A Privacy-preserving Query System Model using Fully Homomorphic Encryption

medicine side effect query system as an example

Yusheng JIANG†, Tamotsu NOGUCHI††, Nobuyuki KANNO†††, and Hayato YAMANA††††

† Department of Computer Science and Communication Engineering, Graduate School of Fundamental Science and Engineering, Waseda University
11F-06, Building 51, 3-4-1 Okubo, Shinjuku-ku, 169-8555 Tokyo, Japan
†† Bioinformatics Laboratory, Department of Mathematical Sciences, Pharmacology Education Research Center, Meiji Pharmaceutical University
2-522-1 Noshio, Kiyose-shi, 204-8588 Tokyo, Japan
††† Regional Medicine Laboratory, Department of Clinical Pharmacy, Pharmacology Education Research Center, Meiji Pharmaceutical University
2-522-1 Noshio, Kiyose-shi, 204-8588 Tokyo, Japan
†††† Department of Computer Science and Communication Engineering, School of Fundamental Science and Engineering, Faculty of Science and Engineering, Waseda University
11F-05, Building 51, 3-4-1 Okubo, Shinjuku-ku, 169-8555 Tokyo, Japan
E-mail: †{amadeus,yamana}@yama.info.waseda.ac.jp, ††{noguchit,nkanno}@my-pharm.ac.jp

Abstract Privacy preservation during a search has become a serious problem in recent years. There is a need to make sure that user queries that contain sensitive private information are not abused or misused by a third party, including the search provider. One way to conduct a privacy-preserving search is to encrypt the user queries. However, traditional encryption methods are only capable of protecting user privacy during a data transfer because the query itself is decrypted at the query server. A query server is usually provided by an untrustworthy cloud provider, and the exposure of data may therefore lead to the risk of a data breach. Fully homomorphic encryption (FHE), being capable of conducting addition and multiplication over a ciphertext, naturally provides a solution to this problem. Using FHE, the privacy of both the user queries and the database of the search provider can be protected. In this paper, we propose a privacy-preserving query system model. We implemented the proposed model on a real-world medicine side-effect query system. We applied a filtering prior to the query to reduce the size of the database and used multi-threading to accelerate the search. The system was tested 10,000 times with a random query over a database of 40,000 records of simulation data and completed 99.71% of the queries within 60 s, proving the real-world application of our system.

Key words privacy-preserving query, fully homomorphic encryption, secured query system

1 Introduction

As a quick and easy way to extract certain information from the Internet, a search process is an important Internet service. As improvements to search algorithms [1] and an upscaling of hardware features reach a turning point [2], search service providers are now turning their attention to improving the search experience of users. One important aspect of user satisfaction regarding a search is whether the private information of the users is carefully protected and only used for query-related purposes. Security issues in many global companies including Facebook [3] and Google [4] are raising awareness regarding the importance of personal information security.

Because users want to hide their query content from leaking to a third party or prevent misuse, they prefer to conduct a "search in local ," namely, they would like to access an entire database of a search provider, and choose the information they need. However, search providers hold their databases as important assets, and do not want users to have full access. Instead, they only want to show users a small portion [5]. A dilemma occurs between protecting the privacy of users

and "database's privacy." Thus, as a compromise, search providers allow users to encrypt their query contents. However, this can only protect user data from leakage during a transfer. Search providers, however, hold a secret key that still can decrypt the contents of a user query. The contents of user queries remain under the threat of leakage or misuse. A third party having access to the database of a search provider, e.g., a cloud server provider, can access the contents of a user query in decrypted form. In addition, untrustworthy search providers may use such contents in ways other than the intended query, e.g., selling the query history of the users to a third party.

In Japan, which is a quickly aging society [6], medical care is an important topic. According to official data [7], over 3.5 million elderly people over 65 years in age receive outpatient treatment each year, and more than 0.9 million received hospitalization treatment in 2014. When patients suffer from side effects from a particular medicine, they usually ask pharmacists for help. Although referring to the attached documentation of a medicine is a way to find the potential ingredient causing a side effect, situations exist in which the attached documentation is insufficiently comprehensive. As a supplementary empirical method, pharmacists may refer to the medicinal history of other patients and find similar situations, i.e., patients with the same gender, a similar age, and who are taking at least one similar medicine and are suffering from at least one similar symptom.

