Genotype Imputation with Homomorphic Encryption

Fook Mun Chan
Chan_Fook_Mun@i2r.a-star.edu.sg
Institute of Infocomm Research, A*STAR
Singapore, Singapore

Ahmad Al Badawi
Ahmad_Al_Badawi@i2r.a-star.edu.sg
Institute of Infocomm Research, A*STAR
Singapore, Singapore

Jun Jie Sim
Sim_Jun_Jie@i2r.a-star.edu.sg
Institute of Infocomm Research, A*STAR
Singapore, Singapore

Benjamin Hong Meng Tan
Benjamin_Tan@i2r.a-star.edu.sg
Institute of Infocomm Research, A*STAR
Singapore, Singapore

Foo Chuan Sheng
foo_chuan_sheng@i2r.a-star.edu.sg
Institute of Infocomm Research, A*STAR
Singapore, Singapore

Khin Mi Mi Aung
Mi_Mi_Aung@i2r.a-star.edu.sg
Institute of Infocomm Research, A*STAR
Singapore, Singapore

ABSTRACT

Genotype imputation is a technique used to determine unobserved genomic markers when sequencing genomic data. This is a cost effective method for sequencing a genome. Due to the large amount of personal identifiable information involved in genomic imputation, there is a rising concern for analysis of such nature to be secure and private.

We describe a method using homomorphic encryption (HE) to perform genotype imputation in a secure and private setting. Our solution first involves training a logistic regression model and performing the imputation in the encrypted domain.

We have implemented our solution over using the open sourced Homomorphic Encryption library, SEAL. We are able to impute 500 SNPs within 5 minutes, with an accuracy of 97.3%.

CCS CONCEPTS

• **Security and privacy → Human and societal aspects of security and privacy**; **Pseudonymity, anonymity and untraceability**; • **Computing methodologies**;

KEYWORDS

homomorphic encryption, genomic imputation, logistic regression

ACM Reference Format:
Fook Mun Chan, Ahmad Al Badawi, Jun Jie Sim, Benjamin Hong Meng Tan, Foo Chuan Sheng, and Khin Mi Mi Aung. 2021. Genotype Imputation with Homomorphic Encryption. In *2021 6th International Conference on Biomedical Signal and Image Processing (ICBIP '21), August 20–22, 2021, Suzhou, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3484424.3484426

This research is supported by Institute for Infocomm Research, A*STAR Research Entities under its RIE2020 Advanced Manufacturing and Engineering (AME) Programmatic Program (Award A19E3b0099).

ICBIP '21, August 20–22, 2021, Suzhou, China

ACM ISBN 978-1-4503-9050-7/21/08…$15.00
https://doi.org/10.1145/3484424.3484426

1 INTRODUCTION

Genomic sequencing is widely used in medical domains to aid research. It is usually expensive to obtain a complete genome sequence. A common technique used is to sequence a subsequence of genetic markers called Single Nucleotide Polymorphism (SNPs) and 'guess' for the SNPs in between. Genotype imputation is a statistical technique used to determine these missing SNPs [10].

Homomorphic Encryption (HE) [6, 9] is a type of encryption that allows computation on encrypted data. where functions, f, can be evaluated on encrypted data $x_1, \ldots, x_n$, yielding ciphertexts that decrypt to $f(x_1, \ldots, x_n)$. In the context of genotype imputation, genomic data will be encrypted and sent to a computational server. The server then performs the genotype imputations on the encrypted data, before sending the encrypted outcome to the data owner for decryption. We argue that this would ensure that genomic data is protected: Throughout the entire workflow, the server cannot access the genomic data, ensuring the privacy of the data.

The iDASH Privacy & Security Workshop [2] is motivated by these security concerns and has organized several competitions on secure genomics analysis since 2011. The aim of these competitions is to evaluate methods that provide data confidentiality during analysis in a cloud environment. To the best of our knowledge, this is the first time genomic imputation has been done in the encrypted domain.

