Privacy-Preserving Whole-Genome Variant Queries

Daniel Demmler, Kay Hamacher, Thomas Schneider, and Sebastian Stammler

Technische Universität Darmstadt, Darmstadt, Germany
{daniel.demmler,kay.hamacher,thomas.schneider,sebastian.stammler}@cysec.de

Abstract. Medical research and treatments rely increasingly on genomic data. Queries on so-called variants are of high importance in, e.g., biomarker identification and general disease association studies. However, the human genome is a very sensitive piece of information that is worth protecting. By observing queries and responses to classical genomic databases, medical conditions can be inferred. The *Beacon project* is an example of a public genomic querying service, which undermines the privacy of the querier as well as individuals in the database.
By secure outsourcing via secure multi-party computation (SMPC), we enable privacy-preserving genomic database queries that protect sensitive data contained in the queries and their respective responses. At the same time, we allow for multiple genomic databases to combine their datasets to achieve a much larger search space, without revealing the actual databases' contents to third parties. SMPC is generic and allows to apply further processing like aggregation to query results.
We measure the performance of our approach for realistic parameters and achieve convincingly fast runtimes that render our protocol applicable to real-world medical data integration settings. Our prototype implementation can process a private query with 5 genetic variant conditions against a person's exome with 100,000 genomic variants in less than 180 ms online runtime, including additional range and equality checks for auxiliary data.

1 Introduction

Genomic data holds the key to the understanding of many diseases and medical conditions. Some genomic variations in individuals in particular might be crucial in the diagnosis of a disease or a treatment regime. As a first step, a doctor or researcher might want to query the world's pool of sequenced genomes for such a variation in order to identify if it has been encountered and studied before. For this purpose, the *Beacon project* was established by the Global Alliance for Genomics & Health[1] to evaluate the eagerness of institutions around the globe to engage in a distributed variant query service. Participating institutions can be queried for a variation and confirm or deny its existence in their database.

However, this open form of collaboration quickly raises privacy concerns about, e.g., re-identification risk [34]. Thus, it would be highly desirable to secure

[1] http://genomicsandhealth.org/

participants of such a variant querying service, as well as individuals in their genome databases. Furthermore, it can be assumed that many small institutions will not be comfortable in joining the Beacon scheme in its current form, since re-identification risk impacts small databases even more severely.

We address these (genomic) privacy demands by developing a secured form of a federated variant query service – eventually an extension of the Beacon project – in which the count of matches is learned, while hiding which institution contributed to what extent. Our solution can optionally apply a threshold t on the count of matches so as to only release data if there are more than t matches. This can mitigates re-identification risk when querying for rare mutations.

Here, secure multi-party computation (SMPC) gives us the powerful ability to run arbitrary computations on sensitive data, while protecting the privacy of this data. In this work, we use two parties, called proxies, to achieve high efficiency. We rely on SMPC for *secure outsourcing* of genomic data from arbitrarily many sources to two proxies that enable clients to run *private queries* on the data. The proxies are assumed to not collude and therefore learn nothing about the outsourced data or the client's query and its response. We focus on the use-case of privately querying a large, aggregated genome database.

1.1 Our Contributions

In this paper, we provide the following contributions:

- We allow private queries to multi-center genome databases. We hide the query, which elements it accesses, and what elements match the query.
- Our protocol allows to privately aggregate data from multiple data sources. Usually this is prohibited by patient privacy laws, which aim at protecting sensitive medical data. We retain privacy, while at the same time providing a larger search space that leads to more expressive query results.
- Due to the generic nature of our protocol we can perform additional multi-property queries that add only negligible overhead to the database lookup.
- We develop a custom format (Variant Query Format – VQF) for the lossy storage of genomic variants that also allows *similar* variants to match. The widely used Variant Call Format (VCF) can easily be compressed to VQF, providing a bridge to existing genomic variant databases.
- We present a prototypical implementation of our protocol in C++ using the ABY framework [9] and achieve practical runtimes for real-world inputs.

1.2 Deployment Setting

We depict our setting in Figure 1. An arbitrary number of genomic database providers DB_i privately outsource their data to two non-colluding proxies D' and D'', who simply aggregate the received data as one large dataset. Privacy is achieved by using XOR-based secret sharing, as described in more detail in §4. DB_i can extend the database by simply sending new entries D'_i and D''_i to the

proxies at any time. Updates of existing entries require sending only the difference as bitwise XOR to one of the proxies.

A client C who wants to query entries from the aggregated databases sends an XOR-secret-shared query to both proxies. Privacy-preserving computation is made possible by the protocol of Goldreich, Micali and Wigderson [15], which enables efficient computation on secret-shared data. Optionally, we allow results to be t-threshold released, i.e., the client receives a query response only if more than t database entries overall match the query criteria.

As a special case, our setting can also be used by a client that runs private queries on a single genomic database held by a server without involving additional parties. For this, the client runs C and D', and the server runs DB_1 and D''.

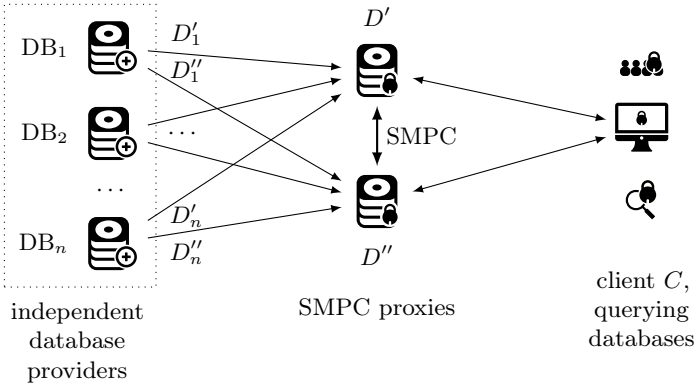

Fig. 1. Deployment setting overview. The plus sign denotes the federation of genomic databases and the lock symbol marks secret-shared data. The client only interacts with the SMPC proxies, which hold XOR secret-shared copies of the genetic variant data.

1.3 Related Work in Genomic Privacy

There exists a long line of work in genomic privacy, starting with [19]. In the subsequent section we provide an overview of recent work related to our solution. We compare the key aspects of related work with our proposal in Table 1.

In [20], count queries on fixed-indexed single-nucleotide polymorphisms (SNPs) databases are performed. They convert the database into an index tree structure and perform an encrypted tree traversal using Paillier's additively homomorphic encryption scheme [32] and Yao's garbled circuits [38] for comparison. The queried SNP indices are leaked and the index tree needs to be built by a single and trusted certified institution that must know all the data in plaintext (coming from possibly multiple data sources as in our setup).

