An AI-Driven Post-Quantum Cryptographically Secure Workflow for Collaborative Credit Scoring

Daniel Aronoff
MIT
daronoff@mit.edu

Nut Chukamphaeng
Senior Research Scientist, SCBX
nut.c@scbx.com

Phoochit Witchutanon
AI Engineer, DataX
phoochit.witchutanon@data-x.ai

Samiran Chanseewong
AI Engineer, DataX
samiran.chanseewong@data-x.ai

Koravich Sangkaew
AI Engineer, DataX
koravich.sangkaew@data-x.ai

Tutanon Sinthupraisth
Head of R&D, SCBX
tutanon.s@scbx.com

Abstract

Credit scoring plays a critical role in the financial industry, allowing institutions to evaluate the creditworthiness of potential borrowers. Typically, a model is estimated from repositories of attributes of past borrowers linked to their loan and payments performance. The model is then used to compute an applicant's score. The training and customer data are subject to regulations that require privacy of financial records. This creates a tension between the full utilization of available data and the prevention of leakage. Recently, the tension has intensified from, on one hand, improvement in AI methods to utilize data from nontraditional sources to develop prediction models and, on the other hand, increased concern over the vulnerability of encrypted data to penetration from quantum computers. We present a credit score workflow that addresses both issues by using AI methods to estimate a credit score model in a collaborative setting, combined with post-quantum cryptographic methods to protect data. We develop a "toy" workflow which can form a base for more complex "real world" implementations. We provide links to a code-base.

1 Introduction

Credit scoring plays a critical role in the financial industry, allowing institutions to evaluate the creditworthiness of individuals and businesses seeking loans or other financial products. Traditional credit scoring methods, such as the widely used FICO score and VantageScore, rely on centralized data collection and analysis. These methods typically involve first estimating a credit score function from historical data on the attributes and credit histories of large numbers of borrowers. An individual's credit score is then generated by gathering extensive information, inter alia, about an individual's credit history, including payment history, outstanding debt, credit utilization, and types of credit used and uploading it into the credit score model for processing. The output is a credit score, which represents the likelihood of a borrower repaying their debt obligations on time. While these traditional methods have been effective in assessing credit risk, they give rise to two concerns. One concern is that data privacy regulations place some limits on the types of analysis that can be performed on borrower data without risking leakage, which reduces the precision of credit score generated for new loan applicants. A second concern is the robustness of the data privacy protections to attacks from quantum computers. Most cryptographic methods that are currently used to protect data messaging and computation are vulnerable to a quantum computer attack. At the same time, there have been improvements in AI methods to utilize data from nontraditional sources to develop prediction models.

We present a credit score workflow that addresses both issues by using AI methods to estimate a credit score model in a collaborative setting, combined with post-quantum cryptographic methods to protect data. Our workflow spans the life-cycle of a credit scoring process, from the estimation of the scoring model to the generation of an applicant's credit score, to the verification of model and borrower data properties. At each step of the workflow we use cryptographic protocols that relax limitations on the types of modeling that can be applied to the data while protecting identity and data privacy against an attempt by a quantum computer to break the encryption. We estimate and deploy the credit score model in a collaborative setting which, in conjunction with the post-quantum cryptography, enables fully flexible model structure and unlimited data input on potential borrowers. The former is accomplished by using federated learning to train a neural network.

We adopt a neural network model to allow complex non-linear relationships in the credit data to be captured; however, our collaborative training approach is *model-agnostic* and could likewise employ simpler models (e.g., gradient boosting trees or decision trees) that are easier to train and interpret if those models suffice. We focus on a neural network here to maximize modeling flexibility and to stress-test the cryptographic workflow with a more complex model. This federated approach enables each data-holding entity to perform model training on its own dataset and share only encrypted model updates (parameters or gradients) with a central aggregator (rather than raw data), thereby preserving privacy—this constitutes the AI portion of our workflow. The latter is accomplished by using multiparty computation ("MPC") and fully homomorphic encryption ("FHE") methods to compute the credit score for an applicant on data from multiple providers that is encrypted or obscured. We partition the workflow into four sequential steps and apply distinct cryptographic methods to each one. Step 1 - Federated learning. Step 2 - fully homomorphic encryption. Step 3 - hashing. Step 4 - non-interactive zero-knowledge proofs of knowledge. In Steps 2, 3, and 4 we compare our chosen methods with alternative post-quantum cryptographic methods across a broad set of performance dimensions.

We aim not only to examine their theoretical potential but also to evaluate their feasibility and effectiveness in enhancing privacy and security within credit scoring processes. By emphasizing practical implementation, we seek to provide actionable insights that enable financial institutions to leverage these cutting-edge technologies in safeguarding sensitive data while maintaining robust credit risk assessments.

Code and simulation of workflow A companion GitHub site contains the code to execute each step of the workflow and a simulation of the entire workflow. The link is `https://anonymous.4open.science/r/fhe-excute-1422`

1.1 Challenges of credit scoring in banking

The design of a credit score workflow must meet several requirements for the handling of sensitive borrower data.

- **Data privacy and security:** Centralized data storage creates a risk of breaches, as exemplified by the Equifax breach [11]. There are several

dimensions of data security. One is ensuring that borrower data remains private. A second is ensuring that customer data is accurate. A third is ensuring that the credit score model has not been secretly altered.

- **Data Silos:** Data on borrower attributes and loan performance are held by separate loan servicing entities. Data privacy requirements can limit the effective pooling of data, which reduces the precision of the estimated credit score model.
- **Regulatory compliance:** Each step in the credit score workflow must comply with data privacy regulations, e.g. GDPR, which imposes strict requirements to protect data privacy. [5].

1.2 The Quantum Computing Threat

Most cryptographic methods used for data privacy rely primarily on one of two computationally difficult mathematical problems: (i) the discrete logarithm problem (including elliptic curve cryptography), and (ii) the integer factorization problem, specifically factoring large composite numbers into their prime factors. For example, RSA encryption and digital signatures—which protect sensitive data, such as financial transactions transmitted over the internet—depend on the computational difficulty of factoring large composite numbers. The computational resources required to decrypt data protected by these algorithms far exceed the practical capabilities of classical (non-quantum) computers, making it infeasible to break such encryption within any reasonable timeframe (e.g., hours, days, or even years). However, quantum computers, through algorithms such as Shor's algorithm, can factor large composite numbers significantly faster, thus potentially breaking RSA encryption and other cryptographic schemes based on these classical hard problems. Although quantum computing is currently expensive and limited in scale, there is increasing concern that as quantum computing becomes more affordable and scalable, existing data protection methods will become vulnerable to quantum attacks. The urgency in developing quantum-resistant cryptographic methods is further heightened by the risk that malicious actors could presently intercept and store encrypted data—such as financial records—for later decryption once quantum computers become widely available.

1.3 Contributions

This paper makes several contributions.