It is the responsibility of pharmacies to protect the medicinal history of their customers from a third party because such information is private and sensitive. Our goal is to develop a scheme to ensure that the queried data and full medicinal history of the patient remain safe during the entire query process. In this paper, we first introduce basic knowledge related with fully homomorphic encryption (FHE), followed by a brief review of related studies. We then describe our system using detailed algorithms, and show the experiment results.

2 Preliminaries

2.1 Fully homomorphic encryption (FHE)

Under traditional encryption methods, decryption must be performed prior to the calculations. Direct calculations over a ciphertext are not supported. However, fully homomorphic encryption (FHE) supports an arbitrary number of additions and multiplications directly over a ciphertext.

$$\begin{aligned} FHE.Dec(FHE.Enc(a) \oplus FHE.Enc(b)) &= a + b \\ FHE.Dec(FHE.Enc(a) \otimes FHE.Enc(b)) &= a \times b \end{aligned} \quad (1)$$

In (1), $a, b \in \mathbb{Z}$, $FHE.Dec$ and $FHE.Enc$ are methods used in FHE, and $\oplus$ and $\otimes$ are additions and multiplications over a ciphertext in FHE.

The idea of homomorphic encryption (HE) was first introduced in 1978 by Rivest el al. [8]. In 2009, Gentry introduced a practical scheme for FHE implementation on arbitrary functions and a number of operations using an ideal lattice [9]. In 2012, Brakerski, Gentry, and Vaikuntanathan introduced the BGV scheme using ring-learning with errors (RLWE) [10]. In 2010, Smart and Vercauteren introduced there SV packing technique to FHE and enabled a single instruction/multiple data (SIMD) style operation to realize a speed-up and compression [11]. IBM Research released an open-source library of HElib based on the BGV scheme, which is one of the most popular FHE libraries available [12]. Herein, we briefly introduce the FHE scheme and SV packing.

2.2 FHE scheme

In the FHE scheme, the following algorithms are included [10]:

- $FHE.SetUp(1^\lambda)$: Given a security parameter λ, output a set of parameters for encryption, $params$.
- $FHE.KeyGen(params)$: Generate a public/secret key pair, pk/sk, and evaluation key, ek.
- $FHE.Enc(pk, m)$: Given a public key pk and plaintext message m, output the corresponding ciphertext, c.
- $FHE.Dec(sk, c)$: Given a secret key sk and ciphertext c, output the corresponding plaintext message, m.
- $FHE.Eval(ek, f, (c_1, ..., c_t))$: Given evaluation key ek, an arithmetic circuit f that accepts t parameters, and t ciphertext c_1 to c_t, output a ciphertext as an encrypted calculation result, c_f.

2.3 SV packing and SIMD-style calculation

In [13], Smart and Vercauteren introduced a packing method that allows encrypting a vector of plaintext into a single ciphertext. These multiple plaintexts are stored separately in spaces called *slot*s. The authors introduced a SIMD-style operation on the packed ciphertext. In a SIMD-style FHE calculation on ciphertexts, operations such as additions and multiplications are conducted in a *slot*-wise manner. In our system, all operations over a ciphertext are applied in a SIMD manner.

3 Related works

A private information retrieval (PIR) scheme was first introduced by Chor et al. in 1995 [14]. Kushilevitz and Ostrovsky constructed the first single-database PIR in 1997 [15]. Since then, PIR has become one of the most important cryptography schemes available [16].

The original work by Kushilevitz and Ostrovsky [15] has already resulted in the introduction of a PIR protocol upon homomorphic encryption. Based on Gentry's improvement

of homomorphic encryption [9], the implementation of PIR based on the HE scheme has become a reality [17], [18]. A set of implementations of PIR using the HE scheme proved its usefulness in various fields including cloud computing [19], chemical compound management [5], data aggregation [20], and e-voting [21]. However, the schemes applied in these studies are not suitable for use in our system. [20] and [5] use additional homomorphic encryption (AHE), whereas [19] uses BGN homomorphic encryption and [21] uses ElGamal homomorphic encryption, which differ from FHE. These methods are effective for specific types of arithmetic circuits, but they are not suitable for our situation, in which a ciphertext and plaintext are contained inside query content at the same time. Based on [18], because FHE is theoretically suitable for any arbitrary arithmetic circuit, we consider it to be a more suitable way to solve our problem. In [22], Dong et al. introduced a general PIR scheme using FHE, although there is still space for optimization for our specific problem.

