






Homomorphic evaluation of large look-up tables for inference on human genome data in the cloud

Antonio Guimarães
Institute of Computing
University of Campinas
 Campinas, Brazil
 0000-0001-5110-6639

Leonardo Neumann
Institute of Computing
University of Campinas
 Campinas, Brazil
 0000-0002-4409-6118

Fernanda A. Andaló
Institute of Computing
University of Campinas
 Campinas, Brazil
 0000-0002-5243-0921

Diego F. Aranha
Department of Computer Science
Aarhus University
 Aarhus, Denmark
 0000-0002-2457-0783

Edson Borin
Institute of Computing
University of Campinas
 Campinas, Brazil
 0000-0003-1783-4231

Abstract—iDash is an annual competition for creating new solutions to tackle the challenges of securing human genome processing in untrusted environments, such as the public cloud. In this work, we propose and analyze a simple but efficient candidate for the homomorphic encryption track of iDash 2022. We focus on different approaches for optimizing its homomorphic evaluation without any loss of precision compared to its cleartext evaluation. We represent our data inference model as a single large look-up table (LUT), which we homomorphically evaluate without model-specific optimizations. In this way, our results represent not only the model we chose but all others that could be represented as a LUT of similar size. We employ three different approaches for encrypting data that trade-off large ciphertexts (and, thus, high demands for storage and IO) and computational performance. We evaluate our solutions in two cloud environments similar to the reference provided by the iDash competition. As a result, we not only show the practicability of our solution in the context of iDash but also provide key insights on the practical issues of employing popular homomorphic encryption techniques, such as LUT evaluation, in a real-world scenario.

Index Terms—fhe, fully homomorphic encryption, decision tree, idash, human genome processing

I. INTRODUCTION

The implications of human genome data sharing are a growing concern for the scientific community, which is working towards the development of collaboration standards and frameworks [14]. Privacy is one of the utmost issues, as the consequences of unrestricted sharing can be disastrous considering the sensitive nature of the source data as well as the inferences that can be made by combining it with other databases [15]. Furthermore, any data disclosures affect not only individual data owners but also their blood relatives. This problem is further aggravated by the increasing use of cloud

computing for providing scalable genomic data processing, as pointed out by the USA National Institutes of Health [16].

On the other hand, several research fields could leverage large genomic databases to advance human health technologies. In Personalized Medicine [4], it is possible to correlate genotypic and phenotypic data for providing targeted diagnoses and treatments for diseases. This process, however, requires not only the patients to share their genomic sequencing information but also the establishment of large databases to carry out statistical analyses. Data inference models trained from these databases are also sensitive and may reveal information about the training dataset. In this context, Fully Homomorphic Encryption (FHE) [10] comes as an ideal solution, allowing evaluating these models over encrypted data, without revealing any information about the user input or the model parameters.

In 2014, the iDash competition [1] was created among the efforts of the scientific community towards seeking new solutions for securing genome data processing in cloud environments. In 2022, its homomorphic encryption track proposes an instantiation of genotype-to-phenotype inference over encrypted data as a challenge. In particular, the inference must be performed from high throughput genomic data without any cleartext preprocessing. In this way, the challenge goes beyond the training of an inference model and its homomorphic evaluation. It is also necessary to handle a relatively large amount of encrypted data, which, as we show in this paper, presents issues on its own.

Contributions: In this work, we present our submission for the homomorphic encryption track of iDash 2022 and analyze it considering different aspects and execution environments. Our focus is mainly on the homomorphic evaluation and handling of encrypted data. We firstly trained a simple inference model based on decision trees of up to depth 5. Then, we investigated efficient ways of homomorphically evaluating it. While our model itself might not be on par with other more

This work was partially performed when the first author was visiting the Department of Computer Science at Aarhus University. It was supported by the São Paulo Research Foundation under grants 2013/08293-7, 2019/12783-6, and 2021/09849-5; by the National Council for Scientific and Technological Development under grant 313012/2017-2; and by the Independent Research Fund Denmark (DRF) project no. 1026-00350B.

elaborate solutions, we evaluate it in a generic way using a popular and flexible homomorphic evaluation approach: the evaluation of look-up tables (LUT) [2], [3], [6], [11]. In this way, the results we obtain in this work and the conclusions we draw from them are not limited to our model. Instead, they can be applied to many other models in the same context, including those that are not based on decision trees. We analyze three different methods for data encryption and their trade-off in terms of storage and execution time in our inference model; and we evaluate the impact of the cloud environment in the methods. All of our experiments are in the context of the iDash competition and consider the restrictions imposed by the challenge.

