Bioinformatics, 34, 2018, i160–i168
doi: 10.1093/bioinformatics/bty269
ISMB 2018

OXFORD

AnoniMME: bringing anonymity to the Matchmaker Exchange platform for rare disease gene discovery

Bristena Oprisanu* and Emiliano De Cristofaro

Department of Computer Science, University College London, London WC1E 6BT, UK

*To whom correspondence should be addressed.

Abstract

Summary: Advances in genome sequencing and genomics research are bringing us closer to a new era of personalized medicine, where healthcare can be tailored to the individual's genetic makeup and to more effective diagnosis and treatment of rare genetic diseases. Much of this progress depends on collaborations and access to data, thus, a number of initiatives have been introduced to support seamless data sharing. Among these, the Global Alliance for Genomics and Health has developed and operates a platform, called Matchmaker Exchange (MME), which allows researchers to perform queries for rare genetic disease discovery over multiple federated databases. Queries include gene variations which are linked to rare diseases, and the ability to find other researchers that have seen or have interest in those variations is extremely valuable. Nonetheless, in some cases, researchers may be reluctant to use the platform since the queries they make (thus, what they are working on) are revealed to other researchers, and this creates concerns with respect to privacy and competitive advantage.

In this paper, we present AnoniMME, a framework geared to enable anonymous queries within the MME platform. The framework, building on a cryptographic primitive called Reverse Private Information Retrieval, let researchers anonymously query the federated platform, in a multi-server setting—specifically, they write their query, along with a public encryption key, anonymously in a public database. Responses are also supported, so that other researchers can respond to queries by providing their encrypted contact details.

Availability and implementation: https://github.com/bristena-op/AnoniMME

Contact: bristena.oprisanu.10@ucl.ac.uk

1 Introduction

Advances in genome sequencing and genomics are enabling tremendous progress in medicine and healthcare, paving the way to making the prevention, diagnosis and treatment of diseases tailored to the individual's specific genetic makeup, thus becoming cheaper and more effective. Researchers are also gaining a better understanding, and developing more successful treatments of rare genetic diseases. However, even though sequencing costs have plummeted from billions to thousands of dollars over the past 15 years (https://www.genome.gov/sequencingcosts/), it is still hard for researchers to gain access to genomic data, especially those pertaining to rare conditions.

Therefore, seamless progress in genomics research hinges on the ability to collaborate and share data among different institutions. Indeed, funding agencies often require that data sharing is considered in grant applications, and a number of initiatives have been announced to gather and share genomic data. For instance, the All Of Us Research Program (formerly known as the Precision Medicine initiative) was launched in the US in 2015, aiming to collect health and genetic data from one million citizens. Similar projects exist elsewhere, e.g. in the UK, Genomics England is sequencing the genomes of 100 000 patients, focusing on rare diseases and cancer. There are also initiatives specifically targeting data sharing, such as the NIH's Genomic Data Commons (GDC), which provides the cancer research community with a unified data repository across cancer genomic studies (https://gdc.cancer.gov/).

Aiming to foster collaborations, the Global Alliance for Genomics and Health (GA4GH) [https://www.ga4gh.org] was established, with core funding from NIH, Wellcome and Canada's CanShare, with the explicit goal of making data sharing between institutes simple and effective. The GA4GH has developed several platforms, e.g. the Beacon Project (Global Alliance for Genomics and Health, 2016), allowing

researchers to search if a certain allele exists in a database of genomic data, as well as the Matchmaker Exchange (MME; Philippakis *et al.*, 2015), which facilitates rare disease discovery.

In this paper, we focus on the latter; The MME platform connects multiple distributed databases through an Application Programming Interface (API) and allows researchers to query for genetic variants in other databases in the network. That is, MME acts as a portal supporting simultaneous querying over multiple databases that are members of the exchange. More specifically, MME allows a researcher to query a specific gene, e.g. 'AP3B2' (a gene where rare mutations have been linked to early-onset epileptic encephalopathy). If a match is found, the researcher is notified of all matches within all databases in the MME, and can get in touch with the user that submitted the case on which a match is generated. Note that, querying a gene really implies querying a known rare variation of that gene.

However, researchers might be reluctant to use the platform since the queries they make are revealed to other researchers, and this exposes what they are working on and what kinds of patients they might have, ultimately resulting in loss of privacy and competitive advantage. Indeed, MME currently requires researchers to submit a registration application to be given access to the platform, with the goal of preventing misuse of the system, thus, queries made on this platform are not anonymous and are revealed to all other researchers with an interest in the same gene.

1.1 Problem statement

This motivates the need to support *anonymous querying* on MME, so that a researcher's interest a gene is not broadcast, but only communicated to relevant contacts, i.e. researchers with same interests or willing to collaborate. To this end, we present AnoniMME, a framework letting researchers anonymously query a gene within the MME, without violating any of MME's current functionalities and requirements. We build AnoniMME using a cryptographic primitive called Reverse Private Information Retrieval (Reverse PIR), using a model similar to that presented by the anonymous messaging system Riposte (Corrigan-Gibbs *et al.*, 2015), while creating queries and implementing the same functionalities as in MME. In other words, researchers can perform anonymous queries to the federated platform, in a multi-server setting, by writing their query, along with a public encryption key, anonymously, in a public database. AnoniMME also supports responses, so that other researchers can respond to queries by providing their encrypted contact details.

1.2 Solution intuition

We build queries in regular epochs, where the length of each epoch is based on the number of write requests. In order to anonymously write to the database, the user selects a random row of the the database and splits the query, containing the gene and her public key, into shares, one for each server (which we denote as *node* servers). This way, the node servers cannot learn anything about the write request, if at least one of them is honest. Then, a *master* server can gather queries that have been collected during an epoch from the node servers and collate them together to recover and publish the actual queries. The MME matching system can then be used in order to generate matches for the queries, in the usual manner and contact details of other researchers/clinicians can be exchanged, encrypted using the public key, and published in the same row as the queried gene, in an adjacent column.

To demonstrate the practicality of AnoniMME, we implement and evaluate our prototype experimentally. We do so in two different settings, one involving two node servers and a master server, and another involving six node servers (and a master server). In both settings, the nodes collect write requests during an epoch, and then forward them to the master server which collates them and publishes the final database.