In the field of privacy-preserving genomic testing many approaches securely calculate the edit distance (ED) with protected genetic databases. ED is a way of measuring the closeness of two strings by calculating the minimum number of operations to transform between the two. Most of the time, the Levensthein distance is meant by ED and it is defined as the minimum amount of substitutions, insertions and deletions to transmute between the strings. It is an important measure in genomics to estimate the closeness of two genomes, and finds applications in similar patient querying (SPQ).

The authors of [1] implemented a form of secure count- and ranked similar patient queries, running in seconds. However, their setup is quite different from ours since the query is known in plaintext to each data center, and the output is learned by a central server (CS). The aggregation/sorting is done on the CS and only individual contributions are hidden from the CS and querier.

In [23], the authors developed a privacy-preserving ED algorithm leveraging Yao's garbled circuits. While their technique calculates the exact edit distance, it is computationally infeasible on a whole-genome scale. Their implementation takes several minutes to calculate the ED of just a few hundred-character long strings. [21] improved those results by up to a factor of 29.

In the whole-genome context it is often more sensible to only approximate the ED, taking advantage of the fact that the human genome is mostly preserved ($> 99\%$ of genomic positions) and most variations are simple substitutions. Two important works leveraging these factors are [37] and [2].

The authors of [37] test their distributed query system GenSets over, theoretically, 250 distributed hospitals, each holding 4,000 genome sequences of around 75 million nucleotides each. It took their system 200 minutes to search through one million cancer patients.

In [2], the authors partition the sequences into smaller blocks and then pre-compute the ED within the blocks. Since the human genome shows high preservation, this greatly reduces the number of distinct blocks. E.g., for a realistic data set of 10,000 genome snippets of length about 3,500 from a region of high divergence, after partitioning them into 15-letter blocks, for each group of blocks they observed at most 40 unique blocks, instead of the theoretic maximum of 10,000. Still, they could determine the t best matching sequences against a query sequence with high success.

In [25], an attack by Goodrich [16] on genome matching queries was investigated. They developed a detection technique against such inference attacks employing zero-knowledge proofs to ensure querier honesty.

The authors of [8] developed a novel method for *secure genomic testing with size- and position-hiding private substring matching* (SPH-PSM). In their setup, a testing facility holds a DNA substring (e.g., a marker) and a patient possesses their full genome sequence. The patient sends their homomorphically encrypted genome and public key to the facility, which computes an accumulator applying their substring. The still encrypted accumulator is sent back to the patient, who decrypts it to learn the binary answer — whether the substring is included in

her genome. In the process, the facility doesn't learn anything and the patient doesn't learn the substring or its position in their genome.

The work [40] presents an innovative and efficient approach to outsourced pattern matching employing a new outsourced discrete Fourier transform protocol called OFFT. It solves a similar problem to the aforementioned method [8] and scales logarithmically in the string to be matched.

In [35], a genomic cloud storage and query solution is presented. VCF data is symmetrically encrypted and sent to cloud storage. Using a custom method based on private information retrieval (PIR, see also §2.2), the data owner, or an authorized entity, can query the cloud storage for a specific variant (utilizing a homomorphically encrypted 0–1–array mask). This solution cannot be generalized to multiple patients as the querier would need access to all VCFs' symmetric keys. A generalization of this work [12] allows computations that are similar to ours and offers strong security guarantees. However, in this case, the runtime depends on the size of the response and thus reveals meta information about the query, which we specifically intend to hide in our work. Padding could solve this issue, but would increase the runtime drastically.

The protocol [4, § 4.2] uses Authorized Private Set Intersection [6] to allow for authorized queries of a list of specific SNPs (a SNP profile, e.g., of SNPs relevant for drug selection and dosage): The querier first submits the SNP profile to a certified institution, which sends back an authorization. The querier can then use this authorization to query for his SNP profile in a patient's genome, learning the matching SNPs. The query is hidden from the patient/database. The protocol [33] extends this idea and uses additively homomorphic Elliptic Curve based El-Gamal [10] to calculate a weighted sum over a set of SNPs, where the weights are authorized by a certified institution.

In conclusion, several solutions have addressed the problem of secure genome queries (see Table 1), but most work directly on the sequence instead of called variants and none grant the easy extensibility that our solution provides and at the same time deliver whole-genome scale protection for all included parties.

2 Preliminaries

2.1 Secure Multi-Party Computation (SMPC)

Computation on multiple parties' secret data was first proposed in the 1980's with Yao's seminal garbled circuits protocol [38] and the work of Goldreich, Micali, and Wigderson (GMW) [15]. More formally, for the case of two parties, an SMPC protocol can be described as the evaluation of a function f on the two parties' respective private inputs x and y. The parties learn nothing but what can be inferred from the function's output $z = f(x, y)$. Due to limited computation power, SMPC protocols were at first seen as merely theoretical results. However, a long line of research has lead to significant algorithmic improvements that,

[2] optionally differentially private
[3] arbitrary (aggregation) function f

Table 1. Comparison of features and limitations of related work to our solution.

	[20]	[1]	[23,21]	[37,2]	[8,40]/ [4]/[33]	[35]	[12]	Our work
Variants (V)/Sequences(S)/Both(*)	S	S	S	S	S	V	*	V
Single trusted party eliminated	✗	✗	✓	✓	✓	✓	✓	✓
Query protected	✗	✗	✓	✓	✓	✓	✓	✓
Top similar patient query	✗	✓	n.a.	✓	n.a.	✗	✗	✗
Whole-genome scale	✗	✓	✗	✓	✓	✓	✓	✓
Easy extensibility	✗	(✓)	✗	✗	✗	✗	✗	✓
Output	count	count & similar patients	exact ED	similar patients	weighted average over SNPs / matching SNPs / substr. match? (Y/N)	variant present? (Y/N)	count & sum[2]	(thresh.) count/matches [3]

combined with an increase in computing power in the last decades, has shown that SMPC can indeed be used for practical purposes. Most SMPC protocols rely heavily on oblivious transfer (OT) as a building block. OT extension [22,3] is one of the key results that enables practical performance for SMPC.

In this work we consider the special case of secure two-party computation with security against semi-honest (passive) adversaries in the offline/online model. This means that we divide the protocol into two phases: First, an offline phase that is independent of the private inputs and requires just some upper bound of their size. It can be pre-computed and stored before the actual computation takes place. Second, a very efficient online phase that uses the pre-computed data to compute the function on the parties' private inputs.

While Yao's garbled circuits protocol offers a constant number of communication rounds, it requires the evaluation of symmetric cryptographic operations in both offline and online phase. The GMW protocol allows the preprocessing of so called Beaver multiplication triples [5] that can be pre-computed in the offline phase. The online phase of GMW requires one communication round per layer of dependent AND gates (circuit depth), but involves only very efficient bit operations (XOR) and no symmetric cryptographic operations.