The rest of this document is organized as follows: Section II describes the basic background on homomorphic encryption, the TFHE scheme, and the homomorphic evaluation of LUTs; Section III introduces the iDash competition, its current challenge, and the high-level of our solution; Section IV details our solution; Section V presents the experimental results; and Section VI concludes the paper.

II. HOMOMORPHIC ENCRYPTION

Cryptography has always been an important tool when handling sensitive information. It can be used to provide formal guarantees of privacy and is broadly used to protect data at rest. During processing, however, data might be equally vulnerable while protecting it presents many additional challenges. An ideal solution is in the use of FHE, which allows performing computation over encrypted data [10]. However, although functionally powerful, it still lacks practical performance levels for many applications.

A. The TFHE and GSW schemes

TFHE [6] is a FHE scheme based on the (Ring) Learning with Errors problem [17]. It is the current state-of-the-art for the evaluation of non-linear functions and works by evaluating small logic components such as binary logic gates and look-up tables (LUTs). In this section, we describe TFHE in terms of high-level operations, taking most of its procedures as black boxes. TFHE encrypts messages in three types of ciphertexts:

TLWE Sample: A pair (a, b) , where a is a vector of n scalars, and b is a scalar given by the equation $b = \langle a, s \rangle + m + e$, where s is the secret key, m is the message, and e is a small error. Angle brackets denote the inner product. Originally, each scalar is an element in the real Torus \mathbb{T} (the set of real numbers modulo 1), but, for our application, we can consider each scalar is an integer in \mathbb{Z}_q , the set of integer numbers modulo q . Notice that an error is added to the message, which is necessary for security. To prevent this error from affecting the message, one usually scales the message by a factor δ greater than the error amplitude before encrypting.

TRLWE Sample: A pair (a, b) , where a is a polynomial of degree $N - 1$ in a cyclotomic ring, and b is also a polynomial of degree $N - 1$ given by the equation $b = as + m + e \bmod (X^N + 1)$, where s , m , and e are, respectively, the secret key, the message, and a small error. Notice that all elements,

including the message, are polynomials of degree $N - 1$, with coefficients in \mathbb{Z}_q .

TLWE and TRLWE samples can be directly added, which will result in a sample of the added messages with a greater error (but which is hopefully still too small to affect the message). However, they do not support multiplication.

TRGSW Sample: A set of 2ℓ TRLWE Samples. In terms of encryption, it has the same capabilities as TRLWE samples (*i.e.*, it encrypts messages encoded as polynomials of degree $N - 1$). However, contrary to them, it allows direct (internal) multiplications between TRGSW ciphertexts and for external products with TRLWE samples, which will result in a TRLWE sample encrypting the multiplication of the messages modulo $(X^N + 1)$. The first ℓ samples of a TRGSW sample encrypt the message, each one scaled by a different power of two in $\{2^0, 2^{B_g}, \dots, 2^{B_g(\ell-1)}\}$. This encoding is broadly used in homomorphic encryption for performing multiplications with small error growth through a technique named *gadget decomposition* [9], where B_g is the decomposition base, and ℓ is the decomposition level. The last ℓ samples of a TRGSW sample encrypts the same content as the first ℓ samples, but multiplied by $-s$ (the negation of the secret key).

B. Managing the error growth

As defined in the last section, each ciphertext presents a very small error added for security. When performing arithmetic, this error increases and eventually affects the message. There are two main approaches for dealing with this issue. In the *leveled setting*, one defines the encryption parameters already considering the function to be evaluated. In this way, it is possible to plan for the error to grow only up to a certain amplitude, which one knows will not (significantly) affect the message. This approach presents better performance for functions with small multiplicative depth, but it would require very large parameters to evaluate some more complex circuits. In the *bootstrapped setting*, one homomorphically evaluates the decryption function to reset the error of a ciphertext. In this way, there are no limits for the multiplicative depth of the function being evaluated, but bootstraps (*i.e.*, the homomorphic evaluation of the decryption function) need to be executed often to prevent the error from growing too much, and they are usually expensive.

C. Look-up table (LUT) evaluation

Look-up tables are a convenient way of representing and evaluating functions using homomorphic encryption. There are several ways of doing so. In this section, we detail the basic procedures introduced by and used in schemes such as FHEW [8] and TFHE [6].

