x \rceil$ rounding down.
- χ denotes Gaussian probability distribution over integer $\mathbb{Z}^d$, and $\boldsymbol{e} \leftarrow \chi$ denotes an element of R with coefficients sampling from χ. $\boldsymbol{a} \leftarrow A$ where A is a set means a uniform sampling from A.

For encryption, we first set R_q for $q \in \mathbb{Z}$ and define the plaintext space to be R_t for $t \in \mathbb{Z}$ with $t < q$. $\Delta \coloneqq \lfloor q/t \rfloor$.

$$
\begin{aligned}
&\mathsf{SecretKeyGen}(1^\lambda) : \text{sample } \boldsymbol{s} \leftarrow \chi \quad \text{and set a secret key } sk = \boldsymbol{s} \\
&\mathsf{PublicKeyGen}(sk) : \text{sample } \boldsymbol{a} \leftarrow R_q, \boldsymbol{e} \leftarrow \chi \quad \text{and set a public key } pk = ([-(\boldsymbol{a}\cdot\boldsymbol{s}+\boldsymbol{e})]_q, \boldsymbol{a}) \\
&\mathsf{Encrypt}(pk, \boldsymbol{m}) : \text{set a plaintext } \boldsymbol{m} \in R_t, \text{define } \boldsymbol{p}_0 = pk[0], \boldsymbol{p}_1 = pk[1] \text{sample } \boldsymbol{u}, \boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \chi \\
&\text{create a ciphertext } ct = ([\boldsymbol{p}_0\cdot\boldsymbol{u} + \boldsymbol{e}_1 + \Delta\boldsymbol{m}]_q, [\boldsymbol{p}_1\cdot\boldsymbol{u} + \boldsymbol{e}_2]_q) \\
&\mathsf{Decrypt}(sk, ct) : \boldsymbol{s} = sk, \text{define } \boldsymbol{c}_0 = ct[0], \boldsymbol{c}_1 = ct[1], \text{then calculate } \left[\left\lfloor \frac{t\,(\boldsymbol{c}_0 + \boldsymbol{c}_1\cdot\boldsymbol{s})}{q} \right\rceil\right]_t
\end{aligned}
\tag{2}
$$

We can find the following property

Proposition $\|\chi\| < B$, then sufficient condition for correct decryption is

$$
2\delta_R B^2 + B < \frac{\Delta}{2} \tag{3}
$$

For a proof, see[9].

Addition and multiplication are defined as follows (we do not discuss them in detail, for detailed discussions, see the original paper[9]). Assume two plaintexts $\boldsymbol{m}^1, \boldsymbol{m}^2$ and the corresponding ciphertexts $ct^1 = (\boldsymbol{c}_0^1, \boldsymbol{c}_1^1), ct^2 = (\boldsymbol{c}_0^2, \boldsymbol{c}_1^2)$.

$$
\begin{aligned}
&\mathsf{Addition}(ct^1, ct^2) : ct^1 + ct^2 = \left(\boldsymbol{c}_0^1 + \boldsymbol{c}_0^2, \boldsymbol{c}_1^1 + \boldsymbol{c}_1^2\right) \\
&\mathsf{Multiplication}(ct^1, ct^2) : ct^1 \cdot ct^2 = \left(\boldsymbol{c}_0^1\cdot\boldsymbol{c}_0^2, \boldsymbol{c}_0^1\cdot\boldsymbol{c}_1^2 + \boldsymbol{c}_1^1\cdot\boldsymbol{c}_0^2, \boldsymbol{c}_1^1\cdot\boldsymbol{c}_1^2\right)
\end{aligned}
\tag{4}
$$

Addition is intuitive, but Multiplication is not. To make sense of it, consider Decrypt. There one has to compute $\boldsymbol{c}_0 + \boldsymbol{c}_1\cdot\boldsymbol{s} = \Delta\boldsymbol{m} + \boldsymbol{v} + q\boldsymbol{r}$. Therefore

$$
\begin{aligned}
(\boldsymbol{c}_0^1 + \boldsymbol{c}_1^1\cdot\boldsymbol{s})(\boldsymbol{c}_0^2 + \boldsymbol{c}_1^2\cdot\boldsymbol{s}) &= \left(\boldsymbol{c}_0^1 + \boldsymbol{c}_0^2\right) + \left(\boldsymbol{c}_0^1\cdot\boldsymbol{c}_1^2 + \boldsymbol{c}_1^1\cdot\boldsymbol{c}_0^2\right)\cdot\boldsymbol{s} + \left(\boldsymbol{c}_1^1\cdot\boldsymbol{c}_1^2\right)\cdot\boldsymbol{s}^2 \\
&= \Delta^2\boldsymbol{m}^1\cdot\boldsymbol{m}^2 + \cdots
\end{aligned}
\tag{5}
$$

where we can see elements of Multiplication as coefficient of each $\boldsymbol{s}^i$. As one can see, Decryption increases of the size of ciphertext. To deal with this problem there is a technique called relinearlize, but we do not discuss here. It is also important to note that both Addition and Multiplication increase "noise" $\boldsymbol{v}$, and particularly for Multiplication increment of noise is exponential.

