# A Privacy-Preserving Solution for Compressed Storage and Selective Retrieval of Genomic Data

Zhicong Huang[1], Erman Ayday[2], Huang Lin[1], Raeka S. Aiyar[3], Adam Molyneaux[4], Zhenyu Xu[4], Jacques Fellay[5], Lars M. Steinmetz[3, 6], Jean-Pierre Hubaux[1]

[1] *School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne*
[2] *Department of Computer Engineering, Bilkent University*
[3] *Stanford Genome Technology Center, Stanford University*
[4] *Sophia Genetics*
[5] *School of Life Sciences, École Polytechnique Fédérale de Lausanne*
[6] *Department of Genetics, Stanford University School of Medicine*

**Corresponding author**: Jean-Pierre Hubaux (jean-pierre.hubaux@epfl.ch)
**Running title:** Encryption for compressed genomic data
**Keywords**: privacy, encryption, compression, selective retrieval, genomic data

## Abstract
In clinical genomics, the continuous evolution of bioinformatic algorithms and sequencing platforms makes it beneficial to store patients' complete aligned genomic data in addition to variant calls relative to a reference sequence. Due to the large size of human genome sequence data files (varying from 30GB to 200GB depending on coverage), two major challenges facing genomics laboratories are the costs of storage and the efficiency of the initial data processing. In addition, privacy of genomic data is becoming an increasingly serious concern; yet no standard data storage solutions exist that enable compression, encryption, and selective retrieval. Here we present a privacy-preserving solution named SECRAM (*Selective retrieval on Encrypted and Compressed Reference-oriented Alignment Map*), for the secure storage of compressed aligned genomic data. Our solution enables selective retrieval of encrypted data and improves the efficiency of downstream analysis (e.g., variant calling). Compared to BAM, the *de facto* standard for storing aligned genomic data, SECRAM uses 18% less storage. Compared to CRAM, one of the most compressed non-encrypted formats (using 34% less storage than BAM), SECRAM maintains efficient compression and downstream data processing, while allowing for unprecedented levels of security in genomic data storage. Compared to previous work, the distinguishing features of SECRAM are that (i) it is position-based instead of read-based, and (ii) it allows random querying of a sub-region from a BAM-like file in an encrypted form. Our method thus offers a space-saving, privacy-preserving, and effective solution for the storage of clinical genomic data.

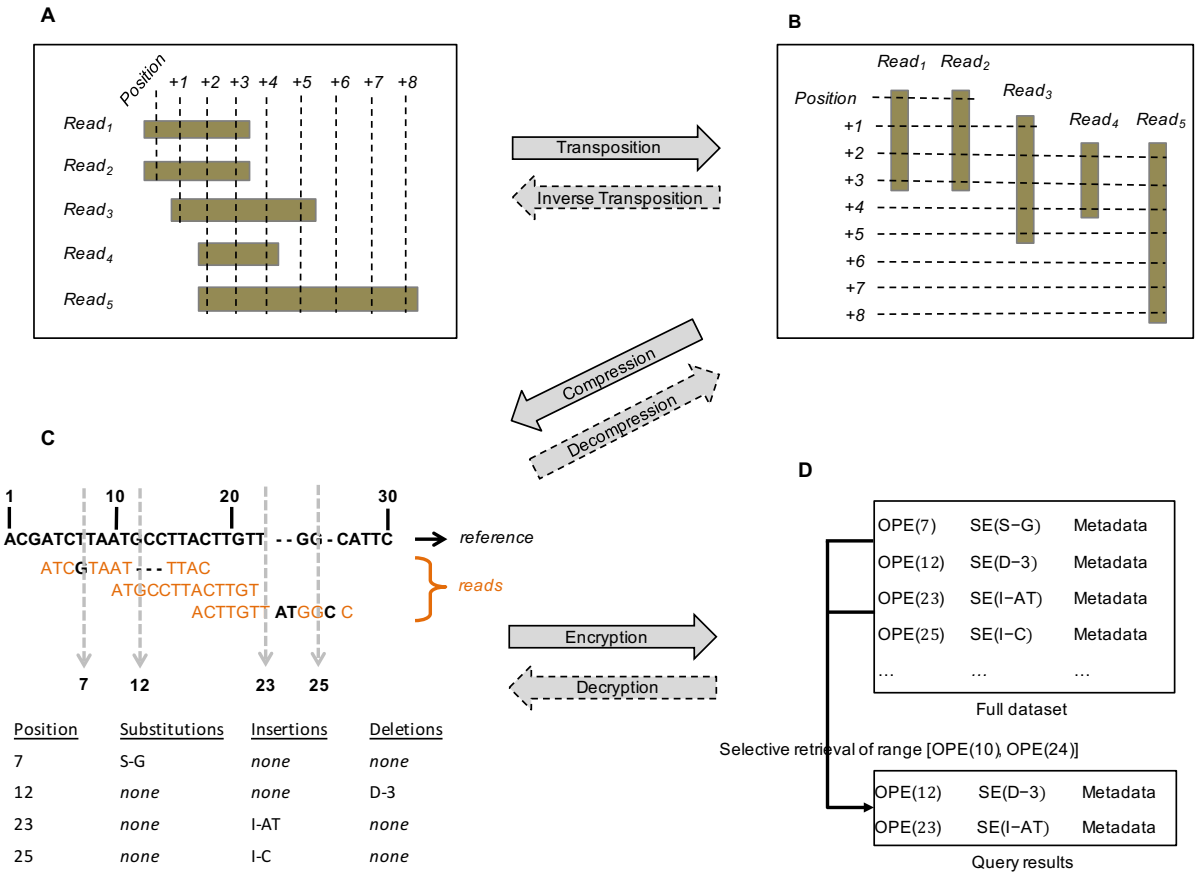
## Introduction
While the generation of genome sequence data is no longer cost-prohibitive, the unprecedented rate of data production presents new challenges for data storage and management. For example, the 1000 Genomes Project generated more data in its first six months than the NCBI GenBank database had accumulated in its 21 years of existence (Pennisi 2011). Sequence data are being more routinely used for diagnostic purposes, which has raised concerns regarding security and privacy. Until recently, it was standard in clinical genetics to screen only 1 or 2 genes for mutations relevant to a specific disease, but high-throughput sequencing technologies have now made whole-genome or whole-exome sequence data commonplace. These comprehensive sequence datasets must then be securely stored and relevant

variants made available to various stakeholders in the healthcare system. Preventing incidental leakage of personal data requires not only data encryption, but also defining data access privileges and enabling selective retrieval of sequencing data. Although some encryption solutions (e.g., in cramtools (www.ebi.ac.uk/ena/software/cram-toolkit)) have been proposed, they remain straightforward applications of encryption standards and do not take into consideration the aforementioned threat model. Addressing these issues of security and privacy while minimizing storage costs will be essential for the large-scale application of personal genomics in research and clinical settings. Here, we describe a solution that minimizes information leakage, stores the sequence data in a lossless compressed format, and optimizes the performance of downstream analysis (e.g., variant calling).

