Efficient Privacy-Preserving Viral Strain Classification via k-mer Signatures and FHE

Adi Akavia
University of Haifa
adi.akavia@gmail.com

Ben Galili
Technion
benga9@gmail.com

Hayim Shaul
IBM Research
hayim.shaul@gmail.com

Mor Weiss
Bar-Ilan University
mor.weiss@biu.ac.il

Zohar Yakhini
Reichman University, Technion
zohar.yakhini@gmail.com

***Abstract*—With the development of sequencing technologies, *viral strain classification* – which is critical for many applications, including disease monitoring and control – has become widely deployed. Typically, a lab (client) holds a viral sequence, and requests classification services from a centralized repository of labeled viral sequences (server). However, such "classification as a service" raises privacy concerns. In this paper we propose a *privacy-preserving viral strain classification* protocol that allows the client to obtain classification services from the server, while maintaining *complete privacy* of the client's viral strains. The privacy guarantee is against active servers, and the correctness guarantee is against passive ones. We implemented our protocol and performed extensive benchmarks, showing that it obtains almost perfect accuracy (99.8%–100%) and microAUC (0.999), and high efficiency (amortized per-sequence client and server runtimes of 4.95ms and 0.53ms, respectively, and 0.21MB communication). In addition, we present an extension of our protocol that guarantees server privacy against passive clients, and provide an empirical evaluation showing that this extension provides the same high accuracy and microAUC, with amortized per sequences overhead of only a few milliseconds in client and server runtime, and 0.3MB in communication complexity. Along the way, we develop an enhanced packing technique in which two reals are packed in a single complex number, with support for homomorphic inner products of vectors of ciphertexts. We note that while similar packing techniques were used before, they only supported additions and multiplication by constants.**

***Index Terms*—privacy preserving, viral strain classification, fully homomorphic encryption**

I. Introduction

Comparing newly-sequenced viral strains to repositories of known variants serves important endeavors of discovering new variants, tracking variant distribution, etc. A notable example is the COVID-19 pandemic, where genomic sequencing is used to track new variants and guide the COVID-19 response. In a typical use case, a medical or biological lab uses sequencing technology [DIB97], [BMC+00] to extract a new viral sequence from a biological sample. The lab wishes to classify its newly-sequenced viral strain by comparing it against a repository containing a collection of observed strains of viridae sequences, which is held by an entity maintaining a centralized repository (e.g., ViPR [PSZ+12], [vpr]). Such centralized repositories then offer a service to the scientific community, allowing their data to be browsed for sequences similar to those the lab had sequenced, or offering machine learning services leveraging their repository.

This work was supported in part by ISF 3380/19

However, this naturally raises *privacy* and *intellectual property (IP) protection* concerns. Indeed, the centralized entity (the server) provides classification services to various labs (the clients) in a "machine-learning-as-a-service" business model. This business model prohibits the server from sending its classification model to the clients, because the server's commercial advantage relies on its unique proprietary knowledge (i.e., the data or model it holds, which is needed to obtain accurate classification). The clients likewise cannot send sequences in cleartext due to privacy and IP concerns. Indeed, sequence incidence in a lab may constitute sensitive information which, when coupled with auxiliary information from other sources (see e.g. [tra]), may be linked to individuals or regions; moreover, from an IP perspective, specific variants studied in labs, in certain clinical contexts, should remain secret until the study is published.

This naturally gives rise to the following question:

> *Can the server provide a* privacy-preserving *viral strain classification service with (1) a single interaction round, which (2) retains privacy of the client's sequence, and (3) maintains good performance?*

To achieve (1)-(2), the client can send the server its sequence encrypted using *Fully Homomorphic Encryption* (FHE) [RAD78], [Gen09], and the server can homomorphically classify the encrypted sequence. *The focus of this work lies on achieving also the third goal: attaining lightweight computation and communication without harming accuracy.*

A. Our Contribution

In this paper we propose the first privacy-preserving solution tailored for viral strain classification. Our solution achieves almost perfect accuracy (99.8%–100%) and microAUC (0.999) as well as fast runtime (milliseconds). Moreover, it is a single-round protocol, requiring no further interaction beyond the insecure baseline solution, and has low communication complexity (210KB per sequence).

Our work is the *first* to employ k-mer signatures, which are commonly used in non-private settings, in the context of *private string classification*. These k-mer signatures provide a concise sequence representation, compressing typical viral sequences by a factor of $15\times$–$30\times$.[1] Such a compression is

[1] Concretely, we compress viral sequences of typical length (10,000-120,000 nucleotide in $\Sigma = \{A, C, G, T\}$) to 6-mer or 7-mer signatures, reducing size from up to $120{,}000\log(|\Sigma|) = 240{,}000$ bits to at most 4^7 =8,192 bits.

particularly important in FHE-based protocols, since it translates to major gains in runtime and communication complexity.

Our solution aligns with clinical usage requirements, both in achieving excellent accuracy and microAUC – the main selection criteria when choosing a viral classification method – and in achieving fast runtime which is well within the clinically-acceptable regime, adding negligible overhead (milliseconds) over sample acquisition and sequencing time (hours).

Our privacy-preserving strain classification protocol is executed between a *client* (the lab in the use case described above) holding sequences (test data), and a *server* (the centralized repository) holding a dataset of viral sequences labeled according to type. The protocol employs FHE to allow the server to perform classification without learning anything about the test data. Specifically, upon termination of the protocol, the client learns estimates of the likelihood that each of its sequences belongs to each of the different types, whereas the server learns no information on these sequences (where privacy holds against active servers, and correctness against passive ones; See Section III). Our protocol attains high accuracy, comparable to classification on cleartext data, fast performance, and provides a rigorous security guarantee.

Theorem I.1 (Informal[2])**.** *Our privacy-preserving viral strain classification protocol (Figure 3) satisfies the following:*

High accuracy. *The client's output is ϵ-close to the output of the (non-private) baseline algorithm executed on cleartext data (see Algorithm 2), where ϵ can be made arbitrarily small via proper parameters settings.*[3]

Fast client and server. *The client only encrypts the test data (in k-mer representation, see Notation II.1) and decrypts the output. The server homomorphically evaluates a circuit of multiplicative depth $r_1 + r_2 + 2$ with $(r_1 + 2)\mathsf{s} + r_2$ multiplications, where r_1, r_2 are parameters controlling the approximation error, and s is the number of viral types. Concretely, in our empirical evaluation, $r_1 = r_2 = 1$, so that the circuit has multiplicative depth 4 and $3\mathsf{s}+1$ multiplications. The communication consists of the encrypted sequences, and the encrypted likelihood vectors.*

Security. *Privacy of client's input is guaranteed against any computationally-bounded malicious server. Correctness is guaranteed in the semi-honest model.*

Extension: privacy of server's data. Our protocol easily extends to additionally guarantee privacy of the server's data against passive clients. This is achieved by proper re-randomizing (so called, "sanitizing") the result ciphertext sent to the client to guarantee circuit privacy. See Section VI.

[2]For the formal statement and additional details, see Theorems IV.1 and IV.5, Corollary IV.7, Lemmas IV.1 and IV.3, and the experimental results (Section V).

[3]Elaborating on the above, the error $\epsilon = \epsilon_{\mathsf{FHE}} + \epsilon_{\mathsf{inv}}$ is the sum of two error terms: ϵ_{FHE} is the error caused by the FHE scheme, and only occurs when employing an *approximate FHE scheme* such as [CKKS17]; and ϵ_{inv} is caused by the use of a low-degree polynomial approximation of the inverse which is used in the homomorphic computation. Both of these error terms can be made arbitrarily small by an appropriate choice of the input precision of [CKKS17] and approximation degree, respectively.

Empirical evaluation. We implemented our protocol and performed extensive benchmarks, showing that it achieves excellent accuracy and fast performance. We present the key findings here; See details in Section V- VI.

High accuracy. Our protocol attains very high accuracy (99.8%–100%) and microAUC (0.999) on the Covid-19 dataset from the iDash competition 2021 [iDa]; See Section V, Table I. To further establish that our solution obtains high microAUC (and consequently, high accuracy) for other datasets as well, we tested our viral strain classification algorithm (Figure 2) on three additional viral datasets representing typical viruses (Hepatitis-C, Dengue and Herpesvirus), showing it has extremely high microAUC on all; See Section V, Figure 4a.

Fast performance. Our system exhibits fast performance. In batched classification over 2048 encrypted sequences, the amortized per-sequence performance is: Client's runtime 4.95ms and RAM 134KB; Communication 0.21MB;[4] Server's runtime 0.53ms and RAM 194KB; See Section V, Table I.

Furthermore, we report results on exploring the protocol's parameters (see Section V, Figures 4a-4b). We show that the proper setting of the parameter k is dataset dependent, with appropriate values in the range $k \in \{6, 7, 8\}$ for the explored datasets representing typical viral sequences. See details in Section V.

In addition we report results of our protocol achieving privacy against both client and server by adding a ciphertext sanitization step; see Section VI and Table II. The extended protocol achieves the same accuracy and microAUC as our protocol with no sanitization. The server runtime is larger by roughly $c_1 + 25(c_2 - 1)$ seconds, where $c_1 \approx 35$ is a constant overhead incurred by increasing the scheme's parameter to support a deeper homomorphic computation, and c_2 is a tuneable parameter controlling the number of randomization-then-bootstrapping cycles executed as part of the sanitization algorithm. The analysis in [DS16] shows that $c_2 = 1, 2$ suffices for [BV14]; a similar analysis for CKKS [CKKS17] is beyond the scope of our paper.

B. Our Approach

Secure similarity measuring on encrypted k-mer signatures. We use FHE to privately classify a given NA (i.e., DNA or RNA) sequence, namely, to measure its similarity to the different types included in the dataset of the centralized repository. Similarity is measured by comparing *k-mer signatures* (see Notation II.1).

In more detail, the client FHE-encrypts the k-mer signature of its NA sequence and sends the resulting ciphertext to the server. The server first computes from its dataset a representative k-mer signature for each viral strain. Next, the server homomorphically compares each representative against the client's (encrypted) k-mer signature. Similarity is measured using Jaccard similarity (i.e., the number of k-mers appearing

[4]The public-key (95Mb) is reused in repeated queries between a client and a server with an ongoing commercial relationship, and hence not accounted for when measuring the per-sequence communication.

in both the representative and the sequence k-mers signatures, normalized by the total number of k-mers appearing in either, see Definition II.2). Then, the server homomorphically normalizes the similarity scores (i.e., the estimates of the likelihood that the sequence belongs to each of the types) by dividing each score by the sum of scores. Lastly, the server sends the vector of encrypted normalized scores to the client. We note that measuring similarity through k-mer signatures has been studied in several contexts related to molecular biology and genomics, as described in Section VII below.

Next, we highlight further key techniques we introduced to improve the accuracy and runtime of our protocol.

Fast inversion under FHE. Our protocol homomorphically computes the inverse of an encrypted value in two occurrences: (1) when computing the Jaccard similarity; and (2) when normalizing the scores. Computing the inverse of an encrypted value is typically an expensive operation when performed under FHE. A natural solution is to employ the low-degree polynomial approximation of the inverse function for values in $[0.5, 1.5]$ from [CKKS17]. However, using the approximation of [CKKS17] introduces two challenges. First, the values which our protocol needs to invert do not lie in the range $[0.5, 1.5]$. Second, the approximation error in [CKKS17] grows proportionally to the distance from 1 of the value to be inverted. To resolve both of these issues, before inversion we first apply a monotone linear transformation to the values, bringing them sufficiently close to 1, and in particular into the allowed range $[0.5, 1.5]$. Crucially, our transformation does not harm the accuracy and microAUC scores (because it is monotone). See Section IV-C for details.

