Privacy Preserving Measurement
Internet-Draft
Intended status: Standards Track
Expires: 20 October 2025

L. Li
F. Liu
Huawei

18 April 2025

Homomorphic Cryptography Protocols for Measurement Information Collection
draft-li-ppm-homomorphic-encryption-01

Abstract

Homomorphic encryption is an algorithm that allows computations to be performed on encrypted data without first having to decrypt it. This document provides a homomorphic cryptographic protocol that supports addition and multiplication in the ciphertext state. In this document, the proposed protocol can be used to protect user privacy in measurement information collection and statistics by using homomorphic encryption. And let the collector server receive the computation results without having to acknowledge the information plaintext of each individual client.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 20 October 2025.

Copyright Notice

Table of Contents

1. Introduction

In the regular process of measurement information collection and statistics, the entity that collects client data (i.e., collector server) has to learn the privacy information. (e.g., bug reports from users, visit views, etc.). However, many statistics tasks only require statistics results computed and generated from the collected information (such as using operation of addition and multiplication). This kind of data tasks is very suitable for using homomorphic cryptography protocols. Homomorphic cryptography protocols are a protocol that apply homomorphic encryption, which allows computations to be performed on encrypted data without first having to decrypt it. Some statistics tasks, such as bug statistics in a period of time and

the number of visitors to website content, can be computing without having to obtain the plaintext information of each individual client.

In the document we propose a homomorphic cryptography protocol to demonstrate how the homomorphic encryption can benefit the user privacy in terms of these tasks. The proposed protocol is parameterized and offers algorithm agility. It is worth mentioning that the specific steps of addition and multiplication operation are vary in terms of different encryption algorithms. But they can be encapsulated as an "algorithmic boxs" in a protocol. None of the algorithms are limited to the document, but we do give the recommendations algorithm in Section 4. Also, this proposed protocol can be integrated with other aggregation protocols, such as DAP [I-D.ietf-ppm-dap].

1.1. Compairson with the MPC protocol

Homomorphic encryption is a classical algorithm. It can be used as a cryptographic primitive in the MPC-based protocol. Some classic MPC protocols are based on the principle of majority honesty. These protocols usually do not require cryptographic algorithms such as homomorphic encryption. For example, VDAF [draft-irtf-cfrg-vdaf-12] are based on secret sharing. Generally, homomorphic encryption can be used in more complex security assumptions. The performance of homomorphic encryption is often facing the challenge due to algorithm complexity. But, algorithm acceleration, such as hardware acceleration, is under development in the community, and we will discuss more performance characteristics in the following section 6.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Prerequisites

Some prerequisites need to be met if homomorphic cryptography protocols are used. In general, the proposed protocol can be used in statistical and computing scenarios. But, it can also be used in the outsourcing or cloud computing. For example, the classic scenario is federated learning for AI. Generally, both the client (or the data owner) and the collector server need to be able to store and use encryption algorithms, and keys between them can be securely configured out-of-band. Because ciphertexts need to be calculated as programmed way, the format of sending ciphertexts may need to be configured through the negotiation or agreement in advance.

3.1. Lattice and Ring

The following prerequisite mainly refers to the description in the survey paper [LWE]. To create a lattice, usually we can denote dots in 2D and 3D dimension. And every two and more dots can be seen as a vector. New vectors formed by a set of existing combination of vectors is called a lattice. the set of vectors which define the denoted lattice is called the "basis" of a lattice. There are a couple of long-standing hard problems when propcessing with the lattice such as shortest vector problem, closest vector problem. And the most well-known hard problem for the cryptography is called Learning with errors (LWE). Originally LWE is from the machine learning field, in a Learning with Errors (LWE) system we take a square array A^(→). We can take a secret vector s^(→) and calculate a value vector b^(→) with following formula (the text below related to arrays, vectors, etc. are depicted with " ^(→) ".).

