RESEARCH

Open Access

SVAT: Secure outsourcing of variant annotation and genotype aggregation

Miran Kim[1], Su Wang[2], Xiaoqian Jiang[3] and Arif Harmanci[2*]

*Correspondence:
arif.o.harmanci@uth.tmc.edu

[1] Department of Mathematics, Hanyang University, Seoul 04763, Republic of Korea
[2] Center for Precision Health, School of Biomedical Informatics, University of Texas Health Science Center, Houston, TX 77030, USA
[3] Center for Secure Artificial Intelligence For hEalthcare (SAFE), School of Biomedical Informatics, University of Texas Health Science Center, Houston, TX 77030, USA

Abstract

Background: Sequencing of thousands of samples provides genetic variants with allele frequencies spanning a very large spectrum and gives invaluable insight into genetic determinants of diseases. Protecting the genetic privacy of participants is challenging as only a few rare variants can easily re-identify an individual among millions. In certain cases, there are policy barriers against sharing genetic data from indigenous populations and stigmatizing conditions.

Results: We present SVAT, a method for secure outsourcing of variant annotation and aggregation, which are two basic steps in variant interpretation and detection of causal variants. SVAT uses homomorphic encryption to encrypt the data at the client-side. The data always stays encrypted while it is stored, in-transit, and most importantly while it is analyzed. SVAT makes use of a vectorized data representation to convert annotation and aggregation into efficient vectorized operations in a single framework. Also, SVAT utilizes a secure re-encryption approach so that multiple disparate genotype datasets can be combined for federated aggregation and secure computation of allele frequencies on the aggregated dataset.

Conclusions: Overall, SVAT provides a secure, flexible, and practical framework for privacy-aware outsourcing of annotation, filtering, and aggregation of genetic variants. SVAT is publicly available for download from https://github.com/harmancilab/SVAT.

Keywords: Variant annotation, Genomic privacy, Allele frequency

Background

As the cost of next-generation sequencing is decreasing, the number of personal genomes and associated personal information is rapidly increasing. Starting with the initial population-wide genotyping projects such as The HapMap Consortium [1], The 1000 Genomes Project [2], Genomics England [3], The Cancer Genome Atlas (TCGA) [4], Trans-omics for precision medicine (TOPMed) [5], The Genotype-Tissue Expression (GTEx) Project [6], and the Precision Medicine Initiative [7], there are now millions of genomes that are deposited in research, clinical, or recreational database. These genomes can provide great insight for developing new therapies and drugs for diseases, prenatal genetic testing [8], and more advanced methods for disease risk prediction [9]. In particular, the high prevalence of genetic data in clinical, recreational, and research areas makes interpretation and management of genomic data challenging. Among these, genomic privacy [10–14] has recently become an important facet of genomic data sharing because the genetic variants are shown to be strong re-identifiers even from de-identified genomic datasets. The sequencing of millions of samples provides genetic variants with allele frequencies spanning a very large spectrum [5, 15]. This makes it even harder to provide privacy as only a few rare variants among millions can easily re-identify an individual [16, 17]. These risks may extend to the relatives of the individuals [18]. While there is a strong urge to share the data for curing diseases, privacy issues are generally not coherently addressed [19]. Recent advances in the usage of DNA as incriminating forensic evidence to solve high-profile cases bring many new ethical questions that may cause concerns for these individuals [17, 20]. Although there is a great hype for open data sharing, there are also policy barriers to sharing datasets, especially from vulnerable and indigenous populations and datasets related to stigmatizing conditions.

The size of datasets makes it necessary to outsource the analysis and interpretation of genetic variants are often outsourced to 3rd parties such as cloud-based service providers such as AWS, Azure, and Google Cloud [21]. While these services have practically unlimited computational power, data confidentiality is not always guaranteed as cloud instance security is very challenging [22]. For instance, the recent breaches of AWS and Solarwind demonstrated that hackers can perform technically advanced attacks against cloud instances [23]. Industrial standards around "encryption-at-rest" and "encryption-in-transit" are insufficient to protect against IT service compromise.

The field of genomic privacy has grown substantially in recent years. Numerous studies have shown that a small number of variants can lead to re-identification attacks [10, 24, 25] and linking attacks [26–28]. One of the major frameworks that have been successfully applied to genomic data analysis is differential privacy [29, 30] (DP). In DP-based approaches, privacy-enabling data release mechanisms are used to share aggregate statistics from the data. Privacy is enabled by the addition of noise such as Laplacian or Gaussian noise [31]. DP has been used to release private GWAS statistics (chi-squared statistics and minor allele frequencies [29]). A common criticism of DP models is the sacrifice of data utility (especially for high dimensional data), which makes it hard to apply in large-scale genomics analysis. Currently, the encryption-based approaches represent the most rigorous route to securely sharing personal genetic information [32]. There are, however, challenges to their practicality [33]. Methods such as homomorphic encryption [34] provide mathematically provable frameworks for processing encrypted data directly without decryption. Recent studies demonstrated that HE is now potentially practical to perform large-scale genomic computations such as GWAS [35] and genotype imputation [36]. Multiparty computation (MPC) [37] is an alternative cryptographically secure approach, which shares the data into multiple non-colluding entities. The entities communicate intermediate statistics (without leaking any information) with one other in the course of analysis. MPC-based systems may not be practical because they rely on large communication between entities [38]. Several studies developed biomedical data analysis frameworks that combine differential privacy and encryption are proposed [39–42]. These studies demonstrate the practicality of the privacy-enabling data sharing and analysis methods.

In this study, we focus on the secure outsourcing of two tasks, namely variant annotation [43] and aggregation [44–49], which are two basic tasks that are performed in genetic variant analysis pipelines. Annotation is the process of determining the biological impact of mutations that overlap with protein-coding transcripts. Annotation tools assign each variant an impact term from a predefined set of terms (e.g. "non-synonymous", "stop gain") that can be used to select and filter variants for downstream analyses. This is an important task to characterize the impact of a variant on genes. VEP [50] and ANNOVAR [51] (and accompanying webservers) perform variant annotation using gene annotations and variant lists as input [52]. The second task is the aggregation of variants, which is the computation of the variant frequency by counting the existence/allele counts of variants over a large number of individuals. In addition to the annotated impact, the allele frequency provides important insight into a variant's potential role in diseases [48, 53]. It is used extensively in statistical modeling and machine learning models to estimate the functional impact of variants. For example, NIH's recently deployed allele aggregation tool, ALFA, which is recently deployed, provides variant aggregation and AF estimation services [45]. In addition, ExAC [47] and GnomAD [49] provide similar services. Aggregation is also utilized in genomic beacons that answer queries about the existence of variant alleles in different databases. It is, however, not clear how the large genotype databases are stored and secured by these tools. From the collaboration perspective, privacy becomes a major challenge. Although there are great incentives for data sharing and collaboration [54, 55], there is a growing concern around sharing personal data, especially among the underrepresented and vulnerable populations [56]. New legislations are being enacted to protect personal data including summary-level statistics. These may put legal barriers against even simple collaborative analyses that are being routinely performed.

Our approach, named SVAT, makes use of post-quantum cryptography techniques [57] to encrypt the variant data to ensure provable confidentiality against untrusted entities. The data always stays encrypted while it is being analyzed, which ensures that the cloud service cannot snoop on the data. Also, even if the encrypted data is hacked or leaked there are no concerns about privacy. SVAT makes use of a vectorized data representation to convert annotations and aggregations into numeric operations, which can be efficiently computed using Homomorphic Encryption schemes [58]. Due to its efficiency and flexibility, the vectorized representation can be used to build custom pipelines using basic operations such as unions, intersections, and genome-wide statistic estimation while data from multiple collaborating sites are securely processed and integrated.

Utilizing the vectorized representation, we first implemented secure annotation of SNVs and indels. We present a memory/time optimization for indels where the annotation is not explicitly dependent on the variant length. In comparison, SVAT assigns the high impact annotations with high concordance when compared to VEP and ANNOVAR. For the sample-wide aggregation of variants from multiple databases, SVAT utilizes a secure proxy re-encryption approach so that the data can be encrypted as multiple sources are combined.

Overall, SVAT provides a secure and flexible framework for privacy-aware outsourcing of processing, filtering, annotation, and aggregation of genetic variants. SVAT requires moderate-to-high memory and storage resources and incurs moderate runtime requirements.

Results

We first review the secure annotation and aggregation methods that SVAT implements. We next present the comparison with plaintext annotation/aggregation and computational requirements.

Overview of secure variant annotation and aggregation

We present the approaches for annotation (Fig. 1a) and aggregation (Fig. 1b) tasks.

Variant annotation

Figure 1a illustrates the secure variant annotation process. The variant annotation takes the set of target regions (i.e., a BED file), the variant coordinates (e.g., a VCF file), and gene annotations as input and annotates each variant with an impact value, e.g., "splice_acceptor", or "missense_mutation". The target regions are used to filter the variants to the regions of interest such as protein-coding sequences or exons. The gene annotations are input as GFF/GTF files and they are central for variant annotation to describe the exact position of the coding sequences so that the untranslated regions (UTRs), start/end codons, and coding frames can be explicitly identified for each gene and transcript. SVAT utilizes GENCODE gene annotations [59] by default. We exclude the transcripts that are annotated as nonsense-mediated decay and incomplete coding sequence because these transcripts may contain incomplete annotations coming from incomplete evidence.

Selection of target regions

The target regions are used to decrease the amount of data that needs to be submitted between the researcher and the annotation/aggregation server. SVAT utilizes, by default, the protein-coding transcript exons or coding sequences (CDS), which harbor the most consequential phenotype-impacting mutations on the genome (Fig. 2a). In addition, SVAT extends the ends of each exon by a certain length (10 base pairs by default) to include the variants in the introns that may impact splice acceptor/donor motifs. For GENCODE v31, we identified 754,713 protein-coding exons for 83,666 protein-coding transcripts covering 205,443,663 nucleotides. Further, when we exclude the untranslated regions (UTRs), the target regions (with only coding sequences) cover 113,479,849 nucleotides. The exclusion of UTRs is reasonable since UTRs contain variants that are relatively much lower impactful mutations compared to the coding sequences.

These target regions contain all possible alleles for all transcripts. We thus refer to it as "transcript-specific target regions". We observed that there is a substantial number of CDSs in alternatively spliced genes where the frame and start/end coordinates are exactly matching. From a variant annotation point of view, these CDSs provide redundant information because any variant that overlaps one of these CDSs will have the same impact on all the CDSs. To decrease the redundancy, SVAT assesses all the CDSs of each gene and uses the CDSs that are unique when the start/end position and the coding frame are considered. It should be noted that this list

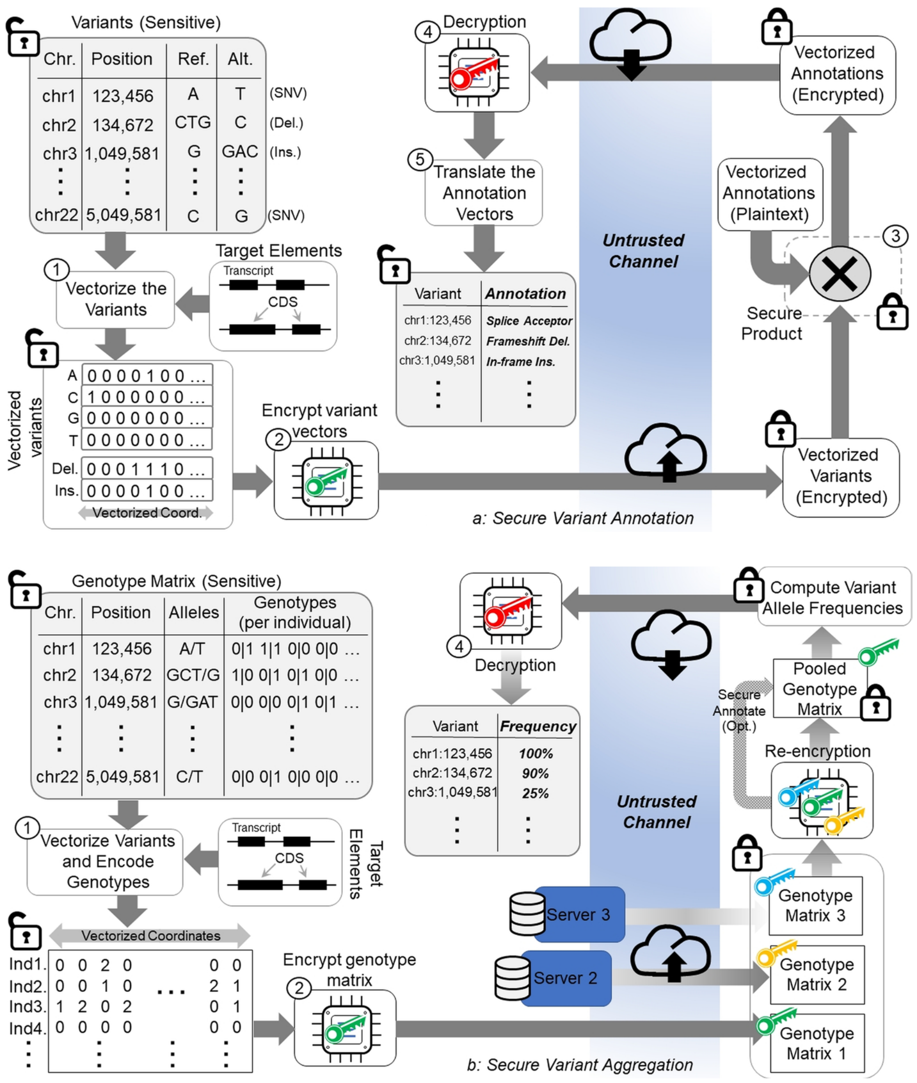
Variants (Sensitive)
Chr. Position Ref. Alt.
chr1 123,456 A T (SNV)
chr2 134,672 CTG C (Del.)
chr3 1,049,581 G GAC (Ins.)
chr22 5,049,581 C G (SNV)
Target Elements
Transcript
CDS
1 Vectorize the Variants
Vectorized variants
Vectorized Coord.
Encrypt variant vectors
2
4 Decryption
5 Translate the Annotation Vectors
Variant Annotation
chr1:123,456 Splice Acceptor
chr2:134,672 Frameshift Del.
chr3:1,049,581 In-frame Ins.
Untrusted Channel
Vectorized Annotations (Encrypted)
Vectorized Annotations (Plaintext)
3
Secure Product
Vectorized Variants (Encrypted)
a: Secure Variant Annotation
Genotype Matrix (Sensitive)
Chr. Position Alleles Genotypes (per individual)
chr1 123,456 A/T 0|1 1|1 0|0 0|0 …
chr2 134,672 GCT/G 1|0 0|1 0|1 0|0 …
chr3 1,049,581 G/GAT 0|0 0|0 0|1 0|1 …
chr22 5,049,581 C/T 0|0 0|1 0|0 0|0 …
1 Vectorize Variants and Encode Genotypes
Target Elements
Vectorized Coordinates
Ind1. Ind2. Ind3. Ind4.
Encrypt genotype matrix
2
4 Decryption
Variant Frequency
chr1:123,456 100%
chr2:134,672 90%
chr3:1,049,581 25%
Untrusted Channel
Compute Variant Allele Frequencies
Pooled Genotype Matrix
Secure Annotate (Opt.)
Re-encryption
Server 3
Server 2
Genotype Matrix 3
Genotype Matrix 2
Genotype Matrix 1
b: Secure Variant Aggregation

