A Technique for Secure Variant Calling on Human Genome Sequences Using SmartNICs

Praveen Rao
University of Missouri, USA
praveen.rao@missouri.edu

Khawar Shehzad
University of Missouri, USA
khawar.shehzad@missouri.edu

***Abstract*—Genome sequences of individuals constitute sensitive data and must be safeguarded from unauthorized access or disclosure. While encryption of human genome sequences at rest and in transit is an attractive solution, the large size of these genomes can introduce non-trivial cost for encryption/decryption of input/intermediate files consumed and produced during genome analysis. In this paper, we focus on a key task called *variant calling*, which is used to identify variants in an individual's genome compared to the reference genome. We present a novel technique for *secure variant calling* on human genomes while safeguarding the privacy of individuals. Our technique employs the Data Plane Development Kit (DPDK) and SmartNICs (smart network interface cards) for offloading the task of encryption/decryption. Using named pipes provided by the underlying OS, our technique enables the SmartNIC to communicate with existing bioinformatics tools without any code modifications to them. As a result, all files consumed and produced during variant calling are always encrypted rendering them unreadable to an adversary if a data breach occurs. We evaluated the efficiency of our technique on different genome sequences and observed that our technique added little overhead to the variant calling pipeline on encrypted sequences (about 1% slower) compared to the normal approach of executing variant calling on unencrypted sequences.**

I. Introduction

With technological advances in whole genome sequencing (WGS) and lower cost of sequencing [58], it is now feasible to employ WGS for large-scale genomic studies and clinical practice [8], [6]. It was predicted that by 2025, exabytes of human genome data could be produced [64]. By analyzing the genomes of individuals, medical professionals and scientists can determine their risk for complex diseases (e.g., cancer) and develop effective treatment protocols.

A whole genome sequence of an individual can consume gigabytes (GBs) of storage space due to millions of reads, which are short (overlapping) fragments of the deoxyribonucleic acid (DNA) in the genome [31]. Variant calling is a key task that involves identifying variants in an individual's genome compared to the reference genome [51]. There are different types of variants such as single nucleotide polymorphisms (SNPs), short insertions/deletions (indels), and structural variants[1]. A variant calling pipeline involves several stages, namely, reading the large sequence files, aligning the reads against the reference genome, additional pre-processing steps to mitigate sequencing errors, and finally invoking a variant caller [42]. The pipeline execution is typically compute and I/O intensive in nature due to the large size of human genomes [52], [36]. The raw variants produced by a pipeline can be further processed to obtain meaningful genomic insights.

Data breaches cost significant financial losses to organizations globally. In 2023, the average cost of a data breach globally was $4.45 million.[2] Recently, 23AndMe, a genetic testing company, confirmed a data breach that affected 6.9 million individuals. DNA information and other personal information were stolen by hackers [14]. Theft of genomic data can also harm a country's economic and national security [11]. In fact, an individual's genome sequence is unique except for identical twins [4]. A study showed that de-identified genomic data can risk re-identification when combined with publicly available geneology databases [33]. Hence, genomic data are considered protected health information (PHI) [26]. Encrypting genomic data at rest and in transit can enable compliance with laws such as the Health Insurance Portability and Accountability Act (HIPAA) [10] and the General Data Protection Regulation (GDPR) [16] with growing use of public cloud service providers for analyzing genomic data [44], [62], [17], [18], [19]. Note that cloud computing offers sufficient compute and storage resources necessary for analyzing large-sized human genome sequences.

Using encryption on genome sequences and other intermediate files produced during variant calling can cause non-trivial overhead as GBs of data are read from/written to persistent storage. For example, a small-sized whole human genome (*15* GB in compressed form) containing 199 million (short) reads, caused a total of *63* GB to be read from and *50* GB to be written to disk storage in plaintext during variant calling. Fortunately, SmartNICs, which are programmable NICs, can be used to offload cryptographic tasks, thus freeing up the host processor cores to complete other tasks. They are used in data centers for efficient networking, storage, and security [13].

Prior research efforts have explored the use of SmartNICs for tasks such as packet processing [46], [22], [69], security [50], [41], [65], [72], and distributed processing [48], [32], [60]. Other efforts have investigated privacy-preserving techniques for analysis in genome wide association studies (GWAS) [45], [67], [23], [43]. Unfortunately, techniques such as homomorphic encryption and secure multiparty computa-

[1]https://m.ensembl.org/info/genome/variation/prediction/classification.html

[2]https://www.ibm.com/reports/data-breach

tion on large genome sequences can lead to high computational/communication cost [43], [28]. Some approaches for secure processing of genomes require specialized computing infrastructure (e.g., quantum secure cloud [28]). To the best of our knowledge, no one has investigated how SmartNICs can be effectively used for *secure variant calling* on individuals' genomes in a cloud infrastructure while protecting their data from adversaries.

Motivated by the aforementioned reasons, we present a technique called SVC (**S**ecure **V**ariant **C**alling) for secure variant calling on human genomes. The design goal of SVC is to safeguard the genomes of individuals from unauthorized users. Our key contributions are as follows:

- We developed SVC to perform the variant calling pipeline on an encrypted human genome sequence such that all files read from and written to persistent storage/disk during variant calling are always encrypted. Only encrypted data can be accessed by an adversary, thus safeguarding the privacy of individuals' genomes.
- SVC has a unique design consisting of a client and a server module. The client module runs on a user's machine that has the human genome. After generating a random symmetric key for a genome, it encrypts the genome and sends it to a remote server machine (e.g., in a cloud environment). The server module executes the variant calling pipeline on the encrypted genome. It offloads the encryption/decryption tasks to a SmartNIC using a custom DPDK application. It leverages named pipes managed by the underlying OS to exchange data between the SmartNIC and existing bioinformatics tools for variant calling without any code modifications to these tools. The encrypted output file of variant calling (containing raw variants) is copied to the user's machine and decrypted by the client module using the same symmetric key for that genome.
- SVC introduced very little overhead to the variant calling pipeline. On the tested whole genome sequences with the Advanced Encryption Standard (AES) encryption, the variant calling pipeline ran slightly slower on encrypted sequences (i.e., about 1% slower) compared to executing on unencrypted/plaintext sequences. Hence, SVC is efficient for secure variant calling on human genome data.