Since our circuits can be optimized for low depth, we decided to implement our protocol using the GMW protocol. There also exist extensions for security against stronger malicious adversaries that can be applied to our protocol [31].

2.2 Related Privacy-Preserving Technologies

Aside from SMPC there are many other privacy-preserving technologies that could potentially achieve similar results as our solution. In this section we briefly provide an overview and motivate why we chose SMPC.

Private information retrieval (PIR) is a technique that allows private queries to a *public* database. This is not applicable in our scenario where the database consists of sensitive genomic data by multiple providers. Furthermore, PIR does not allow for more than the private query, i.e., additional operations like comparisons on the data and threshold checking are not possible.

Homomorphic encryption (HE) is a powerful tool that allows for operations on encrypted values. While somewhat homomorphic encryption is reasonably fast, it only supports a limited set of operations. Fully homomorphic encryption overcomes this limitation. This is an active field of research but implementations still lack the efficiency to be of practical use [30].

Oblivious RAM (ORAM) is another technique that could be used for our use case and was indeed shown to be applicable for, e.g., Bayesian statistics [24] frequently used in bio-informatics [39]. ORAM allows multiple private write accesses to a database, which we do not require, as we need to securely store the genome only once. Thus, our approach is simpler and we expect it to perform better than ORAM-based solutions.

Intel's Software Guard Extensions (SGX) is a recent instruction set extension that allows programs to run in a protected private environment and can be used as an alternative to SMPC [18]. While this is also specifically designed for cloud computing, additional care must be taken to not reveal memory access patterns and timing patterns, which especially affects our use case of privately querying a data set. Observing such access or timing patterns could allow the cloud provider to learn information about the data or the query itself. Learning the query, however, might also reveal critical information about patients, e.g., if a physician queries the database(s) to confirm a particular diagnosis.

Recently the survey [13] categorized the entire field of encrypted or protected search, which offers several additional alternatives to our work. We can imagine a combination of the presented techniques with our ideas as possible future work.

3 Genetic Variant Queries on Distributed Databases

We consider a federation of databases storing genomic variants in our custom Variant Query Format (VQF, see §3.2). They jointly offer a privacy-preserving *Beacon Service*: individual privacy is guaranteed in the sense that it is not learned which dataset from which data-center contributed to which extent to the final count. Optionally, privacy can be strengthened by enforcing a threshold criterion on the query: the actual count will only be returned if it is larger than a predefined threshold parameter t.

3.1 Beacon network and potential extensions

The term *Beacon* stems from the Beacon Network Project, which is "a global search engine for genetic mutations". It was instantiated by the Global Alliance for Genomics & Health to "test the willingness of international sites to share genetic data". The joint service answers queries of the form "Do you have any genomes with an 'A' at position 100,735 on chromosome 3?" and participants each give a simple *Yes/No* answer.

Privacy Concerns It has been shown in [34] that in its original form, Beacon queries are susceptible to re-identification attacks. The authors showed that with 5,000 queries, a person can be re-identified from a Beacon holding 1,000 genomes. Our proposed framework lowers this risk twofold: Firstly, the querier doesn't learn immediately which database contributes to the count. Only in a follow-up consultation can the databases and querier reveal a match should they mutually agree to do so. And secondly, because of an optional t-threshold check in the final step, it is harder for the querier to craft individual-identifying queries. While recent changes to the project's architecture address some privacy concerns by access-control checks[4], those solutions cannot withstand a malicious man-in-the-middle or potential exploits of access-control vulnerabilities. That is, the project doesn't offer privacy by design since queries are sent in clear to the beacons and central web-interface, which also learns all beacons' answers during aggregation of the results.

3.2 Genomic Variant Representation Format

Variant Call Format (VCF) The most popular format to store genomic variants is in the Variant Call Format (VCF) [7], as used by the 1,000 Genomes Project and the Broad institute. It stores the results of a process called *variant calling* on aligned reads (usually stored in *SAM/BAM* files [28] or more recently in the *CRAM* format [11]). It describes in precise detail genetic mutations versus a reference genome.

However, for the application of genetic querying, the VCF is too precise in the sense that a lot of auxiliary data is stored per variation and exact matches are unlikely, especially for complex structural variations. We therefore developed a machine friendly and simplistic format for storing genetic variants in the context of variant querying that also makes it possible for *similar* variations to match.

Variant Query Format (VQF) We store a genome's set of variation against a common reference genome in the following custom format which we will call Variant Query Format (VQF). Variations are stored in a fixed-size dictionary (possibly with multiple entries for the same key), where the key is a variant's position in the reference genome and the value encodes the variation. We set the

[4] Beacon FAQs, 2)

key size κ to 32 bit, which suffices to address every locus in any human reference genome, which has about 3.2 billion (haploid) loci[5].

For the dictionary's value we choose a size ψ of 16 bit. Mapping any genomic variant information into 16 bit is not straightforward and must incur some information loss that we can bound from above (see below). Table 2 lists the proposed mappings from genetic variations to a 16 bit value in our dictionary. The total number of attainable values is $4 + 4s + 2 \cdot 4^{s_{\text{ins}}} + 1 = 32{,}837$, which fit into the 16 bit value space.

Table 2. Details of the variant compression in VQF.

Variant	**Stored Information**	**#Values**
SNP/SNV	We store the alternative nucleotide.	4
Deletion/CNV /Inversion	Store two entries: the up and down rounded logarithm to the base $b = 2$ of deletion/CNV[6]/inversion length, up to log-length s. Also store one frameshift bit for deletions.	$s = 16$
Insertion	Save up to $s_{\text{ins}} = 7$ inserted nucleotides and a frameshift bit.	$2 \cdot 4^{s_{\text{ins}}=7}$
Other	Only flag as *other* variation. This flag captures all other more complex variations, like rearrangements with breakends etc.	1

Note that the chosen logarithm base $b = 2$ for deletion/CNV/inversion events is just exemplary and could be adjusted differently, even per event, without changing the number of possible values. The maximum length logarithm s to store could also be different per event. Storing the up *and* down rounded logarithms for those events increases the chances of a match for closely related variations. E.g., if, at a given locus, one genome has a deletion of length 20 and another of length 60, those would be mapped to two entries each, $\{16, 32\}$ and $\{32, 64\}$, so a query for 32 would match in both genomes. For insertions and deletions (InDels) we also store a frameshift bit, i.e., whether its length is divisible by 3. Substitutions are just stored as combined InDels, if longer than a SNP/SNV.