- **Design of a quantum secure credit score workflow** We design, code and simulate an ensemble of post-quantum cryptographic methods that ensure data privacy over a multi-step workflow.
- **Collaborative credit score using AI** We use federated learning to enable multiple providers of training data to collaborate in training a neural network model without exchanging their data. We then use FHE to enable computation of a credit score without revealing data from multiple providers of data relevant to a potential borrower and limiting decryption of the output credit score to a designated receiver.
- **Comparative Analysis of post-quantum cryptographic methods:** We compare our chosen method with several alternative post-quantum methods across several performance dimensions for the credit score computation (step 2) and verification of selected properties that may be required by regulators (steps 3 and 4).
- **Guidance for Implementation:** We offer actionable insights for financial institutions considering these advanced cryptographic methods. Our recommendations address key practical considerations such as scalability, interoperability, standardization, and performance, ensuring a smooth transition from theory to practice.
- **Codebase and Simulation:** We provide a companion codebase for each Step of the workflow and a simulation for the entire workflow process. The codebase and simulation can be accessed here:

  `https://anonymous.4open.science/r/fhe-excute-1422`

  We provide links to the appropriate sub-folders in each section where a workflow step is discussed.

2 Overview of the Post-Quantum Credit Score Workflow

Figure 1 depicts our toy model of a credit score workflow. It divides into four sequential steps, moving from model estimation, to customer credit score to verification of key elements of the model and its application. Here we describe the elements of our setup and an overview of each step.

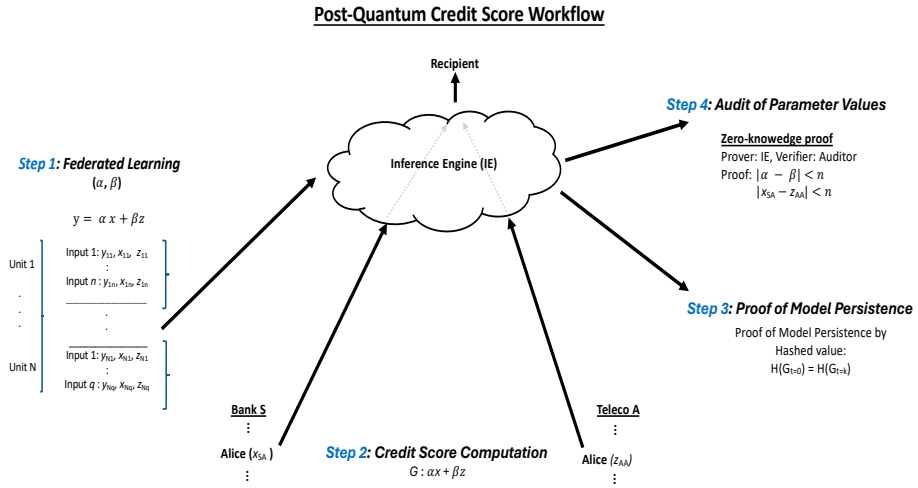

Figure 1

Model elements A credit score model is estimated from borrower attributes and loan performance. The model is then applied to the attributes of a borrower applicant, called Alice, who has applied for a loan.

- *Attributes* Each borrower i has two measured attributes, $\{x_i, z_i\}$. They represent a reduced version of the more numerous measured attributes that are typically collected on loan applications.
- *Outcomes* y_i is the outcome of borrower i's loan (i.e. did she pay, or not?).
- *Units* There are N units, e.g. loan desks at a bank (auto loan, home loan etc...) each with a repository of past borrower data, encoded as the tuple $\{y_{mi}, x_{mi}, z_{mi}\}$ for borrower i in unit m.
- *Model Parameters* $\{\alpha, \beta\}$ are the parameters of the credit score model $G : \alpha x + \beta z \to y$ that is estimated at step 1.
- *Inference engine ("IE")* This is a server (or network of servers) that estimates the credit score function and computes the credit score of a loan applicant (e.g. Alice). The inference engine is controlled an entity who coordinates the workflow, whom we call the mechanism operator.[1]
- *Recipient* The recipient is the unit to which Alice has applied for a loan, e.g. an auto or home loan originator.

[1]The IE could be a third party server such as the Amazon cloud.

- *Banks S, Teleco A* These are sources of attribute data for Alice, e.g. Teleco A might send a scaler that represents her record of paying her utility bills. Bank S sends attribute x_{SA} and Teleco A send attribute z_{AA}.

2.1 Workflow steps

Our workflow uses cryptography to maintain differential privacy of borrower, applicant and credit score data. We employ post-quantum methods for computations on servers and messaging between servers. We briefly summarize the steps in the workflow.

1. *Step 1 - Federated Learning*: The credit score model is estimated from confidential borrower data held by units. The inference engine sends the algebraic structure of the model to the units, $y = \alpha x + \beta y$.[2] Each unit estimates the parameters $\{\alpha, \beta\}$ from its own borrower database and sends the encrypted result to the inference engine. The inference engine decrypts and obtains its parameters by averaging the unit estimates. The units do not share borrower data with each other or with the inference engine.

   Hashing The inference engine hashes the timestamp, algebraic structure and model parameters.

2. *Step 2 - Credit Score Computation*: Alice's credit score is computed from attributes sent encrypted by Bank S and Teleco A to the inference engine. The inference engine computes the score and sends it encrypted to the receiver. We discuss three method the inference engine could use; MPC, FHE on smart contract on a blockchain and a combination of MPC and FHE. For each case we explain how the method limits disclosure of the credit score to the receiver while also enabling the other parties to verify that the inference engine uses the correct model to compute Alice's credit score.

   Hashing The inference engine hashes the timestamp, algebraic structure and model parameters.

[2]We do not address model selection. It is possible to add a prior step using machine learning methods to select the functional form of the credit score model. This could be implemented with federated learning.

3. *Step 3 - Proof of Model Persistence*: Certain parties may require verification that the model in Step 1 was used to compute Alice's credit score (for example, a financial regulator). Verification is provided by applying a one-way function - known as a hash function - to an encoding of the model; the algebraic structure and the parameters. The output of the hash function of the credit score model in Step 1 is is compared to the output of the model used to compute Alice's credit score in Step 2. Identical outputs indicate the same model was used. Otherwise the models are different.

4. *Step 4- Audit of Parameters and Variables* Certain parties may require verification of properties of the model parameters, $\{\alpha, \beta\}$, or Alice's attributes, $\{x_{SA}, z_{AA}\}$ e.g. to detect the possibility of fraud. We use non-interactive zero-knowledge proofs ("NIZK"'s) This method proves the existence or nonexistence of the property in question without leaking any other information about the mode or attributes.

2.2 Cryptographic methods

We use cryptographic methods to ensure that computations within servers and messages between servers are private, secure and verifiable.[3]

Privacy and security of computations At each step of the workflow we describe a post-quantum cryptographic method to protect the data (the "chosen method"). Additionally, for steps 2 - 4 we briefly describe alternative post-quantum methods that could be employed and compare the methods along a set of performance dimensions.[4] We also link to a simulation of the full workflow. The chosen methods, with links, are the following.

- *Step 1:* Federated learning `https://anonymous.4open.science/r/fhe-excute-1422/fl/README.md`
- *Step 2*: MPC `https://anonymous.4open.science/r/fhe-excute-1422/mpc/README.md` and Zama TKMS `https://anonymous.4open.science/r/fhe-excute-1422/t_fhe/README.md`.

[3] An alternative to cryptographic methods is to compute in a trusted execution enclave ("TEE") around a CPU or GPU. Encrypted data is decrypted, computed and re-encrypted in the TEE. Security is provided by the hardware, rather than the mathematical properties that cryptography uses to ensure privacy. We do not evaluate TEE methods.

[4] The same metrics are used at each step.