4 Proposed method

4.1 Goal

Our goal is to provide a privacy-preserving scheme to extract information that can support pharmacists in finding the reason for the occurrence of side effects based on the medicinal history of the patient. To be specific, we need records of patients with the following conditions for a comparison of the query content:

- Gender: same
- Age: R years younger – R years older, $R \in \mathbb{Z}$
- Medicine: At least one the same
- Side Effect: At least one the same

Sensitive information includes the gender and age information of the patients, as well as other private information. Such information must be kept encrypted during the entire query process.

4.2 System

4.2.1 Overview

For a simple description, we assume that we have only three participants in our model: the *cloud*, *server*, and *terminal device*. Here, the *cloud* is an untrustworthy cloud server conducting FHE calculations, provided by third-party cloud service providers, which holds the medicinal history database of encrypted users; the *server* represents the branch servers stored in each branch of a pharmacy, which are considered to be trustworthy; and the *terminal device* represents untrustworthy user devices that send a query, i.e., tablet PCs, which are assumed to be used by pharmacists when they interact with patients. Relations between these participants are shown in Figure 1.

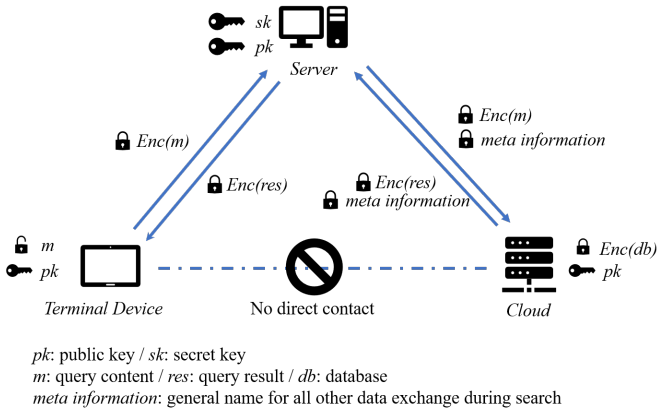

Figure 1 Relation between participants

4.2.2 Privacy

With our system, we plan to use PIR on FHE over the full medicinal history database of patients. A key pair was previously generated and distributed to each participant, as shown in Figure 1. The key pair pk/sk is used for a certain time period (e.g., a month) and renewed periodically. If there are multiple users, they will share the same sk.

Each party only has access to certain content. With our system, the *server* has access to the key pair pk/sk and evaluation key ek; the *terminal device* has access to plaintext query content m, a public key pk, and an evaluation key ek; and *cloud* has access to the ciphertext database $Enc(db)$, public key pk, and evaluation key ek.

Plaintext content m only exists inside the *terminal device*, and db is not accessible to any party, ensuring the privacy of the user's query and database.

4.2.3 Procedure

Before a query, the following preparation steps are conducted. In step 1, the parameters for the encryption environment are prepared using the *server*. In step 2, the database holder prepares a fully encrypted database $Enc(db)$ and sends it to the *cloud*.

（1） step 1: *server* generate a public/secret key pair pk/sk, and evaluation key ek, and share pk and ek with the *cloud* and *terminal device* sides.

（2） step 2: *cloud* receive the entire encrypted database $Enc(db)$ offered by the database holder using pk for encryption.

During the search process, the following steps are conducted. In step 1, *terminal device* prepares the encrypted query content, which is also shown in Algorithm 1. In step 2, a comparison of m and db is conducted using SIMD subtraction, and *slot*s with zero as the subtraction result points to the query result. To make sure the contents of the non-zero *slot*s are not recovered, the subtraction result is multiplied with a non-zero random integer r. This is described in Algo-

rithm 4. In steps 3 to 5, the *server* decrypts the result, finds the zeros, and reports their *index* to the *cloud*, whereas the *cloud* extracts these records accordingly, as shown in Algorithm 5.

