Fully Homomorphically Encrypted (FHE) Identity and AI Ecosystem

Sapiens Labs Litepaper

Abstract

This litepaper presents the architecture and technical implementation of a decentralized Fully Homomorphic Encryption (FHE) identity protocol and vector database designed for Large Language Models (LLMs). Our approach leverages blockchain technology to ensure data integrity and immutability, adapts current blockchains to operate under FHE for secure and efficient consensus, and utilizes emerging decentralized GPU compute solutions to enhance computational scalability and performance. We detail the challenges faced, our innovative solutions, and the technical methodologies employed to create a secure, scalable, and privacy-preserving data processing infrastructure.

1 Introduction

Recent advancements in blockchain technology and Fully Homomorphic Encryption (FHE) have opened new avenues for secure, decentralized data processing. This paper introduces a comprehensive approach to integrating FHE with blockchain systems, aiming to create a decentralized identity and personal vector database optimized for Large Language Models (LLMs). By leveraging these cutting-edge technologies, we address critical issues related to data privacy, computational efficiency, and scalability, ultimately enabling secure and efficient processing of sensitive data in decentralized environments.

2 Problem

Current blockchain implementations face significant challenges in ensuring data privacy while maintaining computational efficiency. Traditional encryption methods do not allow computations on encrypted data, limiting the potential applications of blockchain in sensitive data environments. Specifically, model training typically requires plaintext data, which is a compliance issue with data protection regulations such as GDPR and HIPAA. Moreover, training complex models, such as LLMs, requires vast amounts of data that a typical consumer device will not be able to store. To summarize the problem we aim to address, there is no current solution to integrate blockchain technology with decentralized data storage and while ensuring the privacy of data via homomorphic encryption for model training

3 Solution

We propose a novel approach to address these challenges by incorporating Fully Homomorphic Encryption (FHE) into blockchain technology, which will allow for secure and decentralized training of LLMs through encrypting data with CKKS and using TFHE for encrypted identity and vector storage solutions. A visual demonstration of a message sender/receiver in our framework is displayed in Figure 1. Our solution involves the following key components:

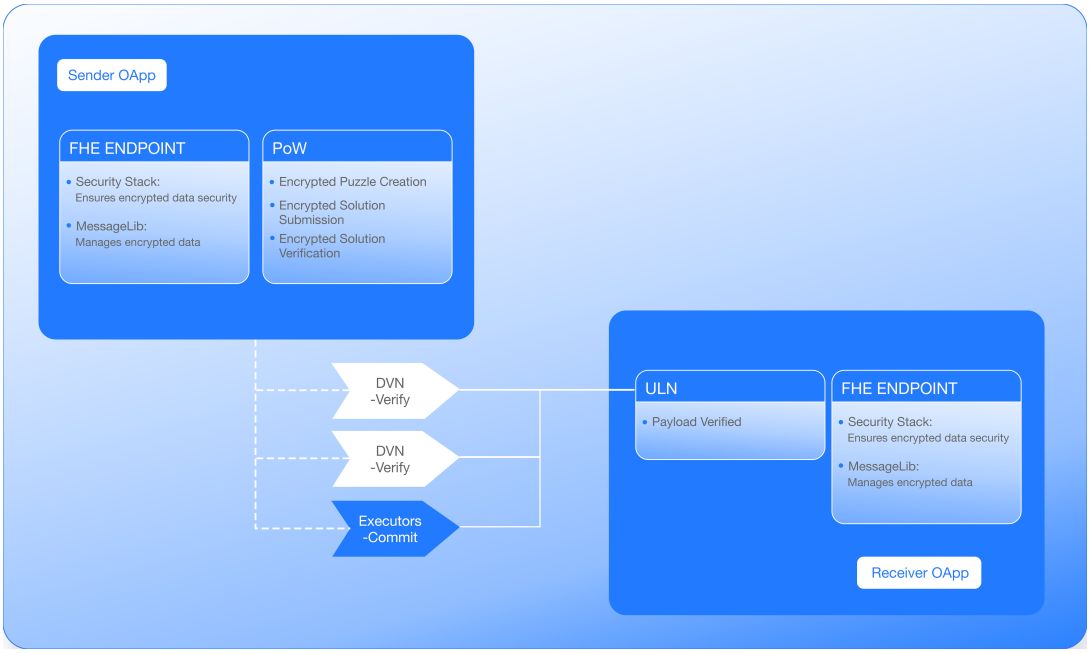

Figure 1: System overview of message and receiver exchange

3.1 SapiensID

SapiensID represents an innovation in personal identification, securely stored on a fully homomorphically encrypted (FHE) decentralized network. This cutting-edge solution ensures unparalleled data security and privacy for users' identity information. By integrating FHE, SapiensID enables the storage and processing of personal ID data without ever decrypting it, safeguarding against unauthorized access and breaches.