Doubling the packing capacity. Packing multiple plaintext values in a single ciphertext, and employing SIMD (single instruction multiple data) computing, is a well-known technique for optimizing amortized complexity in homomorphic computations. In this work, we show how to effectively *double* the number of plaintext slots, packing twice the number of plaintext values in each ciphertext compared to the number of allocated plaintext slots. Our technique can be used with any FHE scheme that supports encryption of complex numbers (e.g., the scheme of [CKKS17]), and leverages the fact that each slot can hold a complex value, whereas our data values are real numbers. Specifically, we pack two real values in each complex number slot, one in the real part of the complex number, and the other in its imaginary part. We support homomorphic evaluation of the inner-product of two vectors of ciphertexts which are packed using our enhanced packing.

We note that an independent work proposed similar packing [HPC+22]. A prior work [BCCW19] proposed another approach for packing two reals in each complex number slot; however their technique supports only linear operations over encrypted data (i.e. multiplication by a scalar and homomorphic addition), whereas our packing supports also ciphertext-to-ciphertext operations, as needed for our protocol.

II. Preliminaries

Notation. We use $\mathsf{negl}(\kappa)$ to denote a function that is negligible in κ. We use the standard notion of computational indistinguishability (e.g., from [Gol04]) against non-uniform distinguishers, denoted by $\approx$; namely if R, R' are random variables then $\mathsf{R} \approx \mathsf{R}'$ denotes that they are computationally indistinguishable. We use PPT as shorthand for Probabilistic Polynomial Time. For a complex number x, we use $\bar{x}$ to denote its complex conjugate, i.e., if $x = a + ib$ then $\bar{x} = a - ib$. The *characteristic vector* of a set S over an (ordered) universe $U = \{u_1, \ldots, u_n\}$ is the length-$|U|$ vector whose ith enrty is 1 if $u_i \in S$, and 0 otherwise. Throughout this paper we abuse notation in identifying sets with their characteristic vectors and using sets vs. vector notations interchangeably. For an arithmetic circuit C, we denote by $\mathsf{depth}(C)$ its multiplicative depth, i.e., the maximal number of multiplication gates on a path from inputs to output.

k-mer signatures. In bioinformatics, k-mers are length-k substrings contained within a genome sequence. The (binary) *k-mer signature* of a genomic (or nucleic acid - NA) sequence is the set of all length k-strings over the alphabet of nucleotides, $\Sigma = \{A, C, G, T\}$, that actually appear in the sequence. For example, the 2-mers signature of the sequence $GCAT$ is $\{GC, CA, AT\}$. We represent k-mer signatures by their characteristic vectors, that is, by the length 4^k vector whose entries correspond to all length-k sequences over Σ in lexicographic order, and where an entry has value 1 if and only if its corresponding k-mer occurs in the sequence. For a sequence $s \in \Sigma^*$ we denote the k-mer signature of s by $\rho^{(k)}(s)$. Formally:

Notation II.1. *For all $j \in \{1, \ldots, 4^k\}$, let σ_j be the jth length-k sequence over Σ (in lexicographic order).*
The k-mer signature of s is

$$\rho^{(k)}(s)_j = \begin{cases} 1, & \text{if } \sigma_j \text{ appears in } s \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

We will omit k when it is clear from the context.

Definition II.2 (Jaccard Similarity)**.** *The* Jaccard similarity *of two binary vectors u, v of dimension 4^k is defined as:*

$$Jaccard(u, v) = \frac{|u \cap v|}{|u \cup v|}$$

where union and intersection are computed as entry-wise OR and AND respectively.[5]

microAUC. We evaluated our protocol using *micro*AUC as implemented in sklearn [auc]. MicroAUC computes the area under the ROC curve over all the labels using *micro*Average. In more detail, ROC is the curve created by plotting the true positive rate (TPR) against the false positive rate (FPR) at

[5]We note that this quantity can be calculated efficiently using inner products, as described in Section IV-B.

various threshold settings, where TPR and FPR are defined as follows (for c classes):

$$TPR = \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FN_c} \quad FPR = \frac{\sum_c FP_c}{\sum_c FP_c + \sum_c TN_c}$$

where TP_c, FP_c, TN_c, FN_c stand for true positive, false positive, true negative and false negative, respectively.

Fully-Homomorphic Encryption (FHE) is a public-key encryption scheme $\mathsf{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ that allows one to compute "under the hood" of the encryption (without the secret decryption key). In this paper we use a generalized *approximate* notion of FHE, which allows for the decryption of a ciphertext to differ from the underlying plaintext, as long as these values are sufficiently close. This captures standard (exact) FHE schemes as a special case, but allows us to also employ approximate schemes such as [CKKS17]. For concreteness, we take the message space to be the complex numbers ($\mathbb{C}$), as supported by [CKKS17].

Intuitively, an FHE scheme has the following three properties: (1) Approximate decryption correctness (allowing for a small decryption error as explained above). (2) Approximate FHE correctness, in the sense that the ciphertext obtained by evaluating a circuit C (using Eval) on a set of ciphertexts decrypts to (a value which is close to) the value that would have been obtained by evaluating C directly on the underlying messages. (3) Computational indistinguishability, namely for every $\mathsf{msg} \in \mathbb{C}$, the joint distribution of the public key pk (randomly generated by Gen) and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$ is computationally indistinguishable from the joint distribution of pk and $\mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$. This is formalized in the full version [Ano22] as an *ϵ-approximate FHE* scheme (where a 0-approximate FHE scheme is simply standard exact FHE). In particular, in this work we use the approximate FHE scheme of [CKKS17].

Sub-circuits for homomorphic computations used in our privacy-preserving protocol.

- *Multiplication:* We denote by Mul the circuit that on input a pair of messages $\mathsf{msg}_1, \mathsf{msg}_2$ outputs their product $\mathsf{msg}_1 \cdot \mathsf{msg}_2$.
- *Addition:* For $k \in \mathbb{N}$, we denote by Add_k the circuit that on input k messages $\mathsf{msg}_1, \ldots, \mathsf{msg}_k$ outputs their sum $\mathsf{msg}_1 + \ldots + \mathsf{msg}_k$.
- *Inverse:* We denote by $\mathsf{inverse}$ the circuit that on input a message msg outputs its inverse $1/\mathsf{msg}$. (See a note below on how this function is homomorphically implemented in [CKKS17].)

A note on computing the inverse over CKKS [CKKS17]: To compute inverses, we use a corollary derived from [CKK+19, Lemma 1]:

Corollary II.3. *For every positive integer r and reals $\alpha \in [0,1)$ and $x \in [1-\alpha, 1+\alpha]$: $\frac{1}{x}(1-\alpha^{2^r}) \leq x^{\mathsf{inv}} \leq \frac{1}{x}$.*

III. Problem statement

Overview. We consider a setting in which a *client* Cl interacts with a *server* S. S holds a private dataset D consisting of m labeled examples for a viral strain-classification algorithm with s possible types,[6] and Cl holds an input x consisting of n sequences that Cl wishes to classify.

Our goal is to allow Cl to accurately classify its input sequences by interacting with S, while guaranteeing privacy in the sense that S learns nothing about the input sequences of Cl. We additionally consider an enhanced model which guarantees that Cl learns nothing on the dataset of S (except what can be inferred from her input and output, and the public parameters). The threat model we address is standard 2-party computational security in the passive setting, with the stronger guarantee of privacy against an *active* server.

In terms of efficiency, we aim to reduce the computational and communication complexity of the parties. Specifically, we aim for single-round protocols, and strive to reduce the number of communicated bits, as well as the complexity of the local computations performed by the parties. Looking ahead, our protocols will rely on *Fully Homomorphic Encryption (FHE)* which allows the server to run the classification algorithm on an encryption of x as if it were not encrypted.

The formal details follow.

Viral strain classification functionality. The viral strain classification functionality involves a *client* Cl whose input x consists of n viral sequences, and a *server* S whose input is a private dataset D consisting of m labeled examples for a viral strain-classification algorithm. The public parameters, denoted params, consist of the number s of strain classes (types), the number n of sequences, the number of precision digits ℓ so that real numbers are normalized to [-1,1] with ℓ digits of precision, as well as any public hyper-parameters of the learning algorithm.[7] The client Cl obtains the classification result as output, whereas the server has no output. Concretely, the classification result consists of a list of *scores* $(\mathsf{sc}_1^i, \ldots, \mathsf{sc}_\mathsf{s}^i)$, for each sequence $i \in [n]$, s.t. $\sum_{\sigma=1}^{\mathsf{s}} \mathsf{sc}_\sigma^i = 1$. The latter allows interpreting each score sc_σ^i as the likelihood that the ith sequence is in class σ. The predicted class can then be determined to be the one with the highest score, i.e., $\arg\max_\sigma \mathsf{sc}_\sigma^i$. We denote this functionality by C_{params} (where C stands for *classification*), and depict it in Figure 1.

Threat model. Our goal is to guarantee correctness of the output in the presence of a passive (so-called "semi-honest") computationally-bounded server S, as well as privacy of Cl's input against an *active* (so-called "malicious") server who may arbitrarily deviate from the protocol to mount any PPT attack. Namely, for correctness we assume that even a corrupted S follows the protocol and is restricted to performing PPT computations (but S tries to infer as much information as

[6] More generally, we note that it suffices for S to hold *the classification model* derived from its dataset, which can significantly improve the server's storage complexity.

[7] In our k-mer based protocol, the hyper-parameters are the k-mer length k, and a threshold $\tau \in (0,1)$.

Parties: A client Cl and a server S.

Public Parameters: the number s of strain classes (types) and n of test sequences to be classified; the precision ℓ so that all values in $\mathbb{R}$ are normalized to $[-1, 1]$ with a precision of ℓ digits; and any public hyper-parameters of the learning algorithm.

Inputs: for S, a dataset D of m labelled viral strains; for Cl, a list x of n sequences to be classified into one of s types.

Output: the client obtains, for every $i \in [n]$, a score $\mathsf{sc}^i = (\mathsf{sc}^i_1, \ldots, \mathsf{sc}^i_{\mathsf{s}})$ s.t. $\sum_{j=1}^{\mathsf{s}} \mathsf{sc}^i_j = 1$; the server has no output.

Fig. 1: Viral Strain Classification Functionality.

possible from its view of the interaction), whereas for privacy the only restriction on the server is that it is PPT. We will also consider extensions that guarantee privacy of S's input against a passive Cl.

Terminology. Let Π be a 2-party protocol executed between the PPT client Cl with input x (a list of n viral strains to be classified into s classes) and the PPT server S with input D (a dataset of m labeled viral strains). For public parameters params (as specified in Figures 1 and 3), we use $\Pi(x, D, \mathsf{params})$ to denote the random variable describing the *output* of a random execution of Π with public parameters params when Cl, S have inputs x, D respectively (where the probability is over the randomness of both parties). The *view* of a party – which consists of the public parameters, its input and randomness, and the messages it received during the protocol execution on inputs x, D with public parameters params – is denoted by $\mathsf{View}^{\Pi}_{\mathsf{Cl}}(x, D, \mathsf{params})$ and $\mathsf{View}^{\Pi}_{\mathsf{S}}(x, D, \mathsf{params})$ for Cl and S, respectively. In an execution of the protocol with an *active* server S^* (who, in particular, may deviate from the protocol), its view is denoted by $\mathsf{View}^{\Pi}_{\mathsf{S}^*}(x, D, \mathsf{params})$.

Security notion. We consider standard 2-party computational security in the passive setting when strengthened to guarantee privacy against an *active* server. Formally, we adapt (and simplify) [HL10, Def. 4.6.1] to the problem of viral strain classification, as follows.

Definition III.1 (ϵ-Private Viral Strain Classification)**.** *We say that a 2-party protocol Π is an ϵ-*private viral strain classification protocol *if the following holds:*

1) $(1-\epsilon)$**-Correctness.** *For every x, D and params (as specified above),*

$$\max_{1 \le \sigma \le \mathsf{s}, 1 \le i \le n} \left| (\Pi(x, D, \mathsf{params}))^i_\sigma - (C_{\mathsf{params}}(x, D))^i_\sigma \right| \le \epsilon$$

where $\Pi(x, D, \mathsf{params})^i_\sigma$ and $(C_{\mathsf{params}}(x, D))^i_\sigma$ denote the σth score assigned to the ith strain in x by Π and C_{params} respectively, and where the probability is over the randomness of Cl and S.