（1） step 1: *terminal device* encrypt the query content m using pk, and send the encrypted query content $Enc(m)$ to the *server* side. Then, *server* send this information to the *cloud* side.

（2） step 2: *cloud* perform a SIMD-style subtraction between $Enc(m)$ and $Enc(db)$, and then multiply the subtraction result with a non-zero random integer r. Then, *cloud* send the resulting ciphertext $(Enc(m) \ominus Enc(db)) \otimes r$ to *server*.

（3） step 3: *server* use sk to decrypt the resulting ciphertext $(Enc(m) \ominus Enc(db)) \otimes r$ to gain the plaintext $(m - db) \times r$. *server* then search for *slot*s with zeros, and gather the *index* of these slots. *server* then share the *index* with the *cloud* side.

（4） step 4: *cloud* refer to $Enc(db)$ according to *indexes*, access these parts and pack them as $Enc(db')$, then, send them to the *server* side.

（5） step 5: *server* use sk to decrypt $Enc(db')$ to gain plaintext db', and share them with the *terminal device* side.

4.3 Optimization

Query content m contains the following four types of data: age, gender, list of medicines, and list of side effects. Here, we encrypt only age and gender to avoid a privacy leakage. The lists of medicines and side effects are handled in plaintext format. Note that all data, mainly, the encrypted age, encrypted gender, and the other two plaintexts, are transferred over encrypted channels, such as AES, to keep them secured during their transfer. We separate m into two parts:

- FHE-encrypted part: age and gender
- non FHE-encrypted part: lists of medicines and side effects

Because the lists of medicines and side effects are not encrypted by the FHE, the *cloud* contains them in plaintext. Thus, we can apply a filtering at the *cloud* before the FHE calculation, reducing the size of the database for calculation purposes. We apply the inverted index method commonly used in search engines to conduct the filtering step [23], as shown in Algorithm 2.

Because the *cloud* is equipped with multiple CPU cores, we use an NTL threading pool [24] in the *cloud* side when conducting the FHE calculations. In addition, as shown in Algorithm 3, SV packing is used.

4.4 Algorithm

The implementation of the *cloud* side contains the main part of our scheme. The *cloud* will receive query content from the *server* prepared using Algorithm 1. Herein, we use m' as the grouped parameters from *gender* and *age*, combining the FHE-related parameters into only a single parameter. This grouping is supported according to [25], provided that each possible pair of (*age*, *gender*) for humans will map to a unique m' within $[0, 255]$, when the upper limit of the age difference is $R \leqslant 5$. Males are mapped to $[0, 127]$, and females are mapped to $[128, 255]$.

Algorithm 1 *QueryGen*

Input $age, (Med_1, ..., Med_n), (Side_1, ..., Side_m)$
Output *query*

if *gender* is male **then**
  $m' \leftarrow age + R$
else
  $m' \leftarrow age + 128 + R$
end if
$c \leftarrow FHE.Enc(pk, m')$
$query \leftarrow c, (Med_1, ..., Med_n), (Side_1, ..., Side_m)$

After receiving the query contents c, $(Med_1, ..., Med_n)$, $(Side_1, ..., Side_m)$ from the *server*, *cloud* is initiated to apply filtering with an inverted index of the medicine $InvMed$ and side effect $InvSide$, and a smaller part is extracted from db for the FHE calculation. The classical pyramidal merge algorithm used in mergesort [26] will be operated over inverted indexes, making a smaller encrypted database $Enc(db')$ of size s for the FHE calculation. The filtering is shown in Algorithm 2. With this algorithm, $Intersect(List)$ takes the intersection of several sets, and $Union(List)$ takes the union of several sets.

