METHOD

Open Access

SQUiD: ultra-secure storage and analysis of genetic data for the advancement of precision medicine

Jacob Blindenbach[1,2,3†], Jiayi Kang[4†], Seungwan Hong[2,3*†], Caline Karam[3], Thomas Lehner[3^] and Gamze Gürsoy[1,2,3*]

†Jacob Blindenbach, Jiayi Kang and Seungwan Hong contributed equally to this work.

^Thomas Lehner is deceased.

*Correspondence: shong@nygenome.org; gamze.gursoy@columbia.edu

[1] Department of Computer Science, Columbia University, New York, USA
[2] Department of Biomedical Informatics, Columbia University, New York, USA
[3] New York Genome Center, New York, USA
[4] COSIC, KU Leuven, Leuven, Belgium

Abstract

Cloud computing allows storing the ever-growing genotype-phenotype datasets crucial for precision medicine. Due to the sensitive nature of this data and varied laws and regulations, additional security measures are needed to ensure data privacy. We develop SQUiD, a *s*ecure *qu*eryable *d*atabase for storing and analyzing genotype-phenotype data. SQUiD allows storage and secure querying of data in a low-security, low-cost public cloud using homomorphic encryption in a multi-client setting. We demonstrate SQUiD's practical usability and scalability using synthetic and UK Biobank data.

Background

Precision medicine aims to tailor medical care to the characteristics of an individual's unique genetic makeup, lifestyle, and environment. This approach has garnered considerable attention worldwide due to its potential to enhance patient outcomes and mitigate healthcare expenses [1]. But several significant obstacles impede the realization of the full potential of precision medicine. One such challenge is the need for extensive and diverse patient genotype-phenotype datasets in order to advance the diagnosis and treatment of future patients [2]. However, this need for large amounts of data is often in conflict with the need to protect patient privacy [3]. This challenge is further complicated by the heterogeneous regulatory landscape governing privacy protection, with varying definitions and practices across different jurisdictions (e.g., General Data Protection Regulation [GDPR] in Europe vs. frameworks in USA) [4, 5]. Furthermore, individual hospitals and institutions maintain their own policies due to the prevalence of health data breaches and privacy attacks.

Genomic data plays a pivotal role in precision medicine research, enabling the customization of medical care based on specific genetic variants, biomarkers, and inherited traits. Thus, there is a surge in data generation, which has challenged the ability of local servers to accommodate the rapid growth of data size and increased computational requirements [6]. Therefore, there is a pressing need and significant push towards cloud computing. However, this exacerbates the concerns about the privacy and prohibitions on use of personal data due to local, global, and/or institutional privacy policies. For example, with the introduction of the GDPR in Europe, the storage of genomic and related data in the cloud has become more stringent with the requirement of appropriate security measures in place such as encryption. Starting from 2023, many states in the US (California, Connecticut, Colorado, Utah, and Virginia) are entering a new GDPR-like privacy era that will have similar requirements about storing genomic and related data in the cloud [7]. Yet, the current state of privacy preservation through laws and institutional policies is fraught with instability and unpredictability, which poses significant challenges to the research community. If the data is kept in the encrypted form in cloud servers, then researchers, who are approved for access, need to download large quantities of data locally and decrypt them to perform analysis, which defeats the purpose of outsourcing the storage to the cloud. This situation creates additional hurdles for scientists, especially when attempting to combine multiple data sets to gain statistical power. Furthermore, it creates significant delays in research and requires large amounts of resources, which impedes the democratization of data access. As a result, advances in medicine will significantly be impacted if new privacy-preserving frameworks that comply with laws and policies are not developed and implemented.

Homomorphic encryption (HE) is one of the cryptographic tools that enables direct computations of functions on encrypted data in the public cloud. Homomorphic encryption has emerged as a useful approach to keep the data secure at rest, at transit, and during analysis. But, this approach also has thus far presented severe bottlenecks in its applicability, scalability, and performance [8, 9]. However, recent advances in algorithm designs and computing power have enabled an increase in the use of homomorphic encryption in genomics. For example, it has been shown that privacy-enhancing genome-wide association studies (GWAS) can be possible [10–13]. It has also been shown that secure genotype imputation is feasible and scalable using homomorphic encryption [14–16]. Homomorphic encryption was also used for genomic variant querying [17], regression analysis for rare disease variants [18], and inference using genetic variants in machine learning applications [19]. These methods have added tremendous algorithmic advances to the field and paved the way for more practical privacy-preserving analysis of genomes. However, their use in genotype-phenotype database settings has been limited. This is primarily attributed to two factors. Firstly, homomorphic encryption relies on public key cryptography, which is designed for client-server scenarios where the client owns the dataset and delegates computation to the cloud. However, in the context of genotype-phenotype databases, the data owner encrypts the data while multiple researchers access and analyze it. Secondly, the computational burden associated with homomorphic encryption makes it infeasible for applications involving large sample sizes. Both the storage size of encrypted data (i.e., ciphertexts) and the computation times for homomorphic encryption are several orders of magnitude greater than those for the original plaintexts [20].

Here, we developed *S*ecure *Qu*eriable *D*atabase (*SQUiD*), a scalable framework designed to store and query genotype-phenotype databases in an ultra-secure cloud-based setting using homomorphic encryption. In our approach, we incorporate several key components: a ciphertext packing storage method to minimize the required storage space for encrypted data, a set of optimizations we developed to reduce query processing time, and an innovative cryptographic primitive (public key-switching) to enable homomorphic encryption for multiple users. We demonstrate that SQUiD is capable of efficiently executing various types of queries on large scale genotype-phenotype datasets, all the while maintaining the encryption of the data in the cloud. Specifically, it can perform tasks such as counting the number of patients in a filtered cohort, computing the minor allele frequency (MAF) of genetic variants in a cohort, calculating polygenic risk scores (PRS) for patients, and generating a cohort of genetically similar patients in remarkably short timeframes. Our findings highlight the potential of SQUiD as a valuable tool for secure, timely, and efficient analysis and interpretation of genetic and phenotypic data. At a time when data breaches are becoming increasingly common in healthcare settings, where data is a commodity, SQUiD provides a key resource to safeguarding patient privacy and enabling data providers to adhere to evolving laws and regulations, while ensuring the democratization of data.

Results

Our conceptual framework allows ultra-secure interactions with encrypted genotype-phenotype databases

Our conceptual framework is focused on solving real-world security challenges encountered in the storage and querying large-scale genotype-phenotype datasets. These challenges involve safeguarding the confidential information contained within such data from third party cloud providers and outside adversaries. Our framework is based on a scenario that involves three parties: the data owner, the researcher(s), and the public cloud. The data owner, who in many cases could be an organization such as the National Institutes of Health (NIH), possesses a vast amount of genotype-phenotype data that can be used for various analyses. Due to the large size of this data and the limited computing power and resources, the data owner encrypts the data and stores it in the public cloud. Their role is limited to authenticating clients who have permission to access the encrypted data, and they do not participate in the computation phase. The client, typically a researcher, seeks to perform computations on the data to obtain results. However, due to the large size of the data and the limitations of their computing power, it is not feasible for them to download, decrypt, and analyze the data locally. The client, therefore, interacts with the encrypted data deposited to the cloud. An overview of each party's role in the architecture of SQUiD is visualized in Fig. 1.

The cloud server does not have knowledge of the information contained in the data because all data stored in the public cloud is encrypted. Computations are performed directly on this encrypted data *without decryption* using homomorphic encryption. In homomorphic encryption [21–23], data, referred to as plaintext, is encrypted into ciphertexts. Addition and multiplication can be performed on ciphertexts such that two ciphertexts can be added or multiplied to produce a new ciphertext, which can be decrypted to the sum or product of the corresponding plaintexts. In our scenario, the

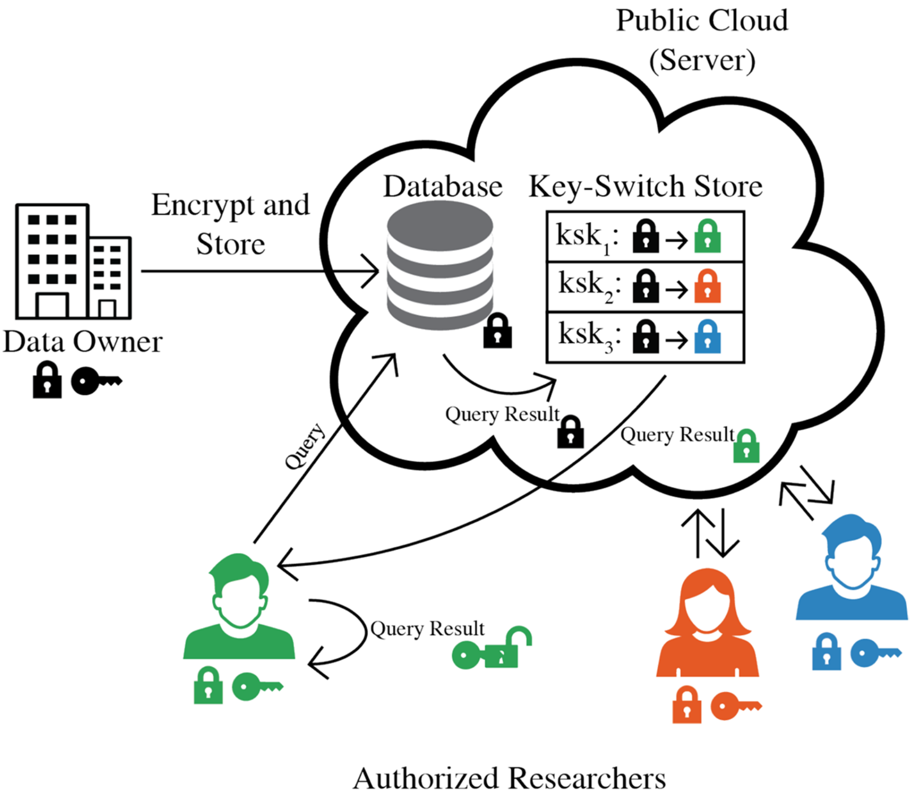
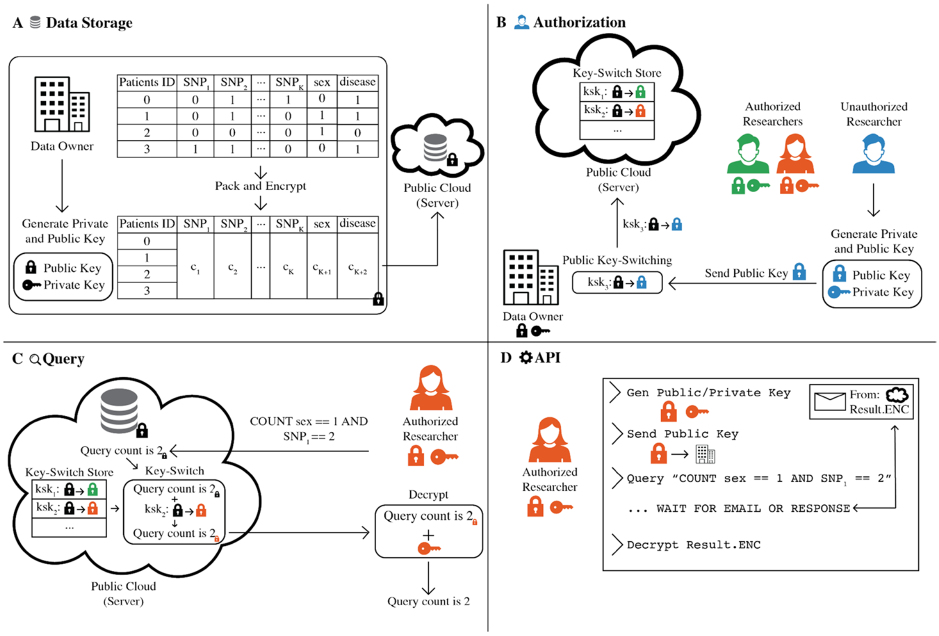