Let L be a look-up table (LUT) of size l . A LUT evaluation on L with input (selector) x consists of looking up for the x -th element of L . Homomorphically, the simplest way of evaluating it is to encrypt L in a TRLWE sample, with the value of each slot of L being encrypted in a monomial of m . For example:

$$[m_0, m_1, m_2, m_3] \rightarrow \text{TRLWE}(m_3X^3 + m_2X^2 + m_1X^1 + m_0)$$

Then, the selector x is encrypted in the exponent of a TRGSW sample as follows:

$$x \rightarrow \text{TRGSW}(X^{2^N-x})$$

To perform the look-up, we multiply the two encryptions. Let $x = 1$, the result of the multiplication would be:

$$\text{TRLWE}(-m_0X^3 + m_3X^2 + m_2X^1 + m_1)$$

TFHE provides a procedure called *SampleExtract* which allows extracting a TLWE sample encrypting a monomial from a TRLWE sample. In this way, we get the LUT evaluation result:

$$\text{SampleExtract}_0(\text{TRLWE}(-m_0X^3 + m_3X^2 + m_2X^1 + m_1)) \rightarrow \text{TLWE}(m_1)$$

This is a toy example where we consider $l = N = 4$. Real parameters are provided in Table II. While this solution is very efficient, with one $\text{TRGSW} \times \text{TRLWE}$ multiplication taking just a few tens of microseconds, it has some limitations. Firstly, LUTs are limited by the size of N . Secondly, the selector is encrypted in the exponent of a TRGSW sample, which is often expensive to obtain in real-world scenarios [5]. These limitations are overcome by techniques such as the *vertical packing* and *blind rotate* [6], which are based on the evaluation of CMUXs.

CMUX-based evaluation: A CMUX [6] is a small multiplexer that receives two TRLWE samples a and b , and a TRGSW sample C encrypting 0 or 1. It returns a if $C = 0$, and b otherwise. The *blind rotate* procedure [6] evaluates LUTs by encrypting them in TRLWE samples (similarly as aforementioned), rotating them by multiplying to $X^{2^N-2^i}$, for $i \in [0, \log_2(N))$, and selecting between the rotations and the original polynomial by using a CMUX gate. In our example with the selector $x = 1$, x would be encrypted in two TRGSW samples (one for each bit) $c_0 = \text{TRGSW}(1)$ and $c_1 = \text{TRGSW}(0)$. Let c_L be the TRLWE sample encrypting the LUT L . The *blind rotate* would first multiply the c_L by X^{2^N-1} , and use a CMUX to select between c_L and $c_LX^{2^N-1}$ using c_0 . Then, it would multiply the result by X^{2^N-2} , and perform the selection based on c_1 . The number of multiplications is now $\log_2(N)$ times greater, but this method allows the selector to be encrypted bit by bit, which is usually easier or cheaper to obtain in real-world applications. To evaluate a LUT larger than N , one can encrypt it in several TRLWE samples and select the desired ones through a sequence of CMUX gates. This process is described as *vertical packing* and is further detailed in Section 5.1 in [6].

III. THE IDASH COMPETITION

The iDASH Healthcare Privacy Protection Challenge [1] is an annual competition promoted by healthcare-related government agencies, the industry, and the scientific community in an effort to, as they define, “*evaluate the performance of state-of-the-art methods that ensure rigorous data confidentiality during data analyses in a cloud environment*” [1]. Recent editions are split into three tracks: blockchain, homomorphic encryption, and confidential computing.

A. The 2022 Homomorphic Encryption Track

The task for the Homomorphic Encryption Track of iDash 2022 [13] is to perform phenotype prediction from high throughput genomic data. Essentially, we are provided with genotypes for 200 individuals, each of them containing 20390 features, and we need to predict values for 5 different phenotypes in 30 minutes. We are free to choose any inference model, which we can train on a dataset with 3000 individuals. Each feature is a value in $\{0, 1, 2\}$, the first three phenotypes are floating-point values, and the last two are binary values. The challenge requires schemes to provide 128-bit security and the entire solution to run in less than 30 minutes while taking at most 500 GB of storage. The evaluation system is also limited to 4 CPUs and 32 GB of RAM. No pre or post-computation is allowed. All input data and all parameters of the trained model must be encrypted. Each solution should present the following workflow:

- 1) *Client*: Sets up the cryptosystem, generates and saves the keys, encrypts the input data.
- 2) *Modeler*: Using the client’s private key, it generates and encrypts the model parameters.
- 3) *Evaluator*: Receives the client’s encrypted input, the modeler’s encrypted parameters, and evaluates the model inference, producing the encrypted output. It only has access to encrypted data and public information (e.g. public evaluation keys). Feature selection (i.e. selecting just a subset of the input features to working over) is allowed to be performed using public information.
- 4) *Client*: Finally decrypts the results.