2) ***Privacy (against an active server).*** *For every non-uniform deterministic polynomial time server S^*, there exists a (non-uniform) polynomial-time simulator $\mathsf{Sim}_{\mathsf{S}^*}$ such that for every x, D and params,*

$$\mathsf{View}^{\Pi}_{\mathsf{S}^*}(x, D, \mathsf{params}) \approx \mathsf{Sim}_{\mathsf{S}^*}(D, \mathsf{params}).$$

We extend Definition III.1 to also guarantee *privacy against the client*.

Definition III.2 (ϵ-Strongly Private Viral Strain Classification)**.** *We say that a 2-party protocol Π is an ϵ-*strongly private viral strain classification protocol *if it is an ϵ-private viral strain classification protocol (as in Definition III.1), and additionally satisfies the following privacy property against a* passive *client:*

- ***Privacy (against a passive client).*** *There exists a PPT simulator $\mathsf{Sim}_{\mathsf{Cl}}$ such that for every x, D and params,*

$$\mathsf{View}^{\Pi}_{\mathsf{Cl}}(x, D, \mathsf{params}) \approx \mathsf{Sim}_{\mathsf{Cl}}(x, \mathsf{params}).$$

Discussion. A few remarks are in order. First, as in [HL10], privacy against the server is defined with respect to *deterministic* servers S^*. This does not limit the generality because S^* is allowed to be non-uniform (so it can have the "best" choice of random coins hard-wired into it).

Second, we note that although the privacy against the server in [HL10] is indistinguishability-based, it is equivalent to our simulation-based definition (Definition III.1). Indeed, our definition implies that of [HL10] through a simple union bound, whereas the indistinguishability-based definition implies simulation in this case because it guarantees that the simulator can emulate the client's operations honestly using the all-0 string as the client's input.

Finally, we note that the *passive versus active* distinction in defining correctness versus privacy (respectively) arises from the underlying business model: violating correctness puts the server at a greater risk of reputation damage and loss of clientele, because inaccurate predictions are more easily detectable by the client than a breach of privacy. So the server has a higher business incentive to follow the protocol and preserve correctness than to protect privacy.

IV. Algorithms and Protocols

A. Viral Strain Classification Algorithm

In this section we describe our *cleartext* strain classification algorithm (Figure 2). We stress that this algorithm does not provide any privacy guarantees (see Section IV-B for our privacy-preserving protocol).

The algorithm includes two phases: First, a pre-processing phase that, given a training set D of labeled sequences with labels in $\{1, \ldots, \mathsf{s}\}$ representing classes, outputs s class-representatives denoted $\rho^1, \ldots, \rho^\sigma$ (the trained model). Second, a classification phase that, given an unlabeled query sequence v to be classified as belonging to one of the s classes, outputs a vector of s scores $\mathsf{sc} = (J_1(v), \ldots, J_{\mathsf{s}}(v))$ where each score $J_\sigma(v)$ is interpreted as the likelihood that the sequence v belong to the σth class. The hyper-parameters

Plaintext Classification Algorithm

Parameters: number of classes s and of k-mers $K = 4^k$, and a threshold $\tau \in (0, 1)$.

Input: A training set D of m labeled sequences for training a model during pre-processing, and an unlabeled query sequence v for classification.

Output: A score vector $\mathsf{sc} \in [0, 1]^{\mathsf{s}}$ assigning a likelihood score for each viral class.

Pre-processing.

1) For each $u \in D$, compute the k-mer signature $\rho(u)$ of u (as defined in Section II)
2) For each $\sigma \in [\mathsf{s}]$, set its class representative to be the k-mer signature ρ^σ defined to be $\rho^\sigma_j = 1$ (for $j = 1, \ldots, 4^k$) if and only if

$$|\{u \in D_\sigma : \rho(u)_j = 1\}| \geq \tau |D_\sigma|$$

where $D_\sigma \subseteq D$ is the set of sequences labeled by σ.

Classification.

1) Compute the k-mer signature $\rho(v)$ of v (as defined in Section II)
2) Compute $J'_\sigma(v) = Jaccard(\rho(v), \rho^\sigma)$ for all $\sigma \in [\mathsf{s}]$
3) Compute $J_\sigma(v) = \frac{J'_\sigma(v)}{\sum_{\sigma'} J'_{\sigma'}(v)}$ for all $\sigma \in [\mathsf{s}]$
4) Output $\mathsf{sc} = (J_1(v), \ldots, J_{\mathsf{s}}(v))$

Fig. 2: Viral strain classification – the cleartext version

are k (or equivalently, the number of k-mers $K = 4^k$) and a threshold $\tau \in (0, 1)$.

The pre-processing proceeds as follows. First, transform each sequence in D to its k-mer signature (Pre-processing, Step 1). Second, for every label $\sigma = 1, \ldots, \mathsf{s}$, compute a representative k-mer signature ρ^σ for the sequences with label σ in D, by setting ρ^σ_j to 1 if the jth k-mer appears in at least a τ-fraction of the sequences with label σ, and otherwise set $\rho^\sigma_j = 0$ (Pre-processing, Step 2).

The classification phase proceeds as follows. First, transform v into its k-mer signature, denoted $\rho(v)$ (Classification, Step 1). For every label $\sigma = 1, \ldots, \mathsf{s}$, compute the Jaccard similarity of the k-mer signatures $\rho(v)$ and ρ^σ (Classification, Step 2). Finally, normalize the scores and output these normalized scores (see Steps 3-4). The predicted label is the one with the highest score. We remark that the normalization in Step 3 is only needed if we want to support a unified classification threshold which, in particular, is the case when using microAUC to evaluate the correctness of the algorithm.

B. Privacy Preserving Protocol for Viral Strain Classification

Our privacy-preserving viral strain classification protocol (Figure 3) securely realizes the viral strain classification algorithm specified in Figure 2.

In the following we first give the high-level overview of our protocol, and then elaborate on: how we compute set operations (intersection, size, and union) via inner-products; our novel data packing that speedup the inner-product computing; and the concrete specification of the sub-circuits we use for computing the set operations while employing our packing.

1) Protocol Overview: Our protocol securely realizes the cleartext classification algorithm of Figure 2. In the protocol, the client encrypts the k-mer signatures of its test sequences and sends them to the server, and the server *homomorphically* computes *over encrypted data* the Jaccard similarity scores (Figure 2, Classification, Steps 2) and the normalization (Figure 2, Classification, Steps 3). The server then sends the resulting ciphertexts to the client. Extending the protocol to ensure privacy against the client is performed by simply having the server sanitize these ciphertexts prior to sending them to the client; see details in Section VI.

Elaborating on the above, the Jaccard similarity score with each strain representative ρ^σ is homomorphically computed in a SIMD manner by first homomorphically computing a vector I_σ (respectively, U_σ) that contains, in its γth slot, the intersection (respectively, union) of ρ^σ with the γth test virus (see Figure 3, Step 4a). Next, the server uses the FHE's homomorphic inversion algorithm $\mathsf{inverse}$ (see "Sub-circuits for homomorphic computations" in Section II) to compute U_σ^{-1}, and then homomorphically multiples I_σ with U_σ^{-1} in a SIMD manner. The resultant vector J'_σ contains, in its γth slot, the Jaccard similarity of ρ^σ and the γth test sequence. Then, the Jaccard similarity scores are normalized by first homomorphically summing all Jaccard scores J'_σ, then homomorphically computing the inverse of the sum, and finally homomorphically multiplying each Jaccard score with the inverse of the sum (see Figure 3, Step 4b).

2) Computing Set Operations and Inner Products:

Similarity scores through set operations. Recall that computing Jaccard similarity scores – which lies at the heart of our protocol – involves computing intersections and unions of subsets (in our case, the subsets are of k-mers that appear in genomic sequences). These operations are expressed in our protocol as inner products of binary vectors (the indicator vectors of the sets), as we now describe.

Reducing set operations to inner-products. Let d be a positive integer and $S \subseteq \{1, \ldots, 2d\}$. The *indicator vector* of S is a vector $\mathsf{IndVec}(S) \in \{0, 1\}^{2d}$ such that $\mathsf{IndVec}(i) = 1$ iff $i \in S$. In addition, for an indicator vector χ, its *indicated set*, $\mathsf{IndSet}(\chi)$ is a set such that $\chi = \mathsf{IndVec}(\mathsf{IndSet}(\chi))$. Then for any two sets $A, B \subseteq \{1, \ldots, 2d\}$:

- $|A| = \langle \mathsf{IndVec}(A), (1, \ldots, 1) \rangle$.
- $|A \cap B| = \langle \mathsf{IndVec}(A), \mathsf{IndVec}(B) \rangle$.
- $|A \cup B| = |A| + |B| - |A \cap B|$.

Efficiently computing inner products using complex numbers. Recall that our protocols operate over reals, whereas the underlying FHE scheme [CKKS17] supports messages that are complex numbers. The naive way to compute inner products in such a scheme is to interpret each real-valued message entry as a complex number with imaginary part 0. However, this "wastes" half of the positions in the complex-number message, leading to increased storage and running time. To prevent this increase, we propose a novel encoding that utilizes the imaginary part.

3) Our Novel Encoding.: We show how to efficiently reduce inner products of vectors in $\mathbb{R}^{2d}$ to operations on vectors in $\mathbb{C}^d$, allowing us to exploit the imaginary part of messages

Privacy-Preserving Classification Protocol

Parameters: Both server and client known the number of strain classes s, k-mers $K = 4^k$ and test sequences n, and an FHE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ and its public parameters params. The server also know a threshold $\tau \in (0, 1)$.
Input: The server holds a training dataset D of m labeled sequences; the client holds n sequences to classify.
Steps:

1) **Training (Server).** The server S performs the following one-time pre-processing phase, over the cleartext D. (This step is performed only once for each dataset D, and can be done offline before the test sequences are known.)
   - Computes representative k-mers signatures $\rho^1, \ldots, \rho^{\mathsf{s}}$ of the strain classes, as specified in Step 2 of Figure 2.
   - Encodes and packs each representative ρ^σ according to the server-encoding and server-packing described in Sections IV-B3 and IV-B4. Specifically, if $b_{l,\sigma}$ is a Binary indicator such that $b_{l,\sigma} = 1$ iff the lth k-mer appears in the representative of the σth strain, then ρ^σ is encoded as $K/2$-many vectors $\mathsf{msg}_1^{\rho^\sigma}, \ldots, \mathsf{msg}_{K/2}^{\rho^\sigma}$, where $\mathsf{msg}_l^{\rho^\sigma} = ((b_{2l-1,\sigma} - i \cdot b_{2l,\sigma}), (b_{2l-1,\sigma} - i \cdot b_{2l,\sigma}), \ldots) \in \mathbb{C}^j$.
2) **Generating keys (Client).** The client generates encryption keys (this step is performed once per client) as follows:
   a) Generates a public key and secret key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa, \mathsf{params})$.
   b) Send pk to the server.
3) **Preprocessing, encoding and encrypting (Client).** The client:
   a) Computes the k-mers signature for each of its test sequences, as described in Step 1 of Figure 2. We denote by $b_{l,j}$ the Binary indicator such that $b_{l,j} = 1$ iff the lth k-mer appears in the jth test sequence.
   b) Encodes and packs the n length-K k-mer signatures using the client-encoding and client-packing (as described in Sections IV-B3 and IV-B4) to get messages $m_1, \ldots m_{K/2}$, where $m_l = (b_{2l-1,1} + i \cdot b_{2l,1}, b_{2l-1,2} + i \cdot b_{2l,2}, \ldots)$. (That is, for example, m_1 contains the indicators of the first and second k-mers over all test sequences.)
   c) Encrypts the messages $c_l \leftarrow \mathsf{Enc}(\mathsf{pk}, m_l)$, for $l = 1, \ldots, K/2$.
   d) Sends $c_1, \ldots, c_{K/2}$ to the server.
4) **Classification (Server).** The server classifies each of the client's test sequences as follows:
   a) Computes the Jaccard similarity scores (see Step 2 of Figure 2, and Definition II.2) between the test sequences and the strain representatives, where the Jaccard similarity J_σ for $\sigma = 1, \ldots, \mathsf{s}$ is computed as follows:
      i) $I_\sigma \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{setIntersect}_{\rho^\sigma}, c_1, \ldots, c_{K/2})$.
      ii) $U_\sigma \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{union}_{\rho^\sigma}, c_1, \ldots, c_{K/2})$.
      iii) $U_\sigma^{\mathsf{inv}} \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{inverse}, U_\sigma)$.
      iv) $J'_\sigma \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{Mul}, I_\sigma, U_\sigma^{\mathsf{inv}})$.
   b) Normalizes the scores as follows:
      i) $\mathsf{sum} \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{Add}_{\mathsf{s}}, J'_1, \ldots, J'_{\mathsf{s}})$.
      ii) $\mathsf{sum}^{\mathsf{inv}} \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{inverse}, \mathsf{sum})$.
      iii) $J_\sigma \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{Mul}, J'_\sigma, \mathsf{sum}^{\mathsf{inv}})$, for each $\sigma = 1, \ldots, \mathsf{s}$.
   c) Sends $J_1, \ldots, J_{\mathsf{s}}$ to the client.