Also note that for CNVs, we do not store the motif, no matter if a short CNV ($\sim$ tandem repeat) or long CNV was called, possibly resulting in false positives. For diploid variations, we create two entries with the same key (same locus) and each value encoding one of the two variations coming from either parent.

The compression of variant information might seem strong, but the information loss mainly concerns complex structural variations, where an exact query match is very unlikely in the overwhelming majority of application scenarios. We deem it sensible to rather have a small number of "false positives", but still closely related matches. When querying for similar patients, it is actually desirable to

[5] $\log_2(3.2 \cdot 10^9) \approx 31.6 < 32$

[6] For CNVs like tandem repeats, we record the absolute number of repeats, not the relative change

match less specific conditions, as is the case for copy number variations. E.g., it doesn't matter if a short tandem repeat gained 20 or 30 motifs in length. Only the magnitude by which it increased or decreased is important when searching for similar genomes.

Note that a mapping (which could also be described as a compression) from the very detailed VCF to our VQF encoding scheme is straightforward.

Reference Genome Since variations are indexed by a reference genome, it is important that all genomes are called against the same reference when querying several individuals' variations. This is assumed for all databases in our setup.

Alternative Variant Format Note that a different variant compression scheme akin to the hashing method found in [35] could be used. For a single VCF row, one could hash the `ID, REF` and `ALT` entry and some important fields from the `INFO` column, and project it to the 16 bit (or larger) value space. Ignoring collisions, this scheme could also serve as a variant compression method with reasonable low probability of false positives. However, since this method doesn't incorporate domain specific knowledge like our VQF, *similar* variations wouldn't have a chance to match.

Typical sizes There are typically two ways to analyze a person's genetic variation. The first option is to store the full genetic variation, which leads to around $N = 3$ million entries. The second, and more viable, option is to only store variation from coding regions, i.e., a person's exome. While a person's exome makes up only about 1% of their genome, this region is the most important in the context of research and disease testing. As such, exome sequencing is a common quicker and cheaper alternative to full-genome sequencing. A person's exome has about $N = 100{,}000$ variations. In our database model, a person's full genetic variation is mapped to N entries of size $\kappa + \psi = 48$ bit.

Auxiliary data Besides information on genetic variation, the databases can also hold auxiliary data like sex, age, weight and health data like blood pressure or disease indicators. For those fields, our protocol allows for range or threshold conditions in the queries, where desired.

3.3 Queries

The queries that we support are of the form

```
SELECT f(*) FROM Variants
WHERE ((locus_1, var_1), ..., (locus_m, var_m)) IN Genome
   AND cancertype = X AND ... AND age_min ≤ age ≤ age_max ...
```

when expressed in SQL for illustration purposes. Here, m is the number of variant equalities to check and the remaining auxiliary queries can be more versatile and include ranges, besides equalities. For most of our benchmarks, we choose $m = 5$.

The querier specifies the values for the highlighted parts of the query. These values are secret-shared and remain private, such that the proxies do not learn them. The *structure* of `f` cannot be chosen by the querier and is fixed a priori. However, since we use the ABY SMPC framework [9], the function `f` can generally be an arbitrary (aggregation) function on the bit string of matching genomes, like the identity (simple output of matches) or the count (Hamming weight).

Technically, any query prompts the two proxies to generate a secret-shared bit string representing the matching genomes. As can later be seen in §6.1, this task causes the bulk computation and communication cost. Next, the proxies apply function `f` to this bit string and output it to the querier (or any other predefined party).

Query Scenario For our experiments, we choose a scenario in which the querier receives the count of matches, together with a random query ID. Each database with at least one match receives the corresponding sub-bit mask of the genomes only in their own database together with the same query ID. This way, the database doesn't learn the query which matched their genomes but can contact the querier for a follow-up discussion of the matched patients.

Optionally, we can substitute the simple count by a t-threshold count, which returns the count only if it is larger than t. This would mitigate the re-identification risk as presented in [34]. However, since Beacon queries are often used to query for rare mutations, this extension brings its own problems. Our solution can easily realize both options and due to the generic nature of SMPC it can also be adapted for additional requirements.

4 Our Protocol for Private Genome Variant Queries

Our protocol ensures the privacy of both, the query to a genomic database and its response. At the same time, the entire database is hidden from the two proxies. This matches closely the cloud computing paradigm where computation and data storage is outsourced to a powerful set of machines that are maintained by an external party. In Figure 1 on page 3, we depict our setting where multiple databases $\mathrm{DB}_1, \ldots, \mathrm{DB}_n$ outsource their data to two proxies D' and D'' that run the SMPC protocol and are assumed to not collude. A client C queries the combined data from all databases for the counts of entries that a) match the query criteria and optionally b) fulfill a pre-defined t-threshold level.

We achieve privacy by using XOR-based secret-sharing between the SMPC parties. More precisely, for every plaintext bit p, we choose a random masking bit r. We send r to D' and $s = p \oplus r$ to D''. The values r and s are called *shares* of the plaintext value p. For bit strings of length ℓ we apply this technique ℓ times in parallel. To further improve communication, we could send to D' a single seed for a pseudo-random generator instead of the values r.

We use the protocol of Goldreich, Micali and Wigderson (GMW) [15] to *privately* evaluate a Boolean circuit that corresponds to our functionality, i.e., querying a genome database. GMW operates directly on XOR-secret-shared values. We use the GMW protocol in the offline/online computation model, i.e., an *offline phase* that is preprocessed at any time before the actual private inputs are known. The data from the offline phase is then used in the efficient *online phase* to compute the function on the private data.

4.1 Protocol Description

In this section, we describe the phases of our overall protocol and depict it graphically in Fig. 2. Optionally, before the protocol, the database providers and proxies agree on a privacy threshold t that defines the minimum amount of matching records that a query response must contain. If a query matches $\leq t$ records, the query response will be the empty set $\emptyset$, i.e., the same as if no record matched the query.

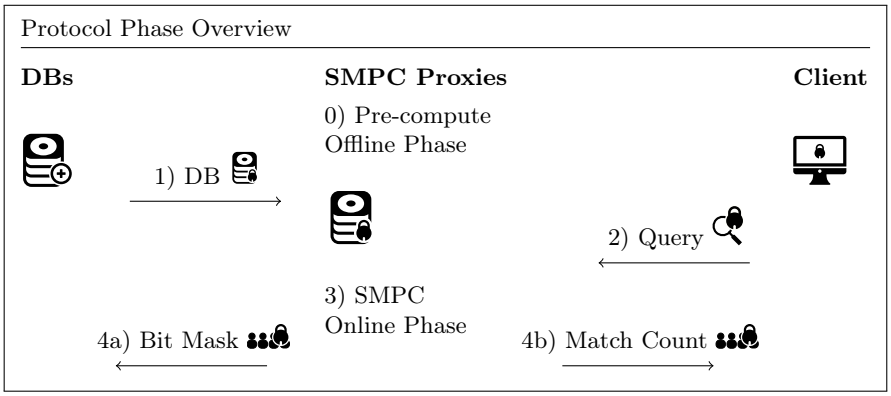

Fig. 2. Protocol phases. Note that all communication with the SMPC proxies happens secret-shared, such that the proxies never gain access to any plaintext.