- *Step 3*: SHA 256 with CRYSTALS-Dilithium `https://anonymous.4open.science/r/fhe-excute-1422/hash/README.md`
- *Step 4*: zk-STARK (Winterfell) `https://anonymous.4open.science/r/fhe-excute-1422/zkp/README.md`

Privacy and security of messages One low overhead post-quantum method to secure channels of communication between agents is PQ - Connect ([9]). The method is implemented by downloading and installing the software. Two servers recognize if the other has installed PQ-Connect. After verification, messages sent between servers are automatically encrypted by the sending server and decrypted by the receiving server.[5] This prevents leakage of the data sent between servers at each step. We use PQ-Connect in Steps 1 3 and 4.

Step 2 requires bespoke messaging. For MPC we use Shamir Secret sharing, which involves sending decomposed elements of the data to other servers. These "secrets" do not allow any inference about the original data, so there is no need to take the further step of encryption. For Zama, we use the FHE encryption generated by the Zama protocol. Again, there is no need to take the further step of encryption.

Verification of computations The correctness of computations at each step can be verified by third parties without leaking data. Step 3 uses a hash function to prove that the same model was used in steps 1 and 2. Step 2 discusses three methods that can be used to verify the computation; multiparty computation, encrypting underneath an FHE computation on a server and FHE on a blockchain smart contract. Step 4 uses a zero-knowledge proof of certain model properties. Each method is post-quantum.

2.3 Limitations on data protection

We use cryptographic methods to protect data from quantum computer attacks and to enable verification of computations. However, there remain vectors of attack that are not protected by cryptography and which might be insuperable. They are parts of the process that rely on unverifiable trust. We review the trust assumptions implicit at each step.

[5]There are several cryptographic methods that can implement this. One popular instantiation is transport layer security ("TLS").[6]

1. The units and the inference engine are trusted to follow the protocol to compute α and β. The inference engine is trusted to hash the correct model.

2. Bank S, Teleco A and the inference engine are trusted to provide correct data. Zama is trusted to (i) dispose of its knowledge of the secret key and (ii) to publish the correct computation to the blockchain.[7]

3. The inference engine is trusted to use hashes from the models actually used at Steps 1 and 2.

4. The inference engine is trusted to use variables from the actual model it computed (i.e. that the referenced α and β etc...are the correct ones.

The lesson here is that the application of cryptography requires trust. Establishing trust can sometimes be achieved with other tools. When the inference engine is a regulated financial institution, trust may be achieved by a competent regulator equipped with adequate penalties for identified trust violations. The veracity of reporting α and β by units in Step 1 and x and z by Banks S and Teleco A in Steps 1 and 2 is statistically tested in Step 4 (which, in turn, relies on Trust in the inference engine).

3 Step 1: Federated Learning to Estimate Credit Score Model

Federated learning in our setting is an approach to estimating a credit score model on separate data sets on borrower histories held by several units, that achieves approximately the same result as if the data was pooled. Since centralizing this sensitive raw data is often impossible due to privacy constraints, FL allows each unit to train a model $\{\alpha,\beta\}$ for $y = \alpha x + \beta y$ locally and send the recovered parameters to the inference engine. The underlying borrower tuple $\{y_{mi}, x_{mi}, z_{mi}\}$ is never revealed. The inference engine then averages the values of $\{\alpha,\beta\}$ across all reporting units. We encrypt the data that is sent from the units to the inference engine with PQ-Connect, a post-quantum method to protect data in transmission.

[7] For a discussion of trust gaps in FHE and the Zama protocol in particular, see Aronoff et.al. [1].

3.1 Description of federated learning

To implement this, we utilize the **Federated Averaging (FedAvg)** algorithm [7]. This algorithm coordinates the collaborative training process through iterative rounds, managed by the inference engine. The key stages within each round are:

1. **Model distribution:** The inference engine distributes the current state of the global model parameters (e.g., the latest $\{\alpha,\beta\}$ values) to the participating units.[8]

2. **Local training and update generation:** Each unit uses its local, private data $\{y_{mi}, x_{mi}, z_{mi}\}$ to train the received model, refining the parameters based on its specific data patterns. Based on this local training, the unit determines the resulting model update, typically represented as the newly learned parameters $\{\alpha,\beta\}$.

3. **Update transmission:** Units securely transmit the model's parameters back to the inference engine.

4. **Central aggregation:** The inference engine aggregates the received updates from all participating units. Following the FedAvg protocol, this usually involves computing a weighted average of the parameters (e.g., $\{\alpha,\beta\}$). This produces the improved global model for the next round.

5. **Iterative updating** The inference engine distributes the updated state of the global model parameters to the participating units, until a prespecified condition is achieved (e.g. number of rounds, convergence of estimates...).

3.1.1 Scalability with AI

Our model is a single layer neural network with two nodes, α and β with independent connections to variable inputs x and z. The model is trained by supervised learning on the output y. Each unit m trains the model on its tuples of borrower characteristics and loan outcomes $\{y_{mi}, x_{mi}, z_{mi}\}$ for borrower i in unit m. Figure 2 displays the structure of the neural network model. The iterative estimation process begins with an initial network

[8]The first-step can be a random assignment of parameter values.

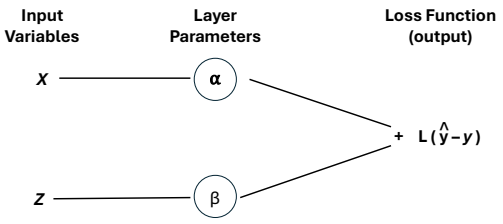

Figure 2: Neural Network for Federated Learning

structure and neuron values. A forward pass generates an output $\hat{y}$. A loss function measures a distance between $\hat{y}$ and the empirical y associated with the input variables. Units use backpropagation to refine the parameters to reduce the loss (possibly with cross-validation or other methods to separate training and test data) and send the parameters to the inference engine. The inference engine averages the parameter values and sends them back to the units for another round of estimation, etc... It can be readily seen that the neural network can be extended and reshaped to accept an unlimited number of inputs, layers, neurons and internal connections.

3.2 Implementation design

See code and simulation here `https://anonymous.4open.science/r/fhe-excute-1422/fl/README.md`

4 Step 2: FHE to Compute a Credit Score

An applicant borrower's credit score (Alice) is generated by computing of a weighted average of two numbers, each representing an attribute, where each number is provided by independent parties. The weights of the computation are generated in Step 1 and are known only to the inference engine or to an entity that operates the inference engine (the "inference engine"). The output of the computation is observed by the receiver. The challenge is to protect the privacy of Alice's attribute numbers and her credit score, and to enable verification of the computation.

4.1 Description of methods

We present three different post-quantum cryptographic methods to implement the program. Each method involves computing arbitrary functions on data that obscures the underlying plaintext from the servers doing the computations and from other agents. Only selected agents are able to decode or decrypt the plaintext output of the computation. The first method is MPC where plaintext is partitioned and distributed among two or more servers. Each server performs the computation and send their output to one agent, the receiver, who decodes the solution. MPC does not use encryption. Privacy is maintained by providing each server with a part, but not the whole, of the plaintext. The second method is FHE, which uses a method of encryption called learning with errors ("LWE"). In this case the computation is performed on the ciphertext by a single operator, which can be a smart contract on a blockchain or a server. The third method combines the first two methods and combines MPC with FHE.

- *Vanilla MPC*: Multiparty computation with Shamir secret shares and no encryption (e.g. BGW).
- *Zama on FHE Blockchain*: use Zama FHE compiler with smart contract FHE computation on a blockchain.
- *MPC with Threshold FHE*: FHE computation on private servers.

