Homomorphic Attribute-based Encryption: Applications to Decentralized Cloud Computing

Alex Rashduni
Class of 2023
Rutgers University
New Brunswick, NJ
ar1570@scarletmail.rutgers.edu

***Abstract*—Lattice-based cryptosystems are the cutting edge of cryptography, due to their apparent resistance to quantum attacks, high asymptotic efficiency and parallelism, and solutions to long-standing open problems in cryptography. Due to these features, lattice-based cryptography is the next step in the natural progression of cryptography, with widespread potential applications. Likewise, the characteristic-based key-policies in attribute-based cryptosystems have made then desirable for further efficiency in large-scale cloud storage systems. Therefore, this work surveys the means by which to construct a lattice-based key-policy attribute-based encryption cloud storage system, and its applications to the anonymization and analytics of confidential ciphertexts, especially in the modern regulatory paradigms provided by HIPAA and RFPA.**

***Index Terms*—attribute-based encryption, homomorphic encryption, cloud storage, distributed computing, machine learning, data regulatory policy**

I. Introduction

In distributed computing and cryptography alike, one of the most fundamental network security problems involves keeping adversaries from discovering our data. Although in a hierarchical network architecture, such as a master-client system, privileges are allocated and internal logic of the coordinator prevents unauthorized users from accessing other users' data, this principle does not transfer over to peer-to-peer networks, does not account for authentication failures on the part of the master computer, and establishes a single point of failure in the network. Thus encryption is necessary to ensure security in our network.

There may also be situations in which we would like certain categories of users to have access to certain collections of data, without the exchanging of secret keys. Although the Diffie-Hellman key exchange algorithm avoids the secure exchange of pairs of secret keys, we would like this to scale up to entire classes of users, which may grow to hundreds, thousands, and more. Thus, an attribute-based cryptosystem is necessary.

Lastly, we may want to perform computation on data for which we only have access to the ciphertexts. There are many important reasons a host of a distributed storage system would want to accomplish this, such as fraud detection, criminal activity, usage metrics, and other types of data analytics. Thus, we would like to use a homomorphically encrypted attribute-based cryptosystem in our distributed network.

In this report, I lay out a software-side architecture of a peer-to-peer cloud storage system, built around homomorphic key-policy attribute-based encryption. In doing so, I focus on the abstraction provided by data marshalling in providing client-side transparency. Afterwards, I elaborate on the technological challenges of developing such a cloud storage system, and current research on its potential applications in industry.

Section II lays out preliminary information used the in the design of this network architecture, Section III explores some of the related work in this field, Section IV lays out the network architecture itself and its potential for implementation in the real world, and Section V provides some example applications for such an architecture.

II. Preliminaries

This section lays out preliminary information which is vital for the implementation laid out in Section IV.

A. Homomorphic Encryption

Homomorphic encryption is a form of encryption in which computations can be performed on ciphertexts prior to decryption with the same result as though the computation were performed on plaintexts and then encrypted. This property of homomorphic encryption allows it to be used for distributed storage and computation while preserving privacy of the individual user uploading their data.

Furthermore, the ability to perform computation on ciphertexts allows corporations to perform data analytics on encrypted client data, alleviating some of the privacy requirements of regulators. This has major implications to large banks, social media companies, pharmaceutical companies, and hospitals, allowing companies to develop vital insights on finance, public health, and marketing without needing to compromise client privacy and security.

B. Attribute-Based Encryption

Attribute-based encryption (ABE) is a type of public-key encryption in which the secret key is dependent upon properties of the user, such as their identity or location, for example. In these cryptosystems, although data may be broadcast or breached, ciphertexts are secure so long as an adversary does not have a secret key with matching properties. This paper focuses on a specific type of ABE, called key-policy attribute-based encryption (KP-ABE). This specific cryptosystem generates secret keys based on a decision tree which defines

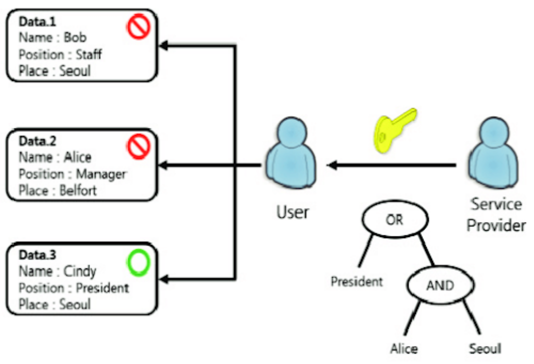

Fig. 1. An example setup of key-policy attribute-based encryption, in which a user is only able to decrypt a file according to the properties that are embedded within their secret key. The decision tree to determine the user's ability to decrypt ciphertexts is shown in the bottom right. Image credit to [1].