Phase 0) SMPC Offline Phase. At any point before phase 3, D' and D'' pre-compute the SMPC offline phase, which is independent of the actual inputs from other parties.

Phase 1) Database Outsourcing. Each database provider DB_i is assumed to hold their data in the VQF format (see §3.2). The provider then generates a random mask of the size of its database and sends the random mask to proxy D' and the XOR of the mask and its database to D''. The proxies concatenate the shares they receive from all database providers and keep track of the mapping of shares to DB providers. Note that this phase needs to be performed only once. The secret-shared database can be queried multiple times. Databases need only send new entries or updates to existing entries if they have changed.

Phase 2) Client Query. Client C secret-shares its query between D' and D'' and sends in plain text the type of auxiliary queries it wants to run. Note that we only reveal the *operation* of the auxiliary queries and not the values that are compared with the datasets.

Phase 3) SMPC Online Phase. The proxies D' and D'' run the SMPC protocol on the databases and the query they received. Due to memory constraints, the client query is run on a single patient dataset, consisting of up to 3 million variants at a time. Multiple patients' datasets can be evaluated in separate SMPC instances, which can be run sequentially or in parallel. Given enough hardware, this step can be ideally parallelized. The individual outcomes (match / no match) are stored in a bit mask, still secret-shared and thus unknown to the proxies.

In our scenario, after the query was run against all patients' datasets, the resulting bit mask is processed by a final circuit that counts the number of matches, i.e., the Hamming weight, optionally compared to a threshold.

Phase 4) Output reconstruction. Both proxies D' and D'' hold the output of the computation in secret-shared form and send their output shares to C, who computes the XOR and thereby receives the plaintext output. If the optional threshold t privacy is enforced, C will only receive the count if there are more than t matches. Each database also receives the two shares of its sub-bit mask of matches, together with a random query ID for potential follow-up. Reconstruction will reveal the matching genomes, if any, or an all-zero bit string otherwise.

4.2 Security Considerations

We discuss the security of our scheme and the attacker model in this section.

The goal of our protocol is to ensure the privacy of the queries clients send to the service, as well as the corresponding responses they receive. At the same time, our protocol ensures the privacy of the genomic databases that outsource their data to our service. We achieve these properties by directly relying on the proven security of the GMW protocol [15], which allows to privately evaluate any computable function that is represented as a Boolean circuit. GMW uses XOR-based secret-sharing as underlying primitive, which protects private values. In its original form secret sharing offers information theoretic security since plaintexts are masked with randomness of the same length. We use a PRG to expand a short seed to the length of the plaintext, thus reducing information theoretic security to computational security of the PRG. In our implementation we rely on AES as PRG.

We make the assumption that adversaries behave semi-honestly and corrupt at most one of the two proxies at the same time. The latter corresponds to a non-collusion assumption between the two proxies. Given the semi-honest non-collusion assumptions we can proof that our protocol is secure, since the transcript of every party can be simulated given their respective inputs and outputs. We consider the following cases of semi-honest corrupt parties, malicious clients or external adversaries:

Corrupt semi-honest client/corrupt database: A corrupt client or corrupt database with input query Q and response R can be simulated by a simulator playing the role of the two proxies, running the GMW protocol. This implies that a corrupt client or database learns no additional information from the protocol execution.

Corrupt semi-honest proxy: For a set of databases and a single client query each separate proxy's view consists of a share of the client's query and a share of each database. In all cases XOR-secret-sharing is used, which makes the corresponding strings appear uniformly distributed and leaks no information about the content. Each proxy's view also contains the other proxy's inputs for the GMW protocol. These are proven not to leak information about private inputs in [14].

Malicious clients: Since the proxies and the client interactively agree on the query structure beforehand, a malicious client can only influence values within the boundaries of the query, i.e., the client can only send an input bit string of the pre-defined length. Malicious clients can send bogus queries with the correct length, which will be processed by the proxies. This leads to corrupt outputs that leak no more information than valid queries. The number of queries a client can send can be controlled by using rate-limiting.

External adversaries: Data in transit is protected from external adversaries by using state-of-the-art secure channels, e.g., TLS, to ensure confidentiality, integrity and authenticity between all communicating parties.

5 Implementation

In this section we describe our implementation decisions and the software design of our application, as well as its limitations.

5.1 The ABY Framework

We implemented our protocols within the ABY framework for secure two-party computation [9]. It provides efficient C++ implementations of recent secure two-party computation protocols and includes many recent optimizations. ABY offers abstractions of the underlying protocols and building blocks and is thus a viable option for implementing SMPC applications. More specifically, we rely on the included implementation of the GMW protocol, which is based on XOR secret-sharing and thus is well-suited for our outsourcing scenario.

5.2 Boolean Circuit Design

The GMW protocol operates on Boolean circuits. Our protocol contains two circuit designs that we optimized for a low multiplicative depth in order to minimize the number of communication rounds between the proxies. The biggest circuit is the query circuit that checks if a user query matches the genome of a patient. It consists of an equality gate that compares every patient variant with a query variant. On the circuit level, this is done in parallel on a person's entire variants and all query variants. From this we get 1 bit per patient variant, which is fed into an OR tree that returns 1 if at any position the query matched with a patient's variant. For all query variants the results from the OR trees are fed into an AND tree that returns 1 if all query variants are in the patient's variants. These gate trees are the reason for the logarithmic circuit depth. The circuit also checks for auxiliary patient properties, which are implemented as single equality or comparison gates.

The circuit's final result is a single bit that indicates if all query variants are in the person's variant dataset and if all auxiliary queries matched. The circuit output for each patient record is stored (still secret-shared). As soon as all patient queries have been run, the previously stored result shares are fed into a smaller (threshold-)counting circuit (that optionally checks that the count is larger than the threshold t). It consists of a Hamming weight circuit that counts the number of 1-bits, and a comparison gate that controls if a multiplexer gate outputs the string of matches or an all-zero bitstring. Its output is a bit string with one bit per patient. Bits are set to one at the indices where the query matched. If it contains more than t ones, it is output, otherwise an all-zero bit string is output, in case threshold t counting is applied.

