Sotiraki *et al. BMC Medical Genomics* 2020, **13**(Suppl 7):72
https://doi.org/10.1186/s12920-020-0718-x

BMC Medical Genomics

RESEARCH Open Access

# Privately computing set-maximal matches in genomic data

Katerina Sotiraki[1*], Esha Ghosh[2] and Hao Chen[2]

*From* 7th iDASH Privacy and Security Workshop 2018
San Diego, CA, USA. 15 October 2018

## Abstract

**Background:** Finding long matches in deoxyribonucleic acid (DNA) sequences in large aligned genetic sequences is a problem of great interest. A paradigmatic application is the identification of distant relatives via large common subsequences in DNA data. However, because of the sensitive nature of genomic data such computations without security consideration might compromise the privacy of the individuals involved.

**Methods:** The secret sharing technique enables the computation of matches while respecting the privacy of the inputs of the parties involved. This method requires interaction that depends on the circuit depth needed for the computation.

**Results:** We design a new depth-optimized algorithm for computing set-maximal matches between a database of aligned genetic sequences and the DNA of an individual while respecting the privacy of both the database owner and the individual. We then implement and evaluate our protocol.

**Conclusions:** Using modern cryptographic techniques, difficult genomic computations are performed in a privacy-preserving way. We enrich this research area by proposing a privacy-preserving protocol for set-maximal matches.

**Keywords:** Set-maximal match, Privacy, Secret sharing

## Background

The abundance of human genomic data in recent years paves the path towards answering very important questions for human nature, such as identifying the genomes responsible for particular illnesses. Simultaneously, the extremely sensitive nature of this data imposes strict restrictions on its use. Fortunately, there is a variety of cryptographic techniques that allow us to create useful yet privacy-preserving systems for computation in genomic data (see [1] for a summary of various techniques). Even though theoretically it is possible to perform every computation in a private way, the generic techniques do not necessarily preserve the efficiency and the accuracy of the original algorithm. Thus, constructing practical privacy-preserving protocols has become a very active area of research.

Quantifying the similarity of genomic sequences is a fundamental problem in genome informatics and there exists numerous proposals for tackling this problem. In this work, we focus on privately computing *set-maximal matches* on genomic data as a way to identify similarity. Initially, research on genomic matching focused on finding efficient and accurate algorithms (e.g. [2–9]). More recently, the issue of privacy has emerged and hence new approaches have been proposed. Freedman et. al. [10] consider the problem of secure keyword-search in a database by relying on a connection to oblivious eval-

*Correspondence: katesot@mit.edu
[1]MIT, CSAIL, 32 Vassar street, 02139 Cambridge, MA, USA
Full list of author information is available at the end of the article

uation of pseudorandom functions. Various other works focus on problems more specific to genomic data. For instance, Jha et al. [11] develop a protocol for securely computing edit distance of DNA sequences, Blanton et al. [12] propose a protocol for outsourcing DNA search via finite automata to multiple computational servers. Baldi et al. [13] focus on similar applications, such as Paternity Testing, and use techniques on private set operations. He et al. [14] construct a protocol that identifies whether two individuals are relatives without revealing any other information about their genomes. More closely related to our work, Shimizu et al. [15] propose a protocol for privately computing set-maximal matches between a database and an individual starting from a genomic position known only to the individual. Even though we also focus on computing set-maximal matches, our problem is more general since our schemes outputs all set-maximal matches without leaking their locations for the client.

The efforts to construct secure but yet efficient and practical protocols for genome analysis have also been reinforced by the establishment of the Integrating Data for Analysis, Anonymization and SHaring (iDASH) privacy and security workshop [16]. Each year, the workshop poses a set of tasks to evaluate the employment of cryptographic techniques for real-world challenges in genome analysis.

In this work, we propose a novel protocol for measuring similarity between a database of DNA sequences and a query DNA sequence privately by identifying the set-maximal matches between the database and the query. This problem was also the competition task on the secure multiparty computation track in the iDASH workshop for 2018 [16].

## Methods

Our goal is to design a protocol for computing the set-maximal matches between a database and a query. We assume that all variable sites are bi-allelic; namely each site has a value in $\{0, 1\}$. Before reviewing the main cryptographic tools used in this work, we explain our notation and give the formal definition of set-maximal matches.

**Notation** If $\mathbf{M}$ is a matrix, then $\mathbf{M}_{j,i}$ denotes the element of the $j$-th row and $i$-th column. We denote by a bold lowercase letter (e.g. $\mathbf{x}, \mathbf{y}$) a sequence of bi-allelic sites; the $i$-th site of $\mathbf{x}$ is denoted by $\mathbf{x}[i]$. Namely, $\mathbf{x} = (\mathbf{x}[1], \mathbf{x}[2], \ldots, \mathbf{x}[n])$. For simplicity, we use the notation $\mathbf{x}[i_1, i_2]$ to denote the substring $(\mathbf{x}[i_1], \ldots, \mathbf{x}[i_2])$. A bold uppercase letter (e.g. $yY$) represents a database of genomic sequences; the $j$-th tuple of the database $\mathbf{Y}$ (i.e. the genomic data of the $j$-th individual in the database) is denoted using a bold lowercase letter as $\mathbf{y}_j$. We call $n$ the number of sites in our genomic sequences and $m$ the number of sequences in the database.

