Side channel attack on a partially encrypted MPEG-G file

Daniel Naro, Jaime Delgado, Silvia Llorente

Departament d'Arquitectura de Computadors (DAC),

Universitat Politècnica de Catalunya · UPC BarcelonaTECH,

C/Jordi Girona, 1-3, 08034 Barcelona

{jaime.delgado, silvia.llorente}@upc.edu, dnaro@ac.upc.edu

http://dmag.ac.upc.edu

Corresponding author
Name: Silvia Llorente
e-mail: silvia.llorente@upc.edu;silviall@ac.upc.edu
Phone: +34 93 401 74 09

Abstract – New genome sequencing technologies have decreased the cost of generating genomic data, thus increasing storage needs. The International Organization for Standardization (ISO) working group MPEG has developed a standard for genomic data compression with encryption features. The approach taken in standard MPEG-G (ISO/IEC 23092) to compress genomic information was to group similar data into streams. Taking this into account, one of the protection options considered was to encrypt each stream separately. In this paper, we show that an attacker can use an unencrypted stream to deduce the encrypted content if streams are encrypted separately. To do so, we present two different attacks, one based on signal processing and the other one based on neural networks. The signal-based attack only works with unrealistic settings, whereas the neural network-based one recovers data with realistic settings (regarding read length and coverage). The presented results made MPEG reconsider the encryption strategy, before final publication of the standard, discarding separate streams encryption approach.

Keywords: Encryption, Genomic information, Information leakage, MPEG-G

1 Introduction

New genome sequencing technologies allow us to obtain the information contained in the DNA (Deoxyribo Nucleic Acid) [1, 2]. The obtained information is sensitive, as it not only identifies the sequenced individual and informs about possible health issues, but also gives information about blood relatives and the diseases they might have to face.

In the new genomic information representation format MPEG-G [3] developed by ISO's working group MPEG, the encoded information is split into different descriptor streams. Each stream encodes one aspect of the data (for example, the genomic position or the mutation position).

The first draft specification of MPEG-G offered a tradeoff between usability and security. It was possible to encrypt only some of the descriptor streams, giving thus as much information as possible while maintaining the most sensitive information confidential. For example, the user could encrypt information regarding the mutations, but leave the information regarding genomic positions available. Partial encryption of the information unit allows to maintain some usability at the cost of not providing the highest level of privacy possible. We will show that by encrypting only the mutation position and mutation type streams, there are ways to recover the position of the mutations. In other words, there is at least one configuration where the supposed tradeoff is flawed. As an adversarial actor can recover the encrypted information, there is no privacy anymore.

The attack we present here allows the recovery of encrypted information. Our findings made the MPEG group reconsider the initial tradeoff approach. The published specification now relies on an entirely accessible access unit or entirely encrypted access unit approach, instead of the original stream-based solution.

Another relevant value and novelty of this work is the combination of different normally independent techniques. Facilitating secure encryption of genomic information in a specific standard, such as MPEG-G, has been the main focus, but we have used different mechanisms, such as neural networks and machine learning, in a simple way in order to achieve our objectives.

The rest of the paper is organized as follows. First, we present some background information describing genomic information features, MPEG-G encoding and existing attacks. Then, we propose the different methods to attack the genomic information when it is represented in streams, one based in signal processing and other in neural networks. After that, we show our results and discuss them. Finally, we describe some conclusions and future work.

2 Background

2.1 Introduction to genomic concepts

In each cell, DNA molecules encode the genomic information. We can interpret a DNA molecule as a sequence of nucleotides [1, 2].

The genomic information is almost identical across all individuals in a species. But the genome of each individual has a unique set of mutations: some nucleotides might change, be inserted, or be removed from the DNA, possibly leading to different medical conditions.

In fact, for each position of the genome, we have two copies of the information: one on a DNA molecule (i.e. chromosome) inherited from the mother, and the other one on a DNA molecule inherited from the father. There are multiple such pairs of chromosomes, each corresponding to different portions of the human genome.

The objective of research in the genomic field is to identify which mutations are linked to certain diseases or advantages. First, a reference genome is constructed by comparing the genomes of many individuals and selecting the most frequent nucleotide for each position. By comparing the reference genome and the genome of an individual we can determine which mutations an individual presents.

In order to obtain the genomic information from one individual, a biological sample is sequenced. Nowadays, Next Generation Sequencing (NGS) machines, like the devices produced by Illumina [4], are used for this. The genome is sequenced by these machines by obtaining many small subsequences of contiguous nucleotides from the DNA molecules, referred to as reads [5]. The origin of each read is random, and therefore the order of appearance of the reads in the FASTQ is also random.

To further process the genomic data, the information contained in the FASTQ file must be aligned to the reference genome [6]. During alignment, each read is aligned to the genome: the aligner selects the likeliest position on the genome for the read to originate from (i.e. on which chromosome and where on the chromosome). In the process, the differences between the reference genome and the read are also detected. As the origin of the read is random, the used strategy is to sequence more reads than needed, in order to maximize the likelihood of having at least one read per position. The consequence is that each nucleotide is possibly sequenced multiple times. We refer with coverage to the average number of instances a nucleotide has been sequenced.

Also during alignment phase, the differences between the read and the reference genome are identified: mainly mutations, insertions and deletions. A mutation means that a nucleotide on the read is different from the nucleotide on the reference for that location. An insertion means that some nucleotides have been added. Finally, a deletion means that some nucleotides are missing. By analyzing the detected differences for all reads, we can detect the mutations presented by the individual at each genomic location.

Sequencing machines can make mistakes when identifying nucleotides. The fact that one nucleotide appears altered in one read does not mean that the studied individual has a mutation at that location. Only if many reads consistently show the difference, we can determine that there is a mutation at that location.

By comparing the mutations of many individuals, a list of existing mutations is created. Some mutations will be more or less frequent across the population. The list of known mutations and their probabilities can be consulted in databases such as [7]. The individual might have mutations not known to the database.

2.2 MPEG-G encoding

Since 2016, the International Organization for Standardization (ISO) has been working on a standardization activity for genomic information representation. This activity was initiated by the MPEG group (ISO/IEC JTC 1/SC 29/WG 11), now separated into several subgroups. ISO/IEC JTC 1/SC 29/WG 8 (MPEG Genomic coding) is the one in charge. The intending was to use the MPEG experience in compression of multimedia data for the compression of genomic information. The result of this work, MPEG-G [3], has been released as a standard and the group is working in new versions. MPEG-G defines, alongside the encoding and among other features, security mechanisms to guarantee the privacy and the integrity of the information.

In order to reduce the amount of data to store, MPEG-G only stores the information required to reconstruct a read, without including the original sequence. In other words, the format only stores, for each read, the differences with the reference (i.e. if there are insertions, deletions or modifications), and the nucleotides involved. For example, for a read presenting an insertion of nucleotide ’A’ at position 2, a deletion at position 4 and a nucleotide ’A’ which differs from the position at position 6, we would store "insertion at 2, deletion at 4 and mutation at 6" and "A;A" (the list of nucleotides inserted or mutated to), as this is all the information required to reobtain the original sequence.

In order to increase the compressibility of the information, the data is subdivided into descriptor streams. One stream encodes the position of the read, another the position of each operation in the read, and yet another the type of the operation, among many other descriptors. In our example, we would have at least the following descriptor streams:

- The position of the read. This stream contains for example the value 1, as the first and only read begins at position 1 of the reference.
- The operation positions. This stream contains the values 2, 4 and 6, as these corresponds to the positions where the read differs from the reference.
- The operation types: This stream contains the values Insert, Deletion and Mutation, the operations to be done to the reference to achieve the resulting genome.
- The inserted nucleotides: This stream contains the value ’A’, as it is the only inserted nucleotide.
- The mutated nucleotides: This stream contains the value ’A’, as it is the only mutated nucleotide.