1.3 Contributions

In summary, our paper makes several contributions:

i. We present AnoniMME, a framework enabling anonymous queries within MME, without breaking any of its current security and functionality requirements.
ii. We build AnoniMME from Reverse PIR (Corrigan-Gibbs *et al.*, 2015), using an information-theoretic approach, extending queries to support public key encryption of contact details and adding a response phase so that users can also anonymously reply to queries.
iii. We show, experimentally, that AnoniMME is efficient and scalable, and can bring anonymity to MME with low overhead. Therefore, we are confident that it can be deployed in the wild and further encouraging researchers to share genomic data.

1.4 Paper organization

The rest of the paper is organized as follows. In the next section, we introduce our approach; specifically, after reviewing the MME platform, we define entities, operations and threat model of our system and present a first attempt at designing a anonymous-query mechanism for MME. In Section 3, we then describe the methods used for collision handling and collision recovery, present the *n*-server protocol and evaluate the performance of the proposed protocol on the client side. Next, in Section 4, we discuss the results from our experimental evaluation and place our protocol in the context of related work. Finally, the paper concludes in Section 5.

2 Approach

2.1 Matchmaker Exchange

As mentioned, GA4GH was established, in 2013, aiming to support simple mechanisms for sharing data between institutes. The GA4GH has developed and deployed various systems, including MME (Philippakis *et al.*, 2015), which facilitates rare disease gene discovery and constitutes the main focus of our work. MME is a federated platform that facilitates the identification of cases with similar phenotypic and genotypic profiles through a standardized API. Essentially, it enables searches in multiple databases, without having to query all of them separately or deposit data in each of them. As of March 2018, it involves seven organizations with full member status (AGHA Patient Archive, DECIPHER, GeneMatcher, Matchbox, Monarch Initiative, MyGene2 and PhenomeCentral) and eight additional participant organizations.

The Matchmaker Exchange Application Programming Interface (MME API; Buske *et al.*, 2015) fully specifies the data format and the protocol for querying databases to identify individuals with similar phenotypic profiles and genetic variations. To ensure the accuracy of the patient comparison, similar phenotypes are determined by matching identical or ontologically similar with the Human Phenotype Ontology (HPO). The MME API also specifies the format of both the query, which is sent to participating databases (called 'matchmaking service') and the response, which contains information about matching individuals in the remote database. It is implemented under a query-by-example methodology: a user can query a specific gene, e.g. 'AP3B2', and she will be notified of all

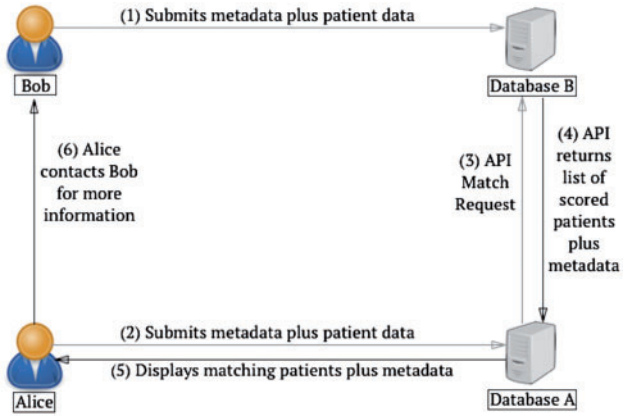

Fig. 1. Visual representation of a MME query sequence

matches within all databases in the MME. Note that querying a gene really implies querying a known rare variation of that gene. If a match is found, the user receives a Case ID for the match, information about the user that submitted the case on which a match is generated, such as name, institution and email address, as well as the corresponding candidate gene or phenotype. In order to query the platform, users must be registered with one of the member databases and have a clinician/researcher account. Some of the member databases allow for patient/family registrations as well, however, the submissions made by these types of users are excluded from matching via MME, due to the current MME rules.

The query protocol is illustrated in Figure 1. A user, Bob, sends the metadata (i.e. Case ID, submitter information) as well as the patient data (gene and/or phenotype) to Database B. Another user, Alice, submits a similar case to Database A; Database A then sends an MME API match request to Database B, which performs the match and returns a list of scored patients, along with relevant metadata, to Database A. After receiving the match results, Database A informs Alice, providing contact information for Bob. The result of querying MME yields a list of matches, where each match has a *patient* object, i.e. the information on the matched patient, consisting of the same information as described in the query and a *score* object. The scoring of the patients is done according to how well the results patient matches the query patient, i.e. it is a numerical value in the range $[0, 1]$, where 0.0 is a poor match and 1.0 a perfect match.

2.2 Entities and operations

As discussed in Section 1, this paper presents AnoniMME, a framework geared to enable anonymous queries within the MME platform, i.e. anonymously querying the federated platform to find patients with similar gene mutations or phenotypes. It involves the following entities:

Querying Users: Researchers/clinicians who query the system to find other users that have patients with a rare mutation or an interest in the same gene. As discussed later, they generate a write request specifying the row at which their query, i.e. the gene of interest and their public key, will be processed.

Responding Users: Researchers/clinicians replying to an existing query. They use the public key of a querying user to encrypt their contact details and generate a write request for the same row as the gene of interest including their (encrypted) contact details.

Nodes: The servers collecting write requests from the users. These are aggregated until the end of an epoch, based on the maximum number of write requests. Each node server can be run by one of the current MME members.

Master Server: A server that gathers the databases from each node at the end of an epoch, and publishes the database with all the write requests revealed. The master server role can also be assigned to one of the existing MME members, and can be re-assigned to another member at the end of each epoch.

Overall, AnoniMME implements the following operations:

- *Query Write Request:* On input row i, query gene X, and public key PK, a querying user generates n write requests, one for each node. Each write request is generated by encoding the gene and the public key into n vectors, so that all of them combined will write the gene/public key at index i.
- *Query Response Request:* On input row i, encrypted contact details c, a responding user generates n write requests, one for each node. Write requests are generated, once again, by encoding the encrypted contact details into n vectors.
- *Database Collation:* On input n databases, the master server collates them into one final database and publishes it.