Algorithm 2 *Filter*

Input $(Med_1, ..., Med_n), (Side_1, ..., Side_m)$
Output $Enc(db')$

$MergeListMed \leftarrow []$
for $i \leftarrow 1$ **to** n **do**
  $MergeListMed.Append(InvMed[Med_i])$
end for ▷ $MergeListMed$ stores inverted indeces for inputted medicines
$ResMed \leftarrow Intersect(MergeListMed)$
$MergeListSide \leftarrow []$
for $i \leftarrow 1$ **to** m **do**
  $MergeListSide.Append(InvSide[Side_i])$
end for ▷ $MergeListMed$ stores inverted indeces for inputted side effects
$ResSide \leftarrow Intersect(MergeListSide)$
$Res \leftarrow Union([ResMed, ResSide])$ ▷ Res is a list of record IDs
$Enc(db') \leftarrow []$
for i in Res **do**
  $Enc(db').Append(Enc(db)[i])$
end for

Because we use the SIMD method for calculation, we use a HElib *EncrypedArray* class to pack a vector of ciphertext $Enc(db')$ into a single ciphertext $PackDB$. This is shown in Algorithm 3.

Algorithm 3 *SVPack*

Input $Enc(db')$
Output $PackDB$

$PackDB \leftarrow Enc(pk, 0)$
for $i \leftarrow 1$ **to** s **do**
  $l \leftarrow$ all-zero integer list of size s
  $l[i] \leftarrow 1$ ▷ l act as a position indicator showing which *slot* to insert
  $PackDB \leftarrow PackDB \oplus l \otimes Enc(db')[i]$
end for

We can then conduct a calculation using FHE. The matching result should have the same gender as m and at an age difference within R with the age information of m; hence, the target value should be in $[m-R, m+R]$ inside the original partial database db'. After the SIMD subtraction of $PackDB$ and c, we need to add it with $[-R, +R]$ and obtain $2R+1$ ciphertexts $MetaRes$. We can SIMD multiply these $2R+1$ ciphertexts into one ciphertext $Enc(res)$ and send it back to the *server* for decryption, as shown in Algorithm 4.

Algorithm 4 *FHECalculate*

Input $PackDB, c, R$
Output $Enc(res)$

$MetaRes \leftarrow []$
for $i \leftarrow -R$ **to** $+R$ **do**
  $MetaRes.Append(PackDB \ominus c \oplus i)$
end for ▷ $MetaRes$ stores $2R+1$ ciphertexts
while $len(MetaRes) > 1$ **do**
  $i \leftarrow 1, j \leftarrow len(MetaRes)$
  while $i < j$ **do**
    $MetaRes[i] \leftarrow MetaRes[i] \otimes MetaRes[j]$
    $MetaRes.Remove(MetaRes[j])$
    $i+=1, j-=1$
  end while
end while ▷ Calculate product of sequences
$Enc(res) \leftarrow MetaRes[1] \otimes random()$

After receiving ciphertext $Enc(res)$, the *server* will apply a decryption and request additional information, e.g., pharmacy instructions, as shown by Algorithm 5.

Algorithm 5 *Decrypt*

Input $Enc(res)$
Output query result

$res \leftarrow FHE.Dec(pk, Enc(res))$
for *item* in res **do**
  if *item* is 0 **then**
    request additional information related with the *index* of *item* from *cloud*
    decrypt the responded additional information and send it to the *terminal device*
  end if
end for

5 Experiment and result

5.1 Experiment

We implemented the scheme in C++ using HElib. We set up our *server* on a desktop PC and the *cloud* on a cloud computing platform provided by Nifty Cloud. The *server* was equipped with an 8-core CPU and 16 GB of memory. The *cloud* was equipped with 28 virtual CPUs and 256 GB of memory. To make full use of the system, the maximum thread for the NTL threading pool was set as 28. In addition, R was set as 5. The FHE parameters used in HElib were set as shown in Table 1.

Table 1 HElib parameters

m	p	r	L
12,097	257	1	11

5.2 Simulation dataset

We prepared the simulation dataset based on statistical data. We referred to official data offered by the Japan Ministry of Health, Labor, and Welfare [7], and by Statistics Japan [25]. From the statistical data, we gained the patient distribution over the different age periods and genders. In addition, we assumed 2,000 types of medicines and 100 types of side effects, distributed according to a Pareto distribution [27]. We arbitrarily assumed that each patient takes fewer than 20 different medicines and suffers from fewer than five side effects. According to these rules, we created a dataset with a size of 40,000.