In order to further increase the compressibility, MPEG-G intends the information to be separated into different classes, which are the following ones:

- Class P: contains only reads which do not present any difference with the reference. Therefore, there is no difference in position, type or associated nucleotide to store.
- Class N: contains only reads where some nucleotides could not be identified. With respect to class P, we only need to add the descriptor for the operation positions (as we already know that all operations will be a mutation to ’N’).
- Class M: contains reads where there are only mutations. In respect to class N, we only need to add the descriptor for the mutated nucleotide.
- Class I: contains all remaining reads (i.e. reads containing insertions and deletions). In this class, all descriptors must be present.

In order to enable random access features, MPEG-G defines Access Units (AU). The information of an access unit can be read independently of all other access units. An access unit can only have one class, and there can be multiple access unit encoding information from the same genomic region. As such, in order to obtain the best compression results, it is to expect that for every region of the file there is one AU of each class encoding the region, and each read will then be stored in the AU which best corresponds to its characteristics.

2.3 Other attacks on genomic information

The most frequent formats [6, 8] for storing genomic information do not provide confidentiality mechanisms, and thus no attacks can be proposed against them. To palliate against this lack of confidentiality, other formats have since been introduced: Cryfa [9] for unaligned data and SECRAM [10] for aligned data. To the best of our knowledge no attacks have been proposed against any of them.

Similar to MPEG-G, Cryfa uses an approach of dividing the input data into different stream of information. Each of these streams are treated separately in their representation. In one of the steps, the authors refer to as compacting (representing multiple symbols of the stream with one input symbol), and the result is then shuffled and encrypted. The rationale behind the shuffling is to increase the difficulty of a brute-force attack. If the first bytes to be decrypted were a known fixed sequence of bytes, this fact could be used to identify the correct key when executing a brute-force attack. However, as the content is shuffled, the first byte to be expected is not known, therefore such an approach cannot be straightforwardly used to identify the end of a brute-force attack.

As previously introduced, SECRAM [10] is designed for aligned data. While in SAM [6] the alignment information is stored for each read, as previously introduced, in SECRAM [10] the information is stored per position: for one position, all alignment results are stored together. In other words, the information provided in a SAM read is broken down in the alignment information for each of the mapped positions, and each of these sub-units of information are appended to the list of alignment results per position. This restructuring of the information, akin to the transposition of a matrix, allows the encryption of the information for certain positions. To the best of our knowledge no attack has been proposed against this. Nevertheless, one might consider that through the usage of a database such as [7] and the assumption that, if the information for a given position is encrypted, then it is likely that this is the position of a mutation. So, there are ways to partly recover the information. One approach which has been considered for a release of genomic information, but maintaining certain levels of confidentiality, is to simply not publish one part of the genome. However, using statistical information of which mutations are usually inherited together, the non-published information can be obtained by inference, as was done in [11]. The danger of such techniques can be lessen by taking into account those same statistics as proposed in [12], where the authors propose to maximize the utility of the published information (e.g. publish as much relevant information for research as possible), while ensuring that certain features which have to remain private cannot be inferred.

Currently, one of the approaches to protect genomic information is by the use of beacons [13]. In a beacon, the genomic information of multiple individuals is stored in the form of the diagnosed mutations. This then allows to query information about the mutations: for example, a query to a beacon might be the frequency of a given mutation in the part of the population with a given disease, or if at least one patient presents the mutation. The beacon approach is intended to protect the information of any given patient as the information of one individual cannot be obtained, only information relating to the whole statistic. Nevertheless, a model of attacks has been proposed against the beacon approach. One of these attacks is proposed in [14], and wants to determine if an individual is present in a beacon or not. In this situation, the genome of the patient is known, and the frequency statistics of mutations in the entire population are also known. As the beacon will contain information for a finite subset of the human population, each individual entry will have an effect on the reported statistics. In [14], the authors propose to test, for the mutations of the individual under attack, if the frequency statistic reported by the beacon is consistently higher than expected. The fact that the individual’s genome is stored in the beacon is much likelier than the divergence from the beacon with the human statistic can be better explained by the presence of the individual in the beacon.

Beacons can also be defined as replying with a Boolean response, i.e. instead of answering with a mutation frequency, the beacon answers with a Boolean indicating if at least one patient has the mutation given as an input parameter. For this type of beacon, the probability of the presence of one individual’s genome in the beacon can be ascertained as described in [15]. In this case, the approach treats the beacon as a Bloom filter [16]: if for all mutations known for the patient the beacons respond that at least one individual has such a mutation, the confidence that the individual is present in the beacon is fairly high.

3 Method

3.1 Available information

We will attempt to revert the first proposed privacy mechanics of MPEG-G. We will be playing the role of an actor having received a copy of an MPEG-G file, with two access units (AU), one of class P and the other of class M, the two encoding information from the same portion of the genome. This consideration is not unrealistic: if we are able to intercept a file, we would receive multiple access units, and among those it is fair to assume that many would be of class P and M. We also suppose that all reads have the same known length. As previously introduced, the privacy relevant information are mainly the differences of each read with the reference. Therefore, we will assume that the information stored in the AU of class P is stored as plaintext, and the information stored in the AU of class M is partly encrypted: the information storing the position of the reads is stored as plaintext, whereas the information storing the position and type of the mutations is encrypted. At first glance, this approach of partially encrypting is secure, as without the position of the mutation the original information cannot be reconstructed. However, we will attempt to discover where the mutations are and if the mutations affect one or the two copies (chromosomes) of the genomic region.

We should stress the fact that we will not decrypt the ciphertext, but infer its content from the public information that we have, that is, from the plaintext. As such, it is not necessary to use the ciphertext as input in any case.

3.2 Signal processing interpretation

We know that for each read stored in an AU of class P, all nucleotides of the read are equal to the reference genome. Therefore, each read in an AU of class P gives us evidence that there are no mutations for a sub-region of the genome in one of the copies. This sub-region is delimited by the first mapped position and the known length of the read. In order to obtain the best compression, a read should only be stored in an AU of class M if there is at least one changed nucleotide. Therefore, we will take the presence of a read in an AU of class M as proof that there is at least one mutation in one of the copies for the same sub-region (as per our assumed initial state, this information is available).

The attack we propose can be represented as a “for” loop where we are going through the genomic information represented in AU’s of different classes, trying to infer the most likely mutations. So, our attempt at deducing mutations will now revolve around contrasting all provided evidence and deducing which hypothetical mutations could better explain the observed results.

Let us propose an interpretation of the attack using three different signals: one to represent genome, called $genome_{signal}$, one to represent class, called $class_{signal}$, and the last one to represent ratio, $ratio_{signal}$. They are further explained in the rest of this section. Doing so, we will diverge from nature’s reality and imagine for a moment that there is only one copy of the information, i.e. there is only one chromosome. Furthermore, we will pretend that there cannot be any insertion or deletion of nucleotides. We will interpret this copy, i.e. the chromosome, as a discrete time signal which can only take two possible values. Each value of the signal is mapped to a nucleotide on the reference genome. If for one position there is no mutation, then the signal is equal to 0, but if there is a mutation then the signal is equal to 1. We will refer to this signal as the $genome_{signal}$. An example of comparison between reference genome and specific genomic information is represented in Tables 1 and 2. In Table 1 there is a difference in the second nucleotide as we find a T whilst in the Reference genome there is a G. This means this read belongs to class M. On the other hand, in Table 2 the nucleotides perfectly match with the reference genome, indicating this read belongs to class P.

Reference Genome	A	G	C	A	A	T	G	C	A
Genomic information	A	T	C	A	A	T	G	C	A
$genome_{signal}$	0	1	0	0	0	0	0	0	0

Table 1. Read belonging to class M, the sum of $genome_{signal}$ is greater than 0.

Reference Genome	A	T	C	C	A	T	A	C	A
Genomic information	A	T	C	C	A	T	A	C	A
$genome_{signal}$	0	0	0	0	0	0	0	0	0

Table 2. Read belonging to class P, the sum of $genome_{signal}$ is 0.