Furthermore, the Client, Modeler, and Evaluator are required to be separate processes communicating through files.

B. Our model

Challenges in iDash always come in two parts: “*How to get the best inference model*” and “*how to evaluate this model homomorphically*”. The scope of our work is limited to the second. Therefore, we considered a basic inference model based on relatively shallow (depth 5) decision trees. Our model achieves accuracy (measured in 1-NRMSE, 1 minus the normalized root mean squared error, and AUC, Area under the ROC Curve, as defined by the challenge [13]) from 81% up to 96%, which might not beat more elaborate models designed for the problem (The competition is still ongoing at the time we are writing this paper). However, when homomorphically evaluating the model, we preserve bit precision throughout the entire execution, so that it represents any model of similar size. Our homomorphic evaluation approach treats the tree decisions as just 32 bits of data, without any assumptions on the type of model data.

IV. HOMOMORPHIC EVALUATING OUR SOLUTION

As each of the input features is represented in 2 bits, and our 5-depth decision trees look at most 11 features, we encoded the entire tree evaluation as a LUT with 2^{22} slots, each one storing a 32-bit floating-point number. Notice that there are

two very obvious optimizations that we could have done at this point:

- At the output: A 5-depth tree has at most 32 possible decisions. So we did not need to keep 32-bit floats as the output of the tree. This is also true (and even more drastic) for the binary phenotypes, which can only be 0 or 1.
- At the input: Each input feature can only assume values in $\{0, 1, 2\}$, and the tree does not necessarily check for all of them. In this way, a big part of the LUT is taken by positions that are never looked up.

We could have implemented such optimizations, but they would make our solution specific to the chosen inference model, which is something we are trying to avoid as we want our solution to be as generic as possible within the context of iDash. Notice that, in the way we designed it, the results we present in this paper represent not only other decision-tree-based models of the same size but all inference models that could be represented as a large LUT evaluation for the competition.

A. LUT evaluation

We evaluate the LUT using the blind rotate and vertical packing methods [6] introduced with the TFHE scheme, which we briefly described in Section II-C. For a complete description of the method, see Section 5.1 in [6]. This method is typically the most efficient way of evaluating LUTs in a leveled setting, as the $\text{TRGSW} \times \text{TRLWE}$ multiplications take just tens of microseconds and even relatively large LUTs require a small amount of them. In a bootstrapped setting, we could consider using the Tree-PBS approach [11] for smaller LUTs, but, for the sizes required by our model, the bootstrapped version of the vertical packing would also be the best choice [2]. As we mentioned in Section IV, we want our results to cover any model in iDash that could be represented as a LUT of similar size. Therefore, we also did not consider methods that target specifically the evaluation of decision trees, such as the SortingHat [7], which would present much better results.

For each of the 200 individuals, our evaluator works as follows:

- 1) It receives the encrypted input data and selects which features are relevant for the decision tree. From the 20390 features, only around 10 of them are considered for inference.
- 2) It generates an array of TRGSW samples, a process that we further discuss in the next section. Each sample encrypts a bit of the selected features.
- 3) It receives the encrypted model. The model is already represented as a LUT encrypted in an array of TRLWE samples.
- 4) Using the results of steps 2 and 3, it performs the evaluation itself, using the vertical packing and blind rotate methods [6].
- 5) It returns the result.

Encryption Approaches: The only undefined part of this workflow is also the most challenging part of our approach: How to generate an array of TRGSW samples, each encrypting a bit of the selected features. We implemented three approaches for that. In all of them, the client encrypts the features bit by bit:

- 1) **TRGSW-single.** Each bit is encrypted alone in a TRGSW sample. This seems to be the ideal solution for optimizing execution time at the tree evaluation, but it requires the client to generate more than 8 million TRGSW samples, which might require hundreds of gigabytes to be stored.
- 2) **TRLWE-packed.** Each bit is encrypted in a monomial of a TRLWE sample. Since each TRLWE sample has N monomials and N is usually around 2^{10} to 2^{14} , the client would need just around 4 to 10 thousand TRLWE samples to encrypt everything. Depending on the parameters, this method would require just a few hundred megabytes of storage. On the other hand, the evaluator would need to extract the bits from the TRLWE samples (which can be done using an inexpensive SampleExtract), and transform it into a TRGSW sample. This transformation requires a Circuit Bootstrap [5], which is a very expensive procedure.
- 3) **TRGSW-packed.** Each bit is encrypted in a monomial of a TRGSW sample. This is a middle-ground solution between the two other approaches. Since the features are already in TRGSW samples, it is not necessary to perform a Circuit Bootstrap. However, it is still necessary to isolate a monomial from the others (a single TRGSW sample is encrypting N bits of the features, and we want just one of them). This is accomplished by a sequence of key switching procedures [6] that zero the unselected monomials and reconstruct the TRGSW sample. These key switchings are faster than a Circuit Bootstrap, but this method also requires 2ℓ times more storage.

V. RESULTS

A. Execution environment and parameters

Table I shows the two execution environments we experimented and the reference specifications provided by the iDash competition. We chose `d3.xlarge` and `i4i.xlarge` instances as they are the closest to the reference machine we could find on AWS. Table II shows the cryptosystem parameters for each approach. In all cases, we built our implementations using the MOSFHET library [12], an optimized implementation of TFHE.

B. Storage

Table III shows the storage requirements of each approach. We did not include the encrypted model, since it does not vary according to the approach we choose to encrypt the input. Notice that the TRGSW-single approach almost reaches the 500GB limit specified by the challenge, whereas the packed approaches take just around 1% of the available storage.

TABLE I
EXECUTION ENVIRONMENTS. THE STORAGE TYPE (HDD/SSD) WAS NOT SPECIFIED FOR THE REFERENCE MACHINE.

	iDash Reference	d3.xlarge	i4i.xlarge
Processor	Intel 8180	Intel 8259CL	Intel 8375C
Frequency	2.5 GHz	2.5 GHz	2.9 GHz
CPUs	4	4	4
RAM	32 GB	32 GB	32GB
Storage	500 GB	5940 GB - HDD	937 GB - SSD

TABLE II
PARAMETERS FOR TFHE

	TLWE		TRLWE		TRGSW	
	n	σ	N	σ	B_g	ℓ
TRGSW-single	-	-	1024	2.99E-08	8	2
Others	632	3.05E-05	2048	5.68E-14	9	4

C. Performance

Tables IV and V presents the execution time, in seconds, for each of our test machines. Each result is the average of 5 executions, for which we calculated a 95%-confidence interval. We compiled our code using the default compiling options from MOSFHET [12] and measured time using Linux wall-clock timestamps (`date +%s`). Client and Modeler are single-thread processes while the evaluator is parallelized in 5 processes, one for each phenotype inference. Although the slightly higher clock frequency of the `i4i.xlarge` may have been an advantage, the differences in storage technologies are certainly a dominant factor in the comparison between machines.

TRGSW-single approach: While disk IO was the bottleneck for the client on the `d3.xlarge`, the same did not happen on the `i4i.xlarge`. The average storage writing speed was 281 MBps, significantly below the NVME writing capacity of more than 1 GBps. Considering this result, we could have parallelized client encryption and reduced the client execution time by up to 3 times. This would certainly make the TRGSW-single approach significantly faster than the alternatives. However, as the storage technology of the iDash reference machine was not specified, we could not follow with this optimization. In the `d3.xlarge`, for example, parallelization would not help as the client was already writing at 139 MBps, near the limit of the HDD writing bandwidth.

TRLWE-packed approach: As expected, client encryption became more than a thousand times faster, as each TRLWE sample is now encrypting $N = 2048$ bits of the features. The evaluator execution time, however, grew considerably, with a slowdown of 17.6 times on the `i4i.xlarge` and of 16.2 times on the `d3.xlarge`.

TRGSW-packed approach: The $\ell = 4$ times increment in the encrypted genotype size (compared to TRLWE-packed) was compensated by a 1.38 times speedup in the evaluator, without observable slowdowns on client encryption. At first, the 4 times increment in storage could be seen as a significant drawback. However, in practice, it requires just an additional 373.4 MB of storage, which even the HDD of the

TABLE III
STORAGE REQUIREMENTS OF EACH APPROACH.