$$b^{(\rightarrow)} = s^{(\rightarrow)} \ A^{(\rightarrow)} + e^{(\rightarrow)}$$

where b^(→), s^(→), and e^(→) are vectors and A^(→) is an array. The value e^(→) is a small vector of error. In general, s^(→) cannot be recovered from just A^(→) and b^(→). therefore LWE has the same features like Diffie-Hellman [DIFFIE]

Moreover, to reduce the size, a common way is to replace the matrix operations with polynomial functions. Instead of array operations, b^(→) = s^(→) A^(→) + e^(→) can turn the array and vectors into a polynomial. For example, a vector: <a1, a2, a3> can map to a polynomial a_1^2 * x^2 + a_2 * x + a_3. Therefore, the equation b^(→) = s^(→) A^(→) + e^(→) can be all represented in polynomial functions in the same way and it can be used in the same protocol as with the standard LWE where s^(→) is still the private key and bs^(→) is the public key, but now they can transfer to be represented in polynomials instead of vectors and matrices. The multiplication is a standard polynomial multiply that is reduced modulo a polynomial (usually x^n + 1). This is a "ring " and therefore it is named Ring Learning with Errors (RLWE). It is worth noting the RLWE is not the same math problem of LWE, since it is mapped into polynomial fieled. It is a different hard problem.

3.2. Document Organization

The remainder of this document is organized as follows:

* Section 3 introduces the proposed homomorphic cryptography protocol.

* Section 3.2 introduces the detail architecture of the proposed protocols.

* Section 4 describes the recommended algorithm for the homomorphic cryptography protocol.

* Section 5 describes conditions of usage including how to combined with other protocols in PPM WG such as DAP

* Section 6-7 describes the analysis of performance and security characteristics

4. Protocol Definition

This section describes the proposed homomorphic cryptography protocol and how procedures, such as addition and multiplication, are performed.

4.1. Overview

The overall system architecture is shown in Figure 1.

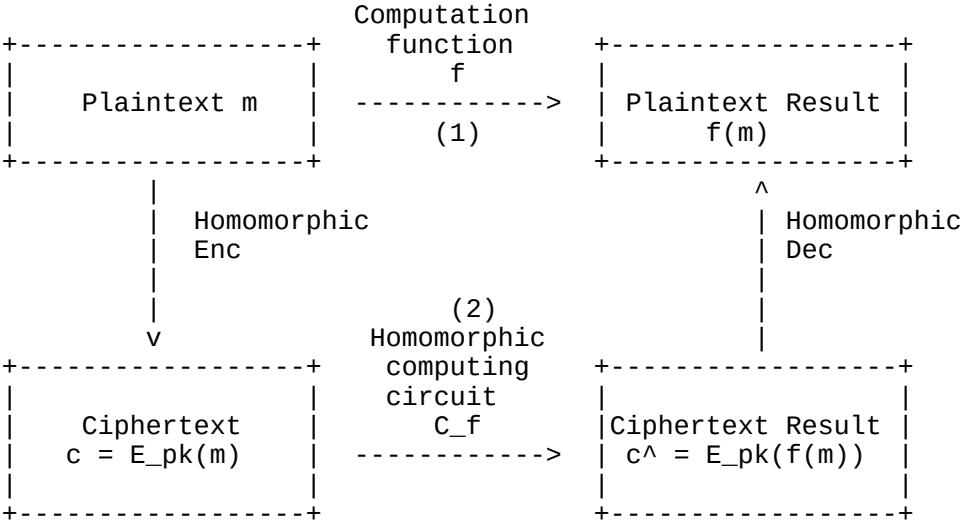

Figure 1: Overview of the Homomorphic Protocol

As a overview, the basic procedures with or without the homomorphic protocol are as follows. Without the homomorphic protocol, The server uses mode (1) to obtain the plaintext and computation functions (e.g., statistical functions include addition and multiplication) to obtain the plaintext results of computation. In

this case, the client needs to allow the original data to be exposed to the server for calculation purposes, which may contain privacy information. In the case of homomorphic cryptography protocol, the server performs the privacy computation in mode (2). First, the client (or a proxy) performs homomorphic encryption E_pk(m) on plaintext, and generate homomorphic ciphertext c. Then, an aggregator gathers the ciphertext and compute the result c^ of the ciphertext through the homomorphic computing circuit C_f and transferred to the server. After that, the server obtains the plaintext calculation result through homomorphic decryption with corresponding key. The client does not need to expose the plaintext data to the aggregator and server. The server only obtains the final result and does not need to know the data of each client.