4 Privacy-Preserving Linear Regression

In this section, we discuss the problem of linear regression. Manipulating data with privacy-preserving manner dates back to Lindell and Pinkas in 2000[22]. After this seminal work, many protocals have been introduced. It is not easy to classify every protocal into some classes since there are so many, but most of all protocols adopt at least one of three techniques. One can split his data and share one part of the original data with one server, and different part with different server. BGW[23] is included in this group. Another way is to use homomorphic encryption. The other way is to use Yao's *garbled circuit protocol* (GCP)[24, 25]

First, few basics of linear regression are summarized. Next, several approaches to the linear regression problem using homomorphic encryption are introduced. We also look at GCP. GCP is another possible choice for implementing privacy-preserving linear regression. Finally, we propose a new scheme which is fast and easy to implement

4.1 Basics of Linear Regression

Let us suppose we are given a data consisting of N independent variables $(x_1, x_2, \cdots, x_N)$ and an output y. From this data we want to relate the output to variables. One ansatz is to assume they are linearly related, i.e., we assume

$$
y = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_N x_N + \epsilon \tag{6}
$$

where ϵ is error. To relate properly we need more data: assume we are given a dataset consisting of D data, i.e., $\left\{(y_i, x_{i,1}, x_{i,2} \cdots, x_{i,N})_{i=1}^{D}\right\}$. We want to find a linear relation

$$y_i = \sum_{j=1}^{D} \beta_j x_{i,j} + \epsilon_i. \tag{7}$$

In the vector and matrix representation, (7) is equivalent to

$$\boldsymbol{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{8}$$

where $(\boldsymbol{y})_i = y_i, X_{ij} = x_{i,j}, (\boldsymbol{\beta})_i = \beta_i, (\boldsymbol{\epsilon})_i = \epsilon_i$. We next consider optimazation of $\boldsymbol{\beta}$. In the least square method, we minimize the L^2 norm of the error $\boldsymbol{\epsilon}$. In other words we should minimize

$$\mathcal{L}(\boldsymbol{\beta}) = |\boldsymbol{y} - X\boldsymbol{\beta}|^2, \tag{9}$$

where the norm of $\boldsymbol{\epsilon}$ is represented as a function of $\boldsymbol{\beta}$, i.e., $\mathcal{L}(\boldsymbol{\beta})$, and it is called the object function. To find the minimum is easy since we can differentiate the object function and obtain

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = -2X^{\mathrm{T}}(\boldsymbol{y} - X\boldsymbol{\beta}) \tag{10}$$

and the optimal $\boldsymbol{\beta}$ is given by $\boldsymbol{\beta} = \left(X^{\mathrm{T}}X\right)^{-1} X^{\mathrm{T}}\boldsymbol{y}$. So essentially the linear regression is to solve an inverse matrix $X^{\mathrm{T}}X$ and multiply it by $X\boldsymbol{y}$.

If we impose a condition that a norm of $\boldsymbol{\beta}$ should not be large, this problem is called *ridge regression*. The redge regression problem is solved easily just by adding another term in (9) such that

$$\mathcal{L}_\lambda(\boldsymbol{\beta}) = |\boldsymbol{y} - X\boldsymbol{\beta}|^2 + \lambda |\boldsymbol{\beta}|^2 \tag{11}$$

where the second term is a penalty for large $\boldsymbol{\beta}$. The optimal $\boldsymbol{\beta}$ is given by $\boldsymbol{\beta} = \left(X^{\mathrm{T}}X + \lambda \mathbf{1}\right)^{-1} X^{\mathrm{T}}\boldsymbol{y}$, where $\mathbf{1}$ is an unit matrix. Therefore in either linear regression or ridge regression, what we should do is first to calculate an inverse matrix and then to multiply it by some vector. Notice that with FHE multiplication is done easily, on the other hand, matrix inversion is not trivial. Therefore many proposed schemes of privacy preserving mainly try to solve this problem.

