# Supplemental Material:
# 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*

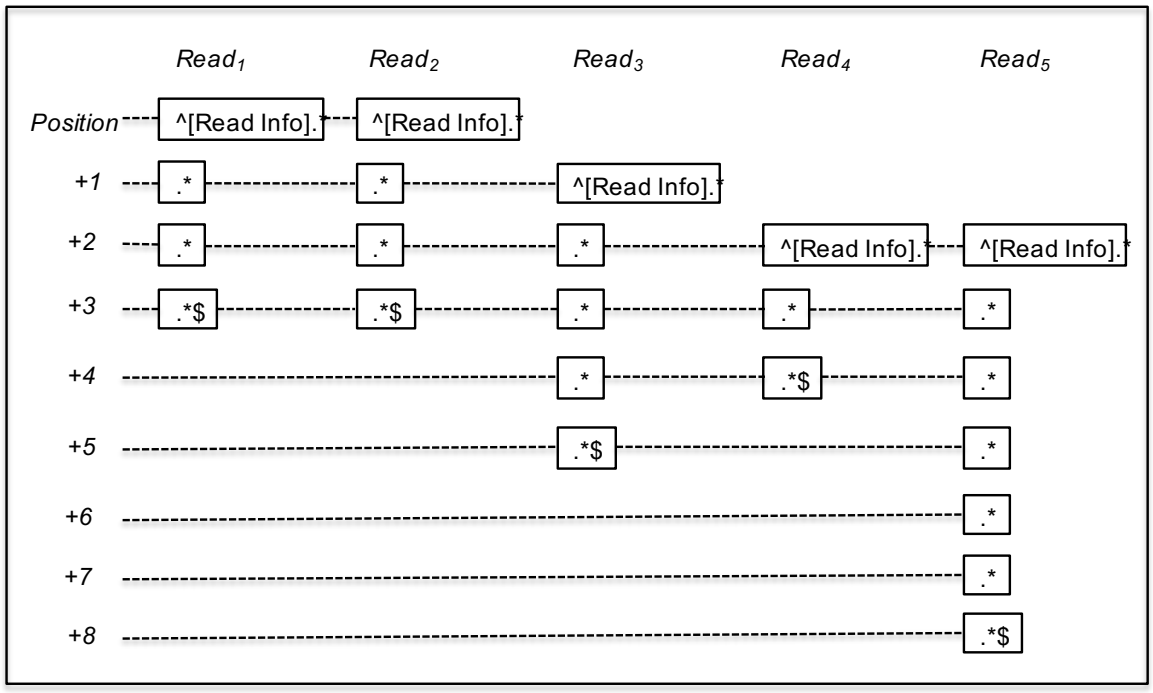
This supplement is organized as follows: Section S1 describes the SECRAM data format structure in detail. Section S2 introduces our proposed reference-based compression technique. Section S3 provides an introduction to the underlying encryption technique of SECRAM, and Section S4 demonstrates how efficient information retrieval can be performed.

## S1. FORMAT STRUCTURE

Supplemental Figure S1 describes the detailed structure of SECRAM in the manner of regular expressions. We denote the important notations in Supplemental Figure S1 as follows:

- '^' is the start marker of a read
- '$' is the end marker of a read
- '.*' denotes the base information at this position.
- '[Read Info]' contains information about this read, such as read name (or id), strand, mapping quality, etc.

If necessary, our format can be converted back to a read-oriented version without losing information; this functionality makes our format compatible with several other applications that suit the read-oriented format well, such as displaying reads overlapping with a region in a viewer. The algorithm for the conversion is detailed in Supplemental Figure S2 (assuming each read is complete in the input, specifically, with start and end markers).




Supplemental Figure S1: The basic structure of SECRAM. '^' is the start marker of a read. '$' is the end marker of a read. '.*' denotes the base information at this position. '[Read Info]' contains information regarding this read, such as read name (or id), strand, mapping quality.

```
INPUT: list of position rows [P1, P2, ..., Pn]
OUTPUT: list of reads [R1, R2, ..., Rm]
read_queue = []
for i = 1 to n do
    len <- number of base entries in Pi
    idx <- 1
    for j = 1 to len do
        if Pi[j] has a start marker then
            creat a new read R with the infomation in Pi[j]
            insert R to read_queue at position idx
            idx <- idx + 1
        elseif Pi[j] has an end marker then
            append the last base to the read read_queue[idx]
            output read_queue[idx]
            delete read_queue[idx] from the queue
        else
            append the base Pi[j] to the read read_queue[idx]
            idx <- idx + 1
        endif
    endfor
endfor
```

Supplemental Figure S2: Conversion algorithm from SECRAM to a read-oriented format (e.g. BAM).