4.2. System Architecture

The main participants in the protocol are as follows:

* Server (collector): An entity that obtains computation results and needs to obtain computation results for a task (such as statistics collection, visit views, bug reports, etc.).

* Client (data owner): An entity that sends data to be collected. The data may contain privacy information and must be provided privacy preserving during its data being used.

* Aggregator: An entity aggregates data of multiple clients for privacy computation and provides the ciphertext result to the server. In some cases, it can be a network function in the server.

The overall interaction functions of participants are defined as follows:

* HE.Keygen: Generates associated keys, usually generated by a trusted third party or related key management function for the client.

* HE.Enc: The encryption execute entity (for example, the client) uses the homomorphic encryption key to generate the ciphertext c.

* HE.Dec: The decryption execute entity (for example, the collector server) uses the homomorphic decryption key to generate the plaintext result f(m).

* HE.Eval: The execute entity (for example, the aggregator) use multiple input ciphertexts and corresponding key to generate ciphertext computation results as the output. It includes addition HE.Eval.Add, and multiplication HE.Eval.Mul.

In the following sections, the asymmetric cryptography use cases are used as examples. The detailed process of homomorphic cryptography protocols and interaction protocols between participants is in the session 3.2.1 to 3.2.5.

4.2.1. Key Generation: HE.Keygen

The key generation process is performed by a trust 3rd party as follows:

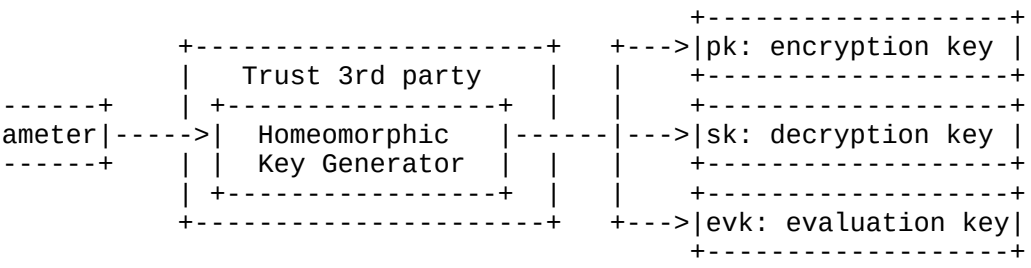

Figure 2: the process of HE.Keygen

(pk, evk, sk) ← HE.Keygen(1^n)

* pk stands for the public key, which is used to encryption in the protocol.

* evk stands for the evaluation key, which is used to computation in the protocol. One may need to obtain the evk for calculation. It is worth noting that not all computation process needs evaluation keys.

* sk stands for the private key, which is used to decryption in the protocol.

* n stands for security parameters used to generate keys.

4.2.2. Encryption Process: HE.Enc

The encryption process is performed by client as follows:

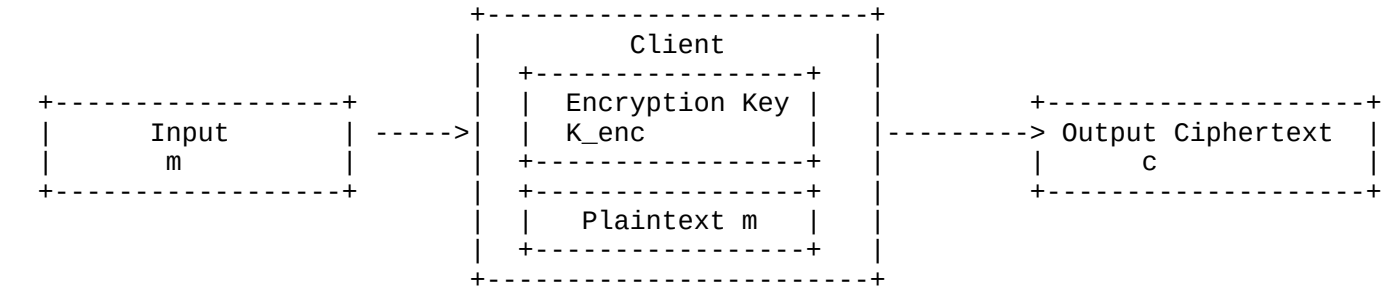

Figure 3: The procedure of Client Encryption

$c \leftarrow HE.Enc_{pk}(m)$

The client encrypt the input plaintext to the ciphertext c as the output, which using homomorphic pk as the encryption key, $m \in \{0,1\}$ and the c stands for the ciphertext.

4.2.3. Decryption Process: HE.Dec

The decryption process is performed by server as follows:

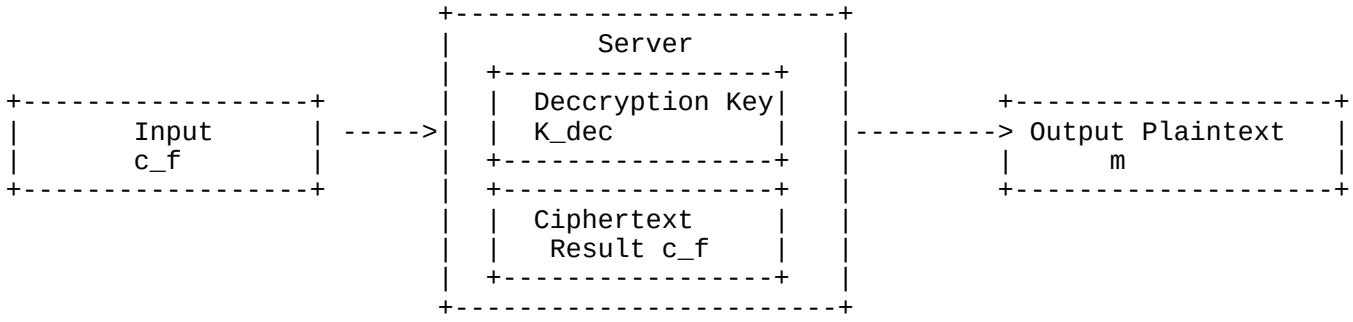

Figure 4: The procedure of Server Decryption

$m \leftarrow HE.Dec_{sk}(c)$

The server decrypt the input ciphertext c to the plaintext m as the output, which using the homomorphic sk as the decryption key, $m \in \{0,1\}$ and the c stands for the ciphertext.

4.2.4. Evaluation Process: HE.eval

The evaluation process is performed by an aggregator as follows:

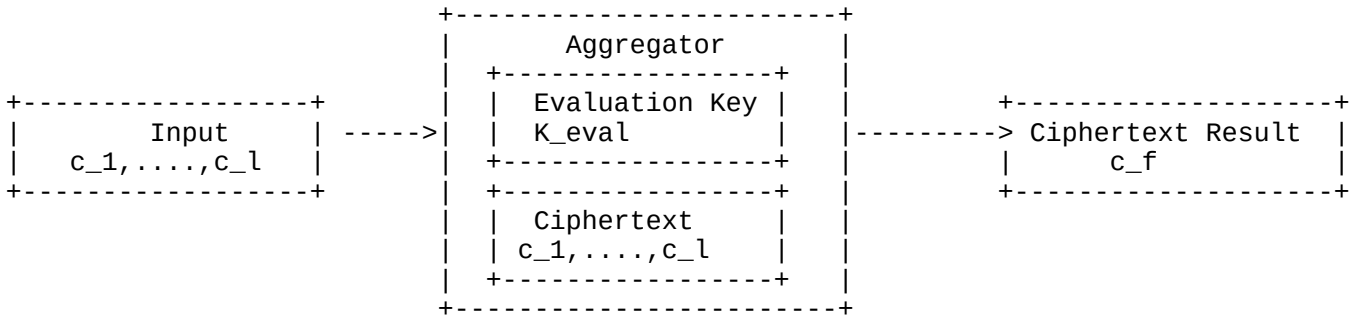

Figure 5: The procedure of aggregator homomorphic evaluation

c_f ← HE.Eval_evk(f: {0,1}^l → {0,1}, c_1 ,..., c_l)

The aggregator using the input ciphertext c_1 ,..., c_l and the evaluation key evk to process the privacy computation, which:

* f: {0,1}^l → {0,1}, stands for the computation function using the homomorphic ciphertext.

* evk stands for the evaluation key used for ciphertext computation, it is optional for some algorithms.

* c_f stands for the ciphertext computation result as the output.

It is worth noting that the HE.Eval_evk(f, c_1 ,..., c_l) can be divided into multiple basic operator. For example, operators of homomorphic addition and multiplication are demonstrated as follows:

c_add ← HE.Eval.add_evk(c_1, c_2)

c_mul ← HE.Eval.mul_evk(c_1, c_2)

In homomorphic cryptography protocols, the ciphertext computation result after the decryption is equivalent to the plaintext direct computation result. The key pk, sk, and evk generated by the trusted 3rd party needs to be configured for the corresponding server, client, and aggregator.

4.3. Fully Homomorphic Encryption (FHE)

According to the definition of homomorphic cryptography, the ciphertext computation result after the decryption is equivalent to the plaintext direct computation result. Many encryption algorithms can satisfy the definition of homomorphic encryption (HE). However, only some algorithms satisfy fully homomorphic encryption (FHE) features. In generally, single HE algorithms only satisfy one of the operations such as addition or multiplication. A classic example is that RSA algorithm satisfies only multiplicative homomorphism, namely HE.Eval.mul_evk, which leads to limited practicability for the single HE. Because both multiplication and addition are basic operators for computation f.

At present, the common FHE algorithm is basically based on the lattice construction, which is based on the Learning with Errors (LWE) or Learning with Errors over Ring (RLWE) problem in lattice cryptography. It has been widely used in lattice cryptography since then. A classical fully homomorphic encryption construction is defined as follows:

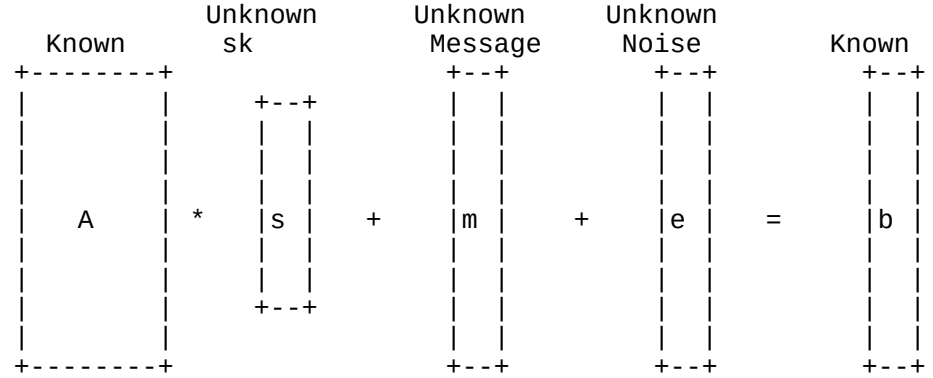

Figure 6: Fully Homomorphic Encryption and Noise

The ciphertext is denoted as c = (a, b^(→)), where b^(→) = [A^(→) * s^(→) + m^(→) + e^(→)]_q. The random vector s^(→) is used as the sk, the positive integer n stands for the dimension of the vector. q stands for the modulus of the ciphertext, and m^(→) stands for the plaintext. A ∈ Z_q ^ (N × n) stands for a uniformly distributed random matrix, where the set estimates (-q / 2 , q / 2] ∩ Z, and N > n. Noise e ∈ Chi ^ N. The noise distribution is usually Gaussian discrete.