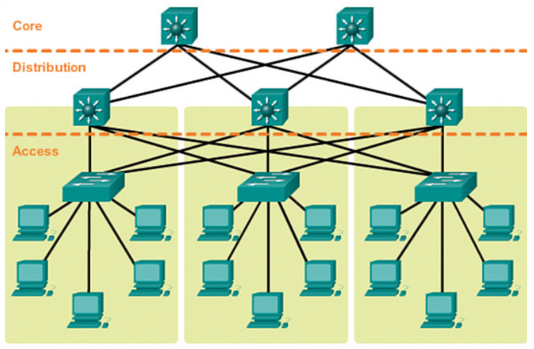

Fig. 2. An example of a multilayer hierarchical distributed system. Image credit to [2]. Note that peer-to-peer protocols within a layer can be used to mitigate single point of failure.

the privileged scope of users. Refer to Figure 1 for a visual representation.

C. Hierarchical Distributed Systems

Hierarchical distributed systems are a category of computer networks in which a coordinator machine(s) is in charge of distributing tasks and data across to worker machines. This type of architecture is commonly employed in highly-consistent distributed systems, where data needs to be kept in the most recent updated form possible. Furthermore, having a central coordinator makes it easier to handle for worker failure. Despite that, this type of distributed system comes with the inherent insecurity of a single point of failure. Therefore, without the use of replication, the entire network is taken down with the coordinator. Refer to Figure 3 for an example of a hierarchical distributed system.

In hierarchical distributed system configurations, there are often many coordinators and many different layers of coordinators, as shown in Figure 2. In these network architectures, the single point of failure can be somewhat mitigated. For further reading, please refer to [3].

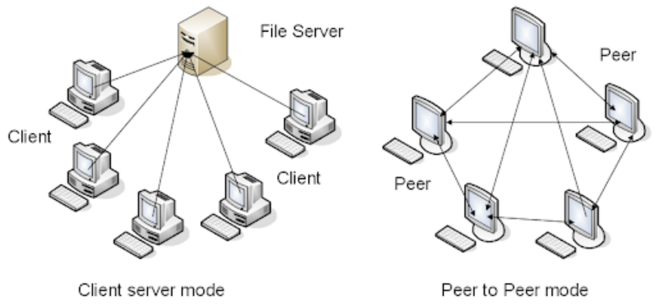

Fig. 3. An example of a hierarchical distributed system (left) and a peer-to-peer distributed system (right). Note that in peer-to-peer distributed systems, not every machine is connected, so transport protocols are needed to move packets, making the system more prone to a variety of cyber attacks. Image credit to [4].

D. Peer-to-Peer Distributed Systems

Another paradigm of distributed systems is the peer-to-peer model, which avoids the single point of failure of a central coordinator by doing away with dedicated coordinator machines. This allows computers to exchange data amongst themselves with increased partition tolerance. Despite the improved availability, the main flaw of peer-to-peer distributed systems from a security standpoint is that transport-layer protocols are required for packets to traverse the network, since there are many pairs of networks which are not connected to each other. This allows for man-in-the-middle attacks, DNS poisoning, packet sniffing, and other such cyber attacks common in networks reliant on intermediate hops in packet routing. Despite these security flaws, a strong cryptosystem, such as a lattice-based variant of KP-ABE, will mitigate the impact of these attacks.

Refer to Figure 3 for an example of a peer-to-peer distributed system. For further reading, please refer to [5].

E. Sockets

Sockets are a programming interface provided by operating systems, and are the building blocks of computer networks. Through sockets, two computers are able to interact by exchanging bits in a serialized form via data marshalling. Sockets also allow for inter-process communication in circumstances where sharing files is not feasible.

1) Data Marshalling: During a network protocol, typically data marshalling is employed to ensure that data is transferred over in a standardized form. This normally allows for the serialization of objects and other data into simple strings of bits. Some examples of data marshalling include JSON and XML.