The first step of the algorithm will be to combine the available information in order to generate the signals required as input to the classifier. We update the counters in order to obtain $ratio_{signal}$ based on every read and its respective $class_{signal}$. Figure 1 shows a complete example. Without access to the encrypted information, we have a clear view of how many reads present a mutation for each position, thus allowing us to use a classifier to resolve whether there is a strong likelihood of mutation.

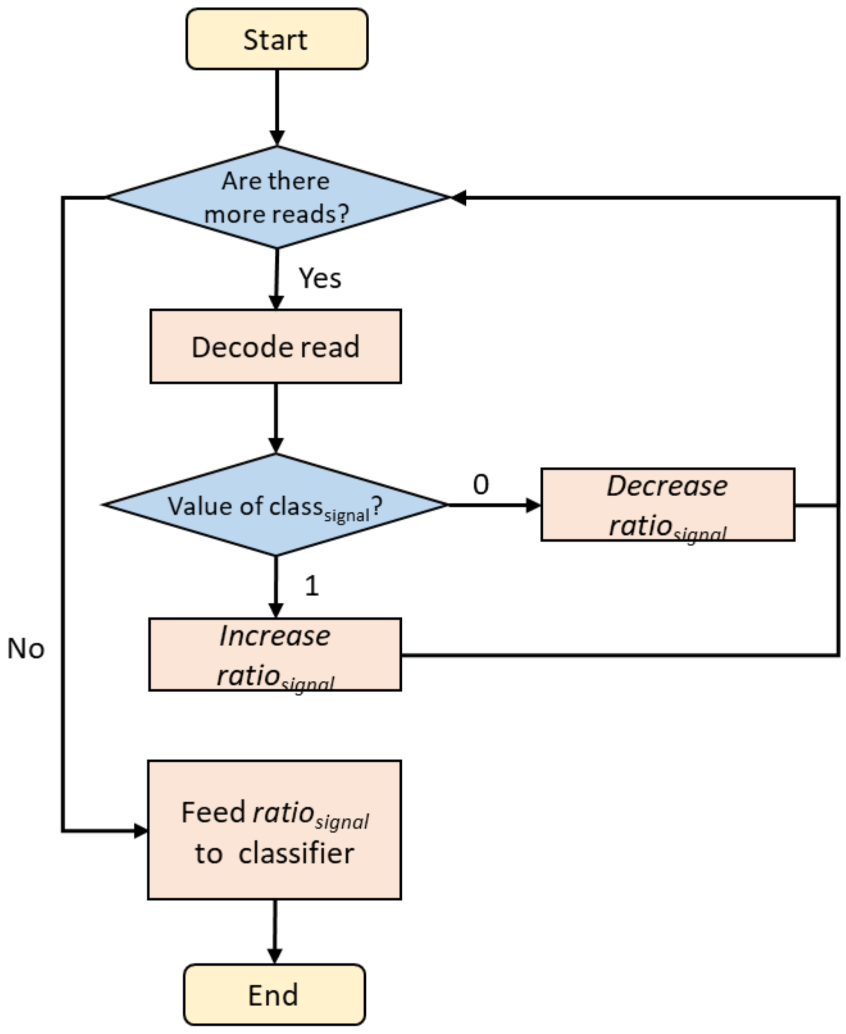
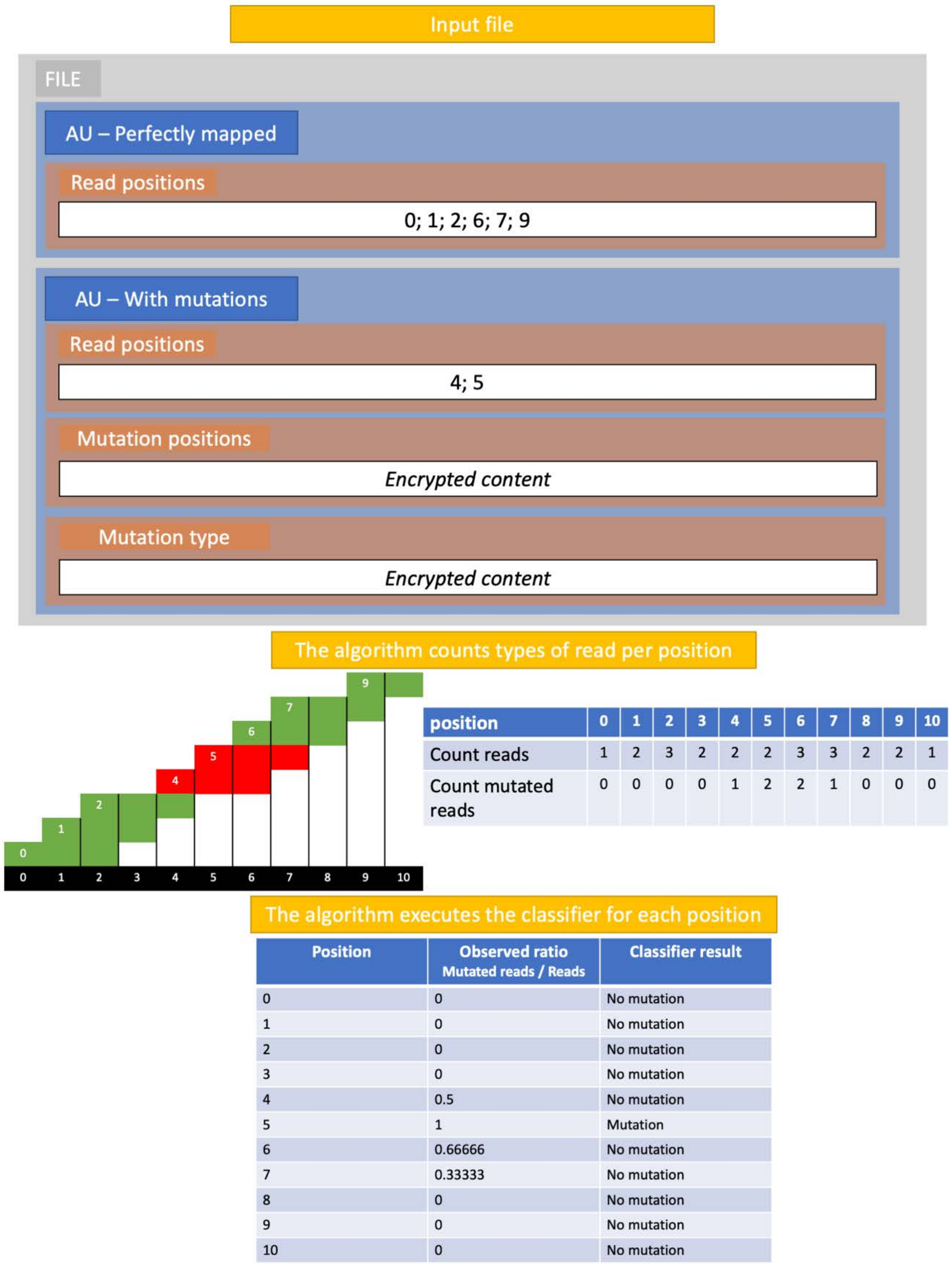

Start
Are there more reads?
Yes
No
Decode read
Value of class$_{signal}$?
0
1
Decrease ratio$_{signal}$
Increase ratio$_{signal}$
Feed *ratio$_{signal}$* to classifier
End

Fig. 1. Flowchart of the generation of the *ratio$_{signal}$* from the *class$_{signal}$* from a genomic file stored using MPEG-G.

Figure 2 shows graphically how from the MPEG-G input file the information is extracted and calculated.



position	0	1	2	3	4	5	6	7	8	9	10
Count reads	1	2	3	2	2	2	3	3	2	2	1
Count mutated reads	0	0	0	0	1	2	2	1	0	0	0

Position	Observed ratio Mutated reads / Reads	Classifier result
0	0	No mutation
1	0	No mutation
2	0	No mutation
3	0	No mutation
4	0.5	No mutation
5	1	Mutation
6	0.66666	No mutation
7	0.33333	No mutation
8	0	No mutation
9	0	No mutation
10	0	No mutation

Fig. 2. Extraction of information from MPEG-G file.