In this work, we provide a solution to Track 2 of the iDASH 2019 competition – *Secure Genotype Imputation using Homomorphic Encryption.* The challenge in this task was to impute 500 SNPs based on two datasets. The datasets contain SNPs that are spaces approximately 10,000 positions and 1000 position apart respectively. Participants would use build a model each for both datasets and perform the imputation using both models. This task seeks to advance the practical boundaries of HE in the medical domain, a continuation from last year's HE task which was to implement a genomic wide association study with HE. We wish to highlight that this challenge is a proof-of-concept for secure imputation and the imputation is performed on a small part of the human genome.

Within the constraints of the competition, including a virtual machine with 16GB of memory and 200GB disk space, a security level of at least 128 bits and at most 10 minutes of runtime, our solution reported a total computation time of less than 10 minutes with an accuracy of 97.3%.

In the following section, we will first describe in detail the challenge. We then describe our methods for training the model that is used in the the imputation of 500 SNPs. We show how the model is used to impute the SNPs, given some unseen encrypted SNPS. Finally, we present our results for the iDASH 2019 competition.

2 PRELIMINARIES

Description of the data provided

The organizers provided two sets of SNPs of chromosome 1 from 2504 individuals spaced approximately 10, 000 positions and 1000 positions apart. The training data is obtained from 1000 Genome Project [1]. These SNPs are referred to as tag-SNPs. The datasets were provided as matrices of size 1045×2508 and 9746×2508 respectively. Each row represents a SNP. The first four columns denotes the chromosome number, the start position, end position and the SNPS variant name. The remaining columns describe the the SNP genotype from the 2504 train-participants. The genotypes are encoded as $0, 1, 2$ depending if it is homozygous reference, heterozygous or homozygous alternate.

Another list of 500 SNPs, subsequently referred to as target-SNPs, were provided. These SNPs come from the same 2504 individuals that provided the tag-SNPs. The target-SNPs are presented as a matrix of size 500×2508. This matrix is structured in a similar manner as the tag-SNPs.

Description of Competition Task

Using the tag-SNPs, build a model that predicts the corresponding target-SNPs for each individual. A new set tag-SNPs from less than 500 unseen test-participants will be used to evaluate the model [2]. The model will impute the corresponding target-SNPs for these unseen test-participants. Solutions will be evaluated based on imputation accuracy. The micro-AUC will be computed and solutions will be ranked based on their micro-AUC score. The total round trip time (encryption, computation, decryption) is capped at 10 minutes [2]. An outline of the task is shown in Figure 1.

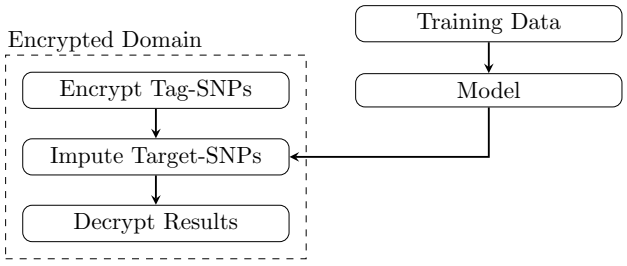

Figure 1: Outline of Competition Task

3 METHODS

Training the Model

We trained a separate model for each of the 500 target-SNPs. We used the logistic regression classifier from scikit-learn [7] to train the model. Logistic regression was chosen as the main component of the inference procedure is a dot product, which can be implemented in HE efficiently. However, non-linear functions cannot be implemented in an efficient manner is HE due to the noise budget in the scheme. In order to evaluate an non-linear function, the noise budget has to be set such that it is able to accommodate the total multiplicative depth. This would affect the performance of the HE scheme. We design a model based on the following flags:

- Windowing
- Feature Selection
- One-Hot Encoding
- Cross Validation

Figure 2 shows an outline of how the model is trained based on the various flags.

