SEARCH AND COMPUTE ON ENCRYPTED DATA

Tushar Agarwal

B.tech computer science, Rajkiya engineering college

Tushar.agarwal2905@gmail.com

Abstract- Encryption of data is done to protect sensitive data of users from malware and explore. Encrypted data is send to the cloud but before sending to the cloud the data has been protected for hackers and malicious insider. Encryption is also done so that only users can access the information only. Cloud also requires compute to that encrypted data. Suppose given a databases of a company and we need to find some information be like.

- How many employees make more than $100000?
- What is average age of the factory?

Answering to that questions cloud need to ***Search*** and ***Compute*** that encrypted data.IN this paper we will find a way or interested to find a efficient way to operation on the fully encrypted data. Our basic solution is to perform various operation on the data at a same time but this can only be done with that help of homomorphism(or fully homomorphic encryption).however we cannot be surely define the technique or no secure connection is there to fulfill the given requirement.

Through this work we have tried to construct a safe way to do private queries on ***"Search" and "Compute"*** on encrypted data. To the end of the process we have finded a way, firstly our work involves devising some underlying circuits. Secondly we provide optimize technique to improve the quality of the circuit primitive. The basic technique to accelerate their basic operation is ***SIMD.*** We rarely choose fully homomorphic encryption (FHE) to high computational complexity, so it is less favorable for the society. Finally we present our idea for various techniques for various experiments by changing various parameters, such as query type or anything.

1. Introduction

While performing any operation on the encrypted data without decrypting data we need private homomorphism. The concept was first introduced by Rivets et al. [29], and further acc. to Feigenbaum and Merritt's question affined the concept. The question was : *Is there an encryption function E(.) such that both E(x+y) and E(x.y) are easy to compute from E(x) and E(y) ?*

Since then, many mathematicians have done sustainable studies to understand and solve that problem. However after a hard work a little progress was made in determining a secure scheme until exists 2009.

Roughly speaking, Gentry's scheme allows anyone to compute E(f(x_1,…….X_n)) from a collection of encrypted data E(X1), . . . , E(Xn) for any computable function f without knowing the actual data. He called this technique a fully homomorphic encryption (FHE) scheme.

We are unable to find the complexity of the several important functions. After the performing several function, when we take our interest to the function database so some of the question arises in our mind: *Given a set of fully encrypted databases, can we construct a set of efficient functions to process queries over the encrypted databases?* If so than another question also arises what would be computational cost of a function?

Although these questions are the used for starting work for us they give us motivation to go ahead with this work. Ultimately all the perspective shows the same way i.e. fully homomorphic encrypted data.

Again all the observation leads to the natural question: *Can we construct a solution to efficiently address such a database query without maintaining multiple contexts of encryption?*

Survey

Definition

There is another to compute an encrypted data known as data flow authentication. Now order to understand the data flow authentication, we must define the overall advisory. This algothrium offer ***HASE*** and security those guaranties.

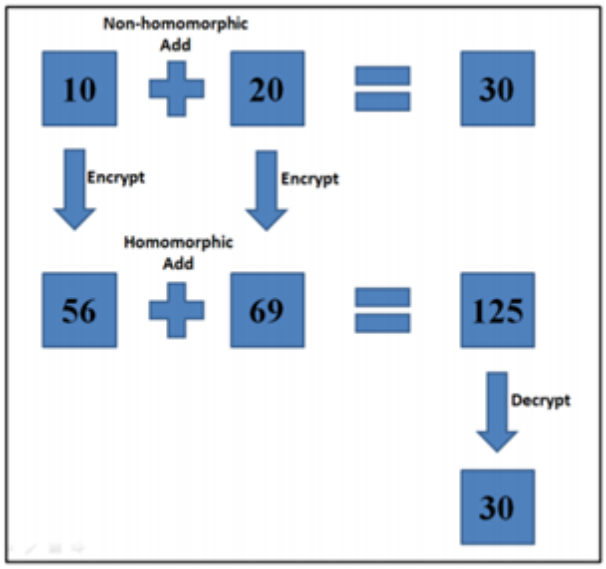

Figure: An example of homomorphic encryption

Model-

We develop or consider a scenario between trusted client and untrusted server or we can say untrusted cloud. Below given figure will remove our confusion about which scenario we are talking about. Ofcrsely Client wants to execute or compute the encrypted data. Our security object is to show or to leak that much only information which is asked not more than or we can say that can be inferred for the program executed control flow.