We use the Big $O$ notation for describing the limiting behavior of the running time of an algorithm. We say that $f(x) = O(g(x))$ if and only if there exists a constant $c$ such that for large enough $x, f(x) \leq cg(x)$.

**Definition** Let $\mathbf{Y}$ be a database and $\mathbf{x}$ be a query sequence, then a substring $\mathbf{x}[i_1, i_2]$ with $i_1 \leq i_2$ is a set-maximal match between $\mathbf{x}$ and $\mathbf{Y}$ if there exists a $j$ such that:

1 $\mathbf{x}[i_1, i_2] = \mathbf{y}_j[i_1, i_2]$
2 $\mathbf{x}[i_1 - 1] \neq \mathbf{y}_j[i_1 - 1]$ and $\mathbf{x}[i_2 + 1] \neq \mathbf{y}_j[i_2 + 1]$

and for any $j' \neq j$, there exists no interval $[i'_1, i'_2]$ such that $\mathbf{x}[i'_1, i'_2] = \mathbf{y}_j[i'_1, i'_2]$ and $[i_1, i_2]$ is a strict subset of $[i'_1, i'_2]$.

The first two properties assure that the match is a *locally maximal match*, namely that the match cannot be extended in either side and the last property is satisfied when the match is not strictly contained in another match with a different database entry. Sometimes, it is useful to keep only long enough matches; in this case, the definition has an extra parameter called *threshold*.

**Definition** A match is a set-maximal match between $\mathbf{x}$ and $\mathbf{Y}$ with threshold $t$ if it is a set-maximal match between $\mathbf{x}$ and $\mathbf{Y}$ and additionally its length is more than or equal to $t$.

### Cryptographic tools

Secure computation allows multiple parties to compute a joint function while preserving the privacy of their individual inputs. There are feasibility results [17, 18] that allow us to compile any computation into a secure one. Unfortunately, these methods do not preserve the efficiency of the original algorithm. Hence, it is often essential to design novel algorithms that take into account the special nature of secure computation protocols.

**Security model.** Let $P_0$ and $P_1$ be two parties holding inputs $x_1$ and $x_2$ respectively. They are interested in jointly computing a function $f(x_1, x_2)$. A protocol between the two parties A and B, is called secure (or privacy-preserving) if it leaks nothing about the inputs $x_1$ (to $P_1$) and $x_2$ (to $P_0$) apart from the output value $f(x_1, x_2)$ (and the length of the inputs). There are two well-studied adversarial models, the semi-honest and the malicious. A semi-honest adversary observes all communication between the computing parties (and tries to learn information about the inputs), but is not allowed to deviate from the protocol. On the contrary, a malicious adversary is allowed to arbitrarily deviate from the protocol in order to try to learn extra information about the inputs. In both case, the adversary is assumed to be computationally bounded. In this work, the two involved parties $P_0$ and $P_1$

are the client and the server and our protocol is secure in the semi-honest model.

In this work, we focus on the Goldreich-Micali-Wigderson (GMW) method [17] for secure computation with *boolean secret shares*. This method gives a way to privately compute XOR ($\oplus$), AND ($\wedge$) and NOT ($\neg$) gates in the semi-honest model. Since all computations can be expressed by a circuit containing only these three types of gates, boolean sharing allows to perform every computation in a private fashion. We briefly describe how operations are performed in boolean sharing and some of the available optimizations and implementations.

Intuitively, secret sharing of a value $x$ is a split of $x$ in many parts such that each part does not reveal any information about $x$, but the knowledge of all the parts allows the recovery of $x$. The GMW framework suggests a specific way to share values such that it is possible to perform any operation on them. Namely, let us assume that there are two parties and each party knows a share of $x$, then using GMW they can end up with shares of any function of $x$.

**Sharing of a bit *b*.** The boolean shares of a bit $b$ are two bits $\langle b\rangle_0$ and $\langle b\rangle_1$ such that $\langle b\rangle_0 \oplus \langle b\rangle_1 = b$.

**Reconstruction of a bit *b*.** If parties $P_0$ and $P_1$ have the shares $\langle b\rangle_0$ and $\langle b\rangle_1$ respectively, then they reconstruct the bit $b$ by exchanging shares and computing $\langle b\rangle_0 \oplus \langle b\rangle_1$.

**Computing XOR privately.** Assume that party $P_i$ knows the secret shares $\langle x\rangle_i$ and $\langle y\rangle_i$ of the bits $x$ and $y$ respectively. Then, party $P_i$ computes a share of $x \oplus y$ by locally computing $\langle x\rangle_i \oplus \langle y\rangle_i$.