4.1.1 Vanilla MPC

In this case the model, M is provided to all parties, e.g. if $G = \alpha \cdot x + \beta \cdot z + w$, the parties are given the linear equation structure. They are not provided the parameters $\{\alpha, \beta\}$ or the variables $\{x, z, w\}$, that are initially possessed by other agents. Here, the challenge is to (a) enable each party to validate that the model is correctly implemented subject to (b) not observing the inputs and ensuring that only the receiver can decode the credit score. *Outline of protocol*:

Parties: Teleco A, Bank B, Recipient.

1. The inference engine sends model M to all parties.
2. Each party sends a Shamir secret share of its data to the other parties.

   x = Teleco A Alice score.

z = Bank B Alice score.

α, β inference engine parameters.

w = receiver score = 0. The inclusion of w enables the receiver to exclusively decrypt the credit score. It is a redundant addition to G that does not affect the credit score.

3. Each party; inference engine, Teleco A, Bank S and receiver computes $G' = \alpha' x' + \beta' z' + w'$, where $\{\alpha', \beta', x', z', w'\}$ are the shares of the inputs it receives.
4. Each party sends its output to the receiver. The receiver does not send its output to any other party.
5. The receiver's accumulation of outputs G' enables it alone to decode the .

Key observations

- The method is post-quantum because there is no way to derive information on the parameters and variables from the secret shares.
- Each party can validate the correctness of the computation because they execute G' on their shares.
- No party, including the receiver, can infer the input values x and z (a property of BGW MPC)
- The receiver is the only party who can decode to obtain the credit score. This follows from the protocol where the receiver is the only party to observe the output G' of each party.

4.1.2 Zama TKMS FHE smart contract on a blockchain

In this case the Zama open source protocol `https://www.zama.ai/post/introducing-zama-threshold-key-management-system-tkms` issues the keys, compiles the data and executes a smart contract on a blockchain. The compiler issues the public and secret key pair $\{FHE.pk, FHE.sk\}$. The model and inputs are sent to the compiler encoded. All parties can validate the computation from the blockchain and any party issued the secret key can decrypt the output. This method ensures privacy through FHE and secret key distribution, and verifiability from observation of the record of smart

contract computations on a blockchain.[9]

Outline of protocol:

1. Zama compiler generates $\{FHE.pk, FHE.sk\}$ and sends pk to all parties and sk to selected parties.
2. Parties encrypt their inputs and send it to the compiler:

   $c_A = FHE.Encrypt(FHE.pk, x)$, $x =$ Teleco A Alice score

   $c_B = FHE.Encrypt(FHE.pk, z)$, $z =$ Teleco S Alice score

   $c_G = FHE.Encrypt(FHE.pk, G)$, $G = \alpha \cdot x + \beta \cdot z$ is the inference engine model, where e.g. encrypted α is denoted c_α.
3. c_A, c_B, c_G, T are sent to the smart contract and executed.
4. Smart contract evaluates the formula $G' = FHE.Encrypt(\alpha \cdot x + \beta \cdot z)$ and compares the encrypted output to T.

   If the smart contract operates on a public blockchain, the computation is carried out by miners and the participants can, by observing the blockchain record, validate that the correct function G' is applied.
5. The smart contract output is decrypted by the receiver, who holds the secret key sk.

Key observations

- Each party validates the integrity of the model in Step 3.
- Each party validates the correctness of the computation in Step 6.
- No parties other than the receiver decode the output.

However, leakage can occur from key generation. Zama generates the secret key, which can decode the output.[10]

[9] There is a trust assumption that the blockchain itself is not corrupted.

[10] Zama attempts to minimize this possibility by a MPC key generation ceremony in the Amazon cloud that can only be corrupted if Amazon participates in a collusion with other parties.

4.1.3 MPC with Threshold FHE

In this case the model, M, is known only to the inference engine and it is not leaked to the other parties. Instead, the inference engine proves that it is using the same model as it used in some prior credit score application. The model is $G = \alpha \cdot x + \beta \cdot z$, where x and z are variables and the values of the parameters α and β are proprietary to the inference engine. Here, the challenge is to enable each party to validate the correctness of the computation while preventing leakage of information on the variables or the parameters and to ensure the receiver is the only party who can decrypt the credit score.

Outline of protocol:

1. *Keys*:

   The inference engine generates a public key and a secret key for Fully Homomorphic Encryption (FHE) operations, which can be based on various schemes such as BGV (Brakerski-Gentry-Vaikuntanathan, [2]), BFV (Brakerski/Fan-Vercauteren, [6]), or TFHE (Fully Homomorphic Encryption over the Torus, [4]), denoted as $\{FHE.pk, FHE.sk\}$. To ensure distributed control and prevent any single party from unilaterally decrypting sensitive information, the inference engine breaks apart the FHE secret key ($FHE.sk$) into shares (e.g., $\{FHE.sk1, FHE.sk2, ...\}$). Each participating party, including the inference engine which retains one share, receives a portion of this secret key.

   The final output of the computation (e.g., the credit score) is encrypted using FHE. To obtain the unencrypted output, a threshold decryption process is employed. This requires a sufficient number of parties, each holding their share of the $FHE.sk$, to collaboratively decrypt the FHE ciphertext. This ensures that the receiver is the intended party who can access the final score, but only through a multi-party computation protocol involving the shared secret key components.

2. Each input provider encrypts their respective input data using the FHE.pk (Fully Homomorphic Encryption public key). The model parameters, α and β, are also encrypted using FHE.pk to produce FHE ciphertexts c_α and c_β. These encrypted inputs and encrypted model

parameters are sent to the Inference Engine (inference engine) for computation.

$$
\begin{aligned}
c_A &= \text{FHE.Encrypt}(\text{FHE.pk}, x_{SA}) \\
c_B &= \text{FHE.Encrypt}(\text{FHE.pk}, z_{AA}) \\
c_\alpha &= \text{FHE.Encrypt}(\text{FHE.pk}, \alpha) \\
c_\beta &= \text{FHE.Encrypt}(\text{FHE.pk}, \beta)
\end{aligned}
$$

where x_{SA} is Bank A Alice's data and z_{AA} is Teleco S Alice's data.

3. The inference engine evaluates the function on the FHE encrypted inputs and model parameters as follows:

   First (Homomorphic Evaluation): Compute the function $G' = \alpha \cdot x + \beta \cdot z$ on the FHE encrypted data to obtain an FHE encrypted result c_v.

$$
\begin{aligned}
c_{ax} &= \text{FHE.Mul}(c_\alpha, c_A) \\
c_{by} &= \text{FHE.Mul}(c_\beta, c_B) \\
c_v &= \text{FHE.Add}(c_{ax}, c_{by})
\end{aligned}
$$

   Second (Output Decryption): The FHE encrypted result, c_v, is then decrypted to obtain the plaintext credit score v. This is achieved through a **threshold decryption protocol**. Multiple parties, each holding a share of the FHE.sk (FHE secret key), collaboratively use their shares to decrypt c_v. The Inference Engine also participates as one of the FHE.sk share-holders. This ensures that the plaintext score v is recovered correctly and only accessible as per the protocol's design, typically to an authorized receiver upon completion of the multi-party decryption.