5.3 Limitations of our Approach

While we only ran queries, where *all* conditions must be met, i.e., all conditions are connected with `AND` ($\wedge$) expressions, the ABY framework would easily allow for more complex or nested queries, such as $A \vee \big((B \wedge C) \vee D\big)$. The performance impact would be minimal and would only depend on the size of the formula. Universal circuits [36,27,26,29,17] could be used to also hide the structure of the formula.

The translation of variants into 16 bit strings (see §3.2) certainly is a limitation and cannot reflect the full spectrum of possible variations. However, as elaborated before, we used this compression as a way to match similar variations while only using strict equality queries.

6 SMPC Benchmarks

In this section we provide benchmark results of our implementation. We performed runtime benchmarks of our SMPC implementation on two identical desktop computers with 16 GiB RAM and a 3.6 GHz Intel Core i7-4790 CPU, connected

via local Gbps network. We ran a 64-bit Debian Jessie with Linux kernel 3.16 and used `gcc` v4.9.2 to compile our code. For all measurements we instantiate the parameters to achieve a symmetric security level of 128 bits. All results are averaged over 25 iterations. The provided communication numbers are the sum of sent and received data of one SMPC proxy. In the following section we use the term offline phase to refer to step 0) from Figure 2, while online phase refers to step 3). We did not measure the time for steps 1), 2), and 4), i.e., the conversion and transmission of databases and query/response, as these are simple and efficient plaintext operations and data transmissions that scale linearly with size and available bandwidth.

6.1 Variant Query Performance

In this section we measure the performance of running a query with a certain length against a person's dataset with a given number of variants. As default parameters we use $N = 100{,}000$ variants, query length $= 5$, and $\kappa + \psi = 48$ bit, which corresponds to a query on a person's exome. In Figure 3 we show the runtimes of the offline and the online phase for *varying number of patient variants* with queries of length 5. In Figure 4 we fix the number of a patient's variants to $N = 100{,}000$ entries and show how the runtimes scale for *varying query length*. Figure 5 shows how a *varying entry bitlen* ($\kappa+\psi$) influences the protocol runtimes. We provide the corresponding detailed numbers in Tables 3, 4, and 5.

In all cases the offline and online runtime and circuit size (number of AND gates) increase linearly with the database size, query length, and entry bitlength. The circuit depth, i.e., the number of communication rounds between the two proxies, scales only with the logarithm of the input sizes.

Regarding the auxiliary queries, we fixed five different equality and range queries, which didn't have any noticeable performance impact. They are thus omitted in the following discussions.

For querying a patient's exome variants ($N = 100{,}000$), assuming a query of length 5 and our proposed entry format with key length $\kappa = 32$ bit, and value length $\psi = 16$ bit, we achieve an offline runtime of 3.4 s and an online runtime of 178 ms. In this case we need to transfer 733 MiB in the offline phase and the online phase requires 28 communication rounds with a total transmission of 11 MiB. The circuit for these parameters consists of 24 million AND gates. Using the same parameters to query a patient's full genome ($N = 3{,}000{,}000$) requires an offline and online runtime of 99.5 s and 4.6 s, respectively.

Our performance is comparable to the single-variant query in Sousa et al. [35], which takes 2.4 – 4.3 s online runtime for 5 million variants. However, their query is not extensible and can only answer whether a single variant is present, without the option for further privacy-preserving aggregation.

6.2 Count Performance

The circuit (`f` in §3.3) that determines the total number of matches and compares this to the privacy threshold t is very small. For processing the results of 100,000

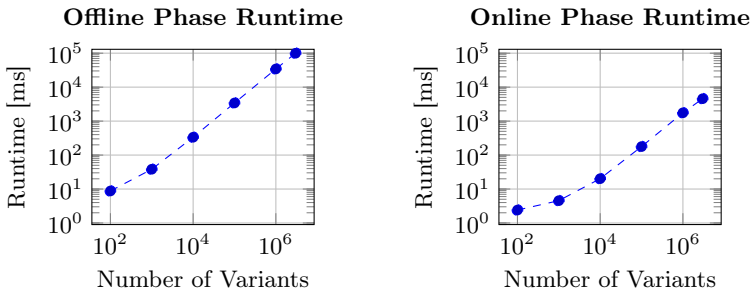

Fig. 3. Offline and online runtime in ms for *varying number of variants per patient* and fixed key length $\kappa = 32$ bit, value length $\psi = 16$ bit, and a query with 5 components.

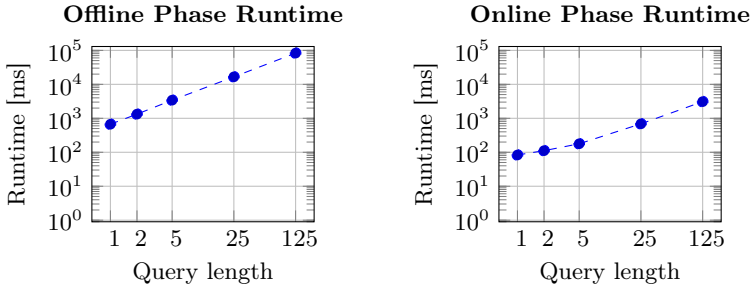

Fig. 4. Offline and online runtime in ms for *varying query length* and fixed key length $\kappa = 32$ bit, value length $\psi = 16$ bit, and variant count of $N = 100{,}000$ entries.

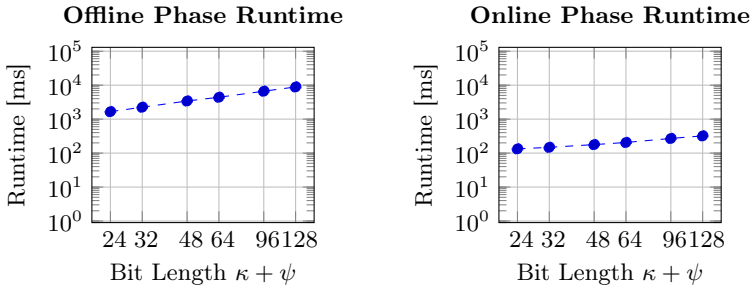

Fig. 5. Offline and online runtime in ms for *varying total element size* $\kappa + \psi$ at a fixed variant count of $N = 100{,}000$ entries and query length of 5.

patient records, the t-threshold count circuit consists of 100,040 AND gates and has a depth of 22. It requires 75 ms (55 ms) runtime and 439 KiB (2,124 KiB) communication in the online (offline) phase. Communication, runtime, and circuit size scale linearly with the input size, while circuit depth grows logarithmically. Since these numbers are negligible for the total runtime, we omit a more detailed analysis at this point.