Since 2007, when the first high-throughput sequencing technology was released to the market, the growth rate of genomic data has outpaced Moore's law – more than doubling each year (www.genome.gov/sequencingcosts/). Big data researchers estimate the current worldwide sequencing capacity to exceed 35 petabases per year (Stephens et al. 2015). For every 3 billion bases of human genome sequence, 30-fold more data (about 100 gigabases) must be collected to ensure sufficient coverage at each nucleotide. More than 100 petabytes of storage are already used by the world's largest 20 biological research institutions; this corresponds to more than $1 million USD in storage maintenance costs if we consider Amazon cloud storage pricing (https://aws.amazon.com/s3/pricing/). This number continues to grow, and it is estimated that 2-40 exabytes of storage capacity will be needed by 2025 to store hundreds of thousands of human genomes. To face this challenge, more efficient approaches to genomic data storage are needed.

Current approaches for genomic data storage use different methods for compression (Zhu et al. 2015). Before high-throughput technologies were introduced, algorithms were designed for compressing genomic sequences of relatively small size (e.g., tens of megabases). These algorithms, such as BioCompress (Grumbach and Tahi 1993), GenCompress (Chen et al. 2000), and DNACompress (Chen et al. 2002), exploit the redundancy within DNA sequences and compress the data by identifying highly repetitive subsequences. The latest sequencing technologies pose new challenges for the compression of genomic data in terms of data size and structure. Due to the high similarity of DNA sequences among individuals, it is inefficient to store and transfer a newly assembled genomic sequence in its entirety, because more than 99% of the data for two assembled human genomes are the same. This has led to the approach of storing only differences from a reference sequence (known as reference-based compression), such as the DNAzip algorithm (Christley et al. 2009). Apart from entire assembled sequences, individuals' sequence data are typically organized as millions of short reads of 100 to 400 bases, as produced by state-of-the-art sequencing technologies. Each genomic position is usually covered by multiple short reads. General-purpose compression algorithms, such as gzip (www.gzip.org), are applicable to these datasets. For example, the BAM format (Li et al. 2009), which remains the *de facto* standard for storing aligned short reads, is already highly compressed by applying gzip compression to the data blocks.

Various advanced compression algorithms have been proposed for high-throughput DNA sequence data (Quip (Jones et al. 2012), Samcomp (Bonfield and Mahoney 2013), HUGO (Li et al. 2014), etc.). Among larger datasets (e.g., 1000 Genomes Project), there is an observable trend towards using the highly compressed format CRAM (Fritz et al. 2011), a reference-based compression algorithm for aligned data. Most advanced algorithms also use variable-length encoding techniques, such as Huffman encoding and Golomb encoding, to compress the metadata (read name, position, mapping quality, etc.). Recently, researchers developed a cloud-computing framework called ADAM (Massie et al. 2013), which uses various engineering techniques (e.g., dictionary coding, gzip compression, distributed processing) to reduce storage costs by 25% compared to BAM. This scheme achieves significant (2-10x) acceleration in genomic data access patterns. A group of MIT researchers released a lossy compression algorithm called Quartz (Yu et al. 2015), which can discard 95% of quality scores without compromising the accuracy of downstream analysis. Their method works on the FASTQ file stage and can be plugged in as a pre-processing step for the aforementioned methods (and our solution) to achieve a higher compression ratio. Our solution is a lossless compression method that enables the user to




| Position | Substitutions | Insertions | Deletions |
|---|---|---|---|
| 7 | S-G | *none* | *none* |
| 12 | *none* | *none* | D-3 |
| 23 | *none* | I-AT | *none* |
| 25 | *none* | I-C | *none* |

Figure 1: SECRAM framework for compressed, encrypted storage of genomic data. (A) The sequencing read-based format used in BAM is transposed into (B) a genome position-based format. The read-based format can be reconstructed from the position-based format via reverse transposition. (C) The position-based storage is compressed and decompressed using a reference-based compression technique. In the table, "S-G" stands for substitution with base 'G', "D-3" stands for deletion of 3 bases, and "I-AT" stands for insertion of 2 bases "AT". (D) The compressed position-based storage is encrypted to generate the final SECRAM format using order-preserving encryption (OPE) and traditional symmetric encryption (SE) scheme. "OPE(POSITION)" represents the OPE ciphertext of POSITION, and "SE(VARIANT)" represents the SE ciphertext of VARIANT. Metadata are not encrypted (e.g., quality scores, mapping quality, read name). The compressed format is recovered from the encrypted format by running the respective decryption algorithms. Our encryption enables efficient selective retrieval. For instance, if a user wants to retrieve data in the range [10, 24], the database executes a normal search between OPE(10) and OPE(24) based on the order-preserving property of OPE, and in the shown example, two positions, OPE(12) and OPE(23), are returned.

completely reconstruct the original data (BAM files). To our knowledge, there is no existing compression solution that provides strong security and privacy control, the issue we address in this paper.

Because genomic information is highly personal, privacy has become a major concern as these data become more widely generated, disseminated, and unwillingly exposed (van Dijk et al. 2014; Kahn 2011). For example, coarse-grained encryption and access control to genomic data could lead to incidental findings that doctors often prefer to avoid (Ayday et al. 2013). Standard sample de-identification has been proven insufficient for complete protection of genetic privacy (Erlich and Narayanan 2014). Establishing a secure and privacy-preserving solution for genomic data storage is urgently needed to facilitate the usage and transfer of sequence data. For example, storing sequence data on a cloud is an attractive option, considering the size and the required availability of the data (Onsongo et al. 2014; Rilak et al. 2014; Reid et al. 2014). However, access threats in this case are even more serious because the data owner has to trust insiders on the cloud (e.g., the cloud administrator, or

high-privileged system software). There is thus a need to integrate encryption methods into compression solutions for genomic data that are secure and privacy-preserving (Zhu et al. 2015). The closest example of such a solution (Ayday et al. 2013) provides a privacy-preserving solution for storing BAM files, but it does not provide an efficient method for compression.

For clinical or research purposes, the most valuable information from human genomic data is the set of genetic variants that are identified across the genome. Typically, sequence data are taken as input for a pipeline and retrieved for downstream analyses, e.g., variant calling. Given this usage scenario, it is crucial to have a storage format that is amenable to efficient downstream analyses. For example, it is common practice to aggregate information on a position from all short reads that cover the position (pileup); hence, it is desirable that the storage format organizes information in this manner. In addition, it is challenging to enable selective retrieval of encrypted data because global encryption solutions obfuscate index information. As we demonstrate in this paper, our SECRAM solution is convenient not only for retrieval, but also for protection in that it enables the retrieval of specific information about the genome without compromising the rest of it.

In this paper, we present our genomic data storage solution to address the challenges of compression, security, and retrieval. SECRAM is a novel aligned data storage format that is: (i) organized in position-based storage that enables random queries anywhere in the genome; (ii) highly compressed through a combination of reference-based and general data compression techniques; and (iii) encrypted with standard secure cryptographic techniques a fine-grained privacy control mechanism. By applying our solution, sequence data are securely protected in storage and can be efficiently retrieved for downstream analysis (e.g., variant calling) without any access to unauthorized information. Below, we compare SECRAM with two state-of-the-art storage solutions, BAM and CRAM, which are two of the most widespread formats for aligned sequence data. Our major goal is to bridge the gap between compression and protection of genomic data.