We can now define which class a read belongs, given its initial position. We sum all values of the signal for the region the read corresponds to. If the sum is greater or equal to 1, there are one or more mutations with respect to the reference, therefore the read belongs to class M as otherwise it could not be represented due to the lack of descriptor streams. If the sum is equal to 0, we will assume that the read belongs to class P, as this is the choice yielding the best compression. Let us define a new signal $class_{signal}$ as the signal indicating to which class a read starting at that position would belong: we will use 1 as class M and 0 as class P. See Figure 3 for an illustration.

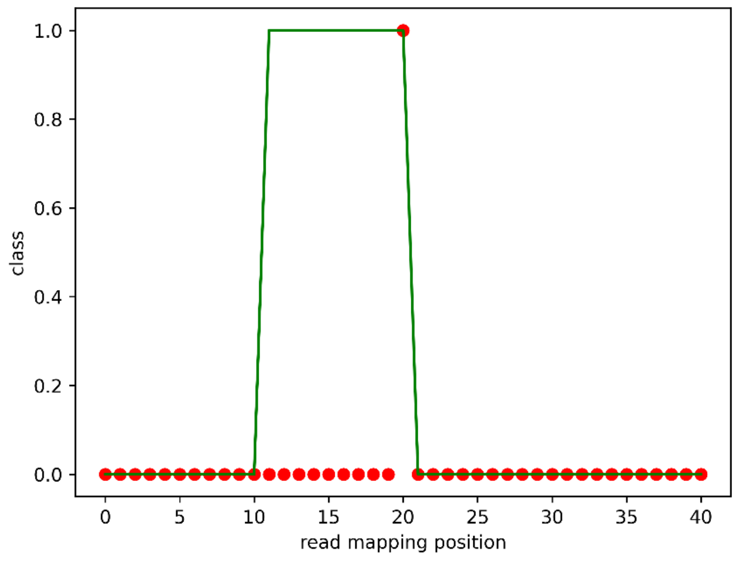


Fig. 3. Illustration of $class_{signal}$ value (plotted as a solid line), based on a $genome_{signal}$ (plotted as dots), in the case where the read length is equal to 10. $class_{signal}$ is equal to 1 if a read starting at that position includes at least one mutation, i.e. a position at which $genome_{signal}$ is equal to 1.

We can see how $class_{signal}$ can be obtained by convoluting $genome_{signal}$ with a rectangular signal integrating over the current and future values. However, the result of the convolution is clipped, thus we lose the information of how many mutations are present.

The last step will be to calculate the expected ratio of class M and class P information for a given position. We know the reads' length, therefore we can compute a signal $ratio_{signal}$, such that for each position p we store the average value of $genome_{signal}$. The average is computed on all reads which starting position implies that they cover the position p. This computation can be understood as a convolution with a rectangular filter, integrating over the read length position prior to p. See Figure 4 for an illustration. This signal does not have any equivalent in nature, but it gives a metric on the likelihood of having a mutation at position p: if the value of $ratio_{signal}$ at a position p is 0, it is very unlikely that there is a mutation close-by, and, if the value is close to 1, it is very unlikely that there is none.

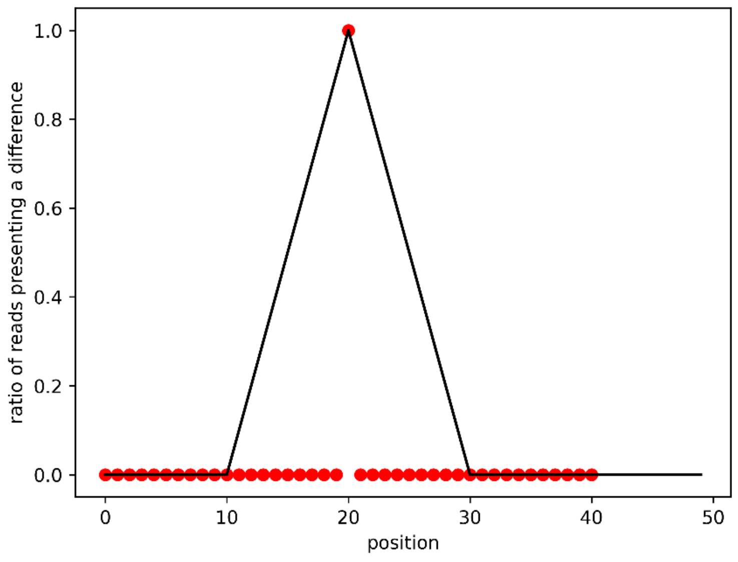


Fig. 4. Illustration of $ratio_{signal}$ value (plotted as a solid line), based on a $genome_{signal}$ (plotted as dots), in the case where the read length is equal to 10.

As previously introduced, a read belongs to class M if there is one or more mutations, therefore $class_{signal}$ loses information of how many mutations are involved. This implies that there are cases where different $genome_{signal}$ generate the same $ratio_{signal}$. See Figure 5 for an illustration of the case where two mutations are separated by a length inferior to the read length leading to the possibility of a third mutation being hidden.

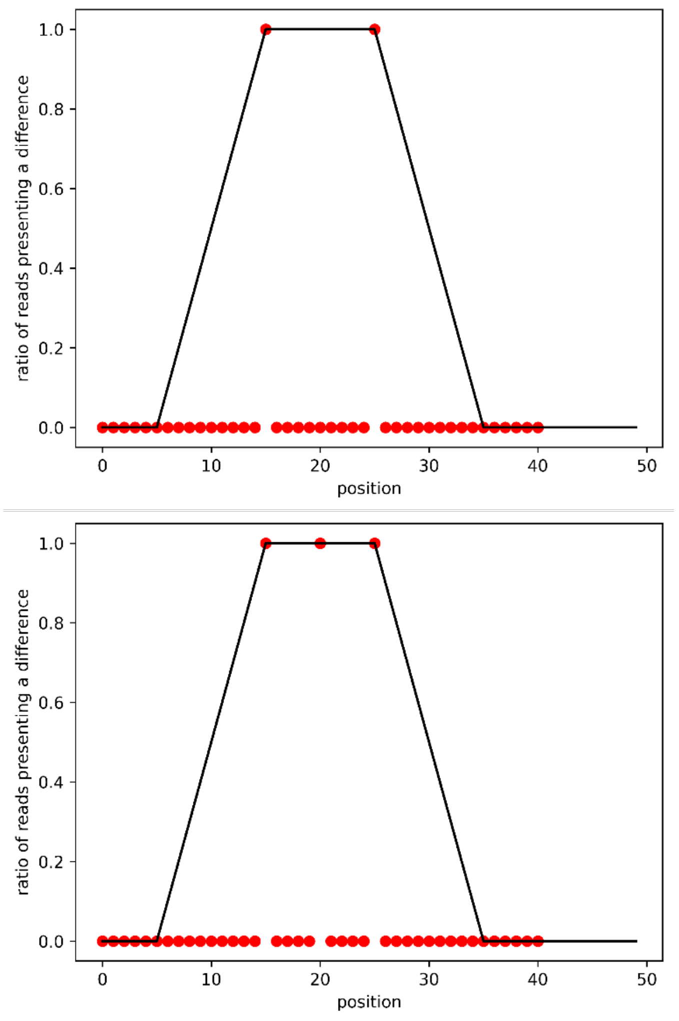


Fig. 5. If the distance between two mutations is inferior to read length, then other mutations might be present between the two without altering the value of $ratio_{signal}$.