Fig. 1 The illustration of annotation, aggregation tasks. **a** The secure annotation task starts by vectorization of the variant loci on the target regions that cover multiple transcripts. The vectorized variant loci signal is encrypted by the researcher's public key and submitted to the annotation server. The server performs secure multiplication of the variant loci signal and the impact signals and generates the annotated vector that is encrypted. The annotated vector is sent back to the researcher. The researcher decrypts the signal using the private key and translates the annotated variants. **b** The secure aggregation task starts by vectorizing the variant loci. Next, the matrices are encrypted by the owners. The encrypted genotype matrices from multiple databases are stored at the aggregation server. When an aggregation task is requested by a researcher, the server re-encrypts the matrices using re-encryption keys so that it can be decrypted by the researcher's private key. The re-encrypted matrices are pooled and securely aggregated to compute the allele frequency at each position on the vectorized positions. The resulting frequency array (encrypted) is sent back the researcher and is decrypted by the researcher

excludes the majority of the untranslated regions (UTRs) since the focus is on CDSs (with an end extension of 10 bp by default). This new set of target regions represents the targets in a gene-specific manner: For each gene, all the CDSs at every possible frame are included. Thus, we refer to these target regions as "gene-specific target regions". The advantage of using gene-specific target regions can decrease the computational requirements. We analyzed the 19,718 protein-coding gene annotations

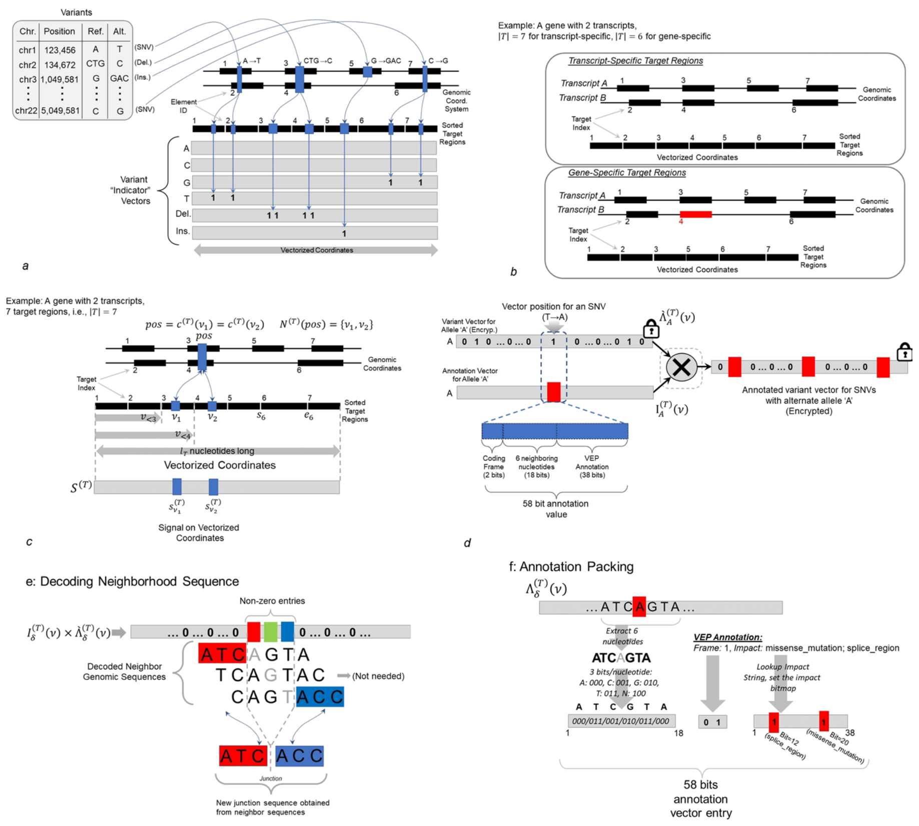

Fig. 2 Illustration of the variant loci vectorization. **a** Example of variant locus vectorization over 7 target regions of 2 transcripts. 4 mutations are shown as they are mapped on the vectorized coordinates. Two SNVs overlap with both transcripts, the corresponding positions on targets 1,2 for 1st SNV and targets 6,7 for 2nd SNV are set to "1" for the vector that is corresponding to the alternate alleles of the SNVs. Similarly, 2 base-pair deletion is mapped to the vectorized positions impacted by the deletion and 2 consecutive entries in the vector are set to "1". For the insertion, the position corresponding to the insertion is set "1". **b** Example of transcript-specific (Top) and gene-specific (Bottom) target regions. The gene-specific regions do not contain one exon that is redundant in terms of variant annotation. **c** Illustration of the notations. **d** Illustration of the multiplication between the impact vector (58-bit annotation value vector) and the variant locus vector. The multiplication is indicated by the cross and the results vector is shown on the right. Any non-zero entry is illustrated by red rectangles. **e** Example decoding of the junction sequence using the neighborhood sequences for a 3-nucleotide deletion. The junction sequence is formed simply by joining the left and right neighborhood sequences of the first and last nucleotides of the deletion. **f** The 58-bit packing of variant annotation

in GENCODE v31 and we identified 259,478 exons that cover 48,041,893 nucleotides in the gene-specific target regions. The gene-specific target regions can be used when the focus is on identifying the most impactful annotation of a mutation among multiple transcripts of a gene, which is the most common way that the mutations are summarized among multiple transcripts (Fig. 2b).

As the vectorized loci must represent every position on every transcript, the genomic coordinates will be redundantly represented on the vectorized loci. This is necessary because we would like to report the impact of any variant on overlapping transcripts which share exons.

Adversarial model

The privacy and confidentiality of the genetic variants and genotypes are treated as sensitive information. We assume that the variant annotations are not explicitly sensitive, and therefore the researcher is not an adversarial entity. In this scenario, the researcher's privacy (the privacy of the participants) needs to be protected against the untrusted annotation server, which we assume does not act maliciously (does not deviate from the protocols and does not collude with other users) but may want to snoop into the contents of the variant and genotype data if they can.

There are two sensitive components for the genetic data: (1) Variant locus and alleles, i.e., chromosome, position, and alternate allele, and (2) the genotypes of the variants for each participant. For the variant annotation task, we assume the variant genotypes are not required and only the variant loci and alleles are necessary. This is reasonable since the impact is generally evaluated with respect to only the position and the alternate allele of the variants. We, therefore, will assume the annotation task requires protection of the variant loci and the alternate allele information.

To protect the variant loci, it is necessary to protect the locations of the variants, for example, by encrypting the chromosome and position values. Next, the encrypted loci can be annotated using, for example, a private (or secure) set intersection protocol, which may induce a high communication cost between the researcher and the annotation server such that the client and the server must be up and responsive to the communication protocols. SVAT utilizes a framework based on homomorphic encryption (HE), which encodes the variants into a vectorized array, then encrypts the sensitive loci and genotype data once at the researcher's computer. The data is encrypted while in transfer and while it is being processed at the annotation server. After the annotation is finished, the researcher receives the annotated data, decrypts it, and translates the vectorized annotation information. We describe these steps below:

Vectorized representation of the variant loci

The vectorization is necessary to protect the variant loci that fall on the target regions. The idea is to enumerate the mutations (i.e., SNVs) (All positions on all transcripts and all alleles) and use the vectorized array for any annotation task. This way, the untrusted entity always receives the encrypted mutational status of all nucleotides (of all alleles) on the target regions regardless of whether there is a mutation or not. Thus, it will not learn anything about the variant loci. The main advantage of the vectorization is that the vector coordinates are not sensitive since they are standardized and do not leak any information (Fig. 2c). Also, as practical homomorphic encryption systems such as Brakerski-Gentry-Vaikuntanathan (BGV) [60], Brakerski/Fan-Vercauteren (BFV) [61, 62], Fully Homomorphic Encryption over the Torus (TFHE) [63], and CKKS [64] support homomorphic operations on encrypted vectors, we only need to use a conventional encryption method of the system and no further elaboration is necessary for homomorphic computation. Thus, the vectorized mutations can be processed using packing and streaming operations to improve performance [33].

We describe the general steps of the vectorization process, which aims at producing a linearly indexed array (i.e., a vector) from the overlapping target regions on the genome (see Methods for details). SVAT first extracts the start/end coordinates of the target regions. Each target region (e.g., a protein-coding exon) is extended (by l_{ext} base pairs) to include variants that may impact splicing. Next, SVAT sorts the extended target regions with respect to first to start positions and then the element ids (e.g., gene/transcript name). Next, the sorted target regions are "stitched" together (i.e., connect the leftmost end of each region to the rightmost end of the region before this region) in the order that they are sorted (Fig. 1a). After stitching, every position on the target regions can be indexed by one index value on the stitched array, which is the final vectorized array. As we have discussed above, the vectorized array contains as many positions as the total number of nucleotides (which we denote by l_T) covered by the target regions. Every position on the vectorized array can be mapped to a unique position on the target regions and vice versa.

A position on the genome can map to multiple positions on the vectorized representation since the genomic position may overlap with multiple target regions, i.e., multiple CDSs of a gene. To map a genomic coordinate to the vectorized coordinates, we first identify the target regions that overlap with the genomic coordinate. Next, the positions on the target regions are mapped to the vector coordinates. Since the target regions are sorted with respect to the start position, the searching of genomic coordinates within sorted target regions can be efficiently performed (Fig. 2b).

Vectorized mutation Loci

Given the target regions T, SVAT allocates a vector of length l_T (indexed by the vectorized coordinates) and stores the variant locus information on the vector. Given a variant allele a ($a \in \{A, C, G, T, \delta, \iota\}$ the nucleotides of SNV, and deletion and insertion (representing 1-base-pair, i.e., 1-bp, deletion, and 1-bp insertion), an array of l_T is allocated, and each array entry whose vectorized position overlaps with a variant is set to 1 (Fig. 2b,c). All other positions are set to 0 (Fig. 2d):

$$\Lambda_a^{(T)}(\nu) = \begin{cases} 1; \ f \ \ \nu \ contains \ alternate \ allele \ a \\ 0; \ \ Otherwise \end{cases}$$

where $\Lambda_a^{(T)}(\nu)$ denotes the variant loci array for allele a. The above equation simply describes that the position ν on the mutation loci array is set to 1 if there exists a mutation with alternate allele a whose genomic coordinate maps to ν. For deletions, a separate variant loci array can be generated for different deletion lengths and set the position where the deletion starts as 1 in the array. Alternatively, SVAT makes use of 1-bp deletion arrays such that each deletion is treated as a sequence of 1-bp deletions and $\Lambda_a^{(T)}(\nu)$ is set to 1 for every position that overlaps with a deletion. This is a more efficient representation because only one variant loci vector is sufficient. For insertions, each variant is described by the position where insertion manifests. The array index that maps to the insertion's coordinate is set to 1.

Encryption of the vectorized variant loci

The researcher (or the data owner) provides the public key for the encryption. Based on the encryption parameter setting of an underlying homomorphic encryption system, we denote l to be the maximal length of a plaintext vector. Then, an array of length l_T is divided into plaintext vectors of size less than l, each of which is encrypted using the public key of the system.

Vectorized annotation information

Similar to the variant locus array, a vector of length l_T stores the annotations of all mutations on the target regions (Fig. 2d). This is very similar to the mutation loci vector, except that for each position ν, the array stores a 58-bit entry that packs the annotation information:

$$\mathrm{I}_a^{(T)}(\nu) = \begin{cases} Impact\ of\ allele\ a\ at\ position\ \nu \\ 0;\ Otherwise \end{cases}$$

The impact value indicates the impact of the mutation located at vector position ν with allele a. The impact information is retrieved from a variant annotation tool, VEP [50], by default. SVAT packs the impact information and the nucleotide information (Fig. 2e) around the mutation locus. As illustrated in Fig. 2f, the packed impact information contains:

1. *Coding frame (2 bit)*: This is the coding frame of a mutation that is a value out of the set {0,1,2}
2. *6-base pair nucleotide neighborhood (18 bits in total)*: SVAT uses 3-bits for each nucleotide and extracts an 18-bit value to encode the 6-base pair vicinity of the mutation, which is the one-codon neighborhood of each position.
3. *The assigned impact string identifier assigned by VEP (38 bits)*: A bitmap (38-bits) that describes the impact values of the mutation out of the 38 impact values that VEP assigns to each mutation. The bitmap is generated by setting the bits to 1 for the indices that correspond to the impact strings assigned by VEP. Table 1 shows the impact strings that SVAT uses.

In order to generate the vectorized annotation, VEP is run to generate the annotation of the mutations on all nucleotide positions of the target regions and all alleles. For deletions, VEP can be either run for multiple deletion lengths, or SVAT can use the 1-bp deletion annotations to build the deletion annotation vector. For insertions, only the impact values for the 1-bp insertion events are stored. Each entry in the annotation vector is allocated and assigned the packed annotation information that contains the corresponding coding frame and VEP annotation string from VEP annotations. Nucleotide neighborhood information is extracted from the hg38 genome sequence:

$$Impact\ Value = \underbrace{(VEP\ Impact\ Index)}_{38\ bits} \Big| \underbrace{(Neighborhood)}_{18\ bits} \Big| \left(\underbrace{Coding\ Frame}_{2\ bits}\right)$$

The values are concatenated and stored in a 64 bits long data type. This is a flexible representation whereby the storage can be extended with more functional information such as SIFT/Polyphen [65] scores and interspecies conservation value [65].

Table 1 VEP Impact terms that impact coding gene regions

Term	CDS	Splice Region	Splice Acceptor	Splice Donor	5′ UTR	3′ UTR	Start Codon	Stop Codon	Intron
intron_variant	0	0	0	0	0	0	0	0	1
3_prime_UTR_variant	0	0	0	0	0	1	0	0	0
5_prime_UTR_variant	0	0	0	0	1	0	0	0	0
coding_sequence_variant	1	0	0	0	0	0	0	0	0
synonymous_variant	1	0	0	0	0	0	0	0	0
stop_retained_variant	0	0	0	0	0	0	0	1	0
start_retained_variant	0	0	0	0	0	0	1	0	0
incomplete_terminal_codon_variant	1	0	0	0	0	0	0	0	0
splice_region_variant	0	1	0	0	0	0	0	0	0
protein_altering_variant	0	0	0	0	0	0	0	0	0
missense_variant	1	0	0	0	0	0	0	0	0
inframe_deletion	1	0	0	0	0	0	0	0	0
inframe_insertion	1	0	0	0	0	0	0	0	0
transcript_amplification	0	0	0	0	0	0	0	0	0
start_lost	1	0	0	0	0	0	1	0	0
stop_lost	1	0	0	0	0	0	0	1	0
frameshift_variant	1	0	0	0	0	0	0	0	0
stop_gained	0	0	0	0	0	0	0	1	0
splice_donor_variant	0	0	0	1	0	0	0	0	0
splice_acceptor_variant	0	0	1	0	0	0	0	0	0

The first column shows the impact term. Each cell with '1' indicates the gene element that is impacted. The full set of impact terms is provided in Supplementary Table 1 in Additional file 1

Secure SNV variant annotation

In the secure annotation scenario, the researcher generates the vectorized mutation array, encrypts it, and sends the encrypted data to the annotation server. The server generates the vector annotation signal as described above. The annotation of the variants is performed by multiplication of the mutation vector and the annotation vector (Fig. 2d). In the resulting vector, the entries for which there is an SNV are set to 0, and others are set to the annotation value. The server performs the multiplication over encryption and sends the results back to the user. For each position on the array that is non-zero, a variant exists. The client can use streaming operations to decrypt the downloaded data and filter out non-zero annotation entries (64-bit packed annotation information), which are unpacked according to the bit packing above. The conversion of the vector coordinates to genomic coordinates is performed while looping over the vectorized coordinates.