## Results

The SECRAM framework is depicted in Figure 1, including transposition from read-based storage (BAM) to position-based storage, compression, encryption, and retrieval (e.g., for variant calling). Following sequence alignment to a reference genome, the data are transposed, compressed, and encrypted. All these steps are one-time operations for each individual file. Afterwards, the SECRAM format can be queried routinely for data retrieval. The SECRAM source code is openly available at https://github.com/acs6610987/secram and in Supplemental SECRAM Source.

### Storage analysis

In Figure 2-A, we show the compression performance of SECRAM compared to BAM and CRAM, for both paired and unpaired simulated datasets of different coverages and error rates (substitution, insertion, and deletion). Both SECRAM and CRAM formats preserve all information from BAM files, including quality scores, read names, and read-pair information. The number of bits per base is calculated by dividing the size of the file by the total number of bases (roughly equal to reference length multiplied by the coverage). Unsurprisingly, for all three formats, the average per-base storage cost decreases as the coverage increases. In contrast to CRAM, SECRAM does not compress efficiently when the coverage is very low (e.g., 1×) because of the storage overhead of encryption. Yet as coverage increases (e.g., 10×), the benefit of compression becomes more pronounced, mitigating the storage costs of encryption. In the best case, for high-coverage unpaired data with a 0.01% error rate, SECRAM and CRAM have almost identical compression ratios, both using nearly 4 bits/base. We also observed that, in general, compression ratios are lower for paired data with higher error rates, because there is more information (e.g., metadata and variants) in the data. Compared to BAM, SECRAM saves 18% of storage on average, when the coverage is higher than 10×; this is slightly lower than CRAM, which saves 34% of storage but has no privacy and security features. Figure 2-B shows the average storage costs on several randomly selected real data files from the 1000 Genomes Project repository (The 1000

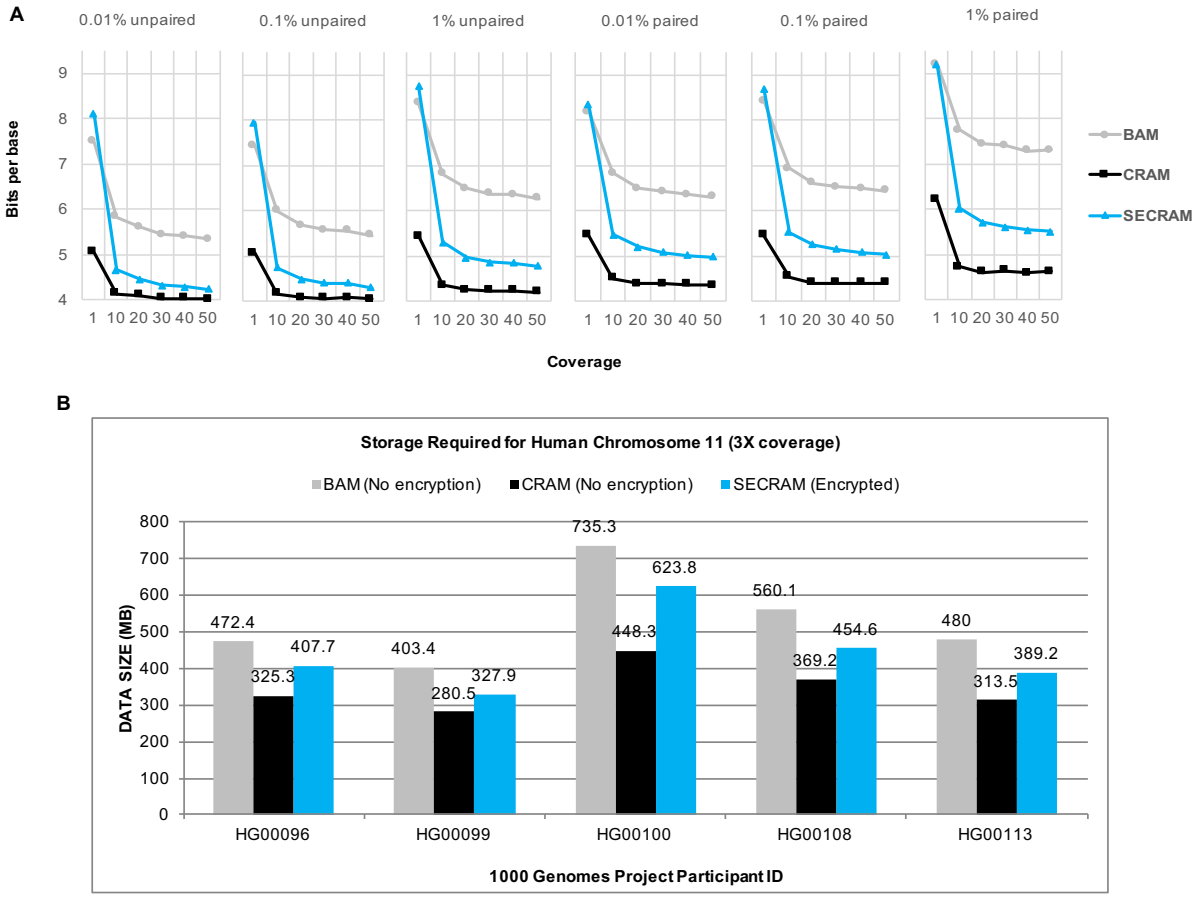



Figure 2: Storage analysis of compression algorithms for genomic data. (A) Storage comparison on simulated datasets. It shows the average number of bits per base (compression ratio) for three storage formats: BAM, CRAM, and SECRAM. The results are based on simulated datasets with different coverages (from 1 to 50) and error rates (0.01%, 0.1%, 1%) for both paired and unpaired data. (B) Storage comparison on Chromosome 11 from 1000 Genomes Project participants with an average coverage of 3X. Only SECRAM is an encrypted format, whereas both BAM and CRAM are in plaintext. Both (A) and (B) show that the compression ratio of SECRAM is between that of BAM and that of CRAM.

Genomes Project Consortium 2015) with an average coverage of 3×. We observed a similar compression ratio as with simulated data in Figure 2-A.

**Runtime analysis**

To assess the runtime efficiency of SECRAM, we considered the performance of its six most important sub-procedures: *transposition*, *inverse transposition*, *compression*, *decompression*, *encryption*, and *decryption*. We ran experiments on simulated datasets with a range of coverages and error rates. Figure 3-A shows the runtime breakdowns in percentages for different sub-procedures relative to total runtime, across four experiments. One important observation is that encryption and decryption are not bottlenecks in the system, demonstrating that our implementation of security does not come at the cost of efficiency. When coverage and error rate are low, compression is slower than encryption and dominates the runtime; this is because there is less information to encrypt, but it still takes time to compress the non-sensitive information. When the error rate is higher, the encryption time surpasses compression time, mainly because of order-preserving encryption on more positions, but the global runtime does not change much even for a high error rate (1%). The high coverage hinders (inverse) transposition more than compression and encryption because the complexity of (inverse) transposition is linearly dependent on the number of total bases. We also observed that decompression is the most efficient sub-procedure; this is because the decompression dictionaries for some compression