The rest of the paper is organized as follows: Section II provides background/related work on variant calling pipelines, privacy-preserving techniques for genomic data, SmartNICs/DPDK, and our threat model. Section III presents the design and implementation of SVC. Section IV presents the performance evaluation of SVC, and we conclude in Section V.

II. Background and Related Work

A. Variant Calling Pipelines

A typical variant calling pipeline for an individual's DNA sample [42] has a set of stages. It starts by reading of raw unmapped reads (e.g., in FASTQ format [2]) output by a sequencer. Then the alignment of the reads with a reference genome (e.g., GRCh38 [51]) is performed using algorithms such as BWA-MEM [47] to produce mapped reads that are stored in the SAM/BAM format [12]. Sorting of aligned reads, marking of duplicate reads, execution of base quality score recalibration (BQSR) [15] and local realignment around the indels are performed next. Finally, a variant calling method (FreeBayes [30], HaplotypeCaller [15], DNAscope [27]) is executed to produce raw variants in the VCF format [29]. The subsequent downstream processing steps include variant filtering and annotation steps on the raw variants. GWAS can also be performed for a specific disease once the VCF files of a group of individuals have been computed.

There is continued interest in accelerating DNA variant calling pipelines using distributed computing, big data technologies, and hardware accelerators. GATK4 [15] employed Apache Spark [70] for multithreading and parallelization. NVIDIA's Parabricks accelerated GATK pipelines using GPUs [54]. Google's DeepVariant [55] used deep learning for variant calling and operated directly on aligned reads. Nothaft et. al. [52] created ADAM [53] and Cannoli [7] for processing large-scale genomic datasets using Spark's primitives and parallelization of the alignment process and variant calling by reusing existing bioinformatics tools. Later, AVAH [57], [56] and AVAH* [25] were developed to process a workload of genomes faster using cluster computing and synchronous computations. Illumina's DRAGEN Platform accelerated the variant calling pipeline using field-programmable gate arrays (FPGAs) [59]. Sentieon [62] developed highly optimized software-based algorithms for variant calling using CPUs.

The aforementioned approaches aimed at improving the performance and efficiency of variant calling rather than data security/privacy of genomes.

B. Privacy-Preserving Techniques for Genomic Data Analysis

Several privacy-preserving techniques have been proposed for GWAS, which is performed after variant calling. Some of these used secure multi-party computation for formation of case-control groups and statistical testing [39]; homomorphic encryption for statistical testing [45], [67], SNP search [63], secure count querying [34], and ancestry inference [49]; and sketching techniques for achieving privacy [35], [43]. Some attempted specialized encryption techniques for protecting aligned reads [20], differential privacy for selecting datasets in GWAS [73], secure distributed genome analysis for GWAS [71], and cryptographic techniques for meta-analysis in GWAS [68]. Trusted execution environments (TEEs) such as Intel software guard extensions (SGX) have also been explored for processing genomic data in GWAS [23], [43].

A few tools were developed to encrypt genomic data. Cryfa [37] is a secure encryption tool for FASTQ, BAM, and VCF files. It uses AES for encrypting files. Crypt4GH [61] is a file format standard for native access to encrypted data and uses envelope encryption. Both are suited for securely transmitting and storing genomic data. However, existing variant calling pipelines such as GATK4 would require code modifications to use Cryfa/Crypt4GH. Although Crypt4GH is natively supported by SAMtools [24], other software tools in a variant calling pipeline (e.g., for alignment and variant calling) require code modifications to use this file format. Alternatively,

if Cryfa/Crypt4GH are used without code modifications to bioinformatics tools, then encrypted data should be first decrypted by these tools and saved to persistent storage before being consumed by bioinformatics tools, thereby exposing the data to an adversary.

The above efforts aim to protect an individual's privacy for (a) a particular file containing genomic data, (b) querying of genomic data, (c) GWAS analysis after producing the VCF file, or (d) require code modifications to existing bioinformatics tools while hindering adoption. Hence, they are insufficient/inapplicable to securely execute the entire variant calling pipeline while protecting the genomes from data breaches. Although Fujiwara et. al.'s work [28] is similar to ours in terms of protecting genomic data against data breaches, their quantum cloud infrastructure is highly specialized and requires quantum key distribution.

C. SmartNICs and DPDK

SmartNICs are programmable NICs that are used for more efficient data center networking, security, and storage [13]. They act as accelerators and contain computing elements (e.g., x86 cores, FPGAs). Tasks from server CPUs (e.g., network packet processing, cryptographic operations) can be offloaded to SmartNICs freeing up CPU cores for running other tasks such as customer applications. DPDK [9] is a framework for fast packet processing in data plane applications and supports 20+ SmartNIC vendors. NICs and virtual I/O devices are accessed by polling as to avoid the performance overhead imposed by interrupt processing. A cryptography device library is used to perform cryptographic operations, both in hardware and software. Note that GPU-based encryption/decryption is beyond the scope of our work.

D. Threat Model and Assumptions

We consider an attacker that has maliciously gained access to a server (e.g., due to a compromised user account) that is used to execute variant calling. The server can be a bare-metal machine or a virtual machine (VM) hosted in a cloud environment. The attacker attempts to steal the FASTQ/BAM/SAM/VCF files of individuals on the server. Using these files, the attacker can exploit the genomic data of individuals to compromise their privacy or illegally advance biotechnology. We assume that the server OS and bioinformatics tools are trusted and cannot be exploited by the attacker.