If we now go back to our initial assumptions, we see that we know the first position of each read, and to which class it belongs. In other words, we can compute, through two convolutions, an experimental version of $ratio_{signal}$ (the first convolution yields the first position where the mutation is detected, the second the actual ratio). By deriving the inverse operation allowing us to go from $ratio_{signal}$ to $genome_{signal}$ (i.e. undoing the convolution through another convolution). Then, we should be able to apply the same procedure to our experimental signal and retrieve a guess for the mutations in the genome, which was supposed to be protected through the encryption of certain fields. An example of this is shown in Figure 6, the mutation positions at position 150 and 250 are inferred from the ratio spikes observed. Arrows 1 and 2 represent the convolution to obtain the first position where the mutation is detected (i.e.the result of this convolution indicates if there is mutation covered by a readstarting at that position). Arrows 3 and 4 perform the same convolution again, obtaining for a given position the ratio of mutated reads, i.e. $ratio_{signal}$. In the ideal case, the two convolutions represented with arrows 1-2 and 3-4, can be undone using the inverse convolutions. This is represented by arrows 5-6 and 7-8. In other words, the left side of Figure 6 represents the theoretic explanationof the observed $ratio_{signal}$. The right side of Figure 6 represents an attempt to derive the $genome_{signal}$ from the observed $ratio_{signal}$ based on this theoretical explanation.

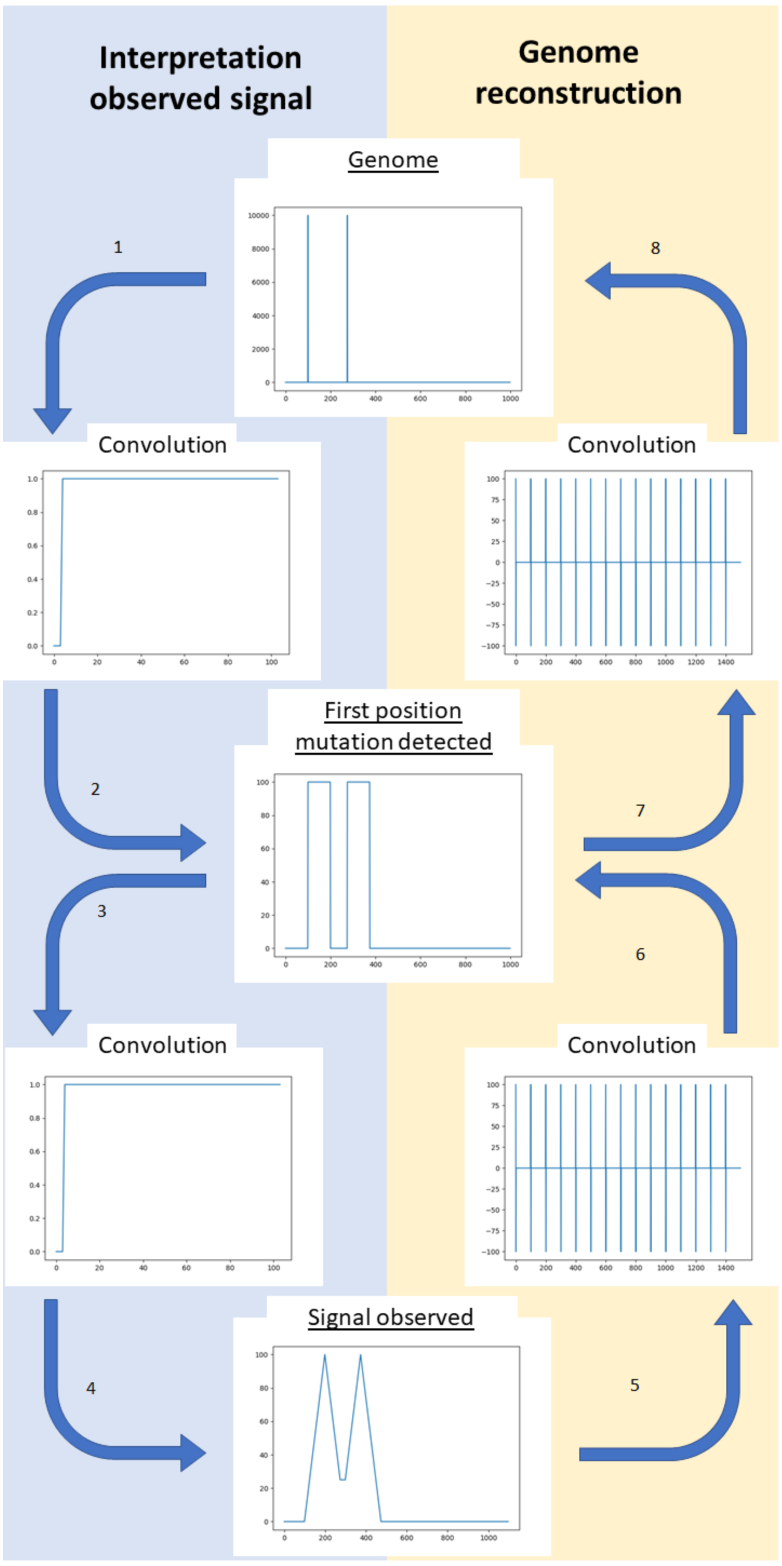
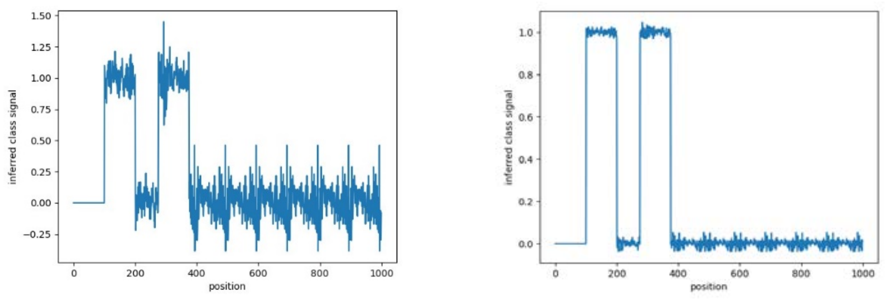


Fig. 6. Summary of convolution interpretation of the attack.

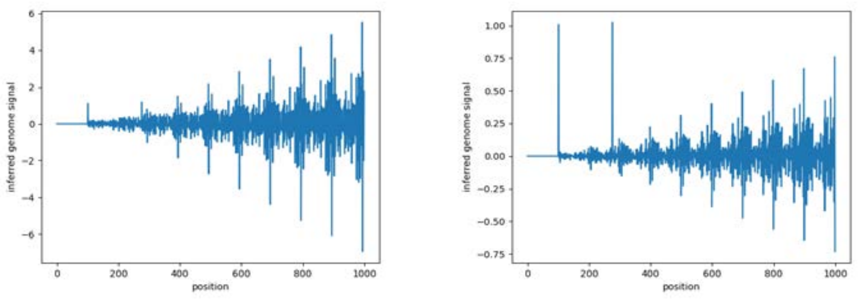
Of course, in nature there are two chromosomes, thus we need to add some complexity to our schema to allow for this. To do so, we will assume that sequencing technology sequences from each chromosome with the same probability. Thus, from

$genome^{Mother}{}_{signal}$ we can derive $ratio^{Mother}{}_{signal}$, as we have shown, and the same for the father. The observed $ratio_{signal}$ will then be equal to the average of both.

In our experiments, however, this approach is too sensible to irregularities in $ratio_{signal}$. Figure 7 shows how the noise increases with each convolution. The higher the coverage, the less noise there is, and the better the prediction is. We can further improve the prediction by reducing the noise by rounding to the closest possible value as shown in Figure 8. Using this approach, we test the performance of the attack across different coverage values. The results of the performance, represented with the F_1 score are provided in Figure 9. The results obtained are not good enough to significantly risk the privacy of the information, even with unrealistically high values of coverage.