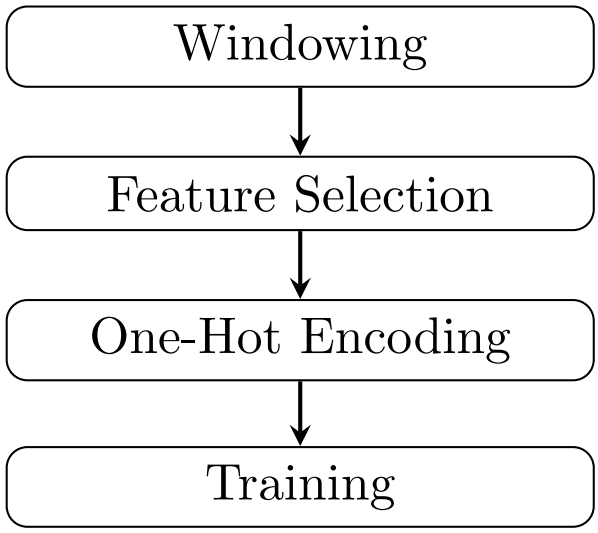

Figure 2: Outline of Model Training

Windowing. To account for linkage disequilibrium (LD) [11] between SNPs, we decide to select a region of tag-SNPs surrounding the target-SNP. We consider a window of 1MB, as suggested by Pritchard and Przeworski [8] to be a close distance. Thus, for each target-SNP, we have a different subset of tag-SNPs which is used for training.

One-Hot Encoding. We apply a one-hot encoding to encode the genotype values, 0, 1, and 2 as $[0, 0, 1]$, $[0, 1, 0]$, and $[1, 0, 0]$ respectively. This is to ensure that the magnitude of the genotype values do not affect the training of the model.

Feature Selection. Using the feature selection module from scikit-learn, we chose 50 features that will be used to train the model. The score function used was the chi square function. We apply this after Windowing and One-Hot Encoding.

Cross Validation. Cross validation further partitions the training dataset and aggregates sub-models to improve prediction performance against an unseen dataset. We use an implementation of logistic regression with cross validation provided by scikit-learn.

The Model. For each SNP, the model has two components, the coefficients matrix and an intercept vector. We denote the coefficients matrix as such

$$\begin{bmatrix} \text{class0\_coeff\_0}, & \text{class0\_coeff\_1}, & \dots & \text{class0\_coeff\_x} \\ \text{class1\_coeff\_0}, & \text{class1\_coeff\_1}, & \dots & \text{class1\_coeff\_x} \\ \text{class2\_coeff\_0}, & \text{class2\_coeff\_1}, & \dots & \text{class2\_coeff\_x} \end{bmatrix}.$$

The intercept vector is denoted as such

$$\begin{bmatrix} \text{class0\_intercept} \\ \text{class1\_intercept} \\ \text{class2\_intercept} \end{bmatrix}.$$

Homomorphic Encryption

Homomorphic Encryption (HE) enables the evaluation of functions on encrypted data, without decrypting the data [6, 9]. The encryption process injects some masking noise into the ciphertext. This results in an increase in the noise level as more operations are performed on the ciphertext. Additions increase the noise level by a relatively small amount as compared to multiplications. This affects the precision of the encrypted data. Often, a noise budget is introduced upon initialization and upon exhausting the budget, no more computations are allowed. The noise budget is thus closely related to the multiplicative depth of the function - the more multiplications performed, the faster the noise budget is consumed.

In this implementation, we use the CKKS [5] scheme as it allows float operations on encrypted data. During the encryption step, the data is first scaled up by a factor. There is a unique function to CKKS that other HE schemes do not have, that is the **Rescale** function. This function helps to maintain the scale of the data after multiplication. This is akin to rounding in float operations [5]. Thus, this allows the CKKS scheme to support float operations.

It takes a predetermined precision number of bits p, More concretely, let L be the noise budget initialized. For every multiplication, subtract from the noise budget the integer p. If the noise level of a ciphertext is below p, the noise budget is said to be depleted.