2.3 Security model

AnoniMME aims to guarantee the following three security goals:

i. *Correctness.* When all nodes execute the protocols correctly and send data to the master server at the end of an epoch, the resulting database contains all the write requests processed as if the requests were directly applied to the final database.
ii. *Anonymous write.* The probability that an adversary guesses at which particular row a user has written is only negligibly better than random guessing.
iii. *Disruption resistance.* An adversary controlling n users can make at most n write requests (i.e. there is a limit to the number of write requests each user can make during an epoch).

2.3.1 Threat model

We assume that the users of the system are untrusted, and may collude with the nodes, the master server, or other users in order to violate the security properties of the system. Both the master server and the nodes are trusted for availability and to follow the protocol correctly, under the assumption that at least one of the nodes is honest (i.e. does not collude with other nodes). We do not consider external adversaries, since their actions can be mitigated via standard network security techniques (i.e. using a secure and authenticated communication channel). Finally, note that the security model of AnoniMME mirrors that of Riposte (Corrigan-Gibbs *et al.*, 2015).

2.4 A first attempt

We now present a first attempt at instantiating AnoniMME, and discuss its limitations, which we address in the actual construction of AnoniMME presented in Section 3.2.

2.4.1 Intuition

We start by attempting to build from a simple extension of Reverse PIR (Corrigan-Gibbs *et al.*, 2015). More specifically, we implement the query phase using the same mechanism of Riposte, i.e. we let users anonymously submit the gene of interest, along with their public key, with a 'write request.' We then add a response phase, allowing users with an interest in the same gene to respond—specifically, by encrypting their contact information using the public key contained in the query and adding it to another write request.

In the following, we present a construction assuming the presence of two servers (S_1 and S_2) and a database with l rows.

2.4.2 Query phase

Assume user A wants to anonymously query gene X_A. She builds a write request, consisting of (X_A, PK_A), where PK_A is her public key and writes this at row i in the database. More specifically, she picks $2l$ random numbers, $r_1, r_2, \ldots, r_l$ and $s_1, s_2, \ldots, s_l$, where l is the size of the database. The query write request vectors are constructed as follows:

$$v_1 = (r_1, r_2, \ldots, r_i + X_A, \ldots, r_l),$$
$$v_1' = (s_1, s_2, \ldots, s_i + \mathrm{PK}_A, \ldots, s_l),$$
$$v_2 = (-r_1, -r_2, \ldots, -r_i, \ldots, -r_l),$$
$$v_2' = (-s_1, -s_2, \ldots, -s_i, \ldots, -s_l).$$

Note that $v_1 + v_2 = X_A \cdot \mathrm{e}_i$, and $v_1' + v_2' = \mathrm{PK}_A \cdot \mathrm{e}_i$, where e_i denotes the unit vector with 0's at all positions except at position i, where it is equal to 1 and thus the construction is correct. Then, A sends (v_1, v_1') to S_1, and (v_2, v_2') to S_2.

Write requests are collected until the end of an epoch, when the servers combine their local states and publish the database with the queries. As long as the two servers do not collude, none of them can re-construct what any given user has written, i.e. none of the servers can recover the gene or public key of the user sent in the write request. Also, in order to achieve disruption resistance, one can limit the number of queries to one per user for each phase of the epoch.

2.4.3 Response phase

After the database with the queries is published, the response phase begins. Here, we can rely on MME's algorithm to generate matches on existing MME data, and simply extend it to encrypt the contact details of the relevant users with an interest in the same gene. This would be inline with the current privacy policy of the MME, as contact details of researchers with an interest in the same gene are already shared.

Users can also be given an option to voluntarily provide their contact details as follows. If user B notices that another researcher (user A) has an interest in the same gene X, say at row i of the database, she gets A's public key PK_A and encrypt her contact information (C_B) under PK_A and generates a write request as a share of $Enc_{\mathrm{PK}_A}(C_B)$, in a similar manner to the first epoch. More specifically, she chooses random $r_1', \ldots, r_l'$ and forms the following vectors:

$$u_1 = (r_1', \ldots, r_i' + Enc_{\mathrm{PK}_A}(C_B), \ldots, r_l'),$$
$$u_2 = (-r'_1, \ldots, -r_{i'}, \ldots, -r'_l)$$

User B then sends u_1 to server S_1 and u_2 to S_2. At the end of this epoch, the results are being published in a column adjacent to the queried gene and the public encryption key. The querying users can use the database to find the row of interest (in this case i), decrypt the contact details and get in touch with the responding users.

2.4.4 Correctness and security

It is straightforward to see that the construction is correct, since, if all nodes execute the protocols correctly the result of combining all their local database states at the end of an epoch by the master server will result in revealing all the write requests processed. An adversary's advantage of guessing at which a certain user has written in the final database is the same as random guessing, hence, the construction guarantees anonymous writes. Disruption resistance can be also achieved in a straightforward manner since MME requires users to register on one of the databases, so they can allow maximum one write request per registered user per epoch.

2.4.5 Limitations

Alas, this construction has the following limitations:

i. *Collisions:* They might occur for writes generated by honest users, which all want to write at the same row.
ii. *Maliciously-formed write requests:* A malicious user can easily send a malformed request to the servers, making all the data within the database non-recoverable.

3 Materials and methods

In this section, we provide methods for collision handling for our first attempt and use it to provide a description of the n-server protocol. We also evaluate the proposed method in terms of time and bandwidth required in order to asses the feasibility of the proposed construction.

3.1 Handling collisions

As discussed previously, collisions might occur whenever multiple users want to write at the same row. Aiming to address them, we set the database size to be large enough to accommodate write requests at a 95% non-collision rate. In other words, 5% of the queries will likely fail due to collisions and will need to be re-submitted.