Secure small deletion variant annotation

Annotation of deletion variants is more complex than SNV annotations since deletions can have variable lengths. One way to approach this is to annotate all deletions up to a certain length and store annotations in a vectorized annotation array for each deletion length. It is straightforward to implement this approach with the current vectorized annotation signal framework.

Rather than keeping the annotation vector for each deletion length, SVAT can also make use of the 1-base pair deletion impact signal to build the variant annotation: For deletion of nucleotides $[a, b]$ $(a < b)$, SVAT loops over all nucleotides and merges the impacts of 1-base pair deletion at nucleotide ν in $[a, b]$. While merging, SVAT keeps track of the impact on coding sequence by counting the number of nucleotides that are deleted from CDSs. In the case that any 1-bp deletion impacts a start/stop codon, SVAT uses the 18-bit nucleotide information (coding frame) and extracts the 6-bp neighborhood to assess start/stop gain/ retainment (Fig. 2f). This is also done for the splice loss events. All retainment and loss events are annotated with the assumption that mRNAs are read from 5' to 3' while in translation, i.e., the reading frames are disrupted towards the 3' end of the mature mRNA sequence after deletion and insertion events [66]. The current framework that SVAT utilizes is flexible enough to accommodate longer neighborhood sizes so regions larger than 1-codon neighborhood can be evaluated while annotating deletions.

As in secure SNV annotations, the researcher generates the vectorized 1-base pair deletion vector for all the deletion mutations that will be annotated. In this vector, any nucleotide that overlaps with deletion is set to 1 and other nucleotides are set to 0, this way the vector $\Lambda_{\delta}^{(T)}(\nu)$ represents the deletion state of all nucleotides. These are encrypted and sent to the annotation server. The annotation server generates the vectorized annotation signal for all the 1-base pair deletions on the target regions. For this, the server annotates all the 1-base pair deletions on the target regions using VEP, then packs the impact values into the 64-bit impact array. The impact array $(I_{Del}^{(T)}(\nu))$ is then securely multiplied with the encrypted 1-base pair deletion vector, i.e., $I_{\delta}^{(T)}(\nu) \times \Lambda'\delta^{(T)}(\nu)$. In the multiplication, any position that does not overlap with any deletion is set to 0 and other positions that overlap with a deletion contain the 64-bit packed annotation value of the 1-base pair deletion at the position. Upon receipt of the results, the researcher decrypts the annotation vector. For every consecutive position with non-zero entries, the server tracks the deletion and sets the annotation as described above.

Secure small insertion variant annotation

Unlike a deletion, an insertion occurs at a single location and is described by the sequence that is inserted at the location. After receiving the product of the impact vector and the encrypted insertion loci vector, $I_{\iota}^{(T)}(\nu) \times \Lambda^{'}\iota^{(T)}(\nu)$, the array is decrypted. After the positions with non-zero entries are mapped back to the insertions. For this, SVAT again utilizes the 6-base pair neighborhood sequence and the coding frame at position ν and builds the inserted junction sequence at the insertion site. This information is used to translate the codons that are inserted and the final impact on CDS, stop/start codons, and splice sites are reported.

Genotype aggregation

The genotype aggregation aims to compute the frequencies of the mutations by aggregating over many samples. $f_a^{(T)}(\nu) = \sum_k G_{a,k}^{(T)}(\nu)$, where n_G denotes the number of individuals in the genotype matrix, $f_a^{(T)}(\nu)$ denotes the frequency of the allele a for the variant at vector coordinate v (indexed on the target regions T), and $G_{a,k}^{(T)}(v)$ indicates the genotype of the variant for individual k. The entries in the genotype matrix can hold the number of alternate alleles or just the existence of mutations. Aggregation of the alternate allele counts provides the allele frequency of a within the sample set, as ExAC database provides. Aggregation of variant existence provides the number of samples with the mutation, i.e., similar to genomic beacons [67, 68].

While aggregating the genotypes, it is necessary to ensure the confidentiality of the genotype matrix $G_{a,k}^{(T)}$ from the untrusted aggregation server, which performs the computationally heavy task of secure aggregation. To compute the above summation securely, the genotype matrix is encrypted, denoted by $G_{a,k}^{'(T)}$ and the summation is evaluated using HE-based secure summation. Another important aspect of aggregation is to accommodate genotype matrices from multiple databases so that many samples can be aggregated together. The server needs to be able to manage multiple datasets that are encrypted with different keys. SVAT implements a proxy re-encryption protocol to convert the genotype matrices into the same key and perform the aggregation using this common key.

SVAT utilizes proxy re-encryption to securely re-code the genotype matrices (or any other type of data) so that they can be decrypted with the same secret key [69]. A trusted entity (such as NIH) is required who will perform the key management to manage the private keys necessary to generate the re-encryption keys. This is a reasonable assumption since the sensitive datasets are generally deployed and protected by entities such as NIH (e.g. database of genotypes and phenotypes – dbGAP). When a researcher requests the aggregation service by aggregating M genotype matrices, the request is sent to the trusted entity, e.g. NIH., The genotypes matrices are all encrypted with different keys (as they are from different sources). The trusted entity first generates a public–private key pair and a corresponding re-encryption key for each of the matrices. The re-encryption keys are sent to the aggregation server, which re-encrypts all of the M genotype matrices decryptable with the same private key. Here, the aggregation server uses only the public re-encryption keys without any knowledge of the private keys. After the genotype matrices are re-encrypted, the aggregation is performed by the secure summation of the encrypted genotype matrices at every position on the target regions. The resulting frequency array, which is encrypted with the researcher's public key, $f'a^{(T)}(v)$, is sent to the researcher who can decrypt the frequency array and obtain frequencies.

If genotype matrices of multiple data providers are encrypted under different keys, then the server wishes to perform computation on such multi-key ciphertexts. One way to approach this is to use multi-key homomorphic encryption [70]. But the framework is still computationally intensive, making the system unsuitable for deployment in practice. Instead, we can alleviate the computational burden by the use of the conventional key switching approach of homomorphic encryption. Suppose that the key management party generates a switching key from a secret key of the input data to the common secret key and the re-encryption key is shared with the server. Then it enables the server to convert ciphertexts of genotype matrices to ciphertext decryptable with the secret key without decrypting ciphertexts.

SNV aggregations

The secure aggregation of the SNVs is straightforward as they are located at one position on the vectors. In other words, the SNV frequency aggregation can be computed simply by marginalizing at every location.

Indel aggregations

Unlike SNVs, the deletions cover must be tracked in each sample and aggregated. As with annotation, we make use of the 1-base pair deletions to build the aggregation of deletion of multiple nucleotides. Given a position ν, and the 1-base pair deletion genotype matrix, $G^{(T)}_{\delta,k}(\nu)$; the indel of length l_δ is aggregated by counting the individuals k that satisfy the following: The deletion state of all base pairs in $[\nu, \nu + l_\delta]$ are set to 1 and that the entries at $(\nu - 1)$ and $(\nu + l_\delta + 1)$ are set to 0. This way, we make sure that the deletion spans exactly the coordinates in $[\nu, \nu + l_\delta]$. This procedure can effectively aggregate all the deletions that are engulfed in the target regions.

The aggregation of insertions requires explicit matching of the inserted nucleotides. This requires enumeration of all possible insertions. SVAT currently does not explicitly support aggregation of short insertion variants. However, the position at which the insertion happens can be aggregated (just by simple aggregation as for SNVs) to compute the frequency of insertion at each position.

Simplifications and extensions It should be noted that as NIH is trusted, the researcher's keys can be generated by the NIH, too, which would minimize the computational load on the researcher, who can receive the data directly from the aggregation server. Secondly, the aggregation server does not have to store the encrypted genotype matrices. They can be discarded after the frequency vector is generated once for all the alleles and deletion lengths of interest. The above aggregation formulations can be expanded to incorporate the variant call qualities as it is done in GnomAD [49].

Comparisons with VEP annotations

To compare SVAT's secure annotations with the plaintext VEP annotations, we simulated mutations on protein-coding genes. For this, we focused on the 10 megabase region on chromosome 1 (40mbase-50mbase) of hg38 assembly and simulated the 3 types of variants (SNVs, deletions, insertions). This region contains 6,996 exons of the 1,152 transcripts, which are used as the target regions after 100 base pair extensions. We excluded the transcripts that are tagged with "incomplete_terminal_codon_variant" and "NMD_transcript" tags. In total, the target regions cover 3,409,574 nucleotides. For all variant types, we used a 25% probability of introducing a mutation.

Comparison of SNV annotations

SNVs are simulated by replacing the reference nucleotide with another nucleotide such that each non-reference allele is selected randomly with equal probability, i.e., $p_{alt} = \frac{1}{3}$. With a per-position SNV probability of 0.25, we generated 853,756 SNVs. Each mutation is annotated with VEP to generate the baseline annotations. We next annotated the simulated SNVs using SVAT. We finally compared the annotations in terms of matches between assigned impact strings for each variant. As expected, VEP and SVAT have yielded the same annotations.

Comparison of deletions

We described 2 approaches by which deletions can be annotated. The first approach is using deletion-length specific annotation vectors (and corresponding mutation vectors). Similar to SNV annotations, this approach will enable us to exactly replicate the VEP annotations as it explicitly encodes the impact strings into the annotation vector for deletions of every length.

The second approach uses the 1-bp deletion annotation vector and 1-bp deletion variant loci vectors. As described earlier, the impact of each deletion whose length is greater than 1-bp needs to be translated using the 1-bp deletion annotations. The impact values assigned by SVAT may, therefore, not perfectly match VEP annotations. To evaluate the mismatches systematically, we simulated 853,145 deletion variants whose lengths are randomly selected (uniformly distributed with $[1, 10]$). The simulated deletions are annotated using VEP. Then we simulated the secure annotation by SVAT where the 1-bp deletion annotation signal is multiplied with the 1-bp variant loci vector and the product vector (i.e., annotated vector) is translated to generate the annotation for all the deletions.

While comparing the annotations, we first focused on the 6 impact values that are assigned by VEP whose impact strings are classified with the HIGH impact category. These are (1) frameshift_variant, (2) splice_acceptor_variant, (3) splice_donor_variant, (4) stop_gained, (5) stop_lost, (6) start_lost. Next, we compared the high-impact annotations that are assigned to the variants by VEP and by SVAT. For each high-impact annotation assigned by SVAT (VEP), we counted the frequency of mismatching annotations assigned by VEP (SVAT).

Mismatch of HIGH impact category

We first counted the number of mismatches in the HIGH impact category where we found the number of annotations where SVAT and VEP did not assign a high impact annotation. We found that out of 2.7 million matching annotations (Per variant and transcript), 2,068 (Less than 0.1%) annotations contain a mismatch where SVAT or VEP assigned one of the HIGH impact annotations while the other did not. We next analyzed the mismatches for each HIGH impact category. Figure 3 shows the comparison of high-impacting terms. We describe the mismatching annotations and justify the annotations assigned by SVAT. We believe this comparison is reasonable since annotation of indels is not definitive and different methods annotate indels differently [71].

SVAT Specific frameshift annotations (missing in VEP annotations)

The most frequent mismatch in annotations are variants that are annotated as splice_acceptor or splice_donor by VEP whereas SVAT assigns them as frameshift mutation, in addition to splice_acceptor or splice_donor. Upon inspection, we found that VEP's annotations are disjoint for these classes, i.e., splice_acceptor and splice_donor annotations do not overlap with frameshift annotations.

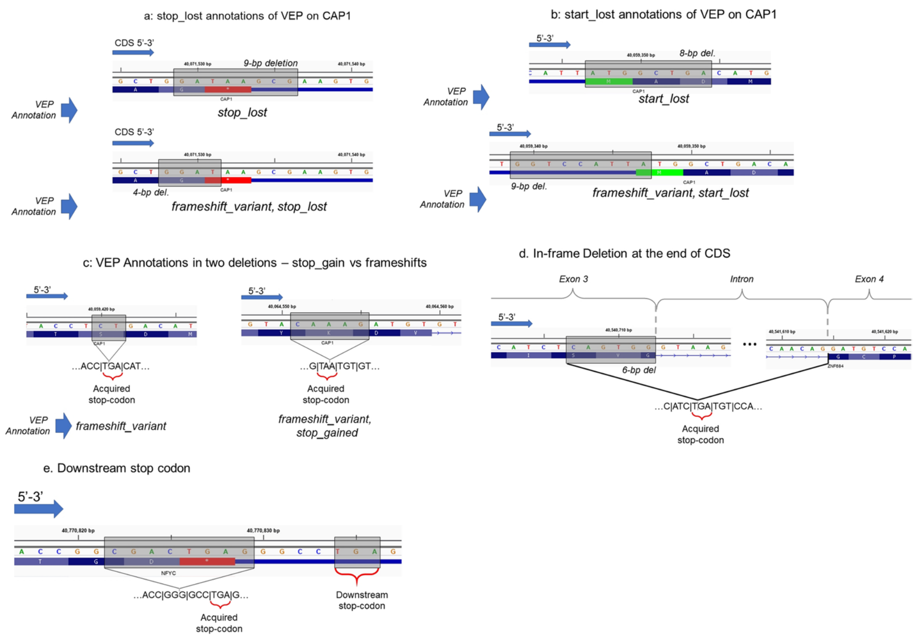

Fig. 3 Examples of mismatches between SVAT and VEP while annotating deletions. **a** Two examples of deletions that impact CAP1 gene as they are annotated by VEP. 9-bp deletion that engulfs the stop-codon (shown in red) are annotated by VEP. This variant is annotated by VEP only as "stop_lost". On the other hand, the 4-bp deletion that deletes only 1 base pair from the stop-codon is annotated as "frame_shift, stop_lost". **b** Similar to (**a**), the start-codon impacting variants are annotated by VEP as "start_lost" for an 8-bp deletion (top) and as "start_lost, frameshift" for a 9-bp deletion (bottom). VEP does not assign "frame_shift" term when the start-codon is engulfed in the variant. In both of these cases, SVAT assigns both frame_shift and start_lost terms. **c** Examples of two deletions that are discordantly annotated by SVAT and VEP. For both deletions, SVAT annotates by "frameshift_variant, stop_gain" whereas VEP annotates the 2-bp deletion (left) as frameshift and the 5-bp deletion (right) as "frameshift_variant, stop_gain". 2-bp deletion (left) also causes a stop-gain at the junction where frame contains a "TGA" stop-codon. This is detected by SVAT while VEP does not report the stop-gain. It should, however, be noted that these deletions are both in HIGH impact class and would be picked up as candidates in downstream analyses. **d** An example of a stop_gain that is detected by VEP and missed by SVAT. The deletion that ends right at the splice junction site creates a junction with a stop codon. Both mutations are in HIGH impact class as this is also a frameshift mutation. **e** Another example of a downstream stop-codon that is missed by SVAT and picked up by VEP. The 8-bp deletion engulfs stop-codon and creates a new frame such that a downstream stop-codon ("TGA" stop codons as shown in figure) is created within the new coding frame. This mutation is marked as HIGH-impact by VEP and by SVAT

The stop and start loss are the two other mismatching VEP annotations where SVAT additionally assigns frameshift annotations. These mutations are also annotated with stop and start loss impacts by SVAT, in addition to frameshift. We also observed that VEP annotates start and stop losses as frameshift only when the deletion is exclusively in the coding sequence and impacts a start or a stop codon by 1 or 2 nucleotides (Fig. 3a). Thus, SVAT performs a more complete annotation of these mutation classes.