III. Design of SVC

In this section, we present the novel design of SVC for secure variant calling. We first discuss challenges in designing SVC and then discuss its client-server architecture.

A. Challenges

Without loss of generality, we consider a typical DNA variant calling pipeline shown in Table I. Each stage is described briefly in the table, wherein S_1 creates the interleaved FASTQ file, S_2 performs alignment, S_3 does pre-processing, and finally S_4 invokes the variant caller. The intermediate files can be used for quality control and generating useful statistics.

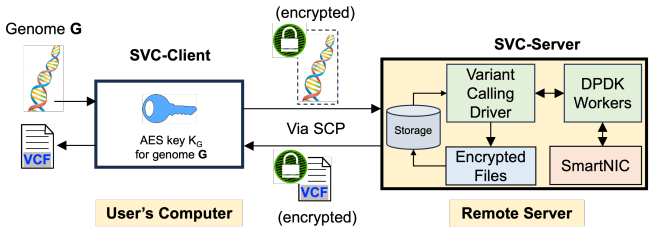

Fig. 1. Overall architecture of SVC

TABLE I
A Typical DNA Variant Calling Pipeline

Stage	Description
S_1	Construct the interleaved FASTQ file for the paired-end FASTQ files
S_2	Align the interleaved FASTQ file against the reference genome (e.g., using BWA-MEM) to produce a `.sam` file
S_3	On the `.sam` file, sort the aligned reads (e.g., coordinate order) and mark duplicate reads to produce a `.bam` file
S_4	Invoke the variant calling method (e.g., FreeBayes) to produce a `.vcf` file

Several challenges arise in designing SVC. First, widely-adopted bioinformatics tools should not require code modifications to work with SVC. These tools operate on plaintext input and produce plaintext output. (Use of TEEs or homomorphic encryption would require code modifications to them.) Otherwise, it will be difficult for bioinformaticians to adopt SVC. Second, all files that can compromise an individual's identity (i.e., FASTQ, SAM/BAM, and VCF files), which are read and written to disk during the execution of the variant calling pipeline, should be encrypted. This means that each encrypted file should be read by a SmartNIC, and the decrypted output should be passed to a variant calling stage securely. Further, the plaintext output produced by a bioinformatics tool should be encrypted by the SmartNIC. The variant pipeline can take several hours to execute depending on the size of the input genome. So any unencrypted genomic data on persistent storage creates the risk of being stolen by an adversary. Finally, SVC should achieve good performance to be competitive with variant calling on unencrypted sequences.

B. SVC-Client

The overall architecture of SVC is shown in Figure 1. The client-side module is called SVC-Client, which executes on the user's computer that has a genome sequence. This sequence will be processed by the remote server to compute the raw variants. The main goal of SVC-Client is to (a) securely transmit the genome to the server for variant calling, and (b) securely obtain the VCF file from the server. Only *encrypted data* are sent and received by SVC-Client to avoid attacks over the network. (Note that SmartNICs are not involved in the data transfer of the genomes/VCF files.)

Algorithm 1 shows the steps executed by SVC-Client. The paired-end FASTQ files are first combined to create an interleaved FASTQ file (Line 2) using a tool like SeqFu [66].

Algorithm 1 `SVC-Client`: Main Steps Performed

Input: `F1`, `F2` - plaintext paired-end FASTQ files; m - message size
Output: `output.vcf` - plaintext VCF file

1: **Stage S_1: Interleaved FASTQ construction**
2: Construct interleaved FASTQ file `G` from `F1` and `F2`
3: For `G`, generate a random AES key K_G
4: `IN` ← "`G`"; `OUT`← "`G.enc`" /* Input and output files */
5: Open `IN` for reading; `OUT` for writing
6: **while** !end-of-file of `IN` **do**
7: Read a data block d of m bytes from `IN`
8: Randomly generate $nonce$ /* 96 bits */
9: Let $counter \leftarrow 0$ /* 32 bits */
10: $IV \leftarrow (nonce||counter)$ /* 128 bits */
11: Use AES (i.e., AES-CTR/AES-GCM) to encrypt d with (K_G, IV) and write o/p to `OUT`
12: Write IV to `OUT`
13: Close `IN` and `OUT`
14: Copy `G.enc` to server (e.g., using SCP)
15: Wait for the server to finish variant calling
16: Copy encrypted file `output.vcf.enc` from the server
17: `IN` ← "`output.vcf.enc`"; `OUT` ← "`output.vcf`"
18: Open `IN` for reading; `OUT` for writing
19: **while** !end-of-file of `IN` **do**
20: Read a data block d of m bytes from `IN`
21: Read IV' from `IN`
22: Use AES (i.e., AES-CTR/AES-GCM) to decrypt d with (K, IV') and write o/p to `OUT`
23: Close `IN` and `OUT`
24: **return** `output.vcf`

`SVC-Client` randomly generates an AES key and initialization vector (Line 3). Note that a password-based key derivation function [1] can be used to generate the AES key. For a chosen message size m, each m-byte block of the interleaved FASTQ file is encrypted using AES encryption. For each block, a random initialization vector (IV) is chosen. This ensures that the same key/IV is used only once for encryption, thereby reducing the chance of attacks [40]. Each IV is appended after the encrypted block, and both are written to the output file. The steps are shown in Lines 6-12.

The encrypted file is transmitted to the server (e.g., using SCP). Once the server completes the variant calling pipeline, and the encrypted VCF file (containing raw variants) is transmitted from the server to `SVC-Client`. The VCF file is decrypted using the same AES key. The steps are shown in Lines 19-22.