Fig. 1 An architecture overview of SQUiD. Initially, the data owner uploads their encrypted genotype and phenotype data to the public cloud. Within the cloud, only authorized researchers are permitted to securely query the data. Authorization is granted through possession of a key-switching key, which is stored in the key-switching store. When a researcher initiates a query on the databases, the database responds by encrypting the query result under the data owner's public key. Subsequently, the key-switching store transforms this encrypted result to be under the querier's key. The encrypted result is then sent back to the querier, who can decrypt it using their own secret key

complex functions behind our queries are sequences of homomorphic additions and multiplications which the cloud server performs on the encrypted genotype-phenotype data. The outputs of these functions will remain encrypted and are only decrypted after the outputs are sent back to the client. Operations on encrypted data are possible because plaintexts and ciphertexts are expressed as polynomials in HE. The algebraic structures of these polynomials are exploited to enable the computation on encrypted data (see Additional file 1: Supplementary Material for more details). Importantly, retrieving the plaintext polynomial from the ciphertext polynomials without the secret key is extremely difficult. This difficulty is equivalent to the difficulty of solving the ring learning with errors (RLWE) problem, which is known to be computationally hard under suitable parameters [24].

Using homomorphic encryption with appropriate parameters ensures that the sensitive information is protected while computations are being performed and the output is provided to the client in the encrypted form. Figure 2 describes the four key components of our framework: encrypted data storage, access authorization, query capabilities, and the API for user-friendly interactions with the database. Unfortunately, this scenario cannot be realized with traditional homomorphic encryption, which is based on a



Patients ID	SNP_1	SNP_2	···	SNP_K	sex	disease
0	0	1	···	1	0	1
1	0	1	···	0	1	1
2	0	0	···	0	1	0
3	1	1	···	0	0	1

Patients ID	SNP_1	SNP_2	···	SNP_K	sex	disease
0	c_1	c_2	···	c_K	c_{K+1}	c_{K+2}
1						
2						
3						

```
> Gen Public/Private Key
> Send Public Key
> Query "COUNT sex == 1 AND SNP₁ == 2"
  ... WAIT FOR EMAIL OR RESPONSE
> Decrypt Result.ENC
```

Fig. 2 **A** Data storage. The owner vertically packs their data to reduce storage costs, then encrypts their data with a public key, and then uploads the data to the public cloud. **B** Authorization. The onboarding process for a new researcher starts with the creation of their public and private key. The researcher sends their public key to the data owner for authorization. The owner authorizes the researcher by creating a key-switching key to switch the encryption of data to the researcher's key and uploading this key to the key-switching key store in the public cloud. The data owner can revoke a researcher's access by removing the key-switching key from the store. **C** Query. An authorized researcher can submit one of four queries to the public cloud, which performs the necessary operations homomorphically on encrypted data under the data owner's key. The result is then re-encrypted under the researcher's public key and sent back for decryption. **D** API. We created a command-line API for researchers to use SQUiD easily. It generates a public and private key for the researcher, sends the public key to the data owner for authorization, sends queries to the server, and decrypts any encrypted results received via email or through a returned file

two-party (data owner and public cloud) system. In a traditional two-party system, the data owner encrypts the data with their public key and decrypts the results with their private key. Thus, the researcher cannot query and decrypt the results since they do not (and should not) have access to the data owner's private key. To overcome this challenge, we adopted the established concept of the proxy re-encryption system [25] to develop a theoretical realization and practical implementation of it within the framework of homomorphic encryption. This adaptation, which we refer to as the public key-switching technique [26–28], enables secure multi-client queries on encrypted data without the need for exchanging secret keys, specifically addressing the needs of our application in the biomedical domain.

The public key-switching operation serves as a cryptographic primitive facilitating the conversion of ciphertexts encrypted under one secret key to ciphertexts encrypted under a second secret key, without the need to decrypt the ciphertexts to plaintexts or possess access to the second secret key. Precisely, in key-switching, the original ciphertext needs to be homomorphically decrypted within the ciphertext space of the second secret key, which requires a key-switching key *ksk*, i.e., encryption of the first secret key under the second secret key. Since the entire key-switching procedure together with *ksk* occurs within an encrypted space, the underlying messages remain secure without knowledge of either the first or the second secret key. Moreover, in public key-switching, *ksk* is generated by encrypting the first secret key with a public key, which does not require knowledge of the second secret key. In our scenario, this means the data owner can use their secret key for this operation without needing to access any of the clients' secret keys. This capability holds immense value in establishing secure interactions with an encrypted database. For example, in SQUiD, the encrypted database can compute a researcher's query under the encryption of the owner's key, convert the computed result from an encryption under the owner's key to under the public key of the researcher, and send this encrypted result to the researcher. Importantly, this conversion takes place without the need to decrypt the query result or disclose any information about it to the cloud. The researcher can effectuate this conversion by solely providing their public key, thereby circumventing any security risks associated with sharing their secret key.

When granting access to a new researcher, both the data owner and the researcher collaborate to create a public key-switching key, which is subsequently added to the key-switching store in the public cloud (Fig. 2B). The key-switching store offers two significant advantages. Firstly, it allows the data owner to remain offline during any query, as the researcher exclusively interacts with the public cloud where the pre-calculated and stored public key-switching keys reside. Secondly, the data owner retains control over data access by managing the inclusion or exclusion of researchers' public key-switching keys within the store, thus ensuring the ability to govern data access. We show that performance overhead from generating a key-switching key and key-switching a ciphertext under the encryption of one key to another to be less than a second (Additional file 1: Fig. S1). The public key-switching key of each authorized researcher is stored in a dedicated key-switching store that dynamically expands to accommodate the number of authorized researchers. We show that the extra storage required for the key-switching store is minimal, around 55 MB per researcher (see Additional file 1: Fig. S2 and Additional file 1: Supplementary Material for an explanation why it is larger than a regular key storage).

Vertical packing allows efficient storage of encrypted large genotype-phenotype databases

We designed SQUiD to handle sensitive genotype-phenotype data from a large number of patients. SQUiD is specifically tailored to ingest data that has already undergone quality control. Here, we represent the data as a table with columns for basic attributes like age, sex, gender, etc.; genotypes for single-nucleotide polymorphisms (SNPs); and the phenotype or disease status of the patients, and rows for each patient in the database (Fig. 2A). While each entry in the table needs to be an integer for the homomorphic encryption libraries we used, continuous phenotypes can be discretized into integers via scaling (see Additional file 1: Supplementary Material for more details). The encryption of this data introduces additional storage requirements compared to its original unencrypted form. Consequently, the storage expenses associated with storing large genotype-phenotype databases in their encrypted state can be substantial. In order to optimize storage within the SQUiD framework, we adopt a vertical packing approach for our data organization (see the "Methods" section). This method involves storing the genotypes of multiple patients for a single SNP within a single ciphertext. Vertical packing in homomorphic encryption is a method where multiple pieces of data are combined into a single, larger unit before encryption. This allows multiple calculations to be performed simultaneously on all the packed data within one operation, rather than processing each piece of data individually in a single instruction, multiple data (SIMD) fashion (see the "Methods" section for details) [29]. By vertically packing our data (Fig. 2A), we effectively reduce the number of ciphertexts necessary to accommodate a substantial volume of data, thereby minimizing the associated storage costs. Such packing still enables homomorphic updates (addition of new patients/SNPs/attributes) to the encrypted database without the need for decryption (see the "Methods" section for details).

We benchmarked the storage requirements of SQUiD on four different types/numbers of SNPs: ClinVar SNPs, Illumina Human1M-Duo v3.0 DNA Analysis BeadChip SNPs, Whole Exome Sequencing (WES) SNPs, and Whole Genome Sequencing (WGS) SNPs. These SNPs can be stored either at a per-chromosome level or genome-wide in SQUiD. Clinvar contains approximately 70,000 SNPs, and Illumina BeadChip arrays contain approximately 1,072,820 SNPs. We estimated that around 8.2 million and 84 million SNPs would be observed in exomes and whole genomes at a population level, respectively, by using the data from 1000 Genomes Project [30]. We have benchmarked the packed storage of SQUiD against an unpacked homomorphic encryption storage, a storage encrypted with the industry standard AES-128-CBC, and a plaintext storage that stores SNPs as single bytes for the various SNP sets (Fig. 3). We found that the storage cost for SQUiD is 49,960x better than the unpacked homomorphic storage cost (Fig. 3). This efficiency is achieved because a single packed ciphertext in SQUiD can store data for up to 49,960 patients, whereas an unpacked ciphertext can only store data for one. Furthermore, vertical packing reduces the time for encryption by 49,960 fold compared to unpacked homomorphic encryption as fewer ciphertexts need to be

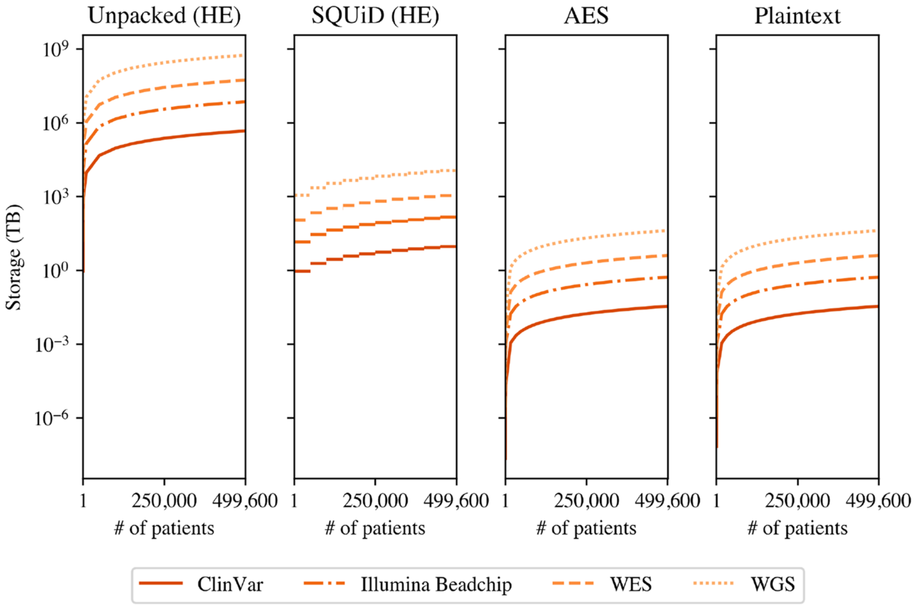


Fig. 3 Plots showing the storage space required to store the ClinVar, Illumina Beadchip, WES, and WGS SNP genotypes with different schemes. The number of SNPs for WES and WGS is approximated using the 1000 Genomes Project

encrypted (Additional file 1: Fig. S3). Vertical packing also improves query performance (Additional file 1: Fig. S4), because HE operations on ciphertexts compute these operations pairwise (i.e., SIMD-like) on the vertically packed data. The query performance of the unpacked solution for the count, MAF, PRS, and similarity queries quickly becomes impractical and is outperformed by the packed solution in databases with as few as 10 patients for the count and MAF queries and just 1 patient for the PRS and similarity queries (Additional file 1: Fig. S4).