Fig. 3: Privacy-preserving classification protocol

in FHE schemes such as [CKKS17] that operate over complex number messages. This effectively reduces the message length by a factor of 2, reducing storage and runtimes.

The Encoding. Our novel encoding $M \colon \mathbb{R}^{2d} \mapsto \mathbb{C}^d$, given a real-valued vector $v = (v_1, \ldots, v_{2d}) \in \mathbb{R}^{2d}$, outputs:

$$M(v) = (v_1 + iv_2, \ldots, v_{2d-1} + iv_{2d}) \in \mathbb{C}^d.$$

(Here, $i = \sqrt{-1}$.) It is easy to see that for all $v, u \in \mathbb{R}^{2d}$,

$$\langle v, u \rangle = (t + \bar{t})/2, \text{ where } t = \langle M(v), \overline{M(u)} \rangle.$$

Particularly, when performing homomorphic operations using [CKKS17], this encoding allows us to reduce the number of ciphertexts by half, at the cost of performing 2 additional conjugate operations and one additional division by a constant. For sufficiently large values of d, this is more efficient than performing the inner product using the naive encoding described in the previous section. Looking ahead, by appropriately encoding the server's input, we can further reduce the overhead in our implementation to a single conjugate operation.

Client's Encoding. The encoding M described above is used to encode the client's data in our protocol.

Server's Encoding. For the server's data we use a slightly modified encoding, which allows us to compute the inner products (between the client's test viruses and the server's strain representatives) more accurately and efficiently. Roughly, the server's input is first encoded using the encoding M described above, then each entry is conjugated and divided by 2. Specifically, a vector $\rho = (\rho_1, \rho_2, \ldots, \rho_{K-1}, \rho_K) \in \{0, 1\}^K$ is encoded into $(\mathsf{msg}_1^\rho, \ldots, \mathsf{msg}_{K/2}^\rho) \in \mathbb{C}^{K/2}$ where $\mathsf{msg}_\ell^\rho = (\frac{\rho_{2\ell-1}}{2} - i\frac{\rho_{2\ell}}{2})$.

Advantages. There are several advantages to using this modified encoding. First, it reduces the overhead needed to compute the inner product, specifically, saving one conjugate operation and the division by 2. Additionally, performing the division on the server's data (which is used in the computation as plaintexts) instead of on the client's encrypted data has the added benefit of increasing accuracy. Indeed, in [CKKS17] there is an asymmetry between plaintexts and ciphertexts because an error is added to ciphertexts. In our protocol, the

ciphertext (and consequently also the error) is multiplied by the plaintext, so decreasing the value of the plaintext improves the accuracy of the resultant product.

4) Data Packing: In this section we discuss our input packing, which allows us to utilize ciphertext slots to classify multiple viruses in parallel. In addition to using different encodings of the client and server data, we also use different packing methods. This asymmetry arises from the different roles of the parties' inputs, specifically, *each* of the client's test viruses should be compared with *all* of the server's strain representatives. We stress that the data encoding described above is orthogonal to how data is packed in slots.

Client's Packing. Suppose the number s of plaintext slots available in each ciphertext equals the number n of viruses in the test set. To encrypt n length-$K/2$ vectors, the client generates $K/2$ ciphertexts (each containing s slots), where the jth slot in the ith ciphertext contains the ith entry of the jth vector. (If $s < n$, the client partition the viruses into subsets of size at most s, and apply the above on each subset. Conversely, if $s > n$, the client packs $\lfloor s/n \rfloor$ k-mers for every virus in each ciphertext.)

Server's Packing. When $s = n$, for each encoded strain representative (which is of length $K/2$), the server computes $K/2$ ciphertexts, where the ith ciphertext holds n copies of the ith entry of the representative. (If $s < n$, the server computes, for each of the $K/2$ entries in his encoded representative, n/s ciphertexts with s copies of that entry. Conversely, if $s > n$, then since the client packs $\lfloor s/n \rfloor$ encoded k-mers for each virus in each ciphertext, the server packs in each ciphertext n copies of each of the corresponding s/n encoded k-mers.)

Further optimization. We note that when each ciphertext on the client includes a single entry (i.e., a single k-mer) from each test vector, then the ciphertext obtained through server-packing encrypts a constant vector (with the same value in all entries). In this case, our protocols are optimized to multiply with a constant (instead of with a constant vector). However, allowing this more general packing on the server allows us to handle a wider range of parameter settings, in which – as described below – ciphertexts may pack several k-mers from each plaintext.

The combination of these client- and server-packing is more amenable to computing inner products than the more direct packing in which all entries of a single vector of plaintext values are packed into a single ciphertext. Indeed, computing inner products over vectors packed in this direct method requires a large number of rotations to sum over the different slots, whereas no rotations are needed when using our packing methods. We remark that when $s > n$ so that several k-mers are packed in each ciphertext then computing the inner product does involve some rotations, albeit still less than when using the direct packing.

5) Sub-Circuits for Set Operations: The protocol uses the sub-circuits specified below for computing set intersection, size and union operations. In the following, server-encoding and server-packing (respective, client-) refer to the aforementioned Server's Encoding and Packing (respectively, Client's).

Set intersection: Let $\rho \in \{0,1\}^K$ be an indicator vector which is server-encoded and server-packed into $\mathsf{msg}^{\rho}_1, \ldots, \mathsf{msg}^{\rho}_{K/2}$, namely, $\mathsf{msg}^{\rho}_l = (\rho_{2l-1} - i \cdot \rho_{2l}, \rho_{2l-1} - i \cdot \rho_{2l}, \ldots)$. Let $v^1, \ldots, v^n \in \{0,1\}^K$ be indicator vectors which are client-encoded and client-packed into $\mathsf{msg}^{v}_1, \ldots, \mathsf{msg}^{v}_{K/2}$, where entries $2l-1, 2l$ of v^γ are packed in the γth slot of msg^{v}_l as $v^\gamma_{2l-1} + i \cdot v^\gamma_{2l}$. We define the circuit $\mathsf{setIntersect}_\rho$ that on input $\mathsf{msg}^{v}_1, \ldots, \mathsf{msg}^{v}_{K/2}$ outputs a ciphertext such that the γ-th slot holds $|\mathsf{IndSet}(\rho) \cap \mathsf{IndSet}(v^\gamma)|$. This is computed as $(z + \overline{z})/2$, where

$$z = \sum_{\ell} \mathsf{msg}^{\rho}_{\ell} \cdot \mathsf{msg}^{v}_{\ell} = \sum_{\ell \in \ell_1 \cup \ell_{1-i}} \mathsf{msg}^{v}_{\ell} - i \cdot \sum_{\ell \in \ell_{-i} \cup \ell_{1-i}} \mathsf{msg}^{v}_{\ell},$$

where $\ell_p = \{\ell \mid \mathsf{msg}^{\rho}_{\ell} = p\}$ for $p \in \{1, -i, 1-i\}$. We note that the sub-circuit computing z requires a single multiplication (by the imaginary unit i).

Set size: Let $v^1, \ldots, v^n \in \{0,1\}^K$ be set indicator vectors which are client-encoded and client-packed into $\mathsf{msg}^{v}_1, \ldots, \mathsf{msg}^{v}_{K/2}$. We define the circuit $\mathsf{setSize}$ which on input $\mathsf{msg}^{v}_1, \ldots, \mathsf{msg}^{v}_{K/2}$ outputs a length-n vector whose γth slot holds $|\mathsf{IndSet}(v^\gamma)|$. More specifically, the output is $(t + \overline{t})/2$, where

$$t = \sum_{\ell} \mathsf{msg}^{v}_{\ell} - i \cdot \sum_{\ell} \mathsf{msg}^{v}_{\ell}.$$

(This is because set size can be computed as the intersection with the set of all k-mers, represented by the vector $(1-i)^{K/2}$.) We note that the sub-circuit computing t requires a single multiplication (by the imaginary unit i).

Set union: Let $\rho \in \{0,1\}^K$ be an indicator vector which is server-encoded and server-packed into $\mathsf{msg}^{\rho}_1, \ldots, \mathsf{msg}^{\rho}_{K/2}$. Also, let $v^1, \ldots, v^n \in \{0,1\}^K$ be indicator vectors client-encoded and client-packed into $\mathsf{msg}^{v}_1, \ldots, \mathsf{msg}^{v}_{K/2}$. We define the circuit union_ρ that on input $\mathsf{msg}^{v}_1, \ldots, \mathsf{msg}^{v}_{K/2}$ outputs a ciphertext such that the γ-th slot holds $|\mathsf{IndSet}(R) \cup \mathsf{IndSet}(v_\gamma)|$. This is computed by first computing $\mathsf{setSize}(\mathsf{msg}^{v}_1, \ldots, \mathsf{msg}^{v}_{K/2}) - \mathsf{setIntersect}_\rho(\mathsf{msg}^{v}_1, \ldots, \mathsf{msg}^{v}_{K/2})$, and then adding $|\mathsf{IndSet}(\rho)|$ to each slot, where $|\mathsf{IndSet}(\rho)|$ is computed in plaintext.

C. Analysis of Our Protocol

We now prove that our protocol is secure.

Theorem IV.1. *Let* $\mathsf{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be an* ϵ*-approximate FHE scheme. Then the protocol of Figure 3, when instantiated with* E *as the underlying FHE scheme, is a private viral strand classification protocol (as in Definition III.1).*

The full correctness analysis – including a precise bound on the approximation error of our protocol – is discussed below. Extensions obtaining privacy *against the client* are described in Section VI.

Proof of Theorem IV.1. **Correctness** is analyzed in Section IV-C1.

Privacy (against the server). Intuitively, privacy against the server follows from the secrecy of the FHE scheme. This holds against an *active* server because the protocol consists of a single message from Cl to S, followed by a single message from S to Cl. Therefore, the (possibly malicious) operations the server performs in the protocol do not affect its view. We proceed to describe the formal proof.