Similarly, numerous mutations SVAT annotates with frameshift and stop_retained are annotated only as stop_retained by VEP. We inspected several of these mutations and found that Annovar annotates these mutations as frameshift as well. A small fraction of these mutations is found to be downstream stop codons that VEP traces. For these mutations, SVAT does not assign stop_retained impact. This is because SVAT analyzes only one codon that the deletion impacts. This is one of the current limitations of the default parameters of SVAT that uses a three-base pair neighborhood sequence and can be mediated by increasing the neighborhood size.

We also would like to highlight the cases of start_retained and start_lost impact classes. VEP annotates some of start_lost mutations with start_retained. We chose to exclude the start_retained variants exclusively because the start codon is surrounded upstream by conserved binding motifs that are used for translation initiation, such as the Kozak motifs [72]. This does not apply to stop_retained class since stop codons do not require surrounding sequence motifs to terminate translation, unlike start codons (Fig. 3b).

SVAT-specific stop-gain annotations

We next focused on SVAT-specific stop-gain annotations. Among these, most of them are annotated as frameshift by VEP. All of these mutations are also annotated as frameshift with SVAT. While we inspected numerous cases, we could not identify a specific reason why VEP chooses not to annotate these variants as stop-gain (Fig. 3c). We also found that VEP does not assign stop-gain to some of the inframe deletion mutations that seem to introduce early stop-gains in the coding sequence. Previous studies have pointed out the discrepancies among annotation tools regarding the assignment of stop-gain mutations [71].

SVAT-specific stop-lost annotations

We observed that majority of the mutations that are annotated as stop-lost only by SVAT are annotated as stop-retained by VEP. These are annotated as stop-retained by SVAT as well (in addition to stop-lost). Upon inspection, we found that the main reason for this discrepancy is that VEP assigns stop-lost and stop-retained annotations disjointly.

VEP-specific stop-gain annotations

Some mutations manifest at the splice sites such that the deletions at the end of CDS create a stop-codon on the processed mRNA sequence after splicing. This case is currently not handled by SVAT yet and represents another limitation (in addition to the tracking of the downstream stop-gains) of SVAT (Fig. 3d,e). This case can be handled by keeping track of the neighborhood coding sequence, in addition to the genomic neighborhood.

Comparison of insertions

For comparison of insertions, we performed a simulation with a 25% chance of having an insertion at any position. For any insertion, the length is selected uniformly between 1 and 10 base pairs, and inserted sequence is generated as a random string of {A, C, G, T} nucleotides. We generated 854,639 variants on the target and annotated them with SVAT and VEP. It should be noted that the translation phase of insertions only requires translating the inserted sequence and computing the frame. Out of the 2,352,611 annotations that are matching in variant and transcript among VEP and SVAT, we identified only 246 annotations exhibit a mismatch in the high impact category, i.e., either one of the methods provides a HIGH impact annotation. When we analyzed each impact string (as for deletions), we observed a similar pattern where SVAT-specific stop-gain and stop-loss annotations are matched by frameshift and stop-retained annotations that are assigned by VEP, as with deletion cases. The stop-gain that manifests by the insertions at the ends of exons is again missed by SVAT (as expected) (Fig. 3d,e).

Overall, these results indicate that SVAT has comparable and more exclusive annotations compared to VEP. We do not provide these results as SVAT performing more accurately than VEP because SVAT is based on VEP annotations. In addition, the default parameters of SVAT may exhibit several limitations for a small number of events, namely (1) downstream stop-codons and (2) indels at the end of CDSs.

Comparison with ANNOVAR annotations

We next compared the variant annotations from SVAT with ANNOVAR [51], one of the most popular tools for variant annotation. ANNOVAR generates annotations for each variant at the gene-level, where each variant is assigned a single annotation term over multiple transcripts. The reported annotation corresponds to the most damaging impact over all transcripts of the gene. This is different from the transcript-level reporting strategy of SVAT (and VEP) where each transcript is independently annotated in a separate line in the output. Because many transcripts share exons at the same coding frame, this creates redundancy in the final report, but it is also systematically more complete because each transcript can be easily extracted from the output with the impact string.

Because of these differences, we focused on a gene-level comparison of SVAT and ANNOVAR rather than an annotation-level comparison that we performed with VEP. For each variant, we pooled the annotation terms (e.g. "inframe_deletion, missense_variant") assigned by SVAT to all of the transcripts. Next, we overlapped the variants by their identifiers between SVAT and ANNOVAR outputs and evaluated the high-impact annotations that are exclusively assigned by SVAT and ANNOVAR. Below we summarize these differences. It is worth noting that previous studies have performed comparisons of VEP and ANNOVAR and these methods have been found to produce slightly discordant results [73].

Comparison of terms assigned by SVAT and ANNOVAR

SVAT and ANNOVAR use different annotation terms for reporting variant impact. It should also be noted that SVAT (similar to VEP) provides more granular annotation information. For example, SVAT separates splice variants into acceptor/donor variants and also variants that are close to splicing motifs as "splice_region" (within 1–3 bases of

Table 2 The matching between the impact terms between ANNOVAR and SVAT

ANNOVAR impact term	SVAT impact term
Frameshift insertion/frameshift deletion	frameshift_variant
Nonframeshift insertion/nonframeshift deletion	inframe_insertion / inframe_deletion
Splicing	splice_acceptor_variant, splice_donor_variant
Startloss	start_lost
Stopgain	stop_gained
Stoploss	stop_lost
Nonsynonymous SNV	missense_variant
Synonymous SNV	synonymous_variant

Each row shows the matching terms that best correspond to the impact implied by the corresponding method the exon or 3–8 bases of the intron). However, ANNOVAR did not make a distinction between splice acceptor and donor sites. For each annotation by ANNOVAR, we concatenated the terms assigned to the splicing (column 6) and protein sequence (column 9) terms to build the collective list of impact strings assigned by ANNOVAR. To match the assigned terms, we assessed each impact term assigned by ANNOVAR and identified the best matching term assigned by SVAT. The matching terms of SVAT and ANNOVAR are shown in Table 2.

Single nucleotide variants

We used the simulated SNVs from our previous comparison and compared their variant-level annotations as described above. Out of 853,756 simulated SNVs, we found differences in annotations of 429 variants on 3 genes. When we studied these in further detail, we found that the discordant annotations are caused by differences in the annotations of these genes. Overall, this comparison indicates a very high concordance between SNV annotations of SVAT and ANNOVAR.

Deletions

We next compared the annotations of simulated deletions between SVAT and ANNOVAR. Over the 853,146 simulated deletions, we found 148 variants were annotated with a high-impact term exclusively by one method. These variants are predominantly located on genes whose annotations are changed between the annotations used by SVAT and ANNOVAR. The assignment of high-impact annotations of SVAT and ANNOVAR exhibit very high concordance for deletions.

We next compared the more detailed annotation differences between ANNOVAR and SVAT among the high-impact variant annotations. For this, we focused on the high-impact terms that were discordantly annotated by the two methods.

Deletions annotated with "frameshift_variant" by SVAT, and "stop-gain" by ANNOVAR The most frequent difference was 2,924 deletions assigned as frameshift deletions by SVAT and stop-gain by ANNOVAR. Assessing these further, we observed that these deletions are generated by a gain of a stop-codon that is downstream of a frameshift deletion (Fig. 4a). Thus, both methods are correct by assigning a high-impact annotation, and ANNOVAR provides more information by reporting a downstream stop-codon.

Deletions annotated as "splice_donor_variant" by SVAT exclusively We found that 475 deletions were exclusively annotated as splice donor impacting deletions by SVAT and were not annotated by ANNOVAR by a high impact annotation term. These deletions reside in the spliced 5' or 3' UTRs without any impact on the coding sequences of the transcripts (Fig. 4b). It is not clear if these mutations should be classified as high-impact mutations because there is no impact on the coding sequence. However, it is known that UTRs harbor regulatory sequences, and changes in the appropriate splicing patterns may regulate, for example, RNA processing [74]. Notably, VEP assigns HIGH_IMPACT to these deletions. The appropriate classification of these variants requires further research into their prevalence and potential association with gene expression or phenotypes.

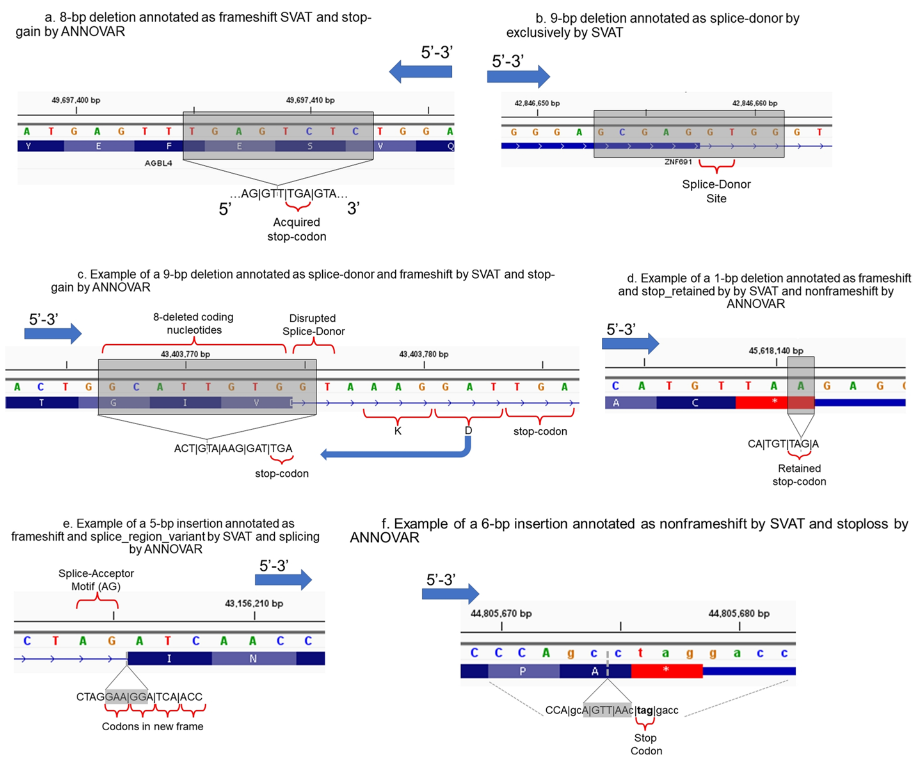
a. 8-bp deletion annotated as frameshift SVAT and stop-gain by ANNOVAR
b. 9-bp deletion annotated as splice-donor by exclusively by SVAT
c. Example of a 9-bp deletion annotated as splice-donor and frameshift by SVAT and stop-gain by ANNOVAR
d. Example of a 1-bp deletion annotated as frameshift and stop_retained by by SVAT and nonframeshift by ANNOVAR
e. Example of a 5-bp insertion annotated as frameshift and splice_region_variant by SVAT and splicing by ANNOVAR
f. Example of a 6-bp insertion annotated as nonframeshift by SVAT and stoploss by ANNOVAR
5'-3'
...AG|GTT|TGA|GTA...
Acquired stop-codon
Splice-Donor Site
8-deleted coding nucleotides
Disrupted Splice-Donor
ACT|GTA|AAG|GAT|TGA
stop-codon
CA|TGT|TAG|A
Retained stop-codon
Splice-Acceptor Motif (AG)
CTAGGAA|GGA|TCA|ACC
Codons in new frame
CCA|gcA|GTT|AAc|tag|gacc
Stop Codon

Fig. 4 **a** 8-bp deletion annotated as frameshift SVAT and stop-gain by ANNOVAR **b** 9-bp deletion annotated as splice-donor by exclusively by SVAT **c** Example of 9-bp deletion annotated as splice-donor and frameshift by SVAT and stop-gain by ANNOVAR **d** Example of a 1-bp deleltion annotated as frameshift and stop_retained by SVAT and nonframeshift by ANNOVAR

Deletions annotated with "frameshift_variant, splice_donor_variant" by SVAT and "stop-gain" by ANNOVAR We found 184 deletions that were annotated in this category. Similar to the previous case, we observed that these mutations are frame-shift mutations that create a downstream stop-codon right after the deletion. In addition, these deletions impact the splice donor site. ANNOVAR did not report a splicing annotation for these deletions (Fig. 4c).

Deletions annotated as "frameshift_variant;stop_lost;stop_retained" by SVAT and "non-frameshift" by ANNOVAR We found 160 deletions where a deletion impacted the stop codon and this codon was compensated by the downstream sequence in the 3' UTR or intergenic region. These deletions were annotated as non-frameshift by ANNOVAR. We believe that the description of SVAT is more informative as it describes the loss of the original stop codon and retention of the new stop codon albeit with redundancy (Fig. 4d).

Deletions annotated with "splice_acceptor_variant" exclusively by SVAT.

We found 103 deletions that impacted the splice acceptor sites in UTR without impact on coding sequences. Similar to the case of splice donor variants, these deletions impact the non-coding UTR region of the genes.

Insertions

We next compared the high-impact terms assigned to 854,639 simulated insertion variants by SVAT and ANNOVAR using the variant-level comparison.

Insertions annotated with "frameshift_variant;splice_region_variant" by SVAT and "splicing" by ANNOVAR The most common mismatches among annotations of insertion variants are in this class (1,250 insertions). These insertions are found to be right at the splice junction of the coding exons between the canonical splicing motif and the coding sequence. These would therefore not impact the splicing motif (Fig. 4e). We expect that they would only extend the coding sequence by the length of the insertion. The lengths of the insertions in this category were not a multiple of 3-base pairs, which indicates that they will likely cause a frameshift in the translation. We, therefore, conclude that SVAT's (and VEP's) frameshift is a slightly more appropriate annotation for these mutations compared to a splicing annotation. It should, however, be noted that there can be unknown motifs within the exon that regulates the splicing, and therefore splicing may be impacted by these insertions. We, therefore, refrain from indicating that ANNOVAR incorrectly annotated these variants.

Insertions annotated with "inframe_insertion;splice_region_variant" by SVAT and "splicing" by ANNOVAR Similar to the frameshift insertion above, we found 480 insertions whose lengths are exact multiples of 3-base pairs were annotated as primarily by splicing by ANNOVAR.

Insertions annotated with "frameshift_variant" by SVAT and "stopgain" by ANNOVAR 367 insertions were found to cause a frameshift coupled with the manifestation of a downstream stop codon in the new coding frame. While SVAT does not report the stopgain for these insertions, the high-impact annotation is reported which can provide be used to select these variants in the downstream analyses.

Insertions annotated with "frameshift_variant" by SVAT and "stoploss" by ANNOVAR We found 349 CDS overlapping insertions, whose lengths were not multiple of 3-base pairs causing frame shift in translation. ANNOVAR assigned these variants exclusively to stoploss annotation. In general, a frameshift should be expected to create a loss of stop codon and both methods imply a similar annotation on the effect.

