This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

Securing Aggregate Queries for DNA Databases

Mohamed Nassar, Qutaibah Malluhi, Mikhail Atallah, Abdullatif Shikfa

***Abstract*— This paper addresses the problem of sharing person-specific genomic sequences without violating the privacy of their data subjects to support large-scale biomedical research projects. The proposed method builds on the framework proposed by Kantarcioglu et al. [1] but extends the results in a number of ways. One improvement is that our scheme is deterministic, with zero probability of a wrong answer (as opposed to a low probability). We also provide a new operating point in the space-time tradeoff, by offering a scheme that is twice as fast as theirs but uses twice the storage space. This point is motivated by the fact that storage is cheaper than computation in current cloud computing pricing plans. Moreover, our encoding of the data makes it possible for us to handle a richer set of queries than exact matching between the query and each sequence of the database, including: (i) counting the number of matches between the query symbols and a sequence; (ii) logical OR matches where a query symbol is allowed to match a subset of the alphabet thereby making it possible to handle (as a special case) a "not equal to" requirement for a query symbol (e.g., "not a G"); (iii) support for the extended alphabet of nucleotide base codes that encompasses ambiguities in DNA sequences (this happens on the DNA sequence side instead of the query side); (iv) queries that specify the number of occurrences of each kind of symbol in the specified sequence positions (e.g., two 'A' and four 'C' and one 'G' and three 'T', occurring in any order in the query-specified sequence positions); (v) a threshold query whose answer is 'yes' if the number of matches exceeds a query-specified threshold (e.g., "7 or more matches out of the 15 query-specified positions"). (vi) For all query types we can hide the answers from the decrypting server, so that only the client learns the answer. (vii) In all cases, the client deterministically learns only the query's answer, except for query type (v) where we quantify the (very small) statistical leakage to the client of the actual count.**

***Index Terms*— DNA Databases, Cloud Security, Secure Outsourcing.**

I. Introduction

DNA or Deoxyribonucleic Acid is the medium of long-term storage and transmission of genetic information for all modern living organisms. Human DNA data (DNA sequences within the 23 chromosome pairs) are private and sensitive personal information. However, such data is critical for conducting biomedical research and studies, for example, diagnosis of pre-disposition to develop a specific disease, drug allergy, or prediction of success rate in response to a specific treatment. Providing a publicly available DNA database for fostering research in this field is mainly confronted by privacy concerns. Today, the abundant computation and storage capacity of cloud services enables practical hosting and sharing of DNA databases and efficient processing of genomic sequences, such as performing sequence comparison, exact and approximate sequence search and various tests (diagnosis, identity, ancestry and paternity). What is missing is an efficient security layer that preserves the privacy of individuals' records and assigns the burden of query processing to the cloud. Whereas anonymization techniques such as de-identification [2], data augmentation [3], or database partitioning [4] solve this problem partially, they are not sufficient because in many cases, re-identification of persons is possible [5]. It follows that the DNA data must be protected, not just unlinked from the corresponding persons.

In this paper, we consider the framework proposed in [1] where the DNA records coming from several hospitals are encrypted and stored at a data storage site, and biomedical researchers are able to submit aggregate counting queries to this site. Counting queries are particularly interesting for statistical analysis.

This paper provides a new method that addresses a larger set of problems and provides a faster query response time than the technique introduced in [1]. Our approach is based on the fact that, given current pricing plans at many cloud services providers, storage is cheaper than computing. Therefore, we favor storage over computing resources to optimize cost. Moreover, from a user experience point of view, response time is the most tangible indicator of performance; hence it is natural to aim at reducing it. Our method enhances the state of the art at both the conceptual level and the implementation level. More concretely:

- At the conceptual level, we provide a deterministic scheme, with zero probability of a wrong answer (as opposed to a low probability). This gives confidence to the users that they get exact results to all their queries, without impacting security.
- We also provide a new operating point in the space-time tradeoff, by giving a scheme that is twice as fast as theirs but uses twice the storage space. A variant of this scheme uses only 1.5 their storage space at the expense of additional latency.

Portions of this work were supported by NPRP grants from the Qatar National Research Fund (award number NPRP 09-622-1-090 and NPRP X-063-1-014); by National Science Foundation Grants CPS-1329979, CNS-0915436; and by sponsors of the Center for Education and Research in Information Assurance and Security. The statements made herein are solely the responsibility of the authors.

M. N., Q. M. and A. S. Authors are with KINDI Center for Computing Research, Doha, Qatar (e-mail: meb.nassar@gmail.com, qmalluhi@qu.edu.qa, and ashikfa@qu.edu.qa).

M. A. Author is with Department of Computer Science, Purdue University, West Lafayette, USA (e-mail: matallah@purdue.edu).

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

- Moreover, our encoding of the data makes it possible for us to handle a richer set of queries than exact matching between the query and each sequence of the database, including:
    i. Counting the number of matches between the query symbols and a sequence;
    ii. Logical OR matches where a query symbol is allowed to match a subset of the alphabet thereby making it possible to handle (as a special case) a "not equal to" requirement for a query symbol (e.g., "not a G");
    iii. Support for the extended alphabet of nucleotide base codes that encompasses ambiguities in DNA sequences (contrary to the previous item this happens on the DNA sequence side instead of the query side);
    iv. Queries that specify the number of occurrences of each kind of symbol in the specified sequence positions (e.g., two 'A' and four 'C' and one 'G' and three 'T', occurring in any order in the query-specified sequence positions);
    v. A threshold query whose answer is 'yes' if the number of matches exceeds a query-specified threshold (e.g., "7 or more matches out of the 15 query-specified positions").
    vi. For all query types we can hide the answers from the decrypting server, so that only the client learns the answer.
    vii. In all cases the client deterministically learns only the query's answer, except for query type (v) where we quantify the (very small) statistical leakage to the client of the actual count.
- At the implementation level, we take advantage of GMP modular arithmetic routines to achieve a much faster implementation of the approach in [1], as well as for the new approaches proposed in this paper.

II. Related Work

There is no universal method to create a protocol for secure multi-party computation and handling aggregate queries on encrypted data is not an exception. Several homomorphic systems only support a subset of mathematical operations, like addition (Paillier [19], Benaloh [23]), multiplication (ElGamal [24], RSA [25]), or exclusive-or (Goldwasser and Micali [26]). From a security perspective, only the additive Paillier and the multiplicative ElGamal are classified to be IND-CPA (stands for indistinguishability under chosen plaintext attack) [27]. Partially homomorphic cryptosystems are more desirable from a performance point of view than somewhat homomorphic cryptosystems, which support a limited operation depth. Fully homomorphic systems have a huge cost and cannot be deployed in practice.

Several works focus on protecting biometric computations over genomic sequence records in the context of secure multi-party computations (SMC). Secure outsourcing is a particular case of SMC where a client with low resources (energy, memory, CPU) requests the service of one or more outsourcing agents with abundant resources. Secure outsourcing finds a real projection in the current business models thanks to the proliferation of cloud-based services. Cloud computing and storage security issues have been subject to ostensive research in the past years [6]. Areas of interest include client authentication, hardware virtualization threats, flooding and denial of service attacks as well as issues of accountability, storage protection and computation protection. In the context of DNA data protection, related works can be divided into five groups depending on the function or the query being addressed: forensic databases, profile matching, sequence comparison, testing by finite automata and aggregate queries.