**Computing AND privately.** Assume that party $P_i$ knows the secret shares $\langle x\rangle_i$ and $\langle y\rangle_i$ of the bits $x$ and $y$ respectively. The shares of $x \wedge y$ are evaluated using a *precomputed multiplication triple* ($\langle a\rangle_i, \langle b\rangle_i, \langle c\rangle_i$) of the bits $a, b, c$ such that $a \wedge b = c$. Initially, $P_i$ computes the shares $\langle e\rangle_i = \langle a\rangle_i \oplus \langle x\rangle_i$ and $\langle f\rangle_i = \langle b\rangle_i \oplus \langle y\rangle_i$. Then, both parties $P_0$ and $P_1$ reconstruct the values $e$ and $f$. Finally, $P_i$'s new share is equal to $(i \wedge e \wedge f) \oplus (f \wedge \langle a\rangle_i) \oplus (e \wedge \langle b\rangle_i) \oplus \langle c\rangle_i$.

**Computing NOT privately.** Assume that party $P_i$ knows the secret share $\langle x\rangle_i$ of the bits $x$. Then, party $P_i$ computes a share of $\neg x$ by locally computing $\langle x\rangle_i \oplus i$.

We remark that the computation of the multiplication triples does not depend on the actual computation or input, so it can be done in advance during a precomputation phase which requires interaction between the parties. The multiplication triples can be computed using the cryptographic primitive of *random oblivious transfer* [19]. Additionally, we note that computing XOR and NOT gates is done locally and does not require any interaction. On the other hand, an AND operation requires one flow of interaction in order to reconstruct the values $e$ and $f$. Therefore, since we can compute many AND operations in parallel, the number of rounds required for computing a function $f$ is proportional to the AND-depth of its circuit representation.

Since the introduction of GMW, various optimizations have been proposed [19–21]. Our implementation is based on the ABY framework [22], which provides semi-honest security. Apart from GMW on boolean shares, this framework provides implementations of two other well-studied methods for secure computation, secure computation using arithmetic shares and Yao's Garbled Circuits. The ABY framework is suitable for mixed-protocols, since it allows for very efficient conversion between the different secure computation methods. Even though our solution is not a mixed-protocol, we use ABY since it includes all known optimizations of GMW and it allows the composition of our protocol with others that are potentially more efficient if implemented using another method of secure computation.

**Two-party secure protocols using the GMW framework.** If party $P_0$ has input $x_0$ and party $P_1$ has input $x_1$, then they can privately compute a function $f$, which is given in the form of a circuit containing XOR, AND and NOT gates, as follows:

1. Party $P_i$ secret shares $x_i$ by sending a uniformly random binary string $r_i$ with length equal to its input to $P_{1-i}$ and setting its shares equal to $x_i \oplus r_i$, where $\oplus$ denotes the bitwise XOR operation.
2. The two parties compute the function $f$ gate-by-gate as described above and end up with boolean shares of the output.
3. The two parties exchange shares and reconstruct the output of the function by computing the XOR of their shares with the shares received by the other party.

By slightly modifying the above protocol, it is possible to achieve *selective reconstruction* of the output, in which case only one of the parties learns the output. For instance, if only the client should learn the output, then at the third step the server sends its shares and the client sends nothing. In this case, the client has enough information to recover the output, whereas the server does not learn the output.

## Results

The GMW framework allows us to transform any computation in a form of a circuit into a privacy-preserving one against semi-honest adversaries. Therefore, we focus on designing a depth-optimized circuit with XOR, AND and NOT gates to compute set-maximal matches. Then, using

the generic protocol described in the previous section, we have a secure protocol for set-maximal matches that requires at most as many rounds of interaction as the depth of the circuit.

### Problem definition

We give an efficient and depth-optimized algorithm for computing set-maximal matches. More specifically, the problem specification is as follows:

**Input:** A genomic database $\mathbf{Y}$ containing $m$ sequences, each of size $n$, a query genomic sequence $\mathbf{x}$ of size $n$, and a threshold value $t$.
**Output:** A matrix $\mathbf{M}$ of size $m \times n$ such that the element $\mathbf{M}_{j,i}$ is equal to the length of the match between $\mathbf{x}$ and $\mathbf{y}_j$ ending at position $i$ if the match is set-maximal with threshold $t$ and 0 otherwise.

We note that the output as described above leaks the position of set-maximal matches. Because of the very sensitive nature of genomic sequences, it is beneficial to hide even this information. Therefore, we slightly modify the above problem so that the list of set-maximal matches is revealed after applying a random permutation to each row of the output.

**Input:** A genomic database $\mathbf{Y}$ containing $m$ sequences, each of size $n$, a query genomic sequence $\mathbf{x}$ of size $n$, a threshold value $t$ and $m$ permutations $(\pi_k)_{k\in\{1,\ldots,m\}}$.
**Output:** Let $\mathbf{M}$ be a matrix of size $m \times n$ such that the element $\mathbf{M}_{j,i}$ is equal to the length of the match between $\mathbf{x}$ and $\mathbf{y}_j$ ending at position $i$ if the match is set-maximal with threshold $t$ and 0 otherwise. The output is the permutation of each row $k$ of $\mathbf{M}$ according to $\pi_k$.