Insertions annotated with "inframe_insertion" by SVAT and "stoploss" by ANNOVAR 145 insertions whose lengths were multiple of 3 base pairs were found to be right before the stop codons and were annotated differently by SVAT and ANNOVAR (Fig. 4f). As such, these insertions would only extend the transcript by a multiple of 3 base pairs and should not impact the stop codons. It is viable that the motifs around the stop codons may be disrupted by the insertions [75] and impact the stop-codon signal.

Insertions annotated with "frameshift_variant;stop_lost;stop_retained" by SVAT and "non-frameshift insertion" by ANNOVAR We found 108 insertions that disrupt a stop-codon. These insertions also compensated for the disruption by introducing a downstream stop codon right after the insertion. As we described above, a similar case was observed for deletions. SVAT provides a detailed impact of the loss of the original stop codon and retention of the new downstream stop codon. In comparison, ANNOVAR annotates these variants as "nonframeshift". This should not impact downstream analysis because these insertions are not impactful on the mRNA, processing, and the final protein sequence.

The remaining differences in the assigned high-impact annotations are generally subclasses of the above cases and a small portion is caused by differences in gene annotations used by ANNOVAR and SVAT. In summary, SVAT and ANNOVAR agree on the overwhelming majority of the high-impactful annotations. A small fraction of the differences that we discuss above stem from differences in interpretation and precedence order of reported annotations by ANNOVAR.

Resource requirements for transcriptome-wide annotation of variants

We test the resource requirements of annotation using the variants identified by the 1000 Genomes Project.

Annotation vector generation (server)

The first step of annotation is the generation of the annotation vector at the server. We extracted the protein-coding transcripts annotated in the GENCODE's version 31. We used the 1,135,699 exons over 152,597 transcripts that cover 297,541,678 nucleotides, which is the length of the vectors for annotation and variant vectors. We have enumerated all possible single nucleotide variants (3 alleles per position) and 1-bp insertions/deletions within the target regions (exons extended by 5-base pairs), which yields a total of approximately 1.5 billion mutations. SVAT relies on the annotation of these mutations to build the annotation vector. These mutations were annotated using XCVATR [76], a lightweight and efficient tool for the annotation of variants on protein-coding genes. Although SVAT utilizes VEP's annotations by default, we used XCVATR for its computational efficiency compared to VEP. Also, our main goal here is to estimate the resource requirements and the choice of variant annotator does not impact the estimates of time and memory. Given the time and computational resources, VEP can be used to annotate the enumerated variants as well. This is a reasonable expectation because the variant annotation vector generation is performed only once on the server-side and it does not incur additional cost for new annotation tasks.

We measured the time, memory, and disk space requirements of annotation vector generation for SNV, and indel signals, which are shown in Table 3. Excluding the

Table 3 Resource requirements of annotation vector generation

Annotation vector step	Time (Sec)	Peak memory usage (GB)	Disk space (GB)
1. Region extraction	22.85	0.26	0.052
2. Region separation	4.49	0.29	
3. Variant enumeration	99.75	6.58	6.98
4. Enumerated variant annotation	66,558.10	357.28	34.23
5. Vectorization	3401.69	2.79	2.49

enumerated variant annotation step, variant vectorization step takes the longest time for parsing approximately 1.1 billion per transcript annotations assigned to the 1.5 billion variants. Enumeration step requires the largest memory. The maximum disk usage is around 41.1 gigabytes (Sum of 6.98 Gb and 34.23 Gb from steps 3 and 4 in Table 3) for storing the annotations of enumerated variants and variants themselves, which are intermediate data. The final annotation vectors occupy 2.5 gigabytes of disk space. These steps are executed over 24 autosomal chromosomes in parallel (other than enumerated variant annotation). Of note, the requirements will scale linearly with the number of variants and the total number of annotations.

Secure variant annotation steps (client and server)

We downloaded the coordinates for genetic variants from The 1000 Genomes Project on the hg38 assembly, which contains 78,229,218 variants (Data and Materials). We separated SNVs (73,257,632), small insertions (3,509,416), and small deletions (1,462,170) (Step 1 in Table 4). We next generated the vectorized loci for all variants, using a 1-bp indel encoding approach. In this step, 2,971,174 variants were encoded on at least one of the target regions described in the previous step. The vectors are then encoded and encrypted (Steps 2 and 3 in Table 4). The encrypted variants occupy approximately 22.4 gigabytes of hard disk space. After upload the encrypted variant vectors, the variants are annotated using the annotation vector (Step 4) and downloaded by the client, decrypted (Step 5), and finally translated (Step 6). The final translation yielded 8,216,281 annotations on the protein-coding genes. Overall, the whole annotation process takes 780 s on the client and 27 s on the server. There is moderate-to-high memory requirement, which is around 123 gigabytes on the client and 108 gigabytes on the server. The disk usage of encrypted data is approximately 22.4 gigabytes on the client and server.

Non-coding and regulatory element annotations

VEP can also annotate the impact of variants on non-coding elements (non-coding RNAs, regulatory regions, and transcription factor binding sites). VEP makes use of the ENSEMBL Regulatory Build [77] as the default non-coding element annotation. The non-coding annotation of variants that overlap with non-coding RNAs and regulatory elements consists of 3 types of consequences:

Table 4 Resource requirements of the variant annotation

Secure annotation step	Site (client/server)	Time (Sec)	Peak memory usage (GB)	Disk space (GB)
1. Separate VCF	Client	120.79	0.001	0.50
2. Vectorization	Client	238.67	0.69	0.17
3. Encoding and Encryption	Client	34.44	58.03	22.45
4. Secure annotation	Server	26.46	108.13	22.45
5. Decryption	Client	82.53	122.97	22.45
6. Translation	Client	303.26	0.93	0.72
7. Cumulative usage	Client/server	779.68/26.46	122.97/108.13	22.45/22.45

(1) Variants overlapping with the non-coding RNA exons ("non_coding_transcript_exon_variant")
(2) Variants overlapping with splicing sites of multi-exonic long non-coding RNAs ("splice_region_variant", "splice_acceptor_variant", "splice_donor_variant")
(3) Variants overlapping with regulatory elements including transcription factor binding sites and peaks ("regulatory_region_variant")

The non-coding annotation terms can be assigned to variants by overlapping them with the non-coding genes and cis-regulatory elements. This information can be stored using an element-level vectorized representation where each entry corresponds to a non-coding element and the value indicates the existence of a variant that overlaps with the respective element. The vectorized non-coding variant representation can be encrypted (similar to variant vector for coding genes) to provide data confidentiality and secure downstream analysis in a collaborative setting at an outsourcing server.

Here we estimate the resource requirements for secure analysis of the non-coding annotation vectors. For this, we downloaded the motif dataset from ENSEMBL Regulatory Build, the ChIP-Seq peaks from the ENCODE Project, and also used the long non-coding RNA exon annotations from The GENCODE Project. Overall, we focused on the 11,552,160 motifs with experimental evidence from ENSEMBL Regulatory Build. We identified 302,364 non-coding exons from GENCODE annotation. We included the splicing sites for each exon by extracting the 2-basepair vicinity at the ends of each exon, which results in 604,728 non-coding splice sites. Finally, we pooled all the ChIP-Seq peaks from the ENCODE project and identified 33,154,766 peaks, which were used as the peak annotations. We finally pooled all of the elements and build the non-coding variant annotation regions that comprise 45,614,018 elements.

The resource requirements of secure processing steps (starting from encryption in Table 4) scale linearly with the length of vectorized representation. By extrapolating from Table 4, we deduced that the resource requirement for secure processing of the non-coding representation (45 million elements) would be approximately 5 times smaller than the processing of the vectorized representations of protein-coding genes (297 million elements). This estimate does not include the resource requirement of the non-coding variant vectorization step (Step 2 in Table 4), which is performed at the client. To estimate the resource requirement of this step, we vectorized the SNV and small indel variants (around 78 million variants) that are cataloged by The 1000 Genomes Project. Out of the 78 million variants, 43,906,437 variants (56%) were encoded into one of the non-coding elements. We report the resource requirements in Table 5.

Overall, our results indicate that the current annotation framework can be utilized with moderate time, memory, and disk space requirements for annotation of the protein-coding variants and also annotation of non-coding variants at the element level.

Table 5 Resource usage by element-level vectorization of non-coding variants cataloged by The 1000 Genomes Project

Variant vector step	Time (Sec)	Peak memory usage (GB)	Disk space (GB)
Non-coding SNV Indel vectorization	989.35	9.31	1.73

Estimation of impact on TF binding affinity

VEP can optionally assign changes induced on TF binding affinity by the mutations that overlap with motifs. The vectorized representation we describe above does not specify the position of the variant within the motifs (or the allele) and therefore must be modified for assigning the motif impact scores, which would require larger resources. Our current approach can be used to handle this by building a motif impact annotation vector that contains the impact of every allele at each position of the TF motifs. It would also be necessary to store the position of the variant within the motif and the alternate allele, which can be encoded into the variant vector.

Resource requirements of genotype aggregation and re-encryption

Genotype aggregation for SNVs is performed by marginalization (i.e. sum) over the rows of the genotype matrix. To evaluate the time/memory and disk space requirements, we focused on the aggregation of the alternative allele frequencies over 1000 individuals using the genotype data from the phase3 of the 1000 Genomes Project [2]. As the target regions, we focused on the exons of 1000 genes on chromosome 1 that covered 6.6 megabases of nucleotides. We divided the aggregation task into 3 different steps and report resource requirements for each step: (1) Encryption, (2) Aggregation, and (3) Decryption. For the aggregation task, SVAT aggregates the genotypes allele-by-allele (i.e., the genotype matrices corresponding to allele A, C, G, T), which can help decrease memory usage. Table 6 shows the resource requirements for genotype aggregation for each allele. The time and memory requirements are measured using 24 threads. The longest time for the aggregation task is spent in encryption: 96 s are spent encrypting the 6.6 million-by-1000 genotype matrix for one allele. The disk space usage for one allele is around 83 gigabytes (approximately 13 bytes/position/sample). Aggregation takes around 2.56 s and uses 107 gigabytes of memory. Finally, the decryption takes around 0.2 s and decrypts the 6.6 million long aggregated counts of the alleles at each position using 0.34 gigabytes of memory.

Our results indicate that the aggregation task can be fairly computationally intensive compared to the annotation task. This is reasonable since the genotypic data used in the aggregation task is much larger than the variant locus information that is used for annotation tasks. Our results also provide evidence that the vectorized representation can enable large optimizations and increased speed in the secure annotation and aggregation tasks; these are enabled by the streaming operations that the new HE schemes take advantage of such as data packing and vectorization operations [33]. There is, however, an accompanying bottleneck on the disk space and memory usage of the vectorized representations.

Table 6 The resource requirements for aggregation of 6.6 megabases-by-1000 individuals genotype matrix for one allele

	Time (Sec)	Memory (GB)	Disk space (GB)
Encryption (Client)	96.26	17.94	83.01
Secure aggregation	2.56	106.98	–
Decryption (Client)	0.17	–	0.34

Table 7 The resource requirements of aggregation tasks

	Time (Sec)	Peak memory (GB)	Disk space (GB)
Encryption (NIH/Owner)			
M_1	54.4605	168.332	41.50772095
M_2	107.5746		83.01544189
Re-encryp. (Server)			
M_1	20.4396		–
M_2	39.5913		–
Aggregation (Server)	4.7757		–
Decryption (Client)	0.1867	–	0.34

Encryption is performed by the owner or the trusted entity, i.e., (NIH). The encrypted data is sent to the aggregation server. After the client requests the aggregation, the trusted entity generates the re-encryption key and sends it to the server. The genotype matrices are re-encrypted and aggregated by the server

We finally evaluated the resource requirements of the re-encryption-based pooled aggregation of genotype matrices, which enables aggregating genotype matrices that are obtained from different sources (different private keys, Fig. 1). For this, we tested the pooling of two genotype matrices where the first matrix (M_1), contains the genotypes at 6.6 million vectorized positions for 500 samples and the second matrix (M_2) contains the genotypes for 1000 samples, i.e., pooling of 1500 samples from two databases. We evaluated the 4 steps: (1) Encryption of each matrix in their own keys, (2) Re-encryption into the same private key and pooling of the matrices, (3) Aggregation of the pooled matrix, (4) Decryption of the results. Table 7 shows the time requirements of these operations for the pooling of the two datasets using 24 threads.

The most time-consuming step is encryption, which took slightly more than 116 s for M_2. The re-encryption operation takes less than 1 s for both matrices. This is reasonable since the encryption can be performed once at the trusted entity of the owner and re-encryption can be performed many times on the aggregation server. The aggregation operation takes less than 10 s and finally, the decryption operation takes less than 1 s. Disk space usage is moderate with 83 gigabytes of data that needs to be stored for the full genotype matrix. It is, however, not necessary to store the genotype data for aggregation operations. The server can generate the encrypted aggregations for each matrix, then store the encrypted vector containing the aggregated allele counts, which takes much smaller space than the genotype matrices.

Utilization of vectorized representations in collaborative settings

The main utility of SVAT and underlying vectorized representation is clearer when we consider a collaborative analysis scenario: For example, assume that multiple institutions would like to compare and process the variants that they have generated from patients or participants (for example in a rare disease study) but they cannot directly share genetic variants. Each institute encodes its variants into vectorized representations, encrypts the data, and submits it to the analysis server, which can be an untrusted server on the cloud. This is reasonable because numerous legislative regulations regarding personal data privacy (such as GDPR) treat encrypted data as safe to store and share [78]. The server can perform the variant overlap via element-wise multiplication of the vectors (similar to the secure annotation operation in Fig. 2d). The entries in the resulting *variant-overlap-vector* that are non-zero indicate the variants that exist in all the sites. Compared to other protocols that perform secure set intersection based on secure multiparty computations, this operation is much faster without the need for online operations. We also believe it fits better in bioinformatics pipelines which are currently not geared to integrate online operations where multiple rounds of communication are necessary between parties. The variant-overlap-vector can then be annotated at the server using the same annotation database and sent back to the sites for further decryption and translation. This approach enables not only data protection but also the annotation is performed once by the server rather than performing it at each site separately.

Similarly, the server can compute the union of the variants by simply summing the vectors elementwise to generate an encrypted *variant-union-vector*. After this, the gene-level, and gene-set-level mutational burden can be securely computed by summing the corresponding positions on the variant-union-vector. These statistics can be processed at the server to estimate further enrichment statistics using, for example, variant allele frequencies as aggregated by SVAT. Within these pipelines, secure variant annotation can be integrated as an intermediate filtering step to exclude variants with respect to their impacts. We believe these are several applications that SVAT can be used for customized and secure variant analysis other than annotation and aggregation operations.