Enabling secure, scalable, and fast analysis of genotypes and phenotypes

We have devised four encrypted query functionalities within the SQUiD framework. This modular design allows for seamless implementation of additional functionalities to accommodate diverse analysis requirements. Our queries include count, MAF, PRS, and similarity. Figure 2C depicts how querying works under the public key-switching framework.

Count queries within the SQUiD framework return the number of patients satisfying specific filters or equality checks. For example, a count query could count the number of patients with type-2 diabetes (T2D), whose SNP on gene TCF7L2 has a heterozygous alternative allele. MAF queries are employed to compute the MAF for a given target SNP within a filtered patient cohort. For instance, SQUiD can compute two MAF queries: one for a target SNP on the TCF7L2 gene within a cohort of patients with T2D and another within a cohort of patients without T2D to study correlations between SNPs on the TCF7L2 gene and T2D. We can further add many different filters to build the cohort such as constraining it to patients with homozygous SNPs on a gene of interest. PRS queries involve the calculation and return of the PRS for all patients given a list of GWAS SNPs and their coefficients. PRS queries require only the coefficients and SNPs to be supplied post training such as those found on the PGS catalog [31]. Finally, similarity queries take a target patient's encrypted genotype as input, build a cohort of genetically similar patients in the database through a scoring function like the squared L_2 -norm, and output the number of similar patients with and without a particular disease of interest (see the "Methods" section and Additional file 1: Supplementary Material). To evaluate the performance of each query, we conducted benchmarking against a plaintext implementation. The plaintext implementation keeps the genotype-phenotype data encrypted at rest (as mandated by the policies) and decrypts the necessary components of the data to compute the query in plaintext, while SQUiD keeps the data encrypted both at rest and during computation, enabling much stronger security as the data no longer has visibility to the computing party. This plaintext implementation models the current data access guidelines set by initiatives such as the dbGaP and UK Biobank where researchers download encrypted data, decrypt the data locally, and then analyze the data in plaintext [32].

We implemented SQUiD using the HE library, HElib [33], and benchmarked SQUiD on an *n2-standard-64* Google Cloud instance with an Intel Xeon Gold 6268 processor running at 2.8 GHz and 256 GB of memory (see Additional file 1: Supplementary Material for more details on the experimental setup and HE parameters). On a dataset with 499,600 patients, SQUiD can perform a count query with 2 filters in 4 min (0.004 s in plaintext), a MAF query with 2 filters in 5 min (0.004 s in plaintext), a PRS query with

1024 effect SNPs in 2 min (0.59 s in plaintext), and a similarity query with 1024 SNPs in 10 h (2.7 s in plaintext) (Fig. 4).

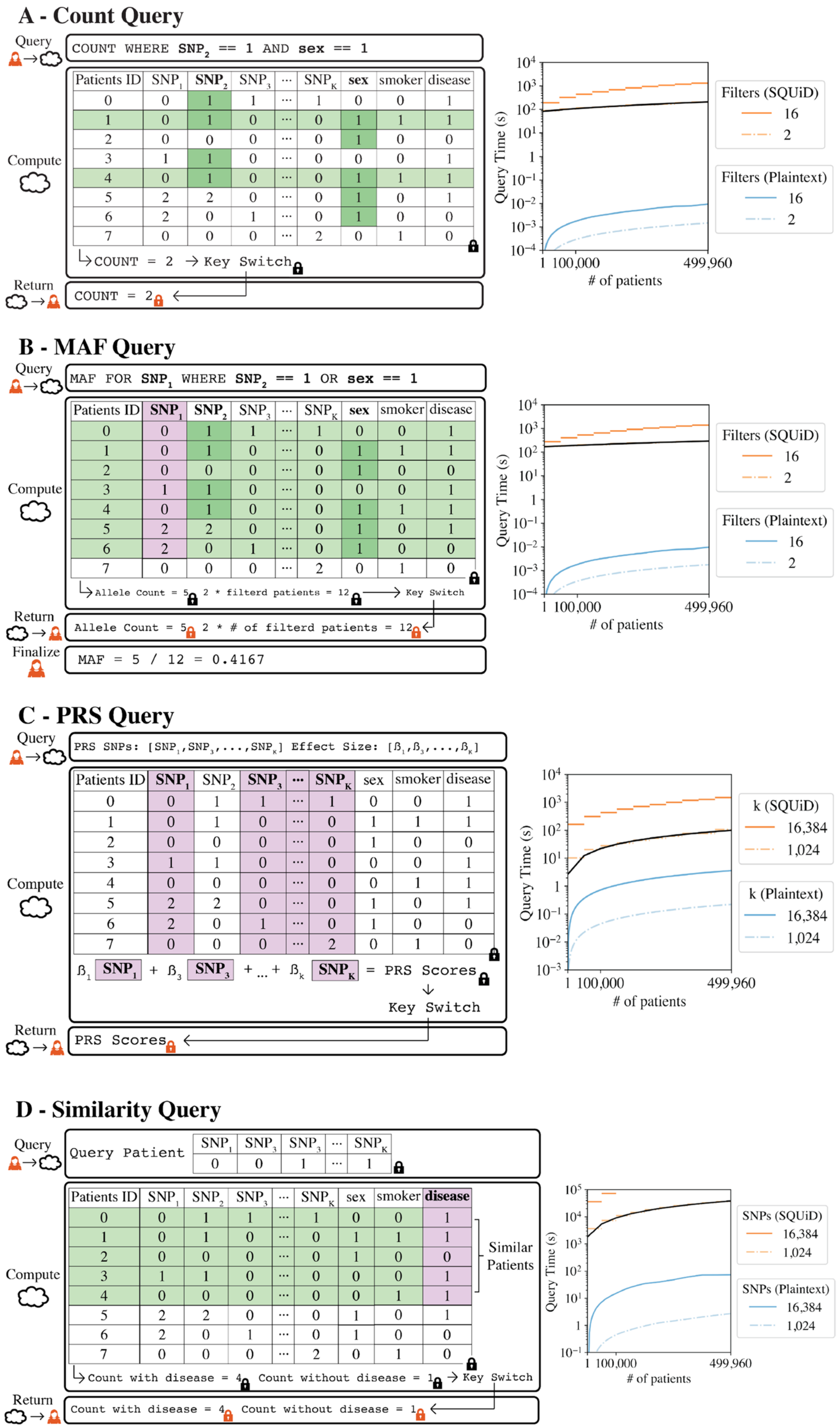
We investigated the overhead of the HE library used in SQUiD and the overhead of the algorithms developed in SQUiD by comparing the query times against a solution that is a direct translation of SQUID algorithms without using the HE library (denoted as "SQUiD without HE"). SQUiD without HE computes the plaintext versions of the SQUiD queries by converting the HE library functions to their plaintext counterparts and then using these functions in the same way that SQUiD does. We found that the SQUiD without HE solution has a 20x overhead for a count and MAF queries with 2 filters, 2.5x overhead for a PRS query with 1024 effect SNPs (k), and a 180x overhead for a similarity query with 1024 SNPs compared to the plaintext solution (Additional file 1: Fig. S5).

We also show that our queries are highly parallelizable because of their linear structure. Each query involves computing a filter, a linear combination, or an L_2 similarity for a set of SNPs across patients. Since these operations are performed for each patient, SQUiD achieves parallelism by chunking the database rows and processing these chunks concurrently. Our benchmarking in a multi-threaded environment shows that query performance scales linearly with the number of cores used (Fig. 5). On large databases with millions of patients, this scaling ensures reasonable query performance. To demonstrate this, we also ran each query with 50 threads on a database with 9,992,000 patients and found that a count query with 2 filters took 3 min, a MAF query with 2 filters took 4 min, a PRS query with 1024 effect SNPs took 5 min, and a similarity query with 1024 SNPs took 4 h (Additional file 1: Fig. S6).

Our results show that all the functionalities implemented in SQUiD exhibit linear scaling relative to the size of their inputs. Specifically, the count and MAF queries scale linearly with the number of filters with a slope of 0.62, the PRS query scales linearly with the number of SNPs with a slope of 0.001, and the similarity query scales linearly with the number of SNPs given for the target patient with a slope of 0.068

(See figure on next page.)

Fig. 4 **A**, **B**, **C**, **D** For each query, the plots on the right show the query time by varying the number of filters for the count and MAF query, by varying the number of SNPs and effect sizes (k) for the PRS query, and by varying the number of SNPs for the similarity query. The query time for SQUiD and the query time of a plaintext solution are shown for comparison. The plaintext solution works on a database encrypted with AES. For each plaintext query, the necessary components for the query are decrypted and then computed on. **A** Count query. The count query returns the number of patients that pass a given filter in the query (patients who pass the filter are highlighted in green, with darker green cells indicating passing a condition). A black line of best fit for a count query with 2 filters is given as the equation time (s) = 0.00025(# of patients) + 82.71. Due to the strict linear scaling, the performance of our query can easily be interpolated by this line of best fit. **B** MAF Query. The MAF query creates a filtered cohort of patients (patients who pass the filter are highlighted in green, with darker green cells indicating passing a condition) and computes the MAF of a target SNP for that cohort (purple SNPs). A black line of best fit for a MAF query with 2 filters is given as time (s) = 0.00025(# of patients) + 170. **C** PRS query. The PRS query returns the PRS score of all patients for a pre-determined PRS SNP set and their effect sizes. A black line of best fit for a prs query with 1024 effect SNPs is given as time (s) = 0.00019(# of patients) + 2.6. **D** Similarity query. The similarity query returns the number of patients with and without a disease from a cohort of patients similar to a target patient (patients highlighted in green). The target patient's genome is encrypted with the owner's public key when it is sent to the public cloud. A black line of best fit for a similarity query with 1024 SNPs is given as time (s) = 0.073(# of patients) + 1800