F. Network Protocols

Network protocols are systems of rules that allow two or more processes to communicate in a standardized form by means of a socket. Without the use of protocols, most data from the client-side transmitted across the socket would be converted to "garbage," with unknown actions being performed

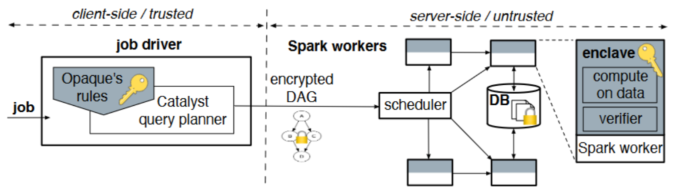

Fig. 4. The architecture for Opaque Spark. The job driver (left) encrypts jobs and sends them to the Spark schedule, who forwards tasks to the trusted enclave of workers when needed. Note that although there are many Spark workers, the database is encrypted, and only the trusted enclave can decrypt it.

on the server-side, and vice-versa. Network protocols are also notable for their ability to provide standardized rules for fault tolerance in a computer network, such as partition tolerance or robustness to garbage inputs and outputs.

G. Remote Procedure Calls

Another key principle of designing this network is remote procedure calls, which are client-server interactions in which a client-side program is able to run a procedure on the server-side computer. This is typically abstracted away from the user through the means of network protocol libraries, which allow the user to simply call the function as though it were being enacted locally. For further reading, please refer to [6].

III. Related Work

In [7], Brakerski et al describe how homomorphisms can be brought into a key-policy attribute-based cryptosystem compactly across attributes through the use of “target policies” and “targeted homomorphisms.” This scheme was shown to be secure under the hardness of learning with errors with strong correctness, as well. They also brought forth another cryptosystem that does not rely on the targeted policies and homomorphisms, and has security in the random oracle model, at the expense of greater space complexity.

Opaque Spark, [8] provides a version of Apache Spark, a SQL-like interface for data science and database management for massive (i.e., larger than RAM datasets) sets of encrypted data. In Apache Spark, data is broken into resilient distributed datasets and are spread across an entire cluster of machines while also giving the abstraction of DataFrames to the end user. This is similar to a Pandas DataFrame, but with much better runtime, and the ability to handle much larger datasets. Opaque Spark, on the other hand, uses trusted enclaves to perform plaintext-dependent operations, and then allows the remainder of its calculations to be performed over the cluster in encrypted form. This use of trusted enclaves alongside Spark allows these operations to be performed efficiently, while also preventing information leak to adversaries. This can be used in conjunction with the cloud storage platform set out in this paper for data analytics purposes. Although it relies on its own cryptosystem, it can be converted to KP-ABE. The repository is found in [9]. Refer to Figure 4 for Opaque Spark’s architecture.

Currently, there are many libraries available that have implemented homomorphic encryption and KP-ABE encryption, such as [10] and [11], respectively. Furthermore, [12], an experimental branch of PALISADE has been adding homomorphic KP-ABE to the PALISADE library, called PALISADE-ABE.

IV. Implementation

Prior to laying out the implementation details below, I lay out the reasons for designing this cloud storage platform in this particular fashion.

The primary reason for leveraging a homomorphic cryptosystem, in particular, is due to the ability to perform computations on the ciphertexts as though they were decrypted. This allows us to impose many SQL-like commands over all the data in the service and perform analytics of the ciphertexts. Another reason for homomorphic encryption is due to the potential robustness to quantum computers. These two, in conjunction, allow the cloud storage service to maintain both profitability through analytics performed on the data, as well as longevity, through long-term robustness, all while preserving the privacy of user clients.

Likewise, the reason for adopting a decentralized peer-to-peer network architecture as opposed to a hierarchical structure is due to the potential for availability problems from the small number of failure points provided by the hierarchy, as well as latency issues from non-locality between the client and server, and economic problems in scaling with a centralized data store. By allowing for decentralization, many of these problems are largely alleviated.

Lastly, although most of the applications are reliant on the homomorphic nature of the underlying cryptosystem rather than it being KP-ABE encryption in particular, KP-ABE is preferred to other lattice-based cryptosystems due to their ability to embed properties within the secret key, thus allowing entire classes of users to access a file without needing to use multiple secret keys, multiple iterations of Diffie-Hellman, throwaway secret keys, or any of the other problems natural to other homomorphic public key cryptosystems.