Discussion

We presented SVAT, a method for secure annotation of variants and secure aggregation of genotypes from multiple databases. The increasing number of genomes frequently makes it necessary to perform batch annotation and aggregation of variants by outsourcing these operations with programmatic or through user interfaces on web servers. SVAT makes use of homomorphic encryption to provide confidentiality to the genotype data and variant loci. The proposed framework makes use of a novel vectorized representation of the variant loci to protect the variant loci information. This new representation is utilized commonly for annotation and aggregation tasks. This can enable building more complex variant analysis pipelines with allele frequency and variant impact filters. An alternative method that is similar in flavor is "private-set-intersection" (PSI) [79, 80], where two entities can securely intersect their confidential datasets without revealing any information other than only the intersecting elements. In particular, our approach is similar to the PSI implemented in Chen et al. [81] albeit not in the genomic data analysis context. Chen et al. the authors design a multiplication statistic that is computed for each data point in the query dataset (i.e., researcher's variant dataset) to compare every data point in the server's dataset using HE-based secure subtraction and multiplication operations. This encrypted data is returned to the client who decrypts and identifies the 0 entries in the returned vector, which points to intersecting elements. As Chen et al. describe, this approach works with small query datasets. Our approach limits the search space by using target regions and similarly enumerates all possible mutations and vectorizes the operations. A simple PSI query must be combined with the extra information so that annotation information is returned, which, we believe, is not a simple task. On the other hand, we assume that the server's annotation dataset is not confidential, which is a strict condition that is satisfied with a PSI search. In addition, the statistic used by Chen et al. can be modified to accommodate plaintext server data.

There are alternative approaches that can enable the confidentiality of genetic data. For example, just for the annotation task, the client can download the vectorized annotation data and perform annotation locally without ever having to upload the data. This would be similar to the "share models, keep data" approach [82], where the researchers can download the annotation vector and extract annotation information for their variants without ever sending the variants from their studies. This is, however, a different paradigm than what is undertaken in this study. SVAT aims to enable a data sharing paradigm where the encrypted data stays with the untrusted entity and can be re-used by other HE-enabled pipelines. The resharing can be enabled by the proxy-re-encryption module of SVAT. We acknowledge that there is more work to be done to render the framework more practical so that it can be run on the whole genome-scale. While the computational framework can accommodate any annotation and aggregation task, there are several limitations of SVAT that are used to bound the computational and memory cost. First, the target regions are used to decrease the vectorized data size. To focus on the most impactful mutations, SVAT uses the protein-coding exons and surrounding regions as the targets. Second, SVAT can decrease the time and memory usage for the annotation of deletions by making use of an annotation vector that contains the 1-bp deletions and making use of this to translate the impact of deletions that span multiple nucleotides. This approach is exact for the highest impact that is assigned to each deletion.

Conclusions

Variant annotation and genotype aggregation are two integral components of genomic data analysis pipelines. For instance, the software pipelines that perform variant association tests use variant annotations to classify variants based on their impact. Similarly, genotype aggregation is used for classifying and filtering genetic variants with respect to population-level allele frequencies. This is an integral step in estimating the selection pressure and detecting potentially disease-causing variants. SVAT can be integrated into these analysis pipelines to securely store and analyze datasets. The presented framework can be used in other contexts and potentially has an overarching impact on other prospective privacy-aware method development efforts.

Methods

We present the details of the annotation and aggregation workflows.

Variant annotation

Target regions are provided as a BED file with name and strand entries. The default target regions comprising the extended CDS coordinates are included with SVAT for the hg38 genome and Gencode v31 annotations [59]. The target regions can be modified using a GFF/GTF formatted gene annotation file with exon, CDS, and UTR entries present. The users can also specify the length of extension of each element, i.e. l_{ext}, to include the functional elements around exons/CDSs using "transcript-specific" and "gene-specific" target regions as described in the previous section. The input variant coordinates to be annotated are taken as a VCF file by default. SVAT processes VCF files to generate the vectorized variant loci signals.

Vectorized representation of the variant loci

The vectorization enables the protection of variants that fall on the target regions. The exon start/end coordinates are extracted for the protein-coding transcripts. Each CDS is extended by l_{ext} to include variants that may impact splicing. The target regions are described with pairs of coordinates:

$$T = [(s_i, e_i)|s_i < e_i; s_i = CDS_{i,1} - l_{ext}; e_i = CDS_{i,2} + l_{ext}]$$

where T denotes the sorted list of target regions, s_i and e_i denote the genomic coordinates for the start and end, respectively, of the ith target region, and $CDS_{i,1}$ and $CDS_{i,2}$ denote the start and end coordinates of the ith CDS. Each target region corresponds to a CDS and is assigned to an element, whose id is known, e.g. transcript/gene name. T is a sorted list and the sorting cannot be changed in the course of annotation for the mapping from genomic to vector coordinates is used multiple times. While the actual sorting of the regions does not have any specific importance, position-based sorting will make the conversion of genomic coordinates to target regions efficient using optimized searching algorithms. By default, SVAT first sorts the target regions by start position, then sorts with respect to the element identifier, i.e., transcript/gene identifier. To denote the ith target region, we use T_i, i.e., $T_i = (s_i, e_i)$. Also, we denote the total nucleotides covered by target regions by l_T:

$$l_T = \sum_i (e_i - s_i + 1)$$

Next, the sorted target regions list is used to build the vectorized signals. For this, SVAT allocates an array of lengths that is equal to the coverage of the target regions T:

$$S^{(T)} = \left(s_\nu^{(T)}\right), 1 \leq \nu \leq l_T$$

where $S^{(T)}$ denotes the vectorized signal, whose coordinates correspond to the target regions T. Every position on $S^{(T)}$ holds a value that corresponds to a position in the target regions. The mapping between the position on the vectorized signal and the sorted target regions list is done by tracking the target regions. It is necessary to identify the target region index, and then a position index on the target region so that we define exactly where the vector position maps to on the genomic coordinates. First, we identify the leftmost sorted target region whose cumulative coverage is smaller than ν:

$$i^{(T)}(\nu) = \underset{1 \leq i \leq |T|}{\operatorname{argmax}} \left(\nu > \nu_{<i}; \nu_{<i} = \sum_{1<j<i} (e_i - s_i + 1) \right)$$

where $\nu_{<i}$ indicates the total cumulative number of nucleotides that are covered by the target regions $T_1, T_2, \ldots, T_i$, and $i^{(T)}(\nu)$ indicates the target region index ($i^{(T)}(\nu) < |T|$) that the vectorized position ν corresponds to. The above formula, although complex looking, simply states that we start from the leftmost target region and track the total number of nucleotides that are covered by the sorted target regions list. The largest region's index such that $\nu_{<k} < \nu < \nu_{<(k+1)}$ where $k = i^{(T)}(\nu)$. This can be performed by a binary search over the sorted target regions list. After the target region index is identified, we identify the position that corresponds to the target region as:

$$c^{(T)}(\nu) = s_k + (\nu - \nu_{<k});\, k = i^{(T)}(\nu)$$

where $c^{(T)}(\nu)$ indicates the genomic coordinate of the vectorized position ν when it is mapped onto the target regions T. The mapping of the target coordinates onto the vector coordinates can be performed similarly: $\nu^{(T)}(i,c) = \nu_{<i} + (c - s_i)$. It should be noted that we can only map the coordinates of the targets to vectors in a 1–1 fashion. The genomic coordinates can be mapped to multiple target regions (and to multiple corresponding vectorized positions). To identify all vector coordinates that map to a genomic coordinate (pos), we identify all the target regions that overlap with this position:

$$\mathrm{N}^{(T)}(pos) = \left\{\nu | c^{(T)}(\nu) = pos\right\}$$

where $\mathrm{N}^{(T)}(pos)$ denotes the set of vector coordinates that map to genomic coordinatepos. To implement the above search, target regions whose genomic coor $c^{(T)}(\nu) = s_k + (\nu - \nu_{<k});\, k = i^{(T)}(\nu)$ dinates overlap with pos can be identified then the formula for $\nu^{(T)}(i,c)$ can be used for each overlapping target region.

Vectorized mutation loci

An array of length l_T that is indexed by the vectorized coordinates (with respect to a sorted target region list) is used to store the variant locus information. Given a variant allele a ($a \in \{A, C, G, T, \delta, \iota\}$ (the alternate alleles of an SNV, 1-bp deletion, and 1-bp insertion), variant loci are mapped onto the vectorized coordinates as described previously. Next, for each allele, an array of l_T is allocated and each array entry whose vectorized position overlaps with a variant is set to 1. All other positions are set to 0 (Fig. 2d):

$$\Lambda_a^{(T)}(\nu) = \begin{cases} 1;\ \exists c \in \mu | \nu \in N^{(T)}(c) \\ 0;\ \mathit{Otherwise} \end{cases}, a \in \{A, C, G, T, \iota\}$$

where $\Lambda_a^{(T)}(\nu)$ denotes the variant loci array for allele a, μ denotes the set of the genomic coordinates of the mutations. The above equation simply describes that the position ν on the mutation loci array is set to 1 if there exists a mutation with allele a whose genomic coordinate maps to ν, i.e. $\nu \in N^{(T)}(c)$. For the deletions, every position on the deletion is set to 1:

$$\Lambda_\delta^{(T)}(\nu) = \begin{cases} 1;\ \exists(\delta_1, \delta_2) \in \Delta | \delta_1 \le c \le \delta_2, \nu \in N^{(T)}(c) \\ 0;\ \mathit{Otherwise} \end{cases}$$

where Δ denotes the list of start–end coordinates for the deletions and δ_1, δ_2 denote the coordinates of the first and last nucleotides that are deleted in the deletion. To set the array, we can simply iterate over all deletions in Δ, then for every genomic position $\delta_1 \le c \le \delta_2$, find the mapping vector coordinates and set the vectorized deletion loci signal to 1.

Encryption of the vectorized variant loci

For simplicity, we assume that l_T is divisible by the plaintext length bound l. Then an array of length l_T is divided into plaintext vectors of size l, each of which is encrypted using the public key of the cryptosystem. Otherwise, an input array is divided in a way that as many sub-vectors as possible have the maximal size l and the only last vector is chosen to have a smaller size than the others. To be specific, the encryption procedure results in $\lceil l_T/l \rceil$ ciphertexts.

Vectorized annotation information

SVAT uses an array of length l_T to hold the annotations of all mutations on the target regions. This is very similar to the mutation loci vector, except that for each position ν, the array stores a 64-bit entry that packs the annotation information.

$$\mathrm{I}_a^{(T)}(\nu) = \begin{cases} \textit{Impact of allelea}; & \exists c \in \mu | \nu \in N^{(T)}(c) \\ 0; & \textit{Otherwise} \end{cases}$$

The impact value indicates the impact of the mutation located at vector position ν with allele a, and μ denotes the set of genomic coordinates (i.e., indices) for all mutations. The impact information is retrieved from Variant Effect Predictor, VEP, by default. SVAT packs the impact information and the nucleotide information around the mutation locus. The packed impact information contains:

1. *Coding frame (2 bits)*: This is the coding frame of a mutation that is a value in {0,1,2}: The frame is converted to a 2-bit value and used as-is.
2. *6-base pair nucleotide neighborhood (18 bits)*: SVAT assigns 3-bits values (A: 000, C: 001, G: 010, T:011, N:100) to each nucleotide and codes the surrounding nucleotide sequence into 18 bits. The sequence of bits is concatenated and used as the neighborhood sequence information.
3. *The assigned impact string identifier assigned by VEP (38 bits)*: The bitmap that describes the impact values of the mutation out of the 38 different impact values that VEP assigns to each mutation. The bitmap is generated by setting the impact values to 1 for every impact string assigned by VEP. For every annotation, the index of the impact string in the bitmap is set to 1.

The annotation server generates the vectorized annotation signal. In order to generate the vectorized annotation information, VEP is run to generate the annotation of all the nucleotides (and all alleles) of the mutations on the target regions. This is performed for 4 alleles of SNVs, 1-bp deletions, and 1-bp insertions on all positions on the target regions. The output of VEP can be directly piped into SVAT to convert the VEP annotations into the vectorized annotation signal. SVAT first maps the position to the vectorized coordinates. Next, the coding frame information and the 38-bit impact bitmap are updated using the assigned frame and the list of impact strings. SVAT finally extracts the 6-bp neighboring nucleotides from the genome sequence (by default, hg38 genome sequence) and builds the 58-bit annotation vector value. The annotation value is assigned to all of the vector coordinates that map to the mutation's genomic coordinates. The packed impact value can be formulated as:

$$\textit{Packed Impact Value} = \left(\left((\textit{VEP Impact Index}) << 18 + (\textit{Neighborhood})\right) << 2\right) + (\textit{coding frame})$$

The whole operation for building the vectorized annotation signal is a compute-intensive task. However, this needs to be performed at the annotation server. We assume that the annotation server has large computational resources and can perform this operation. In addition, the annotations need to be vectorized once for every gene annotation.

Secure SNV variant annotation

After the variant loci and the annotations are vectorized, the annotated variants vector is computed as the multiplication of the variant loci vector and the annotation vector (Fig. 2d). The result from this product is a vector of length l_T:

$$\mathrm{I}_a^{(T)}(\nu) \times \Lambda_a^{(T)}(\nu) \forall a \in \{A, C, G, T\}, \nu \in [1, l_T]$$

In this vector, the entries for which there is no variant are set to 0 (since $\Lambda_a^{(T)}(\nu) = 0$ when no mutation maps to ν) and others are set to the annotation value at the position.

Suppose that for $j = 1, 2, \ldots, \lceil l_k/l \rceil$, a plaintext vector $z_j = (I_a^{(T)}\left(\nu_{(j-1)\cdot l+1}\right), I_a^{(T)}\left(\nu_{(j-1)\cdot l+2}\right), \ldots, I_a^{(T)}\left(\nu_{j\cdot l}\right))$ is encrypted into a single ciphertext ct_j. The server performs the constant-ciphertext multiplication with the ciphertext of the annotation vector and the corresponding variant loci vector as follows:

$$ct_j \cdot \left(\Lambda_a^{(T)}\left(\nu_{(j-1)\cdot l+1}\right), \Lambda_a^{(T)}\left(\nu_{(j-1)\cdot l+2}\right), \ldots, \Lambda_a^{(T)}\left(\nu_{j\cdot l}\right)\right).$$

Then the results are sent back to the user. For each position on the array that is non-zero, a variant exists. Streaming operations are used by the client to decrypt the downloaded data and filter out non-zero annotation entries (64-bit packed annotation information), which are unpacked according to the bit packing above. The conversion of the vector coordinates to genomic coordinates (i.e., from ν to (i, c)) are performed while looping over the vectorized coordinates, i.e., the client can decrypt and decode the annotation values as they are received from the annotation server in a streaming fashion.

For SNVs, the client performs the following steps:

1. Receive and decrypt data
2. For each non-zero value on the annotated vector at position ν,
3. Translate ν: Compute $i^{(T)}(\nu)$ and $c^{(T)}(\nu)$
4. Extract the 38-bit impact bitmap from the annotated value
5. For every bit that is set to 1, use the lookup table and concatenate the variant annotation.

Secure small deletion variant annotation

Annotation of deletion variants needs to account for the variable lengths of the variants. The server can provide all the vectorized annotation signals for deletions that are shorter than a certain value, e.g., $\mathrm{I}_{|\delta|=l\_\delta}^{(T)}(\nu), l_\delta < 20$. Similarly, the researcher can generate the encrypted variant loci vectors for deletions of each length. After this, the annotation can be performed as it is performed for SNVs annotations by first secure multiplication of the variant loci vector and annotation vector for each deletion length, then decoding the 38-bit annotation impact bitmap for each non-zero entry in the product vector for each length.

SVAT also implements an alternative approach for the annotation of deletions. SVAT uses the 1-base pair deletion impact signal to track the impact of consecutively deleted nucleotides: For a deletion of nucleotides $[a, b]$ $(a < b)$, SVAT traces the nucleotides and merges the impacts of consecutive 1-base pair deletions, at every nucleotide ν in $[a, b]$, i.e., $I_\delta^{(T)}(\nu)$. To merge the 1-base pair deletion annotations and to generate the impact of a deletion at $[a, b]$, SVAT makes use of the coding frame information and the neighborhood sequence.