So to complete the process we have divided it into two parts setup and runtime. First, the client chooses the key for the encryption and enters the key in our HASE encrypted scheme (A). Then the client transform the whole data or program using HASE enabled complier and afterword's uploads it to cloud(B). Then the server analyses the data and deploys of the some part to trusted network by remote(C). This all process comes under setup.

During runtime process it up to client if he wants to use multiple inputs or data so he is free to use of its choices. The server (1) atomically encrypts the data using the

information from the complied program and afterwards send cipher texts to the cloud server (2). The cloud server now can execute the uploaded data (3). However we can assume that active advisory is controlling cloud either used by searching or computing. The adversary can be as follows.

- Read all the content provided of all the variables and program text. But remember not in trusted module.
- Can be modified content provided. But remember not in trusted module.
- Continuously observe and modify the control flow, e.g., by breaking the program, executing instructions step-by-step and modifying the instruction pointer (except in the Trusted Module).

After all this process the server or cloud return the result of the program and then client can check or verify the result of computated search.

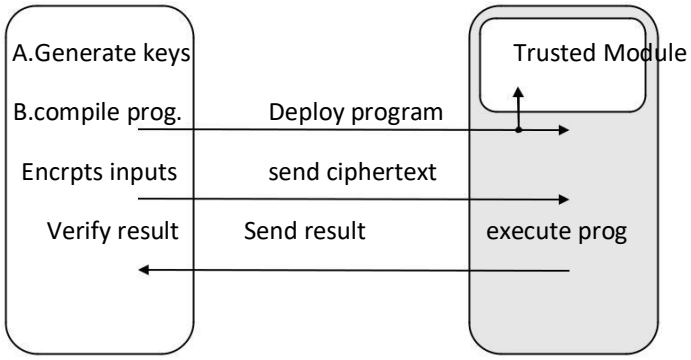

Figure: System overview

We are well aware of that fact how to perform additively and multiplicatively homomorphic operations on encrypted data. Furthermore, if we have interest to know about or to reveal the control flow of a program instead of computer computing, efficient computation seems feasible. Points to remember that any control flow decision on an encrypted variable or data is an intentionally leak by the programmer. Several proposals for program transformation into such encrypted computations have been made. MrCrypt [41], JCrypt [15] and AutoCrypt [42] all the programs offer an increasing set of programs that can be computed on encrypted data. To support encrypted computation on all programs, however, one has to convert between different homomorphic encryption schemes. These conversions are very small routines, such that we can scrutinize their code and implement them safely

in a Trusted Module likely without any software vulnerabilities. In this way we combine the benefits of partially homomorphic encryption with a small code base and the efficiency of unprotected program execution. Our re-encryption modules are small and program-independent and are run protected in the SGX enclave whereas the program runs efficiently on homomorphic encrypted values in unprotected memory. The verification of labels is constant time and does not dependent on the homomorphic computation. To this end we introduce our own authenticated homomorphic encryption scheme HASE.

Homomorphic Authenticated Symmetric Encryption (HASE)

In this section we have tried to discuss about syntax, correction and security of the HASE scheme or program we define confidentiality
In terms of in distinguish ability and authenticity in terms of
un forgeability. In distinguish ability of the HASE scheme is define as an adaption of IND-CPA security which is defined as schemataric encrypted security.

Definition- HASE scheme is a tuple of
PPT algorithms (Gen, Enc, Eval, Der, Deci) such that:

- The key-generation algorithm Gen takes the security parameter 1^p as input and outputs a key pair (ek, sk) consisting of a public evaluation key ek and a secret key sk. The evaluation key implicitly defines a commutative plaintext group (M,+), a commutative cipher text group (C,*)
- The encryption algorithm Enc takes a secret key sk, a plaintext message m € M and an identifier i € I as input and ciphertext c € C.
- The evaluation algorithm Eval takes an evaluation key ek and a set of ciphertexts C € C as input and outputs a ciphertext c 2 C.
- The deterministic label derivation algorithm Der takes a secret key sk and a set of identifiers I I as input and outputs a secret label l € L.
- The deterministic decryption algorithm Dec takes a secret key sk, a ciphertext c € C and a secret label l € L as

input and outputs a plaintext message m € M or ? on decryption error.