A - Count Query

Query: `COUNT WHERE SNP_2 == 1 AND sex == 1`

Patients ID	SNP_1	SNP_2	SNP_3	···	SNP_K	sex	smoker	disease
0	0	1	1	···	1	0	0	1
1	0	1	0	···	0	1	1	1
2	0	0	0	···	0	1	0	0
3	1	1	0	···	0	0	0	1
4	0	1	0	···	0	1	1	1
5	2	2	0	···	0	1	0	1
6	2	0	1	···	0	1	0	0
7	0	0	0	···	2	0	1	0

```
COUNT = 2 → Key Switch
```

Return: `COUNT = 2`

B - MAF Query

Query: `MAF FOR SNP_1 WHERE SNP_2 == 1 OR sex == 1`

Patients ID	SNP_1	SNP_2	SNP_3	···	SNP_K	sex	smoker	disease
0	0	1	1	···	1	0	0	1
1	0	1	0	···	0	1	1	1
2	0	0	0	···	0	1	0	0
3	1	1	0	···	0	0	0	1
4	0	1	0	···	0	1	1	1
5	2	2	0	···	0	1	0	1
6	2	0	1	···	0	1	0	0
7	0	0	0	···	2	0	1	0

```
Allele Count = 5  2 * filterd patients = 12 → Key Switch
```

Return: `Allele Count = 5  2 * # of filterd patients = 12`

Finalize: `MAF = 5 / 12 = 0.4167`

C - PRS Query

Query: `PRS SNPs: [SNP_1,SNP_3,...,SNP_K] Effect Size: [ß_1,ß_3,...,ß_K]`

Patients ID	SNP_1	SNP_2	SNP_3	···	SNP_K	sex	smoker	disease
0	0	1	1	···	1	0	0	1
1	0	1	0	···	0	1	1	1
2	0	0	0	···	0	1	0	0
3	1	1	0	···	0	0	0	1
4	0	0	0	···	0	0	1	1
5	2	2	0	···	0	1	0	1
6	2	0	1	···	0	1	0	0
7	0	0	0	···	2	0	1	0

$ß_1$ SNP$_1$ + $ß_3$ SNP$_3$ + ... + $ß_k$ SNP$_K$ = PRS Scores → Key Switch

Return: `PRS Scores`

D - Similarity Query

Query Patient

SNP_1	SNP_3	SNP_3	···	SNP_K
0	0	1	···	1

Patients ID	SNP_1	SNP_2	SNP_3	···	SNP_K	sex	smoker	disease
0	0	1	1	···	1	0	0	1
1	0	1	0	···	0	1	1	1
2	0	0	0	···	0	1	0	0
3	1	1	0	···	0	0	0	1
4	0	0	0	···	0	0	1	1
5	2	2	0	···	0	1	0	1
6	2	0	1	···	0	1	0	0
7	0	0	0	···	2	0	1	0

Similar Patients

```
Count with disease = 4  Count without disease = 1 → Key Switch
```

Return: `Count with disease = 4  Count without disease = 1`

Fig. 4 (See legend on previous page.)

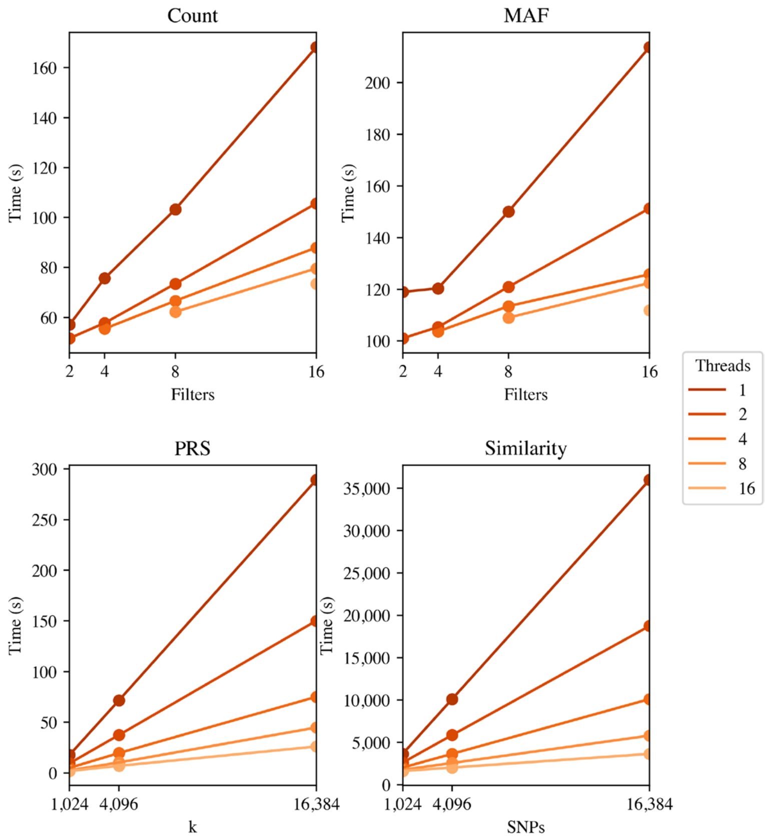


Fig. 5 Plots of count, MAF, PRS, and similarity query time by the number filters, effect SNPs (k), and SNPs varying the number of threads. We benchmarked the time for each query on a database with 49,960 patients using 2, 4, 8, and 16 filters for the count and MAF queries, and 1,024, 4,096, and 16,384 SNPs for the PRS and similarity queries

(Fig. 4). Our slopes consistently indicate a slow growth in runtime. Notably, the runtime of all protocols is proportional to the number of patients in the database and independent of the total number of SNPs in the database (Additional file 1: Fig. S7). A plaintext implementation of our protocols would also scale linearly with the number of patients in the database and the number of filters and SNPs involved in the query. Thus, SQUiD achieves optimal linear scaling as expected from a plaintext implementation, which signifies its ability to efficiently adapt to larger datasets in the future. Furthermore, with the expected decrease in the price of cloud computing in the future, the steady runtime observed for all queries ensures that increasing the size of the databases beyond the limits benchmarked in this study will yield steady performance outcomes, enabling real-world applications of SQUiD with biobank-scale data. We also show that the SQUiD's communication cost for all queries except PRS query is constant regardless of the number of patients in the database while communication cost increases with the number of patients for all query types in plaintext (Additional file 1: Fig. S8 and S9). Overall, the communication is minimal. Comparable to an instagram post which has a maximum size of 4.3 MB (1080 by 1350 pixels) [34], most of our queries use less than 50 MB on databases with 100,000 patients.

We also developed an API and a command line interface (CLI) to facilitate interaction with SQUiD, thereby enhancing its usability for researchers (Additional file 1: Fig. S10). The API and CLI enable researchers to execute various queries and perform essential functions through simple commands. For instance, researchers can generate private and public keys required for encryption and authorization, send the public key to the data owner, execute all desired queries (See Additional file 1: Table S2 for query parameters), and decrypt the returned query results. The API simplifies the deployment process for researchers who are not experts in privacy and security when utilizing SQUiD.

SQUiD can reproduce known genotype-phenotype relationships in UK Biobank

We studied the relationship between patients with T2D and a control group in the UK Biobank dataset to assess the accuracy of the MAF and count queries in SQUiD. Firstly, we calculated the MAFs for the top five SNPs with the largest difference between T2D patients and the control group patients (Fig. 6A). We compared the MAFs computed by SQUiD with the MAFs computed in plaintext to show there is no difference between them. Secondly, for these same five SNPs, we computed a chi-square statistic by using the allele counts for the control and case group (T2D in our case) [12]. We used the count query in SQUiD to get the allele counts and then computed the chi-square statistic in plaintext. The chi-square scores obtained from SQUiD queries are identical to the plaintext computation results (Fig. 6B). Note that SQUiD does not directly execute GWAS, but it has the capability to generate cohorts with specific attributes. We have shown that it can create accurate cohorts that will result in accurate GWAS (Fig. 6).

We further evaluated the accuracy of SQUiD by replicating the sparse PRS calculations for standing height and T2D performed in the UK Biobank PRS study [35] using both plaintext calculations and the SQUiD PRS query. The standing height and T2D PRS use 51,209 and 183,830 SNPs, respectively. They are the traits with the most number of SNPs involved in PRS calculations in the UK Biobank. We performed these calculations for 20,000 randomly selected patients in the UK Biobank. Our analysis revealed no observable difference in the PRS distribution and scores between plaintext and SQUiD queries (Fig. 7). Notably, the sole discrepancy between the calculations arose from a marginal loss in precision. To accommodate the requirements of using integers in SNP effect sizes in SQUiD PRS queries, the effect sizes were multiplied by 1000 and converted to integers. However, the resulting precision loss was minimal (Fig. 7B, C).

Discussion

We introduce SQUiD, a novel, secure, and user-friendly queryable genotype-phenotype database implemented using homomorphic encryption. We envision SQUiD as a valuable tool for data owners, including hospitals, non-profit academic research institutions, and government health agencies, offering them a secure means to store genotype-phenotype data in the cloud while enabling authorized researchers to securely analyze this data. We propose that our system has the potential to replace existing genotype-phenotype databases, delivering enhanced security measures without compromising functionality. By employing homomorphic encryption, SQUiD offers a robust, scalable, and practical solution to mitigate privacy risks associated with sensitive genetic and phenotypic data. We demonstrate this by showing that SQUiD can scale with increasing

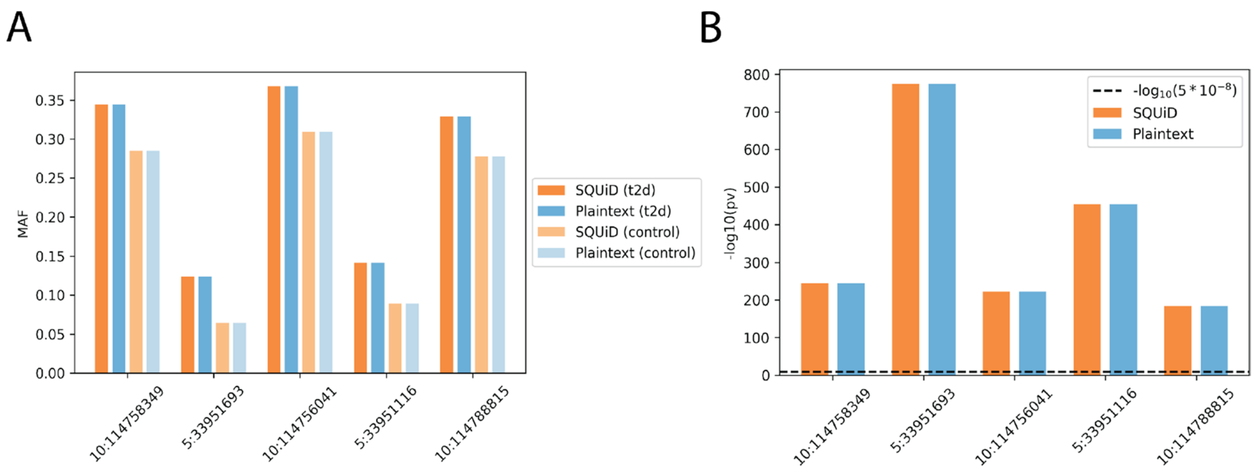


Fig. 6 **A** Histograms of the MAFs of SNPs exhibiting the most substantial difference between control and T2D patient groups. The MAFs were calculated with SQUiD (orange), and in plaintext (blue). **B** Plot of -log(*p*-value) for the SNPs in **A**