Table 3. Benchmark results and circuit properties for *varying variant count* at fixed key length $\kappa = 32$ bit, value length $\psi = 16$ bit, and query length 5.

Variants N	Query Length	$\kappa + \psi$ [bit]	#AND Gates	Circuit Depth	Offline Phase [ms]	Offline Phase [MiB]	Online Phase [ms]	Online Phase [MiB]
100	5	48	$2.4 \cdot 10^4$	18	9	1	2.4	0
1,000	5	48	$2.4 \cdot 10^5$	21	38	7	4.5	0
10,000	5	48	$2.4 \cdot 10^6$	25	335	73	20.2	1
100,000	5	48	$2.4 \cdot 10^7$	28	3,420	733	177.9	11
1,000,000	5	48	$2.4 \cdot 10^8$	31	34,297	7,325	1,756.2	114
3,000,000	5	48	$7.2 \cdot 10^8$	33	99,507	21,975	4,567.7	343

Table 4. Benchmark results and circuit properties for *varying query length* at fixed key length $\kappa = 32$ bit, value length $\psi = 16$ bit, and variant count $N = 100{,}000$.

Variants N	Query Length	$\kappa + \psi$ [bit]	#AND Gates	Circuit Depth	Offline Phase [ms]	Offline Phase [MiB]	Online Phase [ms]	Online Phase [MiB]
100,000	1	48	$4.8 \cdot 10^6$	27	669	146	83.2	2
100,000	2	48	$9.6 \cdot 10^6$	27	1,327	293	111.5	5
100,000	5	48	$2.4 \cdot 10^7$	28	3,420	733	177.9	11
100,000	25	48	$1.2 \cdot 10^8$	29	16,629	3,662	687.5	57
100,000	125	48	$6.0 \cdot 10^8$	32	83,141	18,312	3,096.9	286

Table 5. Benchmark results and circuit properties for *varying total element size* $\kappa+\psi$ at fixed query length of 5 and variant count $N = 100{,}000$.

Variants N	Query Length	$\kappa + \psi$ [bit]	#AND Gates	Circuit Depth	Offline Phase [ms]	Offline Phase [MiB]	Online Phase [ms]	Online Phase [MiB]
100,000	5	24	$1.2 \cdot 10^7$	27	1,662	366	133.2	6
100,000	5	32	$1.6 \cdot 10^7$	27	2,241	488	147.9	8
100,000	5	48	$2.4 \cdot 10^7$	28	3,420	733	177.9	11
100,000	5	64	$3.2 \cdot 10^7$	28	4,409	977	205.6	15
100,000	5	96	$4.8 \cdot 10^7$	29	6,640	1,465	269.6	23
100,000	5	128	$6.4 \cdot 10^7$	29	8,860	1,953	321.3	31

6.3 Conclusions from the Benchmarks

We consider our solution practical for typical private genome queries. While the computation of responses for databases with thousands of patients still do not answer instantaneously, we can run these private queries over night or increase throughput by running them in parallel on dedicated hardware and faster networks. As we can see from our performance evaluation, both runtime and communication complexity scale linearly with the input size. Our circuit constructions are optimized for use with the GMW protocol and their depth grows only logarithmically with increasing input size. Network traffic and memory requirements are well within the limits of modern hardware.

The generation of the bit string representing the matching genomes takes the bulk computation and communication cost, while the cost for evaluating the auxiliary conditions, aggregation, and threshold comparison is negligible. Therefore, possibly complex and versatile post-processing functions can be applied to the matches thanks to the use of generic SMPC techniques. This ability sets our system apart from related works in this field.

7 Summary

In this work we have presented a new, privacy-preserving protocol to allow multi-center variant queries on genomic databases. The achieved performance renders this approach applicable in real-world scenarios with some dozens of centers. Full genome studies are supported based on the state-of-the-art VCF format. Our approach leverages the custom Variant Query Format, which can be built from existing VCF data. Variants have to be called against a pre-defined reference genome to be agreed upon before setting up the federated data analysis platform. An interesting and demanding research question immediately arises: how would one use our (and many previously developed) genomic privacy techniques on genomic data while facing the problem of different reference genomes? The simple answer is to regenerate the genomic data against the new reference genome via the same pipeline. But this approach might not always be feasible or possible, if the pipeline is not fully automated or recorded. Note that this applies to almost all previous work where genomic data from different patients is compared, as is the case in the Beacon project. This "transcription" to other reference genomes is beyond the scope of the present work but will be addressed in the future.

Another open problem is how to effectively mitigate the re-identification risk as presented in [34], while still handling queries of rare mutations in a sensible way. We described two query types which our framework supports: a regular count, which is susceptible to the aforementioned re-identification risk, even though to a lesser extent, since the querier doesn't learn in which databases the matches occurred, and a threshold-t-count, which only outputs the count if it is larger than t. While the latter method provides more privacy, it may render the system unusable for very rare mutations.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback on our paper. This work has been supported by the German Federal Ministry of Education and Research (BMBF) and by the Hessian State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP (www.crisp-da.de), by the DFG as part of project E4 within the CRC 1119 CROSSING, as well as by collaborations within the BMBF-funded HiGHmed consortium.

References

1. Al Aziz, M.M., Hasan, M.Z., Mohammed, N., Alhadidi, D.: Secure and efficient multiparty computation on genomic data. In: 20. International Database Engineering & Applications Symposium (IDEAS'16). pp. 278–283. ACM (2016)
2. Asharov, G., Halevi, S., Lindell, Y., Rabin, T.: Privacy-preserving search of similar patients in genomic data. Cryptology ePrint Archive, Report 2017/144 (2017), https://eprint.iacr.org/2017/144
3. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: 20. ACM Conference on Computer and Communications Security (CCS'13). pp. 535–548. ACM (2013)
4. Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering GATTACA: Efficient and Secure Testing of Fully-sequenced Human Genomes. In: 18. ACM Conference on Computer and Communications Security (CCS'11). pp. 691–702. ACM (2011)
5. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: 11. Advances in Cryptology – CRYPTO'91. LNCS, vol. 576, pp. 420–432. Springer (1991)
6. Cristofaro, E.D., Tsudik, G.: Practical Private Set Intersection Protocols with Linear Complexity. In: Financial Cryptography and Data Security. pp. 143–159. LNCS, Springer (Jan 2010)
7. Danecek, P., Auton, A., Abecasis, G., Albers, C.A., Banks, E., DePristo, M.A., Handsaker, R.E., Lunter, G., Marth, G.T., Sherry, S.T., McVean, G., Durbin, R.: The variant call format and VCFtools. Bioinformatics 27(15), 2156–2158 (2011)
8. De Cristofaro, E., Faber, S., Tsudik, G.: Secure genomic testing with size- and position-hiding private substring matching. In: 12. ACM Workshop on Privacy in the Electronic Society (WPES'13). pp. 107–118. ACM (2013)
9. Demmler, D., Schneider, T., Zohner, M.: ABY – a framework for efficient mixed-protocol secure two-party computation. In: 22. Network and Distributed System Security Symposium (NDSS'15). The Internet Society (2015)
10. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
11. Fritz, M.H.Y., Leinonen, R., Cochrane, G., Birney, E.: Efficient storage of high throughput DNA sequencing data using reference-based compression. Genome Research 21(5), 734–740 (May 2011)
12. Froelicher, D., Egger, P., Sousa, J.S., Raisaro, J.L., Huang, Z., Mouchet, C., Ford, B., Hubaux, J.P.: UnLynx: A decentralized system for privacy-conscious data sharing. In: Proceedings on Privacy Enhancing Technologies. vol. 4, pp. 152–170 (2017)