3.1.1 Minimizing collisions

Our intuition is to follow a 'balls and bins' approach, i.e. if we throw m balls uniformly and randomly into the l bins, we can estimate how many bins will contain exactly one ball. In our model, we can associate write requests to the m balls and the rows of the database to the l bins. Let B_{ij} be the event that ball i falls into bin j: for all i and j, we have $\Pr[B_{ij}] = \frac{1}{l}$. Then, let $O_j^{(1)}$ be the event that exactly one ball falls in bin j. We have that:

$$\Pr\left[O_j^{(1)}\right] = \frac{m}{l}\left(1 - \frac{1}{l}\right)^{m-1} \approx \frac{m}{l} - \left(\frac{m}{l}\right)^2 + \frac{1}{2}\left(\frac{m}{l}\right)^3$$

using the binomial theorem and ignoring low order terms. Then, $l\Pr\left[O_j^{(1)}\right]$ is the expected number of bins with exactly one ball, i.e. the expected number of messages successfully received. Dividing by m, we get the expected success rate as

$$E[\mathrm{SuccesRate}] = \frac{l}{m}\Pr\left[O_j^{(1)}\right] \approx 1 - \frac{m}{l} + \frac{1}{2}\left(\frac{m}{l}\right)^2$$

Thus, for a 95% expected success rate, we need $l \approx 19.5m$.

In AnoniMME, in order to set the size of the database, we need to estimate the expected number of write requests for each epoch. Looking at the three MME members which show statistics on the number of users, we find that GeneMatcher has 4066 registered users, MyGene2 345 registered families and Decipher 247 registered projects (users have to be part of a project in order to join Decipher) as of November 2017. This yields an average of approximately 1550 users per database. Assuming that this is representative of the number of users for all MME databases, we can approximate the total number of users to be in the order 10 000. We also need to estimate how many users make queries in each epoch: assuming 5% of users do so at each epoch, each epoch can run for 500 queries, yielding a database of size $l \approx 10\,000$. Further, note that we design AnoniMME's write request so that the row number at which we

write is determined at random, given the number of write requests in the epoch as well as the database size, in order to avoid biases in choosing rows. This method, however, does not provide any way to recover in the case where a collision occurs, in that case the queries are irrecoverable and the users would need to re-submit their queries in a future epoch.

3.1.2 Recovering from collisions

We also use a simple technique for recovering from collisions if/when these occur. Assume α messages have been written at row i, i.e. we have $a = m_1 + m_2 + \cdots + m_\alpha$. Inspired by (Corrigan-Gibbs *et al.*, 2015), we can modify the way in which the queries are built to recover each of the individual message m_j, for $1 \leq j \leq \alpha$; specifically, we can use a system of α equations, which allows us to solve for each of the colliding messages. Without loss of generality, we consider the case $\alpha = 2$ and explain how to recover from collisions occurring for the gene name, but similar methods can be used for $\alpha > 2$ and to recover public key and/or encrypted contact details. When a collision occurs at row i, we have an entry $a = X_A + X_B$, where X_A is the gene sent by user A, and X_B is the gene sent by user B. If, rather than just sending the queried gene X, users send (X, X^2), we can recover X_A and X_B by solving a system of two equations with two variables.

In this case, we also compute the size of the database needed for an expected success rate as follows:

$$E[\text{SuccessRate}] = \frac{l}{m}\Pr\left[O_j^{(1)}\right] + \frac{2l}{m}\Pr\left[O_j^{(2)}\right],$$

where $l\Pr\left[O_j^{(1)}\right]$ is the expected number of rows with exactly one write request applied to them, computed as before and $2l\Pr\left[O_j^{(2)}\right]$ is the expected number of rows with exactly two write requests applied to them. Computing $\Pr\left[O_j^{(2)}\right] = \binom{m}{2}\frac{1}{l^2}\left(1-\frac{1}{l}\right)^{m-2}$, we obtain the value of the expected success rate as:

$$E[\text{SuccessRate}] \approx 1 - \frac{1}{2}\left(\frac{m}{l}\right)^2 + \frac{1}{3}\left(\frac{m}{l}\right)^3.$$

In this case, for an epoch of m write requests, with a 95% expected success rate, we need a database with $l' \approx 2.7\,m$ cells (two columns and $l = \frac{l'}{2}$ rows). This implies that with 500 write requests per epoch, the database needs $l' \approx 2.7 \cdot 500 = 1350$ cells for each vector.

We now generalize for any value of α. Users submit $X, X^2, \ldots, X^\alpha$ for any gene X to be queried. This allows us to recover from an α-way collision as, in that case we obtain a system of α equations with α variables. The expected success rate is:

$$E[\text{SuccessRate}] = \frac{l}{m}\Pr\left[O_j^{(1)}\right] + \frac{2l}{m}\Pr\left[O_j(2)\right] + \\ + \cdots + \frac{\alpha l}{m}\Pr\left[O_j^{\alpha}\right]$$

where $l\Pr\left[O_j^{(k)}\right]$ is the expected number of rows with exactly k write requests applied to them. Each $\Pr\left[O_j^{(k)}\right]$ is computed as $\Pr\left[O_j^{(k)}\right] = \binom{m}{k}\frac{1}{l^k}\left(1-\frac{1}{l}\right)^{m-k}$. Hence, we obtain:

$$E[\text{SuccessRate}] \approx 1 + \frac{(-1)^{\alpha+1}}{\alpha!}\left(\frac{m}{l}\right)^{\alpha} + \frac{(-1)^{\alpha+2}}{(\alpha+1)!}\left(\frac{m}{l}\right)^{\alpha+1}$$

We solve this equation for l, given the expected success rate $E[\text{SuccessRate}]$, the collision recovery factor α and m the number of write requests to be written in a certain epoch. If this method is used throughout both epochs, colliding requests from the query phase will have to be recovered before the response phase can begin.

Due to the nature of our query/response model, we can expect collisions to occur more often in the response phase. Hence, we will can build the system using different collision recovery factors α_q for the query phase and α_r for the response phase, with $\alpha_r \geq \alpha_q$.

3.2 *N*-server construction

We now present the generalized model for the case with n servers and a database with l rows. We use collision parameters α_q and α_r for the query and response phase, respectively. The various steps of the construction are illustrated in Figure 2.