A. Forensic databases

In a forensic database, a suspect record has to be tested against an entire database. A record of the database can be decrypted only if it matches the suspect record. This protects the other records from being unveiled [7]. Similarly, negative databases prevent the enumeration of its members by reversely saving the non-members, in a compressed form [8].

B. Profile matching

In [9] the authors address a multitude of tests such as identity, ancestry and paternity tests based on Short Tandem Repeat (STR) profiles. The STR profile is composed of a number of loci and for each locus the number of repetitions for a given repeat structure. The authors translate each test into an algebraic expression and provide a homomorphic encryption scheme allowing two semi-honest parties to compare their stored profiles in a semantically secure manner. The proposed approach allows exact answers or small error tolerance as practically required by the tests.

C. Sequence comparison

The edit distance is the optimal cost of insertion, deletion and substitution of characters to go from a sequence λ to a sequence μ. The edit script is the chart of the steps leading to the optimal edit distance. Atallah *et al*. [10] offers a solution for securely outsource a dynamic programming solution for finding the edit distance and the edit script for two given sequences (particularly genomic sequences with small alphabet size). The outsourcing protocol is based on two non-colluding (honest-but-curious) agents that securely collaborate to performing table lookup and minimum finding. The secure minimum finding protocol determines the minimum of an additively split vector based on Yao's garbled circuits and a blind-and-permute protocol for hiding the index of this minimum. In [11] their scheme has been improved for performance and requires space only linear in the input size.

The work in [12] addresses a similar dynamic programming solution for finding the longest common subsequence. By using the "four Russians" technique in a new way, the authors propose a communication-efficient SMC protocol that improves over the generic solution based on Yao's garbled circuits. Their results feature an asymmetry in the work required by each participant, which makes it more suitable to

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

an outsourcing scenario. In [13] the authors address the longest common subsequence as a private search problem.

Another example of genome sequence comparison is the Smith-Waterman algorithm which performs local sequence alignment. In [14] the authors transform the formulation of this algorithm for crowdsourcing (i.e., outsourcing to distributed volunteers). Their scheme, based on computation with obscured data, preserves a reasonable level of accuracy but does not provably protect the privacy of the inputs.

D. Sequence testing by finite automata

Sometimes the queries on DNA need to take into account various errors such as irrelevant mutations, incomplete specifications and sequencing errors. Therefore, the pattern of the query should be expressed using regular expressions. Many works address practical and privacy-preserving outsourcing of this regular expression type of queries, implemented as oblivious evaluation of finite automata [15]–[17].

E. Aggregate queries

For biomedical researchers, important queries have often the form "How many records contain a diagnosis of Alzheimer disease and gene variant X?" Secure outsourcing of the database and allowing such type of queries without requiring the server to decrypt the data has been addressed in [1]. The paper presents very practical results. For example, a count query over 40 records in a database of 5000 records takes 30 minutes. Our paper extends these results by proposing a variant storage and computation scheme.

III. Problem Definition and Framework

Computer scientists often represent DNA by a large sequence of characters from the alphabet $\Sigma = \{A, C, G, T\}$, representing the four nucleotide types. This alphabet can be augmented with additional characters representing ambiguity in the sequence. This extended alphabet is denoted by $\Sigma' = \{A, C, G, T, N, M, R, W, S, Y, K, V, H, D, B\}$ as defined by IUPAC [18], see Table 1. Given a database of d sequences $s_1, s_2, \ldots, s_d$ each having m characters; the query is represented as a list of tuples (j_i, v_i) of characters v_i and positions j_i; for $i = 1..k$. The result of the query is the number of sequences where $s[j_i] == v_i$ for all the tuples (j_i, v_i). The pseudo (Python-like) code of a query in clear is shown in Listing 1.

TABLE 1
NUCLEOTIDE BASE CODES (IUPAC)

Symbol	Nucleotide Base
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
N	A or C or G or T
M	A or C
R	A or G
W	A or T
S	C or G
Y	C or T
K	G or T
V	Not T
H	Not G
D	Not C
B	Not A

In our model, hospitals who have DNA sequences do not have the computing and processing capabilities to process researchers' requests, so they all store their DNA sequences at a server (which is also more convenient to do queries across all hospitals). The clients, who are typically researchers, query the server to obtain statistics on the occurrence of a given subsequence in the pool of DNA sequences stored on the server. Due to the sensitivity of DNA, all these operations have to be performed securely: the goal of securing queries is making both the client and the server ignorant of exactly which sequences match the query but only knowing the aggregated result of the query (i.e., the count).

LISTING 1
PSEUDO-CODE OF AN AGGREGATE QUERY

```
#Example of a query:
q=[(A,0), (A,2), (C,3), (G,6)]
#Example of sequence matching the query:
#AAACAGG
#D is the set of all sequences
count=0
for s in D:
   match=True
   for (v,j) in q:
     if s[j]!=v:
       match=False
       Break
   if(match):
     count+=1
```

To be more precise the security model is as follows:

- Hospitals want to protect the confidentiality of the DNA sequences that they own and no external party has the right to access these DNA sequences for privacy reasons. Thus, other parties (be it the server or the clients) should only work on encrypted sequences and never have access to the DNA cleartext.
- The server is an external repository of DNA sequences provided by the various hospitals. The server is considered honest-but-curious by hospitals: hospitals trust him to perform the queries requested by clients but they do not want the server to access the DNA sequences in clear.
- Clients are entities authorized to perform queries on the

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

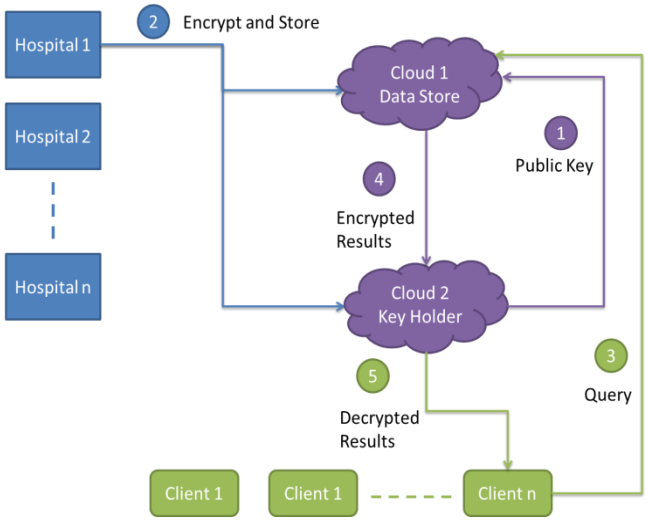

Fig. 1. Framework of secure aggregate queries over encrypted DNA database

database of encrypted DNA sequences. They are only allowed to obtain statistics on the database: the number or percentage of sequences matching a given query. The queries are not confidential and are processed by the server, however the server should not know the outcome of the queries.

- We assume that none of these entities collude.