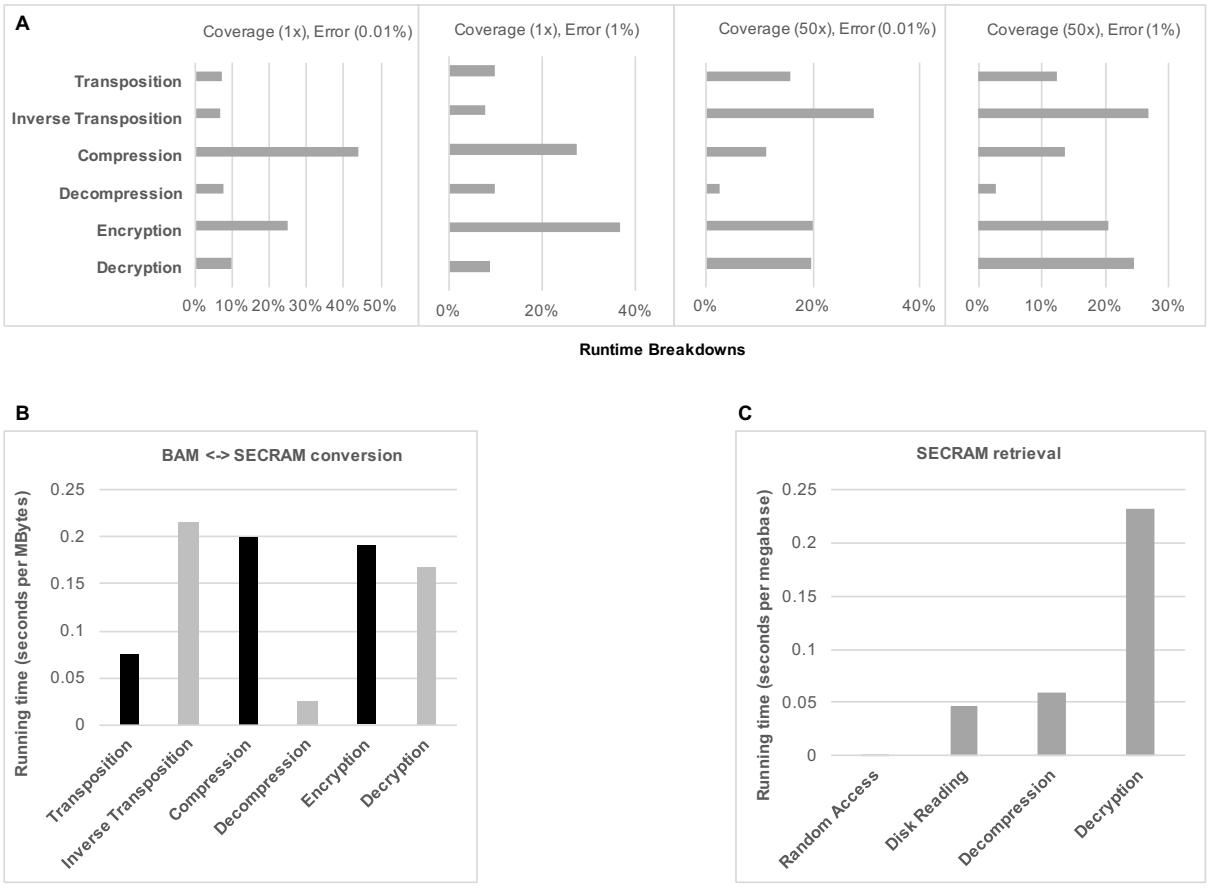



Figure 3: Runtime analysis of SECRAM system for storing and retrieving genomic data. (A) Runtime breakdowns on simulated datasets. It shows the costs for the 6 most important procedures in SECRAM: transposition, inverse transposition, compression, decompression, encryption, and decryption, relative to total runtime (=100%). The experiments were repeated for 4 simulated datasets with low (1x) / high (50x) coverages and low (0.01%) / high (1%) error rates. In all cases, the runtime cost of encryption/decryption is comparable with other necessary procedures, showing that enforcing security does not result in performance overhead. (B) Conversion time on real datasets (same datasets as those in Figure 2-B). It shows the average conversion time between BAM and SECRAM formats on real datasets, running with a single thread on a machine equipped with Mac OS X Yosemite system and 3.1 GHz Intel Core i7 processor. The black bars (transposition, compression, encryption) represent the three steps of conversion from BAM to SECRAM, whereas the light gray bars (decryption, decompression, inverse transposition) represent the three steps of conversion from SECRAM to BAM. Each step takes less than 0.25 seconds per megabyte of data. (C) Retrieval time on real datasets (same datasets as those in Figure 2-B). It shows the average runtime cost of retrieving data within a range of 1 million genomic positions. The actual data size corresponding to 1 million positions depends on the coverage. Shown are experiments on real datasets from Figure 2-B with a coverage of ~3X, and a size of slightly more than 1 megabyte.

algorithms (e.g., Huffman encoding) are stored with SECRAM and can be directly used, whereas compression has to build these dictionaries.

Figure 3-B shows the runtime costs of conversion between BAM and SECRAM on real data sets as used in Figure 2-B. The conversion from BAM to SECRAM consists of three procedures (transposition, compression, and encryption), whereas the conversion from SECRAM to BAM encompasses the remaining three (inverse transposition, decompression, and decryption). On average, to process one megabyte of data, each procedure takes less than 0.25 seconds; in total, the conversion in either direction takes less than 0.5 seconds. We should emphasize that our current implementation is parallelizable (i.e., numerous data files or chromosomes can be processed in parallel). For instance, a whole-genome BAM

file of 100 gigabytes can be converted to SECRAM in parallel with 24 threads (one thread for each reference chromosome: 22 autosomes plus 2 sex chromosomes) in approximately one hour.

So far, we have analyzed the runtime costs of converting a whole BAM file or a whole SECRAM file, which is a one-time cost. For most applications, conversion is not necessary, but selective retrieval is crucial for efficiency. We designed this new format so that laboratories need only to store the inherently encrypted, compressed SECRAM files and develop analysis pipelines for accessing the data directly from these files. Therefore, it is critical to evaluate the efficiency of the SECRAM format for data retrieval. Figure 3-C shows the average runtime performance of random access of a data region of one megabase (the cost scales linearly with the size of the region). The total time for the retrieval procedure is less than 0.5 seconds, which is efficient for real-time usage in any analysis pipeline. In traditional use, without any security measures, data retrieval consists of at least three steps (*random access*, *disk reading*, and *decompression*); *decryption* roughly doubles the runtime overhead, which we believe to be acceptable, given the strong privacy and security features.

**Security and privacy analysis**

Two encryption schemes are used in SECRAM (see the Methods section for detailed explanation). The symmetric encryption (SE) scheme used in SECRAM provides semantic security that is a standard and strong security guarantee in cryptography. Order-preserving encryption (OPE) is less secure than SE (see Discussion). In SECRAM format, information is organized by genomic position. Therefore, during retrieval, the storage server can search the encrypted data (due to the properties of OPE) to locate the exact range of data in the query. The stream cipher mode enables the server to restrict the decryption key stream to only the information within the query range; hence, private information about other positions is protected from clients who send the query. This is in contrast to applying straightforward encryption techniques to read-based data, which risks information leakage from overlapping reads with each retrieval. Therefore, the position-based storage used in SECRAM is crucial for its fine-tuned data protection and privacy control.