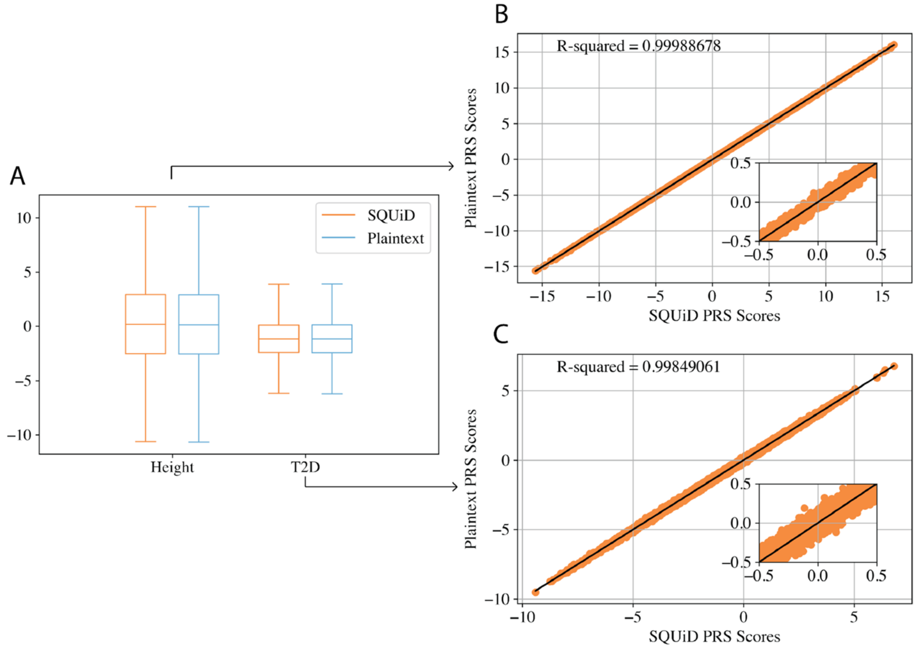


Fig. 7 **A** Boxplots of the PRS score distributions of UK Biobank patients for standing height and type 2 diabetes (T2D) calculated with SQUiD (orange) vs plaintext (blue). **B** A scatter plot of the height PRS calculated by SQUiD vs. plaintext, where each point represents a patient. The black line is a line of best fit with an R^2 of 0.9999. **C** The same plot as **B** for T2D with an R^2 value of 0.9985

numbers of patients and SNPs in a genotype-phenotype database, by performing a simple study on UKBB data, as well as by replicating PRS calculations in UKBB [35]. All our query protocols (count query, MAF query, PRS query, and similarity query) and encryption protocols (setup of the database) were run on single-threads unless otherwise indicated.

SQUiD leverages homomorphic encryption, which, to date, presents three key challenges. Firstly, traditional homomorphic encryption was designed for a two-party setting involving a server and a client. Secondly, it is known to incur a high storage cost. Lastly, analysis with homomorphic encryption tends to be slow. To overcome the first challenge, we adopted the established concept of the proxy re-encryption system [25] and adapted it to develop a theoretical and practical implementation within the framework of homomorphic encryption. This adaptation, which we refer to as the public key-switching technique, enables secure multi-client queries on encrypted data, specifically addressing the needs of our application in the biomedical domain.

Furthermore, we demonstrate a significant improvement in storage efficiency through the application of a well-known vertical packing storage method, achieving a storage enhancement of 49,960 times compared to a naive homomorphic encryption solution. While storing SNP genotypes using homomorphic encryption increases storage costs relative to state-of-the-art encryption methods like AES, this approach is indispensable as homomorphic encryption allows execution of functions on encrypted data. While our queries demonstrate slower performance compared to plaintext solutions, we consider the trade-off between security and performance to be within acceptable limits. Additionally, implementing multi-threading significantly enhances performance. This performance overhead is unlikely to significantly impact the usability and utility of the framework for researchers. This is because the alternative is to download a large database and analyze the data locally, which is a much more time-consuming and resource-intensive process. Therefore, we believe that our framework offers an optimal balance of security and performance.

Although encrypted database systems do exist, to the best of our knowledge, none of them offer the same level of security guarantees and functionality as SQUiD. A developed secure database framework named CryptDB [36] offers efficient secure data storage and query performance. However, it does not offer the functionalities provided by SQUiD for two main reasons. Firstly, this framework is unable to compute the same set of queries as SQUiD. For instance, CryptDB lacks the ability to add *and* multiply encrypted database items, a necessary requirement for computing the linear combinations in PRS queries. Secondly, and more critically, CryptDB exhibits significant information leakage during equality checks used in the filtering process in count and MAF queries. Specifically, CryptDB exposes the count of unique items within the columns used for the equality checks. For genotype-phenotype databases that store SNPs with just three possible genotypes with known allele frequencies, CryptDB would expose the patients with the same genotypes for each SNP. This information could be combined with the known and well-studied population frequencies of each SNP to devise a simple attack that reconstructs the genotype values for each patient in the database, resulting in a complete breach of security. Furthermore, while databases that keep data encrypted at rest with AES can answer the same queries as SQUiD, they cannot perform these queries as securely as SQUiD does. For any query, these databases must first decrypt the relevant data to compute the query, exposing the data to potential attacks. In contrast, SQUiD can perform all queries without the need for decryption.

Privacy-preserving MAF calculations using homomorphic encryption were proposed before [37]. Notably, SQUiD's MAF query differs from this approach as it computes the MAF within a filtered patient cohort, where the filtering is done via protocols developed in this work. For a detailed mathematical exposition of these distinctions, refer to Additional file 1: Supplementary Material.

We compared our patient similarity queries to existing private patient similarity queries (SPQ). Many existing SPQ protocols such as Wang et al. [18] privately compute patient similarity under the secure multiparty computation security assumptions, (i.e., non-colluding parties). Since SQUiD employs homomorphic encryption, no assumptions about collusion between parties are necessary. Additionally, our query process involves a single round of communication, with the querying researcher sending a query to the cloud and receiving a prompt response. In contrast, the protocol outlined in [18] necessitates an interactive protocol with multiple rounds.

We also empirically compared SQUiD to another study [38] due to the similar security settings. This study proposes a partial homomorphic encryption algorithm that supports only ciphertext addition and scalar multiplication operations for computing patient similarity using a squared L_2-norm. We implemented the euclidean distance (equivalent to the squared L_2-norm protocol) from the study [38] to the best of our understanding for comparison purposes. Additional file 1: Fig. S11 shows that SQUiD can compute the squared L_2-norm faster for larger datasets with an approximately 2x speed up for datasets with 49,960 patients.

We envision three use cases for this framework: (1) funding agencies such as NIH can employ this framework to disseminate insights derived from the data currently available through the NIMH Data Archive (NDA) or Database of Genotypes and Phenotypes (dbGAP), (2) multi-site consortia can employ this framework to disseminate data to their members while keeping the data secure in cloud storage, (3) learning health systems can employ this framework to disseminate data to their researchers while keeping the data secure in cloud storage. Our secure framework is designed to enable users to form specific patient cohorts based on desired characteristics. Within this system, users can also determine the distribution of PRS for a particular disease across various patient populations. For example, one can explore the PRS distribution for schizophrenia among patients diagnosed with bipolar disorder. Additionally, the framework allows for the analysis of disease outcomes in patients who share genetic similarities with a specific patient of interest, facilitating more personalized and targeted approaches to healthcare and research.

We designed SQUiD with ease of use in mind for both researchers and data owners. Data owners are only required to provide a VCF file for the genotypes and a CSV file for the phenotype data. SQUiD then handles the packing, encryption, and uploading of this data to a public cloud platform. The SQUiD codebase includes a cloud-deployable API framework, allowing researchers to query the data through API calls seamlessly.

SQUiD's design is scalable to support multiple data owners. In a multi-owner system, each data owner independently prepares, encrypts, and uploads their data to the public cloud. Each data owner's information is stored in a separate encrypted database along with a corresponding key store. When a researcher wishes to query the data, they send their public key to each data owner. The data owners then generate a public key-switching key using their secret key and store this in their key-switching store. The researcher's query is processed in the public cloud, where it is evaluated across the encrypted databases. The results are key-switched using the respective data owner's key-switching store, ensuring that the final query results are encrypted under the researcher's public key. These results are aggregated and sent back to the researcher, who can then decrypt them to obtain the final output.

This multi-owner implementation introduces additional storage, as the public cloud must store multiple key-switching stores for each data owner. It also needs to maintain evaluation keys (relinearization keys, rotation keys, and bootstrapping keys) for each data owner. Despite this, the process ensures the privacy of each data owner is maintained, and the researcher can securely access aggregated results without compromising individual data security. There are also approaches that do not require multiple evaluation keys for each owner [39], however, they are not tailored to the specific needs of genotype-phenotype data.

SQUiD can handle missing data. For count and MAF queries, SQUiD defaults to excluding patients with missing values from the cohort being analyzed. For the PRS and similarity queries, any column with missing SNPs will be excluded from the PRS and similarity query calculation.

Conclusions

SQUiD presents an innovative and impactful solution for a world grappling with escalating concerns surrounding security and privacy of genetic and clinical data. By circumventing the challenges posed by the ever-changing, heterogeneous landscape of data protection laws, SQUiD offers a robust framework to safeguard sensitive information. Moreover, we firmly believe that SQUiD has the potential to enhance patient trust by ensuring the security and controlled utilization of their data for specific research purposes and thus has the potential to increase participation in genetic research. Lastly, although this study focused on genotype-phenotype analyses for proof of principle, SQUiD's modular design allows for the integration of other discrete data modalities and analytic approaches, as the need arises. This adaptability will be critical at a time when precision medicine research is rapidly expanding to encompass more complex molecular and clinical datasets.

Methods

Security and threat models

Our security assumption is based on the current data-sharing policies within many public and private entities, that is, the data owner and authorized researchers are mutually trusted. Thus, authorized researchers are allowed to query the genotype-phenotype data that do not threaten the confidentiality of patients according to the data use agreements. The inherent data leakage from query results and potential inference attacks from authorized researchers are therefore not considered.