Through SapiensID, we aim to provide a secure, efficient, and scalable identity solution that seamlessly integrates with our decentralized FHE infrastructure, enhancing privacy and trust in the digital world.

3.2 SapiensBD

Our current database comprises over 1 million contacts, including business development, venture capital, family offices, and angel investors. This extensive network forms the backbone of our identity protocol.

To access SapiensBD, the Large Language Model (LLM), users are required to stake SPN tokens. This staking mechanism not only secures access to SapiensBD but also ensures that users are committed participants in the ecosystem. Users must sign up and provide their data to leverage the full capabilities of SapiensBD, fostering a comprehensive and interconnected web3 business development environment.

Our vision is to onboard the entire web3 BD ecosystem into the Sapiens universe, encouraging widespread staking of SPN tokens. While the platform supports traditional web2 payment methods for a software-as-a-service (SaaS) fee, using SPN tokens offers a distinct advantage. The platform is free for users who opt to stake SPN tokens, requiring only the staking commitment, thereby promoting the use of SPN and fostering a strong, token-driven community.

3.3 SapiensStorage

For handling large datasets, encrypted data vectors will be distributed to each node within the ecosystem so that individual participants can share the burden of large datasets. Moreover, since blockchain will insure immutability of encrypted data, this methodology will secure the storage and processing of encrypted data vectors. For the act of data distribution, a peer-to-peer network will facilitate the direct data exchange between nodes.

Large Language Models (LLMs) will be trained on these encrypted vectors, allowing them to process and learn from data without ever decrypting it. Moreover, since the data is decentralized, each node will act as a participant of model optimization and training while ensuring data integrity and privacy.

3.4 Security Considerations

In addition to end-to-end encryption protecting data between nodes, several other security measures are implemented to ensure the robustness of the system:

3.4.1 Key Management

A secure key management system involves multi-party computation (MPC) to handle key generation and decryption among multiple parties without revealing individual inputs.

3.4.2 Network Security

For software deployment, we will confirm network security by ensuring secure communication protocols such as TLS/SSL are used to protect data in transit. Additionally, intrusion detection and prevention systems (IDPS) monitor network traffic for suspicious activities and potential security breaches.

3.4.3 Audit Trails

Comprehensive audit trails are maintained to track all data access and modifications, ensuring transparency, accountability, and the ability to detect and investigate unauthorized activities.

3.4.4 Data Redundancy and Backup

Redundant storage and regular backups are implemented to protect against data loss. This ensures that data can be recovered in the event of hardware failures, cyberattacks, or other disasters.

3.4.5 Regular Security Assessments

Periodic security assessments, including penetration testing and vulnerability scans, are conducted to identify and mitigate potential security risks. This proactive approach helps in maintaining a high level of security in the system.

These comprehensive security considerations ensure that the decentralized storage system not only protects data at rest and in transit but also maintains its integrity, confidentiality, and availability against various threats.

3.5 Fully Homomorphic Encryption (FHE) for Data Vectors

3.5.1 Cheon-Kim-Kim-Song (CKKS) FHE Scheme

The Cheon-Kim-Kim-Song (CKKS) scheme was selected for its unique ability to handle real and complex numbers, making it ideal for applications requiring precise arithmetic operations, such as LLM training and proof-of-work computations.

CKKS works by encoding real or complex numbers into polynomials, which are then encrypted. This approach allows the scheme to approximate arithmetic on encrypted real numbers. Encrypted data can undergo homomorphic operations, such as addition and multiplication, which correspond to operations on the coefficients of these polynomials.