Additively homomorphic encryption is suitable for the purpose of performing count statistics on encrypted data. Paillier's homomorphic encryption [19] possesses the following properties: (i) It's a public key scheme, which means encryption can be performed by anyone who knows the public key, whereas decryption can only be done by the matching private key, known only to a trusted party. (ii) It is probabilistic. In other words, it is impossible for an adversary to tell whether two ciphertexts are encryptions of the same plaintext or not. (iii) It possesses the homomorphic properties for addition, in particular:

- $E_{pk}((m_1 + m_2)\ mod\ N) = E_{pk}(m_1) * E_{pk}(m_2)\ mod\ N^2$
- $E_{pk}\big((a * m_1)\ mod\ N\big) = E_{pk}(m_1)^a\ mod\ N^2$

Where N is the modulus of the encryption and a part of the public key. Note that the sign " = " above stands for equivalence not equality.

We consider a framework similar to [1] composed of several hospitals, several clients representing biomedical researchers and two non-colluding servers (can be two different cloud providers, or one cloud provider and one trusted host). In Fig. 1 we call these two servers *Cloud1* and *Cloud2* to emphasize that the framework can be deployed in a cloud environment:

- Cloud1 represents the data store where all the encrypted DNA records are stored and is responsible of processing the queries.
- *Cloud2* is a trusted party that generates and holds the private and public keys of the homomorphic encryption scheme. In step 1 the public key is sent to the other parties. Cloud2 is later used as a decyption oracle and it also shares security associations with the *clients* in order to send them the results securely.
- The hospitals obtain the public key in order to encrypt their DNA records and upload them to *Cloud1* (step 2).
- A client representing a biomedical researcher submits a query to *Cloud1* (step 3). The cloud processes the query over the encrypted records and sends the results to *Cloud2* in order to be decrypted (step 4). *Cloud1* is required to permute the results for individual records before sending them out. The permutation protects the records if in any case the order of the records can be linked to some protected information. Finally the client receives from *Cloud2* the decrypted count of matches (step 5) through a secure channel (built thanks to the security association etablished at step 1).

Cloud2 may assist the data encryption at the data owners (the hospitals) through pre-encrypting a large number of values for the encoding of each letter in the alphabet and transferring them to the data owners.

IV. Storage and Computation Schemes

A. Summary of the scheme in [1]

The proposed protocol is based on a binary storage scheme. Each letter has a binary representation over two bits and each bit is encrypted using Paillier encryption. For example the letter 'A' is coded in binary as two bits 00. Similarly the query is translated to binary encoding. For example finding the letter 'A' at position 6 is equivalent to finding the bit 0 at position 12 in the encoded sequence and the bit 0 at position 13 in the encoded sequence. Therefore, the required storage capacity for a sequence is $2 * m * 2b$ where m is the length of the sequence and $2b$ is the size for storing an encrypted value (b is the bit length of the key modulus). The query is computed as an algebraic expression that evaluates to an encryption of 0 for each record matching the query. Two random numbers are used in order to limit false positives. Without loss of generality, consider an encoded sequence s and a query of the form $(j_i, 1)$, for $i = 1..t$ and $(j_i, 0)$, for $i = t + 1..2k$ where k is the length (i.e., number of letters) of the query; the server computes an expression of the form:

$$R(q,s) = \left(\left(\prod_{i=1}^{t} \mathrm{E}(\mathrm{s}[\mathrm{j_i}])\right) * \mathrm{E}(-\mathrm{t})\right)^{\mathrm{r_1}} * \left(\prod_{i=t+1}^{2k} \mathrm{E}(\mathrm{s}[\mathrm{j_i}])\right)^{\mathrm{r_0}}$$

Where r_1 and r_0 are random numbers. If s matches the query, the result of this expression is an encryption of zero with a high probability. The server sends a permutation of the results of expressions for all the sequences $s_1, \ldots, s_d$. The key holder decrypts and counts the zeros to obtain the result of the query. Note that the number of modular exponentiation required is equal to 2, in addition to $2k$ modular multiplications.

B. Our Schemes

We present two different schemes; the first one requires more storage capacity but provides better query response time

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

than the second one.

1) Quaternary storage, quaternary query

We encode a sequence $s = [L_i]$, $i = 1..m$, using four vectors:

- $s_A = [1\ if\ L_i ==' A',\ 0\ otherwise,\ i = 1..m]$
- $s_C = [1\ if\ L_i ==' C',\ 0\ otherwise,\ i = 1..m]$
- $s_G = [1\ if\ L_i ==' G',\ 0\ otherwise,\ i = 1..m]$
- $s_T = [1\ if\ L_i ==' T',\ 0\ otherwise,\ i = 1..m]$

For example, sequence "CCGATAT" is encoded as:

- $s_A = [0, 0, 0, 1, 0, 1, 0]$
- $s_C = [1, 1, 0, 0, 0, 0, 0]$
- $s_G = [0, 0, 1, 0, 0, 0, 0]$
- $s_T = [0, 0, 1, 0, 1, 0, 1]$

The four vectors representing each sequence are encrypted and uploaded to *cloud1*. Therefore, the required storage capacity for a sequence is $4 * m * 2b$ where *m* is the length of the sequence and *2b* is the size for storing an encrypted value. Note that this encoding enables us to support the extended alphabet contrary to the scheme of [1]. Indeed if there is ambiguity at a given position, say position i_0 and the letter at this position is M which stands for A or C, we can simply define $s_A[i_0] = s_C[i_0] = 1$.

A query q=(j_i, v_i), $i = 1..k$ is decomposed into four queries, and represented by four vectors as follows:

- Initialize q_A to a vector of m zeroes; then assign $q_A[j_i] = 1\ if\ L_i ==' A'$ in the query, $i = 1..k$
- q_C is a vector of m zeroes; $q_C[j_i] = 1\ if\ L_i ==' C',\ i = 1..k$
- q_G is a vector of m zeroes; $q_G[j_i] = 1\ if\ L_i ==' G',\ i = 1..k$
- q_T is a vector of m zeroes; $q_T[j_i] = 1\ if\ L_i ==' T',\ i = 1..k$

Note that although we used *m* as the size of the query to ease the presentation and understanding, the knowledge of *m* is not required at the client side (moreover m may vary from one sequence to another). The query simply needs to be as long as the position of the last element in the subsequence of the query (v_k). The positions after that will all be automatically assumed as containing 0. The query is then computed over an encrypted sequence using the following equation:

$$R(q,s) = E(q_A s_A + q_C s_C + q_G s_G + q_T s_T) =$$

$$E\left(\sum_{i,q_A[j_i]=1} s_{A,j_i} + \sum_{i,q_C[j_i]=1} s_{C,j_i} + \sum_{i,q_G[j_i]=1} s_{G,j_i} + \sum_{i,q_T[j_i]=1} s_{T,j_i}\right) = \prod_{i=1..k} E\left(s_{L_i,j_i}\right)^{q_{L_i,j_i}}$$

Note that since q_{L,j_i} is either 0 or 1, no modular exponentiation is needed during the computation of *R* but only *k* modular multiplications. The result of the equation decrypts to exactly *k* if the sequence matches the query.