4. Parties then collaboratively decrypt the FHE ciphertext c_v (the encrypted credit score) using their respective shares of the FHE secret key (FHE.sk). The result of this threshold decryption, for instance, $\text{FHE.ThresholdDecrypt}(\text{FHE.sk}_1, \text{FHE.sk}_2, \ldots, c_v)$, is the plaintext credit score v. Crucially, the nature of the shared FHE secret key and its use in threshold decryption implies:

   (a) No single party (including the Inference Engine (inference engine), or input providers such as Teleco A or Bank B) possesses the entire

FHE.sk. Each party involved in the threshold scheme holds only a share.

(b) All designated parties holding a share must combine their shares according to the threshold decryption protocol to decrypt c_v. This ensures that the decryption is a collaborative and controlled process. The final plaintext score v is revealed only when a sufficient threshold of these parties participates. (The original inputs, like x_{SA} and z_{AA} from Bank A and Teleco S respectively, and the model parameters α and β, remain protected by their FHE encryption and are not revealed by the decryption of c_v.)

5. The plaintext credit score v (which can be denoted as $\mathsf{creditscore}_{CS}$) is thus obtained by the intended Receiver as the direct result of the successful threshold FHE decryption of c_v. There is no subsequent decryption step involving a separate scheme like CRYSTALS-Dilithium for the credit score itself.

6. To ensure that the Inference Engine (inference engine) correctly computed c_v based on the agreed model G' (where $G' = \alpha \cdot x + \beta \cdot z$), participating parties can perform a verification. Assuming they have access to the FHE encrypted inputs (c_A, c_B) and the FHE encrypted model parameters (c_α, c_β), they can independently re-compute the expected FHE encrypted result:

$$c'_v = \text{FHE.Add}(\text{FHE.Mul}(c_\alpha, c_A), \text{FHE.Mul}(c_\beta, c_B))$$

If their locally computed c'_v does not match the c_v that was provided by the inference engine (and is to be decrypted), the parties abort the protocol and can inform the Receiver that the credit score computation has been compromised or is incorrect. If c'_v matches c_v, they can proceed with or accept the result of the threshold decryption with greater confidence.

Key Observations

- The receiver only observes the credit score.
- The splitting of the FHE secret key enables the inference engine and the inputting parties to verify the computation while preventing leakage of data.

- The method enables verification of the computation where the computation is performed on a single server.
- The method uses LWE , which involves computational complexity and scaling issues.

4.2 Feasibility and scalability of FHE

In our prototype implementation we kept the credit score model intentionally simple (only two input features with a linear relationship) to ensure feasibility. Deploying a larger neural network with multiple layers in Step 2 would significantly increase the FHE computation cost. Each additional hidden layer or non-linear activation requires a substantial number of homomorphic operations (often necessitating expensive bootstrapping to manage noise growth), which dramatically slows down computation. For example, homomorphic inference on even a modest neural network (e.g., one hidden layer with tens of neurons) can take on the order of seconds to minutes per evaluation on current FHE platforms—impractical for real-time scoring. This highlights a trade-off: our cryptographic approach can support flexible, complex models in principle, but the current performance of FHE constrains the model complexity that is practical. Future work should explore optimizations (such as model quantization, ciphertext batching, or hardware acceleration) and carefully balance model complexity with cryptographic overhead to improve scalability.

4.3 Implementation design

The code for implementation of the Vanilla MPC is here `https://anonymous.4open.science/r/fhe-excute-1422/mpc/README.md`. The code for implementation Zama method is here `https://anonymous.4open.science/r/fhe-excute-1422/t_fhe/README.md`.[11]

4.4 Comparison of methods

The following is the list of desired attributes of the cryptographic model that implements the credit score computation.

[11] Writing the code for MPC with Threshold FHE is a future project.

Data Transmission

① **Identity:** Identity of sender is verified.

② **Transaction privacy:** Transmission is private (no eavesdropping).

③ **Data integrity:** Data has integrity (i.e., cannot be tampered with).

Computation of function

④ **No leakage:** No leakage of information (including to IE).

⑤ **Validate function:** Validate function if it is common knowledge.

⑥ **Validate no change:** Validate parameters have not changed over time.

⑦ **Exclusive decryption:** Ensure receiver is the only party who decrypts output of computation.

Online presence

⑧ **No online dependency:** No parties other than IE are online during computation.

Post-quantum secure

⑨ **Post-Q secure:** A quantum computer attacker does not expose new vulnerabilities in the protocol.

Scalability

⑩ **Scaling:** Cost increase as number of operations increases. $\leq$ linear is scalable.

Table 1 in Appendix A compares the performance of each cryptographic model across the set of 10 attributes described above.

4.4.1 Tradeoffs

There are several tradeoffs between the vanilla MPC method and the other two methods in terms of liveness, data privacy and computational complexity.

1. **Liveness** Vanilla MPC involves splitting data and sourcing computation of multiple servers. This requires multiple servers involved in the credit score data pipeline to be online. The Zama method does not require data providers to remain online, but it does rely on the liveness of a blockchain, which requires multiple servers to remain online. Our threshold FHE method involves a single server performing the credit score computation, but it requires multiple servers to be online to verify the computation.

2. **Data privacy** Vanilla MPC secures data by splitting it up between servers, not by encryption. The other methods use secret key encryption schemes. Zama has an attack vector arising from the fact that Zama presides over the generation of the secret key.

3. **Computational complexity and scaling** Vanilla MPC replicates the algebraic computation of the credit score on multiple servers. The other two methods use LWE FHE, which requires matrix multiplication for each computation and additional complexity for the bootstrapping operation to reset the noise parameter after some number of computations.

To summarize, MPC meets all criteria except for its requirement that all parties servers remain online. TFHE meets all criteria except for scalability. Zama has gaps in scalability, exclusivity of decryption and potential leakage of data to Zama. On the other hand, Zama is commercially implementable and is supported by a large community of developers.

Comparing MPC and FHE

A simple comparison of vanilla MPC to FHE elucidates the tradeoff in cost. Both methods compute $G = \alpha \cdot x + \beta \cdot z + w$.

Let C represent, interchangeably the computations (or gates) required to generate G and the number of gates. $C = \{\alpha \cdot x, \beta \cdot z, w\}$ and 3. n is the number of servers involved in the computation.

Vanilla MPC We refer to our formulation in Section 4.1.1 where four parties compute M gates. $n = 4$.

- *Computational complexity*: Each computation involves a very small operation. Adding them together yields complexity $\mathcal{O}(1)$.

- *Rounds of communication*: Secret sharing, each of the multiplications and sharing of outputs requires a round of communication between the four servers. Each round, and therefore the sum of four rounds, involves complexity of $\mathcal{O}(n^2)$
- *Duration of computation* Since complexity is trivial, the time it takes to complete the computation depends on the latency in the communications network connecting the servers and the liveness of the servers. Ignoring latency the duration is $\mathcal{O}(C \cdot n)$.