C. *SVC-Server*

Next, we present the design of the server-side module called `SVC-Server` to overcome the challenges described earlier.

a) Use of Named Pipes: To address the first challenge of avoiding modification to existing bioinformatics tools, `SVC-Server` employs named pipes provided by the underlying OS for inter-process communication. Two processes can exchange data where one serves as the producer and the other serves as the consumer. By setting appropriate file permissions, only authorized user names/processes can read from/write to a named pipe. So existing tools (e.g., BWA-MEM, Picard [5], SAMtools [24], FreeBayes) can now read (plaintext) input from one named pipe (say $Pipe1$) and write (plaintext) output to another named pipe (say $Pipe2$). A SmartNIC must now do two tasks each time any of the aforementioned tools is executed: (a) read the encrypted input file, decrypt it, and write the plaintext output to $Pipe1$; and (b) read from $Pipe2$, encrypt the data, and write to a file. The SmartNIC can have one or more crypto devices. Due to the blocking nature of reading from and writing to named pipes, our initial design made with a single crypto device made it difficult to program correctly as it caused the blocking of a variant calling pipeline stage. Hence, we improved our design to use two crypto devices on the SmartNIC: one will write to $Pipe1$ and the other will read from $Pipe2$.

b) Use of DPDK: We developed a new DPDK module to manage the SmartNIC during variant calling. This module initializes and accesses the crypto devices for encryption/decryption tasks as well as reads from $Pipe2$ and writes data to $Pipe1$. Algorithm 2 shows the initialization steps. The DPDK Environment Abstraction Layer (EAL) is first initialized (Line 1). The aforementioned named pipes $Pipe1$ and $Pipe2$ are created (Line 2). Assuming two Ethernet ports and two crypto devices are needed, each port is mapped to one crypto device. After that a DPDK Worker thread is pinned to a logical core to manage one port/crypto device pair. One named pipe is created for each DPDK Worker to receive input instructions to either encrypt/decrypt, the location from where to read input data, and the location to write output data. (See Lines 3-6). The instruction format is "`IN`⇒`OP`⇒`OUT`". The value of `IN` (location to read) and `OUT` (location to write) can be a regular file or a named pipe. The value of `OP` is either `ENCRYPT` or `DECRYPT`.

Finally, `SVC-Server` prompts the user on standard input to enter the AES key used for encryption to avoid saving the key on storage. Recall that `SVC-Client` uses a random AES key to encrypt a genome.

Algorithm 2 `SVC-Server`: DPDK Initialization

Input: N - number of crypto devices ($N = 2$)

1: Initialize DPDK Environment Abstraction Layer (EAL)
2: Create two named pipes, $Pipe_1$ and $Pipe_2$
3: **for** i in 1 to N **do**
4: Initialize Ethernet port/crypto device pair on the SmartNIC
5: Pin a DPDK Worker thread to a logical core to manage the Ethernet port/crypto device
6: Create a named pipe $Ipipe_i$ for the DPDK Worker to receive instructions during variant calling
7: $K \leftarrow$ Prompt the user to input AES key via standard input

Algorithm 3 shows the steps performed by a DPDK Worker based on the input instructions received. The Worker i first

opens the named pipe $Ipipe_i$, performs a *blocking read* to fetch the instruction from the pipe, and then closes the pipe (Lines 2-4). Therefore, any write to this pipe (with new instructions) will block as the pipe is closed by the Worker. The instruction is parsed to identify the input location, output location, and operation. The input location is read as fixed-size blocks till the end of file is reached. For each m-byte block, if the operation is `ENCRYPT`, then the block is encrypted (using the AES key K and a randomly generated IV) by the assigned crypto device and written to the output location. (See Lines 9-15.) If the command is `DECRYPT`, then each block is decrypted using the crypto device and written to the output location. (See Lines 16-19.) Finally, the input/output locations are closed (Line 20).

Algorithm 3 `SVC-Server`: DPDK Worker

Input: i - ID of DPDK Worker; $IPipe_i$ - Named pipe to read instructions; m - message size

1: **while** True **do**
2: Open $IPipe_i$
3: Read instruction "`IN⇒OP⇒OUT`" ← $IPipe_i$
4: Close $IPipe_i$
5: Let C_i denote the crypto device managed by Worker i
6: Open `IN` for reading, `OUT` for writing
7: **while** !end-of-file of `IN` **do**
8: Read a data block d of m bytes from `IN`
9: **if** `OP` = "ENCRYPT" **then**
10: Randomly generate $nonce$ /* 96 bits */
11: Let $counter \leftarrow 0$ /* 32 bits */
12: $IV \leftarrow (nonce||counter)$ /* 128 bits */
13: Enqueue d to C_i for encryption using AES (i.e., AES-CTR/AES-GCM) with (K, IV)
14: Dequeue encrypted output of d from C_i & write to `OUT`
15: Write IV to `OUT`
16: **else if** `OP` = "DECRYPT" **then**
17: Read IV from `IN`
18: Enqueue d to C_i for decryption using AES (i.e., AES-CTR/AES-GCM) with (K, IV)
19: Dequeue decrypted output of d from C_i & write to `OUT`
20: Close `IN` and `OUT`

c) Variant Calling Pipeline: Before we discuss the entire variant calling pipeline of `SVC-Server`, we first present how named pipes and crypto devices are synergistically used to process a single pipeline stage. (This design enables us to address the third challenge of achieving good performance.) Without loss of generality, let us consider the alignment stage (Stage 2). Figure 2 shows an illustration. The Variant Calling Driver in `SVC-Server` sends one instruction to $Ipipe_1$ to decrypt the input file and another instruction to $Ipipe_2$ to encrypt the output of the alignment stage. The encrypted interleaved FASTQ file is then read by the DPDK Worker managing Crypto device 1. This device decrypts the encrypted blocks using AES and writes plaintext FASTQ data into $Pipe_1$. The unmodified BWA [47] binary reads from $Pipe_1$ (in a blocking manner) and processes the input sequence. The plaintext SAM output produced by BWA is written to $Pipe_2$. The DPDK Worker managing Crypto device 2 reads from $Pipe_2$ in a blocking manner. It encrypts the plaintext blocks using AES and writes ciphertext to a file. Our design is *generic* and handles every pipeline stage the same way. All the files on persistent storage are in ciphertext, thus addressing the second challenge of safeguarding an individual's privacy.