Let S^* be a (possibly actively corrupted, possibly non-uniform) server in the protocol of Figure 3. We describe a PPT simulator $\mathsf{Sim}_{\mathsf{S}^*}$ which simulates the view $\mathsf{View}^{\Pi}_{\mathsf{S}^*}$ of S^*. $\mathsf{Sim}_{\mathsf{S}^*}$ is given the input D of S^*, as well as N and the public parameters params (which include m, ℓ, κ, and the length n of the client's input), and operates as follows:

- Generates encryption keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa, N, \mathsf{params})$, and samples random coins R for the server.
- Let $v^1, \ldots, v^{K/2}$ denote n length-$K/2$ vectors where $v_l = (0 + i \cdot 0)^n$ for every $1 \leq l \leq K/2$. Then $\mathsf{Sim}_{\mathsf{S}^*}$ randomly encrypts $v^1, \ldots, v^{K/2}$ by computing $c^l \leftarrow \mathsf{Enc}(\mathsf{pk}, v^l)$ for $l = 1, \ldots, K/2$.
- Outputs $D, N, \mathsf{params}, \mathsf{pk}, R$ and $c^1, \ldots, c^{K/2}$ (that is, $c^1, \ldots, c^{K/2}$ simulate the encrypted test viruses which the client sent the server).

We claim that the simulated and real views are computationally close. Indeed, the inputs and public parameters are identical in both views, and the random coins R and the public key pk are distributed identically in the real and simulated views. Therefore, it remains to prove that conditioned on these values, the simulated test sequences $c^l, \ldots, c^{K/N}$ are computationally indistinguishable from the real-world ciphertexts which the client sent the server. As we now explain, this follows from the computational security of E using a standard hybrid argument in which we move from the output of $\mathsf{Sim}_{\mathsf{S}^*}$ to $\mathsf{View}^{\Pi}_{\mathsf{S}^*}$ by replacing each of the $K/2 = \mathsf{poly}(\kappa)$ ciphertexts from real to simulated.

Specifically, let $v^{1,R}, \ldots, v^{K/2,R}$ denote the vectors which the client generates in the real protocol execution. We define hybrids $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_{K/2}$, where in $\mathcal{H}_l, 0 \leq l \leq K/2$ the first l ciphertexts $c_1, \ldots, c_l$ are sampled as fresh encodings of $v^{1,R}, \ldots, v^{l,R}$, and the rest of the ciphertexts are sampled as fresh encodings of the all-zero vector. Then $\mathcal{H}_0$ is the output distribution of the simulator, whereas $\mathcal{H}_{K/2}$ is distributed identically to $\mathsf{View}^{\Pi}_{\mathsf{S}^*}$. Since the are $K/2 = \mathsf{poly}(\kappa)$ pairs of hybrids, it suffices to show that $\mathcal{H}_{l-1} \approx \mathcal{H}_l$ for every $1 \leq l \leq K/N$. Indeed, assume towards negation that $\mathcal{H}_{l-1}, \mathcal{H}_l$ are not computationally close, and let $\mathcal{D}$ be a distinguisher between $\mathcal{H}_{l-1}, \mathcal{H}_l$. We describe a (non-uniform) distinguisher $\mathcal{D}'$ between encryptions of $v^{l,R}$ and v^l. $\mathcal{D}'$ has $v^{1,R}, \ldots, v^{K/2,R}, D, N$ and params hard-wired into it. Given a public key pk and a ciphertext c (encrypting either $v^{R,l}$ or v^l) it generates $c'_z \leftarrow \mathsf{Enc}(\mathsf{pk}, v^{z,R})$ for every $1 \leq z < l$, and $c''_z \leftarrow \mathsf{Enc}(\mathsf{pk}, v^z)$ for every $l < z \leq K/2$. Then, it samples random coins R for the server, runs $\mathcal{D}$ on input $D, N, \mathsf{pk}, \mathsf{params}, R, c'_1, \ldots, c'_{l-1}, c, c''_{l+1}, \ldots, c''_{K/2}$ and outputs whatever $\mathcal{D}$ outputs. Notice that if c encrypts v^l then $\mathcal{D}$ is run with a sample from $\mathcal{H}_{l-1}$, otherwise it is run with a sample from $\mathcal{H}_l$. Therefore, $\mathcal{D}'$ obtains the same (non-negligible) distinguishing advantage as $\mathcal{D}$, which contradicts the security of the FHE scheme. □

1) Correctness and Approximation Error Analysis: Our protocol *approximates* the algorithm from Figure 2, in the sense of "$(1-\epsilon)$-correctness" of Definition III.1 (for an appropriate parameter ϵ as specified below). The approximation error emanates from two sources. First, when we instantiate our protocol with an *approximate* FHE scheme for the underlying encryption scheme (as indeed we do in our empirical evaluation), then the scheme itself introduces an approximation error in decryption. Second, in our protocol we use a *polynomial approximation* to compute the inverse, which introduces an additional approximation error. In this section, we analyze the effect each of these has on the total error incurred by our protocol, and bound the approximation error. We first establish some notations.

Notation IV.2 (Non-private protocol $\tilde{\pi}$.)**.** *We denote by $\tilde{\pi}$ the protocol implementation of the non-private classification algorithm of Figure 2 in Section IV-A. More accurately, $\tilde{\pi}$ denotes: (1) a parallel-repetition version of this algorithm, in which several unlabeled sequences are classified simultaneously; and (2) all computations are performed by the server, except for computing the k-mer signatures $\rho(v)$ of each of the client's sequences v (used in Step 2) which are computed by the client and sent to the server.*

Notation IV.3. *Let π denote the protocol of Figure 3. We denote by π^{CKKS} the protocol obtained when π is instantiated with [CKKS17] as the underlying FHE scheme.*

Definition IV.4. *Let π be a 1-round protocol between a client and server, in which the client sends encrypted data to the server, the server homomorphically computes on it, and sends the encrypted outcome back to the client. The* multiplicative depth $\mathsf{depth}(\pi)$ *of π is the multiplicative depth of the circuit computed by the server on the client's encrypted data, i.e., the largest number of multiplication gates on a path from input to output.*

With these notations in place, we can now bound the error introduced by our protocol (See Corollary IV.7 below for a detailed analysis of the error.)

Theorem IV.5 (Correctness)**.** *Let π^{CKKS} denote the protocol of Figure 3 when instantiated with [CKKS17] as the underlying FHE scheme, and let $\tilde{\pi}$ denote its non-private counterpart specified in Notation IV.2. Let ϵ_{inv} denote the accumulated error induced by approximating the inverse in Steps 4(a)iii and 4(b)ii in Figure 3, and let $\epsilon_{\mathsf{CKKS}} > 0$ denote the error induced by the encryption operation of [CKKS17]. Then there exists a constant ϵ' such that*

$$\max_x |\pi^{\mathsf{CKKS}}(x) - \tilde{\pi}(x)| < \epsilon_{\mathsf{inv}} + \epsilon_{\mathsf{CKKS}} \cdot 2^{\mathsf{depth}(\pi)} + \epsilon' \cdot (2^d - 1)$$

where $\mathsf{depth}(\pi)$ *denotes the multiplicative depth of the protocol* π *of Figure 3 (see Definition IV.4). Moreover,* ϵ' *depends only on the parameters used to instantiate the encryption scheme of [CKKS17], and can be made arbitrarily small by appropriately setting these parameters.*

Roadmap towards proving Theorem IV.5. We briefly highlight the main points in the proof; see the full version [Ano22] for the full proof. Recall that our protocol has two sources of error: (1) error resulting from encryption error in the underlying FHE scheme (which is accumulated over homomorphic computations); and (2) the error caused by the approximate computation of the inverse function in Steps 4(a)iii and 4(b)ii in Figure 3. In the full version [Ano22], we first bound (1) using the analysis of [CKKS17] which roughly shows that for an appropriate choice of the scheme's parameters, each multiplication doubles the error induced by encryption. Consequently, the overall error caused by performing homomorphic computations with an approximate FHE scheme is roughly $\epsilon_{\mathsf{CKKS}} \cdot 2^{\mathsf{depth}(\pi)}$, where ϵ_{CKKS} is the error introduced during encryption, and $\mathsf{depth}(\pi)$ is the multiplicative depth of the protocol. Then, we bound (2) by analyzing the accumulation of error over the different steps of our protocol. More specifically, the analysis isolates the error caused by the approximate computation of the inverse, by assuming the FHE scheme is exact. (see the full version [Ano22] for the full analysis.) We approximate the inverse of a value x using a polynomial approximation, where the quality of the approximation depends on the tail. In particular, the approximation error depends both on the distance of x from 1, namely on the bound α one can give on $|1-x|$, and on the number r of summands which we keep (i.e., the tail begins with summand number $r+1$).[8] The following notation will be useful.

Notation IV.6 (Parameters of the approximation)**.** *Recall that the protocol* π *of Figure 3 computes two approximate inverses: (1) the inverse* U_σ^{inv} *of* U_σ *in Step 4(a)iii, and (2) the inverse* $\mathsf{sum}^{\mathsf{inv}}$ *of* sum *in Step 4(b)ii, where the quality of these approximations depend on the distance of the inputs from 1, and the number of summands used in the polynomial approximation (equivalently, the multiplicative depth of the circuit computing the approximation). We denote these bounds by* α_1, α_2*, namely* $|1-U_\sigma| \leq \alpha_1$*, and* $|1-\mathsf{sum}| \leq \alpha_2$*. Moreover, we denote by* r_1, r_2 *the multiplicative depths of the circuits used to approximate the inverses of* U_σ *and* sum *(respectively).*

With these notations in place, we are now ready to give a more accurate bound on the error of our protocol:

Corollary IV.7. *Let* π *denote the protocol of Figure 3, and let* $\tilde{\pi}, \pi^{\mathsf{CKKS}}, \alpha_1, \alpha_2, r_1, r_2$ *be as in Notation IV.2, Notation IV.3 and Notation IV.6 (respectively). Assume that* $\alpha_2 \geq 1 - \min\{\mathsf{sum}\} + \mathsf{s}\frac{1-\alpha_1^{2^{r_1}}}{1-\alpha_1}$*. Then for any* $\epsilon' > 0$ *a key can be initialized such that,* $\max_x |\pi^{\mathsf{CKKS}}(x) - \tilde{\pi}(x)|$ *is at most* $\frac{2\alpha_2}{1-\alpha_2^2} + \epsilon_{\mathsf{CKKS}} \cdot 2^{r_1+r_2+2} + \epsilon' \cdot (2^{r_1+r_2+2}-1)$ *where* s *is the number of strains in the server's dataset, and* ϵ_{CKKS} *bounds the error introduced during the encryption of the inputs* x *to* π^{CKKS}*.*

[8] We note that r corresponds to the depth of the circuit approximating the inverse, because the polynomial approximation is computed iteratively.

D. Complexity Analysis

We analyze the server complexity in the protocol of Figure 3, specifically its depth, RAM requirement, and the number of multiplications it makes. The client complexity analysis is straightforward; details omitted due to space constraints.

Depth. Steps 4(a)i and 4(a)ii in Figure 3 both involve only a multiplication by the imaginary unit i (see Section IV-B5) which in some implementations of CKKS does not require a multiplication level.

In Step 4(a)iii we apply the inverse sub-circuit (see "Sub-circuits for homomorphic computations" in Section II) whose depth is r_1. We then multiply I_σ and U_σ^{inv}, which adds one additional layer to the multiplicative depth. In Step 4(b)ii we again apply the inverse sub-circuit, this time with depth r_2, and in Step 4(b)iii we normalize the Jaccard scores (by multiplying them with the outcome of Step 4(b)ii) which adds one additional layer to the multiplicative depth. In summary, the depth of the protocol of Figure 3 is $r_1 + r_2 + 2$:

Lemma IV.1. *The multiplicative depth of the protocol of Figure 3 is* $r_1 + r_2 + 2$*.*

RAM requirements. Using our packing the server can work in a pipeline manner, i.e., (1) receive a ciphertext from the client, (2) add its contribution to the computation, and (3) discard it. Specifically, only Steps 4(a)i and 4(a)ii involve computations on ciphertexts directly received from the client. In these steps $2\mathsf{s}$ subsets of ciphertexts are added to compute the inner product with s representatives. Also, the ciphertexts are summed to compute $\mathsf{setSize}$ for the client's test virus. In the following steps the protocol uses $O(1)$ ciphertexts for each strand. We therefore get:

Lemma IV.2. *The server in the protocol of Figure 3 needs to keep* $O(\mathsf{s}\lceil\frac{n}{m}\rceil)$ *ciphertexts to execute the protocol, where* m *is the number of slots in a ciphertext.*

Number of Multiplications Steps 4(a)i-4(a)ii involve only multiplications by a scalar. Steps 4(a)iv and 4(b)iii involve $2\mathsf{s}$ multiplications. Applying the inverse in Steps 4(a)iii and 4(b)ii require r_1 and r_2 multiplications, respectively. Therefore:

Lemma IV.3. *The server in the protocol of Figure 3 needs to perform* $(r_1+2)\mathsf{s} + r_2$ *multiplications of pairs of ciphertexts.*

V. Experimental Results

We implemented our protocol of Figure 3 into a system and ran extensive experiments to measure its performance. We describe our system in Section V-A, the experiments in Section V-B and the experimental results in Section V-C.