3.2.1 Query phase

Assume user A wants to query gene X_A, but does not want to reveal that she is the person querying it. As in the construction presented in Section 2.4, A builds her write request, consisting of (X_A, PK_A), where PK_A is her public key, aiming to write at row i in the database. She picks random numbers $r_{1,1}, \ldots, r_{1,l}, r_{1,l+1}, \ldots r_{1,l\alpha_q}, r_{2,1}, \ldots, r_{n,l\alpha_q}$ and $r'_{1,1}, \ldots, r'_{1,l}, r'_{1,l+1}, \ldots, r'_{1,l\alpha_q}, r'_{2,1}, \ldots, r'_{n,l\alpha_q}$, where l is the size of the database, n the number of nodes the write request will be sent to, and α_q the number of allowed collisions. The query write request vectors are then constructed as follows:

$$\begin{aligned}
v_{1,1} &= (r_{1,1}, r_{1,2}, \ldots, r_{1,i} + X_A, \ldots, r_{1,l}) \\
v'_{1,1} &= (r'_{11,}, r'_{1,2}, \ldots, r'_{1,i} + \text{PK}_A, \ldots, r'_{1,l}) \\
v_{1,2} &= (r_{1,l+1}, \ldots, r_{1,l+i} + X_A^2, \ldots, r_{1,2l}) \\
v'_{1,2} &= (r'_{1,l+1}, \ldots, r'_{1,l+i} + \text{PK}_A^2, \ldots, r'_{1,2l}) \\
&\vdots \\
v_{1,\alpha_q} &= \left(r_{1,l(\alpha_q-1)+1}, \ldots, r_{1,l(\alpha_q-1)+i} + X_A^{\alpha_q}, \ldots, r_{1,l\alpha_q}\right) \\
v'_{1,\alpha_q} &= \left(r'_{1,l(\alpha_q-1)+1}, \ldots, r'_{1,l(\alpha_q-1)+i} + \text{PK}_A^{\alpha_q}, \ldots, r'_{1,l\alpha_q}\right)
\end{aligned}$$

$$\begin{aligned}
v_{2,1} &= (r_{2,1}, r_{2,2}, \ldots, r_{2,i}, \ldots, r_{2,l}) \\
v'_{2,1} &= (r'_{2,1}, r'_{2,2}, \ldots, r'_{2,i}, \ldots, r'_{2,l}) \\
&\vdots \\
v_{n,1} &= -(r_{1,1}, r_{1,2}, \ldots, r_{1,i}, \ldots, r_{1,l}) - \sum_{j=2}^{n-1} v_{j,1} \\
v'_{n,1} &= -(r'_{11,}, r'_{1,2}, \ldots, r'_{1,i}, \ldots, r'_{1,l}) - \sum_{j=2}^{n-1} v'_{j,1} \\
&\vdots \\
v_{n,\alpha_q} &= -\left(r_{1,l(\alpha_q-1)+1}, \ldots, r_{1,l(\alpha_q-1)+i}, \ldots, r_{1,l\alpha_q}\right) - \sum_{j=2}^{n-1} v_{j,\alpha_q} \\
v'_{n,\alpha_q} &= -\left(r'_{1,l(\alpha_q-1)+1}, \ldots, r'_{1,l(\alpha_q-1)+i}, \ldots, r'_{1,l\alpha_q}\right) - \sum_{j=2}^{n-1} v'_{j,\alpha_q}.
\end{aligned}$$

The querying user A ends $\left(v_j, v'_j\right)$ to server j for each j, $1 \leq j \leq n$, where $v_j = (v_{j,1}, \ldots, v_{j,\alpha_q})$ and $v'_j = (v'_{j,1} \ldots, v'_{j,\alpha_q})$. We also consider the special case of $\alpha_q = 1$, when there is no recovery for collisions, but, instead, we adjust the database size according to the minimizing collisions case. The servers collect write requests until the end of the epoch and then send their local databases to the master server, which will combine them to reveal the database.

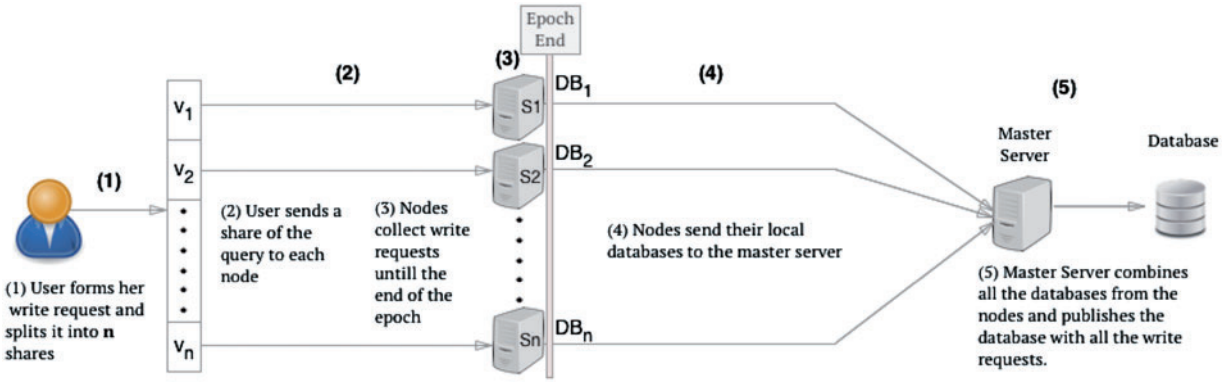

Fig. 2. *n*-server write request processing. At the end of the epoch the Master Server publishes the database with all the write requests and the nodes will be reset to hold an empty database