**Real case study**

We envision SECRAM as most useful when deployed for routine clinical care. There are two phases that typically comprise the clinical application of genomic data: (i) variant calling: A hospital outsources a large volume of sequence data to the cloud and delegates variant annotation to a specialized bioinformatics company (e.g., Sophia Genetics); (ii) mutation querying: A clinician in the hospital queries for hotspot mutations for precise diagnosis of a patient. In the first phase, the hospital should restrict the company's access to a predefined set of genes for variant calling in order to respect patient privacy, whereas in the second phase, a clinician should only access the genes that are related to their area of specialty. Both phases require the property of selective retrieval on encrypted sequence data on the cloud, and ideally without having the hospital frequently handle access control on data requests. Typically, the data corresponding to a gene (or a set of genes) are accessed twice, necessarily in the first phase of variant calling and potentially in the second phase of mutation querying if the clinician requires verification of raw sequence data.

As a case study, we evaluate the storage and retrieval performance by using high-coverage sequence data in clinical care: specifically, the diagnosis of Cystic Fibrosis associated with the *CFTR* gene (roughly between positions 117120017 and 117308719 on Chromosome 7 of the reference sequence GRCh37). The sequence data come from a public cell line (NA12878), based on a gene panel that includes the *CFTR* gene. The maximum coverage is 10642×, whereas the average coverage is 1035×. In addition, we compare SECRAM with another solution that we call ARHMH2013 (Ayday et al. 2013) that provides similar security and privacy properties for BAM files. In Figure 4-A, we observe a consistent compression performance of SECRAM compared to the low-coverage results in Figure 2; however, the ARHMH2013 approach does not scale well with high-coverage data due to the lack of

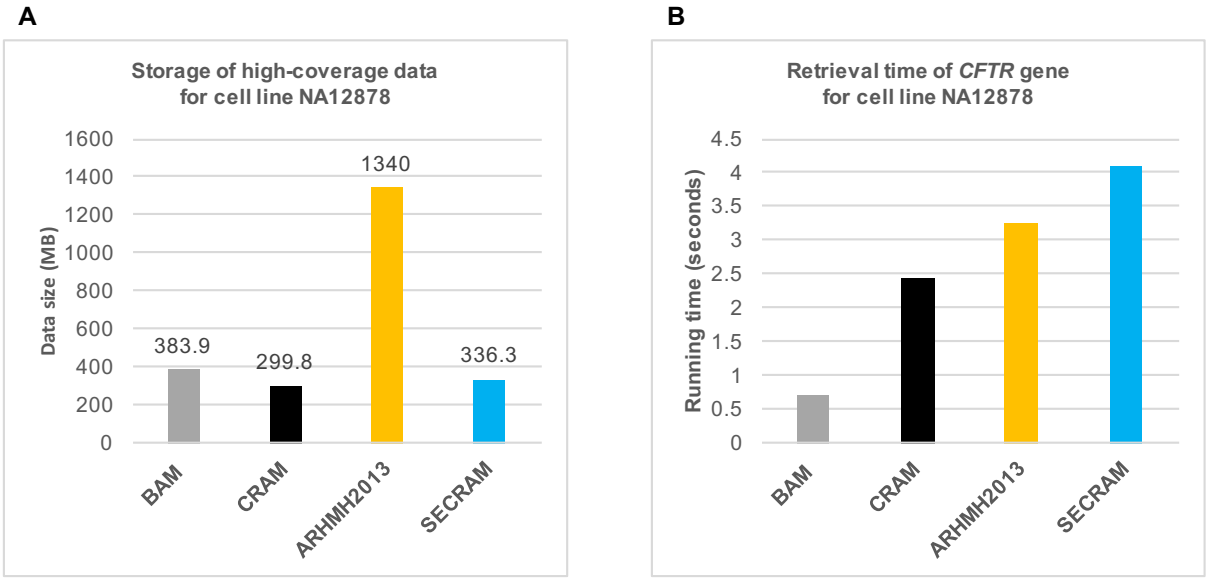



Figure 4: Storage and runtime on high-coverage clinical data. The data come from a public cell line (NA12878), based on a gene panel that includes the CFTR gene, queried for diagnosing Cystic Fibrosis. The average coverage of this data is 1035×, containing more than 4 million reads of length around 300. ARHMH2013 (Ayday et al. 2013) is a privacy-preserving solution on BAM files which does not address the compression requirement. We observe a consistent compression performance of SECRAM on high-coverage clinical sequence data. Moreover, querying for the CFTR gene on SECRAM takes less than twice the time on the non-encrypted CRAM.

compression and the overhead of encryption. In Figure 4-B, we also see that SECRAM performs very well in selective retrieval, i.e., less than twice the time needed when using non-encrypted CRAM. We should emphasize that with SECRAM, one can query for a small region of data, while standard encryption approaches necessitate downloading the entire genome – making SECRAM more efficient as well as privacy-preserving.

## Discussion

In this work, we present a novel, position-based, compressed, encrypted format for storing genomic data that enables selective and secure retrieval of variant information. We demonstrate that our solution compresses data more effectively than widely used formats like BAM. Importantly, it offers a level of security not possible with existing standard storage formats, preventing the data leakage to which read-based formats are susceptible, without slowing down data retrieval or analysis. We provide algorithms for conversion to and from BAM/read-based formats, compression, decompression, encryption, and decryption.

### Attacks against order-preserving encryption

As described below, the ciphertext of OPE contains both order and equality information about the underlying data. Therefore, assuming that an attacker has enough information, OPE-encrypted data are vulnerable to some well-known attacks that exploit this property, such as frequency analysis attacks (Boldyreva et al. 2011; Kolesnikov and Shikfa 2012; Naveed et al. 2015). However, these frequency-analysis attacks would not be effective against our scheme, because (i) positions are encrypted only once regardless of the coverage, because all information about each position is clustered together in the transposition phase of SECRAM; (ii) we use different OPE keys for data from different individuals; and (iii) the underlying message in our adopted OPE scheme is usually a subset of the whole position space of genomic data, and it matches the requirement of the OPE scheme for message unpredictability. Regarding the third reason, however, the level of data protection depends on the data, more specifically, on the distribution of genetic variants and sequencing errors across the genome. If average coverage and sequencing error rate are high enough such that there is at least one different base from the reference genome on almost every position, SECRAM protection with OPE will be less effective. In this case,

OPE would have to encrypt the whole plaintext space and thus provide little protection. Compromise of position information also leads to potential threats for the content encrypted by symmetric encryption. For example, if positions are known and coverage can be inferred from the encrypted content size, CNV (copy number variation) analysis can identify deletions / insertions without the need for the sequence itself, and other single nucleotide variants can be inferred by observing peaks of encrypted information size on the compromised positions. Nevertheless, a countermeasure would be to shuffle the OPE ciphertext (Ayday et al. 2013), which is essentially a second-level encryption and reduces the efficiency of retrieval. An alternative countermeasure is to add random padding to each position to obfuscate the size of encrypted content.

The purpose of adopting OPE as a building block is to maximize the difficulty of launching a successful attack against the encryption scheme while still allowing for efficient retrieval from the ciphertext. To enable efficient retrieval, the order information must be observable from the ciphertext, and under this premise OPE is the optimal available solution with the best security guarantee, as the database community has recognized long ago (Agrawal et al. 2004). We do not rule out the possibility that there could be other variants of frequency-analysis attacks against the OPE scheme adopted in this work; however, there already exist several enhanced OPE schemes (Chenette et al. 2015; Kerschbaum 2015; Roche et al. 2015) as potential replacements of the current version. Another alternative to OPE would be secure multiparty computation (SMC), for example, SMC schemes based on homomorphic encryption. Considering the high volume of data in the problem addressed here, however, SMC would introduce a prohibitively high overhead to storage and retrieval.