4.4. Noise Control and Evaluation Key

Most existing fully homomorphic encryption algorithms usually add a random noise "e" during the encryption and the decryption. Therefore, the values of plaintext and ciphertext can only be approximately equal. Generally, it is acceptable. However, conducting the homomorphic computation on ciphertext may cause these noises to increase. When a homomorphic computation is performed with multiple ciphertexts, the noise contained in ciphertexts will be also expanded. In particular, multiplication causes a dramatic increase of the noise. When the total noise value is under the control, the decryption algorithm can restore the ciphertext back to the plaintext. But, if f contains too complex operators, once the noise range exceeds the critical value, the original plaintext cannot be recovered, resulting in homomorphic decryption failure.

A feasible solution was proposed in Gentry’s paper [GENTRY] in 2009, called the "Bootstrapping". the Bootstrapping is a processing for ciphertext to reduce the noise. After Bootstrapping, c = (a, b^(→)) can be "refreshed" into a new ciphertext with lower noise. A key, named as a Bootstrapping key (BSK), needs to be used in the process. In addition to bootstrapping, there is also other types of homomorphic evaluation keys, such as key switching keys (KSK). They are all designed to reduce the expansion of noise in the process of calculation and achieves multi-layer operators. Therefore, evaluation key is needed in some operators to control noise, such as the multiplication of multiple ciphertexts.

5. Recommandation of the (F)HE Algorithm

TODO

/ * present 1-2 recommended (F)HE Algorithm(s).

6. Conditions of Usage

Homomorphic Key configuration needs to be considered. Keys should be generated by a trusted third party and be configured at the granularity of tasks and for every participant. The client (or its proxy) needs to perform encryption, so the public key needs to be configured for the client. In one task, multiple clients must be configured with the same public key as the encryption key. In addition, the aggregator needs to perform evaluation process, and therefore an evaluation key (i.e., a computation key) needs to be configured. Finally, the server needs to perform a decryption operation, and therefore the associated private key needs to be configured as a decryption key. During the information collection process, the collector MUST NOT obtain the intermediate computing

data by other entities, and the aggregator itself MUST NOT be configured with a decryption key.

6.1. Discuss the Combination with DAP

The proposed homomorphic protocol can reuse the architecture of Distributed Aggregation Protocol (DAP). The homomorphic protocols also meet the security requirement of DAP. i.e., the robustness and privacy. A malicious client cannot interfere with the reports of other clients (i.e., robustness) and the aggregator cannot acknowledge each report of each client and only know the final result (i.e., privacy). In the homomorphic cryptography protocols, neither the aggregator nor the collector (i.e., the server) can read the customer’s data. To achieve this, the aggregator needs to do most of the statistics computation task and only transmits the ciphertext results to the collector server.

For comparison, there are pros of homomorphic protocols that DAPs cannot support at present. First, the homomorphic protocol does not require on-line computation. In most cases, the client only needs to report once using homomorphic encryption while other DAP algorithms may require multiple interactions between the clients and aggregators. Second, an aggregator performs homomorphic evaluation (i.e., computation) without requiring on-line joint validation with other aggregators. the asynchronous collection and evaluation can be supported. Third, the existing DAP protocol requires at least two aggregators for aggregation. While homomorphic protocols require minimal one aggregator.