A. The Implemented System

Our system comprised of a client application and a server application. Both applications were written in C++. We used the FHE implementation of the Microsoft SEAL library [sea21], as well as IBM's HElayer library [hel] that provides a convenient development environment with wrapper access to leading FHE libraries, implementation of common basic functionalities such as computing inverse over encrypted data, and support for fast testing of versatile packing options [AAB+20].

Batched computing. We submitted our system to the iDASH competition 2021 [iDa], in which we classified a batch of 2000 viruses. The parameters of the key however dictated that each ciphertext has 4096 plaintext slots. To utilize this, we modified our packing (using the HElayers library) to allot two slots to each virus, which required a minor change in the protocol of Figure 3 (specifically, using a single rotation in Step 4(a)ii and another in Step 4(a)i to sum the two slots together). Similarly supporting smaller batches of viruses can be done efficiently by allotting more slots to each virus.

B. Experiments and Measuremeants

We first describe the experimental setup.

Hardware. We ran the client and the server on a system with a single Intel Xeon core running in 2.4GHz.

Systems' parameters. In our experiments, the threshold when computing the representatives was $\tau = 0.2$, and we tested performance on $k = 6, 7, 8, 9$, $r_1 = 1, 2, 3, 4$ and $r_2 = 1$. Parameters were chosen using standard ML methods (cross-validation), showing that $k \leq 7$ suffices for our datasets (representing typical viral sequence lengths).

The SEAL parameters were set as follows. All experiments are with a key with 128 bits of security. The scale and precision are set to 2^{34} and 2^{39}, respectively. To improve performance, the degree of the cyclotomic polynomial is the lowest that supports the depth of our protocol, which is 8192 in the experiments with $r_1 = 1$, and 16384 otherwise. The number of available plaintext slots in each ciphertext is 4096 and 8192 respectively.

The data. We ran extensive tests on the dataset provided by the iDASH competition [iDa], which included labeled DNA sequences of 4 strains of COVID-19 viruses (2000 viruses from each strain, for a total of 8000 viruses). Each virus was given as a sequence of about 29,000 DNA letters. We split the data into disjoint training and test sets. The training (test) set included 1500 (500) viruses from each strain for a total of 6,000 (2,000) viruses. Moreover, to demonstrate that our solution consistently obtains high microAUC, we tested our algorithm (Figure 2) on 3 additional viral datasets: Hepatitis-C [hcv], Dengue [dng] and Herpes [hrp], demonstrating high microAUC on all these datasets.

The experiments. To test our protocol we used the aforementioned sets of labeled viral DNA sequences. We split each set into disjoint *training* and *test* sets. The former is the server's input in our protocol, and the latter – when stripped from the labels – is the client's input. That is, the server trains (i.e., finds a representative for each strain) using the training set. The client converts the (striped from labels) test set to k-mer signatures, encrypts and sends the ciphertext to the server for classification. The server sends the encrypted outcome of the homomorphic classification to the client for decryption. Accuracy and microAUC are measured by comparing the results of the homomorphic classification in our protocol to the true labels of the test set.

Our experiments measure performance in batched classification, where (1) the client's input (i.e., the test set) consists of multiple sequences to be labelled; and (2) multiple entries of the k-mer signatures of the test set sequences are packed in each ciphertext, and processed in SIMD fashion. The threshold τ (cf. Figure 2) was set to 0.2, using cross-validation.

We repeated the experiment for different values of k (the k-mer size) and r_1 (the depth of the sub-circuit computing $\mathsf{inverse}(U_\sigma)$ in Step 4(a)iii of Figure 3) to explore how our measurements change for different values.

The measurements. We measure the following key performance metrics in our experiments:

- **Encryption time.** The client's runtime for encrypting the k-mer signatures of all sequences in its test set.
- **RAM requirement.** The amount of RAM the client used in the encryption process.
- **Communication size.** The size of the data transferred from the client to the server, i.e., the size of the encrypted test set.
- **Classification time.** The server runtime for homomorphically classifying the viruses in the encrypted test set.
- **Server RAM.** The amount of RAM the server needed to homomorphically evalute the classification task.
- **Accuracy.** The percentage of successful classifications.
- **microAUC.** See definition in Section II.

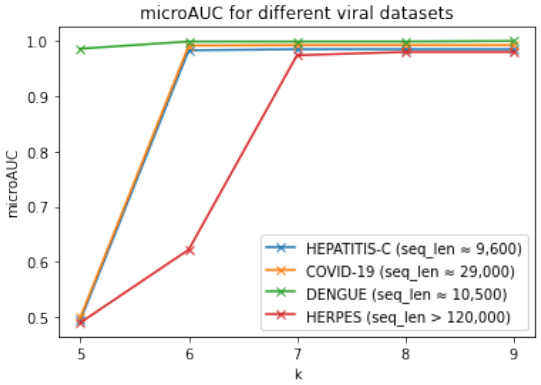
In addition we measure: (1) the *total client runtime* which includes on top of encryption time also the time it took the client to convert all viruses in the test set to their k-mer signatures, and to decrypt the encrypted result it receives from the server; and (2) *total communication size* which includes on top for the client-to-server communication also the server-to-client response.

To account for the fact that we pack multiple k-mer entries in each ciphertext, we additionally report the amortized performance per virus in the test set. Concretely, we report the amortized encryption time, amortized communication size and amortized classification time, defined to be the corresponding non-amortized measures as specified above, when divided by the number of viruses in the test set.

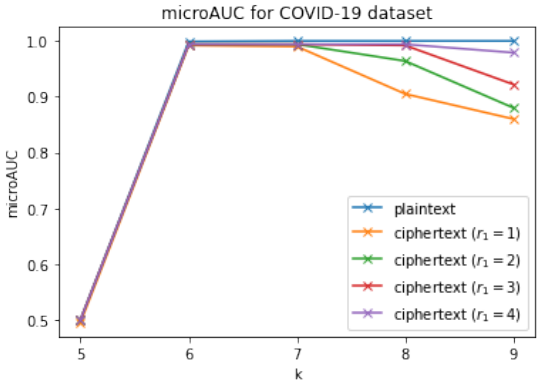
C. Results

We now discuss our key finding, which are summarized in Figure 4a and Table I.

microAUC. Our experiments show (Figure 4a) that for an appropriate choice of k our classification algorithm (Figure 2) yields almost perfect microAUC on *all* tested datasets.



(a) our algorithm on four viral datasets



(b) our protocol on $r_1 \in [1..4]$ vs. our algorithm

Fig. 4: MicroAUC in our secure protocol on encrypted data (Figure 3) vs. our algorithm on cleartext data (Figure 2). (Curves are slightly jittered, for better visualization.)

We note that the parameter k should be set in accordance with the length of the sequences in the dataset: for sequences of length roughly 10,000-30,000 DNA letters (Covid-19 [iDa], Hepatitis-C [hcv], Dengue [dng]), $k = 6$ suffices; whereas for datasets of sequences of length 150,000 (Herpes [hrp]), $k = 7$ is needed. We note that for sequences of length >150,000, larger k may be required, but this is usually not the case in viral sequences.

Our privacy-preserving protocol (Figure 3) obtains almost identical microAUC to the cleartext algorithm (Figure 2) for $k = 6, 7$; See Figure 4b. However, as k grows there is a gradual deterioration in the microAUC obtained over encrypted data compared to over cleartext. This is because the error of the low-degree polynomial approximation of the inverse grows (see ϵ_{inv} in Corollary IV.7), due to the combination of the two following facts. First, this approximation error grows proportionally to the distance from 1 of the number to be inverted. Second, the number to be inverted (i.e., the denominator in our Jaccard similarity) is the fraction of k-mers appearing in the k-mer signatures of *both* the test sequence and the type representative; and this fraction decreases when k increases, so increasing k also increases the distance from 1. To remedy the situation, we can increase the degree r_1 of the polynomial used to approximate the inverse over encrypted data, in accordance to the increase in k. Indeed, when increasing r_1 from 1 to 4, we get high microAUC even for $k = 9$; See Figure 4b.

Accuracy. In addition to measuring microAUC we also measure accuracy, showing (Table I) that our protocol obtains high accuracy (in the range of $99.8\% - 100\%$).

Runtime, communication and RAM performance over encrypted data. The performance of our privacy-preserving protocol are summarized in Table I.

The results demonstrate the good performance of our protocol. For example, for $k = 6$ and $r_1 = 1$, performance is as follows. The server's runtime for homomorphic classification of a batch of 2,048 viruses is only 1,083 milliseconds (0.53 ms amortized time), using 398 MByte RAM and obtaining 99.8% accuracy and microAUC 0.999. The client's total amortized runtime – which is dominated by the 4.46 ms it takes to encrypt the test set (amortized) – is 4.95 ms, using 276 MB of RAM. The communication is likewise dominated by sending the encrypted test set to the server, communicating 435 MB, with 1MB communicated back from server to client.

Dependency of time and memory on k. Table I demonstrates that when k increases by 1 the encryption and classification time, as well as communication (amortized and non-amortized) increase fourfold. This is supported by our theoretical analysis since there are 4^k k-mers so all indicator vectors used by our algorithm grow fourfold. The client and server RAM requirements also grow, but by smaller rates, because only parts of the memory grow fourfold. Specifically, on the client (server) only the plaintext k-mer signatures of the test sequences (the plaintext representatives of the types) grow.

Dependency of accuracy and microAUC on k, r_1. Table I shows that increasing k improves accuracy, which is also supported by our plaintext experiments. Interestingly, increasing k causes the microAUC to decrease. This is because when k grows, $|1 - \frac{U_\sigma}{4^k}|$ also grows (recall that we scale U_σ) which (as noted above) makes the inverse approximation $\mathsf{inverse}(\frac{U_\sigma}{4^k})$ less accurate. The approximation can be made more accurate by increasing r_1, as indeed demonstrated by our experiments. For example, for $k = 9$ the microAUC improved from 0.86 ($r_1 = 1$) to 0.979 ($r_1 = 4$). Another interesting point is that accuracy does not decrease. This is because increasing k perturbs the values in the s-tuple of Jaccard scores that we compute for each test sequences, but *does not change the ordering of scores within the tuple*. Since accuracy depends only on the largest score, this perturbation keeps the accuracy unchanged. However, the relative ordering of values with respect to other s-tuples *does* change, which decreases the microAUC.