2) Ternary storage, quaternary query

Since the presence of a letter in a given position of a sequence can be directly inferred by the absence of the three other letters, we can reduce the encoding to only three vectors:

- $s_A = [1\ if\ L_i ==' A',\ 0\ otherwise,\ i = 1..m]$
- $s_C = [1\ if\ L_i ==' C',\ 0\ otherwise,\ i = 1..m]$
- $s_G = [1\ if\ L_i ==' G',\ 0\ otherwise,\ i = 1..m]$

If we retake the same example, sequence "CCGATAT" is encoded as:

- $s_A = [0, 0, 0, 1, 0, 1, 0]$
- $s_C = [1, 1, 0, 0, 0, 0, 0]$
- $s_G = [0, 0, 1, 0, 0, 0, 0]$

In other words, the letter 'A' is encrypted by the column vector [1,0,0], the letter 'C' by [0,1,0], the letter 'G' by [0,0,1] and the letter 'T' by [0,0,0]. The encoding can be changed to use column vector [0,0,0] to encode the least frequent letter. This would improve the query performance as demonstrated later by the query computation formula. In this scheme, the required storage capacity for a sequence is $3 * m * 2b$. This scheme does not support the extended alphabet though.

Similar to the previous scheme, a query $q=(j_i, L_i)$, $i = 1..k$ is decomposed into four queries, and represented by four vectors:

- q_A is a vector of m zeroes; $q_A[j_i] = 1\ if\ L_i ==' A'\ i = 1..k$
- q_C is a vector of m zeroes; $q_C[j_i] = 1\ if\ L_i ==' C',\ i = 1..k$
- q_G is a vector of m zeroes; $q_G[j_i] = 1\ if\ L_i ==' G',\ i = 1..k$
- q_T is a vector of m zeroes; $q_T[j_i] = 1\ if\ L_i ==' T',\ i = 1..k$

The query is then computed over an encrypted sequence using the following equation:

$$R(q,s) = E(q_A s_A + q_C s_C + q_G s_G + q_T s_T) =$$

$$E\left(\sum_{i,q_A[j_i]=1} s_{A,j_i} + \sum_{i,q_C[j_i]=1} s_{C,j_i} + \sum_{i,q_G[j_i]=1} s_{G,j_i} + \sum_{i,q_T[j_i]=1} \left(1 - \left(s_{A,j_i} + s_{C,j_i} + s_{G,j_i}\right)\right)\right) = \prod_{i=1..k} E\left(s_{L_i,j_i}\right)^{q_{L_i,j_i}}$$

Where $s_{T,j_i} = 1 - \left(s_{A,j_i} + s_{C,j_i} + s_{G,j_i}\right) \Rightarrow E\left(s_{T,j_i}\right) = \left(E\left(s_{A,j_i}\right)E\left(s_{C,j_i}\right)E\left(s_{G,j_i}\right)\right)^{-1}E(1)$. In case all the letters have the same frequency in the query (i.e., the number of 'T' letters in the query is $k/4$), the computation of R requires $1.75k$ modular multiplications and $0.25k$ modular multiplicative inversions. The resulting value decrypts to exactly k if the sequence matches the query.

The two proposed schemes accounts for approximate matches which is very useful in practice because of many sources of error such as genome synthesis error. For example if $D\left(R(q,s)\right) = k - 1$ then the query matches the sequence with only one error at one of the query positions.

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

TABLE 2
SPACE-TIME COMPARISON

	Binary Mode [1]	**Ternary Storage Quaternary Query**	**Quaternary Storage Quaternary Query**
Storage	$4mb$	$6mb$	$8mb$
Query Time	$2k * MM + 2 * ME + AC$	$1.75k * MM + 0.25k * MI + ME + AC$	$k * MM + ME + AC$
Query Time with pre-computation	$2k * MM + 2 * ME$	$1.75k * MM + 0.25k * MI + ME$	$k * MM + ME$
Query Answer	Match/No-Match	Match/No-Match except for query (iv)	Match/No-Match except for query (iv)
Approximate Matches	No	Yes	Yes
Extended alphabet (ambiguity)	No	No	Yes
Probability of error	Small	Deterministic (Zero)	Deterministic (Zero)

3) Match/No-Match answer

Nevertheless the scheme can output a binary result of the query: $R'(q,s) = (R(q,s) * E(-k))^r$ where r is a random non-zero number. Note that one modular exponentiation is needed in this case. R' decrypts to 0 if a match is found and to a random number if not. Modular multiplication by a random perfectly hides the answer (except for 0) and has been widely used in blind signature protocols [20].

4) Space-time comparison

We consider one sequence as a comparison unit and show in Table 2 the space and time costs. We consider only queries and sequences over Σ to be able to compare our scheme against [1], as [1] does not support queries over Σ'. In terms of storage we assume all the sequences have the same length *m*. *2b* is the size of an encrypted value using Paillier encryption and given modulus *N* of bit length *b*. Our schemes require three or four encrypted words per letter compared to two encrypted words for [1].

For the computation cost, *k* is the size of the query, *MM* is the time for modular multiplication, *MI* is the time for modular multiplicative inversion, *M*E is the time for modular exponentiation and *AC* is the time for adding a constant to a ciphertext. *MM*, *MI*, *ME* and *AC* are the major computation factors.

The *AC* time represents the time to add $-t$ (in binary mode) and $-k$ (in both quaternary modes) under encryption. Since *t* and *k* are constant for a given query, $E(-t)$ and $E(-k)$ can be pre-computed once for all the sequences. In practice *k* and *t* are small (less than *m*) and an encryption of all their possible values can be pre-computed[1]. The value required by a given query can be fetched and used in computing *R* for all the sequences.

The gain ratio of the quaternary storage scheme compared to the binary mode is equal to 2. For the ternary storage scheme, *MM* is much smaller than *ME* and can be ignored (especially when the exponent parameter in *ME* has the same bit length of the modulus)**.** *MI* is also smaller than *ME* but cannot be ignored. The gain ratio is on the average $2 * ME/(0.25k * MI + ME)$.

In conclusion, we have presented two new operating points in the space-time tradeoff of the private query problem, by giving two schemes that are up to twice as fast as [1] but uses 1.5 to 2 times their storage space.

[1] Precomputation is widely used in cryptography, for example to speed up fixed-base modular exponentiation [22].

Note that aside from performance aspects, our proposed schemes have other advantages such as the support of ambiguity in the queries for both our schemes (see paragraphs 7 and 8 below) and the quaternary storage solution supports the full extended alphabet of nucleotides Σ' at server side as well. Our schemes are also deterministic because we compute the exact number of matches as we have one different array for each symbol of Σ, while the scheme of [1] gives only probabilistic answer as they use binary mode to save space. As a result, it might happen that an incorrect match at a given position is cancelled out by another incorrect match at another position, although the probability is very small because different random numbers are used each time. The probability of a wrong answer is of $\frac{1}{n-1}$ for a single query, which is negligible for large n, while the probability of having at least one false query result is $1 - \left(\frac{n-2}{n-1}\right)^q$ for q queries.