We observe that the output still reveals the lengths of set-maximal matches for each index. This information could be hidden by applying a random permutation on all the entries of $\mathbf{M}$, instead of each row. However, it seems that this would reduce the applicability of the computation, since this information seems central for certain application.

In the secure protocol, the database $\mathbf{Y}$ and the permutations $(\pi_k)_{k\in\{1,\ldots,m\}}$ are the input of the server, the query $\mathbf{x}$ is the input of the client and the threshold $t$ is known to both parties. To avoid leakage to the server, the protocol has selective reconstruction to the client.

### Algorithm description

We describe our algorithm. We include a sample execution in Fig. 1.

1. We compute a matrix $\mathbf{M}$ of size $m \times n$ such that $\mathbf{M}_{j,i}$ is equal to the length of the match between the query $\mathbf{x}$ and the database entry $\mathbf{y}_j$ ending at position $i$ (Fig. 1b).
2. We set $\mathbf{M}_{j,i}$ to 0 if the match of $\mathbf{x}$ and $\mathbf{y}_j$ ending at position $i$ is below the threshold $t$ (Fig. 1c).
3. We compute a matrix $\mathbf{L}$ of size $m \times n$ such that $\mathbf{L}_{j,i}$ is 0 if there is a $j' \neq j$ such that $\mathbf{M}_{j',i} > \mathbf{M}_{j,i}$ and 1 otherwise (Fig. 1d).

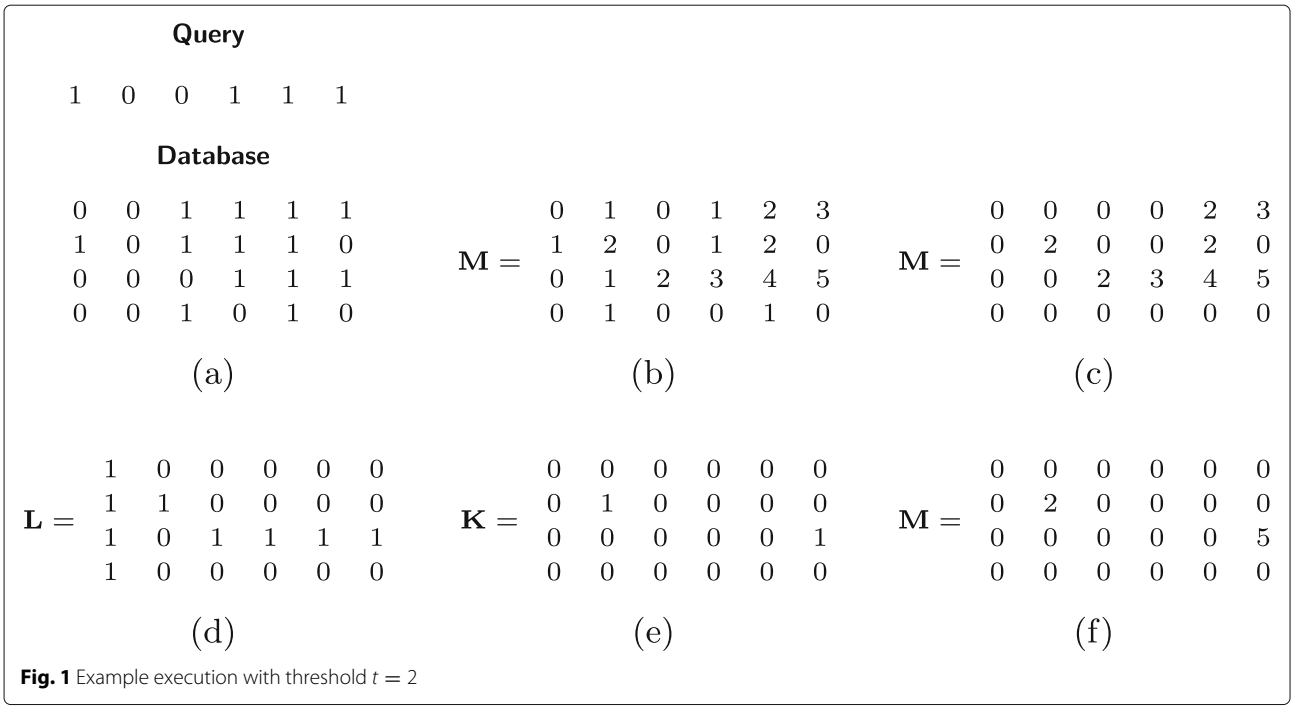


**Fig. 1** Example execution with threshold $t = 2$

4 We compute $\mathbf{K}$ such that $\mathbf{K}_{j,i}$ is 0 if there is a $j'$ (may be equal $j$) such that $\mathbf{L}_{j',i} = yL_{j',i+1} = 1$. Namely, there exists a match that is extended to position $i+1$ (Fig. 1e).
5 We set $\mathbf{M}_{j,i} \leftarrow \mathbf{M}_{j,i}\mathbf{K}_{j,i}\mathbf{L}_{j,i}$ and we permute the row $\mathbf{M}_k$ according to permutation $\pi_k$ (Fig. 1f).