3.2.2 Response phase

As the database with the queries is published, the response phase begins. As discussed in Section 2.4, we can rely on MME's algorithm to generate matches on existing data from the platform, encrypt the contact details of the relevant users with an interest in the same gene and extend it to allow for voluntary responses. More specifically, user B can add their contact details C_B by sending a write request as a share of $c = Enc_{\mathrm{PK}_A}(C_B)$, in a similar manner to the first epoch. That is, first, she picks random $s_{1,1}, \ldots, s_{1,l}, s_{1,l+1}, \ldots, s_{1,l\alpha_r}, s_{2,1}, \ldots, s_{n,l\alpha_r}$ and forms the following vectors:

$$
\begin{aligned}
u_{1,1} &= \left(s_{1,1}, \ldots, s_{1,i} + c, \ldots, s_{1,l}\right),\\
u_{1,2} &= \left(s_{1,l+1}, \ldots, s_{1,l+i} + c^2, \ldots, s_{1,2l}\right),\\
&\vdots\\
u_{1,\alpha_r} &= \left(s_{1,l(\alpha_r-1)+1}, \ldots, s_{1,l(\alpha_r-1)+i} + c^{\alpha_r}, \ldots, s_{1,l\alpha_r}\right)\\
u_{2,1} &= \left(s_{2,1}, \ldots, r'_{2,i}, \ldots, s_{2,l}\right)\\
&\vdots\\
u_{n,1} &= -\left(s_{1,1}, s_{1,2}, \ldots, s_{1,i}, \ldots, s_{1,l}\right) - \sum_{j=2}^{n-1} u_{j,1},\\
&\vdots\\
u_{n,\alpha_r} &= -\left(s_{1,l(\alpha_r-1)+1}, \ldots, s_{1,l(\alpha_r-1)+i}, \ldots, s_{1,l\alpha_r}\right) -\\
&\quad -\sum_{j=2}^{n-1} u_{j,\alpha_r}
\end{aligned}
$$

User B then sends $u_j = \left(u_{j,1}, \ \ldots \ u_{j,\alpha_q}\right)$ to server S_j. At the end of this epoch, the results are being published in a column adjacent to the queried gene and the public encryption key. In case of collisions, the individual ciphertexts can be recovered up to α_r collisions. Finally, the querying users can use the database to find the row of interest (in this case i) and decrypt the contact details received and contact the person.

3.3 Experimental evaluation

We now present an experimental evaluation of AnoniMME, aiming to demonstrate its practicality for real-world deployment.

We have implemented the n-server construction (Section 3.2) using Python 3.6 and evaluated our prototype on a Macbook Pro running MacOS Sierra 10.12.6 and equipped with a 2.7 GHz Intel i5 processor, and 16GB of RAM. Experiments are performed in two different settings, with two and six node servers, respectively, and always averaged over 1000 executions. We also use three different epoch sizes, namely, 100, 500 and 1000 write requests per epoch during the query phase. For the response phase, we keep the database size fixed from the query phase. Overall, we evaluate running times needed to generate the write requests and the bandwidth overhead supporting the recovery of 2, 5 and 10 colliding messages, all on the client side (i.e. one request per epoch).

The servers run Flask with RESTful interface, so we use HTTP requests to send the messages and the payload is built in JSON, therefore, we measure, in bytes, the size of the JSON payload (plus HTTP headers) to estimate the total bandwidth required for sending write requests.

On the client side, the cryptographic layer includes generating public/private keys (done only once) and building the vectors to be sent to the n servers as part of the write request, which incurs $O(n)$ complexity. Gene name and contact details are assumed to be no longer than 64 characters, while random numbers used for vector generation during query phase are up to 1 024 bits long, for $\alpha_q \in \{1, 2\}$ and $\alpha_r = 2$. For the response phase, the length of the random values varies according to the collision recovery factor α_r. For $\alpha_r = 5$, their length is 2560 bits, while for $\alpha_r = 10$ it is 5120.

Finally, note that plausible gene queries are generated using the set of gene symbols (e.g. 'BRCA2') from http://gfuncpathdb.ucdenver.edu/iddrc/iddrc/data/officialGeneSymbol.html.

3.3.1 Two node servers

We start with the setting involving two node servers and a master server, considering epochs of size 100, 500 and 1000. As mentioned above, we evaluate bandwidth overhead and running times required for query and response write requests.

The database size required for each of the three test cases is calculated according to the method presented in Section 3.1 for minimizing collisions, thus, $l = 19.5m$, where l denotes the number of rows required and m is the number of write requests for the epoch. It follows that the l amounts to 2000, 10 000 and 20 000 rows for m equal to 100, 500 and 1000, respectively.

Running times for both the query write and the response (considering $\alpha_r \in \{2, 5, 10\}$) are shown in Figure 3. Overall, we find that, during the query phase, with a database size of 2000 rows, it takes approximately 0.014 s to generate vectors in our testbed. Running times scale linearly, i.e. it takes 0.062 s with 10 000 rows and 0.126 s with 20 000 rows. The bandwidth overhead, shown in Figure 4, ranges from 2.5 MB for the smallest database size to 25 MB for the largest case considered in our test cases, which can be

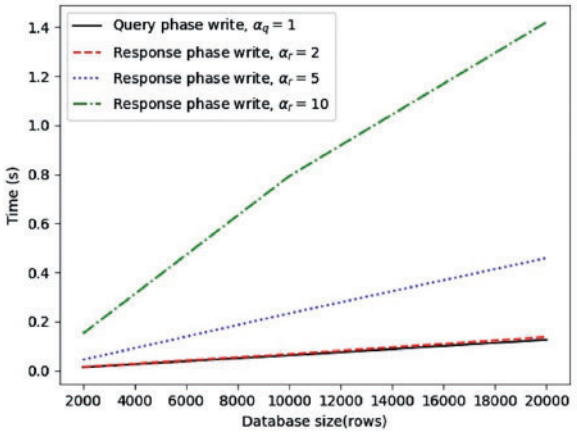

Fig. 3. Two nodes running times for query write request, response write request with recovery from 2 collisions, response write request with recovery from 5 collisions, response write request with recovery from 10 collisions

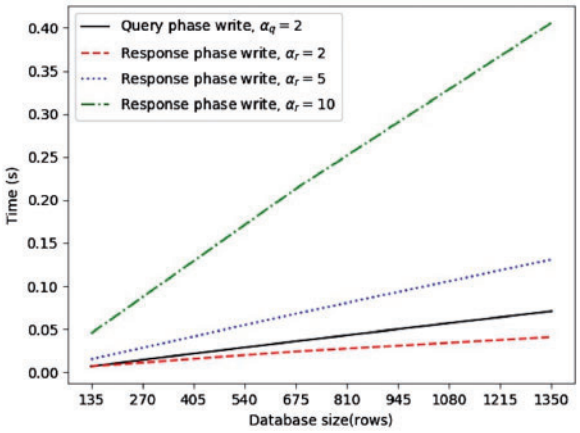

Fig. 5. Six nodes running times for query write request, response write request with recovery from 2 collisions, response write request with recovery from 5 collisions, response write request with recovery from 10 collisions