# S2. REFERENCE-BASED COMPRESSION

The basic structure in Supplemental Figure S1 does not take the redundancy of short reads into account, as a large portion of most reads are likely to match the reference. Reference-based compression is applied in most compression methods. To explain how we apply the idea in SECRAM, we make use of a new concept called *position cigar string*, PosCigar for short. For a given position, if we imagine that the reads that cover the position are ordered by their starting positions, then each read is attached with a unique order. Note that one read could have different orders for different positions because the corresponding lists of covered reads in those positions vary. Given a position, a PosCigar is composed of mainly three possibilities:

1. Order ‖ 'S' ‖ [A|T|C|G] : the read (specified by Order) has a substitution with the specified letter compared to the reference.
2. Order ‖ 'I' ‖ $i$ ‖ $\{A,T,C,G\}^i$ : the read has an insertion of $i$ letters that are listed.
3. Order ‖ 'D' : the read has a deletion.

For example, a PosCigar that looks like "9I4ATTG…23SA…57D", means:

- 9I4ATTG: an insertion of 4 letters "ATTG" ***in the 9$^{th}$ read***
- 23SA: a substitution with letter 'A' in the 23$^{rd}$ read
- 57D: a deletion in the 57$^{th}$ read

More operators (e.g., soft clipping, hard clipping, skipping region (Li et al. 2009)) are also encoded in SECRAM, and we omit the redundant details here because each of them is handled similarly as one of the three aforementioned operators. As one possible design option, each position row can be encoded as:

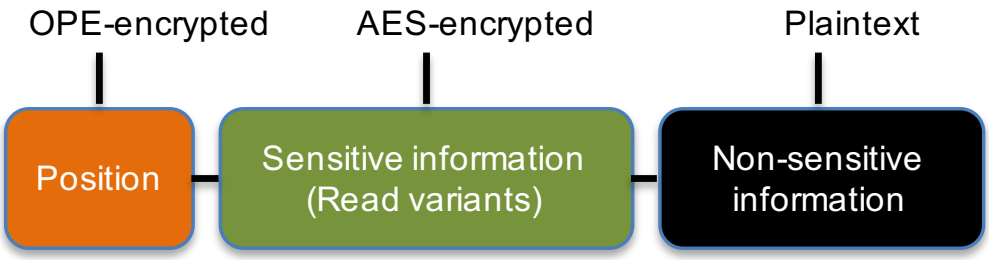
$$\text{Block size} \parallel \text{Position} \parallel \text{List of read headers} \parallel \text{Quality scores} \parallel \text{PosCigar}$$

In the following, we explain the terminology used in this option.

- **Block size**: Length (measured in bytes) of a position row;
- **List of read headers**: List of information of the reads that start at this position. It is decomposed as $(\text{Order} \parallel \text{Read Info})^*$, with "*" meaning an arbitrary number of such headers. The read information would also include read length so that we do not have to store the end marker as in Supplemental Figure S1.
- **Quality scores**: Quality scores for the bases of this position.
- **PosCigar**: The variants information.

# S3. ENCRYPTION

The data is protected with secure symmetric encryption techniques (Supplemental Figure S3). To provide fine-grained privacy control and avoid information leakage in data retrieval, our solution encrypts each row of Supplemental Figure S1 independently. We encrypt positions with order-preserving encryption (OPE) in order to enable us to efficiently retrieve encrypted data without first decrypting it. This is a critical feature to prevent insider threats in the server that stores the encrypted data. We encrypt other sensitive information of a row (e.g., short read differences with respect to the reference) with AES-256 in the CTR mode. The stream cipher mode enables the server to tailor the decryption key stream to only those positions matching the data retrieval request, hence sensitive information of other positions is also protected from clients who send the data request. With our solution, the privacy control is precise at the




Supplemental Figure S3: Protection methods for different types of information of a position row.

position level, in contrast to some existing solutions that provide coarse-grained encryption on the data, which leaks non-negligible information in each data retrieval.

More details of the encryption design are explained here. All fields, except "Position" and "PosCigar", are left in clear, because they do not contain any highly sensitive information. The encryption scheme can be described as follows:

- The patient is assigned with a master key $K_m$ that is used to derive various encryption keys.
- The position field is encrypted with order-preserving encryption (OPE) (Boldyreva et al. 2011) for the efficiency of a random access based on position comparison (details in Section D). A key, $K_{ope}$, is derived from $K_m$ and used for OPE.
- The "PosCigar" field is encrypted with a stream cipher (SC). A key, $K_{sc}$, is derived from $K_m$ and used for the SC. For the $i$-th GZIP block in the file, the "PosCigar" fields of all the included position rows are concatenated to generate a single data block, $D_i$, which is encrypted with $K_{sc}$ and a 64-bit random salt $R_i$. The random salts are stored in the index file.