FHE We refer to the BGV formulation of the learning with errors ("LWE") method.[12] The ciphertext is $\mathbb{A}s+b\epsilon$, where $\mathbb{A} \in \mathbb{Z}_q^{poly(\lambda)xn}$, $s \in \mathbb{Z}_q^n$, $\epsilon \in \mathbb{Z}_q^{poly(\lambda)}$ and $b \in \{0,1\}^{poly(\lambda)}$ is plaintext. s is the secret key and λ is the security parameter, $poly(\lambda) = m$. ϵ denotes the noise.[13]

- *Computational complexity* Each addition requires $poly(\lambda)$ separate additions with complexity $\mathcal{O}(1)$. Each multiplication involves, inter alia, multiplying two mxn matrices with complexity around $\mathcal{O}(mlog(m))$.[14] This is the computational complexity of the entire operation.
- *Rounds of communication* There are only two rounds. Initially, each of three agents send their encrypted data to the inference engine. After the computation is completed the inference engine sends the encrypted output to the receiver.
- *Duration of computation* For a "small" secret key and security parameter, we can assume each computation is very fast. Therefore time is a function of the number of computations, which is $\mathcal{O}(C \cdot poly(\lambda))$.

Comparison The computational complexity of FHE is higher and grows at an exponential rate. The duration of computation of FHE may be comparable for a low security parameter when there are a large number of MPC servers, but it, too, grows at an exponential rate in the security parameter, while MPC grows at a polynomial rate in the number of servers. On the other hand, the timing of an FHE computation may be more reliable, since

[12](Vaikuntanathan et.al. [3] and Regev [10])

[13]Our example is drawn for simplicity. BGV is an FHE method that enables addition and multiplication, but cannot accommodate comparisons. Threshold FHE, which is the FHE method used in the examples above, enables comparisons.

[14]This is the complexity of the fast Fourier transformation ("FFT") for matrix multiplication. See Lecture 3:Divide & Conquer:FFT at MIT OpenCourseWare `https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/resources/lecture-3-divide-conquer-fft/?utm_source=chatgpt.com`.

it depends on only one server, whereas MPC relies on multiple servers.

5 Step 3: Hashing to Prove Model Persistence

The goal of Step 3 is to enable a third-party to verify that Alice's credit score in Step 2 (at time t_1) was generated by the same model derived by federated learning at Step 1 (at time t_0), without leaking any information about Alice's inputs x and z or her credit score. The model information we wish to make verifiable is (i) the algebraic structure of the credit score model and (ii) the values of the parameters α and β, as well as timestamps. Our approach is to hash the model information at each time, $\{t_0, t_1\}$, and verify by comparing hash outputs. To ensure all verifiers use the same reference model hash, the hash output from Step 1 (time t_0) should be published or recorded on an immutable, publicly accessible ledger (e.g., a blockchain or audit log) at the time of model training. This way, every verifier can independently obtain the canonical hash value. Requiring a common, tamper-proof record of the Step 1 hash prevents a malicious inference engine from sending different "original" hashes to different verifiers. In our design, we assume the Step 1 model hash is distributed to all relevant parties (for instance, by posting it to a public ledger or registry) immediately after training. This ensures that all verifiers obtain the same Step-1 hash, preventing server equivocation.

5.1 Description of chosen method

We use a hash function to validate the persistence of the model. The basic idea is that any alteration in the model will change the output of the hash function. The data is encrypted with the SHA 256 hash function, which is post-quantum secure, using a fixed-point representation with quantization (rounding) before hashing, ensuring zero tolerance for numerical changes.[15] It is combined with a digital signature using CRYSTALS-Dilithium, a lattice-based, post-quantum signature algorithm.[16] It provides strong identity attri-

[15] `https://csrc.nist.gov/glossary/term/sha_256`

[16] CRYSTALS-Dilithium has been chosen as one of the first three Federal Information Processing Standards (FIPS) for post-quantum cryptography by the US National Institute of Standards and Technology `https://csrc.nist.gov/News/2024/postquantum-cryptography-fips-approved`.

bution (authenticity) due to the digital signature, ensuring that data can be verifiably linked to its creator. Transaction privacy is maintained since the hashed representation hides underlying details, preventing direct inference of secret parameters. Data persistence is robustly enforced, as any alteration of the data invalidates the hash, and only authorized parties holding the private signing key can generate valid signatures, though it does not directly support selective or exclusive decryption. Scalability is excellent, benefiting from the computational simplicity of hashing and the efficient signing and verification capabilities provided by CRYSTALS-Dilithium. The data structures are;

- Hash preimage representation:

$$\|Equation : \alpha * x + \beta * z\|Parameters : \{\alpha, \beta\}\|Metadata : TimestampT\|$$

- Fixed-point representation: Converts parameters $\{\alpha, \beta\}$ into integer form by multiplying by a predefined scale factor (default: 1000 for three decimal places). This eliminates nondeterministic rounding difference that can cause identical preimages to generate different outputs, as can occur with floating-point representations.
- Deterministic serialization: Ensures that the equation representation is always the same before hashing.

We implement the procedure with the following technology.

- SHA-256 hashing: Computes a unique hash for the equation representation. Verification function: Compares hashes to check for consistency.
- UNIX timestamps to capture when the equation was recorded.
- CRYSTALS-Dilithium signatures to prove the authenticity of the preimage. This is the NIST Post-Quantum Cryptography designated primary digital signature standard.
- Public key verification so that an observer can independently verify that the stored hash was generated using the correct private key.

Regarding storage and retrieval, we consider two options:

- Centralized storage: The hash and signature are stored in a database or file.
- Decentralized storage: Using blockchain or distributed ledger technology for tamper-proof verification.

5.2 Implementation design

See code here `https://anonymous.4open.science/r/fhe-excute-1422/hash/README.md`

5.3 Comparison of Alternative methods

We describe two commercially implementable alternative methods and one emerging method that include tradeoffs with our chosen method for some attributes.

- **Pedersen Commitments + ZKP:** (ensures transaction privacy and data binding via commitments, plus ZK proofs for correctness). Unlike Hash+Sig with CRYSTALS-Dilithium, it lacks built-in identity attribution and is not post-quantum secure. However, identity verification can be layered on by having the model owner digitally sign the commitment (analogous to how we sign the hash in our method). Of course, using a post-quantum signature scheme would be necessary to maintain post-quantum security in such a design. Transaction privacy and computational correctness validation are strong, and exclusive authorized decryption is supported, but scalability is moderate due to the overhead of generating zero-knowledge proofs. [17]
- *Merkle Tree + zk-STARK* This post-quantum secure method combines Merkle trees for efficient persistence commitments and zk-STARK proofs for verifiable computation and zero-knowledge privacy. Unlike Hash+Sig with CRYSTALS-Dilithium signatures, it validates computational correctness and maintains transaction privacy rigorously through zero-knowledge proofs. On the other hand, it does not provide identity attribution or exclusive decryption, which Hash+Sig with CRYSTALS-Dilithium signatures provides through cryptographic signatures. Merkle trees are highly scalable. zk-STARK's have lower computational overhead than traditional ZKP methods like zk-SNARKs, but zk-STARK proofs still incur significant computational overhead and produce larger proofs, resulting in moderate scalability relative to hashing or simple digital signatures.[18]
- *ZKML and opML frameworks:* An additional approach - not yet com-

[17] `https://github.com/BenWolfaardt/interactive_ZKP`
[18] `https://github.com/remonyffenegger/stark-tutorial`

mercially viable - is to employ emerging *verifiable ML* techniques to ensure model integrity. Zero-knowledge machine learning (ZKML) frameworks use zero-knowledge proof systems to cryptographically prove that a model's training or inference was performed correctly on the given data and model parameters, without revealing any sensitive information. Optimistic machine learning (opML) approaches, on the other hand, execute the model inference in a decentralized, trust-minimized environment (such as a blockchain or rollup) under the assumption of honesty, and rely on a challenge mechanism whereby any incorrect result can be detected and economically penalized. These approaches could strengthen Step 1 by providing guarantees that the model has not been tampered with and that the inference computations are correct, beyond the simple hash check we use. However, current ZKML implementations are computationally expensive for complex models (proving the correctness of a deep neural network can be prohibitively slow), and opML requires a robust on-chain infrastructure and careful incentive design to be effective. Therefore, we did not integrate these techniques into our present workflow. They remain promising avenues for future work to enhance the cryptographic integrity of the model training and inference process.