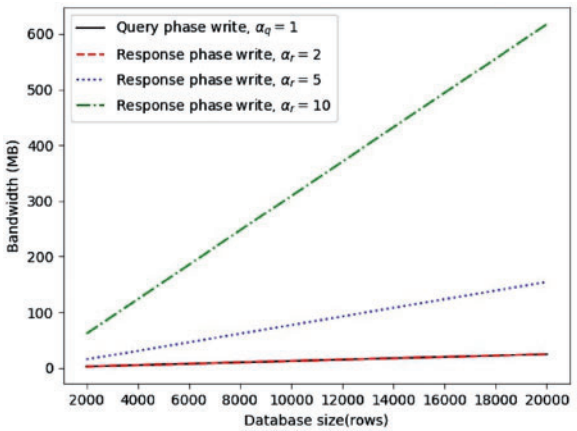

Fig. 4. Two nodes bandwidth averages for query write request, response write request with recovery from 2 collisions, response write request with recovery from 5 collisions, response write request with recovery from 10 collisions

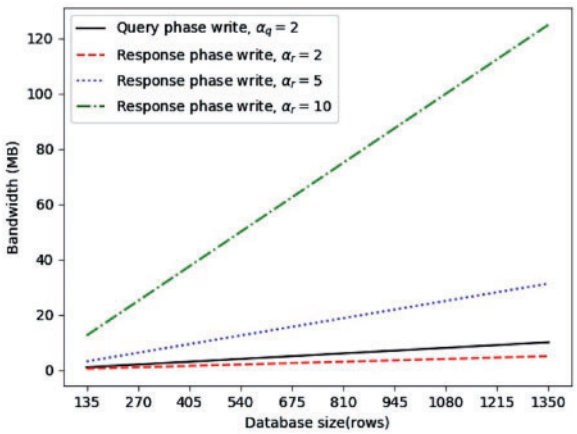

Fig. 6. Six nodes bandwidth averages for query write request, response write request with recovery from 2 collisions, response write request with recovery from 5 collisions, response write request with recovery from 10 collisions

considered an acceptable amount of traffic expected from the client side.

For the response phase, we find that, when $\alpha_r = 2$, the results are similar to the query phase since responding users need to generate two vectors in order to allow collision recovery, same as for the querying user. When α_r equals 5 or 10, we notice an increase in both running times and bandwidth. Nonetheless, computational complexity is still acceptable, since, even with the largest database size, write request generation takes less than 0.5 s for $\alpha_r = 5$ and less than 1.5 s for $\alpha_r = 10$. Communication overhead, on the other hand, increases to 160 MB and 617 MB, respectively, with the largest database size.

However, one can adjust the collision minimization parameter so that 10-way collision recovery is not needed.

3.3.2 Six node servers

We also experiment with an instantiation of AnoniMME using six node servers, thus mirroring the current MME setting, which involves seven members. Once again, we consider three settings (100, 500 and 1000 write requests per epoch), and obtain the resulting database size based on the recovery from collisions method discussed in Section 3.1. We support recovery from two colliding messages for the query phase, i.e. $\alpha_q = 2$. Therefore, the number of rows required is $l = \frac{2.7m}{2}$, where m is the number of write requests for the epoch, thus, l equals 135 675 and 1350 for $m = 100$, 500 and 1000, respectively. As per the response phase, we run tests with different values $\alpha_r \in \{2, 5, 10\}$, considering the database size fixed as for the query phase.

Once again, we estimate running times (Fig. 5) and the bandwidth overhead (Fig. 6). Even though this requires more vectors to be generated by the users compared to the two-node setting (Section 3.3.1), we observe a considerable decrease in both running times and bandwidth overhead for the same epoch sizes due to the decreased number of rows in the database. Specifically, computational complexity is again linear over all test cases, but the write request generation taking less than half the time. There is also a big improvement in terms of communication complexity: even in the most bandwidth-heavy case (i.e. $\alpha_r = 10$), with 1000 write requests per epoch, we observe a 5-fold improvement, with bandwidth decreasing from 617 MB to 125 MB.

On the other hand, the query phase is less efficient than the response phase (with $\alpha_r = 2$), compared to the two-node setting, since the querying user now has to generate two vectors for each gene so

that collision recovery is possible, hence, four vectors in total; whereas, the responding user only generates two vectors.

4 Discussion

In this section, we discuss the experimental data, as well as provide an overview of related work.

4.1 Remarks

Our experimental evaluation attests to the practicality of using AnoniMME to bring anonymity to the MME. Overall, using the method proposed in Section 3.3.1 to recover write requests in case of collisions yields better running times and bandwidth complexities, even when the number of nodes increases.

Since AnoniMME is based on Riposte (Corrigan-Gibbs *et al.*, 2015), one might want to compare the two systems; however, Riposte focuses on experimental results from the server's perspective, while we evaluate performance from a user perspective.

Also note that the bandwidth overhead in our n-server construction is non-negligible, especially with a high collision recovery factor and increasing database sizes (as discussed in Section 3.3.1). A possible solution would be to use distributed point functions to reduce bandwidth complexity, similar to Riposte. However, we leave this to future work.

As the anonymity set size in AnoniMME corresponds to the number of users querying in a given epoch, one could increase it by requiring users to send empty queries to the system, following a certain probability distribution. The write requests would be formed as discussed in Section 3.2, although, instead of inputting a gene, the public key, or the contact details, the users just send an empty query. This is also used in Riposte, to minimize statistical disclosure attacks on their platform.

Finally, note that our implementation currently allows for 64 character messages, thus, queries can also include phenotypes from the HPO (as currently supported by MME), although, to ease of presentation we have discussed our experiments by only considering gene names. In future work, we plan to conduct a user study simulating a real-world deployment of AnoniMME with users of the MME, aiming to evaluate its usability with respect to anonymity protection, delays introduced by epochs, etc.