4.2 Proposed Scheme: Masking Matrix Method

4.2.1 Informal Description of the Method

When one tries to send message, it is possible that he adds some random number so that no one can guess it. Also in linear regression using homomorphic encryption, this trick is sometimes used such as[5, 7]. Not only in the context of homomorphic encryption for linear regression, but also many other context[27, 28]. The basic idea is simple: assume Alice want to calculate $\boldsymbol{\beta}$ which satisfies $A\boldsymbol{\beta} = \boldsymbol{b}$ but she cannot compute it. Bob can compute it but Alice does not want Bob to know the matrix A and the vector $\boldsymbol{b}$. Then Alice can generate some random invertible matrix R and some random vector $\boldsymbol{r}$. She compute AR and $\boldsymbol{b} + A\boldsymbol{r}$ and send them to Bob. Bob cannot extract any information of $A, \boldsymbol{b}$ from what he gets since they are masked. He, then calculates $(AR)^{-1}(\boldsymbol{b} + A\boldsymbol{r})$, which is equivalent to $R^{-1}A^{-1}\boldsymbol{b} + R^{-1}\boldsymbol{r}$. Therefore Alice can get $A^{-1}\boldsymbol{b}$ by multiplying R from the left side followed by the substruction by $\boldsymbol{r}$. Alice outsources difficult matrix inversion to Bob while the original matrix is kept secret. Combination of FHE and this method gives us a fast, easy-to-implement, and exact method of linear regression.

4.2.2 Model

Our architecture follows that of [5, 7], which means we assume the following entities,

- *Data Provider (DP)*: it is made up of m users. ith user holds dataset i which is denoted by $\mathcal{DO}_i = (x_{i,1} \cdots x_{i,N}, y_i)$. There is no need for physical existence of DP, i.e., we do not need any institution which manages users' information, rather we just group people who want to use some service and therefore have to send their information.
- *Machine Learning (ML) Server*: ML server collects the information from DP and is resposible for calculating the model. In the example shown above, Alice plays a ML server.
- *Crypto Service Provider (CSP)*: CSP is responsible for generating a set of public key and secret key. CSP also coorperate ML server for calculating the model which Bob did in the above-mentioned example.

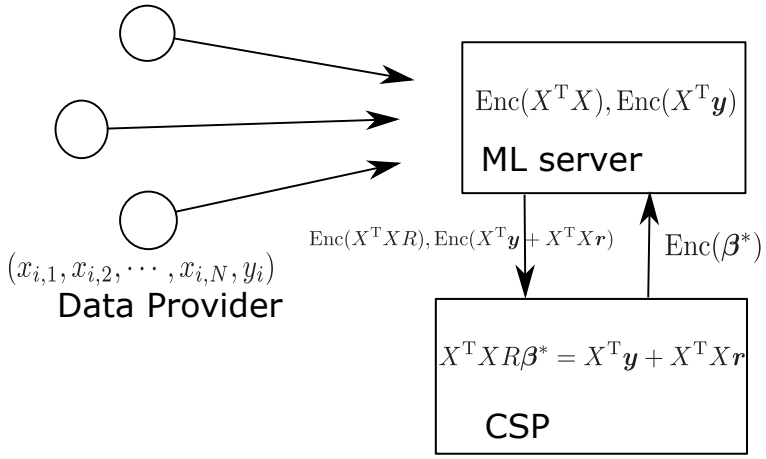

Figure 1: A schemetic figure of training phase. For the meaning of each notation, refer to Sec.4.2.4

4.2.3 The Threat Model

For a sake of privacy, both the ML server and the CSP should not learn anything other than what is revealed by the linear regression. In the following, first we assume those two parties are *honest-but-curious*, i.e., they follow the algorithm, but they also try to extract information from what is given to them. Therefore the algorithm in Sec.4.2.4 is secure only against honest-but-curious entities.

However, the ML server and the CSP can be malicious. We assume the following cases as [5] does: the ML server might use only dataset partially given by DP to extract some information. Also the ML server deliverately executes wrong calculations. The CSP can make mistakes in its calculation intentionally as well. Morever, it decrypts ciphertexts wrongly. We address how to deal with these behaviors in the subseqent section.

We do not assume the ML server and the CSP collude and we assume key distribution is properly done so that everyone has a set of correct public key and secret key.

4.2.4 Training Phase

First let us consider the training phase: make a proper model $\boldsymbol{\beta}$ which minimizes $\mathcal{L}(\boldsymbol{\beta})$ in (9) or $\mathcal{L}_\lambda(\boldsymbol{\beta})$ in (11). We take the following step:

- DP first encrypts data and sends it to the ML server.