Table 2 in Appendix B compares our chosen method, Hash+Sig with CRYSTALS-Dilithium signatures, to relevant alternatives on attributes related to those listed in Section 4.4.

5.3.1 Tradeoffs

The hashing method meets all criteria except for its inability to limit disclosure - any party can compare the output of the hashes - and function validation - verification is limited to comparison of hash outputs. Pederson commitments meet all criteria except for post-quantum and scalability (the former is disqualifying). Merkle performs similar to hashing except that it is not scalable.

5.4 Limitations of hash-based verification

It should be noted that while our hash+signature method allows verifiers to detect any change in the model's parameters between training and deployment, it does *not* guarantee that the credit score output was actually

computed using those parameters. In principle, a malicious inference engine could produce a credit score by some arbitrary means (or using a different model) and still attach the correct hash of the original model parameters, thereby passing the Step 3 check. In our current design we assume the inference engine faithfully uses the Step 1 model in Step 2's computation (an assumption that might be enforced by audits or reputation in practice). Removing this assumption would require a stronger cryptographic proof of correct execution, such as a SNARK/STARK that verifies the computation of the score was carried out with the committed model parameters. Implementing such end-to-end verifiable computation is non-trivial and is left for future work.

6 Step 4: Zero-Knowledge Proof of Compliance

The goal of step 4 is to enable a third party to verify certain properties of the model without leaking information beyond the subject matter. The generic question we address is whether an inequality of the form $|x - y| < n$ holds, where $x,\ y$ and n are scalers. We employ ZKP to answer this question and apply it to the Federated Learning model of Step 1 and the credit score computation of Step 2.

- **Testing for compliance with parameter restrictions** Financial regulations may impose certain restrictions on the credit score model, such as the weight of certain attributes. Anti-discrimination laws might limit the extent to which certain of Alice's attributes, such as her age or gender, affect the credit score. Targeted neighborhood development policies might require the amplification of the effect of certain attributes, such as the location of Alice's residence. For our application we suppose there is a rule that limits the relative weighting of the two parameters, α and β in the credit score model to at most n. The inference engine proves to the regulator that $|\alpha - \beta| < n$.[19]
- **Detecting fraud** The inference engine does not observe Alice's attributes and therefore cannot confirm the correctness of the attribute

[19] Another compliance test that can be carried out with ZKP is to prove that the units in Step 1 used to proscribed method - in our model backpropagation on a neural network - in estimating model parameters.

values $\{x_{BA}, z_{AA}\}$ sent by Bank S and Teleco A. It can, however, use statistical inference to detect incorrect data. Moments of distributions of population attributes can be inferred from the data stored by the units in Step 1. Suppose the moment of interest is the absolute value difference between the two attributes. The recipient considers an unusually wide dispersion in attributes to indicate possible error. It will disregard Alice's credit score when the estimated probability of the occurrence of a dispersion that exceeds Alice's observed dispersion is below some threshold q.[20] The inference engine constructs the distribution of absolute values as follows. Each unit can sends the absolute value difference for each borrower in its database to the inference engine, or, to prevent leakage of borrower information, averages over clusters of borrowers. The inference engine constructs an empirical distribution and chooses a right tail cutoff value n such that $q\%$ of the borrower population is estimated to have the moment above n. The inference engine proves to the recipient that $|x_{SA} - z_{AA}| < n$.[21]

6.1 Description of the chosen method

We use an version of **zk-STARK** implemented by Meta's Winterfell, which is an open-source STARK prover and verifier (Winterfell [8]). STARK's require more computation than SNARK's, but its proving complexity is quasi-linear (often on the order of $O(n \log n)$ for n operations due to FFT-based polynomial commitments), and verification is sublinear (typically $O(\log n)$ or slightly higher, often dominated by hashing costs). So, it i roughly linear, which meets our scaling criteria.zk-STARK meets all of the criteria save one. It does not limit the verifier, however this can be remedied by adding an authentication channel. The FHE methods also meet all of our criterial. We prefer this method for two reasons; (i) the AIM coding language is high-level and relatively easy to implement and (ii) Winterfell is actively maintained by Meta.

[20] We are describing a one-sided $p - value$.

[21] A similar application would be to test proprieties of the parameters reported by the units in Step 1 for fidelity with prior model knowledge.

6.2 Implementation design

See code here `https://anonymous.4open.science/r/fhe-excute-1422/zkp/README.md`

6.3 Comparison of alternative methods

There are several alternative commercially available ZKP protocols that could perform the task in Step 4. We compare five proof techniques with respect to attributes related to those listed in Section 4.4. The techniques examined are:

- *GKR Protocol* This is an interactive proof system, which requires that prover and verifier remain online throughout the transaction.[22] GKR can be extended by an authentication channel prove the verifier identity, thereby limiting the disclosure to selected parties. GKR is doubly efficient (therefore scalable). The verifier cost grows at an approximate linear rate in the size of the original computation, e.g.,proving a circuit of N gates might take on the order of O(N) or O(N log N) time, but not worse, and verification is extremely fast (does not depend on N).
- *zk-SNARK.* This is a succinct non-interactive zero-knowledge proof, which means the verifier does not need to stay online during the transaction.[23] It scales at a sublinear rate. It's main drawback is that it is not post-quantum secure.
- *FHE-based NIZK* This approach uses FHE to build a non-interactive ZKP. There are two methods. One is threshold FHE. [24] The other is the CKKS FHE scheme.[25] Both are based on ring-LWE. There are two salient tradeoffs. One is that CKKS can have lower complexity for certain tasks, but can also be less exact compared to TFHE.

Table 3 in Appendix Ccompares our chosen method, $zk - STARK$ and relevant alternatives on certain attribute dimensions.

[22] `https://github.com/VirPathak00/gkr-python-implementation`
[23] `https://github.com/Merricx/zksnake/`
[24] `https://github.com/openfheorg/openfhe-python`
[25] `https://github.com/sarojaerabelli/py-fhe`

6.3.1 Tradeoffs

Only two methods are scaleable, zk-SNARK and GKR. The latter is interactive, which means that the verifier needs to stay online. No other method is scalable. The tradeoff between the two is the online requirement (in favor of zk-SNARK) and the verifiability of identity (in favor of GKR).

6.4 Limitations of compliance testing

We have used a linear model with only two parameters. In a more complex scenario involving a multi-layer neural network with many parameters and non-linear activation functions, it becomes significantly more challenging to prove a property like "no attribute influences the score beyond a certain limit." The influence of a given input feature in a deep model is dispersed across numerous weights and layers, making it infeasible to capture with a single inequality constraint such as $|\alpha - \beta| < n$. Enforcing analogous regulatory constraints on a complex model would likely require constructing a complete zero-knowledge proof of the model's inference computation or imposing structural restrictions on the model to make its behavior more transparent. Given the current state of ZK proof technology, proving compliance properties for large neural networks is impractical. For this reason, we focus on the simple two-weight example to demonstrate how compliance auditing could work in principle. Extending such proofs to richer models is left for future exploration.