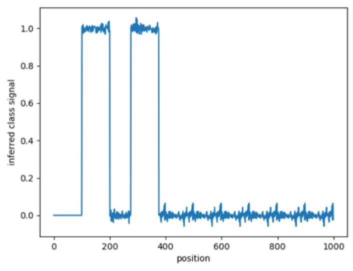


(a) Recovered first position mutation detected, coverage 10000 (b) Recovered first position mutation detected, coverage 50000

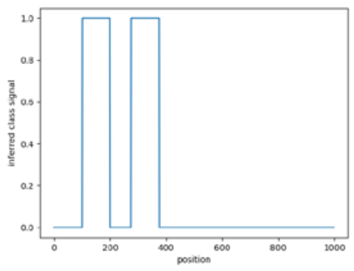


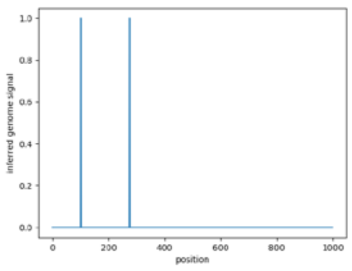
(c) Recovered genome, coverage 10000 (d) Recovered genome, coverage 50000

Fig. 7. Examples of genome reconstruction using convolutions.



(a) Prior to noise reduction.





(b) Noise reduced. (c) Recovered genome.

Fig. 8. Example of noise reduction, by rounding to the closest half unit**.**

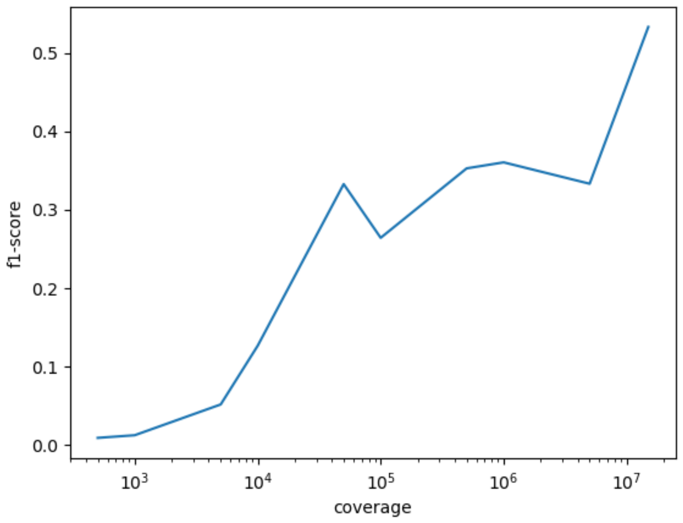


Fig. 9. *F_1* score obtained by the convolution attack**.**

3.3 Neural network

In Figures 4 and 5, we have seen that through visual inspections the mutations in the genome can be inferred. So, the problem we are trying to solve is a classification one: for each position, we use a signal as input to determine which class the test data belongs.

We choose to construct our classifier using a simplistic neural network, due to the ease with which such classifier can be written with current tools. Our neural network is trained to retrieve from a shorter experimental version of the $ratio_{signal}$, if there is a mutation at the central position of the corresponding genome. For a given $read_{length}$, the neural network receives as input $read_{length} * 3 * 2 + 1$ floating point values bounded in the range [0; 1): this corresponds to the $read_{length} * 3$ ratio values prior to the point of interest, to the $read_{length} * 3$ ratio values after the point of interest, and to the ratio value at the point of interest. Furthermore, it also receives $read_{length} * 3 * 2 + 1$ floating point values bounded in the range [0; 1) corresponding to the likelihood of a mutation at that position. The output of the neural network is a value equal to either 0, 1 or 2. Value 0 indicates that the neural network considers that there is no mutation, 1 indicates a mutation on one of the copies of the chromosome and 2 indicates a mutation on both copies. Examples of this are shown in Figures 10 and 11: the input of the neural network is the $ratio_{signal}$, and the output is the predicted $genome_{signal}$ for the central position (i.e. the red cross in the examples).

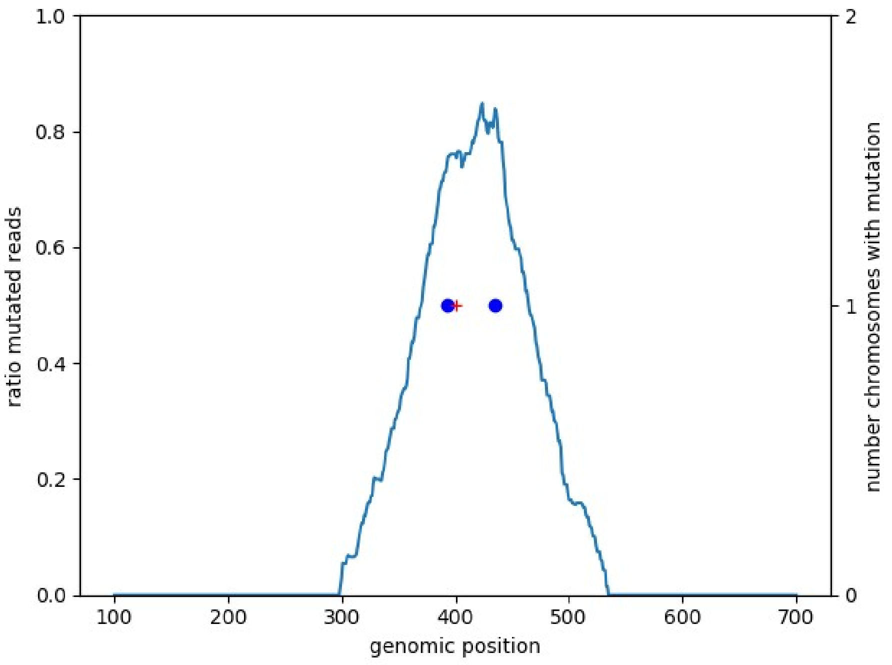


Fig. 10. Example of a wrongly predicted mutation when within the window formed by two mutations.

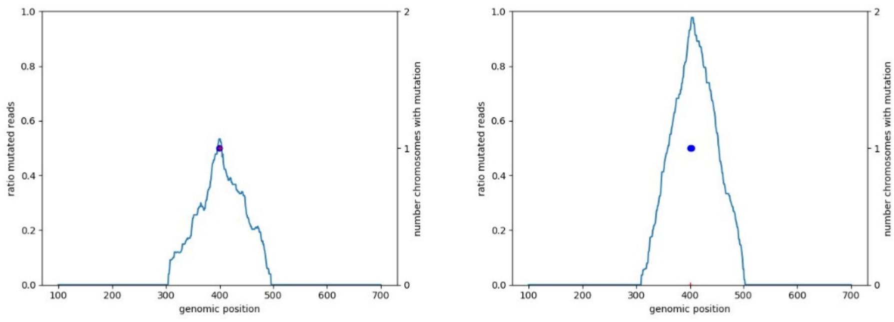


Fig. 11. Example of two misclassifications likely caused by the close proximity of a mutation.

As previously defined, the $ratio_{signal}$ is for the entire genome, and we want to obtain the entire $genome_{signal}$. Therefore, to accomplish the initial goal, we would need to apply the neural network to every position. In other words, we have to loop over the entire signal, applying the classifier (and thus the attack) to each position. Furthermore, from Figure 5, we can see that there are possible transient effects. In order to ensure that these effects are as limited as possible (and thus closer to what is possible using the entire $genome_{signal}$), we use an input window size big enough that no mutation outside the window could have an effect on the currently studied position. The information used for training and testing the neural network is obtained by randomly generating material from real information. The process is as follows: from the online database ensembl.org [7], a gene was randomly chosen, and the list of known mutations for the gene was obtained. For each position, we retain the most frequent mutation only. We also consider each mutation to be independent of one another. For each generated case, we select randomly one position p with a known mutation and generate two fake chromosomes: the chromosomes have size $read_{length} + read_{length} * 3 * 2 + 1$, and are an array of 0's, except for those positions corresponding to a location on the reference genome with a known mutation where we randomly put either 0 or 1, following the distribution indicated in the retrieved statistics. The mapping between the fake chromosome and the statistics is done in such a way that, after dropping the first $read_{length}$ values, the central position of the signal corresponds to p. The data dropped ensures that we have the transient effect of data prior to the input window present in the input window.