After the steps described above, the matrix $\mathbf{M}$ contains the correct output:

- The output $\mathbf{M}$ contains the correct length of matches computed in step 1.
- The output $\mathbf{M}$ contains no matches of length less than $t$, since all such matching have been removed in step 2.
- The output $\mathbf{M}$ does not output matches strictly contained in another match. If a match is contained in another larger match, then either there is a match with a preceding starting point or a match with a succeeding ending point or both. In the first and third case, this match is excluded in step 3, since there is another database entry with larger value in the corresponding positions of $\mathbf{M}$. In the second case, the match is excluded in step 4, since there is another extendable match in the corresponding positions.
- The output $\mathbf{M}$ contains only locally maximal matches. After step 1, $\mathbf{M}$ contains the length of matches from their starting position, so it is not possible for a match to be extendable toward a previous position. Additionally, from step 4, $\mathbf{M}$ does not contain matches extendable towards the next position.

We now describe how to implement each of these steps using boolean circuits with AND, XOR and NOT gates in a depth-optimized way.

**Compute the length of matches:** First, we compute an auxiliary matrix $\mathbf{B}^{(0)}$ such that $\mathbf{B}^{(0)}_{j,i} = 1$ if $\mathbf{x}[i] = \mathbf{y}_j[i]$. Namely, $\mathbf{B}^{(0)}_{j,i} = \neg(\mathbf{x}[i] \oplus \mathbf{y}_j[i])$. We observe that $\mathbf{B}^{(0)}$ indicates whether a match has length greater than or equal to one. Using $\mathbf{B}^{(0)}$, we compute whether a match has length greater than or equal to two by setting $\mathbf{B}^{(1)}_{j,i} \leftarrow \mathbf{B}^{(0)}_{j,i} \wedge \mathbf{B}^{(0)}_{j,i-1}$. More generally, if $\mathbf{B}^{(k)}$ indicates whether a match has length more than $2^k$ or not, then it can be updated to indicate if the length of a match is more than $2^{k+1}$ by setting $\mathbf{B}^{(k+1)}_{j,i} \leftarrow \mathbf{B}^{(k)}_{j,i} \wedge \mathbf{B}^{(k)}_{j,i-(2^k-1)}$.

Concurrently, in each iteration we compute a bound on the length of each match by setting $\mathbf{M}^{(0)} = \mathbf{B}^{(0)}$ and $\mathbf{M}^{(k+1)}_{j,i} \leftarrow \mathbf{B}^{(k)}_{j,i-(2^k-1)}\mathbf{M}^{(k)}_{j,i-(2^k-1)} + \mathbf{M}^{(k)}_{j,i}$. We observe that this computation gives the actual length of a match if it is less than or equal to $2^k$ and returns the lower bound of $2^k$ otherwise. Hence, after $\lceil \log(n) \rceil$ iterations, we set $\mathbf{M} = \mathbf{M}^{(\lceil \log(n) \rceil)}$.

The addition is computed using a depth-optimized adder, which has AND-depth proportional to the logarithm of the bit length of the numbers returned [23]. Namely, the AND-depth of the adder is $O(\log(\log n))$. Overall, the AND-depth of the length computation is $O(\log(n)\log(\log n))$.

**Remove matches with length below the threshold:** Let $t_k \equiv t \pmod{2^k}$ for $k \in \{1, \ldots, \lceil \log(n) \rceil\}$ and $(b_{\lceil \log(n) \rceil}, \ldots, b_1)$ be the bit decomposition of $t$, where $b_{\lceil \log(n) \rceil}$ is the most significant bit and $b_1$ is the least significant bit. Let $\mathbf{T}^{(k)}$ be $m \times n$ matrices that indicate candidate matches; initially $\mathbf{T}^{(0)}_{j,i} = 1$ for all $i$ and $j$. Similarly to the length computation, we define the auxiliary matrix $\mathbf{B}^{(0)}$ that initially indicates whether a database and query position match or not. In each iteration, we update $\mathbf{B}^{(k)}$ as in the length computation; namely, we set $\mathbf{B}^{(k)}_{j,i} \leftarrow \mathbf{B}^{(k-1)}_{j,i} \wedge \mathbf{B}^{(k-1)}_{j,i-(2^k-1)}$. In the $k$-th iteration if $b_k = 1$, we update $\mathbf{T}^{(k)}_{j,i} \leftarrow \mathbf{B}^{(k)}_{j,i-(t_k-1)}\mathbf{T}_{j,i}$, otherwise $\mathbf{T}^{(k)} = \mathbf{T}^{(k-1)}$. Finally, we set $\mathbf{M}_{j,i} \leftarrow \mathbf{M}_{j,i}\mathbf{T}^{(\lceil \log(n) \rceil)}_{j,i}$.

This computation intuitively splits the $t$ positions preceding a specific position $i$ into parts of increasing powers of two, then it iteratively checks whether each of these parts is a match. Splitting a number into increasing powers of two is equivalent to computing its bit decomposition. Even though the AND-depth of this step is $O(\log(n))$, it can be performed in parallel to the length computation where we also use the same auxiliary matrix $\mathbf{B}$. So, this step does not increase the depth of the circuit.