**Lossy compression**

In the open-source implementation of CRAM (www.ebi.ac.uk/ena/software/cram-toolkit), multiple lossy compression options are provided, such as removing read names and reducing the precision of quality scores. These options are also made available in our solution. For instance, we noticed that quality scores account for most of the storage space. We provide a parameter to specify the precision of quality scores; the default option maintains the original scores to allow exact reconstruction of the BAM file. A compression algorithm called Quartz (Yu et al. 2015) can be used to discard 95% of quality scores without affecting the accuracy of downstream analysis. Although their solution requires further validation, it indicates that lossy compression is a possibility for reducing storage requirements without jeopardizing the value of the data.

**Datatype applicability**

SECRAM is designed to be compatible with SAM/BAM files; hence it enables, or is extensible to, a variety of sequence data that can be handled by SAM/BAM. These include (but are not limited to):

- Multiple mapping. A read can be mapped to multiple alignments, e.g., due to repetition. One of these alignments is considered *primary*, whereas the other alignments are annotated as *secondary* and have a link to the primary alignment. In BAM files, the read bases and quality scores of secondary alignments are set to "empty" to reduce the file size. SECRAM can be extended similarly to nullify bases and quality scores of the corresponding positions of secondary alignments, and to include a link to the primary alignment in the read headers (see Supplemental Methods) of these secondary alignments.
- Long reads. Several high-throughput sequencing systems produce long reads (e.g., the Sequel system (http://www.pacb.com/products-and-services/pacbio-systems/sequel/), with read length 10kb - 15kb), for which SECRAM's prevention of read-based information leakage is even more valuable. Similar to CRAM, SECRAM is expected to have a slightly higher compression ratio with longer reads, assuming the same coverage depth, because there is less read information (e.g., read name). However, this improvement is modest because the majority of storage (more than 80% in our data) is occupied by read bases and quality scores.
- RNA-seq data. RNA sequence data are stored in the same way as DNA sequence data in BAM formats, and hence can be processed in essentially the same manner by SECRAM.

# Methods

**Transposition**
Aligned genomic sequence data are typically stored by sequencing reads (e.g., BAM, CRAM formats; Figure 1-A). The first step of our solution is to organize the data by position instead of by read (Figure 1-B). This is crucial because (i) it facilitates multiple downstream analysis procedures, notably the variant calling pipeline; and (ii) it allows us to seamlessly combine compression and encryption (fine-grained privacy control). The conversion between the two formats is equivalent to a matrix transposition, when we consider the index of the reads as one dimension and the position of the reference genome as the other dimension. If necessary, our format can be inversely transposed to BAM without losing information; this functionality makes our format compatible with several other applications designed for a read-based format, such as visually displaying reads that overlap with a specific genomic region. The transposition algorithms (from BAM to SECRAM, and from SECRAM to BAM) are provided in Supplemental Methods.

**Compression**
Our algorithm takes advantage of several efficient compression methods used in CRAM:

- Reference-based compression. As shown in Figure 1-C, we use reference-based compression and store only the differences at each position relative to that of a chosen reference sequence.
- Variant-length encoding (VLC). This technique is used to further compress the differences found in reference-based compression along with read metadata, such as the mapping quality. Depending on the information content, the most efficient encoding technique is selected, e.g., Huffman, Golomb and Beta encodings (www.ebi.ac.uk/ena/software/cram-toolkit).
- Block compression. After the previous compression phases, positions are grouped into blocks and compressed with gzip. Blocks are gzipped separately, so that they can be decompressed independently for fast random access. This phase is important for the compression of information such as read name, quality score, and other auxiliary text.

**Encryption**
SECRAM protects genomic data using secure symmetric encryption techniques. To provide fine-grained privacy control and avoid information leakage in data retrieval, our solution encrypts the variant information for each position independently. We encrypt positions with order-preserving encryption (Boldyreva et al. 2011) to enable efficient retrieval of encrypted data without decryption. This is a critical feature to prevent insider attacks on servers that store the encrypted data. We encrypt other sensitive information (e.g., short read differences relative to the reference) using conventional symmetric encryption (SE). Notably, encryption adds randomness to the data, which inevitably affects the compression ratio. We explain the basics of these two encryption schemes below.

**Symmetric encryption (SE)**. A symmetric encryption scheme usually consists of an encryption and a decryption algorithm. The encryption algorithm takes a message and an encryption key to generate an encrypted message, i.e., ciphertext. Using the ciphertext and a decryption key as input, the decryption algorithm can generate the original message. In a symmetric encryption scheme, the decryption can only be successful when the encryption and decryption keys are the same. To draw an analogy, the key in this scheme is like a physical key that can be used to lock (encrypt) and unlock (decrypt) a box containing a private message sent via an insecure channel. The essential feature of the symmetric encryption scheme follows: for the receiver to successfully unlock the box, the sender has to share a key copy with the receiver.
In SECRAM, we make use of the stream cipher mode (Lipmaa et al. 2000) of symmetric encryption, which is done in two steps: 1) by using the encryption key, the data sender generates a random bit stream of identical length to the data; and 2) the encryption is performed via bitwise XOR of the stream with the data. The decryption is performed similarly: using the decryption key, the ciphertext receiver first generates the bit stream as the sender does, and then decrypts the ciphertext via bitwise XOR of the ciphertext with the stream.