DP can make a choice. It can send a raw data, i.e., each $\mathcal{DO}_i$ just sends $(x_{i,1}, \cdots, x_{i,N}, y_i)_{i=1}$, or it can also process its data for better performance, i.e., $\mathcal{DO}_i$ sends an *encrypted matrix* $\mathrm{Enc}(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i)$ and an *encrypted vector* $\mathrm{Enc}(\boldsymbol{x}_i^{\mathrm{T}}y_i)$, namely

$$\mathrm{Enc}(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i) = \begin{pmatrix} \mathrm{Enc}(x_{i,1}x_{i,1}) & \mathrm{Enc}(x_{i,1}x_{i,2}) & \cdots & \mathrm{Enc}(x_{i,1}x_{i,N}) \\ \mathrm{Enc}(x_{i,2}x_{i,1}) & \mathrm{Enc}(x_{i,2}x_{i,2}) & \cdots & \mathrm{Enc}(x_{i,2}x_{i,N}) \\ \vdots & & \ddots & \vdots \\ \mathrm{Enc}(x_{i,N}x_{i,1}) & \mathrm{Enc}(x_{i,N}x_{i,2}) & \cdots & \mathrm{Enc}(x_{i,N}x_{i,N}) \end{pmatrix}, \mathrm{Enc}(\boldsymbol{x}_i^{\mathrm{T}}y_i) = \begin{pmatrix} \mathrm{Enc}(x_{i,1}y_i) \\ \mathrm{Enc}(x_{i,2}y_i) \\ \vdots \\ \mathrm{Enc}(x_{i,N}y_i) \end{pmatrix}. \quad (12)$$

- The ML server constructs a matrix and a vector for the linear regression while they are encrypted.

If DP sends a raw data, then the ML server first constructs $\mathrm{Enc}(X)$ and calculates $\mathrm{Enc}(X^{\mathrm{T}}X)$ and same for $\mathrm{Enc}(X^{\mathrm{T}}\boldsymbol{y})$. If DP sends a processed data, then the ML server just sums up all the matrices and vectors, i.e., $\mathrm{Enc}(X^{\mathrm{T}}X) = \sum_i \mathrm{Enc}(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i)$ and $\mathrm{Enc}(X^{\mathrm{T}}\boldsymbol{y}) = \sum_i \mathrm{Enc}(\boldsymbol{x}_i^{\mathrm{T}}y_i)$ using homomorphic property of encryption scheme.

- The ML server masks the original matrix and vector and sends them to CSP

The ML server invents a random invertible matrix R and a random vector $\boldsymbol{r}$ and calculates $\mathrm{Enc}(X^{\mathrm{T}}XR)$ and $\mathrm{Enc}(X^{\mathrm{T}}\boldsymbol{y} + X^{\mathrm{T}}X\boldsymbol{r})$ and sends to CSP. At this stage the ML does not know anything about the original information and CSP even after receiving those matrix and vector, does not know anything about the original information since they are masked by random matrix and vector. Therefore unless the ML server and CSP collude, this is secure.

- CSP decrypts what it receives from the ML server and calculates the model which is not an actual model we want, and sends this model to the ML server

CSP decrypts and calculates $(X^{\mathrm{T}}XR)^{-1}(X^{\mathrm{T}}\boldsymbol{y} + X^{\mathrm{T}}X\boldsymbol{r}) = R^{-1}(X^{\mathrm{T}}X)^{-1}\boldsymbol{y} + R^{-1}\boldsymbol{r}$ and sends it to the ML server after encrypting it.

- The ML server multiplies what it receives from CSP by R from the left followed by subtracting $\boldsymbol{r}$, which gives $(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}\boldsymbol{y}$.

4.2.5 Security

Both the training scheme and the evaluation scheme are secure against honest-but-curious entities, In this section we try to extend these schemes to cover even malicious entites. Again we assume the ML server and CSP can be malicious, but they do not collude.

In the training phase, the possible senarios for the malicious ML server are

- the ML deliberately outputs a wrong calculation
- it does not pick all the information given by DP to extract some information

These misbehaviors can be detected by the following test: DP prepares a "dummy" training set and send it to the ML server. The dummy data is constructed such that for N dimensional data, i.e., the data with N variables, threre are N data which means $\mathcal{DO}_i (i = 1, \cdots, N)$. For ith $\mathcal{DO}_i$ the data is as follows

$$\mathcal{DO}_i = (0, \cdots, x_{i,i} = x_i, \cdots, 0, y_i) \tag{13}$$

Only ith variable is a nonzero arbitrary number and y_i is also arbitrary. Using $\mathcal{DO}$ the ML server calculates a matrix and a vector as in Sec.4.2.4 and sends the masked matrix to CSP. Notice that only when the ML properly calculate $X^{\mathrm{T}}X$ without dropping any information $X^{\mathrm{T}}X$ becomes an invertible matrix. Then CSP decrypts the matrix and calculate the determinant of it. Then if the determinant is zero, which means the ML server cannot properly calculate the matrix so CSP can reject it.