Coding frame impact

SVAT traces the impact values of all nucleotides between $[a, b]$, and counts the number of coding nucleotides that are affected by the deletion. If the number of coding nucleotides is a multiple of 3, the deletion is marked as an in-frame deletion. Otherwise, the deletion is marked as a frameshift deletion.

Start/stop loss

If any of the nucleotides in $[a, b]$ have an impact on the start or stop codons, the deletion is marked with a start/stop loss.

Splice donor/acceptor loss

If any of the nucleotides have an impact on the splice donor/acceptor sites, the deletion is marked with splice donor/acceptor.

Stop gain/retain

SVAT utilizes the neighborhood and the coding frame to identify the coding frame around the deletion breakpoint. This way, SVAT computes the new codon (using a lookup table) that is introduced by the deletion. If the new codon is one of the three stop codons, i.e., “TAG”, “TAA”, “TGA”, SVAT assigns the stop gain impact. If the deletion impacts a stop codon and generates a new one around the breakpoint, “Stop Retained” impact is added to the aggregate impact value of the variant.

Secure small insertion variant annotation

Insertions are short sequences that are inserted with respect to the reference genome. Unlike a deletion, an insertion occurs at a single location and is described by the sequence that is inserted at the location. The annotation of insertions is carried out the same way as deletions by first identifying the position of the insertion and then translating the coding nucleotides that are inserted to assign the impact.

Coding frame impact

SVAT traces the number of coding nucleotides that are inserted into a CDS and assigns in-frame or frameshift annotation.

Sstart/stop loss/gain/retainment

For an insert that overlaps with a start or a stop codon, the neighborhood nucleotide and the coding frame is used to identify the inserted codons and set whether a stop/start codon is lost/gained or retained.

The scenario of secure insertion annotation is the same as secure deletion annotation, where a 1-base pair insertion annotation vector ($I_{Ins}^{(T)}(\nu)$) is multiplied by the encrypted 1-base pair insertion mutation vector ($\Lambda' Ins^{(T)}(\nu)$). The multiplied encrypted vector is sent back to the researcher, which is decrypted and processed.

Genotype aggregation

The genotype aggregation aims as computing the frequencies of the mutations by aggregating over many samples:

$$f_a^{(T)}(\nu) = \sum_{1 \le k \le n_G} G_{a,k}^{(T)}(\nu)$$

where n_G denotes the number of individuals in the genotype matrix, $f_a^{(T)}(\nu)$ denotes the frequency of the allele a for the variant at vector coordinate ν (indexed on the target regions T), and $G_{a,k}^{(T)}(\nu)$ indicates the genotype of the variant for individual k: $G_{a,k}^{(T)}(\nu) \in \{0, 1, 2\}$, where the entry indicates the number of alternate alleles equal to a. The genotypes can also be encoded with 2-level encoding [39] to indicate the existence of the allele at the location: $G_{a,k}^{(T)}(\nu) \in \{0, 1\}$.

While aggregating the genotype matrix, it is necessary to ensure the confidentiality of the genotype matrix $G_{a,k}^{(T)}$ from the untrusted aggregation server, which performs the computationally heavy task of secure aggregation. To compute the above summation securely, the genotype matrix is encrypted, denoted by $G'a, k^{(T)}$ and the summation is evaluated using HE-based secure summation.

As practical homomorphic encryption systems support computation on encrypted vectors, we encrypt the whole genotype matrix $G'a, k^{(T)}$ by taking entries in row-major order and generating plaintext vectors. In other words, each row vector is divided into sub-vectors of length l, each of which is encrypted using the public key. This vectorization makes it possible to use SIMD instructions that operate the aggregation on vectors (i.e., with support for entry-wise addition). To be specific, suppose that for each $k = 1, 2, \ldots, n_G$, a plaintext vector $z_k = (G_{a,k}^{(T)}(\nu_1), G_{a,k}^{(T)}(\nu_2), \ldots, G_{a,k}^{(T)}(\nu_l))$ is encrypted into a single ciphertext ct_k. Then we perform an aggregation over ciphertexts: $\sum_{k=1}^{n_G} ct_k$. By the homomorphic property, we have

$$\sum_{1\le k\le n_G}^{n_G} ct_k = Enc\left(\sum_{1\le k\le n_G}^{n_G} G_{a,k}^{(T)}(\nu_1), \sum_{1\le k\le n_G}^{n_G} G_{a,k}^{(T)}(\nu_2), \ldots, \sum_{1\le k\le n_G}^{n_G} G_{a,k}^{(T)}(\nu_l)\right),$$

which is an encryption of the frequency vector $(f_a^{(T)}(\nu_1), f_a^{(T)}(\nu_2), \ldots, f_a^{(T)}(\nu_l))$. The ciphertext packing technique enables us to encrypt l different messages into a single ciphertext and perform l operations at a time over encryption.

Another important aspect of aggregation is to accommodate genotype matrices from multiple databases so that many samples can be aggregated together. The server needs to be able to manage multiple datasets that are encrypted with different keys. SVAT implements a proxy re-encryption protocol to convert the genotype matrices into the same key and perform the aggregation using this common key.

Proxy re-encryption protocol

The proxy re-encryption aims at converting the genotype matrices (or any other type of data) to the same encryption key [70]. For this, a trusted entity is required who will perform the key management and who holds the private keys necessary to generate the re-encryption keys. This is a reasonable assumption since the sensitive datasets are generally deployed and protected by entities such as NIH (e.g., dbGAP).

We assume that there are M genotype matrices (on the same vectorized coordinates system, i.e., common sorted target regions T), such that mth matrix is encrypted with the public key denoted by $pk_m^{(G)}(m \le M)$. The corresponding private keys, $sk_m^{(G)}$ $(m \le M)$, are stored by the trusted entity such as NIH, and mth matrix can be decryptable only with the private key $skm(G)$. The data is stored at the untrusted server in an encrypted format. When a researcher asks for the aggregation service, the researcher provides the public key, $pk^{(R)}$ (Researcher public-key) and a corresponding private key $sk^{(R)}$. It is necessary to re-encrypt the encrypted genotype matrices $G'a, k^{(T)}(\nu)$ so that they can be decrypted with the private key $sk^{(R)}$. The researcher sends the public and private keys to NIH, which generates the re-encryption key for the data, i.e. $swkm = swksk_m^{(G)}, sk^{(R)}$ foreach$m \le M$ where $skwm$ denotes the re-encryption key. The aggregation server receives $skwm$ for all genotype matrices and re-encrypts so that they can be decrypted by the same key as the user $sk^{(R)}$.

Traditional homomorphic encryption systems can convert a ciphertext decryptable with a secret key sk_1 to a new ciphertext with a secret key sk_2, which is called the *key-switching* operation. To be specific, if the switching key $swk = swk(sk_1, sk_2)$ is generated, then the key-switching operation takes as input a ciphertext ct with the secret key sk_1 and perform the key-switching operation.

$ct'' \leftarrow KeySwitch(ct, swk)$.

Then the resulting ciphertext $ct^{''}$ is decryptable with the secret key sk_2. In our case, the secret keys are set as $sk_1 = sk_m^{(G)}$ and $sk_2 = sk^{(R)}$, and the aggregation server performs the key-switching operations on the encrypted genotype matrices using the public switching key swk_m without knowing any information about the secret keys. The aggregation service then securely aggregates the frequency counts at every position on the T. The resulting frequency array, which is encrypted with the researcher's public key, $f'^{(T)}_{a}(\nu)$, is sent to the researcher who can decrypt the frequency array and obtain frequencies.

SNV aggregations

The secure aggregation of the SNVs is straightforward as they are located at one position on the vectors. In other words, the SNV frequency aggregation can be computed simply by marginalizing at every location.

Indel aggregations

Unlike SNVs, the deletions must be tracked in each sample and aggregated. As with the annotation task, we make use of the 1-base pair deletions to build the aggregation of deletion variants that span longer than 1 bp. Given a position ν, and the 1-base pair deletion genotype matrix, $G^{(T)}_{\delta,k}(\nu)$; the indel of length l_δ is aggregated by tracing the deleted nucleotides:

$$f'^{(T)}_{\delta(l_\delta)}(\nu) = \sum_{k} \alpha' \begin{pmatrix} G^{(T)}_{\delta,k}(\nu-1) = 0, & G^{(T)}_{\delta,k}(\nu+l_\delta+1) = 0, \\ \forall l \in [0, l_\delta]; & G^{(T)}_{\delta,k}(\nu+l) = 1 \end{pmatrix}$$

where $f'^{(T)}_{\delta(l_\delta)}(\nu)$ indicates the frequency of the l_δ-deletion at vector position ν, and $\alpha(\cdot)$ is an indicator function, i.e., it returns 1 if the arguments are true and 0 otherwise. In the formula above, the summation aggregates the individuals for which, the 1-base pair deletion genotype matrix contains a deletion state for all positions in $\{\nu, \nu+1, \nu+2, \ldots, \nu+l_\delta\}$. In addition, the deletion state is unset (i.e., $G^{(T)}_{\delta,k}(\nu-1) = 0$) for the individual k at positions $\{\nu-1, \nu+l_\delta+1\}$. This ensures that the deletion is set exactly over the deleted interval we would like to aggregation, i.e., $[\nu, \nu+l_\delta]$ and also the positions right out of the deleted interval are set to undeleted. It should be noted that for any aggregation of deletions, the aggregation can be performed for multiple deletion lengths (i.e., $l_\delta < l_{max}$) and stored at the aggregation server.

The aggregation of insertions requires explicit matching of the inserted nucleotides and requires enumeration of all possible insertions. SVAT currently does not explicitly support aggregation of short insertion variants. However, the position at which the insertion happens can be aggregated (just by simple aggregation as for SNVs) to compute the frequency of insertion at each position.

Parallelization of computations

There are several options for parallelization of different steps:

Chromosome and ciphertext-level parallelization

The most natural parallelization is to separate the variant data with respect to chromosomes. The computations can then be performed on different instances on the cloud or the server such that each instance processes one chromosome. It may however be more efficient to divide the target regions with equal coverage such that the number of ciphertexts is similar when distributed among the compute instances so that the load is balanced among instances while performing annotation or aggregations.

Annotation-level parallelization

Another parallelization option is to parallelize different types of annotations among different compute instances and storage buckets. The coding and non-coding annotations can be managed by different buckets and instances on the server side and could also be uploaded by different computers on the client side. Uploading the data to different buckets from different clients also enable a faster upload speed for the large datasets such that the total bandwidth from multiple computers is utilized simultaneously, assuming that there is no other bottleneck at the network level (e.g., gateways, firewalls, etc.).

Abbreviations

i	A sorted target region index.
c	A genomic coordinate.
pos	A genomic coordinate.
v	Index of the vectorized coordinates
T	Sorted target regions list
s_i	Genomic position for the start of ith target region.
$\Lambda_a^{(T)}(\nu)$	Variant loci vector of length l_T for allele $a, a \in \{A, C, G, T, \delta, \iota\}$ where δ and ι indicate 1-base pair deletion and 1-base pair insertion.
$I_a^{(T)}(\nu)$	Variant impact vector of length l_T.
e_i	Genomic position for the end of ith target region.
l_{ext}	Extension length by which each target has been extended at start and end.
$\nu_{<k}$	The total coverage of the target regions to the left of kith sorted target region.
$i^{(T)}(\nu)$	Index of the target region whose coordinates overlap with the vectorized coordinate ν.
$c^{(T)}(\nu)$	Genomic coordinates for the vectorized coordinate ν.
$N^{(T)}(pos)$	The set of vectorized coordinates for which the genomic coordinate is pos

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s12859-022-04959-6.

Additional file 1. Supplementary Information for "SVAT: Secure Outsourcing of Variant Annotation and Genotype Aggregation.

Acknowledgements
We acknowledge and thank Luyao Chen for providing technical support to set up the computational environment for performing benchmarking experiments.

Author contributions
AH, MK, and XJ conceived the methodological ideas, gathered datasets, and designed and implemented experiments and devised the conceptual design in Fig. 1. SW ran experiments, collected and tabulated results for Figs. 2, 3 and 4 and all Tables. AH and MK drafted the manuscript, reviewed the results and all authors read and approved of the final manuscript.

Funding
During this work, SW and AH were supported by startup funds from The University of Texas Health Science Center, Houston. XJ is CPRIT Scholar in Cancer Research (RR180012), and he was supported in part by Christopher Sarofim Family Professorship, UT Stars award, UTHealth startup, the National Institute of Health (NIH) under Award number R13HG009072, R01GM114612, and the National Science Foundation (NSF) RAPID #2027790.

Availability of data and materials
The datasets that are generated and/or analyzed in this study are publicly available. Data, source code, documentation with examples, and intermediate data files are publicly available from our github repository: https://github.com/harmancilab/SVAT. Simulated datasets are generated using SVAT. The 1000 Genomes Project call sets and genotypes datasets are publicly available from: https://ftp-trace.ncbi.nih.gov/1000genomes/ftp/phase1/analysis_results/integrated_call_sets/. The variant loci with coordinates on the hg38 assembly are downloaded from: https://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000_genomes_project/release/20190312_biallelic_SNV_and_INDEL/ALL.wgs.shapeit2_integrated_snvindels_v2a.GRCh38.27022019.sites.vcf.gz. ENSEMBL Regulatory Build motif datasets were downloaded from: http://ftp.ensembl.org/pub/release-106/regulation/homo_sapiens/MotifFeatures/. ChIP-Seq peak calls are downloaded from: http://encodeproject.org.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 22 August 2021 Accepted: 20 September 2022
Published online: 01 October 2022