VI. Extension to Strong Privacy

Privacy against the client is not necessarily guaranteed by our protocol of Figure 3. This is because the ciphertexts which Cl receives in Step 4c might reveal information about the *homomorphic computation* which S performed on the ciphertexts it received from Cl, and consequently on S's input

k	r_1	Batch size	Encryption (ms)	Encryption amortized (ms)	Client RAM (MB)	Comm. (MB)	Comm. amortized (MB)	Classify (ms)	Classify amortized (ms)	Server RAM (MB)	Accuracy	microAUC
6	1	2,048	9,152	4.5	276	435	0.2	1,083	0.5	398	99.8%	0.999
7			36,153	18.0	464	1,700	0.8	3,238	1.6	960	100.0%	0.991
8			148,595	73.0	1,213	6,800	3.3	9,506	4.6	3,211	100.0%	0.905
9			581,989	284.0	4,213	41,000	20.0	31,111	15.2	12,212	100.0%	0.860
6	2	4,096	22,359	5.4	686	1,536	0.4	2,591	0.6	804	99.8%	0.999
7			86,827	21.0	874	6,144	1.5	8,819	2.1	1,366	100.0%	0.999
8			340,452	83.0	1,624	24,577	6.0	22,946	5.6	3,617	100.0%	0.964
9			1,416,951	345.0	4,624	98,311	24.0	85,173	20.8	12,618	100.0%	0.880
6	3	4,096	25,033	6.1	885	1,792	0.4	3,234	0.8	1,003	99.8%	0.999
7			97,699	23.8	1,072	7,168	1.7	9,498	2.3	1,565	100.0%	0.999
8			386,287	94.3	1,822	28,673	7.0	32,343	7.9	3,815	100.0%	0.996
9			1,529,859	374.0	4,822	114,695	28.0	82,411	20.1	12,816	100.0%	0.922
6	4	4,096	27,746	6.8	1,117	2,048	0.5	4,076	1.0	1,230	99.8%	0.999
7			118,878	29.0	1,303	8,192	2.0	13,699	3.3	1,792	100.0%	0.999
8			442,453	108.0	2,053	32,769	8.0	31,175	7.6	4,043	100.0%	0.999
9			1,721,851	420.0	5,054	131,079	32.0	97,681	23.8	13,044	100.0%	0.979

TABLE I: Performance of our protocol over encrypted data (Figure 3) for different values of the k-mer size k (1st column); and depth r_1 of the inverse sub-circuit of Step 4(a)iii (2nd column). Other columns show: batch size, i.e., number of viruses we classify in the protocol (3rd column); client runtime to encrypt the entire batch (4th column), and amortized for a single test virus (5th column); the amount of RAM consumed by the client during the protocol (6th column); the size of client-to-server communication for the entire batch (7th column), and amortized for a single test virus (8th column); server runtime to classify a batch (9th column), and amortized for a single virus (10th column); the amount of RAM consumed by the server during classification (11th column); the classification accuracy (12th column) and microAUC (13th column).

D (since the computation which S performs depends on D). We next discuss how to extend the protocol to achieve privacy against the client by employing sanitization.

Sanitization and Circuit Privacy. A ciphertext *sanitization* scheme re-randomizes ciphertexts such that any two sanitized ciphertexts decrypting to the same plaintext are statistically close *even given the secret decryption key*. Sanitization implies *circuit privacy* (in the semi-honest model) in the sense that a sanitized ciphertext reveals no information on the circuit used to compute it via homomorphic evaluation, even if the adversary knows the secret decryption key.

Definition VI.1 (Sanitization Scheme [DS16])**.** *A PPT algorithm* San *for an FHE scheme* $\mathsf{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is a* Sanitization Scheme *if it takes a public key* pk *and a ciphertext* c *and returns a ciphertext, so that with probability* $\geq 1 - \mathsf{negl}(\kappa)$ *over the choice of* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ *the following holds:*

- (Message-preservation) *for every ciphertext* c*:* $\mathsf{Dec}(\mathsf{sk}, \mathsf{San}(\mathsf{pk}, c)) = \mathsf{Dec}(\mathsf{sk}, c)$.
- (Sanitization) *for every pair* c, c' *of ciphertexts s.t.* $\mathsf{Dec}(\mathsf{sk}, c) = \mathsf{Dec}(\mathsf{sk}, c')$ *the statistical distance*

$$\Delta\left((\mathsf{San}(\mathsf{pk}, c)|(\mathsf{pk}, \mathsf{sk})), (\mathsf{San}(\mathsf{pk}, c')|(\mathsf{pk}, \mathsf{sk}))\right)$$

is at most $\mathsf{negl}(\kappa)$.

Extending the protocol. We enhance the protocol of Figure 3 to guarantee privacy against both server and client, simply by instructing the server to sanitize the result ciphertexts prior to sending them to the client.

Protocol VI.2 (Strongly Private Viral Strain Classification)**.** *The protocol employs a sanitization scheme* San*. It is identical to the protocol of Figure 3, except for the following change in Step 4c: instead of sending* $J_1, \ldots, J_\mathsf{s}$ *to the client, the server first performs* $J'_\sigma \leftarrow \mathsf{San}(\mathsf{pk}, J_\sigma)$ *for every* $\sigma = 1, \ldots, \mathsf{s}$*, and sends* $J'_1, \ldots, J'_\mathsf{s}$ *to the client.*

Security Analysis. The enhanced protocol (Protocol VI.2) guarantees privacy against the client (as well as the server), because the sanitization ensures that the client cannot learn anything from these ciphertexts beyond their underlying message (see Definition VI.1). In particular, the client learns nothing (beyond what can be inferred from her input and output) on the the server's data or model although they had influenced the homomorphic operations performed by the server. More formally, the sanitized ciphertexts are statistically close to any other sanitized ciphertext encrypting the same message, and therefore they can be efficiently simulated from Cl's output. This is summarized in the following theorem:

Theorem VI.3. *Let* $\mathsf{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be an* ϵ*-approximate FHE scheme, and let* San *be a sanitization scheme. Then Protocol VI.2, when instantiated with* E *as the underlying FHE scheme and* San *as the underlying sanitization scheme, is a* strongly *private viral strain classification protocol (as in Definition III.2).*

We provide here a proof sketch, see the full version [Ano22] for the complete proof.

Proof of Theorem VI.3 (sketch). **Correctness** is analogous to the analysis of Section IV-C1 while accounting also for the error incurred by executing sanitation.

Privacy (against the server) follows identically to the proof of Theorem IV.1.

Privacy (against the client). We describe a PPT simulator

$\mathsf{Sim}_{\mathsf{Cl}}$ which simulates the view $\mathsf{View}^{\Pi}_{\mathsf{Cl}}$ of the honest Cl in Π. Sim_{Π} is given the input x of Cl, its output $\left(J^i_1, \ldots J^i_{\mathsf{s}}\right)_{1\leq i\leq n}$, as well as N and the public parameters params (which include m, ℓ, κ, and the length n of the client's input), and operates as follows:

- Samples random coins R for Cl and uses it to determine the encryption keys $(\mathsf{pk}, \mathsf{sk})$.
- Randomly encrypts $\overline{J}^i_\sigma \leftarrow \mathsf{Enc}(\mathsf{pk}, J^i_\sigma)$ for $i = 1, \ldots, n$ and $\sigma = 1, \ldots, \mathsf{s}$.
- Randomly sanitizes $\widehat{J}^i_\sigma \leftarrow \mathsf{San}(\mathsf{pk}, \overline{J}^i_\sigma)$ for $i = 1, \ldots, n$ and $\sigma = 1, \ldots, \mathsf{s}$.
- Outputs x, N, params, R and $\widehat{J}^1_1, \ldots, \widehat{J}^n_{\mathsf{s}}$ (that is, $\widehat{J}^1_1, \ldots, \widehat{J}^n_{\mathsf{s}}$ simulate the encrypted scores which the server sent to the client).

We claim that the simulated and real views are computationally close. Roughly, we need to prove that the simulated encrypted scores $\widehat{J}^1_1, \ldots, \widehat{J}^n_{\mathsf{s}}$ are computationally indistinguishable from the real-world ciphertexts which the client received from the server. This follows from the security of the sanitization scheme San using a standard hybrid argument in which we move from the output of $\mathsf{Sim}_{\mathsf{Cl}}$ to $\mathsf{View}^{\Pi}_{\mathsf{Cl}}$ by replacing each of the $n \cdot \mathsf{s} = \mathsf{poly}(\kappa)$ ciphertexts from real to simulated. See the full version [Ano22] for the complete proof. □.

Complexity overhead. Protocol VI.2 is generic in the sense that it can be instantiated with any sanitization scheme for the FHE scheme used in the protocol. The complexity overhead over the protocol of Figure 3 is the complexity of the employed sanitization. We leave choosing the appropriate algorithm – based on the specific requirements of the application – to future work.

Empirical evaluation. We empirically evaluate Protocol VI.2 while following the sanitization approach of Ducas and Stehlé [DS16]. The sanitization algorithm of [DS16] operates by repeatedly randomizing the ciphertext and then bootstrapping. Randomization is by homomorphically adding an encryption of zero with appropriate (small) amount of noise. This is reminiscent to the noise flooding technique [Gen09], albeit with a much smaller amount of noise, which is sufficient due to the use of repeated cycles of randomization-then-bootstrapping [DS16]. The encryptions of zero can be precomputed in an offline phase preparing sufficiently many encryptions to provide a fresh encryption for each classification query; so the exact parameters have little influence on the online complexity. The online complexity the time to compute the required number of cycles, each computing one homomorphic addition followed by bootstrapping. The required number of cycles is proved to be 1-2 for [BV14] (when parameters are as in [HS15]); see [DS16, Sec. 4]. Analyzing the number of cycles that suffices for CKKS is beyond the scope of our work. Instead, we set the number of cycles as a tuneable parameter to be set by the user, and provide an empirical evaluation on 1-5 cycles that can be extrapolated to an arbitrary number of cycles.

Our experimental setup is analogous to the one specified in Section V, using the same hardware, the same encryption scheme (CKKS) with key initialized with the same security parameter, scale and precision (128, 2^{34}, and 2^{39}, respectively), using the same parameters $\tau = 0.2$, $k = 6$, $r_1 = 1$ and $r_2 = 1$, running experiments on the same data, and taking the same measurements. However, we use a different library implementing CKKS: HEAAN [Cry22] rather than Microsoft SEAL [sea21]. This is because the Microsoft SEAL library used in Section V does not support bootstrapping, which is required for the sanitization approach of [DS16]. Moreover, incorporating sanitization into the homomorphic computations at the server side increases the multiplicative depth by 4, which in turn requires initializing larger keys to support the computation. Consequently, we set the degree of the cyclotomic polynomial to be 65,536 (cf. 8K-16K in our experiments without sanitization). The number of available plaintext slots in each ciphertext is half the degree of the cyclotomic polynomial, i.e., 32,768, and the batch size in our experiments is 16,384 (because we use 2 slots for each virus).

The results are given in Table II. The first row shows a baseline of running with no sanitization ($BS = 0$) using the HEAAN library. The results are comparable to the results when running with SEAL. When adding sanitization ($BS \neq 0$ in the next rows) we generated a deeper key. Indeed we see that the encryption time and the communication increased from 53 seconds and 6,100MB to 79 seconds and 11,000MB, respectively (the client generates the same key regardless to the number of bootstrapping the server performs). The classification time increased from 12 seconds when $BS = 0$ to 47 seconds when $BS = 1$ and then with another ≈ 25 seconds with each additional bootstrapping iteration. The accuracy and microAUC did not change when applying bootstrapping.[9]

VII. Related Works

Privacy-preserving genome analysis. Prior work on privacy-preserving genome analysis focused on Genome Wide Association (GWAS), where statistical genotype to phenotype (genomic sequence to physical properties) correlations are securely inferred from large datasets, see e.g. [LYS15], [SB16], [BMA+18], [BGPG20]. In contrast, we focus on viral sequence (strain) classification.

Comparison through k-mer signatures. The use of k-mer signatures has been studies in several contexts related to molecular biology and genomics. Drmanac et al. [DDS+93] and Shamir et al. [ST01] introduced k-mer signatures in the context of sequencing by hybridization, and several works (e.g., [BP97], [ST01]) then formalized algorithmic approaches to inference based on k-mer signatures. Following these initial works, k-mers were used in other sequenceing-related

[9]Notice that the accuracy and microAUC are unaffected by using bootstrapping, despite the increase in decryption noise due to the approximate nature of CKKS. Intuitively, accuracy is determined by the relative class scores for the tested instance. The difference between the scores is larger by orders of magnitude compared to the added noise. Therefore, the relative ranking between the classes is unaffected by sanitization. The explanation for microAUC is similar.