5) Security evaluation

From a security perspective the framework that we use is similar to [1]. Both schemes are based on well-known security building blocks like Paillier's encryption, public key encryption and symmetric key encryption. This is why our security evaluation focuses on the interconnection of these building blocks and what can go wrong in our settings. To be more precise:

- The DNA sequences are always encrypted at *Cloud1*, so *Cloud1* cannot access these sequences in clear. The only entity which could decrypt them is *Cloud2* which is a trusted entity by the hospital (again with named Cloud2 to emphasize that it can be deployed in the cloud but it is very different from cloud1 in that it performs minimal operations (decryption oracle) and does not have to store huge data, so it is a specific trusted platform in the cloud. The confidentiality of DNA sequences is thus correctly preserved.
- *Cloud1* also does not get any leakage from the queries of clients because he processes the queries in an encrypted (he cannot decrypt the outcome). The result is decrypted by *Cloud2* which sends the result directly to the client who made the query through a secure channel thanks to the security associations between *Cloud2* and the clients. This also means that *Cloud1* cannot act as a *Client* and get the result of his own queries, unless he colludes with a real Client, which is out of scope of our model.
- Clients only obtain statistics on the number of DNA sequences across all hospitals which match their query but they don't get the DNA sequences themselves. Learning

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

the number of sequences is a leakage that is acceptable in our model as this leakage would happen even with the ideal model of a trusted entity doing all the processing between hospitals and clients.

- The only entity which really has an edge in our framework is *Cloud2* as it owns the private key. We argue however that:
  - *Cloud2* is a trusted entity
  - *Cloud2* does not have access to encrypted DNA sequences unless he colludes with Cloud1 or a Hospital
  - *Cloud2* sees the queries and the outcome of the queries on each DNA sequence individually so potentially he has a higher leakage than the clients. However to avoid this situation, the Clients can collude with *Cloud1* to prevent *Cloud2* from getting the outcomes on each sequence individually, as explained in the next paragraph.

6) Hiding from the decrypting server

The client and *Cloud1* can collaborate to hide the answer from the decrypting server *Cloud2*. The client and *Cloud1* exchange a seed for a pseudo-random generator. For each (query, sequence) tuple the client and *Cloud1* synchronize to independently generate the same random number *r1* (distinguishing it from *r* in the previous section). *Cloud1* adds *r1* to the answer for the query under encryption: $R'' = R' * E(r1)$. *Cloud1* decrypts R'' but cannot discover the answer since it doesn't know *r1*. The client receives the decryption of R'' which is equivalent to $D(R') + r1$, and subtracts *r1* to obtain the result.

Table 3 presents an analysis of the information exchanged between the different entities and their impact on security.

TABLE 3: PER-ELEMENT SECURITY ANALYSIS

Element	Description	Analysis
N	Part of the public key. It reveals the length of plaintext and the space of ciphertext.	As this is part of the public key, anybody should be able to see N. Revealing the ciphertext space is not considered a major issue in the literature.
E(M)	Paillier's encryption of a message M.	This encryption is malleable. An adversary intercepting a message can change it. But since we typically assume that all communications between entities are secured in a classical sense, for example they are performed over TLS, a message modification attack is not possible. Otherwise Paillier's scheme is IND-CPA.
Seed_C	This seed is shared between cloud 1 and a client C. It has for goal to hide the results from the decryption oracle (cloud 2).	We choose the seed as a secret key (randomly and with length 128 bits at least), and we use a secure PRNG (for example one which is derived from a symmetric block cipher). Obtaining the seed is as difficult as obtaining the secret key of an encrypted message which is computationally infeasible with appropriately long seed (or key) size. Hence there is no leakage at Cloud2.

Finally, our scheme takes benefit of existing security infrastructure which is normally available at a cloud service provider such as authentication, confidentiality and integrity.

7) Set match query

This helps supporting ambiguity from the query side

- Example: q=[((A|C),0), (G,1), (T,2), (G,3), ((C|T),4) (T,5)]
- Solution: Put 1 at position 1 in q_A and 1 at position 1 in q_C. Put 1 at position 5 in q_C and 1 at position 5 in q_T. Encode the remaining letters of the query as in the initial scheme.

8) Negation query

- Example: q=[((!A),0), (C,1), ((!A^!T),2), (T,3)]
- Solution: Put 1s in q_C, q_G and q_T at position 1. Put 1 in q_C *and* q_G at position 3, because (!A^!T) = (C|G)

9) Exactly (k'<k) matches

In this case we compute $R' = (R(q,s) * E(-k'))^r$ where r is a random number. $D(R') == 0$ if exactly k' matches are found.

10) Exactly (a As , c Cs , g Gs and t Ts)

This query can be computed as follows: $R' = \left(E(q_A s_A)E(-a)\right)^{r_1}\left(E(q_C s_C)E(-c)\right)^{r_2}(E(q_G s_G)E(-g))^{r_3}$ $(E(q_T s_T)E(-t))^{r_4}$

Where r_1, r_2, r_3 and r_4 are random number. R' decrypts to 0 if the query is matched. Four modular exponentiations are needed in this case.

11) At least k' matches out of k

We compute $R' = (R(q,s) * E(-k'))^r$ where r is a positive random number. If R' decrypts to 0 we have exact match of k'. if D(R') < N/2 we have more than k' matches, if D(R') > N/2 we have less than k' matches, assuming r is chosen within a sufficiently small margin ($r * \Delta << N/2$), where Δ is the estimated maximum difference $D(R) - k'$ for all the records.

V. EXPERIMENTAL EVALUATION

We have implemented a prototype for evaluating the two outsourcing schemes: binary mode [1] and quaternary mode. By quaternary mode we refer to quaternary query encoding. In our experiments we use ternary storage. However, our discussion includes comments on the expected results for quaternary storage. The implementation uses the Python language and the Gmpy2 library[2] for supporting arithmetic operations in Paillier's cryptosystem. To simplify the implementation, we used the special case of Paillier's cryptosystem where p and q are two primes of equivalent bitlength, $N = pq;\ g = N + 1,\ \lambda = \varphi(N)$ and $\mu = \varphi(N)^{-1} \bmod N$, where $\varphi(N) = (p-1)(q-1)$. Table 4 describes the different parameters of our experiments. These experiments are run on one processor Intel Xeon CPU 2.90GHz on a Linux server machine.

[2] Gmpy2 is C-coded Python extension modules that support fast multiple-precision arithmetic (https://pypi.python.org/pypi/gmpy2).

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

TABLE 4
PARAMETERS OF THE EXPERIMENTS

Parameter	*Description*
d	Number of records in the database
m	Length of a record
b	Public key modulus size in bits
k	Query size
Mode	(B) for binary, (Q) for quaternary

A. *Response time*

We first evaluate our implementation by running 1000 operations on randomly generated numbers for each type of the basic operations of the cryptosystem. The results are shown in Fig. 2. Note that adding and inverting ciphertexts (*MM* and *MI*) are small compared to *ME* and *AC*. In fact modular exponentiation have $O(b.M(b))$ complexity, where $M(b)$ is the complexity of modular multiplication and depends on the used multiplication algorithm (textbook multiplication is $O(b^2)$) [21]. We show the results of 10,000 operations for *MM* and *MI* in the histogram instead of 1000 operations to show that *MI* is relatively more significant than *MM*. Key generation is done once at setup and has recourse to the primality test in Gmpy2 (taking much less than 1s for *b=1024)*.

We second evaluate the query response time. In these experiments we ignore the input/output time and only account for the time of computational operations (this situation is legitimate for scenarios where access time is constant). In Fig. 3, the query response time for the two modes and different database sizes is depicted (k=[10, 20, 30, 40], d=[5000, 10000, 15000, 20000], m=300, b=1024).