Another FHE method that was considered was Fast Fully Homomorphic Encryption over the Torus (TFHE). While this method is sufficient for PoS, it is not selected for handling data vectors due to its incompatibility with approximate arithmetic, which is a requirement for LLMs. The benefits of CKKS over TFHE are as follows:

- **Arithmetic on Encrypted Data**: CKKS supports efficient arithmetic operations on encrypted data, enabling complex calculations directly on ciphertexts.

  TFHE is optimized for exact computations on binary data, which is less flexible for applications needing approximate arithmetic.

- **Handling Real and Complex Numbers**: CKKS is uniquely suited for applications involving real and complex numbers, broadening its applicability to various fields.

  TFHE is primarily focused on Boolean gate operations and binary data, making it less suitable for tasks involving real or complex numbers.

- **Noise Management**: CKKS includes sophisticated mechanisms to manage and mitigate noise growth during arithmetic operations, ensuring that computations remain precise and reliable over multiple operations.

  TFHE manages noise effectively for gate operations, but faces challenges with the noise levels in arithmetic computations on large datasets.

- **Better Suitability for Machine Learning** CKKS's ability to handle real numbers and perform efficient arithmetic operations, CKKS is better suited for machine learning applications, including training and inference on encrypted data.

  TFHE is designed for binary gate operations, making it incompatible with the arithmetic needs of machine learning tasks, particularly for LLMs.

3.6 Vector Database

The vector database will utilize Hierarchical Navigable Small Worlds (HNSW) for vector searching, as implemented in Qdrant. HNSW is chosen for its scalability and ability to provide fast searches at scale, which is essential for handling large datasets generated by LLMs.

Why HNSW is Feasible Under FHE: HNSW is particularly suited for use with Fully Homomorphic Encryption (FHE) because of its efficient search algorithms and the ability to handle high-dimensional data. The key reasons for selecting HNSW are:

- **Efficiency**: HNSW offers logarithmic search complexity, which significantly reduces the time required for querying large datasets. This efficiency is crucial when dealing with the additional computational overhead of FHE.

- **Scalability**: HNSW can handle millions of vectors, making it suitable for large-scale applications such as LLM training and inference on encrypted data.

- **Incremental Construction**: HNSW allows for incremental construction of the graph, meaning new vectors can be added without the need to rebuild the entire structure. This flexibility is beneficial for continuously updating datasets.
- **Adaptability to Encrypted Data**: HNSW's search process involves distance calculations and neighbor traversals, which can be efficiently mapped to homomorphic operations on encrypted data.

4 Technical Methods

4.1 FHE Integration

Our integration of CKKS with Zama-ai will enable secure computations on encrypted data without decryption.

4.1.1 Encryption and Computation Processes

The following are the processes for encryption, computation, and decryption:

Data Encryption:

We will extend Zama-ai's encryption framework to include CKKS, allowing us to encrypt data vectors. This process transforms plaintext data into ciphertexts, ensuring confidentiality while permitting arithmetic operations on the encrypted data. For transaction details such as amounts, sender, and receiver information, TFHE will be sufficient since that data will not require approximate arithmetic on encrypted data while it is stored.

Homomorphic Computation:

Building on Zama-ai's infrastructure, we will enable nodes to store and process encrypted data vectors on the blockchain using CKKS. Nodes perform homomorphic operations such as addition and multiplication directly on the ciphertexts, enabling complex computations while preserving data privacy.

Decryption:

Once computations are completed, the results can be decrypted by authorized parties. The decryption process converts the homomorphically computed ciphertexts back into plaintext, ensuring that the results match those that would have been obtained if the operations had been performed on the original plaintext data.

4.2 Blockchain Consensus

The blockchain layer is adapted for FHE operations under TFHE, balancing security and efficiency. This involves several key components:

Adapted PoS Mechanism:

Our consensus mechanism is specifically tailored for FHE. Validators are selected based on the amount of cryptocurrency they stake, with all computations, including validator selection and transaction validation, performed on encrypted data. This ensures that the validation process upholds data privacy while maintaining network security.

Staking and Validator Selection:

Validators are chosen to create new blocks and validate transactions based on their staked cryptocurrency. This selection is conducted using homomorphic encryption to ensure that the process is secure and confidential, preventing manipulation and maintaining the integrity of the blockchain.