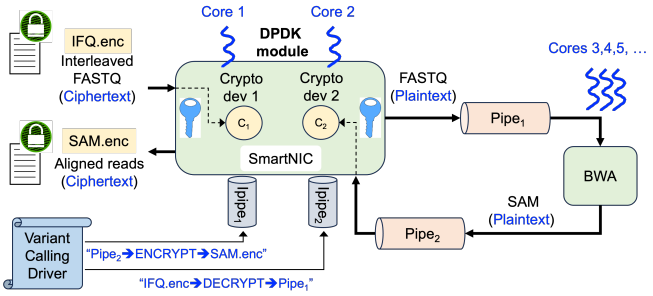

Fig. 2. Generic design using named pipes/crypto devices

The overall steps executed by the Variant Calling Driver are shown in Algorithm 4. The input is an encrypted interleaved FASTQ file sent by `SVC-Client`. The final output is an encrypted VCF file containing raw variants. The unmodified bioinformatics tool for a stage reads data from $Pipe1$ and writes data to $Pipe2$. Stage 2 involving alignment is started by first sending on instruction to $Ipipe_1$ to decrypt the input and another instruction to $Ipipe_2$ to encrypt the output of alignment (Lines 2-3). BWA is executed to read a pipe input and write to a pipe output (Line 4). Stage 3 is executed in two phases: In the first phase, sorting of aligned reads is done. Two instructions are sent to $Ipipe_1$ and $Ipipe_2$ to decrypt/encrypt and then the SAMtools binary is executed for sorting (Lines 6-8). In the second phase, marking duplicates is performed. Once again, two instructions are sent to the named pipes and the Picard binary is executed (Lines 10-12). Lastly, the variant calling method is executed (e.g., FreeBayes) after the instructions to decrypt/encrypt are sent to the pipes (Lines 14-16). The encrypted VCF file can now be transmitted back to `SVC-Client`. (For security reasons, `SVC-Server` can create new named pipes before executing the pipeline and delete them after use.)

IV. Performance Evaluation

In this section, we present the performance evaluation of `SVC`. Specifically, we measured the overhead incurred by `SVC-Server` for the variant calling pipeline on encrypted sequences compared to executing the pipeline on plaintext/unencrypted sequences. We could not compare with tools such as Crypt4GH [61] as this requires code modifications to the bioinformatics tools used by the pipeline.

A. Implementation and Experimental Setup

We used C, DPDK, Python, and Bash for our implementation. The DPDK code was compiled and built using Meson.

Algorithm 4 `SVC-Server`: Variant Calling Driver

Input: $IFQ.enc$ - encrypted interleaved FASTQ file; $Pipe_1$, $Pipe_2$ - named pipes for passing data; $Ipipe_1$, $Ipipe_2$ - named pipes for passing instructions

Output: `output.vcf.enc` - encrypted VCF file

Stage S_2: Alignment
  Send "$IFQ.enc$⇒DECRYPT⇒$Pipe_1$" → $Ipipe_1$
  Send "$Pipe_2$⇒ENCRYPT⇒$SAM.enc$" → $Ipipe_2$
  Run BWA (to align reads) by reading input sequence from $Pipe_1$ and sending output to $Pipe_2$
Stage S_{3a}: Sorting aligned reads
  Send "$SAM.enc$⇒DECRYPT⇒$Pipe_1$" → $Ipipe_1$
  Send "'$Pipe_2$⇒ENCRYPT⇒$BAM.enc$" → $Ipipe_2$
  Using SAMtools, sort the SAM file by reading input aligned reads from $Pipe_1$ and writing the sorted output to $Pipe_2$
Stage S_{3b}: Mark duplicates
  Send "$BAM.enc$⇒DECRYPT⇒$Pipe_1$" → $Ipipe_1$
  Send "$Pipe_2$⇒ENCRYPT⇒$DEDUP.enc$" → $Ipipe_2$
  Using Picard, perform mark duplicates by reading input from $Pipe_1$ and writing output to $Pipe_2$
Stage S_4: Variant calling
  Send "$DEDUP.enc$⇒DECRYPT⇒$Pipe_1$" → $Ipipe_1$
  Send "$Pipe_2$⇒ENCRYPT⇒$VCF.enc$" → $Ipipe_2$
  Run variant caller (e.g., FreeBayes) by reading from $Pipe_1$ and writing output to $Pipe_2$
return `output.vcf.enc`

We conducted all our experiments on FABRIC [21], a testbed for next-generation Internet design and scientific applications. Note that FABRIC is available for academic research at no charge. A slice containing VMs can be created on a site. Hardware accelerators such as SmartNICs can be attached to a slice/VM. We created a slice with one VM containing one NVIDIA Mellanox ConnectX-6 VPI MCX653 dual port (100 Gbps) SmartNIC. The VM had 10 cores, 32 GB RAM, and 100 GB SSD storage running Ubuntu Linux (22.04 LTS). DPDK 23.03, the MLNX_OFED 23.04-1.1.3.0 driver, and Intel Multi-Buffer Crypto Library 1.3 were installed. Note that the chosen SmartNIC on FABRIC supported only software-based crypto operations via DPDK's *cryptodev* library. For the single sample DNA variant calling pipeline, we used BWA-MEM (0.7.17) [47], Picard (2.27.4) [5], SAMtools (1.18) [24], and FreeBayes (1.3.6) [30].