A brief description of the CKKS scheme is provided below. We denote sampling s from a ring R according to distribution $\mathcal{D}$ by $s \leftarrow \mathcal{D}(R)$. Let $\mathcal{HWT}(h)$ denote the distribution that chooses a polynomial uniformly from $\mathcal{R}_{2^L}$, with h nonzero coefficients. Let $\mathcal{U}(\mathcal{R}_{2^L})$ denote the distribution that chooses a ring element uniformly from $\mathcal{R}_{2^L}$. Let $\mathcal{DG}(\sigma^2)$ denote the Discrete Gaussian distribution with mean 0, standard deviation σ.

- **KeyGen($1^\lambda, L$):**
  Let 2^L be the initial ciphertext modulus. Sample secret $s \leftarrow \mathcal{HWT}(h)$, random $a \leftarrow \mathcal{U}(\mathcal{R}_{2^L})$ and error $e \leftarrow \mathcal{DG}(\sigma^2)$. Set the secret key as $sk \leftarrow (1, s)$, public key as $pk \leftarrow (b, a) \in \mathcal{R}_L^2$ where $b = -a \cdot s + e \pmod{L}$. Finally, sample $a' \leftarrow \mathcal{U}(\mathcal{R}_{2^L})$, $e' \leftarrow \mathcal{DG}(\sigma^2)$ and set the evaluation key $evk \leftarrow (b', a')$, where $b' = -a' \cdot s + e' + L \cdot s^2 \pmod{2^{2L}}$.
- **Encrypt(pk, m):**
  For a message $m \in \mathcal{R}$, sample $v \leftarrow \mathcal{U}(\mathcal{R}_{2^L})$ and $e_0, e_1 \leftarrow \mathcal{DG}(\sigma^2)$. Let $v \cdot pk + (m + e_0, e_1) \pmod{2^L}$ and return (v, L).
- **Decrypt(sk, ct):**
  For $ct = ((c_0, c_1), \ell) \in \mathcal{R}_\ell^2$, return $c_0 + c_1 \cdot s \pmod{2^\ell}$
- **Add(ct_1, ct_2):**
  For $ct_1 = ((c_{0,1}, c_{1,1}), \ell)$, $ct_2 = ((c_{0,2}, c_{1,2}), \ell)$, compute $(c'_0, c'_1) \leftarrow (c_{0,1}, c_{1,1}) + (c_{0,2}, c_{1,2}) \pmod{2^\ell}$ and return $(c'_0, c'_1), \ell)$.
- **Mult(ct_1, ct_2):**
  For ciphertexts $ct_1 = ((c_{0,1}, c_{1,1}), \ell)$ and $ct_2 = ((c_{0,2}, c_{1,2}), \ell)$, let $(d_0, d_1, d_2) = (c_{0,1}c_{0,2}\ ,\ c_{1,1}c_{0,2} + c_{0,1}c_{1,2}\ ,\ c_{1,1}c_{1,2}) \pmod{2^\ell}$. Compute $(c'_0, c'_1) \leftarrow (d_0, d_1) + \lfloor 2^{-L} \cdot d_2 \cdot evk \pmod{2^\ell} \rceil$ and return $(c'_0, c'_1), \ell)$.
- **Rescale(ct, p):**
  For a ciphertext $ct = ((c_0, c_1), \ell)$ and an integer $p \le \ell$, return $((c'_0, c'_1), \ell - p)$, where $(c'_0, c'_1) \leftarrow \lfloor 2^{-p} \cdot (c_0, c_1) \rceil \pmod{2^{\ell - p}} \rceil$.

An optimization by Smart & Vercauteren [12] allows more data to be packed within a ciphertext. A ciphertext can thus be viewed as an encrypted array. Each element of the encrypted array is referred to as a *slot*. In addition, there is a SIMD structure where computations can be performed across all slots simultaneously. This allows encrypted computations to be done efficiently.