6.5 Security Assumptions and Adversaries

Our proposed workflow relies on certain trust assumptions at each stage of the pipeline. We assume that the participants in the federated learning step are honest (or at most semi-honest): the data-providing institutions do not maliciously poison the training data or model updates, and the aggregator/inference engine follows the prescribed protocol during model training. In Step 2, if a third-party cloud service performs the FHE computation, we trust that service to dispose of secret keys and not deviate from the protocol (in practice, this trust could be reduced by distributing the FHE computation among multiple parties or using a threshold FHE scheme so no single party knows the entire secret key). We also assume that any ledger or blockchain used in Steps 3 and 4 is reliable and tamper-proof, providing a trustworthy

record of hashes and proofs. Admittedly, these strong assumptions make the security guarantees of our system more contingent on participant honesty, which in turn can make some cryptographic protections appear unnecessary if every party behaves correctly.

Nevertheless, in a more adversarial setting, additional measures would be required to strengthen security when these trust assumptions do not hold. A fully malicious inference engine, for example, would necessitate a verifiable computation mechanism (as noted, a ZK-SNARK/STARK for the scoring step) to prove it applied the model correctly. Malicious or colluding data providers in Step 1 might require robust aggregation techniques or outlier detection to prevent poisoning attacks. A more formal security analysis—specifying adversary capabilities and security goals for each stage—is needed to rigorously guarantee security if participants may deviate. Providing such a formal treatment (and corresponding proofs of security) is beyond the scope of our current work; however, we acknowledge its importance. In future work, we plan to refine our threat model and reduce these trust assumptions, for instance by employing protocols secure against malicious participants and by formally proving the cryptographic security properties of the entire workflow.

Additionally, we have not yet addressed certain well-known attack vectors against federated learning and the deployed model. For instance, even without direct data sharing, an honest-but-curious aggregator could attempt *model inversion* attacks on the Step 1 model updates to infer sensitive information about individual training data, or perform *membership inference* to determine if a specific person's data was included in training. A malicious participant in FL could also inject carefully crafted (poisoned) data or model updates to influence the model (so-called model poisoning attacks). Our current implementation does not include defenses like differential privacy (which could mitigate inference attacks by adding noise to updates) or secure multiparty aggregation (which could hide individual contributions), so it implicitly assumes that such attacks will not be carried out. Similarly, we assume the inference engine and data providers do not engage in any unauthorized data extraction or manipulation beyond the protocol. These are acknowledged limitations of our prototype. In a production deployment, one would want to incorporate additional privacy-enhancing techniques and robust aggregation methods to guard against data leakage and malicious behavior. We consider integrating these protections as important future work

to improve the system's resilience against adversarial attacks.

7 Conclusion

In this study, we introduced a comprehensive, post-quantum secure workflow for credit scoring, addressing critical issues surrounding data privacy, security, and computational integrity. By integrating federated learning, fully homomorphic encryption (FHE), cryptographic hashing, and zero-knowledge proofs (ZKPs), our approach mitigates vulnerabilities posed by quantum computing while maintaining robust credit risk assessment practices. The comparative analysis of alternative cryptographic methods revealed essential trade-offs, underscoring our selected techniques' optimal balance between security, scalability, and practical implementation.

Our workflow demonstrates that post-quantum cryptographic methods can feasibly be integrated into complex financial processes without compromising computational performance or regulatory compliance. Furthermore, the provided codebase and simulation offer a practical foundation, enabling financial institutions to smoothly transition these advanced technologies from theory into operational environments. Future enhancements could expand this model to handle richer datasets and more sophisticated credit scoring algorithms, ensuring continued relevance and adaptability in evolving financial and technological landscapes.

References

[1] Daniel Aronoff, Adithya Bhat, Panagiotis Chatzigiannis, Mohsen Minaei, Srinivasan Raghuraman, Robert M. Townsend, and Nicolas Xuan-Yi Zhang. SoK: Fully-homomorphic encryption in smart contracts. Cryptology ePrint Archive, Paper 2025/527, 2025.

[2] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3), July 2014.

[4] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Paper 2018/421, 2018.

[5] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official Journal of the European Union, L 119, pp. 1–88, April 2016.

[6] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012.

[7] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[8] Meta. Winterfell: A stark prover and verifier library. `https://github.com/facebook/winterfell`, 2024. Accessed: 2024-06-24.

[9] PQConnect Project. Pqconnect: Post-quantum cryptography library and tools. `https://www.pqconnect.net/index.html`, 2024. Accessed: 2024-06-24.

[10] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), September 2009.

[11] United States House of Representatives Committee on Oversight and Government Reform. The equifax data breach. Technical report, U.S. Government Publishing Office, December 2018.

A Table 1.: Comparison of Methods to Compute Credit Score

Table 1 compares the performance of each cryptographic model across the set of 10 attributes described above.

Attribute	**Vanilla MPC**	**Zama FHE on Blockchain**	**MPC with Threshold FHE**
Identity	✓	✓	✓
Transaction privacy	✓	✓	✓
Data integrity	✓	✓	✓
No leakage	✓		✓
Validate function	✓	✓	✓
Validate no change	✓	✓	✓
Exclusive decryption	✓		✓
No online dependency		✓	✓
Post-Q secure	✓	✓	✓
Scalability	✓		

Table 1: Attributes of Cryptographic Methods for Credit Scoring

B Table 2: Comparison of Cryptographic Methods for Proving Model Persistence

Table 2 compares our chosen method, Hash+Sig with CRYSTALS-Dilithium signatures, to relevant alternatives on attributes related to those listed in Section 4.4.

C Table 3: Comparison of Methods for Zero-Knowledge Proof of Compliance

Table 3 compares our chosen method, $zk-STARK$ and relevant alternatives on certain attribute dimensions.

Attribute	**Hash+ Sig**	**Pedersen+ ZKP**	**Merkle+ ZKP**
Identity (Authorship)	✓		
Transaction Privacy	✓	✓	✓
Data persistence	✓	✓	✓
No Secret Leakage	✓	✓	✓
Validate Correct Function		✓	✓
Validate Unchanged (Over Time)	✓	✓	✓
Exclusive Decryption (Authorized)		✓	
Post-Quantum Security	✓		✓
Scalability (Efficiency)	✓		

Table 2: Comparison of Cryptographic Methods for Proving Model Persistence

Attribute	**zk-STARK**	**GKR**	**zk-SNARK**	**FHE NIZK** (TFHE)	**FHE NIZK** (CKKS)
Prover's Identity Verifiable		✓			
Transaction Privacy (No Eavesdrop)	✓	✓	✓	✓	✓
Data Integrity (Tamper-Proof)	✓	✓	✓	✓	✓
No Information Leakage (ZK)	✓	✓	✓	✓	✓
Validates Commitment	✓	✓	✓	✓	✓
No Change to Underlying Data	✓	✓	✓	✓	✓
Exclusive Decryption (Designated)				✓	✓
No Online Dependency (Offline)	✓		✓	✓	✓
Post-Quantum Secure	✓	✓		✓	✓
Scalability ($\leq$ Linear Cost Growth)	✓	✓			

Table 3: Comparison of Cryptographic Methods for ZKP