B. Results

We chose three (paired-end) low-coverage genome sequences (of different sizes) from the 1000 Genomes Project [3]. Table II shows the size of the interleaved FASTQ file for each sequence. We used AES-CTR [38] with 128- and 256-bit key sizes for symmetric encryption in `SVC-Server`. We measured the wall-clock time taken by `SVC-Server` to execute different stages of the variant calling pipeline (i.e., S_2, S_3, and S_4) while ensuring all files are encrypted. (Note that stage S_1 was performed by `SVC-Client`.) We also measured the wall-clock time taken to execute the same stages on plaintext input sequence producing plaintext intermediate files and VCF file. (Of course, the SmartNIC and named pipes were not used for plaintext data.) We refer to this baseline approach as `PlaintextVC`. For correctness, we ensured that the raw variants computed by `SVC-Server` and `PlaintextVC` were identical. Table II shows the total time taken by `SVC-Server` and `PlaintextVC` as well as the time taken for different stages. The results demonstrate that `SVC-Server` introduced very little overhead and was slightly slower than `PlaintextVC` (i.e., about 1% slower). For example, on the 15 GB interleaved FASTQ file, `SVC-Server` took 5 h 22 m, whereas `PlaintextVC` required 5 h 19 m. Thus, `SVC`'s unique design is efficient for enabling secure variant calling on sensitive human genomes.

TABLE II
PERFORMANCE COMPARISON

Sequence ID (Interleaved FASTQ size, overhead)	Pipeline Stage	Time Taken (**hh:mm:ss**)		
		`SVC-Server`		`Plain textVC`
		128-bit	256-bit	
ERR062934 (3.8 GB, 0.72% slower)	S_2	0:26:46	0:26:59	0:26:17
	S_3	0:19:33	0:19:08	0:19:16
	S_4	0:24:50	0:25:05	0:25:08
	Total	1:11:09	1:11:12	1:10:41
ERR022463 (8.5 GB, 1.07% slower)	S_2	1:12:36	1:11:21	1:11:53
	S_3	0:57:11	0:58:15	0:56:11
	S_4	0:48:27	0:48:31	0:48:16
	Total	2:58:14	2:58:07	2:56:20
SRR111943 (15.0 GB, 1.0% slower)	S_2	2:26:58	2:27:18	2:26:08
	S_3	1:40:50	1:42:47	1:41:19
	S_4	1:11:50	1:11:47	1:11:14
	Total	5:19:38	5:21:52	5:18:41

V. CONCLUSION

We presented `SVC`, a new technique for secure variant calling on human genomes. `SVC`'s client sends an encrypted interleaved FASTQ file to `SVC`'s server, which executes the variant calling pipeline. Using DPDK, `SVC`'s server leverages a SmartNIC to offload the encrypt/decrypt operations during variant calling. Named pipes are used to communicate between the DPDK module and existing bioinformatics tools without any code modifications to these tools enabling wider adoption. The encrypted file containing raw variants is transmitted back to the client. All files consumed and produced during variant calling are always encrypted on storage rendering them unreadable to an adversary if a data breach occurs. `SVC` achieved competitive performance on different genome sequences with very little overhead due to encryption/decryption. In summary, `SVC` enables users to efficiently and securely perform variant calling on servers such as those managed by cloud providers. The software is available at https://github.com/MU-Data-Science/GAF/SVC.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under Grant No. 2201583. We thank the anonymous reviewers for their suggestions and the FABRIC team.

REFERENCES

[1] The scrypt Password-Based Key Derivation Function. https://datatracker.ietf.org/doc/html/rfc7914.html.

[2] FASTQ Format Specification. https://maq.sourceforge.net/fastq.shtml, 2000.

[3] 1000 Genomes Phase 3 Release. https://www.internationalgenome.org/, 2015.

[4] Privacy in Genomics. https://www.genome.gov/about-genomics/policy-issues/Privacy, 2018.

[5] Picard Toolkit. https://broadinstitute.github.io/picard/, 2019.

[6] The "All of Us" Research Program. *New England Journal of Medicine*, 381(7):668–676, 2019.

[7] Big Data Genomics, 2020. https://github.com/bigdatagenomics/.

[8] The COVID Human Genetic Effort. https://www.covidhge.com/, 2020.

[9] Data Plane Development Kit. https://www.dpdk.org/, 2021.

[10] HIPAA Encryption: What You Should Know. https://compliancy-group.com/hipaa-encryption/, 2021.

[11] National Counterintelligence and Security Center. https://www.dni.gov/files/NCSC/documents/SafeguardingOurFuture, 2021.

[12] Sequence Alignment/Map Format Specification. https://samtools.github.io/hts-specs/SAMv1.pdf, 2021.

[13] The Rise of SmartNICs. https://semiengineering.com/the-rise-of-smartnics/, 2021.

[14] 23andMe Confirms Hackers Stole Ancestry Data on 6.9 Million Users. https://techcrunch.com/2023/12/04/23andme-confirms-hackers-stole-ancestry-data-on-6-9-million-users/, 2023.

[15] GATK4. https://github.com/broadinstitute/gatk, 2023.

[16] GDPR - Encryption. https://gdpr-info.eu/issues/encryption/, 2023.

[17] Genomics on AWS. https://aws.amazon.com/health/genomics/, 2023.

[18] Life Sciences Solutions. https://cloud.google.com/life-sciences-solutions, 2023.

[19] Terra. https://terra.bio/, 2023.

[20] E. Ayday, J. L. Raisaro, U. Hengartner, A. Molyneaux, and J.-P. Hubaux. Privacy-Preserving Processing of Raw Genomic Data. In *Data Privacy Management and Autonomous Spontaneous Security*, pages 133–147. Springer, 2014.

[21] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth. FABRIC: A National-Scale Programmable Experimental Network Infrastructure. *IEEE Internet Computing*, 23(6):38–47, 2019.

[22] A. Caulfield, P. Costa, and M. Ghobadi. Beyond smartnics: Towards a fully programmable cloud: Invited paper. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, 2018.