In the scoring phase, the ML server also can be malicious by sending a different model. On ther other hand, in terms of security this assumption is meaningless since we assume the ML server aims at extracting $\mathcal{DO}$'s information by its misbehaviors, but sending a different model reveals nothing.

CSP can be malicious as well, and we assume CSP can produce a wrong result. Even if CSP is malicious, CSP cannot obtain any personal information from what it has been given since the matrix and the vector it receives are just random: suppose $X^{\mathrm{T}}XR \in GL(N, \mathbb{Z})$ which means this $N \times N$ matrix is invertible and $X^{\mathrm{T}}\boldsymbol{y} + X^{\mathrm{T}}X\boldsymbol{r} \in \mathbb{Z}^N$. Random matrix and vector are defined such that there exists an integer M and for any integer $m \in [-M, M]$ each component of them, i.e., $R_{i,j}, r_k$ satisfy

$$\Pr[R_{i,j} = m] = \frac{1}{2M}, \Pr[r_k = m] = \frac{1}{2M}. \tag{14}$$

It is easy to see that for sufficiently large M, for any matrix A and any vector $\boldsymbol{a}$ with each component in the interval $[-M, M]$,

$$\begin{aligned} \Pr[X^{\mathrm{T}}XR = A] = \Pr[R = (X^{\mathrm{T}}X)^{-1}A] = \left(\frac{1}{2M}\right)^{N^2} \\ \Pr[X^{\mathrm{T}}\boldsymbol{y} + X^{\mathrm{T}}X\boldsymbol{r} = \boldsymbol{a}] = \Pr[\boldsymbol{r} = (X^{\mathrm{T}}X)^{-1}(\boldsymbol{a} - X^{\mathrm{T}}\boldsymbol{y})] = \left(\frac{1}{2M}\right)^{N} \end{aligned} \tag{15}$$

Therefore this method is secure in terms of privacy even against a malicious CSP. Moreover, one can even detect a malicious CSP by the following trick. DP prepares a "dummy" training set and send it to the ML server. The dummy data is constructed such that for N dimensional data, i.e., the data with N variables, threre are N data which means $\mathcal{DO}_i (i = 1, \cdots, N)$ and $rank(X) = N$. From this detaset one can construct the model $\boldsymbol{\beta}$ such that $\boldsymbol{y} - X\boldsymbol{\beta} = \boldsymbol{0}$. The ML server follows the scheme and sends the masked matrix and the masked vector to CSP. Finally the ML server obtain the model which is encrypted $\mathrm{Enc}(\boldsymbol{\beta})$. Then the ML server calculates $\boldsymbol{y} - X\boldsymbol{\beta}$ and let us denote this answer by $\boldsymbol{s}$. Although the ML server only has $\mathrm{Enc}(\boldsymbol{s})$, it can determine if $\boldsymbol{s}$ is $\boldsymbol{0}$ or not. The ML server generate random scalars, vectors, matries and multiplies by $\boldsymbol{s}$ from the right. Then the ML server can detect the malicious CSP when if all the results are not same.

4.3 Experiment

In this section we look at performance of each method.

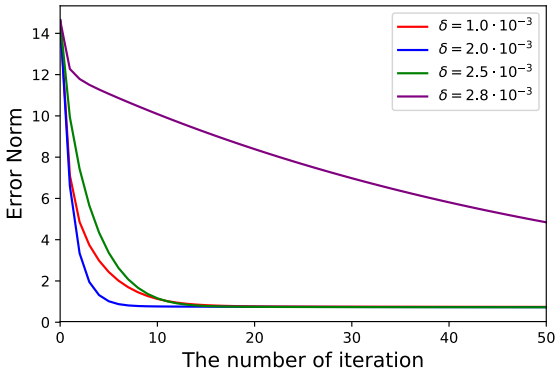

Figure 2: Convergence of gradient descent with various learning rate δ.

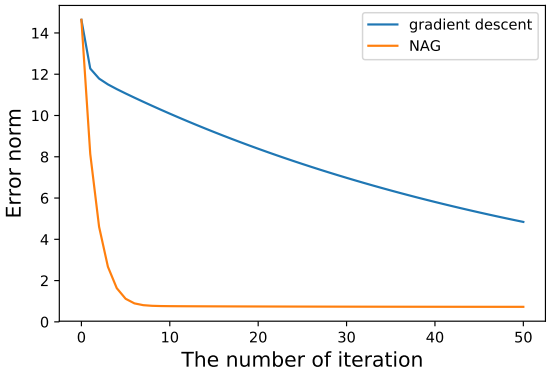

Figure 3: Comparison between the simple gradient descent and NAG when $\delta = 2.8 \times 10^{-3}$. The other parameter is $\eta = 0.1$. The simple gradient shows slow convergence, but NAG is relatively fast, only to need less than 10 iterations.