Homomorphic Imputation

We impute the target-SNPs of the test-participants securely by encrypting the corresponding tag-SNPs. The model coefficients, however, are not encrypted. Our implementation leverages on a method to pack the model coefficients for fast imputation.

Packing method. We encode each coefficient into a Plaintext object, repeating it across as many slots as the number of test-participants. Since there are 500 test participants and approximately 2000 slots, we can pack the coefficients for the other classes (a total of 1500 coefficients) into the same Plaintext object.

$$\text{CoeffPtxt\_(i)} = \Big[\underbrace{\text{class0\_coeff\_i}}_{\text{500 copies}}, \underbrace{\text{class1\_coeff\_i}}_{\text{500 copies}}, \underbrace{\text{class2\_coeff\_i}}_{\text{500 copies}}, 0, \dots, 0\Big].$$

The intercepts are encoded in the same manner

$$\text{InterPtxt} = \Big[\underbrace{\text{class0\_intercept}}_{\text{500 copies}}, \underbrace{\text{class1\_intercept}}_{\text{500 copies}}, \underbrace{\text{class2\_intercept}}_{\text{500 copies}}, 0, \dots, 0\Big].$$

We encrypted the unseen tag-SNPs in a different manner. An array containing the same feature from all test-participants is first constructed. The array will be concatenated by itself two more times. This is to process the tag-SNPs of all the test-participants simultaneously across all classes.

$$\text{Ctxt\_(i)} := \Big[\underbrace{\text{P0\_feature\_i}, \text{P1\_feature\_i}, \dots \text{P500\_feature\_i}}_{\text{3 copies}}\Big].$$

With this packing method, we are able to compute the dot product with 1 homomorphic multiplication. This would result in a faster computation time and smaller HE parameters as the multiplicative depth of the circuit is small.

Our Algorithm. The homomorphic imputation of a target-SNP is essentially evaluating the corresponding logistic regression model that comprises of a dot product and a nonlinear normalization function.

As described in the packing method above, we are able to evaluate all test-participants simultaneously. The slots of the resultant ciphertext contain the scores whether the imputed genotype is 0, 1 or 2 for each participant. Applying the normalization function,

Algorithm 1: `Homomorphic Imputation`

Input: CoeffPtxt_(i), Ciphertext_(i), InterceptPtxt
Output: results

```
for each feature i do
    results ← Ciphertext_(i) * CoeffPtxt_(i)
end
results += InterceptPtxt
```

the softmax function, on each score will transform the score into probabilities.

We did not evaluate the softmax function in the encrypted domain. For this implementation, the softmax function was evaluated in the clear upon decryption. This is well within the competition rules [1].

4 RESULTS

We used the provided dataset of 2504 users and imputed SNPs on both the 10k (1045 tag-SNPs) and 1k dataset (9746 tag-SNPs). To test the accuracy of our model, we split the tag-SNPs into two, using 90% of it to train the model and evaluate its accuracy on the remaining 10%. This gives us an evaluation set of 250 users.

The HE library used is the SEAL library [3]. The HE parameters used are $n = 4096$, $\log Q = 90$. This gives us 2048 slots within a ciphertext. Furthermore, we run **Rescale** on the output with $\log p = 30$ for multiplications to control noise growth. This gives us a security level of about 158 bits based on the LWE estimator provided by Albrecht *et. al.* [4]. Our choice of parameters were mainly motivated to pack all 1500 SNPs - 500 SNPs to predict each class 0, 1, 2. We wish to highlight that if the number of SNPs to be imputed were to be increased, one can simply use more ciphertext, one for each class, instead of increasing n, thereby increase the number of slots to accommodate all SNPs. This is because increasing n would result in a slower computation time for the same $\log Q$.

For our server, the CPU model used is Intel Xeon Platinum 8170 CPU at 2.10GHz with 26 cores and the OS used is Arch Linux. We have restricted our computations to 4 cores for the encrypted imputation.