**Remove matches contained in other matches:** We first compute the maximum length of a match for each position $i$ across all the database entries and then perform an equality check between $\mathbf{M}_{j,i}$ and the maximum for position $i$ to compute each $\mathbf{L}_{j,i}$. Each such maximum is computed using the D&C comparison circuit [24] in AND-depth $O(\log(\log n)\log(m))$. The equality check can be implemented with a depth-optimized circuit in $O(\log(\log n)\log m)$ depth in which the comparison of the bits across the database entries is performed in parallel.

**Remove extendable matches:** A match between $\mathbf{x}$ and $\mathbf{y}_j$ is extendable at position $i$ if $\mathbf{B}_{j,i+1} = 1$. So, in order to remove the extendable matches, we compute $\mathbf{K}_{j,i} = \neg \max_{j'} \{\mathbf{L}_{j',i} \wedge \mathbf{L}_{j',i+1}\}$ that indicates whether an extendable match exists. Finally, we update $\mathbf{M}_{j,i} \leftarrow \mathbf{K}_{j,i} \wedge \mathbf{L}_{j,i} \wedge \mathbf{M}_{j,i}$. Using the D&C comparison, this operation requires $O(\log(\log n)\log m)$ AND-depth.

**Permute matches:** Finally, to remove the information regarding the position of matches, we permute each row $k$ of the matrix $\mathbf{M}$ according to a permutation $\pi_k$, which is given as an input. All permutations are performed in parallel and require AND-depth $O(\log(n))$ using the Waksman permutation network [25].

The total AND-depth of the above circuit is $O(\log(n) + \log(\log n)\log(m))$. Since in practice $n >> m$, the depth can be assumed to be proportional to $O(\log(n))$.

We now present an optimization that reduces the output size and the computational cost of the permutations. Even though this does not offer an asymptotic optimization, it is definitely helpful in the experimental efficiency of the protocol.

**Output size reduction and efficient permutations:** Before the permutation step, the above circuit outputs a matrix $\mathbf{M}$ that contains all set-maximal matches with threshold $t$. We note that the threshold $t$ guarantees that there exists no set-maximal matches ending at positions with distance less than $t$. In other words, in each row of the matrix $\mathbf{M}$ there is at most one none zero element for every $t$ positions. By combining every $t$ columns of $\mathbf{M}$ into one equal to their XOR, we reduce the size of the output by a factor of $t$ without losing the desired information about set-maximal matches.

After the output reduction, we need to permute each row of a matrix of size $m \times n/t$. Therefore, the Waksman permutation network has depth only $O(\log(n/t))$.

#### Experimental evaluation

We have implemented our protocol for secure computation of set-maximal matches using the ABY framework to evaluate its efficiency. We use the network configuration included in ABY for the communication between the two parties and the default method for precomputing multiplication triplets via oblivious transfer. Then, we build the computation circuit gate-by-gate and let ABY handle the sharing procedures and secure computation. The problem has three parameters: $n$, the size of the genomic sequences, $m$, the size of the database and $t$, the threshold of set-maximal matches. We run a series of simulations in a single machine of 16GB RAM and Quad-core 2.8 GHz CPU that simulates both the server and the client to evaluate the efficiency of the protocol with respect to these parameters.

In our implementation, the server has as input the database, which defines the parameters $n$ and $m$, and the threshold value $t$. Similarly, the client's input is a query of size $n$, the database size $m$ and the threshold value $t$. We note that because of the nature of the GMW protocol both parties need to know the values of all three parameters $n$, $m$ and $t$. We evaluate how our protocol scales as $n$ increases for database size $m = 10, 100, 1000$ and threshold $t = 1$ and $t = 2^k$, where $k$ is the bit length of $n$. Even though the threshold value is an input to the protocol, we plot only two values for clarity of exposition. The two values represent the best and worst values in terms of efficiency. The threshold value $2^k$ maximizes the efficiency gain form the output reduction optimization. On the contrary, when the threshold value $t = 1$, this optimization is not in use, and hence this value corresponds to the worst case running time. In Fig. 2, we observe that for larger enough values of $n$ indeed there is an improvement of both the running time and the depth due to the output reduction technique.

Figure 2a, c, and e show the running time of the protocol for three different values of $m$. For small $n$ and $m$, the running time essentially depends on the precomputation phase, but when $m$ and $n$ are large enough the running time for a given $m$ depends almost linearly in $n$. We observe that for small values of $n$, there is almost no difference on the running time for the two threshold values. In this case, the output is reduced by a small factor when $t = 2^k$, whereas the precomputation time increases, since the computation circuit needs to include the output reduction. Hence, for small values of $n$, there is no improvement in the efficiency from the output reduction optimization.

In Fig. 2b, d, and f, we notice that the depth depends mainly on $\log(n)$, which is what was expected by the analysis in the previous section.

## Discussion

Because of the developments on efficiently acquiring DNA data, computing on genomic data has gained a lot of attention in the recent years. At the same time, progress on cryptographic techniques has allowed us to design numerous protocols for private computation that work well in practice. The connection of these two areas of research has led to fascinating directions and applications.