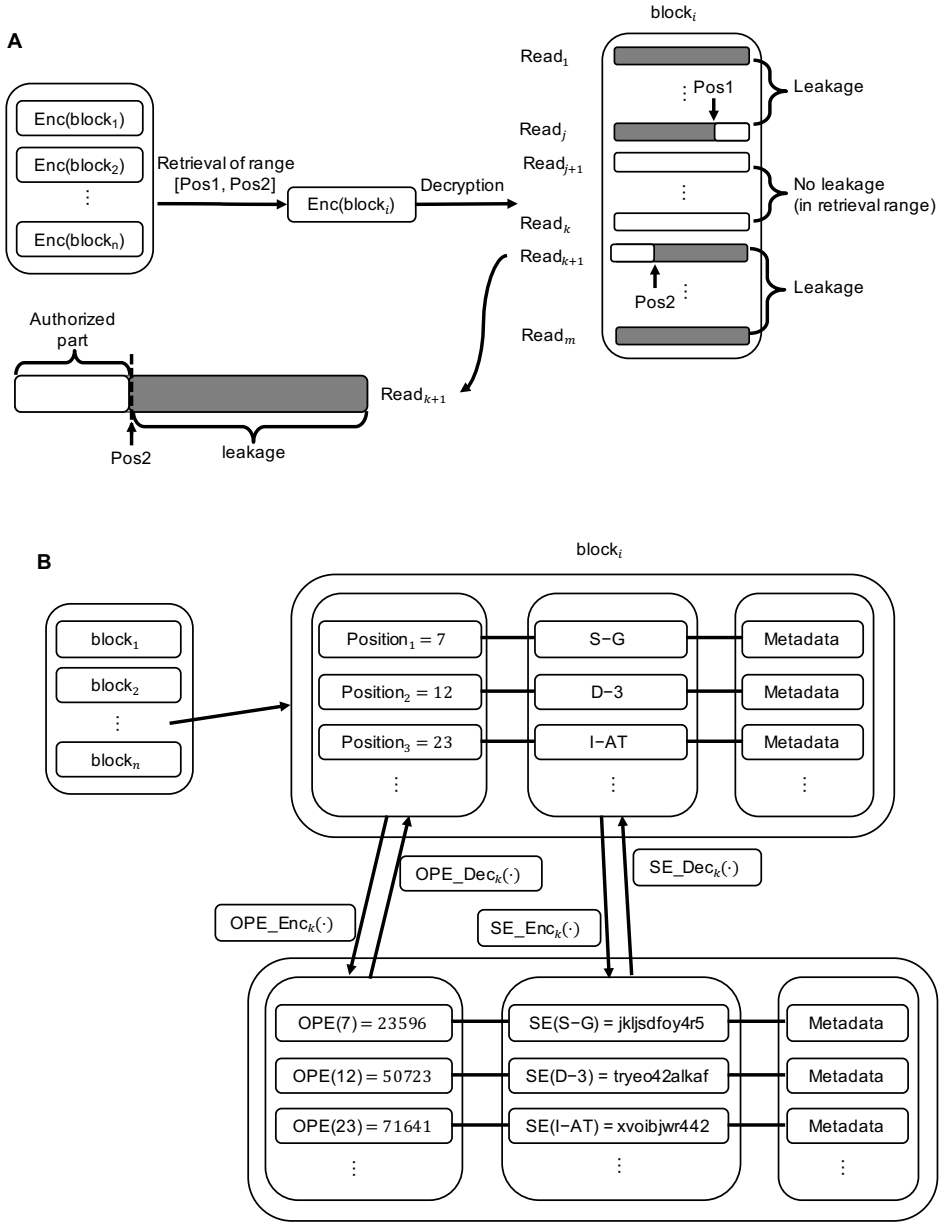



Figure 5: Solutions for encrypting genomic data. (A) A standard symmetric encryption solution on CRAM-compressed data. It leads to potential data leakage when querying specific genomic regions. The encryption is performed over each individual data block. As a block in CRAM usually contains multiple sequencing reads, it is usually the case that the retrieved block will reveal reads or positions that are not in the query position range. (B) SECRAM encryption. Our solution encrypts each block in SECRAM format based on positions. Position information is encrypted with OPE, and the compressed content at each position with SE. This ensures that only information corresponding to the query position range is retrieved and decrypted. "OPE(POSITION)" represents the OPE ciphertext of POSITION, and "SE(VARIANT)" represents the SE ciphertext of VARIANT. Metadata are not encrypted (e.g., quality scores, mapping quality, read name). Note that OPE preserves the order of the positions, i.e., 23596 < 50723 < 71641 because 7 < 12 < 23; but SE encrypts the original message to a random string, e.g., from "S-G" to "jkljsdfoy4r5".

**Order-preserving encryption (OPE)**. In a traditional symmetric encryption scheme, to an observer who does not have the encryption key, the ciphertext is indistinguishable from a random bit stream. Therefore, the ciphertext does not reveal any useful information about the underlying data. OPE differs from traditional symmetric encryption, however, in that the OPE ciphertext reveals both the order and equality information of the underlying data, i.e., if $m_1 \leq m_2$, then $\mathrm{OPE}(m_1) \leq \mathrm{OPE}(m_2)$.

To understand the encryption design of SECRAM, it is important to understand the challenges of securing existing formats (e.g., CRAM). Consider a straightforward, blockwise, symmetric encryption solution for CRAM-formatted data (Figure 5-A). The problem is that the solution leaks information on other positions outside of the query range during retrieval and decryption. As a block in CRAM usually contains multiple reads, it is usually the case in practice that the retrieved block reveals parts of reads outside of the query position range (Ayday et al. 2013). Even if encryption is performed readwise, the possibility remains that a single read can leak information on positions outside of the query range.

Figure 5-B shows how encryption is applied in SECRAM, with each block encrypted based on positions. We encrypt the position information using OPE, and the compressed sensitive content at the respective positions using SE. Therefore, during retrieval, our scheme only outputs the sequence and metadata in the query range without the risk of revealing any information about undesired (or unauthorized) positions.

**Metadata**. The SECRAM format contains all necessary information to enable reconstruction of the original BAM files. This is achieved with the metadata field at each position. This field is not encrypted and contains two categories of information: (i) quality scores, and (ii) information about reads that begin at that position. The first category is a numerical score (the phred-scaled base error probability: $-10\log_{10}\Pr\{\text{base is wrong}\}$), while the second has a slightly more complicated structure. For each read that begins at position $P$, the metadata for $P$ will store the following information: read name, read length, mapping quality, and pair information.

**Selective Retrieval**

**Indexing**. The indexing of aligned genomic data is a map from positions to file offsets such that, given queried positions, a data reader can randomly access the file. Indexing is easy to implement in SECRAM. Our index file contains a list of tuples of *position* and *file offset*, where *position* is the genomic position of the first position row in a gzip block, and *file offset* is the byte offset of a gzip block in the compressed file. Hence, when we retrieve data for a position, a binary search first locates the gzip block containing the position, and then a linear search locates the position in the block.

**Locating sequence read information**. In most cases, a query range is partially overlapped by some reads, e.g., when the starting position of a read is before the starting position of a query. Because the metadata (read name, mapping quality, read flags, etc.) are stored only at the starting position of a read, the server must trace back to that position after locating the starting position of the query. Consider the query range $[P_1, P_2]$. If the position row $P_1$ contains any read that is not complete in block $B_i$ (specifically, it does not start at this block), the storage server traces back to the previous block(s) and crops the corresponding metadata fields for the incomplete reads of row $P_1$. As a block normally contains thousands of positions, the server usually needs to look back to at most one previous block, as long as the reads are not excessively long (e.g., in our experiments, we use a block size of 50000 positions). The server then returns these metadata fields along with the complete sequence data (only) within the query range.

**Implementation**

For processing BAM files and applying VLC and block compression techniques, SECRAM uses the open-source Java library, HTSJDK (https://samtools.github.io/htsjdk/), designed for high-throughput sequencing data. For encryption, SECRAM builds its security solution based on the open-source Java library, Bouncy Castle (https://www.bouncycastle.org/). Depending on the category of information, SECRAM chooses appropriate compression methods to achieve high compression ratio (Table 1).

Overall, our solution addresses the pressing issues of data storage and security brought about by advances in sequencing technology and the emergence of personal and clinical genomics. By bridging the persistent gap between compression and security in the storage of genomic data, SECRAM offers an effective balance between the needs of researchers for efficient data analysis and the needs of individuals to maintain their genetic privacy. It will be important to continuously reevaluate the

standards of genomic data storage as novel technologies are developed, security threats arise, and more complex phenotypic analyses become possible. Integrative solutions that carefully consider the use and misuse of personal genomic data are essential for ensuring its secure, effective storage and maximizing its utility in treating and preventing disease.