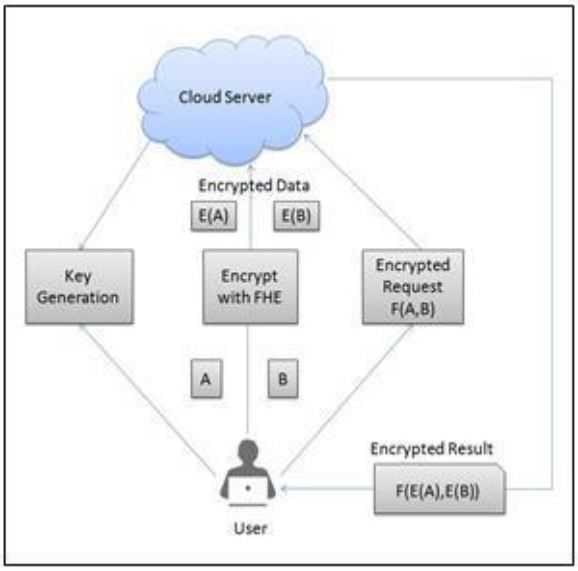

Figure : Applying FHE to encrypted data

Theorem

Let us denote
by tr a program transformation conducted by the attacker, e.g., the transformation explained above, which inserted a binary search. Then, by construction, it holds that:
8M1;M2; tr : P(M1 # C) =L P(M2 # C)
! (tr(P))(M1 # C) =L (tr(P))(M2 # C)
In other words: disregarding the explicitly declassified information within C, also the transformed program does not
leak any additional information.

Proof by Contradiction.

Assume that Theorem did not hold.
Then there would exist a transformation tr that would cause the transformed program tr(P) to compute values in at least one low memory location despite low-equivalent inputs. But this is impossible, as any such transformation would necessarily have to insert additional PDG-edges, destroying the label computations, and hence invalidating the decryptions in our HASE encryption scheme.

Result Verification:

Note that the client can verify the result of the computation using a simple check on the variable's label – just as the conversion routine does. The result is just another variable, which albeit not being converted, can be checked for correct data flow computation. That way, a client can ensure that it receives a valid output of the program.

Literature review

In this section, we tried to present a brief overview of studies. We begin with a study on private information retrieval (PIR) primitives. PIR enables a DB user to retrieve a tuple from a DB without revealing which tuple the DB user is retrieving, and with the communication complexity lower than Op € q. However, because the DB user may learn additional bits of information in addition to the originally requested tuples, PIR does not ensure the privacy of the DB server. This issue has been resolved in [21], but in turn, a DB user is required to provide an index of tuples that they would like to obtain. Sometimes, the DB user may not have any information on the index. The next important research topic is searchable encryption (SE) . These techniques allow a DB user to encrypt and store their data on a DB server in combination with block ciphers and stream ciphers. Later, the DB user can search for a specific keyword by submitting a trapdoor without revealing keywords and original data. Generalized this into the public-key setting. Using SE as a primitive, Yang et al. proposed a scheme to privately process a conjunctive query. There are different research areas focused on realizing private query processing. Hacig¨um¨us et al. tried to support general DB queries in a private manner. Hore et al. claimed their schemes can support range queries that maintain privacy. However, they later were found to reveal the underlying data distributions. Olumofin and Goldbeg extended PIR into SQL-enabled PIR to privately process general DB queries. They focused on ensuring query privacy but did not consider the privacy of databases. Other works, such as assumed that there is a set of mutually trusted and host participants. Ge and Zdonik considered the same security model . Their scheme is, however, restricted to aggregate queries. Ada Popa et al.'s CryptDB processed general types of database queries using layers of different encryption schemes: deterministic encryption for equality condition queries, order-preserving encryption for range queries, and homomorphic encryption for aggregate queries. The disadvantage of their work is that in the long run, it downgrades to the lowest level of data privacy provided by the weakest encryption scheme. For example, it may enable one to determine the data order. Recently, TrustedDB achieved the goal by placing the DB engine and all sensitive data processing inside a secure co-processor.

References

- Google
- Homomorphic Encryption and Applications (Springer Briefs in Computer Science)
- Homomorphic Signature Schemes: A Survey
- Survey paper on computation on encrypted data using data flow authentication.
- Survey paper on compute encrypted data.

Conclusion

Securing the sensitive data or private data is a big concern in today's era. So the user or system has to encrypt the data or program before sending to the server or to the cloud. So, we have to design some technique so that we can search and compute on encrypted data. There are the techniques to encrypt the users data but fully homomorphic encryption is the best way or best solution to secure the client sensitive data and cloud computing. Because this solution or the scheme of this program allow us or to enable to perform action without decrypting the whole data. Through this paper we have tried to discuss the FHE scheme and its implement using library. The work also addresses performing various operations on encrypted data using Scarab library. Our ongoing work is focused on realization of iterative quick sort approach discussed in this paper.