A. Software Architecture

The main unit of storage in this cloud storage platform is the server-side process. These processes can be hosted by computers pro bono, similar to Tor or BitTorrent, or can be hosted by universities and corporations in exchange for exclusive access over the ciphertexts and resulting studies. These server-side processes can be connected to clients by means of TCP-IP, which allows us to avoid the problems in developing our own routing protocol and simply use the one already existing throughout the Internet. Although this does mean that our packets will arrive asynchronously and potentially go missing, we can mitigate these possibilities within our network protocol.

The client-side process simply interacts with the server-side processes, and provides data by means of the below-mentioned network protocol. The degree of replication in the cloud can be determined by the client-side process by means of simply sending its request to as many servers as necessary. To prevent spam or overloading the network with small changes to the same file, timestamps and counters can be employed to determine when to delete data repositories. Another option would be to limit the client's available storage throughout the entire cloud, and use a distributed hash table to determine when they have gone above the allocated limit. The latter is appealing if spam occurs with larger files, since the client will run out of storage space through spamming; the former is preferred for smaller files, since the user is unlikely to quickly run out of space, and will siphon away bandwidth from other clients.

For our description of the network protocol, we focus on the interactions between each individual client-server pair in transferring a file. The additions of routing and spam prevention are trivial from there.

B. Network Protocol

The network protocol begins by establishing a socket between the client-side process and the server-side process. At this point, sanity checks such as handshakes and server authentication can be performed to ensure that the client and server are both functional, truthful of their identity, and not suffering from Byzantine failures, among other things. After all is said and done, the user can be granted a secret key, in accordance to the policy function of KP-ABE.

From there, the client program can perform GET and STORE operations. Likewise, DELETE operations can be made, but these are a trivial change upon GET. A simple overview of the GET and STORE operations are shown in Figures 5 and 6. Furthermore, remote procedure calls regarding metadata such as a user checking how much memory they have left available may also be implemented, as a simple variation of GET. Prior to running these functions, and in between the encryption and sending over the socket, data marshalling is employed to serialize the data to a common form.

At this point, the client-side user may send any operations they would like to perform on the server-side process. The server-side process, of course, performs authentication and sanity checks to ensure that the user truly has access to those pieces of data. As mentioned in the Introduction, KP-ABE in particular is used in this protocol to both allow groups to access the data with minimal overhead, and to allow for a failsafe in case of authentication mishaps (i.e., false positive authentications), Byzantine failures on the server-side, and other errors that may arise in this critical component.

C. Current Technological Challenges

Currently, there are many technological challenges in bringing such a system to market in the present day. First, there are no bug-free, non-experimental libraries for lattice-based attribute-based encryption. Although PALISADE-ABE has come a long way with progress in their own experimental implementation, there are still many issues to resolve with their library before a full-blown server can be implemented upon it. These come both in the form of major performance issues, and in correctness issues. From my own experiences in testing it, input data after encryption and decryption is not preserved, leading to problems in a large-scale storage setting. Despite this, because encryption is somewhat probabilistic in nature, it is indeed possible that while testing out the PALISADE KP-ABE library, I was simply unlucky. Likewise, the performance issues of KP-ABE were so severe, that I was unable to run it on a larger scale on my computer. Thus, for cloud servers, which run many client and server processes, this is currently infeasible without large dedicated clusters.

```
Require: A message, $m$, secret key, $sk$, socket $sock$.
  $ciphertext \leftarrow$ABE.Enc($m$, $sk$)
  $sock$.Send($ciphertext$)
  $k \leftarrow sock$.Receive()
  return  $k$, the index key for $ciphertext$ in the server corresponding to $sock$.
```

Fig. 5. Overview of the STORE operation, on the client side.

```
Require: An index key, $k$, public key, $pk$, socket $sock$.
  $sock$.Send($k$)
  $ciphertext \leftarrow sock$.Receive()
  $m \leftarrow$ABE.Dec($m$, $pk$)
  return  $m$, the message plaintext.
```

Fig. 6. Overview of the GET operation, on the client side.

Although there are a great variety of possible applications of this type of cloud storage system, it cannot be completely implemented at this exact time. Despite that, it would be possible to complete as soon as the required libraries are completed and issues are resolved. As a result of its strong capabilities, this is certainly a technology to look out for.

Note that although non-lattice-based cryptosystem are available through OpenABE, and that lattice-based cryptosystems are available through OpenFHE and PALISADE's base release, the former compromises both the ability to be used in the below-mentioned applications and quantum security, while the latter has the aforementioned key exchange problems.