4.2 Related work

Rapid and effective progress in genomics and personalized medicine is often promoted as being dependent on the ability to share sequenced genomes, and make them accessible to researchers for different research purposes. However, it is often hard to share data due to privacy, ethical, legal and informed consent hurdles. To address these issues, a few privacy-preserving methods have been presented to facilitate genomic data sharing. Kamm *et al.* (2013) use secret sharing for distributing data among several entities. Using secure multi-party computations on the data, computations can be done across multiple independent entities, without violating the privacy of individual donors or leaking the data to third parties. Then, Wang *et al.* (2015) allow clinicians to find similar patients in biorepositories, with similarity being defined as the edit distance. Their construction is based on a combination of a novel genomic edit distance approximation algorithm and new construction of private set difference size protocols. Chen *et al.* (2017) introduce a framework using Intel's Software Guard Extension and hardware for trustworthy computations. This way, secure and distributed computation over encrypted data is performed, respecting institutional policies and regulations for protected health information.

Another initiative developed by GA4GH, besides MME, is the Beacon Project (Global Alliance for Genomics and Health, 2016); a beacon is a service that any institution can implement to share genetic data. Users can query the system through a federated search engine, the Beacon Network. The queries are of the form '*Do you have any genomes with an 'A' at position 100 735 on chromosome 3?*', and the beacon responds with either 'Yes' or 'No', keeping all other sequence data concealed. This kind of queries can be used to either search all beacons or specific databases. The result is then shown as a list of databases where the allele has been previously observed, including the institution that holds that database. Shringarpure and Bustamante (2015) present an attack on beacons, showing that re-identification is possible using a likelihood-ratio test. Mitigations for this attack are presented by Raisaro *et al.* (2017), however these mitigations comes with a diminished utility of the beacon. The original attack has been improved by Thenen *et al.* (2017) in terms of number of queries needed to determine the presence of an individual in a beacon. Note that these attacks do not apply to MME, since no genotype information or aggregate data is released publicly and the querying is done only on specific genes, with no genotype information.

Overall, a number of attacks to anonymized/de-identified genomic data have been presented. Homer *et al.* (2008) show how to detect the presence of an individual genotype in a mixture of pooled DNA, while Gymrek *et al.* (2013) recover the surnames of individuals from a genomic data repository by profiling short tandem repeats on the Y chromosome, querying recreational genealogy databases and relying on metadata like age and state to recover the identity of the target.

As already mentioned, our construction is similar in nature to Riposte (Corrigan-Gibbs *et al.*, 2015), an anonymous broadcast messaging system, which also built using Reverse PIR. Riposte allows a large number of clients to post messages anonymously on a 'bulletin board' maintained at a small set of servers. The main goal of the system is to provide a platform for whistleblowers, allowing them to anonymously post 160 byte length messages. Besides using Reverse PIR in a different setting, and thus addressing different challenges in scalability, also note that our AnoniMME framework also allows replies to messages.

5 Conclusion

This paper presented AnoniMME, a framework geared to bring anonymity to the MME platform. Specifically, AnoniMME supports anonymous queries, by relying on Reverse PIR, while mirroring the functionalities of MME. Queries include the gene name as in MME, but also the querying user's public key and are collected during epochs whose length is based on the number of write requests. By taking advantage of the underlying MME matching protocol, these queries can be seamlessly responded to, without publicly revealing the contact details of other researchers/clinicians which generated a match, by using the public key provided to encrypt the match. Also, other users can provide their (encrypted) contact details if they so wish.

The proposed protocol is compatible with the functionalities and the requirements of MME, but adds anonymous queries with a low overhead, as we demonstrated empirically. Thus, we are confident that AnoniMME can eventually be deployed in the wild and further

encouraging researchers to share genomic data, by minimizing the possibility of exposing confidential research when using MME.

As part of future work, we plan to include and experimentally evaluate an extension to malicious users in our prototype, support the execution of the response phase over multiple query epochs, further reduce bandwidth complexity and perform a user study to evaluate its usability.

Acknowledgement

The authors thank Christophe Dessimoz for valuable feedback provided, as well as insights from users of the platform.

Funding

This work was supported by a Google Faculty Award on Enabling Progress in Genomic Research via Privacy-Preserving Data Sharing.

Conflict of Interest: none declared.

References

Buske,O.J. *et al.* (2015) The Matchmaker Exchange API: automating patient matching through the exchange of structured phenotypic and genotypic profiles. *Human Mutat.*, **36**, 922–927.

Chen,F. *et al.* (2017) PRINCESS: privacy-protecting Rare disease International Network Collaboration via Encryption through Software guard extensionS. *Bioinformatics*, **33**, 871–878.

Corrigan-Gibbs,H. *et al.* (2015) Riposte: an anonymous messaging system handling millions of users. In: *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, IEEE Computer Security, pp. 321–338.

Global Alliance for Genomics and Health. (2016) A federated ecosystem for sharing genomic, clinical data. *Science*, **352**, 1278–1280.

Gymrek,M. *et al.* (2013) Identifying personal genomes by surname inference. *Science (New York, N.Y.)*, **339**, 321–324.

Homer,N. *et al.* (2008) Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLOS Genet.*, **4**, e1000167.

Kamm,L. *et al.* (2013) A new way to protect privacy in large-scale genome-wide association studies. *Bioinformatics*, **29**, 886–893.

Philippakis,A.A. *et al.* (2015) The Matchmaker Exchange: a platform for rare disease gene discovery. *Human Mutat.*, **36**, 915–921.

Raisaro,J.L. *et al.* (2017) Addressing Beacon re-identification attacks: quantification and mitigation of privacy risks. *J. Am. Med. Inform. Assoc.*, **24**, 799–805.

Shringarpure,S.S. and Bustamante,C.D. (2015) Privacy risks from genomic data-sharing Beacons. *Am. J. Human Genet.*, **97**, 631–646.

Thenen,N.v. *et al.* (2017) Re-identification of individuals in genomic data-sharing Beacons via allele inference. *bioRxiv*.

Wang,X.S. *et al.* (2015) Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ACM. pp. 492–503.