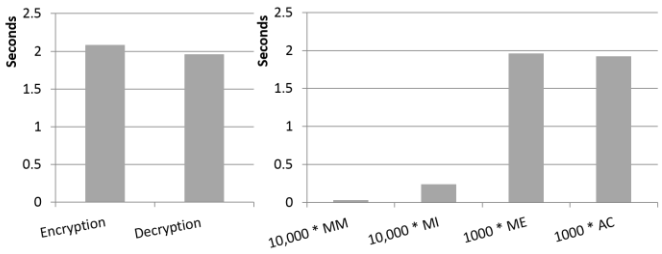

Fig. 2. Performance evaluation of implemented Paillier's basic operations (*b=1024*): cost of 1000 operations on randomly chosen operands

As expected, Quaternary mode (ternary storage) is faster with a speed up of approximately 2 for small query sizes. The gain ratio decreases when the query size increases because the number of MI increases proportionally. For this set of experiments the gain ratio ranges between 1.87 and 1.98. Note that if quaternary storage has been used, this ratio would be constantly 2. In both modes, the query response time increases linearly with the database size. In binary mode the query size affects marginally the response time because it only increases the number of *MM* operations having relatively small cost.

In the third experiment we study the effect of the key size on the execution time of the queries *(k=20, d=10000, m=300, b=[64, 128, 512, 1024, 2048])*. Theoretically the query execution time is $O(b.M(b))$ where b is the size of the key. Fig. 4 shows that an approximate speed up of 2 is maintained as the key size varies. For the choice of the key, a size of 1024 bits is fairly considered as semantically secure.

Fig. 5 (log scale) shows the database encryption time, which is also $O(b.M(b))$. The quaternary mode (ternary storage) time is 1.5 the time of the binary mode since it requires three encryptions per letter, compared to only two for the binary mode (in quaternary storage this ratio becomes 2). Note that database encryption is done only once at setup time.

For decryption (Fig. 6), the binary mode requires exactly two decryptions per letter while the quaternary mode (both ternary and quaternary storage) takes 2.25 decryptions on average if the letters are uniformly distributed (it stops if the decrypted value is one or continues till the three decryptions are done: $0.25 * 1 + 0.25 * 2 + 0.5 * 3$). The order of decryption can be changed according to the frequency distribution of the values in the database if known to be non-uniform.

The experiments show that encryption/decryption is roughly twice longer in the quaternary mode compared to the binary mode (although decryption is less than twice the time), but query time is twice faster. Query is the operation which has to be performed many times as opposed to encryption which is performed only once at set up time.

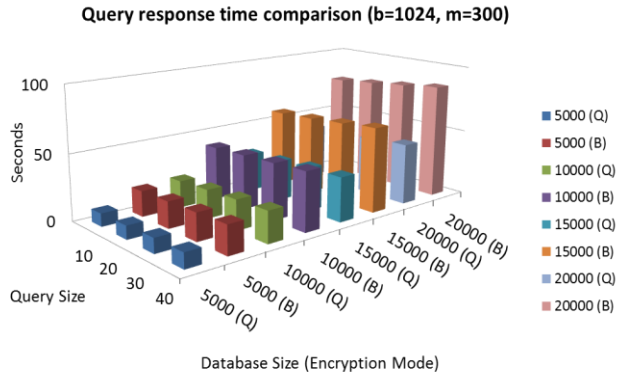

Fig. 3. Comparison of query response time for Binary (B) mode and Quaternary (Q) mode

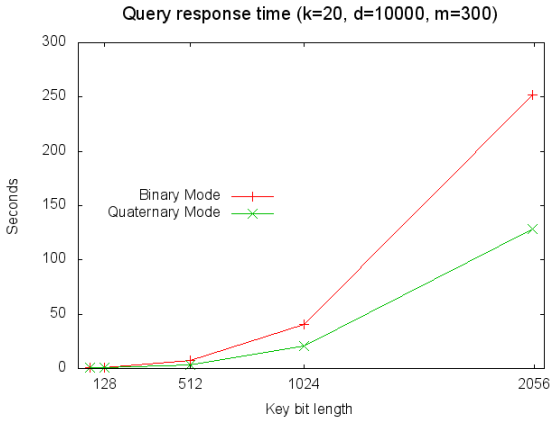

Fig. 4. Comparison of query response time for different key bit length

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

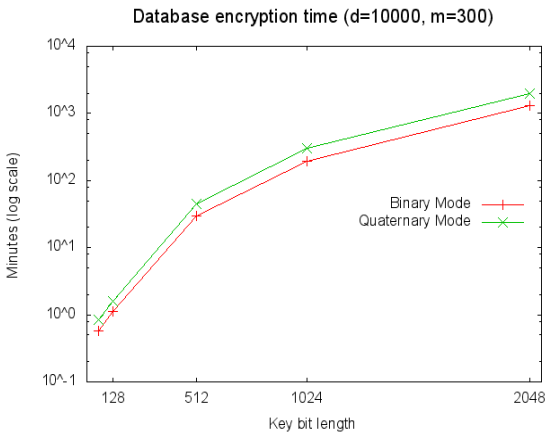

Fig. 5. Comparison of database encryption time for different key bit length

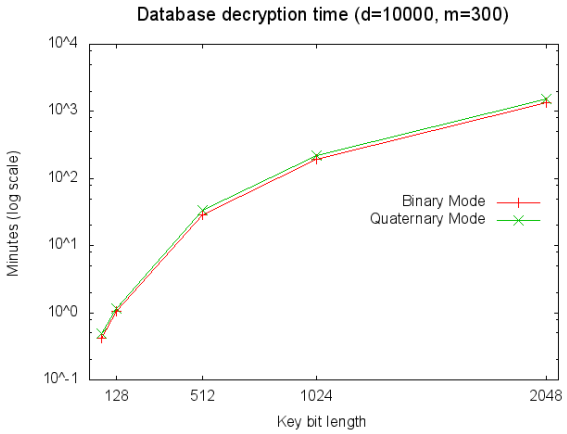

Fig. 6. Comparison of database decryption time for different key bit length

Note that all our experiments are reproducible by downloading the source code of the tool "private_dna" and the Paillier library at the following checkout URL: https://github.com/mnassar/private-dna-queries.

B. *Experiments with real DNA data on the cloud*

We have also performed experiments on real DNA data. We use a genomic data set from the UCSC Genome Browser[3]. We have used a prepackaged download of 1000 base pairs (bp) upstream sequences of annotated transcription starts of RefSeq genes with annotated 5' UTRs. The upstream sequences typically have the control elements for gene transcription. Therefore, finding the same upstream pattern in multiple sequences is important because it could imply that the corresponding genes are co-regulated.

The dataset contains the starts of 41,782 sequences up to lengths of 300 letters in one subset and 500 letters in another. The dataset has a 5 letters alphabet: A, C, G, T and N. Table 5 shows the distribution of these letters in the data set.

TABLE 5
FREQUENCIES OF LETTERS IN UPSTREAM1000 DATA

Length	*A*	*C*	*G*	*T*	*N*
300	20.5%	29.7%	29.4%	20.4%	0.02%
500	21.8%	28.4%	28.1%	21.6%	0.02%

Notice the presence of the N indicating an unknown base in the DNA sequence. The encoding of N can be incorporated easily in our quaternary scheme by putting 1s in the four encoding vectors, whereas it is impossible in the binary mode without increasing the number of encoding bits.