[23] F. Chen, M. Dow, S. Ding, Y. Lu, X. Jiang, H. Tang, and S. Wang. PREMIX: PRivacy-preserving EstiMation of Individual admiXture. In *AMIA Annual Symposium Proceedings*, volume 2016, pages 1747–1755, 2016.

[24] P. Danecek, J. K. Bonfield, J. Liddle, J. Marshall, V. Ohan, M. O. Pollard, A. Whitwham, T. Keane, S. A. McCarthy, R. M. Davies, and H. Li. Twelve Years of SAMtools and BCFtools. *GigaScience*, 10(2), 2021.

[25] M. Das, K. Shehzad, and P. Rao. Efficient Variant Calling on Human Genome Sequences Using a GPU-Enabled Commodity Cluster. In *Proc. of 32nd ACM Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 3843–3848, 2023.

[26] B. J. Evans and G. P. Jarvik. Impact of HIPAA's Minimum Necessary Standard on Genomic Data Sharing. *Genetics in Medicine*, 20(5):531–535, 2018.

[27] D. Freed, R. Pan, H. Chen, Z. Li, J. Hu, and R. Aldana. DNAscope: High Accuracy Small Variant Calling Using Machine Learning. *bioRxiv*, 2022.

[28] M. Fujiwara, H. Hashimoto, K. Doi, M. Kujiraoka, Y. Tanizawa, Y. Ishida, M. Sasaki, and M. Nagasaki. Secure Secondary Utilization System of Genomic Data Using Quantum Secure Cloud. *Scientific reports*, 12(1):18530, 2022.

[29] GA4GH. The Variant Call Format (VCF) Version 4.2 Specification. https://samtools.github.io/hts-specs/VCFv4.2.pdf, 2021.

[30] E. Garrison and G. Marth. Haplotype-Based Variant Detection from Short-Read Sequencing, 2012. https://github.com/freebayes/freebayes.

[31] S. Goodwin, J. D. McPherson, and W. R. McCombie. Coming of Age: Ten Years of Next-Generation Sequencing Technologies. *Nature Reviews Genetics*, 17(6):333–351, 2016.

[32] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren. SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud. In *Proc. of SIGCOMM 2020*, page 681–693, 2020.

[33] M. Gymrek, A. L. McGuire, D. Golan, E. Halperin, and Y. Erlich. Identifying Personal Genomes by Surname Inference. *Science*, 339(6117):321–324, 2013.

[34] M. Z. Hasan, M. S. R. Mahdi, M. N. Sadat, and N. Mohammed. Secure Count Query on Encrypted Genomic Data. *Journal of biomedical informatics*, 81:41–52, 2018.

[35] D. He, N. A. Furlotte, F. Hormozdiari, J. W. J. Joo, A. Wadia, R. Ostrovsky, A. Sahai, and E. Eskin. Identifying Genetic Relatives Without Compromising Privacy. *Genome research*, 24(4):664–672, 2014.

[36] J. R. Heldenbrand, S. Baheti, M. A. Bockol, T. M. Drucker, S. N. Hart, M. E. Hudson, R. K. Iyer, M. T. Kalmbach, E. W. Klee, E. D. Wieben, et al. Performance Benchmarking of GATK3.8 and GATK4. *BioRxiv*, page 348565, 2018.

[37] M. Hosseini, D. Pratas, and A. J. Pinho. Cryfa: A Secure Encryption Tool for Genomic Data. *Bioinformatics*, 35(1):146–148, 07 2018.

[38] R. Housley. Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP). RFC 3686, Jan. 2004.

[39] L. Kamm, D. Bogdanov, S. Laur, and J. Vilo. A New Way to Protect Privacy in Large-Scale Genome-Wide Association Studies. *Bioinformatics*, 29(7):886–893, 2013.

[40] P. Kampanakis, M. Campagna, E. Crocket, and A. Petcher. Practical Challenges With AES-GCM and The Need for a New Cipher. *Third NIST Workshop on Block Cipher Modes of Operation*, 2023.

[41] D. Kim, S. Lee, and K. Park. A Case for SmartNIC-Accelerated Private Communication. In *Proc.of the 4th APNET Workshop*, page 30–35, 2020.

[42] D. C. Koboldt. Best Practices for Variant Calling in Clinical Sequencing. *Genome Medicine*, 12(1):91, 2020.

[43] C. Kockan, K. Zhu, N. Dokmai, N. Karpov, M. O. Kulekci, D. P. Woodruff, and S. C. Sahinalp. Sketching Algorithms for Genomic Data Analysis and Querying in a Secure Enclave. *Nature Methods*, 17(3):295–301, Mar. 2020.

[44] B. Langmead and A. Nellore. Cloud Computing for Genomic Data Analysis and Collaboration. *Nature Reviews Genetics*, 19(4):208–219, 2018.

[45] K. Lauter, A. López-Alt, and M. Naehrig. Private Computation on Encrypted Genomic Data. In *International Conference on Cryptology and Information Security in Latin America*, pages 3–27. Springer, 2014.

[46] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. M. Swift, and T. V. Lakshman. UNO: Uniflying Host and Smart NIC Offload for Flexible Packet Processing. In *Proc. of the 2017 Symposium on Cloud Computing*, page 506–519, 2017.

[47] H. Li. Aligning Sequence Reads, Clone Sequences and Assembly Contigs With BWA-MEM. *arXiv preprint arXiv:1303.3997*, Mar. 2013.

[48] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta. Offloading Distributed Applications onto SmartNICs Using IPipe. In *Proc. of the ACM SIGCOMM Conference*, page 318–333, 2019.

[49] P. J. McLaren, J. L. Raisaro, M. Aouri, M. Rotger, E. Ayday, I. Bartha, M. B. Delgado, Y. Vallet, H. F. Günthard, M. Cavassini, et al. Privacy-Preserving Genomic Testing in The Clinic: A Model Using HIV Treatment. *Genetics in medicine*, 18(8):814–822, 2016.