Meanwhile, genotypes and phenotypes as well as a subset of the queries are protected from the public cloud and attackers. More precisely, we consider the following three threat models for database management [40, 41]:

- Snapshot attackers that obtain a snapshot of the database
- Persistent passive attackers that compromise the cloud server to obtain not only the database but also queries and all server's operations
- Active attackers that fully compromise the server to deviate from pre-designed protocols for queries

In our SQUiD construction, snapshot attackers receive ciphertexts of the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme. We use the security level estimator from HElib [33, 42] to choose BGV parameters that provide a 128-bit security level against known attacks. Consequently, the security towards snapshot attackers inherits from BGV's IND-CPA security, i.e., the ciphertexts are almost indistinguishable from random characters.

For persistent passive attackers, there are many ways that querying encrypted databases can result in private information leakage [43–46]. Most prominent ones include leakage through (1) *access pattern*, which determines if certain records are consistently accessed, and (2) *search pattern*, which indicates if and when an encrypted query is repeated. Many cryptosystems, including property-preserving encryption (PPE) [47, 48] and searchable encryption (SE) [49, 50], fail to protect against these types of information leaks. This is primarily due to their inherent functionality, which inadvertently discloses properties of datasets, thereby compromising privacy. However, homomorphic encryption schemes such as BGV provide a solution that does not leak access and search patterns [51]. Using HE to encrypt databases propels algorithms that have to touch all the relevant records in the dataset for a single query. For example, to find out whether an encrypted input is in the encrypted database, the input needs to be compared with every single encrypted value in the database homomorphically. This prevents access pattern leakages since the access pattern remains uniform for all queries. In addition, search pattern leakages are prevented due to the IND-CPA security under carefully selected parameters, since encrypted queries are indistinguishable from one another, regardless of their contents [51].

It is worth mentioning that persistent passive attackers do not learn additional information about the database from knowledge of the server's computation patterns. Precisely, when an authorized researcher sends a query *f*, the server performs a series of operations on the encrypted database $\mathbf{Enc}(m)$ to obtain $\mathbf{Enc}(f(m))$. The function *f* is in plaintext for Count, MAF and PRS queries and contains ciphertexts for similarity queries. In all these cases, the computation pattern for the server is predefined and contains operations such as homomorphic additions, multiplications, and key switching. As such, inference attacks from persistent passive attackers are also prevented, as only computational patterns of different functions are revealed but not any computation result $f(m)$.

While the problem of defending against active attackers is challenging and still unsolved [40, 52], our SQUiD construction provides reasonable mitigation towards active attackers. Namely, active attackers can deviate from pre-determined operations in SQUiD and therefore send wrong computation results to authorized researchers, but they can not learn information about the database.

Homomorphic encryption

Encryption is a procedure that maps the *plaintext* data into its *ciphertext*, such that the plaintext can not be deduced from the ciphertext without knowing the secret key. Homomorphic encryption is a class of encryption schemes with an additional property: computations can be performed over ciphertexts without knowing the secret key.

Figure 8 visualizes this property in a commutative diagram, which enables secure computation outsourcing.

To compute a function *f* on plaintexts $m_1, \ldots, m_t$ without revealing them, plaintexts are encrypted and corresponding ciphertexts $\mathbf{Enc}(m_1), \ldots, \mathbf{Enc}(m_t)$ are sent to the public cloud. Then, a function $\tilde{f}$, which corresponds to the HE-friendly version of the desired function *f*, is evaluated among the ciphertexts homomorphically. As a result, a ciphertext of $f(m_1, \ldots, m_t)$ is derived, which contains the evaluation result equivalent

$$\begin{tikzcd}
m_1, \ldots, m_t \arrow[r, "f"] \arrow[d, "\mathbf{Enc}"] & f(m_1, \ldots, m_t) \\
\mathbf{Enc}(m_1), \ldots, \mathbf{Enc}(m_t) \arrow[r, "\tilde{f}"] & \mathbf{Enc}(f(m_1, \ldots, m_t)) \arrow[u, "\mathbf{Dec}_{sk}"']
\end{tikzcd}$$

Fig. 8 The homomorphic evaluation of a function *f* on ciphertexts

to that of a plain evaluation. Therefore, the decryption of the final ciphertext outputs the desired evaluation result.

HE ciphertexts contain a *noise* component, whose value grows with homomorphic operations. This is controlled by pre-fixed HE parameters, which is also used to set a noise budget. If the number of operations in an algorithm is too large such that the noise consumption exceeds the budget, then the result can no longer be decrypted correctly. To avoid this, a *bootstrapping* operation is introduced to refresh the ciphertexts, enabling the *fully* homomorphic encryption (FHE) schemes that support evaluations of *arbitrary* circuits for different operations including multiplications and additions (i.e., arbitrary *f*) [53]. Detailed realizations of homomorphic operations are included in the supplementary material.

Brakerski-Gentry-Vaikuntanathan scheme (BGV)

The BGV scheme is an FHE scheme that relies on the hardness of the ring learning with error (RLWE) problem [54]. Its basic building blocks are homomorphic addition ADD and multiplication MULT. Since any computable function can be realized with additions and multiplications, the homomorphic evaluation of any computable *f* can be realized with ADDs and MULTs. Bootstrapping in BGV is a very costly operation [55, 56]. It is, therefore, common to use BGV in the *leveled* manner, i.e., to choose the HE noise parameter with large noise capacity such that computations can be performed without bootstrapping. Our study uses the leveled version of BGV.

BGV allows efficient computations in the amortized sense. It supports single instruction, multiple data (SIMD) operations, which allows multiple values to be packed into one BGV ciphertext, enabling computations over a single ciphertext to be performed on all packed values in an efficient manner [57]. Details of the SIMD packing are included in the supplementary material.

Public key-switching

In general, HE binary operations only support input ciphertexts that are encrypted under the same key. Therefore, in the scenario of multiple users each holding their own keys, there is a natural need to convert a ciphertext encrypted under one key to another ciphertext that encrypts the same message under a different key. A naive approach is to decrypt and re-encrypt with a different key, but this exposes the original secret key and the message to the party that performs this procedure. To prevent such leakages, the above procedure can be done *homomorphically* such that the evaluation party can not access the message in the clear. Such a technique is called *key switching*. Mathematically, when converting the key system from $(\mathsf{pk}, \mathsf{sk})$ to $(\mathsf{pk}^*, \mathsf{sk}^*)$, the evaluation party does not need to know sk, but a key-switching key $\mathsf{ksk}_{(\mathsf{sk} \rightarrow \mathsf{sk}^*)}$ which leaks no information about secret keys.

Our scenario exploits the key-switching key $\mathsf{ksk}_{(\mathsf{sk} \rightarrow \mathsf{sk}^*)}$. While the traditional key-switching key generation uses both sk and sk^*, only sk and pk^* are needed in our design; hence, it is called public key-switching. This design preserves the confidentiality of sk^* as it does not need to be shared to compute the key-switching key. In our scenario, the secret key of the authorized researchers does not need to be sent to the data owner to generate the key-switching key. Please see supplementary material for the mathematical details of the realization of public key-switching with BGV and how we control the increasing noise.

Database construction with vertical packing

The dataset in SQUiD is represented as a matrix $M = \{m_{(i,j)} | 1 \leq i \leq r, 1 \leq j \leq k\}$, where r is the number of patients, k is the number of attributes (features), and the value in position (i, j) corresponds to the j-th feature of the i-th patient (e.g., the genotype of j-th SNP of i-th patient). We use the term *vertical* or *horizontal* for the direction in the matrix, which corresponds to an attribute for all patients or all attributes for a single patient, respectively.

As we explained earlier, BGV supports packing multiple messages into one ciphertext. SQUiD packs elements *vertically*: let ℓ denote the packing capacity in a ciphertext, then the r elements in the jth column are encrypted into $\lceil r/\ell \rceil$ ciphertexts

$$\mathsf{ct}_{(s,j)} = \mathbf{Enc}\big(\{m_{(\ell \cdot s+1,j)}, m_{(\ell \cdot s+2,j)}, \cdots, m_{(\ell \cdot (s+1),j)}\}\big)$$

where $1 \leq s \leq \lceil r/\ell \rceil$ and $m_{i,j}$ is considered as 0 for $i > r$. Overall, entire dataset is encrypted into $C = \{\mathsf{ct}_{(s,j)} \mid 1 \leq s \leq \lceil r/\ell \rceil, 1 \leq j \leq k\}$.

The update, insert, and delete operations on a vertically packed encrypted database vary slightly from their typical implementations.

- *Update:* To update a single value m' at index i, j, a new encryption of $\mathsf{ct}_{(s,j)}$ where $s = \lceil i/\ell \rceil$ needs to be uploaded where

$$\mathsf{ct}_{(s,j)} = \mathbf{Enc}\big(\{m_{(\ell \cdot s+1,j)}, m_{(\ell \cdot s+2,j)}, \cdots, m', \cdots, m_{(\ell \cdot (s+1),j)}\}\big)$$

- *Insert:* To insert a new row at $r + 1$, if ciphertexts are not fully packed (i.e., $\ell \not| r$), then the last row of packed ciphertexts contains zeros at row index $r + 1$, which are updated. Otherwise, the following k fresh ciphertexts are added, forming the last row of C.

$$\big\{\mathsf{ct}_{(s+1,j)} = \mathbf{Enc}\big(\{m_{(\ell \cdot r+1,j)}, 0, \cdots, 0\}\big),\ 1 \leq j \leq k\big\}$$

- *Delete:* To delete an entry at index i, j, a plaintext, which encodes zero at the i mod ℓ-th slot and one elsewhere is multiplied with $\mathsf{ct}_{(\lceil i/\ell \rceil, j)}$.

Note that update and insert operations both upload new ciphertexts with low noise, but the delete operation increases the noise with a plaintext-ciphertext multiplication. To bound the noise growth, we set a number α for the maximum times of consecutive delete operations. On the $(\alpha + 1)$-th time to delete an entry, an update should be performed instead, after which α deletes are again allowed. For SQUID with our experimental parameters, the value α is taken to be 5.

Functionalities

In this section, we describe the supported functionalities of SQUiD and the evaluation procedures using homomorphic encryption.

Count queries

The first category of queries is to *count* the number of patients *whose* attributes satisfy certain conjunctive (AND) and/or disjunctive (OR) relations. Its evaluation contains two stages, filtering and vertical aggregation.

Filtering Suppose the researcher specifies $\tau > 1$ selection criteria (either in plaintext or ciphertext) and their relation (AND and/or OR). The filtering stage outputs a *predicate vector* $\mathbf{p}$ composed of r encrypted binary numbers. If the element $\mathbf{p}[i]$ decrypts to 1, then the patient i is in this pre-defined cohort.

First, we explain how to homomorphically check a single selection criterion, which amounts to performing a homomorphic equality test between the given value in a query and a value in the matrix. The key idea is to find a polynomial representation, which can be evaluated as a sequence of homomorphic additions and multiplications.

$$\text{Function } \mathsf{EQTest} \Longleftrightarrow \text{Polynomial} \Longleftrightarrow \text{Sequence of } \mathsf{ADD}\text{s and } \mathsf{MULT}\text{s}$$

Without loss of generality, we consider the inputs of EQTest as genotype values in $\{-1, 0, 1, 2\}$ where -1 indicates a missing SNP, and denote them as u and v. As shown in Table 1, this function determines a unique truth table.