4.3.1 Gradient Descent

To compare MMM's performace with the existing method, we first consider the gradient descent. It is particularly efficient if a matrix considered is sparse. Staring from an initial value $\boldsymbol{\beta}^{[0]}$, using (10) with learning rate δ the model can be updated as follows:

$$\boldsymbol{\beta}^{[k]} = \boldsymbol{\beta}^{[k-1]} + \delta X^{\mathrm{T}} \left(\boldsymbol{y} - X\boldsymbol{\beta}^{[k-1]} \right). \tag{16}$$

Particularly, Esperança et al. studied gradient descent method[26]. Mathematically, with suitable choice of the initial value $\boldsymbol{\beta}^{[0]}$ and learing rate δ, a convergence to the optimal result is assured. Suitable choice of δ means $\delta \in (0, 2/(\mathcal{S}(X^{\mathrm{T}}X)))$ where $\mathcal{S}(A)$ is the spectral radius (difference between the largest eigenvalue and the smallest eigenvalue) of operator A. In [26], authors found oscillatory nature of $\boldsymbol{\beta}^{[k]}$,

$$\boldsymbol{\beta}^{[k]} = \sum_{n=1}^{k} (-1)^{n+1} \binom{k}{k-n} \delta^n (X^{\mathrm{T}}X)^{n-1} X^{\mathrm{T}} \boldsymbol{y} \tag{17}$$

Using (17), we can accelerate the gradient descent as one can find in [26] First, even before using FHE we should look at convergence of sequence $\boldsymbol{\beta}^{[k]}$ which this method outputs. We generate seven datasets whose dimension is five, i.e., $(x_{i1}, \cdots, x_{i,5}, y_i)_{i=1}^{7}$. Figure.2 shows convergence of the model $\boldsymbol{\beta}^{[k]}$

From Fig.2 we can understand performance is sensitive to the learning rate δ and when $\delta > 3.0 \times 10^{-3}$ the model diverges, which agrees with theory(for example, [29]). We also introduce *Nesterov's accelerated gradient* (NAG)[30]. In NAG, gradient descent is modified such that

$$\begin{aligned} \boldsymbol{s}^{[k]} &= \boldsymbol{\beta}^{[k-1]} + \delta X^{\mathrm{T}} \left(\boldsymbol{y} - X\boldsymbol{\beta}^{[k-1]} \right) \\ \boldsymbol{\beta}^{[k]} &= \boldsymbol{s}^{[k-1]} - \eta \left(\boldsymbol{s}^{[k]} - \boldsymbol{s}^{[k-1]} \right) \end{aligned} \tag{18}$$

This method is useful particularly when the simple gradient descent exhibit slow convergence

From this result, we choose to use NAG for the calculation. The implementation of linear regression with FHE is executed using PySEAL[31]. In our implementation, around 4∼5 iterations are possible and the following is the result of our experiment. Here our dataset is made up of eight data whose dimension is five, i.e., $N = 8, D = 5$.

To obtain Fig.4 and Fig.5, we set parameters δ and η to be $D/\mathrm{tr}(X^{\mathrm{T}}X)$ and -0.5, respectively. Although division is not supported by FHE, approximation is possible by Newton's method. Figure.5 suggests that our naive implementation using PySEAL allows only $4 \sim 5$ iterations, which might not be sufficient. Therefore it is highly expected that if other ways to implement gradient descent, or other libraries such as HElib enable more iterations, then gradient descent would be a good candidate for achieving linear regression.

Moreover, as is evident from Fig.8, the more dimensions the system has, the more complicated calculation becomes and time consumption is lenearly denpendent on the dimension of dataset.

4.3.2 Matrix Masking Method

Finally we explore the validity of MMM. As shown in Fig.9, the result obtained by MMM is very acurate and

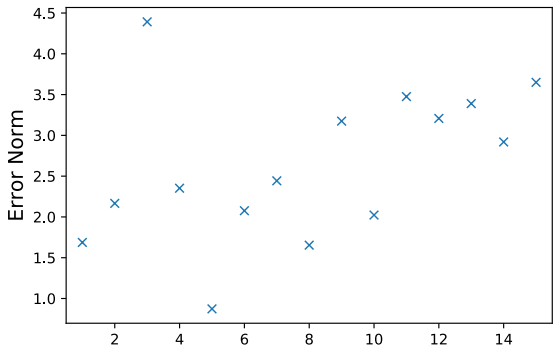

Figure 4: Plot of error norm $|\boldsymbol{y} - X\boldsymbol{\beta}|^2$ for 15 datasets.