The overall architecture of the attack is still the same as shown in Figures 1 and 2. The same input data is used, the same input data is unavailable, and the same output data is generated. The only difference is the classifier we are using: previously we used a classifier solely based on convolutions, and now a classifier based on a neural network.

The distribution of reads is then simulated: using as input a target *coverage* (i.e. an average number of reads mapped to a given position), we determine how many reads are to be generated. Then, we select one of the two fake chromosomes, randomly select an initial position for the read, and then it is determined if it belongs to class P or class M. The neural network is trained to classify if for the given position there is no mutation, one mutation in one of the chromosomes or a mutation in both. By doing so we have simulated the content we are assuming is in plaintext, the content which is assumed to be encrypted is not even generated as it is not used in the attack. The neural network uses an output layer of three neurons, while there are five internal layers of sizes 256, 128, 64, 16, and 8 using the ReLu (Rectified Linear Unit) activation function [17]. We trained the neural network against different situations by varying the value of read length and coverage.

4 Results

As we have two chromosomes for each position (i.e. each entry in our testing material), there are three different possible outcomes: neither chromosomes have a mutation, one has a mutation or both have it. We take the arbitrary decision to refer with "positive" to the fact that there is at least one mutation. However, when measuring accuracy, we consider necessary to report the correct number of mutations for a given position (either 0, 1 or 2, depending on the mutation being present on neither chromosome, on one or on both). We report here the obtained results, after training the neural network on 16 batches of 200000 cases for 240 iterations each. The reported numbers are obtained by testing the neural network on an additional batch of 50000 cases.

Figure 12 shows the fraction of accurate results obtained across the different read lengths and coverages. However, as the input statistic we use leads to a low probability of having mutations, we should not take into account the correctly predicted negative results, as this will be correct approximately in 93.44% of the cases. Therefore, we turn ourselves to the F_1 score as shown in Figure 13.

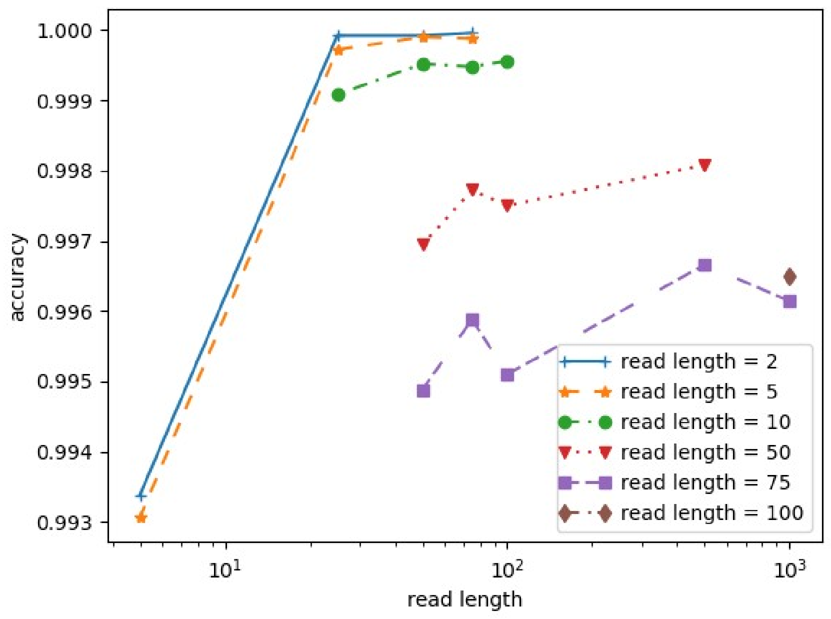


Fig. 12. Accuracy results obtained.

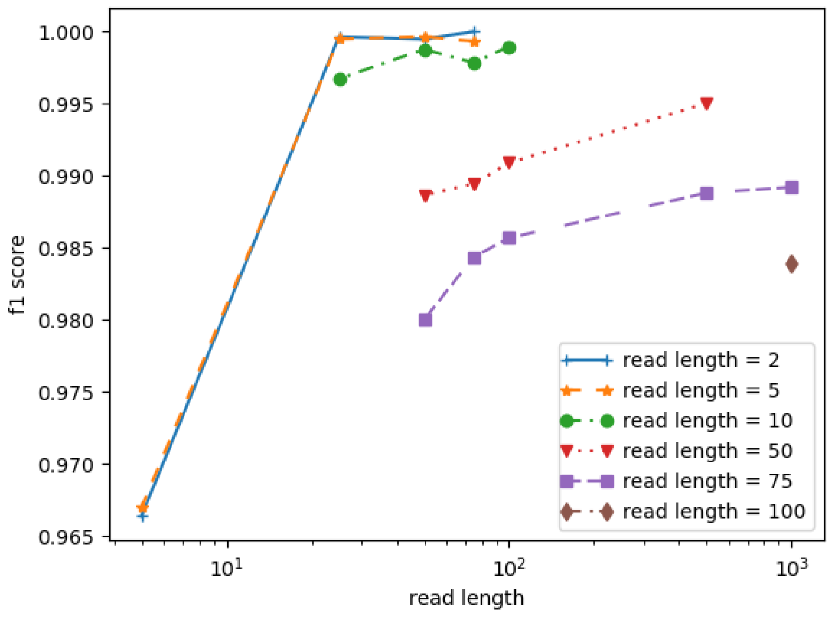


Fig. 13. F_1 score obtained.

5 Discussion

This section discusses the results obtained. It is worth noting that the smaller the read length, the less doubts there are around which position has a mutation. For example, in the case of a read having a length of two, there are only two possible positions which can be mutated. So, results need to be compared with our longer simulated reads of 100, where the evidence applies to 100 different positions. Therefore, we can expect to obtain better results with smaller reads, and indeed it is the case.

Similarly, if the coverage is higher, i.e. the average number of reads mapping to a position is higher, we can expect to have more evidence in the form of more reads. For example, in the case of testing a region with only one mutation which is not at the point of interest but which could be both mapped with one read, the results of the simulation could both be a read without mutations or with mutation. A higher coverage will translate in more evidence, which will increase the likelihood of having evidence for the absence of a mutation, thus simplifying the decision. As such, we expect the results to improve with higher coverage, and this is a trend we observe in the results. Nevertheless, we do not deduce every result correctly. As previously introduced, some of these errors can be linked to the problem of two mutations possibly shadowing a middle mutation. In this kind of situations, the network can only do a random guess. Figure 12 shows an example of this situation, where the neural network wrongly predicted a mutation (the circle indicates the input genome, the cross the classification output for the position of interest, and the plotted line the ratio of mutated reads for the given position). As there are more negative results than positive results, the neural network is likelier to return a negative result, as shown in Table 3.

	No mutation	Mutation	Total
Predicted no mutation	90.20% (true negative)	2.61% (false negative)	92.81%
Predicted mutation	5.56% (false positive)	1.63% (true negative)	7.19%
Total	95.76%	4.24%	

Table 3. Results for read length = 100, when the position of interest was within the window formed by two mutations.

Furthermore, this type of input where the position of interest is windowed, appears only in 0.67% of the cases with read length equal to 100, and even less for data with shorter reads. Therefore, this cannot be the main source of false categorizations. If we observe the results which were misclassified, we can see that many of them share the same characteristics of having a mutation close-by, as shown in Figure 14. If we take all misclassified outputs, and we observe the distribution of the distance between the closest mutation and the point of interest, we notice that the majority of incorrect results can be linked to close mutations: in more than 90% of the cases the closest mutation was 4 positions away, as can be observed in the distance histogram presented in Figure 14. Although we are not recovering all the data, we believe that these results are proving that, in the studied settings, the privacy of the information is at risk. On the one hand, we have obtained highF1scores, and, on the other hand, we have proven that, if we wrongly classified an input, it is likely that there is indeed a mutation in the neighboring positions. A read length value of 100 would be what is generated with an Illumina sequencer [4]. The recommended coverage can vary depending on the objectives. However it appears that the recommendations [4, 18, 19, 20] for a broader set of tests are between 33 and 100, as summarized in [21]. At this recommended coverage levels, we consider the attack to recover too much of the supposedly protected information, to consider the initial settings as privacy preserving.