Incentive Mechanisms:

Incentives encourage validators to actively participate in the validation process, thereby maintaining the network's security and stability. Validators that successfully propose and validate blocks are rewarded with tokens or cryptocurrency. This reward system compensates participants for their stake and efforts, ensuring continued engagement and contributing to the blockchain's overall health and functionality.

Handling Tokens with FHE:

By encrypting transaction details such as the amount, sender, and receiver using Fully Homomorphic Encryption (FHE), the privacy of transactions is preserved even though the blockchain ledger remains publicly verifiable. This method ensures that sensitive transaction information remains confidential while still allowing the network to perform necessary validations and computations on the encrypted data to maintain the integrity and security of the blockchain.

4.3 Decentralized FHE Vector Database

The vector database is designed to store and manage FHE-encrypted vectors efficiently. Key features include:

Scalability:

The decentralized vector database is designed to efficiently manage and process large volumes of encrypted data. Its architecture supports scalability, making it suitable for applications with extensive datasets. This is achieved through distributed storage and computation, allowing the system to handle increasing data loads and query demands without compromising performance or security.

Query Optimization:

The database is engineered for efficient querying of encrypted data. It employs techniques such as Hierarchical Navigable Small Worlds (HNSW) to facilitate secure and precise vector searches. This optimization ensures that search operations are performed swiftly, even with large datasets, by minimizing query times and maximizing retrieval accuracy within the encrypted domain.

To demonstrate how this will work for a Distributed Database of CKKS Encrypted Vectors, we will provide an Algorithm and a diagram in Figure 2:

Step 1: Build the HNSW Index

- Construct the HNSW index for the encrypted vectors distributed across multiple nodes.
- Each node builds a local HNSW index for the encrypted vectors it stores.
- Nodes communicate to share metadata about their local indices, such as the structure of the HNSW layers and representative points.

Step 2: Query Initialization

- A query vector, encrypted using CKKS, is submitted to the distributed system.
- The query is first processed to identify the initial entry points for the search. These entry points can be either locally stored vectors or representative points from other nodes.

Step 3: Distributed Search Coordination

- The search process is coordinated across multiple nodes to find the k-nearest neighbors.
- Coordinator Node: Designate a node as the coordinator for the query. This node manages the search process and aggregates results from other nodes.

Step 4: Local Search

- Perform a local HNSW search on each node using the encrypted query vector.
- Each node starts the search from its local entry points and explores the HNSW graph to find approximate nearest neighbors.
- The search uses homomorphic computations to compare the query vector with the encrypted vectors stored locally.

Step 5: Communication and Aggregation

- Nodes communicate their local search results to the coordinator node.
- The results include distances and identifiers of the nearest neighbors found in each local search.
- Communication is performed securely to maintain the confidentiality of the encrypted vectors.

Step 6: Global Aggregation and Finalization

- The coordinator node aggregates the local search results to determine the global k-nearest neighbors.
- It combines and sorts the results based on the homomorphically computed distances.
- The coordinator node may request additional local searches if needed to refine the results.

Step 7: Return Results

- The coordinator node decrypts the final k-nearest neighbors (if permissible) and returns the results to the querying entity.
- The results include the nearest neighbor vectors and their corresponding distances to the query vector.