We derive the polynomial representation of EQTest(u, v) as follows. Let v be an encrypted matrix value, and u be the query value, which can be either in the clear or encrypted depending on the researcher. If u is provided in the clear, then we can interpolate the u-th column of the truth Table 1 into a degree-3 polynomial F_u with input variable v. If u is also encrypted, then we precompute a polynomial F of degree 5 that maps 0 to 1 and $\{\pm 1, \pm 2, 3\}$ to 0, whose input variable is $u - v \in \{0, \pm 1, \pm 2, 3\}$. Note that we do not consider the case where both u and v are missing because it is assumed that the query value would never look for missing SNPs.

Second, we explain how to homomorphically combine the results of multiple equality checks using AND and OR. Let $\{(u_k, v_k)\}_{k=1}^{\tau}$ be the set of (encrypted or unencrypted) queries and (encrypted) matrix values, then for each patient i we compute

Table 1 The truth table of $\mathsf{EQtest}(u, v)$ for SNPs. We assume that the query value u is not missing ($u \neq -1$)

u			
v	**0**	**1**	**2**
−1	0	0	0
0	1	0	0
1	0	1	0
2	0	0	1

the expression homomorphically as Eq. 1, where d and b are constants in Table 2. The evaluation decrypts to 1 if the data of the patient i matches the selecting criteria, and 0 otherwise.

$$x_i = d + \prod_{k=1}^{\tau} [b + \mathsf{EQTest}(u_k, v_k)] \tag{1}$$

Vertical aggregation Suppose each ciphertext provides ℓ SIMD slots, then the predicate vector for r patients is batched into $\lceil r/\ell \rceil$ ciphertexts. The procedure of summing over these batched messages is a *vertical* aggregation.

Our design fully exploits the advantages of parallel computing. Namely, we perform $\mathcal{O}(r/\ell)$ homomorphic additions with additive depth $\mathcal{O}(\log(r/\ell))$ to obtain one ciphertext, whose ℓ slots are then aggregated with $\mathcal{O}(\log \ell)$ homomorphic rotations and additions. To support larger databases sizes, not all ℓ slots might be aggregated as the aggregated value would overflow the ring in the BGV scheme. In these cases, it is up to the client to aggregate the remaining slots. Please see supplementary material for details of homomorphic addition and rotations with BGV.

PRS queries

The second category of queries is to obtain *polygenic risk scores* of all the patients.

Definition 1 The polygenic risk score (PRS) of a patient is a linear combination of values of attributes in a subset S. For given coefficients (i.e., effect sizes) $\{\beta_j\}_{j\in S}$, the PRS for patient i is $f_i = \sum_{j\in S} \beta_j \cdot m_{(i,j)}$, where $m_{(i,j)}$ is the genotype of the j-th SNP for i-th patient.

The PRS for each patient can be calculated with homomorphic multiplication and additions. Please see supplementary material for details of homomorphic addition and multiplications with BGV.

Horizontal aggregation PRS queries aggregate information horizontally. We use parallel computing to minimize the execution time, and as can be seen from the "Results" section, answering PRS queries is relatively fast.

MAF queries

The third category of queries is to calculate the *minor allele frequency* for a target SNP of a filtered cohort of patients.

Table 2 Constants in circuit (1) [58]

Query type	*b*	*d*
Conjunction	0	0
Disjunction	1	1

Definition 2 Minor allele frequency (MAF) is the frequency at which the minor allele occurs in a given population or a cohort. Let **p** be the predicate vector for r patients, where $\mathbf{p}[i]$ indicates whether the patient i is in the cohort. Then, for the dataset $M = \{m_{(i,j)}\}$, the MAF for SNP j with **p** is

$$\mathsf{AF}(\mathbf{p}, j) = \left(\sum_{i=1}^{r} m_{(i,j)} \cdot \mathbf{p}[i]\right) / \left(2\sum_{i=1}^{r} \mathbf{p}[i]\right),$$

$$\mathsf{MAF}(\mathbf{p}, j) = \min(\mathsf{AF}(\mathbf{p}, j), 1 - \mathsf{AF}(\mathbf{p}, j)).$$

As the homomorphic division and minimum comparisons are currently expensive operations, the cloud instead computes the numerator and denominator homomorphically and then returns the results to the clients for decryption, division, and the minumum operation.

Similarity queries

The fourth category of queries determines whether a specific individual (denoted as d) is genetically similar to patients *with* a certain disease or those *without*. There are two similarity metrics for researchers to choose from.

Definition 3 Suppose the database stores k attributes and the last attribute is the disease.

1. The L_2-distance similarity score $S_{L_2}(i, d)$ is defined as

$$S_{L_2}(i, d) = \sum_{j=1}^{k-1} (m_{(i,j)} - d_j)^2.$$

2. The Jaccard similarity score $S_{\mathsf{Jcd}}(i, d)$ is defined as

$$S_{\mathsf{Jcd}}(i, d) = \sum_{j=1}^{k-1} \mathsf{EQTest}(m_{(i,j)}, d_j),$$

where $\mathsf{EQTest}(\cdot, \cdot)$ equals to 1 if two inputs are equal and 0 otherwise.

In other words, the similarity score $S_{(\cdot)}(i, d)$ horizontally aggregates the result of the squared difference or EQTest.

As a result of this query, the researcher will receive two encrypted values r_1, r_2 from the cloud. The value r_1 is the number of patients that are genetically similar to the target d with this disease, r_2 is the number of patients that are genetically similar to the target d and *do not* have the disease.

These two values are homomorphically computed as follows.

1. Similar to the filtering method in "Filtering" section, the cloud computes a predicate **p** for patients with this disease.

2. To count similar patients, the cloud computes the similarity score $S_{(\cdot)}(i,d)$ between the target d and patient i. Then the cloud homomorphically checks if $S_{(\cdot)}(i,d)$ is greater than the pre-determined threshold t, which is done by evaluating the interpolation polynomial of degree $\mathrm{Range}(S_{(\cdot)}(i,d)) - 1$. In our implementation, we use the Paterson-Stockmeyer method [59], a well-established technique [60–63], to evaluate polynomials efficiently. As such, we get a predicate $\mathbf{p}_s$.
3. Multiplying the two predicates $\mathbf{p}$ and $\mathbf{p}_s$ component-wise realizes the `AND` relation and leads to another vector, whose vertical aggregation gives r_1.
4. Multiplying the inverse predicate $\neg\mathbf{p}$ and predicate $\mathbf{p}_s$ component-wise realizes the `AND` relation and leads to another vector, whose vertical aggregation gives r_2.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s13059-024-03447-9.

Additional file 1. Supplementary material, tables, and figures.

Additional file 2. Review history.

Acknowledgements
This study is dedicated to the memory of Dr. Thomas Lehner, whose visionary leadership made this work possible.

Review history
The review history is available as Additional file 2.

Peer review information
Veronique van den Berghe was the primary editor of this article and managed its editorial process and peer review in collaboration with the rest of the editorial team.

Authors' contributions
GG, TL, and CK conceived the study. JB, JK, and SH developed the theoretical foundation with supervision from GG. JB and SH implemented the theory. JB developed the software and analyzed the results with supervision from GG and SH. JB, JK, SH, CK, TL, and GG drafted the initial manuscript. All authors read and edited the manuscript.

Funding
This project was funded by NIH grants R00HG0110909 and R35GM147004 to GG, funding from Warren Alpert Foundation to GG and TL, a National Science Foundation GRFP fellowship to JB, and funding from by CyberSecurity Research Flanders (reference number VR20192203) to JK.

Data availability
The code for SQUiD, including the SQUiD API and CLI, is available under the GNU General Public License at https://github.com/G2Lab/SQUiD/ [64] and 10.5281/zenodo.14166727 [65]. This repository contains all benchmarking code necessary to reproduce the results presented in this paper. Instructions for running the code are provided in the Benchmarking section of the README. UK Biobank data was downloaded from https://www.ukbiobank.ac.uk/ [66] under application number 100316, and simulated data can be generated using https://github.com/G2Lab/SQUiD/. Example data is also included in the repository. The variants and their effect sizes used in the PRS calculations were download from the PGS Catalog at https://www.pgscatalog.org/ [31, 67].

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 12 March 2024 Accepted: 26 November 2024
Published online: 18 December 2024