5.3 Result

We conducted 10,000 uniformly random queries based on a random age, gender, list of medicines, and list of side effects between the *server* and *cloud*. The relation between the filtered data size and the query processing time is shown in Figures 2 and 3. The distribution of the filtering percentage, i.e., the percentage of filtered database size based on the original database size is shown in Table 2.

From Figure 2 and Table 2, we can see that filtering is effective in scaling down the database size. In 98.36% of all cases, the database was scaled down to less than 400 items, which is only 1% of the original database size. Using filtering, FHE calculations over the full database are not required, which saves time and memory. From Figure 3, we can observe an improvement through multithreading. We set 100 as the *slot* number in a single ciphertext in our experiment. Thus, in

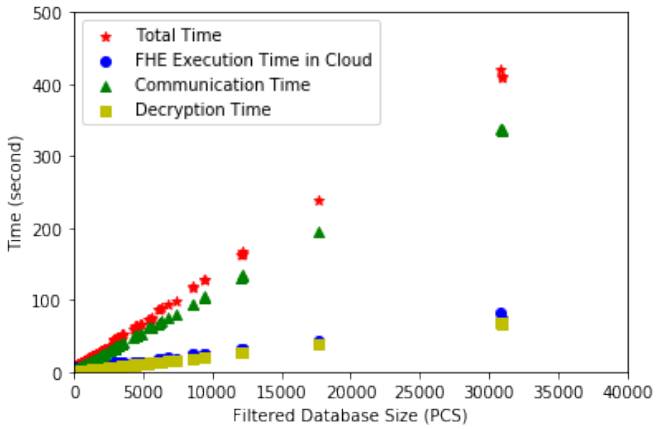

Figure 2 Relation between time consumption and filtered database size in 10,000 experiments

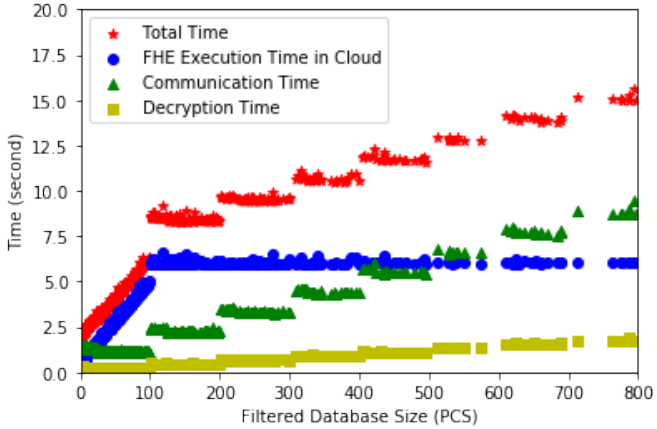

Figure 3 A closer view when filtered down to a dataset of 2.00%

Table 2 Distribution of Filter Percentage

Filter Percentage	Percentage for a Total of 10,000 Attempts
0%	60.52%
0%-1%	37.84%
1%-2%	0.57%
2%-5%	0.56%
5%-10%	0.22%
10%-20%	0.16%
20%-50%	0.10%
>50%	0.03%

Figure 3, the server calculation time, which was originally considered the most time-consuming, increases much more slowly after the filtered database size s reaches 100. As a general result, 99.71% of all queries completed the entire process, i.e., from the completed collection of m at the *terminal device* to the complete output, within 60 s.

The maximum memory usage is approximately 2.48 GB.

6 Discussion

The results of our experiment conducted on a simulation dataset show that our privacy-preserving query system model is capable of conducting a search for the side effects of a medicine.

A filtering application using an inverted index and multithreading was successful in decreasing the time and memory use. From the results of Table 2, we can see that, in most cases, only less than 1% of the database needed to be used in the calculation. If such filtering is not applied, a FHE calculation must be performed over a full database, resulting in a waste of both time and memory. From Figure 3, we can see that the server calculations require the greatest amount of time for a filtered database size s of less than 500. At greater than 500, the communication time, however, becomes the most time-consuming part. We introduced an NTL threading pool in the server calculations. When s is less than 100, the process is the same as with a single thread, and the server calculation time rapidly increases. When s is greater than 100, multithreading is applied, and the increase in the calculation time becomes much slower.