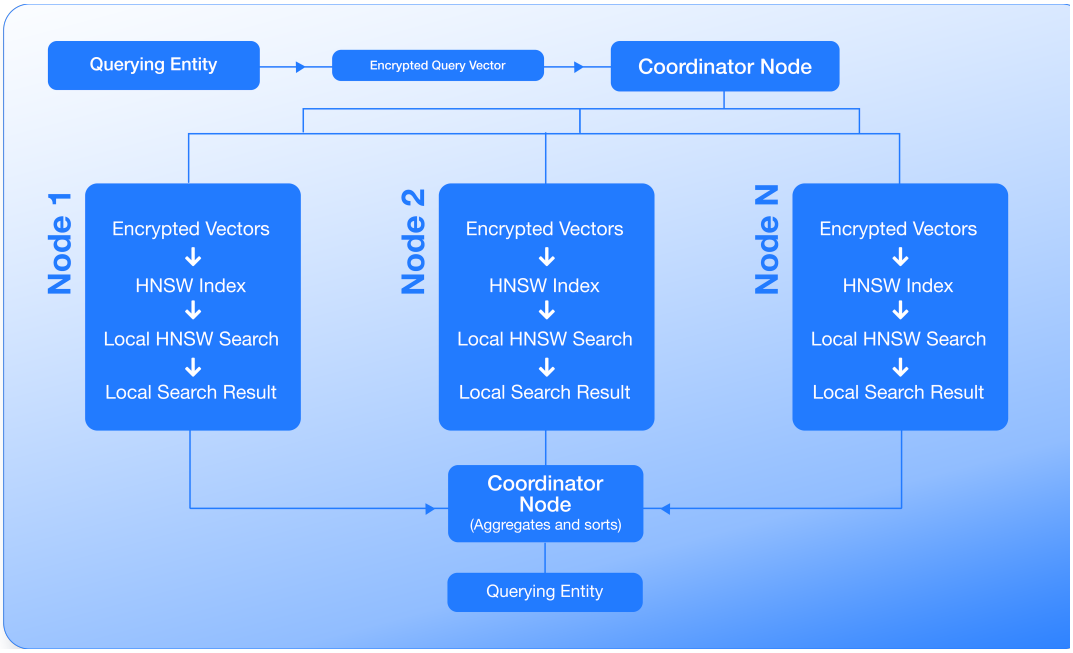

Figure 2: Querying an encrypted vector in the decentralized database

4.3.1 Fully Homomorphic Encryption Integration for LLM Training

To further enhance security and privacy, our system allows Large Language Models (LLMs) to process encrypted inputs without ever decrypting them. This is achieved through the following methods:

Secure LLM Training:

Large Language Models (LLMs) are trained directly on encrypted data, ensuring the confidentiality of sensitive information throughout the training process. This method adheres to privacy and compliance standards, such as GDPR and HIPAA, by maintaining data encryption at all stages. Hence, it protects sensitive data from unauthorized access or exposure.

Optimized Computation:

The integration of CKKS with Zama-ai enables efficient arithmetic operations on encrypted data, crucial for the resource-intensive process of training LLMs. Techniques to manage noise and maintain computational accuracy ensure the feasibility of training sophisticated models on encrypted data.

4.4 SapiensAI

SapiensAI is your personal AI, uniquely trained on your encrypted personal storage, ensuring your data is never decrypted. This personal LLM operates as a federated AI model, allowing you to create a personalized AI agent tailored to your needs.

With SapiensAI, you can name your AI, design its functionality, and select an avatar, making it a truly personalized experience. Integration software is provided to seamlessly incorporate LLMs using your encrypted data into your existing products or communication tools, such as Slack and MS Teams.

SapiensAI offers a secure, customizable AI solution that enhances your digital interactions while maintaining the highest standards of privacy and data protection.

To further enhance computational efficiency and scalability, we utilize decentralized idle GPU compute, leveraging existing solutions like Gensyn and IoNet. Following Yuan et al., this phase involves harnessing idle GPU power from a diverse range of nodes globally, making computational resources more accessible and cost-effective. Thus, our Idle GPU resource utilization will be ideal for training LLM's on data encrypted under FHE.

4.4.1 Optimized Algorithms for GPUs

Given that FHE is computationally intensive, we optimize the underlying algorithms for parallel processing on GPUs. Within our methods, neural network computations are tailored to be efficient on encrypted data using techniques compatible with GPU architectures. This involves breaking down complex computations into smaller, parallelizable tasks that can be executed simultaneously on multiple GPU cores. For training LLMs, each training stage will be divided into receiving, computation, and sending slots, which are assigned to different CUDA streams to overlap communication with computation and hide latency.

4.4.2 Scaling and Efficiency

We address the challenge of scaling encrypted computations by developing methods to efficiently distribute tasks across multiple decentralized GPU networks through partnerships. Tasks are dynamically assigned to nodes based on their current load and processing power. Specifically, as from Yuan et al., a scheduling algorithm will allocate "tasklets" to decentralized GPU devices To strategically distribute tasks and optimizing communication protocols.