BS	Batch size	Encryption (ms)	Encryption amortized (ms)	Client RAM (MB)	Comm. (MB)	Comm. amortized (MB)	Classify (ms)	Classify amortized (ms)	Server RAM (MB)	Accuracy	microAUC
0	16,384	53,000	3.23	10,800	6,100	0.37	12,000	0.73	11,337	99.8%	0.999
1							47,000	2.86	14,125		
2							68,000	4.15	14,200		
3	16,384	79,000	4.82	10,820	11,000	0.67	92,000	5.61	14,268	99.8%	0.999
4							118,000	7.20	14,282		
5							148,000	9.03	14,300		

TABLE II: Performance of our extended protocol over encrypted data (Protocol VI.2, Section VI) for number of sanitization randomize-then-bootstrap cycles ranging from 0 (no sanitization) to 5 cycles (first column). In all experiments $k = 6$ and $r_1 = 1$ (the depth of the inverse sub-circuit of Step 4(a)iii), the other columns are as in Table I.

tasks such as resequencing [PAS02], motif finding [ELYY07], [LY12] and system design [BDKSY00]. More recently, methods were developed for using k-mer signatures of short reads sequence reconstruction [LCRP16], [CPT11].

Fully Homomorphic Encryption (FHE) [RAD78], [Gen09] allows one to compute over encrypted data – without knowing the secret decryption key – as if it weren't encrypted. There has been much interest recently in obtaining fast privacy-preserving machine learning solutions utilizing HE, e.g., in [GBDL+16], [SKGK18], [HTGW18], [JKLS18], [KSW+18], [CKKS18], [KSK+18], [BLCW19], [ASWY19], [ALR+20], [RLPD20]. In particular, the challenge of the iDASH secure genome analysis competition 2021 [iDa] was to perform viral strain classification over HE encrypted sequences.[10]

Packing two reals in each complex number slot in CKKS. Another approach for packing two reals in each complex number slot was previously explored in [BCCW19]. However their technique supports only linear operations over encrypted data (i.e. multiplication by a scalar and homomorphic addition). In contrast, our technique supports, on top of linear operations, also computing the ciphertext-to-ciphertext operations needed for our protocol; specifically, we support homomorphic evalution of the inner-product of two vectors of encrypted values. The independent work [HPC+22] proposed a packing similar to ours that likewise supports computing inner-product over packed encrypted vectors.

VIII. Conclusions and Future Work

In this paper we proposed a practically-efficient privacy-preserving viral strain classification protocol. Specifically, we developed a protocol that allows a client (a lab) to obtain from the server (a centralized repository) information about the client's private virus sequences without disclosing the sequences themselves, or any pertinent information about them. Privacy holds against active servers, and correctness against passive ones. This is motivated by scenarios in which client's privacy, business or legal concerns disallow exposing the sequence to the server, whereas the server's classification-as-a-service business model disincentivizes sending its dataset (or the classification model extracted from it) to the client. Our solution achieves excellent accuracy ($99.8\% - 100\%$) and microAUC (0.999); low complexity (homomorphically evaluating circuits of multiplicative depth 4 and $3\mathsf{s}+1$ multiplications, for s the number of viral types); fast concrete performance (amortized client and server runtime of 4.95ms and 0.53ms respectively); and a rigorous security guarantee. Furthermore, we presented an extension of our protocol that guarantee also the server's privacy against passive clients. The extended protocol provides the same high accuracy and microAUC, with amortized per-sequence overhead of 1.6ms in client runtime, under 10ms in server runtime, and 0.3MB in communication.

Our protocol extends further to settings where both the sequences and model are encrypted. This is motivated, e.g., by settings in which the centralized repository wishes to outsource the computation to a cloud service without exposing information on its model or training data.

Strengthening our protocol to guarantee privacy of the server's data against *active* clients is an interesting problem. A possible approach to enhance the server privacy is to develop business-oriented strategies targeting *rational* clients (in the game theoretic sense) rather than active ones. For example, this may entail developing per-query pricing strategies that increase with the number of queries, to financially protect the server against the growing privacy risk. We leave the development of such modeling and analysis to future work.

[10] Our solution won 3rd place in iDASH competition [iDa].

References

[AAB+20] Ehud Aharoni, Allon Adir, Moran Baruch, Gilad Ezov, Ariel Farkash, Lev Greenberg, Ramy Masalha, Dov Murik, and Omri Soceanu. Tile tensors: A versatile data structure with descriptive shapes for homomorphic encryption. *CoRR*, abs/2011.01805, 2020.

[ALR+20] Adi Akavia, Max Leibovich, Yehezkel S Resheff, Roey Ron, Moni Shahar, and Margarita Vald. Privacy-preserving decision trees training and prediction. In *ECML/PKDD (1)*, pages 145–161, 2020.

[Ano22] Anonymous. *Efficient Privacy-Preserving Viral Strain Classification via k-mer Signatures and FHE (anonymized full version, attached to this submission as supplementary material, also available at: https://github.com/viralclassification2/paper2)*. 2022.

[ASWY19] Adi Akavia, Hayim Shaul, Mor Weiss, and Zohar Yakhini. Linear-regression on packed encrypted data in the two-server model. In *Proc. of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Crypto.*, pages 21–32, 2019.

[auc] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score.

[BCCW19] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. Ngraph-he2: A high-throughput framework for neural network inference on encrypted data. New York, NY, USA, 2019. ACM.

[BDKSY00] Amir Ben-Dor, Richard Karp, Benno Schwikowski, and Zohar Yakhini. Universal dna tag systems: a combinatorial design scheme. *Journal of Comput. Biology*, 7(3-4):503–519, 2000.

[BGPG20] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. Secure large-scale genome-wide association studies using homomorphic encryption. *Proceedings of the National Academy of Sciences*, 117(21):11608–11613, 2020.

[BLCW19] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. ngraph-he: a graph compiler for deep learning on homomorphically encrypted data. In *Proc. of the 16th ACM Intl. Conf. on Computing Frontiers*, pages 3–13, 2019.

[BMA+18] Charlotte Bonte, Eleftheria Makri, Amin Ardeshirdavani, Jaak Simm, Yves Moreau, and Frederik Vercauteren. Towards practical privacy-preserving genome-wide association study. *BMC bioinformatics*, 19(1):1–12, 2018.

[BMC+00] Meltzer Bittner, Paul Meltzer, Yidong Chen, Youfei Jiang, Elisabeth Seftor, M Hendrix, M Radmacher, Richard Simon, Zohar Yakhini, Amir Ben-Dor, et al. Molecular classification of cutaneous malignant melanoma by gene expression profiling. *Nature*, 406(6795):536–540, 2000.

[BP97] Igor Belyi and Pavel A Pevzner. Software for dna sequencing by hybridization. *Bioinformatics*, 13(2):205–210, 1997.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on computing*, 43(2):831–871, 2014.

[CKK+19] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 415–445. Springer, 2019.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Intl. Conf. on the Theory and Application of Crypto. and Info. Security*, pages 409–437. Springer, 2017.

[CKKS18] Jung Hee Cheon, Duhyeong Kim, Yongdai Kim, and Yongsoo Song. Ensemble method for privacy-preserving logistic regression based on homomorphic encryption. *IEEE Access*, 6:46938–46948, 2018.

[CPT11] Phillip EC Compeau, Pavel A Pevzner, and Glenn Tesler. Why are de bruijn graphs useful for genome assembly? *Nature biotechnology*, 29(11):987, 2011.

[Cry22] CryptoLab. HEaaN: Homomorphic Encryption for Arithmetic of Approximate Numbers, version 3.1.4, 2022.

[DDS+93] R Drmanac, S Drmanac, Z Strezoska, Tt Paunesku, I Labat, M Zeremski, J Snoddy, WK Funkhouser, B Koop, L Hood, et al. Dna sequence determination by hybridization: a strategy for efficient large-scale sequencing. *Science*, 260(5114):1649–1652, 1993.

[DIB97] Joseph L DeRisi, Vishwanath R Iyer, and Patrick O Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278(5338):680–686, 1997.

[dng] https://www.viprbrc.org/brc/home.spg?decorator=flavi_dengue.

[DS16] Léo Ducas and Damien Stehlé. Sanitization of FHE ciphertexts. In *Advances in Cryptology – EUROCRYPT 2016*, pages 294–310. Springer Berlin Heidelberg, 2016.

[ELYY07] Eran Eden, Doron Lipson, Sivan Yogev, and Zohar Yakhini. Discovering motifs in ranked lists of dna sequences. *PLoS computational biology*, 3(3):e39, 2007.

[GBDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Intl. Conf. on machine learning*, pages 201–210. PMLR, 2016.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[Gol04] Oded Goldreich. *The Foundations of Crypto. - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[hcv] https://www.viprbrc.org/brc/home.spg?decorator=flavi_hcv.

[hel] IBM HElayers (release 1.0).

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.

[HPC+22] Seungwan Hong, Jai Hyun Park, Wonhee Cho, Hyeongmin Choe, and Jung Hee Cheon. Secure tumor classification by shallow neural network using homomorphic encryption. *BMC genomics*, 23(1):1–19, 2022.

[hrp] https://www.viprbrc.org/brc/home.spg?decorator=herpes.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for helib. In *EUROCRYPT (1)*, pages 641–670. Springer, 2015.

[HTGW18] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. Privacy-preserving machine learning as a service. *Proc. Priv. Enhancing Technol.*, 2018(3):123–142, 2018.

[iDa] iDASH privacy & security workshop 2021 – secure genome analysis competition. http://www.humangenomeprivacy.org/2021/competition-tasks.html.

[JKLS18] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proc. of the 2018 ACM SIGSAC Conf. on Computer and Commun. Security*, pages 1209–1222, 2018.

[KSK+18] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11(4):23–31, 2018.

[KSW+18] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, Xiaoqian Jiang, et al. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics*, 6(2):e8805, 2018.

[LCRP16] Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on de bruijn graphs. *BMC bioinformatics*, 17(1):1–12, 2016.

[LY12] Limor Leibovich and Zohar Yakhini. Efficient motif search in ranked lists and applications to variable gap motifs. *Nucleic acids research*, 40(13):5832–5847, 2012.

[LYS15] Wen-Jie Lu, Yoshiji Yamada, and Jun Sakuma. Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption. In *BMC medical informatics and decision making*, volume 15, pages 1–8. Springer, 2015.

[PAS02] Itsik Pe'er, Naama Arbili, and Ron Shamir. A computational method for resequencing long dna targets by universal oligonucleotide arrays. *Proc. of the National Acad. of Sciences*, 99(24):15492–15496, 2002.

[PSZ+12] Brett E Pickett, Eva L Sadat, Yun Zhang, Jyothi M Noronha, R Burke Squires, Victoria Hunt, Mengya Liu, Sanjeev Kumar, Sam Zaremba, Zhiping Gu, et al. Vipr: an open bioinformatics database and analysis resource for virology research. *Nucleic acids research*, 40(D1):D593–D598, 2012.

[RAD78] 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.

[RLPD20] M Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. Heax: An architecture for computing on encrypted data. In *Proc. of the 25th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 1295–1309, 2020.

[SB16] Sean Simmons and Bonnie Berger. Realizing privacy preserving genome-wide association studies. *Bioinformatics*, 32(9):1293–1300, 2016.

[sea21] Microsoft SEAL (release 3.6.6), 2021. Microsoft Research, Redmond, WA.

[SKGK18] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. Tapas: Tricks to accelerate (encrypted) prediction as a service. In *Intl. Conf. on Machine Learning*, pages 4490–4499. PMLR, 2018.

[ST01] Ron Shamir and Dekel Tsur. Large scale sequencing by hybridization. In *Proc. of the 5th Annual Intl. Conf. on Comput. biology*, pages 269–277, 2001.

[tra] https://www.who.int/en/activities/tracking-SARS-CoV-2-variants/.

[vpr] https://www.viprbrc.org/.