V. Applications

The primary applications of this cloud storage platform come from the fact that homomorphic encryption allows for the computation on ciphertexts. This allows us to obfuscate information to whatever service is performing the computation, while maintaining the ability to perform it in the first place. In the case where our KP-ABE were not homomorphic, we would no longer be able to perform any computations over the data, making the ciphertexts relatively useless. Although a cloud storage service leveraging a non-homomorphic cryptosystem is certainly appealing to the clients' perspective, it does not provide nearly as many capabilities to the host, and does not allow for any studies to be performed on said data.

A. Machine Learning & Analytics

As shown in [13] and [14], the underlying mathematical properties of homomorphic encryption makes a cloud storage platform that employs it both desirable and lucrative. This is due to the fact that classification and regression models can be run over the ciphertexts of the dataset, thus allowing researchers to perform analytics over the data while preserving privacy. This is critical, as it finally provides a solution to the longstanding problem of regulations requiring privacy to be preserved across a vast array of datasets, which originally resulted in many useful datasets that could provide immense benefits going completely unused.

[13] goes into a specific architecture of how machine learning models can be trained locally on different host servers and then after each epoch, can be aggregated by a trusted third-party to prevent information leak. As shown in Figure 7, the overall network protocol for this machine learning setup involves each host server training a model on their data locally, then sending a replica of their model to a third-party parameter server, alongside gradients. The parameter server is then able to aggregate the gradients and weights via a simple linear operation, and then send the updated weights back to all the local models. In the long run, this results in convergence over the network. For a more rigorous and detailed overview of the weights adjustment, rather than the overall network protocol, please refer to Figure 8.

[14] elaborates on the potential applications of a hierarchical cloud storage platform in preserving privacy of individuals' data in a healthcare setting. In particular, Jia et al focus on the removal of identifying data from healthcare datasets, and applying the anonymized data for prediction, classification, and other analytics. The authors took on a hierarchical approach in particular due to their needs to have one authoritative server throughout the entire healthcare system and to prevent information leak to other servers. Despite the benefits this does provide in information leak prevention, the peer-to-peer layout described in this paper makes the data more available to different labs, academics, and industry, thus allowing for data science to be performed more efficiently. It is definitely worth looking into the tradeoffs, both ethically and computationally, in deciding between these two approaches; is it more ethical to be more certain no data gets lost, at the expense of data availability? Personally, I believe in strength on homomorphic encryption, and that if an adversary were truly determined, they would break into a ciphertext regardless of the information leak. Likewise, the speed at which new discoveries could be made within a dataset by the open availability of ciphertexts would benefit more lives compared to the relatively smaller number that would be disrupted if a small amount of information leak occurred, especially since information leak is not the same as plaintext leak.

VI. Conclusions

In this paper, I laid out the preliminaries, architecture, network protocol, and current challenges in implementing a cloud storage platform leveraging KP-ABE. I also laid out

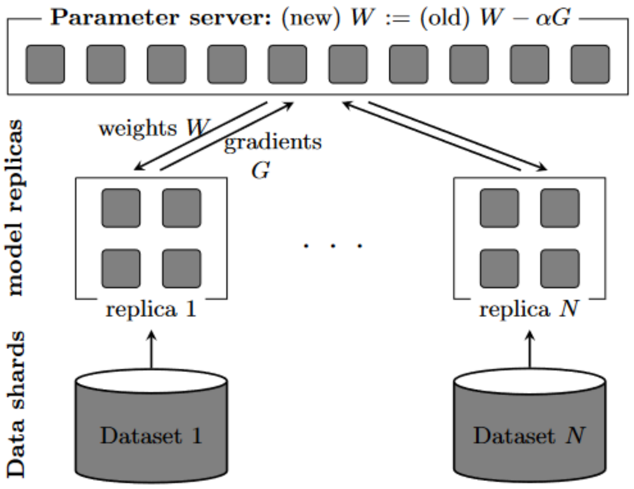

Fig. 7. Weight adjustment in the large-scale privacy-preserving machine learning setting. Image credit to [13].