To understand how this works, we will provide an algorithmic description:

Problem Definition:

- **Tasklet**: A tasklet represents a unit of computation, such as a micro-batch of training data and a subset of model layers, in the context of training a neural network.
- **Devices**: A set of decentralized GPUs (devices) available for processing the tasklets.
- **Network Characteristics**: The network has varying bandwidth and latency between different devices.

Objective:

- Minimize the total communication cost while maximizing the computational throughput.
- Ensure load balancing across the GPUs to prevent any single GPU from becoming a bottleneck.

Input:

- A set of tasklets $T = \{t_1, t_2, \ldots, t_n\}$.
- A set of devices $D = \{d_1, d_2, \ldots, d_m\}$.
- Communication cost matrix C, where $C[i][j]$ represents the communication cost between device d_i and d_j.

Output:

- An allocation strategy σ that maps each tasklet t_i to a device d_j.

Algorithm

Step 1: Initialize Population:

- Generate an initial population of candidate allocations. Each candidate is a random assignment of tasklets to devices.

Step 2: Evaluate Fitness:

- For each candidate allocation, evaluate its fitness based on the total communication cost and computational load balance.
  - **Communication Cost**: Sum of the costs of data transfers required for the assigned tasklets.
  - **Load Balance**: Variance in the number of tasklets assigned to each device.

Step 3: Selection:

- Select pairs of candidate allocations (parents) from the population based on their fitness. Better fitness increases the likelihood of being selected.

Step 4: Crossover:

- Generate new candidate allocations (offspring) by combining parts of the selected parents. This involves swapping subsets of tasklets between the parents.

Step 5: Mutation:

- Apply random changes to some of the offspring to introduce variability. This can involve reassigning a few tasklets to different devices.

Step 6: Local Search Optimization:

- Perform local search on the offspring to improve their fitness. This involves:
  - Swapping tasklets between devices to reduce communication costs.
  - Adjusting tasklet assignments to balance the computational load.

Step 7: Replace Population:

- Replace the worst-performing candidates in the population with the new offspring.

Step 8: Repeat:

- Repeat Steps 2-7 for a fixed number of iterations or until convergence (no significant improvement in fitness).

5 Deployment and Maintenance

5.1 Deployment Strategy

To ensure a smooth rollout and updates of the system, we focus on several key aspects. Firstly, node setup is streamlined through automated deployment scripts and configuration management tools, enabling consistent deployment across various environments. Network configuration is optimized for decentralized operations, with secure communication protocols and efficient data routing mechanisms. Scalability is a core consideration, with the infrastructure designed to dynamically adjust to varying workloads and expand seamlessly as demand grows. Comprehensive documentation and support resources are provided to assist users in the deployment process.

5.2 Monitoring and Logging

Monitoring and logging mechanisms are implemented to track the system's health, performance, and security in real-time. Monitoring tools continuously assess the status of nodes, network traffic, and computational workloads, providing detailed insights into system operations. Logging captures comprehensive data on transactions, computational tasks, and system events, facilitating thorough auditing and troubleshooting. Real-time alerts and dashboards enable prompt identification and resolution of issues, ensuring the system operates reliably and securely.

5.3 Maintenance and Updates

Regular maintenance and updates are critical for keeping the system secure and efficient. A structured maintenance schedule includes routine checks, performance optimizations, and security assessments. Updates and patches are rolled out systematically to address vulnerabilities, enhance features, and improve overall system stability. Automated update mechanisms minimize downtime and ensure that nodes remain up-to-date with the latest enhancements. Detailed maintenance logs and reports are maintained to track changes and inform future improvements.

6 Conclusion

Our approach provides a comprehensive solution to integrating FHE into blockchain technology, addressing data privacy and computational efficiency challenges. By leveraging Zama-ai and extending that framework with CKKS, implementing a decentralized FHE vector database, and utilizing emerging decentralized GPU networks, we enable secure and efficient processing of sensitive data in decentralized environments, specifically tailored for Large Language Models.

References

[1] Zama, "fhEVM: A Solidity library for interacting with an fhEVM blockchain," 2023. `https://github.com/zama-ai/fhevm`.

[2] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[3] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pp. 409–437, Springer, 2017.

[4] D. Rotaru, N. P. Smart, T. Tanguy, F. Vercauteren, and T. Wood, "Actively secure setup for spdz," *Journal of Cryptology*, vol. 35, no. 1, p. 5, 2022.