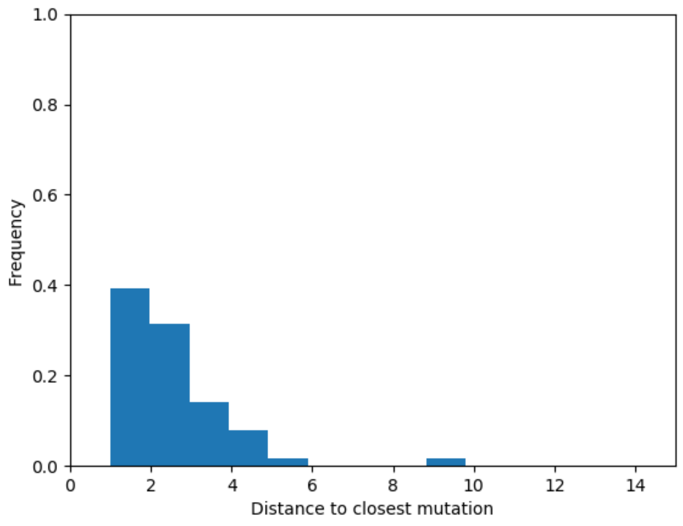


Fig. 14. Histogram of distance between the point of interest and the closest mutation, when the mutation was wrongly predicted and the read length is 100.

6 Future work

In order to improve this method, other neural networks could be employed. For example, and if the memory requirements can be met, a neural network predicting the classification at multiple locations at once, and using a prior round of predictions to refine the results of the next one could improve the obtained results.

In the here presented exercise we have not considered linkage disequilibrium: this metric measures the likelihood that two mutations appear together. If this information could be represented as input to the neural network alongside the classification of a linked mutation, the point of interest could be better predicted.

Finally, this work makes the assumption that the read length is constant and known. However, the reads length might be variable. In order to take this into account, the stream giving the read length should either be available as plaintext, or a maximum value for the read length should be assumed. Furthermore, we have not taken into account the possibility of paired reads: in this configuration two reads are paired together in the most general class necessary: if the two reads would have belonged to class P, then the pair belongs to class P, if the two would have belonged to class M then the pair belongs to class M, and if one of the reads would have belonged to class P and the other to class M, then the pair belongs to class M. This could be modelled in the here presented neural network by considering the two reads as a very long read, but, as we have seen, the length of the read reduces the amount of information recovered. Alternatively, both reads could be treated separately, but in this case it could happen that one read is mislabeled as providing evidence for mutations, increasing the noise present in the input.

7 Conclusion

With the initial approach of the privacy protection of MPEG-G, it was possible to encrypt only some of the blocks of information for a given access unit. We have here shown that taking advantage of this option, one could create a file where the mutation types and mutation positions were hidden, and consider the data to be private. We have shown, however, that this information can be partly recovered. The magnitude of the recovery depends on the read length and the coverage.

In order to defeat the here presented attack, the easiest path is to also encrypt the first position of each read. However, other attacks might be possible: other non-encrypted information might be used to recover other encrypted information. This fact led the privacy approach of MPEG-G to be redefined in order to force all streams in an access unit to be encrypted.

Acknowledgements
The work presented in this paper has been partially supported by the Spanish Research Agency/ERDF (EU), through the project Secure Genomic Information Compression (GenCom, TEC2015-67774-C2-1-R, TEC2015-67774-C2-2-R) and by the Generalitat de Catalunya (2017 SGR 1749).

References

1. Eric S. et al. Lander. Initial sequencing and analysis of the human genome. Nature, 409 (6822):860–921, feb 2001.
2. J. Craig et al. Venter. The Sequence of the Human Genome. Science, 291 (5507):1304–1351, feb 2001.
3. ISO/IEC JTC 1/SC 29/WG 11. MPEG-G, ISO/IEC 23092 Genomic Information Representation, 2019.
4. David R. et al. Bentley. Accurate whole human genome sequencing using reversible terminator chemistry. Nature, 456 (7218):53–59, nov 2008.
5. Peter J A Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer, and Peter M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Research, 38(6):1767–1771, dec 2009.
6. Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The Sequence Alignment/Map format and SAMtools. Bioinformatics, 25(16):2078–2079, aug 2009.
7. Andrew et al. Yates. Ensembl 2016. Nucleic Acids Research, 44(D1):D710–D716, jan 2016.
8. Markus Hsi Yang Fritz, Rasko Leinonen, Guy Cochrane, Ewan Birney, M. Hsi-Yang Fritz, Rasko Leinonen, Guy Cochrane, and Ewan Birney. Efficient storage of

high throughput DNA sequencing data using reference-based compression. Genome Research, 21(5):734–740, may 2011.

9. Morteza Hosseini, Diogo Pratas, and Armando J. Pinho. Cryfa: a secure encryption tool for genomic data. Bioinformatics, 35(1):146–148, jan 2019.
10. Zhicong Huang, Erman Ayday, Huang Lin, Raeka S. Aiyar, Adam Molyneaux, Zhenyu Xu, Jacques Fellay, Lars M. Steinmetz, and Jean-Pierre Hubaux. A privacy-preserving solution for compressed storage and selective retrieval of genomic data. Genome Research, 26(12):1687–1696, oct 2016.
11. Dale R Nyholt, Chang-En Yu, and Peter M Visscher. On Jim Watson's APOE status: genetic information is hard to hide. European journal of human genetics: EJHG, 17(2):147–149, mar 2009.
12. Mathias Humbert, Erman Ayday, Jean-Pierre Hubaux, and Amalio Telenti. Reconciling Utility with Privacy in Genomics. In Proceedings of the 13thWorkshop on Privacy in the Electronic Society - WPES '14, pages 11–20, New York, New York, USA, 2014. ACM Press.22
13. Global Alliance for Genomics & Health. Beacon Network. https://beacon-network.org/#/.
14. Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V Pearson, Dietrich A Stephan, Stanley F Nelson, and David W Craig. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. PLoS genetics, 4(8):e1000167, aug 2008.
15. Emiliano De Cristofaro, Sky Faber, and Gene Tsudik. Secure genomic testing with size- and position-hiding private substring matching. In Proceedings of the ACM Conference on Computer and Communications Security, pages 107–117, New York, New York, USA, 2013. ACM Press.
16. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422–426, jul 1970.
17. Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Journal of Machine Learning Research, 2011.
18. Han Fang, Yiyang Wu, Giuseppe Narzisi, Jason A ORawe, Laura T Jimenez Barrón, Julie Rosenbaum, Michael Ronemus, Ivan Iossifov, Michael C. Schatz, and Gholson J. Lyon. Reducing INDEL calling errors in whole genome and exome sequencing data. Genome Medicine, 6(10):89, dec 2014.
19. Michael J. Clark, Rui Rong Chen, Hugo Y K Lam, Konrad J. Karczewski, Rui Rong Chen, Ghia Euskirchen, Atul J. Butte, and Michael Snyder. Performance comparison of exome DNA sequencing technologies. Nature Biotechnology, 29(10):908–914, oct 2011.
20. Alison M. Meynert, Louise S. Bicknell, Matthew E. Hurles, Andrew P. Jackson, and Martin S. Taylor. Quantifying single nucleotide variant detection sensitivity in exome sequencing. BMC Bioinformatics, 14(1):195, dec 2013.
21. Genohub. Recommended Coverage and Read Depth for NGS Applications. https://genohub.com/recommended-sequencing-coverage-by-application/, 2019.