[50] S. Miano, R. Doriguzzi-Corin, F. Risso, D. Siracusa, and R. Sommese. Introducing SmartNICs in Server-Based Data Plane Processing: The DDoS Mitigation Use Case. *IEEE Access*, 7:107161–107170, 2019.

[51] NCBI. Genome Reference Consortium Human Build 38, 2013. https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.26.

[52] F. A. Nothaft. *Scalable Systems and Algorithms for Genomic Variant Analysis*. PhD thesis, UC Berkeley, ProQuest, 2017.

[53] F. A. Nothaft, M. Massie, T. Danford, Z. Zhang, U. Laserson, C. Yeksigian, J. Kottalam, A. Ahuja, J. Hammerbacher, M. D. Linderman, M. J. Franklin, A. D. Joseph, and D. A. Patterson. Rethinking Data-Intensive Science Using Scalable Analytics Systems. In *Proc. of the 2015 ACM SIGMOD Conference*, pages 631–646, 2015.

[54] K. A. O'Connell, Z. B. Yosufzai, R. A. Campbell, C. J. Lobb, H. T. Engelken, L. M. Gorrell, T. B. Carlson, J. J. Catana, D. Mikdadi, V. R. Bonazzi, and J. A. Klenk. Accelerating Genomic Workflows Using NVIDIA Parabricks. *BMC Bioinformatics*, 24, 2023.

[55] R. Poplin, P.-C. Chang, D. Alexander, S. Schwartz, T. Colthurst, A. Ku, D. Newburger, J. Dijamco, N. Nguyen, P. T. Afshar, S. Gross, L. Dorfman, C. McLean, and D. Mark. A Universal SNP and Small-Indel

Variant Caller Using Deep Neural Networks. *Nature Biotechnology*, 36(10):983–987, 2018.

[56] P. Rao and A. Zachariah. Enabling Large-Scale Human Genome Sequence Analysis on CloudLab. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–2, 2022.

[57] P. Rao, A. Zachariah, D. Rao, P. Tonellato, W. Warren, and E. Simoes. Accelerating Variant Calling on Human Genomes Using a Commodity Cluster. In *Proc. of 30th ACM Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 3388–3392, 2021.

[58] A. Regalado. China's BGI Says It Can Sequence a Genome for Just $100. *MIT Technology Review, February*, 26:2020, 2020.

[59] K. Scheffler, S. Catreux, T. O'Connell, H. Jo, V. Jain, T. Heyns, J. Yuan, L. Murray, J. Han, and R. Mehio. Somatic Small-Variant Calling Methods in Illumina DRAGEN™ Secondary Analysis. *bioRxiv*, 2023.

[60] H. N. Schuh, W. Liang, M. Liu, J. Nelson, and A. Krishnamurthy. Xenic: SmartNIC-Accelerated Distributed Transactions. In *Proc. of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, page 740–755, 2021.

[61] A. Senf, R. Davies, F. Haziza, J. Marshall, J. Troncoso-Pastoriza, O. Hofmann, and T. M. Keane. Crypt4GH: A File Format Standard Enabling Native Access to Encrypted Data. *Bioinformatics*, 37(17):2753–2754, 02 2021.

[62] Sentieon. Cost-effective and accurate genomics analysis with Sentieon on AWS. 2023. https://www.nature.com/articles/d44224-023-00020-w.

[63] K. Shimizu, K. Nuida, and G. Rätsch. Efficient Privacy-Preserving String Search and an Application in Genomics. *Bioinformatics*, 32(11):1652–1661, 2016.

[64] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson. Big Data: Astronomical or Genomical? *PLOS Biology*, 13(7):1–11, 2015.

[65] K. Taranov, B. Rothenberger, A. Perrig, and T. Hoefler. sRDMA – Efficient NIC-based Authentication and Encryption for Remote Direct Memory Access. In *2020 USENIX Annual Technical Conference*, pages 691–704, 2020.

[66] A. Telatin, P. Fariselli, and G. Birolo. SeqFu: A Suite of Utilities for the Robust and Reproducible Manipulation of Sequence Files. *Bioengineering*, 8(5), 2021.

[67] S. Wang, Y. Zhang, W. Dai, K. Lauter, M. Kim, Y. Tang, H. Xiong, and X. Jiang. HEALER: Homomorphic Computation of ExAct Logistic rEgRession for Secure Rare Disease Variants Analysis in GWAS. *Bioinformatics*, 32(2):211–218, 2016.

[68] W. Xie, M. Kantarcioglu, W. S. Bush, D. Crawford, J. C. Denny, R. Heatherly, and B. A. Malin. SecureMA: Protecting Participant Privacy in Genetic Association Meta-Analysis. *Bioinformatics*, 30(23):3334–3341, 2014.

[69] J. Xing, Y. Qiu, K.-F. Hsu, S. Sui, K. Manaa, O. Shabtai, Y. Piasetzky, M. Kadosh, A. Krishnamurthy, T. S. E. Ng, and A. Chen. Unleashing SmartNIC Packet Processing Performance in P4. In *Proc. of the ACM SIGCOMM 2023 Conference*, page 1028–1042, 2023.

[70] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, pages 1–7, Boston, 2010.

[71] Y. Zhang, M. Blanton, and G. Almashaqbeh. Secure Distributed Genome Analysis for GWAS and Sequence Comparison Computation. *BMC Medical Informatics and Decision Making*, 15(5):S4, Dec. 2015.

[72] J. Zhao, M. Neves, and I. Haque. On the (dis)Advantages of Programmable NICs for Network Security Services. In *2023 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2023.

[73] Y. Zhao, X. Wang, X. Jiang, L. Ohno-Machado, and H. Tang. Choosing Blindly But Wisely: Differentially Private Solicitation of DNA Datasets for Disease Marker Discovery. *In JAMIA*, 22(1):100–108, 2015.