13. Fuller, B., Varia, M., Yerukhimovich, A., Shen, E., Hamlin, A., Gadepally, V., Shay, R., Mitchell, J.D., Cunningham, R.K.: Sok: Cryptographically protected database search. In: 38. IEEE Symposium on Security and Privacy (S&P'17). pp. 172–191 (2017)
14. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
15. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: 19. ACM Conference on Theory of Computing (STOC'87). pp. 218–229. ACM (1987)
16. Goodrich, M.T.: The mastermind attack on genomic data. In: 30. IEEE Symposium on Security and Privacy (S&P'09). pp. 204–218. IEEE (2009)
17. Günther, D., Kiss, Á., Schneider, T.: More efficient universal circuit constructions. In: 23. Advances in Cryptology – ASIACRYPT 2017. LNCS, Springer (2017), http://thomaschneider.de/papers/GKS17.pdf, to appear. Full version: http://ia.cr/2017/798
18. Gupta, D., Mood, B., Feigenbaum, J., Butler, K., Traynor, P.: Using Intel Software Guard Extensions for efficient two-party secure function evaluation. In: 4. Workshop on Encrypted Computing and Applied Homomorphic Cryptography (WAHC'16). LNCS, vol. 9604, pp. 302–318. Springer (2016)
19. Hamacher, K., Hubaux, J.P., Tsudik, G.: Genomic Privacy (Dagstuhl Seminar 13412). Dagstuhl Reports 3(10), 25–35 (2014), http://drops.dagstuhl.de/opus/volltexte/2014/4426
20. Hasan, M.Z., Mahdi, M.S.R., Mohammed, N.: Secure count query on encrypted genomic data. In: 3. International Workshop on Genome Privacy and Security (GenoPri'16) (2017), https://arxiv.org/abs/1703.01534
21. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: 20. USENIX Security Symposium (USENIX Security'11). USENIX (2011)
22. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: 23. Advances in Cryptology – CRYPTO'03. LNCS, vol. 2729, pp. 145–161. Springer (2003)
23. Jha, S., Kruger, L., Shmatikov, V.: Towards practical privacy for genomic computation. In: 29. IEEE Symposium on Security and Privacy (S&P'08). pp. 216–230. IEEE (2008)
24. Karvelas, N., Peter, A., Katzenbeisser, S., Tews, E., Hamacher, K.: Privacy-preserving whole genome sequence processing through proxy-aided ORAM. In: 13. Workshop on Privacy in the Electronic Society (WPES'14). pp. 1–10. ACM (2014)
25. Kerschbaum, F., Beck, M., Schönfeld, D.: Inference control for privacy-preserving genome matching. CoRR abs/1405.0205 (2014), https://arxiv.org/abs/1405.0205
26. Kiss, Á., Schneider, T.: Valiant's universal circuit is practical. In: 35. Advances in Cryptology – EUROCRYPT'16. LNCS, vol. 9665, pp. 699–728. Springer (2016)
27. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: 12. International Conference on Financial Cryptography and Data Security (FC'08). LNCS, vol. 5143, pp. 83–97. Springer (2008), http://thomaschneider.de/papers/KS08UC.pdf, code: http://encrypto.de/code/FairplayPF
28. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R.: The Sequence Alignment/Map format and SAMtools. Bioinformatics 25(16), 2078–2079 (2009)

29. Lipmaa, H., Mohassel, P., Sadeghian, S.S.: Valiant's universal circuit: Improvements, implementation, and applications. IACR Cryptology ePrint Archive 2016(17) (2016), http://ia.cr/2016/017
30. Naehrig, M., Lauter, K.E., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: 3. ACM Cloud Computing Security Workshop (CCSW'11). pp. 113–124. ACM (2011)
31. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: 32. Advances in Cryptology – CRYPTO'12. LNCS, vol. 7417, pp. 681–700. Springer (2012)
32. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: 17. Advances in Cryptology – EUROCRYPT'99. LNCS, vol. 1592, pp. 223–238. Springer (1999)
33. Perillo, A.M., Cristofaro, E.D.: PAPEETE: Private, Authorized, and Fast Personal Genomic Testing. Tech. Rep. 770 (2017), https://ia.cr/2017/770
34. Shringarpure, S., Bustamante, C.: Privacy risks from genomic data-sharing beacons. The American Journal of Human Genetics 97(5), 631–646 (2015)
35. Sousa, J.S., Lefebvre, C., Huang, Z., Raisaro, J.L., Aguilar-Melchor, C., Killijian, M.O., Hubaux, J.P.: Efficient and secure outsourcing of genomic data storage. BMC Medical Genomics 10(2), 46 (Jul 2017)
36. Valiant, L.G.: Universal circuits (preliminary report). In: 8. ACM Symposium on Theory of Computing (STOC'76). pp. 196–203. ACM (1976)
37. Wang, X.S., Huang, Y., Zhao, Y., Tang, H., Wang, X., Bu, D.: Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In: 22. ACM Conference on Computer and Communications (CCS'15). pp. 492–503. ACM (2015)
38. Yao, A.C.C.: How to generate and exchange secrets. In: 27. Symposium on Foundations of Computer Science (FOCS'86). pp. 162–167. IEEE (1986)
39. You, N., Murillo, G., Su, X., Zeng, X., Xu, J., Ning, K., Zhang, S., Zhu, J., Cui, X.: SNP calling using genotype model selection on high-throughput sequencing data. Bioinformatics 28(5), 643 (2012)
40. Zhou, J., Cao, Z., Dong, X.: PPOPM: more efficient privacy preserving outsourced pattern matching. In: 21. European Symposium on Research in Computer Security (ESORICS'16). LNCS, vol. 9878, pp. 135–153. Springer (2016)