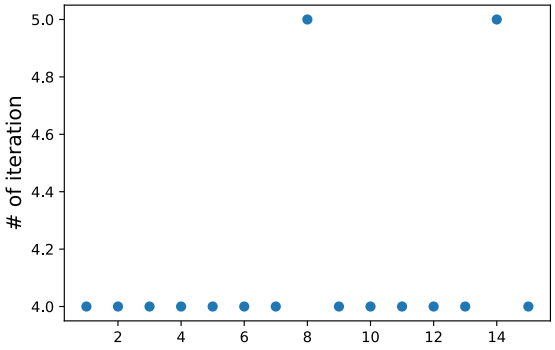

Figure 5: Plot of the number of iteration for each training phase

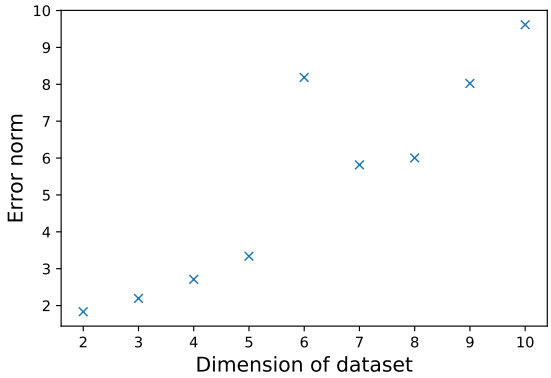

Figure 6: Plot of the average error norm $|\boldsymbol{y} - X\boldsymbol{\beta}|^2$ for datasets. with $2 \sim 10$ dimension of data.

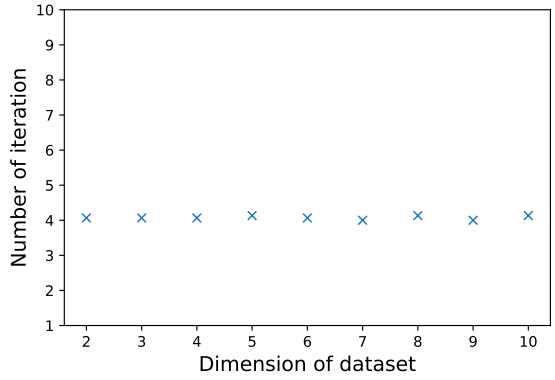

Figure 7: Plot of the average number of iteration for each training phase with $2 \sim 10$ dimension of data.

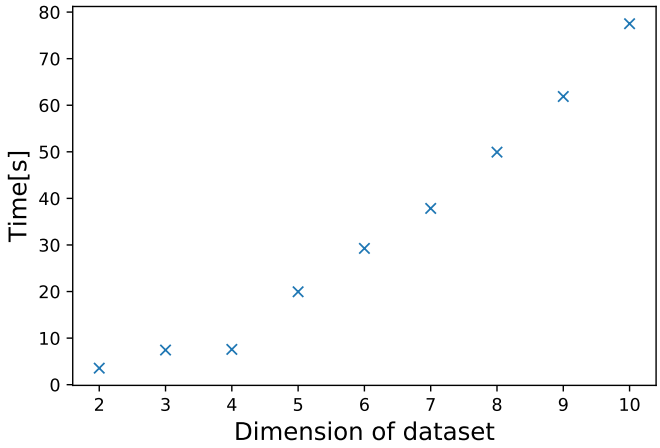

Figure 8: Plot of the average time for executing the program for given datasets with $2 \sim 10$ dimension of data

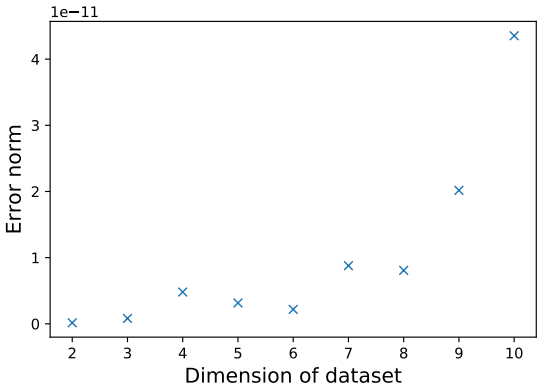

Figure 9: Plot of the average error norm $|\boldsymbol{y} - X\boldsymbol{\beta}|^2$ for datasets. with $2 \sim 10$ dimension of data.