# S4. EFFICIENT INFORMATION RETRIEVAL

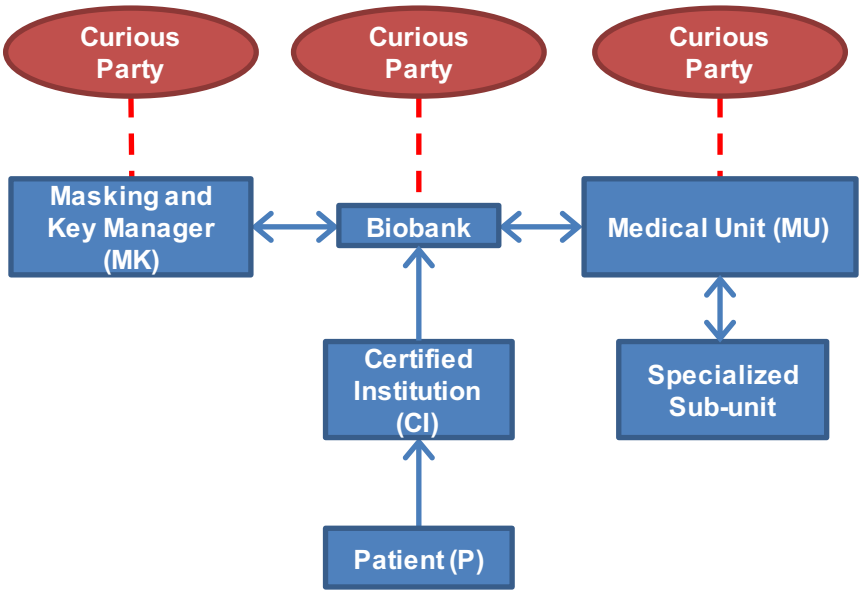
SECRAM is essentially a storage format that could be used locally like BAM. Users can employ the API to efficiently extract any region of data from the file without decrypting and decompressing the whole file.

However, in order to provide strong security and privacy guarantees in a cloud computing scenario where no single party (e.g., the storage server, the client) is trusted, we propose the following system for efficient information retrieval by using SECRAM.

## A. *System Structure*

For achieving the aforementioned security and privacy guarantees, we adopt the system architecture (Supplemental Figure S4) proposed by Ayday et al. (2013). For the sake of brevity, we mention only the security and privacy goals we intend to achieve in the system:

- The original content of a short read cannot be accessed without explicit permission from the owner;
- Users (e.g., medical units) can access only restricted regions of data that are authorized by the owner; each user is allowed to access only specific region(s), e.g., a medical unit that specializes in female breast and ovarian cancers might be allowed to access only BRCA1 and BRCA2 genes;
- In order to prevent the inference attack on the purpose of user access, the storage center (i.e., the Biobank) should not learn about the region that a user accesses.




Supplemental Figure S4: The system architecture for privacy-preserving processing of raw genomic data (image taken from Ayday et al. (2013))

Throughout this section, we consider a query that retrieves data in the position range $[P_1, P_2]$. The bounds $P_1$ and $P_2$ are sent from a medical unit MU to the Biobank.

## B. *Basic Retrieval*

The retrieval process consists of the following steps:

1. MU and MK setup a one-time symmetric session key $K_{ss}$;
2. MU encrypts the range $[P_1, P_2]$ with AES under the key $K_{ss}$, to generate the ciphertext $[AES(K_{ss}, P_1), AES(K_{ss}, P_2)]$ that is sent to MK;
3. MK decrypts $[AES(K_{ss}, P_1), AES(K_{ss}, P_2)]$ and re-encrypts it with OPE under the key $K_{ope}$, generating ciphertext $\left[OPE\left(K_{ope}, P_1\right), OPE(K_{ope}, P_2)\right]$ that is sent to the Biobank;
4. The Biobank retrieves the $k$ GZIP blocks $(B_i, B_{i+1}, \dots, B_{i+k-1})$ that overlap with the range $\left[OPE\left(K_{ope}, P_1\right), OPE(K_{ope}, P_2)\right]$, with the help of the index file. The Biobank decompresses the block $B_i$, and computes an offset value of the position row $OPE\left(K_{ope}, P_1\right)$, denoted by $OFF_{P_1}$: the byte offset of its "PosCigar" field in the data block $D_i$ (see Section C-2). $OFF_{P_2}$ is computed similarly. The Biobank sends $OFF_{P_1}$, $OFF_{P_2}$, and random salts $(R_i, R_{i+1}, \dots, R_{i+k-1})$ to MK;
5. With key $K_{sc}$, MK derives the corresponding decryption keys $(DK_i, DK_{i+1}, \dots, DK_{i+k-1})$ for the $k$ blocks, masking those rows out of the range $[P_1, P_2]$. MK encrypts these keys, thus generating $(AES(K_{ss}, DK_i), AES(K_{ss}, DK_{i+1}), \dots, AES(K_{ss}, DK_{i+k-1}))$, and it sends these encrypted keys to the Biobank;
6. The Biobank sends the $k$ GZIP blocks $(B_i, B_{i+1}, \dots, B_{i+k-1})$ and the encrypted keys $(AES(K_{ss}, DK_i), AES(K_{ss}, DK_{i+1}), \dots, AES(K_{ss}, DK_{i+k-1}))$ to MU;
7. MU decompresses the blocks and decrypts the content.