Our results are shown in Table 1 and Table 2.

Table 1: Time Taken for 10k dataset

Process	Time Taken (sec)	Memory (GB)
Encoding Model	7.323	0.0318
Encryption	1.344	0.296
Imputation	14.662	0.131
Decryption	0.283	0.0125

The accuracy of our models under different flags are shown below in Table 3.

5 CONCLUSIONS

In this paper, we describe our implementation for imputing genotype for encrypted data. We used a packing method that reduces the depth of the homomorphic circuit. We did not perform any normalization in the encrypted domain.

Our work has demonstrated that it is possible to perform genotype imputation in a secure manner. One possible future direction would be imputing a whole genome.

Table 2: Time Taken for 1k dataset

Process	Time Taken (sec)	Memory (GB)
Encoding Model	2.737	0.0220
Encryption	11.516	2.714
Imputation	6.132	0.0975
Decryption	0.378	0.0126

Table 3: Accuracy of Model Under Different Flags

Windowing	One-Hot Encoding	Feature Selection	Cross Validation	# Features	Accuracy
0	0	0	0	1045	97.38%
1	0	0	0	522	95.60%
0	1	0	0	3135	92.32%
0	0	1	0	50	97.38%
0	0	0	1	1045	94.73%
1	1	0	0	1566	94.26%
1	0	1	0	50	97.49%
1	0	0	1	522	96.50%
0	1	1	0	50	97.52%
0	0	1	1	50	97.33%
0	1	0	1	3135	95.12%
1	1	1	0	50	97.57%
1	0	1	1	50	97.46%
1	1	0	1	1566	96.33%
0	1	1	1	50	97.45%
1	1	1	1	50	97.51%

ACKNOWLEDGMENTS

This research is supported by Institute for Infocomm Research, A*STAR Research Entities under its RIE2020 Advanced Manufacturing and Engineering (AME) Programmatic Program (Award A19E3b0099).

REFERENCES

[1] [n.d.]. FAQ for iDASH Privacy Protection competition. https://docs.google.com/document/d/1bVkRyrYQFbhpJdIdBgb4vkOeXpv5h51dkQuqw-y3U-s/edit Last Accessed 28 May 2021.
[2] [n.d.]. iDASH Privacy & Security Workshop 2019. http://www.humangenomeprivacy.org/2019/competition-tasks.html Last Accessed 28 May 2021.
[3] 2019. Simple Encrypted Arithmetic Library (release 3.2.0). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA., Last Accessed 18 July 2019, commit e5fee8f.
[4] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. Cryptology ePrint Archive, Report 2015/046. https://eprint.iacr.org/2015/046.
[5] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2016. Homomorphic Encryption for Arithmetic of Approximate Numbers. Cryptology ePrint Archive, Report 2016/421. http://eprint.iacr.org/2016/421.
[6] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *41st ACM Symposium on Theory of Computing*. ACM Press, 169–178.
[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[8] Jonathan K. Pritchard and Molly Przeworski. 2001. Linkage Disequilibrium in Humans: Models and Data. *The American Journal of Human Genetics* 69, 1 (2001), 1–14. https://doi.org/10.1086/321275
[9] R L Rivest, L Adleman, and M L Dertouzos. 1978. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation, Academia Press* (1978).

[10] P. Scheet and M. Stephens. 2006. A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and haplotypic phase. *Am J Hum Genet* 78, 4 (Apr 2006), 629–644.

[11] M. Slatkin. 2008. Linkage disequilibrium–understanding the evolutionary past and mapping the medical future. *Nat Rev Genet* 9, 6 (Jun 2008), 477–485.

[12] N.P. Smart and F. Vercauteren. 2011. Fully Homomorphic SIMD Operations. Cryptology ePrint Archive, Report 2011/133. https://eprint.iacr.org/2011/133.