But the homomorphic protocol also has limitations. First, homomorphic algorithms may need to be configured on the client side, resulting in complex algorithms. Second, depending on the content of the statistics, it may not be possible to implement multiple statistics in one round of reporting at same time, which makes the protocol unable to efficiently perform some statistics, such as building histograms, compared to other DAP algorithm e.g., Poplar1 [POPLAR]

7. Performance Characteristics

TODO

/* Parameter choice and security margins

/** **Discuss about the trade-off between attack success probability and noise complexity.

/* Communication security (authentication)

/* Protocol quantum resilience on lattice-based homomorphic encryption algorithm

8. Security Characteristics

TODO

/* Communication - number of bits sent/received between the server, aggregator and client

/* Computation - how many times aggregator finish the evaluation process.

/* Memory size requirements

/** **key generation and storage

/** **Trade-offs - time/memory on the condition of different noise settings

9. Minitues in Data Processing Security Side-Meeting in #122

There was the sidemeeting to discuss about this contribution and the Homomorphic Encryption usage, linked: https://trello.com/c/JwtMkJzN/56-1315-1445-enabling-data-security-trust-and-privacy-for-ai-in-future-network

the minitues is in the following:

About the data procssing security,

1. The meeting discussed data processing security, transparency, and privacy of AI in future networks. The specific contents include: The relationship between data infrastructure and AI: Data is the lifeblood of AI. The key to promoting AI interoperability lies in data, especially in cases where data is distributed or federated. Recommended IETF work items include Meta - data model, policy descripter, AAA, etc.

2. Emerging technologies: Some emerging cryptographic technologies, including homomorphic encryption are considered for use in future networks.

About the Homomorphic Encryption (HE) in IETF WGs,

* Advantages: Allows direct data processing (such as AI inference) on encrypted data, protecting the privacy of original data.

* Application scenarios: examples like autonomous driving of drones (encrypted data procssing on the network side).

* Constraints, challenges, and progress: Currently, the processing performance still needs to be approved, but it can be significantly improved through algorithm and hardware acceleration.

* Others: Different from the TLS protocol that focuses on transmission security, homomorphic encryption may be related to data - processing protocols, which may be a new class of data - related protocols. The performance of homomorphic algorithms still needs to be improved, but standardization can be considered in advance. Existing security technologies will affect performance but bring more commercial benefits.

10. IANA Considerations

This document has no IANA considerations.

11. References

11.1. Normative Reference

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

11.2. Informative References

[DIFFIE] Diffie, W., "Diffie-Hellman Key Exchange", 1976, <https://app.ptuk.edu.ps/PTUK-stuff/UploadsMaterial/Material_151723_12140527_Section10.pdf>.

[draft-irtf-cfrg-vdaf-12] Barnes, R., Cook, D., Patton, C., and P. Schoppmann, "Verifiable Distributed Aggregation Functions", Work in Progress, Internet-Draft, draft-irtf-cfrg-vdaf-12, 4 October 2024, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-vdaf-12>.

[GENTRY] Gentry, C., "Fully homomorphic encryption using ideal lattices", Proceedings of the forty-first annual ACM symposium on Theory of computing , 2009, <https://dl.acm.org/doi/abs/10.1145/1536414.1536440>.

[I-D.ietf-ppm-dap]
Geoghegan, T., Patton, C., Pitman, B., Rescorla, E., and C. A. Wood, "Distributed Aggregation Protocol for Privacy Preserving Measurement", Work in Progress, Internet-Draft, draft-ietf-ppm-dap-12, 10 October 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-ppm-dap-12>.

[LWE] Regev, O., "The Learning with Errors Problem", <https://cims.nyu.edu/~regev/papers/lwesurvey.pdf>.

[POPLAR] Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., and Y. Ishai, "Lightweight Techniques for Private Heavy Hitters", IEEE S&P 2021 , 2021, <https://ia.cr/2021/017>.

Acknowledgments

TODO

Authors' Addresses

Lun Li
Huawei
Email: lilun20@huawei.com

Faye Liu
Huawei
Email: liufei19@huawei.com