We make progress in an important problem lying in the intersection of these areas concerning the similarity of genomic data. Our motivating application is that of identifying relatives without compromising the privacy of the genomic data of the individuals involved.

## Conclusions

We construct an efficient algorithm for computing set-maximal matches, which is compatible with secure computation approaches. More specifically, our algorithm is designed carefully so that it remains efficient when compiled in the GMW framework, which offers a generic way to perform secure computation in the semi-honest model of security.

We implement and evaluate our algorithm using the ABY framework. Our algorithm runs for relatively large

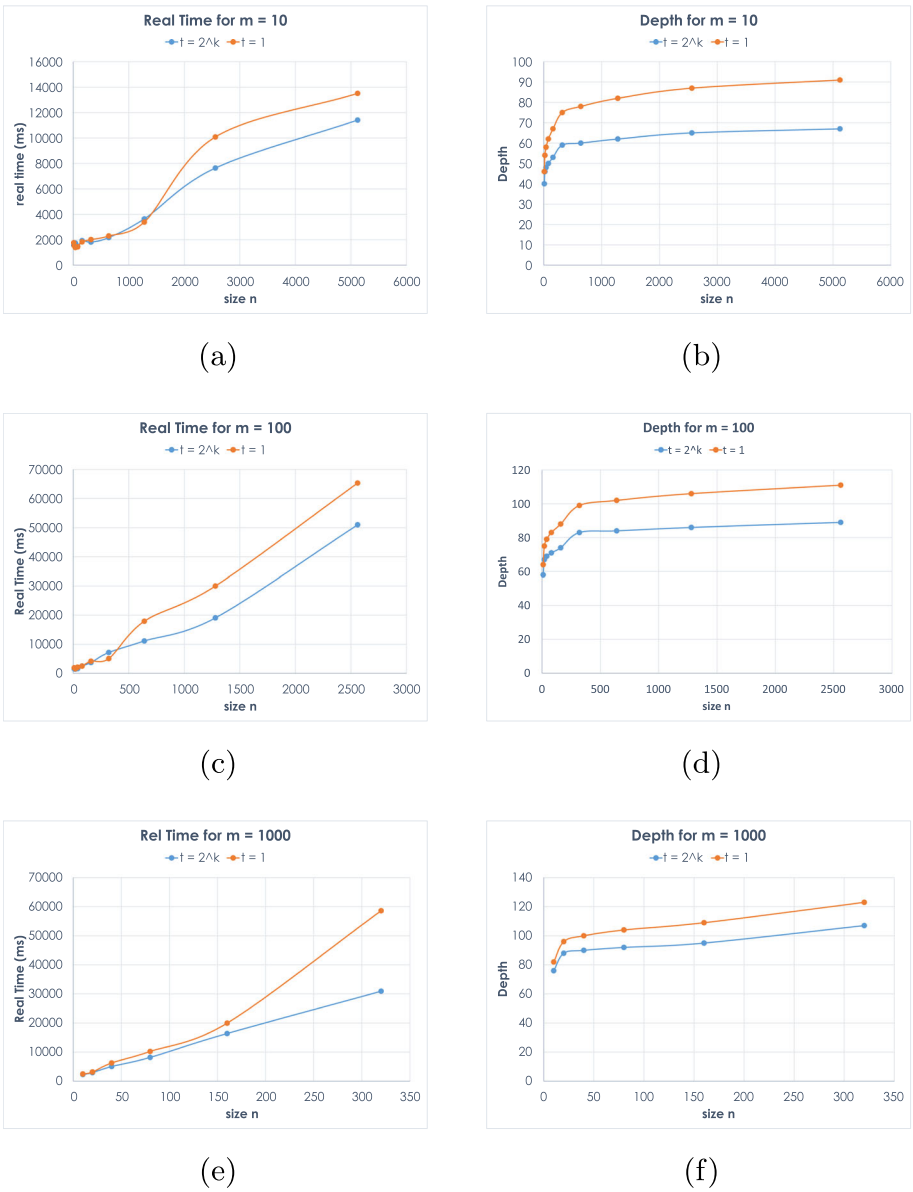


**Fig. 2** Timings and circuit depths for various values of $n$, $m$ and $t$

datasets and the behavior of the running time and the rounds of interaction is compatible with our theoretical analysis. The ABY framework is a very general framework that offers many capabilities. Unfortunately, this generality hurts the efficiency of our protocol, so it would be beneficial for the practical efficiency of our scheme to implement it using a tailored secure computation protocol, which is more lightweight and contains only the parts necessary for our protocol.

This work extends an exciting line of research that combines cryptographic techniques for secure computation with efficient and accurate algorithms for genomic analysis. Our main contribution is on securely finding set-maximal matches between a database and a query sequence.

**Acknowledgements**
The authors would like to thank the iDASH workshop organizers and the reviewers for the helpful comments.

**About this supplement**
This article has been published as part of *BMC Medical Genomics Volume 13 Supplement 7, 2020: Proceedings of the 7th iDASH Privacy and Security Workshop 2018*. The full contents of the supplement are available online at https://bmcmedgenomics.biomedcentral.com/articles/supplements/volume-13-supplement-7.