Table 1: Compression and encryption methods for different types of information in SECRAM.

| Information type | Compression / encryption method |
|---|---|
| Position | Step 1: Order-preserving encryption |
| | Step 2: Gzip block compression |
| Read bases | Step 1: Reference-based compression and VLC compression |
| | Step 2: AES encryption in CTR mode* |
| Quality scores | Gzip block compression (optional lossy compression) |
| Read name | Gzip block compression |
| Mapping quality | VLC compression |
| Template length | Gzip block compression |
| Flag (e.g., strand) | VLC compression |
| Auxiliary text (e.g., tags in BAM) | Gzip block compression |

Only positions and read bases are encrypted, with OPE and SE respectively. Note that positions are encrypted with OPE before block compression in order to enable indexing. Block compression is chosen for the information that has a wide variety of values, or for text information, or merely as a default way to save storage.
* AES encryption in CTR mode stands for *advanced encryption standard in counter mode*, which is one way of applying a block cipher in a stream-cipher mode (essentially equivalent to a stream cipher).

## Data access

Our simulated dataset is available in Supplemental Simulated Data. Our high-coverage data for cell line NA12878 are available in Supplemental Data NA12878.

## Software availability

SECRAM includes a software application implemented in Java, and is open source at https://github.com/acs6610987/secram. It is also available in Supplemental SECRAM Source.


## Acknowledgements

We thank Jean Louis Raisaro, Jesus Garcia and Adam Novak for thoughtful comments and discussion. We also thank Linus Gasser and Khoffi Ismail for checking the source code. This work is funded by the Swiss Commission for Technology and Innovation (CTI).


## Disclosure declaration

A.M. and Z.X. are employees of Sophia Genetics, a company that does visualization, interpretation and storage of genomic data. A.M., Z.X., L.M.S. and J.P.H. hold stock in Sophia Genetics. L.M.S. is co-founder and chair of the scientific advisory board (SAB). J.P.H. is a member of the SAB.

Z.H., E.A., H.L., A.M. and J.P.H. have filed a provisional European patent on 9 March 2016 entitled "Methods to compress, encrypt and retrieve genomic alignment data". Application number: EP16159314.0. Sophia Genetics has an exclusive license (granted by EPFL) for the commercial use of this patent.

## References

Agrawal R, Kiernan J, Srikant R, Xu Y. 2004. Order Preserving Encryption for Numeric Data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, *SIGMOD '04*, pp. 563–574, ACM, New York, NY, USA.

Ayday E, Raisaro JL, Hengartner U, Molyneaux A, Hubaux J-P. 2013. Privacy-Preserving Processing of Raw Genomic Data. In *8th International Workshop on Data Privacy Management and Autonomous Spontaneous Security*, pp. 133–147, New York, NY, USA.

Boldyreva A, Chenette N, O'Neill A. 2011. Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. In *Advances in Cryptology – CRYPTO 2011*, pp. 578–595.

Bonfield JK, Mahoney MV. 2013. Compression of FASTQ and SAM Format Sequencing Data. *PLoS ONE* **8**: e59190.

Chenette N, Lewi K, Weis SA, Wu DJ. 2015. *Practical Order-Revealing Encryption with Limited Leakage*. https://eprint.iacr.org/2015/1125 (Accessed December 18, 2015).

Chen X, Kwong S, Li M. 2000. A Compression Algorithm for DNA Sequences and Its Applications in Genome Comparison. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, ACM, New York, NY, USA.

Chen X, Li M, Ma B, Tromp J. 2002. DNACompress: fast and effective DNA sequence compression. *Bioinformatics* **18**: 1696–1698.

Christley S, Lu Y, Li C, Xie X. 2009. Human genomes as email attachments. *Bioinformatics* **25**: 274–275.

Erlich Y, Narayanan A. 2014. Routes for breaching and protecting genetic privacy. *Nat Rev Genet* **15**: 409–421.

Fritz MH-Y, Leinonen R, Cochrane G, Birney E. 2011. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Res* **21**: 734–740.

Grumbach S, Tahi F. 1993. Compression of DNA sequences. In *Data Compression Conference, 1993. DCC '93.*, pp. 340–350.

Jones DC, Ruzzo WL, Peng X, Katze MG. 2012. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Res* gks754.

Kahn SD. 2011. On the Future of Genomic Data. *Science* **331**: 728–729.

Kerschbaum F. 2015. Frequency-Hiding Order-Preserving Encryption. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, *CCS '15*, pp. 656–667, ACM, New York, NY, USA.

Kolesnikov V, Shikfa A. 2012. On The Limits of Privacy Provided by Order-Preserving Encryption. *Bell Labs Tech J* **17**: 135–146.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, Subgroup 1000 Genome Project Data Processing. 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics* **25**: 2078–2079.

Li P, Jiang X, Wang S, Kim J, Xiong H, Ohno-Machado L. 2014. HUGO: Hierarchical mUlti-reference Genome cOmpression for aligned reads. *J Am Med Inform Assoc* **21**: 363–373.

Lipmaa H, Wagner D, Rogaway P. 2000. *Comments to NIST concerning AES modes of operation: CTR-mode encryption*.

Massie M, Nothaft F, Hartl C, Kozanitis C, Schumacher A, Joseph AD, Patterson DA. 2013. Adam: Genomics formats and processing patterns for cloud scale computing. *EECS Dep Univ Calif Berkeley Tech Rep UCBEECS-2013-207*.

Naveed M, Kamara S, Wright CV. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, *CCS '15*, pp. 644–655, ACM, New York, NY, USA.

Onsongo G, Erdmann J, Spears MD, Chilton J, Beckman KB, Hauge A, Yohe S, Schomaker M, Bower M, Silverstein KAT, et al. 2014. Implementation of Cloud based Next Generation Sequencing data analysis in a clinical laboratory. *BMC Res Notes* **7**: 314.

Pennisi E. 2011. Will Computers Crash Genomics? *Science* **331**: 666–668.

Reid JG, Carroll A, Veeraraghavan N, Dahdouli M, Sundquist A, English A, Bainbridge M, White S, Salerno W, Buhay C, et al. 2014. Launching genomics into the cloud: deployment of Mercury, a next generation sequence analysis pipeline. *BMC Bioinformatics* **15**: 30.

Rilak Z, Wernicke S, Bogicevic I. 2014. Keeping Genomic Data Safe on the Cloud. *J Biomol Tech JBT* **25**: S5.

Roche D, Apon D, Choi SG, Yerukhimovich A. 2015. *POPE: Partial Order-Preserving Encoding*. https://eprint.iacr.org/2015/1106 (Accessed December 18, 2015).

Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, Iyer R, Schatz MC, Sinha S, Robinson GE. 2015. Big Data: Astronomical or Genomical? *PLoS Biol* **13**: e1002195.

The 1000 Genomes Project Consortium. 2015. A global reference for human genetic variation. *Nature* **526**: 68–74.

van Dijk EL, Auger H, Jaszczyszyn Y, Thermes C. 2014. Ten years of next-generation sequencing technology. *Trends Genet* **30**: 418–426.

Yu YW, Yorukoglu D, Peng J, Berger B. 2015. Quality score compression improves genotyping accuracy. *Nat Biotechnol* **33**: 240–243.

Zhu Z, Zhang Y, Ji Z, He S, Yang X. 2015. High-throughput DNA sequence data compression. *Brief Bioinform* **16**: 1–15.