References

1. Ginsburg GS, Phillips KA. Precision medicine: from science to value. Health Aff. 2018;37(5):694–701.
2. Ward R, Ginsburg GS. Local and global challenges in the clinical implementation of precision medicine. Genomic and precision medicine. Academic Press, 2017. 105-117
3. Acosta JN, Falcone GJ, Rajpurkar P, Topol EJ. Multimodal biomedical AI. Nature Medicine 28.9 (2022):1773–1784.
4. European Parliament. Regulation (EU) 2016/679 of the European Parliament and of the Council. 2016. https://data.europa.eu/eli/reg/2016/679/oj. Last Accessed 11 Dec 2024.
5. U S Department of Health and Human Services. Health Insurance Portability and Accountability Act. U.S. Government Printing Office; 1996.
6. Tanjo T, Kawai Y, Tokunaga K, Ogasawara O, Nagasaki M. Practical guide for managing large-scale human genome data in research. J Hum Genet. 2021;66(1):39–52.
7. U.S. data privacy laws to enter new era in 2023. Reuters. https://www.reuters.com/legal/legalindustry/us-data-privacy-laws-enter-new-era-2023-2023-01-12/. Last Accessed 11 Dec 2024.
8. Liu J, Lu YH, Koh CK. Performance analysis of arithmetic operations in homomorphic encryption. 2010. Purdue University e-Pub. https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1403&context=ecetr. Last Accessed 11 Dec 2024.
9. Introduction. https://homomorphicencryption.org/introduction/. Accessed 3 Apr 2023.
10. Sim JJ, Chan FM, Chen S, Meng Tan BH, Mi Aung KM. Achieving GWAS with homomorphic encryption. BMC Med Genomics. 2020;13(Suppl 7):90.
11. Yang M, Zhang C, Wang X, Liu X, Li S, Huang J, et al. TrustGWAS: a full-process workflow for encrypted GWAS using multi-key homomorphic encryption and pseudorandom number perturbation. Cell Syst. 2022;13(9):752-767.e6.
12. Blatt M, Gusev A, Polyakov Y, Goldwasser S. Secure large-scale genome-wide association studies using homomorphic encryption. Proc Natl Acad Sci U S A. 2020;117(21):11608–13.
13. Kim D, Son Y, Kim D, Kim A, Hong S, Cheon JH. Privacy-preserving approximate GWAS computation based on homomorphic encryption. BMC Med Genomics. 2020;13(Suppl 7):77.
14. Zhou J, Lei B, Lang H, Panaousis E, Liang K, Xiang J. Secure genotype imputation using homomorphic encryption. J Inf Secur Appl. 2023;72:103386.
15. Chan FM, Badawi AQAA, Sim JJ, Tan BHM, Sheng FC, Aung KMM. Genotype imputation with homomorphic encryption. In: Proceedings of the 6th International Conference on Biomedical Signal and Image Processing. ICBIP '21. New York: Association for Computing Machinery; 2021. pp. 9–13.
16. Gürsoy G, Chielle E, Brannon CM, Maniatakos M, Gerstein M. Privacy-preserving genotype imputation with fully homomorphic encryption. Cell Syst. 2022;13(2):173-182.e3.
17. Çetin GS, Chen H, Laine K, Lauter K, Rindal P, Xia Y. Private queries on encrypted genomic data. BMC Med Genomics. 2017;10(Suppl 2):45.
18. Wang S, Zhang Y, Dai W, Lauter K, Kim M, Tang Y, et al. HEALER: homomorphic computation of ExAct Logistic rEgRession for secure rare disease variants analysis in GWAS. Bioinformatics. 2016;32(2):211–8.
19. Sarkar E, Chielle E, Gürsoy G, Mazonka O, Gerstein M, Maniatakos M. Fast and scalable private genotype imputation using machine learning and partially homomorphic encryption. IEEE Access. 2021;9:93097–110.
20. Sidorov V, Wei EYF, Ng WK. Comprehensive performance analysis of homomorphic cryptosystems for practical data processing. arXiv preprint arXiv:2202.02960 (2022).
21. Brakerski Z. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: Safavi-Naini R, Canetti R, editors. Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings, volume 7417 of Lecture Notes in Computer Science. Springer; 2012. pp. 868–886. https://doi.org/10.1007/978-3-642-32009-5_50.
22. Fan J, Vercauteren F. Somewhat practical fully homomorphic encryption. IACR Cryptol ePrint Arch. 2012:144. http://eprint.iacr.org/2012/144. Last Accessed 11 Dec 2024.
23. Brakerski Z, Gentry C, Vaikuntanathan V. (Leveled) fully homomorphic encryption without bootstrapping. ACM Trans Comput Theory (TOCT). 2014;6(3):1–36.
24. Lyubashevsky V, Peikert C, Regev O. On ideal lattices and learning with errors over rings. In: Gilbert H, editor. Advances in Cryptology - EUROCRYPT 2010. Springer, Berlin Heidelberg: Berlin, Heidelberg; 2010. pp. 1–23.
25. Polyakov Y, Rohloff K, Sahu G, Vaikuntanathan V. Fast proxy re-encryption for publish/subscribe systems. ACM Trans Priv Secur. 2017;20(4):1–31.
26. Blaze M, Bleumer G, Strauss M. Divertible protocols and atomic proxy cryptography. In: Advances in Cryptology — EUROCRYPT'98. Springer Berlin Heidelberg; 1998. pp. 127–144.
27. Ivan A, Dodis Y. Proxy cryptography revisited. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=626ecbdfdf0f92ef306865cc28503350d2591008. Accessed 28 July 2023.
28. Ateniese G, Fu K, Green M, Hohenberger S. Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans Inf Syst Secur. 2006;9(1):1–30.
29. Smart NP, Vercauteren F. Fully homomorphic SIMD operations. 71(1):57–81. https://doi.org/10.1007/s10623-012-9720-4.
30. 1000 Genomes Project Consortium, Auton A, Brooks LD, Durbin RM, Garrison EP, Kang HM, et al. A global reference for human genetic variation. Nature. 2015;526(7571):68–74.
31. Lambert SA, Gil L, Jupp S, Ritchie SC, Xu Y, Buniello A, et al. The Polygenic Score Catalog as an open database for reproducibility and systematic evaluation. Nat Genet. 2021;53(4):420–5.
32. UK Biobank Data Access Guide at 2023. https://uk-biobank.gitbook.io/data-access-guide. Last Accessed 11 Dec 2024.
33. IBM. HElib: An implementation of homomorphic encryption (2.0.0). 2021. https://github.com/homenc/HElib. Last Accessed 11 Dec 2024.
34. Help Center. https://help.instagram.com/1631821640426723. Accessed 18 July 2023.

35. Tanigawa Y, Qian J, Venkataraman G, Justesen JM, Li R, Tibshirani R, et al. Significant sparse polygenic risk scores across 813 traits in UK Biobank. PLoS Genet. 2022;18(3):e1010105.
36. Popa RA, Redfield CMS, Zeldovich N, Balakrishnan H. CryptDB: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. SOSP '11. New York: Association for Computing Machinery; 2011. pp. 85–100.
37. Kim M, Lauter K. Private genome analysis through homomorphic encryption. BMC Med Inform Decis Mak. 2015;15(5):S3. https://doi.org/10.1186/1472-6947-15-S5-S3.
38. Salem A, Berrang P, Humbert M, Backes M. Privacy-preserving similar patient queries for combined biomedical data. Proc Priv Enhancing Technol. 2019;2019(1):47–67.
39. Mouchet C, Troncoso-Pastoriza J, Bossuat JP, Hubaux JP. Multiparty homomorphic encryption from ring-learning-with-errors. Proc Priv Enhancing Technol. 2021;4:291–311.
40. Grubbs P, Ristenpart T, Shmatikov V. Why Your Encrypted Database Is Not Secure. In: Fedorova A, Warfield A, Beschastnikh I, Agarwal R, editors. Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS 2017, Whistler, BC, Canada, May 8-10, 2017. ACM; 2017. pp. 162–168. https://doi.org/10.1145/3102980.3103007.
41. Alves PGMR, Aranha DF. A framework for searching encrypted databases. J Internet Serv Appl. 2018;9(1):1:1–1:18. https://doi.org/10.1186/S13174-017-0073-0.
42. Albrecht MR, Player R, Scott S. On the concrete hardness of learning with errors. J Math Cryptol. 2015;9(3):169–203.
43. Fuller B, Varia M, Yerukhimovich A, Shen E, Hamlin A, Gadepally V, et al. Sok: Cryptographically protected database search. In: 2017 IEEE Symposium on Security and Privacy (SP). IEEE; 2017. pp. 172–91.
44. Islam MS, Kuzu M, Kantarcioglu M. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: Ndss, vol. 20. Citeseer; 2012. pp. 12.
45. Liu C, Zhu L, Wang M, Tan Ya. Search pattern leakage in searchable encryption: attacks and new construction. Inf Sci. 2014;265:176–188.
46. Oya S, Kerschbaum F. Hiding the access pattern is not enough: exploiting search pattern leakage in searchable encryption. 30th USENIX security symposium (USENIX Security 21). 2021.
47. Agrawal R, Kiernan J, Srikant R, Xu Y. Order preserving encryption for numeric data. Proceedings of the 2004 ACM SIGMOD international conference on Management of data. 2004.
48. Lewi K, Wu DJ. Order-revealing encryption: new constructions, applications, and lower bounds. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016. pp. 1167–1178.
49. Curtmola R, Garay J, Kamara S, Ostrovsky R. Searchable symmetric encryption: improved definitions and efficient constructions. Proceedings of the 13th ACM conference on Computer and communications security. 2006. pp. 79–88.
50. Song DX, Wagner D, Perrig A. Practical techniques for searches on encrypted data. In: Proceeding 2000 IEEE symposium on security and privacy. S &P 2000. IEEE; 2000. pp. 44–55.
51. Kamara S, Kati A, Moataz T, Schneider T, Treiber A, Yonli M, SoK: cryptanalysis of encrypted search with LEAKER-a framework for LEakage AttacK Evaluation on Real-world data. In: 2022 IEEE 7th European Symposium on Security and Privacy (EuroS &P). IEEE; 2022. pp. 90–108.
52. Grubbs P, McPherson R, Naveed M, Ristenpart T, Shmatikov V. Breaking web applications built on top of encrypted data. In: Weippl ER, Katzenbeisser S, Kruegel C, Myers AC, Halevi S, editors. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. ACM; 2016. pp. 1353–1364. https://doi.org/10.1145/2976749.2978351.
53. Gentry C. A fully homomorphic encryption scheme. Stanford University; 2009.
54. Lyubashevsky V, Peikert C, Regev O. On ideal lattices and learning with errors over rings. J ACM. 2013;60(6):1–35.
55. Chen H, Han K. Homomorphic lower digits removal and improved FHE bootstrapping. In: Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I. Springer; 2018. pp. 315–337.
56. Halevi S, Shoup V. Bootstrapping for helib. J Cryptol. 2021;34(1):7.
57. Smart NP, Vercauteren F. Fully homomorphic SIMD operations. Des Codes Crypt. 2014;71:57–81.
58. Kim M, Lee HT, Ling S, Ren SQ, Tan BHM, Wang H. Better security for queries on encrypted databases. 2016. Cryptology ePrint Archive, Paper 2016/470. https://eprint.iacr.org/2016/470. Last Accessed 11 Dec 2024.
59. Paterson M, Stockmeyer L. On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM J Comput. 1973;03(2):60–6. https://doi.org/10.1137/0202007.
60. Chen H, Chillotti I, Song Y. Improved Bootstrapping for Approximate Homomorphic Encryption. In: Ishai Y, Rijmen V, editors. Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II. vol. 11477 of Lecture Notes in Computer Science. Springer; 2019. pp. 34–54. https://doi.org/10.1007/978-3-030-17656-3_2.
61. Cheon JH, Kim D, Kim D. Efficient Homomorphic Comparison Methods with Optimal Complexity. In: Moriai S, Wang H, editors. Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II. vol. 12492 of Lecture Notes in Computer Science. Springer; 2020. pp. 221–256. https://doi.org/10.1007/978-3-030-64834-3_8.
62. Cong K, Moreno RC, da Gama MB, Dai W, Iliashenko I, Laine K, et al. Labeled PSI from homomorphic encryption with reduced computation and communication. In: Kim Y, Kim J, Vigna G, Shi E, editors. CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021. ACM; 2021. pp. 1135–1150. https://doi.org/10.1145/3460120.3484760.
63. Iliashenko I, Zucca V. Faster homomorphic comparison operations for BGV and BFV. Proc Priv Enhancing Technol. 2021;2021(3):246–64.
64. Blindenbach J, Kang J, Hong S, Karam C, Lehner T, Gürsoy G. SQUiD. GitHub; 2024. https://github.com/g2lab/squid. Last Accessed 11 Dec 2024.

65. Blindenbach J, Kang J, Hong S, Karam C, Lehner T, Gürsoy G. SQUiD. Zenodo. 2024. https://doi.org/10.5281/zenodo.14166727.
66. Bycroft C, Freeman C, Petkova D, Band G, Elliott LT, Sharp K, et al. The UK Biobank resource with deep phenotyping and genomic data. Nature. 2018;562(7726):203–9. https://doi.org/10.1038/s41586-018-0579-z.
67. Lambert SA, Wingfield B, Gibson JT, Gil L, Ramachandran S, Yvon F, et al. Enhancing the Polygenic Score Catalog with tools for score calculation and ancestry normalization. Nat Genet. 2024. https://doi.org/10.1038/s41588-024-01937-x.

Publisher's Note