**Authors' contributions**
KS was a major contributor in designing the algorithm, implementing the protocol and writing the manuscript. EG contributed in the protocol implementation and in examining the correctness of the algorithm. HC suggested using the Waksman permutation networks in the last step of the protocol and the optimization for the output size reduction All author(s) read and approved the final manuscript.

**Funding**
Publication costs were funded by Microsoft Research.

**Availability of data and materials**
The code used during the current study is available from the corresponding author on reasonable request.

**Ethics approval and consent to participate**
Not applicable.

**Consent for publication**
Not applicable.

**Competing interests**
This work was conducted when KS was an intern at Microsoft Research and EG and HC were employed by Microsoft Research.

**Author details**
[1]MIT, CSAIL, 32 Vassar street, 02139 Cambridge, MA, USA. [2]Microsoft Research, 14820 NE 36th Street, Building 99, 98052 Redmond, WA, USA.

Published: 21 July 2020

**References**

1. Aziz MMA, Sadat MN, Alhadidi D, Wang S, Jiang X, Brown CL, Mohammed N. Privacy-preserving techniques of genomic data–a survey. Brief Bioinform. 2017;20(3):1–9.
2. Lipman D, Pearson W. Rapid and sensitive protein similarity searches. Science. 1985;227(4693):1435–41.
3. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990;215(3):403–10.
4. James Kent W. Blat - the blast-like alignment tool. Genome Res. 2002;12: 656–64.
5. Ma B, Tromp J, Li M. Patternhunter: faster and more sensitive homology search. Bioinformatics. 2002;18(3):440–5.
6. Li H, Durbin R. Fast and accurate short read alignment with burrows–wheeler transform. Bioinformatics. 2009;25(14):1754–60.
7. Langmead B, Trapnell C, Pop M, Salzberg SL. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. Genome Biol. 2009;10(3):25.
8. Li H, Homer N. A survey of sequence alignment algorithms for next-generation sequencing. Brief Bioinform. 2010;11(5):473–83.
9. Durbin R. Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). Bioinformatics. 2014;30(9):1266–72.
10. Freedman MJ, Ishai Y, Pinkas B, Reingold O. Keyword search and oblivious pseudorandom functions. In: Proceedings Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, February 10-12, 2005,. Cambridge: Springer Berlin Heidelberg; 2005. p. 303–24.
11. Jha S, Kruger L, Shmatikov V. Towards practical privacy for genomic computation. In: 2008 IEEE Symposium on Security and Privacy (sp 2008). IEEE; 2008. p. 216–30.
12. Blanton M, Aliasgari M. Secure outsourcing of dna searching via finite automata. In: In Conference on Data and Applications Security (DBSec). Berlin: Springer; 2010. p. 49–64.
13. Baldi P, Baronio R, De Cristofaro E, Gasti P, Tsudik G. Countering gattaca: Efficient and secure testing of fully-sequenced human genomes. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. CCS '11. New York: ACM; 2011. p. 691–702.
14. He D, Furlotte NA, Hormozdiari F, Joo JWJ, Wadia A, Ostrovsky R, Sahai A, Eskin E. Identifying genetic relatives without compromising privacy. Genome Res. 2014;24(4):664–72.
15. Shimizu K, Nuida K, Rätsch G. Efficient privacy-preserving string search and an application in genomics. Bioinformatics. 2016;32:1652–61.
16. iDASH. 2018. http://www.humangenomeprivacy.org/2018/. Accessed 17 June 2019.
17. Goldreich O, Micali S, Wigderson A. How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. STOC '87. New York: ACM; 1987. p. 218–29.
18. Yao AC. Protocols for secure computations. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science. SFCS '82. Washington: IEEE Computer Society; 1982. p. 160–4.
19. Asharov G, Lindell Y, Schneider T, Zohner M. More efficient oblivious transfer extensions. J Cryptol. 2017;30(3):805–58.
20. Ishai Y, Kilian J, Nissim K, Petrank E. Extending oblivious transfers efficiently. In: Boneh D, editor. Advances in Cryptology - CRYPTO 2003. Berlin, Heidelberg: Springer; 2003. p. 145–61.
21. Schneider T, Zohner M. Gmw vs. yao? efficient secure two-party computation with low depth circuits. In: Sadeghi A-R, editor. Financial Cryptography and Data Security. Berlin: Springer; 2013. p. 275–92.
22. Demmler D, Schneider T, Zohner M. ABY - A framework for efficient mixed-protocol secure two-party computation, February 8-11. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015. San Diego: Internet Society; 2015.
23. Ladner RE, Fischer MJ. Parallel prefix computation. J ACM. 1980;27(4): 831–8.
24. Garay J, Schoenmakers B, Villegas J. Practical and secure solutions for integer comparison. In: Public Key Cryptography. Berlin: Springer; 2007. p. 330–42.
25. Waksman A. A permutation network. J ACM. 1968;15(1):159–63.

## Publisher's Note