References

1. The International HapMap 3 Consortium. Integrating common and rare genetic variation in diverse human populations. Nature. 2010;467:52–8. https://doi.org/10.1038/nature09298.
2. The 1000 Genomes Project Consortium. A global reference for human genetic variation. Nature. 2015;526:68–74. doi:https://doi.org/10.1038/nature15393.
3. Caulfield M, Davies J, Dennys M, Elbahy L, Fowler T, Hill S, et al. The 100,000 Genomes Project Protocol. Genomics Engl. 2015.
4. Collins FS. The Cancer Genome Atlas (TCGA). Online. 2007;:1–17.
5. NHLBI. NHLBI Trans-Omics for Precision Medicine Whole Genome Sequencing Program. TOPMed. https://www.nhlbiwgs.org/. 2016.
6. Consortium TG. The Genotype-Tissue Expression (GTEx) project. Nat Genet. 2013;45:580–5. https://doi.org/10.1038/ng.2653.
7. Collins FS. A new initiative on precision medicine. N Engl J Med. 2015;372:793–5.
8. Allyse M, Minear MA, Berson E, Sridhar S, Rote M, Hung A, et al. Non-invasive prenatal testing: A review of international implementation and challenges. International Journal of Women's Health. 2015.
9. Abraham G, Inouye M. Genomic risk prediction of complex human disease and its clinical application. Current Opinion in Genetics and Development. 2015.
10. Erlich Y, Narayanan A. Routes for breaching and protecting genetic privacy. Nat Rev Genet. 2014;15:409–21. https://doi.org/10.1038/nrg3723.
11. Francis LP. Genomic knowledge sharing: a review of the ethical and legal issues. Appl Transl Genomics. 2014;3:111–5.
12. Naveed M, Ayday E, Clayton EW, Fellay J, Gunter CA, Hubaux J-P, et al. Privacy in the genomic era. ACM Comput Surv. 2015;48:1–44. https://doi.org/10.1145/2767007.
13. Chen X, Gururaj AE, Ozyurt B, Liu R, Soysal E, Cohen T, et al. DataMed—an open source discovery index for finding biomedical datasets. J Am Med Inform Assoc. 2018;25:300–8.
14. Ohno-Machado L, Sansone SA, Alter G, Fore I, Grethe J, Xu H, et al. Finding useful data across multiple biomedical data repositories using DataMed. Nat Genet. 2017;49:816–9.
15. Taliun D, Harris DN, Kessler MD, Carlson J, Szpiech ZA, Torres R, et al. Sequencing of 53,831 diverse genomes from the NHLBI TOPMed Program. Nature. 2021;590:290–9.
16. Wei YL, Li CX, Jia J, Hu L, Liu Y. Forensic identification using a multiplex assay of 47 SNPs. J Forensic Sci. 2012;57:1448–56.
17. Pakstis AJ, Speed WC, Fang R, Hyland FCL, Furtado MR, Kidd JR, et al. SNPs for a universal individual identification panel. Hum Genet. 2010;127:315–24.
18. Cyranoski D. China's crackdown on genetics breaches could deter data sharing. Nat. 2018;2018:5637731.
19. Telenti A, Jiang X. Treating medical data as a durable asset. Nat Genet. 2020;52:1005–10. https://doi.org/10.1038/s41588-020-0698-y.
20. Yousefi S, Abbassi-Daloii T, Kraaijenbrink T, Vermaat M, Mei H, van't Hof P, et al. A SNP panel for identification of DNA and RNA specimens. BMC Genomics. 2018. https://doi.org/10.1186/s12864-018-4482-7.
21. Patel A, Shah N, Ramoliya D, Nayak A. A detailed review of Cloud Security: Issues, Threats Attacks. In: Proceedings of the 4th International Conference on Electronics, Communication and Aerospace Technology, ICECA 2020. Institute of Electrical and Electronics Engineers Inc.; 2020. p. 758–64.
22. Qayyum A, Ijaz A, Usama M, Iqbal W, Qadir J, Elkhatib Y, et al. Securing machine learning in the cloud: a systematic review of cloud machine learning security. Front Big Data. 2020;3:43. https://doi.org/10.3389/fdata.2020.587139.
23. Satya Nadella: SolarWinds Hack Underscores Need For 'Moving To The Cloud.' https://www.crn.com/news/cloud/satya-nadella-solarwinds-hack-underscores-need-for-moving-to-the-cloud-. Accessed 17 Jun 2021.
24. Gymrek M, McGuire AL, Golan D, Halperin E, Erlich Y. Identifying personal genomes by surname inference. Science. 2013;339:321–4. https://doi.org/10.1126/science.1229566.
25. Im HK, Gamazon ER, Nicolae DL, Cox NJ. On sharing quantitative trait GWAS results in an era of multiple-omics data and the limits of genomic privacy. Am J Hum Genet. 2012;90:591–8.
26. Harmanci A, Gerstein M. Quantification of private information leakage from phenotype-genotype data: linking attacks. Nat Methods. 2016;13:251–6. https://doi.org/10.1038/nmeth.3746.
27. Harmanci A, Gerstein M. Analysis of sensitive information leakage in functional genomics signal profiles through genomic deletions. Nat Commun. 2018. https://doi.org/10.1038/s41467-018-04875-59.
28. Backes M, Berrang P, Bieg M, Eils R, Herrmann C, Humbert M, et al. Identifying personal DNA methylation profiles by genotype inference. In: Proceedings—IEEE Symposium on Security and Privacy. 2017. p. 957–76.

29. Fienberg SE, Slavković A, Uhler C. Privacy preserving GWAS data sharing. In: Proceedings—IEEE International Conference on Data Mining, ICDM. 2011. p. 628–35.
30. Dwork C. Differential privacy. Int Colloq Autom Lang Program. 2006;4052:1–12. https://doi.org/10.1007/11787006_1.
31. Dwork C, Lei J. Differential privacy and robust statistics. In: Proceedings of the Annual ACM Symposium on Theory of Computing. 2009. p. 371–80.
32. Dowlin N, Gilad-Bachrach R, Laine K, Lauter K, Naehrig M, Wernsing J. Manual for using homomorphic encryption for bioinformatics. Proc IEEE. 2017.
33. Naehrig M, Lauter K, Vaikuntanathan V. Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop—CCSW '11. 2011. p. 113. doi:https://doi.org/10.1145/2046660.2046682.
34. Gentry C. A fully homomorphic encryption scheme. PhD Thesis. 2009; p. 1–209. doi:https://doi.org/10.1145/1536414.1536440.
35. Kim M, Song Y, Li B, Micciancio D. Semi-parallel logistic regression for GWAS on encrypted data. BMC Med Genomics. 2020. https://doi.org/10.1186/s12920-020-0724-z.
36. Kim M, Harmanci A, Bossuat J-P, Carpov S, Cheon JH, Chillotti I, et al. Ultra-fast homomorphic encryption models enable secure outsourcing of genotype imputation. bioRxiv. 2020;20:1122.
37. Ishai Y, Kushilevitz E, Ostrovsky R, Sahai A. Zero-knowledge from secure multiparty computation. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing—STOC '07. 2007.
38. Orlandi C. Is multiparty computation any good in practice? In: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing—Proceedings. 2011.
39. Raisaro JL, Choi G, Pradervand S, Colsenet R, Jacquemont N, Rosat N, et al. Protecting privacy and security of genomic data in i2b2 with homomorphic encryption and differential privacy. IEEE/ACM Trans Comput Biol Bioinf. 2018;15:1413–26. https://doi.org/10.1109/TCBB.2018.2854782.
40. Raisaro JL, Troncoso-Pastoriza JR, El-Zein Y, Humbert M, Troncoso C, Fellay J, et al. Genoshare: Supporting privacy-informed decisions for sharing individual-level genetic data. In: Studies in Health Technology and Informatics. IOS Press; 2020. p. 238–41. doi:https://doi.org/10.3233/SHTI200158.
41. Chen F, Wang S, Jiang X, Ding S, Lu Y, Kim J, et al. PRINCESS: Privacy-protecting Rare disease international network collaboration via encryption through Software Guard Extensions. Bioinformatics. 2017;33:btw758.
42. Kim M, Lee J, Ohno-Machado L, Jiang X. Secure and differentially private logistic regression for horizontally distributed data. IEEE Trans Inf Forensics Secur. 2020;15:695–710.
43. Yang H, Wang K. Genomic variant annotation and prioritization with ANNOVAR and wANNOVAR. Nat Protoc. 2015;10:1556–66. https://doi.org/10.1038/nprot.2015.105.
44. Bahcall OG. Genetic variation: ExAC boosts clinical variant interpretation in rare diseases. Nat Rev Genet. 2016;17:584–584. https://doi.org/10.1038/nrg.2016.121.
45. ALFA: Allele Frequency Aggregator. https://www.ncbi.nlm.nih.gov/snp/docs/gsr/alfa/. Accessed 31 May 2021.
46. Lek M, Karczewski KJ, Minikel EV, Samocha KE, Banks E, Fennell T, et al. Analysis of protein-coding genetic variation in 60,706 humans. Nature. 2016;536:285–91. https://doi.org/10.1038/nature19057.
47. Karczewski KJ, Weisburd B, Thomas B, Solomonson M, Ruderfer DM, Kavanagh D, et al. The ExAC browser: displaying reference data information from over 60 000 exomes. Nucleic Acids Res. 2017;45:D840–5. https://doi.org/10.1093/nar/gkw971.
48. Cummings BB, Karczewski KJ, Kosmicki JA, Seaby EG, Watts NA, Singer-Berk M, et al. Transcript expression-aware annotation improves rare variant interpretation. Nature. 2020;581:452–8. https://doi.org/10.1038/s41586-020-2329-2.
49. Karczewski KJ, Francioli LC, Tiao G, Cummings BB, Alföldi J, Wang Q, et al. The mutational constraint spectrum quantified from variation in 141,456 humans. Nature. 2020;581:434–43. https://doi.org/10.1038/s41586-020-2308-7.
50. McLaren W, Gil L, Hunt SE, Riat HS, Ritchie GRS, Thormann A, et al. The ensembl variant effect predictor. Genome Biol. 2016;17:122. https://doi.org/10.1186/s13059-016-0974-4.
51. Wang K, Li M, Hakonarson H. ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. Nucleic Acids Res. 2010;38:e164–e164. https://doi.org/10.1093/nar/gkq603.
52. Oscanoa J, Sivapalan L, Gadaleta E, Dayem Ullah AZ, Lemoine NR, Chelala C. SNPnexus: a web server for functional annotation of human genome sequence variation (2020 update). Nucleic Acids Res. 2020;48:W185–92. https://doi.org/10.1093/NAR/GKAA420.
53. Chen J, Harmanci AS, Harmanci AO. Detecting and annotating rare variants. In: Encyclopedia of Bioinformatics and Computational Biology. New York: Elsevier; 2019. p. 388–99. https://doi.org/10.1016/B978-0-12-809633-8.20121-0.
54. Psaty BM, Rich SS, Boerwinkle E. Innovation in genomic data sharing at the NIH. N Engl J Med. 2019;380:2192–5.
55. Sim I, Stebbins M, Bierer BE, Butte AJ, Drazen J, Dzau V, et al. Time for NIH to lead on data sharing. Science. 2020;367:1308–9.
56. Haeusermann T, Fadda M, Blasimme A, Tzovaras BG, Vayena E. Genes wide open: Data sharing and the social gradient of genomic privacy. AJOB Empir Bioeth. 2018;9:207–21. https://doi.org/10.1080/23294515.2018.1550123.
57. Bernstein DJ, Lange T. Post-quantum cryptography. Nature. 2017;549.
58. Dowlin N, Gilad-Bachrach R, Laine K, Lauter K, Naehrig M, Wernsing J. Manual for Using Homomorphic Encryption for Bioinformatics: This paper provides a new homomorphic encryption algorithm and associated software for bioinformatics to enhance the security and privacy associated with computing on human genomes. Proc IEEE. 2017;105:552–67.
59. Frankish A, Diekhans M, Ferreira AM, Johnson R, Jungreis I, Loveland J, et al. GENCODE reference annotation for the human and mouse genomes. Nucleic Acids Res. 2019;47:D766–73. https://doi.org/10.1093/nar/gky955.
60. Brakerski Z, Gentry C, Vaikuntanathan V. (Leveled) fully homomorphic encryption without bootstrapping. In: ITCS 2012—Innovations in Theoretical Computer Science Conference. New York, New York, USA: ACM Press; 2012. p. 309–25. doi:https://doi.org/10.1145/2090236.2090262.

61. Brakerski Z. Fully homomorphic encryption without modulus switching from classical GapSVP. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer, Berlin, Heidelberg; 2012. p. 868–86. doi:https://doi.org/10.1007/978-3-642-32009-5_50.
62. Fan J, Vercauteren F. Somewhat Practical Fully Homomorphic Encryption. Proc 15th Int Conf Pract Theory Public Key Cryptogr. 2012; p. 1–16. https://eprint.iacr.org/2012/144.
63. Chillotti I, Gama N, Georgieva M, Izabachène M. TFHE: fast fully homomorphic encryption over the torus. J Cryptol. 2020;33:34–91.
64. Cheon JH, Kim A, Kim M, Song Y. Homomorphic encryption for arithmetic of approximate numbers. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2017; p. 409–37.
65. Adzhubei I, Jordan DM, Sunyaev SR. Predicting functional effect of human missense mutations using PolyPhen-2. Curr Protoc Hum Genet. 2013. https://doi.org/10.1002/0471142905.hg0720s76Chapter7:Unit7.20.
66. Cooper GM. Translation of mRNA. 2000. https://www.ncbi.nlm.nih.gov/books/NBK9849/. Accessed 31 May 2021.
67. Wang S, Jiang X, Tang H, Wang X, Bu D, Carey K, et al. A community effort to protect genomic data sharing, collaboration and outsourcing. npj Genomic Med. 2017;2:33. https://doi.org/10.1038/s41525-017-0036-1.
68. von Thenen N, Ayday E, Cicek AE. Re-Identification of individuals in genomic data-sharing beacons via allele inference. Bioinformatics. 2018;10:43.
69. Ateniese G, Benson K, Hohenberger S. Key-private proxy re-encryption. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer, Berlin, Heidelberg; 2009. p. 279–94. doi:https://doi.org/10.1007/978-3-642-00862-7_19.
70. Chen H, Kim M, Dai W, Song Y. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: Proceedings of the ACM Conference on Computer and Communications Security. New York, NY, USA: Association for Computing Machinery; 2019. p. 395–412. doi:https://doi.org/10.1145/3319535.3363207.
71. Yen JL, Garcia S, Montana A, Harris J, Chervitz S, Morra M, et al. A variant by any name: quantifying annotation discordance across tools and clinical databases. Genome Med. 2017;9:7. https://doi.org/10.1186/s13073-016-0396-7.
72. De Arce AJD, Noderer WL, Wang CL. Complete motif analysis of sequence requirements for translation initiation at non-AUG start codons. Nucleic Acids Res. 2018;46:985–94. https://doi.org/10.1093/nar/gkx1114.
73. McCarthy DJ, Humburg P, Kanapin A, Rivas MA, Gaulton K, Cazier JB, et al. Choice of transcripts and software has a large effect on variant annotation. Genome Med. 2014;6:26.
74. Witten JT, Ule J. Understanding splicing regulation through RNA splicing maps. Trends Genet. 2011;27:89–97.
75. Wangen JR, Green R. Stop codon context influences genome-wide stimulation of termination codon readthrough by aminoglycosides. Elife. 2020. https://doi.org/10.7554/eLife.526119.
76. Harmanci Arif O, Harmanci AS, Tiemo Klisch AJP. XCVATR: characterization of variant impact on the embeddings of single -cell and bulk RNA-sequencing samples. Biorxiv. 2021;3:1.
77. Zerbino DR, Wilder SP, Johnson N, Juettemann T, Flicek PR. The ensembl regulatory build. Genome Biol. 2015. https://doi.org/10.1186/s13059-015-0621-516.
78. Scheibner J, Raisaro JL, Troncoso-Pastoriza JR, Ienca M, Fellay J, Vayena E, et al. Revolutionizing medical data sharing using advanced privacy-enhancing technologies: technical, legal, and ethical synthesis. J Med Internet Res. 2021;23:e25120.
79. De Cristofaro E, Tsudik G. Practical Private Set Intersection Protocols with Linear Computational and Bandwidth Complexity*.
80. Pinkas B, Segev G, Zohner M, Schneider T. Phasing: Private Set Intersection using Permutation-based Hashing. 2015. https://www.eff.org/deeplinks/2012/09/deep. Accessed 16 Jun 2021.
81. Chen H, Laine K, Rindal P. Fast private set intersection from homomorphic encryption
82. Baza M, Salazar A, Mahmoud M, Abdallah M, Akkaya K. On Sharing Models Instead of Data using Mimic learning for Smart Health Applications. 2020 IEEE Int Conf Informatics, IoT, Enabling Technol ICIoT 2020. 2019;:231–6. http://arxiv.org/abs/1912.11210. Accessed 17 Jun 2021.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.