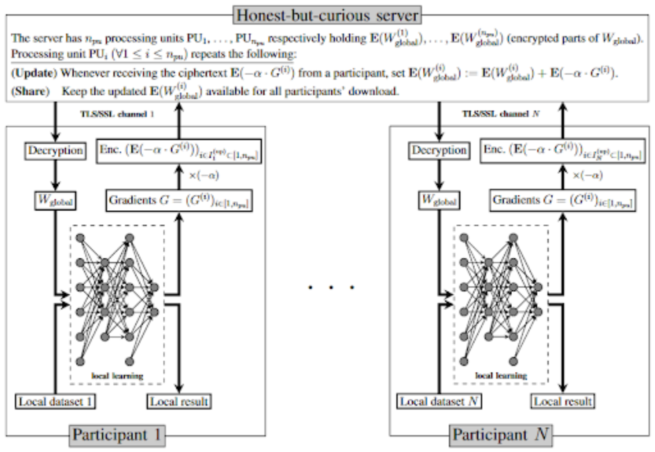
The server has n_{pu} processing units $PU_1, \ldots, PU_{n_{pu}}$ respectively holding $\mathbf{E}(W^{(1)}_{global}), \ldots, \mathbf{E}(W^{(n_{pu})}_{global})$ (encrypted parts of W_{global}).
Processing unit PU_i ($\forall 1 \leq i \leq n_{pu}$) repeats the following:
(Update) Whenever receiving the ciphertext $\mathbf{E}(-\alpha \cdot G^{(i)})$ from a participant, set $\mathbf{E}(W^{(i)}_{global}) := \mathbf{E}(W^{(i)}_{global}) + \mathbf{E}(-\alpha \cdot G^{(i)})$.
(Share) Keep the updated $\mathbf{E}(W^{(i)}_{global})$ available for all participants' download.
Honest-but-curious server
TLS/SSL channel 1
Decryption
Enc. $(\mathbf{E}(-\alpha \cdot G^{(i)}))_{i \in I_1^{(up)} \subset [1, n_{pu}]}$
$\times(-\alpha)$
W_{global}
Gradients $G = (G^{(i)})_{i \in [1, n_{pu}]}$
local learning
Local dataset 1
Local result
Participant 1
TLS/SSL channel N
Decryption
Enc. $(\mathbf{E}(-\alpha \cdot G^{(i)}))_{i \in I_N^{(up)} \subset [1, n_{pu}]}$
$\times(-\alpha)$
W_{global}
Gradients $G = (G^{(i)})_{i \in [1, n_{pu}]}$
local learning
Local dataset N
Local result
Participant N

Fig. 8. A more in-depth figure on the actions within the large-scale privacy-preserving machine learning setting. Image credit to [13].

the ramifications and applications of such a platform, and its potential benefits for both privacy preservation and data availability. This technology, when introduced, will fundamentally change the regulatory regimes around personal privacy protection, and bring forth a new gold rush of data science, especially in highly-regulated fields with immense repertoires of data, such as health, banking, and law enforcement.

References

[1] https://www.researchgate.net/figure/KP-ABE-Key-Policy-Attribute-Based-Encryption_fig1_282500797
[2] https://www.ciscopress.com/articles/article.asp?p=2202410&seqNum=4
[3] https://www.sciencedirect.com/topics/computer-science/hierarchical-architecture
[4] https://www.researchgate.net/figure/Architecture-of-C-S-and-P2P_fig1_238638509
[5] https://en.wikipedia.org/wiki/Peer-to-peer
[6] https://en.wikipedia.org/wiki/Remote_procedure_call
[7] Z. Brakerski et al, "Targeted Homomorphic Attribute Based Encryption." *Theory of Cryptography Conference*. 2016. Available: https://link.springer.com/chapter/10.1007/978-3-662-53644-5_13.

[8] Wenting Zheng, Ankur Dave, Jethro Beekman, Raluca Ada Popa, Joseph Gonzalez, and Ion Stoica. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. NSDI 2017, March 2017.

[9] https://opaque.co/blog/how-to-run-spark-sql-on-encrypted-data/.

[10] Y. Polyakov. PALISADE Release. 2022. https://gitlab.com/palisade/palisade-release.

[11] OpenABE. 2021. https://github.com/zeutro/openabe.

[12] M. Triplett. PALISADE abe. 2022. https://gitlab.com/palisade/palisade-abe.

[13] L. T. Phong et al, “Privacy-Preserving Deep Learning via Additively Homomorphic Encryption,” ATIS. *8th International Conference on Applications and Techniques in Information Security*. 2017. Available: https://ieeexplore.ieee.org/document/8241854.

[14] Q. Jia et al, “Efficient Privacy-preserving Machine Learning in Hierarchical Distributed System,” IEEE. 2018. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7970733/.