## C. *Context-Aware Retrieval*

In the basic retrieval scheme, we do not address the problem that the returned data might contain incomplete reads and thus cannot perform queries of read information, such as mapping quality, strand,

and a relative position inside a read. To answer these queries, we need to retrieve the complete context of a position row. One solution is to return the "Read Info" of the position row.

Now, consider the retrieval 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 Biobank traces back to the previous block(s) and crops the corresponding "Read Info" fields for the incomplete reads of row $P_1$. As a block normally contains hundreds of position rows, the Biobank usually needs to look back only one previous block as long as the reads are not extremely long. The Biobank then returns these "Read Info" fields along with the data from the basic retrieval scheme.

### D. *Efficient Query Processing*

We discuss how to efficiently answer the queries that are interesting to MU. Before going into details, we state the following assumption:

- The genomic positions are queried sequentially. In other words, if the position rows in a genomic range $[P_1, P_2]$ are returned, these rows are queried from L to U one by one. This assumption does not affect the correctness of our system, rather it affects the efficiency.

The queries related to one position are more efficient than those related to a complete read. Examples of queries of one position include the coverage of a position, the variants (substitution, insertion and deletion) and quality scores. The complexity of answering these queries is $O(C)$, where $C$ is the coverage of this position. The overall complexity of iterating all positions in the range $[P_1, P_2]$ is $O((P_2 - P_1)C)$.

For queries related to a complete read, we need to access the "Read Info" field. But this field does not exist in each row. The main idea is that we run the reconstruction algorithm for a position row whenever these queries are needed. Then the complexity of answering these queries for all rows in the range $[P_1, P_2]$ is $O((P_2 - P_1)CL)$, where $L$ is length of a read and $C$ is the coverage. But with the above assumption, we can carry out the task much more efficiently. In the context-aware retrieval, the Biobank needs to run the reconstruction algorithm for the position $P_1$. In the data returned to MU, "Read Info" fields already exist for all the reads of position $P_1$. To answer queries from position $P_1$ to $P_2$, MU maintains a cache that contains the "Read Info" fields of all reads in the current position. The cache is initialized to the "Read Info" fields of position $P_1$, and is updated as MU move to the next position. The complexity of this method is $O(CL + (P_2 - P_1)C)$, if the one-time reconstruction in the Biobank is taken into account.

## S5. A SNAPSHOT OF SECRAM FORMAT

Below (Supplemental Figure S5) is a snapshot of a SECRAM file (unencrypted). If a cell is empty, it means nothing is stored. We replace several lengthy fields with "XXX" for legibility. "Number of read headers" denotes the number of reads that start at this position, and then the content of these read headers are listed.

| Reference | Position | Coverage | PosCigar | Quality scores | Number of read headers | Reference Length | Mapping quality | Read name | Flag | Template length | Next reference index | Next position | Auxiliary data |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 29443773 | 26 | 0D | 38, 37, 38, 39, 38, … | 0 | | | | | | | | |
| 1 | 29443774 | 28 | * (all matches) | 38, 37, 38, 39, 38, … | 2 | 28 | 22 | read1 | 81 | 214 | 1 | 29443774 | XXX (as BAM) |
| | | | | | | 29 | 22 | read2 | 161 | -214 | 1 | 29443774 | XXX (as BAM) |
| 1 | 29443775 | 28 | 8XA | 39, 18, 21, 33, 32, … | 0 | | | | | | | | |
| 1 | 29443776 | 28 | 6XG 21XG | 39, 32, 35, 35, 20, … | 0 | | | | | | | | |
| 1 | 29443777 | 29 | 29S3TAC | 33, 38, 38, 38, 38, … | 1 | 31 | 44 | read3 | 161 | 214 | 1 | 29443771 | XXX (as BAM) |

Supplemental Figure S5: Several example rows in a transposed SECRAM file.

## References

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.

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.