We have experimented with this data set using Amazon web services. The experiments are run on a m3.xlarge instance (4 vCPU 15 GiB 2 x 40 GB SSD storage) with high frequency Intel Xeon E5-2670 v2 (Ivy Bridge) processors (25M Cache, 2.50 GHz). The DNA dataset is stored in-memory using a REDIS4 database.

[3] https://genome.ucsc.edu/

[4] Redis is an open source, in-memory data structure store, used as database, cache and message broker (http://redis.io/)

Fig. 7 shows the response time for a match/no-match query of 40 loci with different key sizes comparing our approach with the binary scheme in [1] (In the binary scheme we ignore the letter 'N'). We show the total cumulative CPU time of the different cryptographic functions: e_add (which is a modular multiplication MM), e_add_const (AC) and e_mult_const (which is a modular exponentiation ME). By cumulative time we mean the time spent in a function and all sub functions invoked from that function. By total time we mean the time for all the calls to a function during a query computation. We also measure the overall cryptography functions wall clock time and the overall query response wall clock time. The wall clock time is governed by the crypto time and the Redis server access time. We reduce the number of accesses to the Redis in-memory server to only one access per record. Redis Mget allows obtaining values for multiple keys in one access. So the number of Redis accesses is equal to the size of the database in terms of records. The timing of individual crypto functions is obtained using the Python CProfile module. For keys of size 64, 128 and 256 we fit the whole encrypted dataset in memory. For keys of higher size, we only cache the sub-dataset based on the query indices.

The results in Fig. 7 follow the theoretical analysis and the simulation based performance shown in Fig. 4. For this set of experiments the gain ratio is 1.7 in average. It is also worth mentioning that approximate queries take less time than match/no-match queries in our approach.

C. *Storage vs. computation cost discussion*

In current cloud pricing plans, storage is much cheaper than computational power. For example, current Amazon prices for S3 storage starts with 0.03$ per GB / Month and decreases to 0.0275$ per GB / Month if more than 5000 TB are reserved[5].

Using Amazon Elastic Map Reduce to take benefit of parallelism in query processing, we pay an hourly rate for every instance hour of usage (so a 10-node cluster running for 10 hours costs the same as a 100-node cluster running for 1 hour). Hourly prices range from $0.011/hour to $0.27/hour.

[5] Checked on 9/28/2016

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

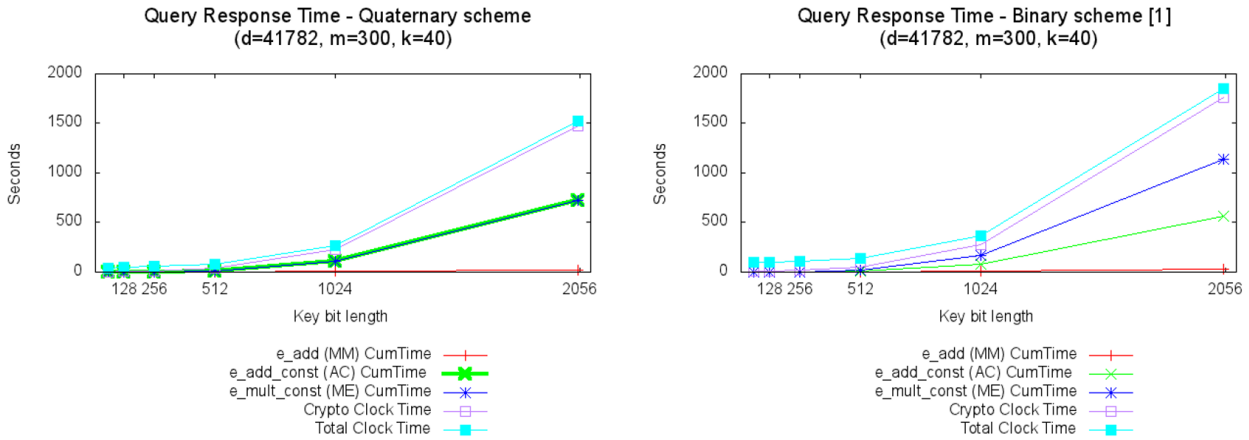

Fig. 7. Query response time on real DNA dataset

In our scenario, processing a query nearly requires half the time at the expense of doubling storage. For example, our relatively small dataset having 41,782 records of 300 letters each, encrypted using a 1024 bits key, requires about 31 GB of storage. It means that storage costs only about 1 dollar per month. On the other hand, the cost of renting an m3.xlarge instance is $0.266 per Hour. A query of 40 loci requires roughly 4.5 minutes in our setting but requires about 6.5 minutes in [1]. Therefore, for a batch of more than 25 queries, our approach has lower cost than [1]. The same analysis applies for larger databases since both the query cost and the storage cost vary linearly in the database size, under the same key size. The storage cost is paid once at setup time, and is amortized through subsequent queries. Moreover, from the user perspective, we are gaining a faster response. This scenario assumes that data is stored in S3 and transferred intermittently to computing node or cluster. The data transfer time in our scenario is quite tolerable and comparable to cluster allocation time. For our transfer size of 31 GB, we record a transfer time of around 29 seconds between our EC2 instance and S3; for either download or upload. Note that most cloud providers have specialized transfer services for big data (peta-byte scale).

For cost effectiveness, we suggest two deployment schemes:

1. Rent one or a cluster of Redis machines, and cache encrypted DNA sequences of importance (or just the segments of importance) prior to starting a batch of queries.

2. Rent a Spark/Hadoop map-reduce cluster and distribute data in a load balanced manner prior to executing batches of queries, therefore each node would process its part of the data.

In case a cache is used, additional network transfer overhead is required intermittently. The authors in [1] suggest estimating the result of a query for the whole database based on the result of the query for a random sample of the database, mainly to improve performance. We suggest using the same approach but for handling limited available cache and optimizing the cache replacement policy. An estimation of the query for the whole database can be determined within a given error margin, solely based on the cached sequences.

VI. Conclusion

In this paper, we have revisited the challenge of sharing person-specific genomic sequences without violating the privacy of their data subjects in order to support large-scale biomedical research projects. We have used the framework proposed by Kantarcioglu *et al.* [1] based on additive homomorphic encryption, and two servers: one holding the keys and one storing the encrypted records. The proposed method offers two new operating points in the space-time tradeoff and handles new types of queries that are not supported in earlier work. Furthermore, the method provides support for extended alphabet of nucleotides which is a practical and critical requirement for biomedical researchers.

Big data analytics over genetic data is a good future work direction. There are rapid recent advancements that address performance limitations of homomorphic encryption techniques. We hope that these advancements will lead to more practical solutions in the future that can handle larger-scale genetics data. It is worth mentioning that our approach is not restricted to a fixed homomorphic encryption technique and therefore, it would be possible to use and inherit the advantages of newly developed ones.

References

[1] M. Kantarcioglu, W. Jiang, Y. Liu, and B. Malin, “A cryptographic approach to securely share and query genomic sequences,” *Inf. Technol. Biomed. IEEE Trans.*, vol. 12, no. 5, pp. 606–617, 2008.