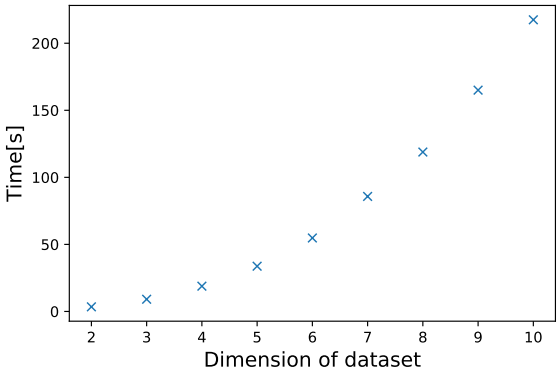

Figure 10: Plot of the average time for executing the program for given datasets with $2 \sim 10$ dimension of data

almost exact. Figure 10 implies time needed to execute linear regression by MMM is quadratically increasing. This can be explained as follows: since MMM requires matrix multiplication with matrix, therefore one has to calculate D^2 elementwise multiplications. Compared with Fig.8, it seems that gradient descent is faster particularly when dimension is big, but we should take into account the fact that gradient descent requires many iteration and possibly bootstrapping.

5 Discussion

Here we discuss practicability of each method: Gradient descent and Matrix Masking Method.

Gradient descent: gradient descent is the simplest method we can use for the linear regression. This means implementation is straightforward. It is also secure against at least honest-but-curious entities. However, we should care about the implementation of FHE scheme since Sec. 4.3.1 shows naive implementation using PySEAL fails to obtain satisfactory results. Moreover, even if some other libraries can obtain good result, our experiment suggests that bootstrapping should be used and it might take a long time to finish calculation. Collection of data also consumes noise budget since if local does not calculate encrypted matrix and encrypted vector locally, then the corresponding server has to calculate them instead. If the number of datasets is large, then just preparing encrypted matrix and vector take much of noise budget, which might lead to faiure in estimation of the model.

MMM: implementation of MMM is simple and this method avoids the most difficult part of the linear regression, inversion of matrix. Since CSP calculate inversion of matrix which is not encrypted, the result is exact and speedy calculation is expected. This method is also secure against at least honest-but-curious entites. Although one does not have to care about matrix inversion, this does not mean we can collect data from arbitrary number of data owners since addition consumes noise budgets although it is very small.

	Method	Computational Complexity	Security	Limitation of Calculation
Cramer's formula	Exact	$O(N^3N!)$	semi-honest	there is
Gradient Descent	Iterative	$O(N^2)$	semi-honest	there is
GC	Iterative	$O(N^2)^*$	semi-honest	No limitation
MMM	Exact	$\times$	semi-honest	$\triangle$

Table 1: Table of characteristics of each method introduced.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[2] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *ADV. IN CRYPTOLOGY, SPRINGER-VERLAG*, 1985.

[3] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[4] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[5] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348, May 2013.

[6] Fida Dankar. Privacy preserving linear regression on distributed databases. *Transactions on Data Privacy*, 8, 04 2015.

[7] Irene Giacomelli, Somesh Jha, Marc Joye, C. David Page, and Kyonghwan Yoon. Privacy-preserving ridge regression with only linearly-homomorphic encryption. *IACR Cryptology ePrint Archive*, 2017:979, 2017.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277, 2011.

[9] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[10] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[11] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[12] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, July 1985.

[13] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[14] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2009/616, 2009.

[15] N.P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. Cryptology ePrint Archive, Report 2009/571, 2009.

[16] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 116–137, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[17] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. Cryptology ePrint Archive, Report 2011/279, 2011.

[18] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

[19] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. Cryptology ePrint Archive, Report 2013/340, 2013.

[20] Leo Ducas and Daniele Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Report 2014/816, 2014.

[21] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Report 2016/870, 2016.

[22] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 36–54, London, UK, UK, 2000. Springer-Verlag.

[23] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. ACM.

[24] A. C. Yao. Protocols for secure computations. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 160–164, Los Alamitos, CA, USA, nov 1982. IEEE Computer Society.

[25] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.

[26] P. M. Esperança, L. J. M. Aslett, and C. C. Holmes. Encrypted accelerated least squares regression. *arXiv:1703.00839*, 2017.

[27] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, PODC '89, pages 201–209, New York, NY, USA, 1989. ACM.

[28] C. Wang, K. Ren, J. Wang, and Q. Wang. Harnessing the cloud for securely outsourcing large-scale systems of linear equations. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1172–1181, June 2013.

[29] V Ryaben'kii and S Tsynkov. *A Theoretical Introduction to Numerical Analysis*. Chapman and Hall/CRC, 2007.

[30] Yu. E. Nesterov. A method of solving a convex programming problem with convergence rate $o\left(\frac{1}{k^2}\right)$. *Dokl. Akad. Nauk SSSR*, 269:543–547, 1983.

[31] Alexander J. Titus, Shashwat Kishore, Todd Stavish, Stephanie M. Rogers, and Karl Ni. Pyseal: A python wrapper implementation of the seal homomorphic encryption library. *ArXiv*, abs/1803.01891, 2018.

[32] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 486–498, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[33] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *Cryptology and Network Security*, pages 1–20, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.