7 Conclusion

We proposed a scheme for a privacy-preserving query system, and implemented it into a real-world case. We proved the usefulness of the scheme through experiments conducted on real datasets. For future optimization, we can see that the communication time contributes significantly to the overall session time when the filtered database size is large. This is understandable because the size of $Enc(res)$ is proportional to filtered database size s. To reduce this heavy time consumption, we can consider packing more slots into a single ciphertext. However, this requires that we adjust the parameters to obtain a larger packing capacity.

Acknowledgment

This work was supported by JST CREST, Grant Number JPMJCR1503, Japan.

References

[1] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*, pages 149–190. Springer, 2016.

[2] Hassan N Khan, David A Hounshell, and Erica RH Fuchs. Science and research policy at the end of moore's law. *Nature Electronics*, 1(1):14, 2018.

[3] Phee Waterfield and Timothy Revell. Huge new facebook data leak exposed intimate details of 3m users. New Scientist, URL: https://www.newscientist.com/article/mg23831782-100-huge-new-facebook-data-leak-exposed-intimate-details-of-3m-users, May 2018. Accessed: 2019-01-05.

[4] Julia Carrie Wong and Olivia Solon. Google to shut down google+ after failing to disclose user data leak. The Guardian, URL: https://www.theguardian.com/technology/2018/oct/08/google-plus-security-breach-wall-street-journal, Oct 2018. Accessed: 2019-01-05.

[5] Kana Shimizu, Koji Nuida, Hiromi Arai, Shigeo Mitsunari,

Nuttapong Attrapadung, Michiaki Hamada, Koji Tsuda, Takatsugu Hirokawa, Jun Sakuma, Goichiro Hanaoka, et al. Privacy-preserving search for chemical compound databases. *BMC bioinformatics*, 16(18):S6, 2015.

[6] Naoko Muramatsu and Hiroko Akiyama. Japan: super-aging society preparing for the future. *The Gerontologist*, 51(4):425–432, 2011.

[7] 大臣官房統計情報部. 平成 26 年 (2014) 患者調査の概況. *https://www.mhlw.go.jp/toukei/saikin/hw/kanja/14/*, Dec 2015.

[8] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[9] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing-STOC\'09*, pages 169–169. ACM Press, 2009.

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[11] Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

[12] Shai Halevi and Victor Shoup. Algorithms in helib. In *34rd Annual International Cryptology Conference, CRYPTO 2014*. Springer Verlag, 2014.

[13] Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71(1):57–81, 2014.

[14] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.

[15] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 364–373. IEEE, 1997.

[16] Rafail Ostrovsky and William E Skeith. A survey of single-database private information retrieval: Techniques and applications. In *International Workshop on Public Key Cryptography*, pages 393–411. Springer, 2007.

[17] Vinod Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 5–16. IEEE, 2011.

[18] Xun Yi, Mohammed Golam Kaosar, Russell Paulet, and Elisa Bertino. Single-database private information retrieval from fully homomorphic encryption. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1125–1134, 2013.

[19] Yang HaiBin and Zhang Ling. A secure private information retrieval in cloud environment. In *Intelligent Networking and Collaborative Systems (INCoS), 2016 International Conference on*, pages 388–391. IEEE, 2016.

[20] Tsotsope Daniel Ramotsoela et al. *Data aggregation using homomorphic encryption in wireless sensor networks*. PhD thesis, University of Pretoria, 2015.

[21] Shubhangi S Shinde, Sonali Shukla, and DK Chitre. Secure e-voting using homomorphic technology. *International Journal of Emerging Technology and Advanced Engineering*, 3(8):203–206, 2013.

[22] Changyu Dong and Liqun Chen. A fast single server private information retrieval protocol with low communication cost. In *European Symposium on Research in Computer Security*, pages 380–399. Springer, 2014.

[23] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems (TODS)*, 23(4):453–490, 1998.

[24] Victor Shoup. Ntl: A library for doing number theory. *http://www.shoup.net/ntl/*, 2001.

[25] 総務省. 日本の統計*: 2018*. 日本統計協会, 2018.

[26] Richard Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

[27] Tudor I Oprea. Property distribution of drug-related chemical databases. *Journal of computer-aided molecular design*, 14(3):251–264, 2000.