[2] B. Malin and L. Sweeney, “How (not) to protect genomic data privacy in a distributed network: using trail re-identification to evaluate and design anonymity protection systems,” *J. Biomed. Inform.*, vol. 37, no. 3, pp. 179–192, 2004.

[3] Z. Lin, A. B. Owen, and R. B. Altman, “Genomic research and human subject privacy,” *Science (80-.).*, vol. 305, no. 5681, p. 183, 2004.

[4] A. E. Nergiz, C. Clifton, and Q. M. Malluhi, “Updating outsourced anatomized private databases,” in *Proceedings of the 16th International Conference*

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

on Extending Database Technology, 2013, pp. 179–190.

[5] L. Sweeney, A. Abu, and J. Winn, "Identifying Participants in the Personal Genome Project by Name," *Available SSRN 2257732*, 2013.

[6] E. Aguiar, Y. Zhang, and M. Blanton, "An Overview of Issues and Recent Developments in Cloud Computing and Storage Security," in *High Performance Cloud Auditing and Applications*, 2014, pp. 3–33.

[7] P. Bohannon, M. Jakobsson, and S. Srikwan, "Cryptographic Approaches to Privacy in Forensic DNA Databases," in *Public Key Cryptography*, vol. 1751, H. Imai and Y. Zheng, Eds. Springer Berlin Heidelberg, 2000, pp. 373–390.

[8] F. Esponda, E. S. Ackley, P. Helman, H. Jia, and S. Forrest, "Protecting data privacy through hard-to-reverse negative databases," *Int. J. Inf. Secur.*, vol. 6, no. 6, pp. 403–415, 2007.

[9] F. Bruekers, S. Katzenbeisser, K. Kursawe, and P. Tuyls, "Privacy-preserving matching of dna profiles," *IACR Cryptol. ePrint Arch.*, vol. 2008, p. 203, 2008.

[10] M. J. Atallah and J. Li, "Secure outsourcing of sequence comparisons," *Int. J. Inf. Secur.*, vol. 4, no. 4, pp. 277–287, Mar. 2005.

[11] M. Blanton, M. M. J. Atallah, K. B. K. Frikken, and Q. Malluhi, "Secure and Efficient Outsourcing of Sequence Comparisons," *Comput. Secur. 2012*, pp. 505–522, 2012.

[12] M. Franklin, M. Gondree, and P. Mohassel, "Communication-efficient private protocols for longest common subsequence," in *Topics in Cryptology--CT-RSA 2009*, Springer, 2009, pp. 265–278.

[13] M. Gondree and P. Mohassel, "Longest common subsequence as private search," in *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, 2009, pp. 81–90.

[14] D. Szajda, M. Pohl, J. Owen, B. Lawson, and V. Richmond, "Toward a practical data privacy scheme for a distributed implementation of the Smith-Waterman genome sequence comparison algorithm," in *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS 06)*, 2006.

[15] M. Blanton and M. Aliasgari, "Secure outsourcing of DNA searching via finite automata," in *Data and Applications Security and Privacy XXIV*, Springer, 2010, pp. 49–64.

[16] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik, "Privacy preserving error resilient dna searching through oblivious automata," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 519–528.

[17] K. B. Frikken, "Practical private DNA string searching and matching through efficient oblivious automata evaluation," in *Data and Applications Security XXIII*, Springer, 2009, pp. 81–94.

[18] K. Kozl and C. Listy, "Biochemical nomenclature and related documents," *Chem. List.*, vol. 72, pp. 288–305, 1978.

[19] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of the 17th international conference on Theory and application of cryptographic techniques (EUROCRYPT'99)* , 1999, pp. 223–238.

[20] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*, 1983, pp. 199–203.

[21] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[22] E. Brickell, D. Gordon, K. McCurley, and D. Wilson, "Fast Exponentiation with Precomputation," in *Advances in Cryptology — EUROCRYPT' 92*, vol. 658, R. Rueppel, Ed. Springer Berlin Heidelberg, 1993, pp. 200–207.

[23] Benaloh, J. (1994, May). Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography* (pp. 120-128).

[24] ElGamal, T. (1984, August). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Workshop on the Theory and Application of Cryptographic Techniques* (pp. 10-18). Springer Berlin Heidelberg.

[25] Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120-126.

[26] Goldwasser, S., & Micali, S. (1984). Probabilistic encryption. *Journal of computer and system sciences*, *28*(2), 270-299.

[27] Fontaine, C., & Galand, F. (2007). A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, *2007*(1), 1-10.

Mohamad Nassar is currently a visiting assistant professor at American University of Beirut (AUB), Lebanon. He was a Post Doc fellow at the Kindi Research Lab in Qatar University (2011-2014). He received the research master degree (DEA) in computer science in 2005 and the PhD degree in 2009, both from Nancy University, France. He worked as a research engineer in INRIA Nancy and Ericsson, Ireland (2009-2011). His PhD research focuses on monitoring and intrusion detection in VoIP networks. His current research interests are practical machine learning, data security and privacy.

Qutaibah M. Malluhi joined Qatar University in September 2005. He is the Director of the KINDI Center for Computing Research. He served as the head of Computer Science and Engineering Department at Qatar University between

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2682860, IEEE Transactions on Cloud Computing

2005-2012. Before joining Qatar University he was a professor of Computer Science at Jackson State University where he served as a faculty member between 1994 and 2005. During 1995 and 1996, he was a research faculty at Lawrence Berkeley National Laboratory, Berkeley California. Dr. Malluhi was the co-founder and CTO of Data Reliability Inc. between 2001 and 2005. He was also the Co-Founder and Executive Advisor for the Qatar Mobility Innovation Center at the Qatar Science and Technology Park. Prof. Malluhi was a consultant for several telecommunication companies where he built networks, designed internet/intranet systems, developed distributed applications and telecommunication management software.

Mikhail J. Atallah 's current research interests are primarily in information security, and also include algorithms, parallel computation, and computational geometry. His work in information security centers on protocols for online collaborations between entities that do not completely trust each other, on key management issues in access control, and on watermarking digital objects (particularly non-media, such as relational data and natural language text). As a Fellow of both the ACM and IEEE, he has served on the editorial boards of top journals, and on the program committees of top conferences and workshops. He was keynote and invited speaker at many national and international meetings, and a speaker nine times in the Distinguished Colloquium Series of top Computer Science Departments. He was selected in 1999 as one of the best teachers in the history of Purdue University and included in Purdue's Book of Great Teachers, a permanent wall display of Purdue's best teachers past and present. He is a co-founder of Arxan Technologies Inc.

Abdullatif Shikfa is a research assistant professor at KINDI center for computing research of Qatar University. Before joining Qatar University, Abdullatif held both research and industry positions: he served as research engineer at EURECOM, as scientific advisor and deputy head of the security research department at Bell Labs, Alcatel-Lucent and then he worked as technical project manager and security expert at Thales. His research interests and experience span a wide range of topics in information and communication security from trust and cooperation enforcement to secure routing, through protection of users' privacy mainly by applying advanced cryptographic primitives to enable computation on encrypted data. Dr. Shikfa obtained his PhD from Telecom ParisTech in 2010 and he is an alumnus of Ecole Polytechnique (MSc. 2004), of University of Nice Sophia Antipolis (MSc. 2005), and of ENST-EURECOM (MSc. 2006).

2168-7161 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.