	TRGSW-single	TRLWE-packed	TRGSW-packed
Private Key	8 KB	16 KB	16 KB
Eval. Key	0	5.1 GB	3.8 GB
Genotype	497.8 GB	124.4 MB	497.8 MB

TABLE IV
EXECUTION TIME, IN SECONDS, IN A `d3.xlarge`

	TRGSW-single	TRLWE-packed	TRGSW-packed
Client	3664.4 ± 13.2	69.2 ± 11.4	63.6 ± 1.6
Modeler	14.6 ± 1.0	12.0 ± 3.0	10.2 ± 1.3
Evaluator	102.4 ± 0.5	1804.0 ± 13.8	1300.0 ± 0.9

`d3.xlarge` can load in a bit less than 3 seconds. The 1.38 times speedup in execution time, on the other hand, represents at least 504 seconds, which is more than a hundred times the possible slowdown caused by the additional use of storage.

VI. CONCLUSION

In this paper, we introduced a proposal for the homomorphic encryption track of iDash 2022 and addressed the challenges of homomorphically evaluating it in a cloud environment. We chose homomorphic evaluation methods that are generic enough for our results to apply to other models and applications in a similar context. In our final submission to the competition, we used the TRGSW-packed approach for encrypting data, as its entire execution takes around 23 minutes to run, well within the 30-minute limit of the competition. Nonetheless, we note that the performance enabled by TRGSW-single encryption could be leveraged by other applications, as each LUT takes less than 0.1s to be evaluated on average, even considering all the overhead introduced by IO operations.

REFERENCES

- [1] About - IDASH PRIVACY & SECURITY WORKSHOP 2022 - secure genome analysis competition. <http://www.humangenomeprivacy.org/2022/about.html>. (Accessed on 09/23/2022).
- [2] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter Optimization & Larger Precision for (T)FHE. Cryptology ePrint Archive, Paper 2022/704, 2022. <https://eprint.iacr.org/2022/704>.
- [3] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New Techniques for Multi-value Input Homomorphic Evaluation and Applications. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 106–126, Cham, 2019. Springer International Publishing.
- [4] Isaac S. Chan and Geoffrey S. Ginsburg. Personalized medicine: Progress and promise. *Annual Review of Genomics and Human Genetics*, 12(1):217–244, 2011. PMID: 21721939.
- [5] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 377–408, Cham, 2017. Springer International Publishing.
- [6] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.
- [7] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V. L. Pereira. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. Cryptology ePrint Archive, Paper 2022/757, 2022. <https://eprint.iacr.org/2022/757>.

TABLE V
EXECUTION TIME, IN SECONDS, IN A I4I.XLARGE

	TRGSW-single	TRLWE-packed	TRGSW-packed
Client	1812.4 \pm 37.4	36.2 \pm 0.7	32.4 \pm 2.5
Modeler	4.6 \pm 0.5	4.4 \pm 0.5	4.2 \pm 0.4
Evaluator	113.8 \pm 5.8	1841.8 \pm 57.5	1332.0 \pm 50.7

- [8] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [9] Nicholas Genise, Daniele Micciancio, and Yuriy Polyakov. Building an Efficient Lattice Gadget Toolkit: Subgaussian Sampling and More. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 655–684, Cham, 2019. Springer International Publishing.
- [10] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.
- [11] Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):229–253, Feb. 2021.
- [12] Antonio Guimarães, Edson Borin, and Diego F. Aranha. MOSFHET: Optimized Software for FHE over the Torus. Cryptology ePrint Archive, Paper 2022/515, 2022. <https://eprint.iacr.org/2022/515>.
- [13] Miran Kim, Arif Harmanci, and Xiaoqian Jiang. Task 2 - Competition tasks - IDASH PRIVACY & SECURITY WORKSHOP 2022 - secure genome analysis competition. <http://www.humangenomeprivacy.org/2022/competition-tasks.html>. (Accessed on 09/23/2022).
- [14] Bartha Maria Knoppers. Framework for responsible sharing of genomic and health-related data. *The HUGO journal*, 8(1):1–6, 2014.
- [15] Tsung-Ting Kuo, Xiaoqian Jiang, Haixu Tang, XiaoFeng Wang, Tyler Bath, Diyue Bu, Lei Wang, Arif Harmanci, Shaojie Zhang, Degui Zhi, et al. iDASH secure genome analysis competition 2018: blockchain genomic data access logging, homomorphic encryption on GWAS, and DNA segment searching, 2020.
- [16] National Institutes of Health et al. NIH security best practices for controlled-access data subject to the NIH Genomic Data Sharing (GDS) policy. *NIH Office of Science Policy*, 2015.
- [17] Oded Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM*, 56(6), September 2009.