Deployment and Orchestration of Quantum-Secured Federated Learning workloads

Pol González, Marc Ruiz, Jaume Comellas, and Luis Velasco*

Optical Communications Group (GCO), Universitat Politècnica de Catalunya (UPC), Barcelona, Spain

**e-mail: luis.velasco@upc.edu*

ABSTRACT

Federated Learning (FL) enables decentralized Machine Learning (ML) by allowing clients to train a shared model locally, reducing privacy risks and eliminating centralized data collection. However, FL deployments face significant challenges, related to orchestration heterogeneity, scalability, and security. Clients often differ in computational capacity, availability, and connectivity, requiring advanced orchestration mechanisms for efficient resource management and reliability. Moreover, FL communications remain vulnerable to attacks such as eavesdropping and model poisoning. In this paper, we showcase an architecture for deploying and orchestrating FL workloads secured with quantum keys and post-quantum cryptography to ensure confidentiality and integrity against classical and quantum threats.

Keywords: Federated Learning, Post-Quantum Cryptography

1. Introduction

Federated Learning (FL) proposes a different way of implementing Machine Learning (ML) workloads, moving from the classic centralized data model to a decentralized and heterogeneous architecture [1]. In a conventional ML environment, raw data is collected from different sources and then conveyed to a central server for processing. By moving the data from local sources to a centralized site, we compromise the data privacy and add a single point of failure in case of data leaking. By allowing devices to train a shared global model locally and only exchange their local results, FL enables distributed intelligence while ensuring that sensitive information does not leave the local device. However, the implementation of FL systems introduces new challenges related to the orchestration and deployment of FL workloads [2]. FL environments are usually heterogeneous, consisting of clients with different computational capabilities, availability, and network connectivity. Orchestration mechanisms are required to manage resource allocation, connectivity, client selection, aggregation strategies, and fault tolerance while ensuring scalability and reliability.

Despite the privacy preservation capabilities introduced by FL, security is not guaranteed on the communication channels used to exchange model updates and control messages. While FL reduces the need for sending raw data, it does not protect against security attacks such as eavesdropping or model poisoning [3]. Securing network communications is necessary to preserve the privacy and integrity of federated learning deployments. Conventional security mechanisms rely on classical public-key cryptographic solutions for authentication, key exchange, and secure transport, which are commonly used in web services and microservices. However, the advance of quantum computing and the development of more powerful quantum computers imply a threat to classical cryptographic algorithms [4]. Post-Quantum Cryptography (PQC) addresses this challenge by developing algorithms that are resistant to classical and quantum attacks. In opposition to quantum-based security solutions that require specialized hardware on all the compromised locations, PQC algorithms are designed to be compatible with environments where only one site is inside the secure quantum perimeter. The authors in [5] proposed a solution where a Quantum Random Number Generator (QRNG) inside the secure perimeter is used to generate quantum keys that are encapsulated and exchanged using the Bit-flipping Key Encapsulation (BIKE) scheme. This approach allows symmetric encryption while only one of the sites is inside the security perimeter. In this paper, we showcase an architecture to deploy and orchestrate PQC secure FL workloads based on the aforementioned solution, involving a QRNG inside a security perimeter.

2. Federated Learning PQC Deployment and Operation

Figure 1 presents the network scenario with several locations where FL workloads can be deployed. The Telecom Cloud contains a Key Management System (KMS) that manages the access and key generation of the QRNG system. In the control plane, a Customer Orchestrator and a Telecom Orchestrator are responsible for the deployment and orchestration of FL workloads in the Customer and Telecom Cloud respectively. The Customer Orchestrator receives requests from the user to deploy FL workloads and is in charge of allocating resources in the Customer Cloud, deploying the workloads and orchestrating the FL operation. Moreover, it performs an initial key exchange between the server and the client that generates a unique header used for synchronization purposes between the server and the clients. The connectivity between the Telecom Cloud and the different locations of the Customer Cloud is established by the Telecom Orchestrator through a Software Defined Network (SDN)

979-8-3195-4420-9/26/$31.00

Controller. Each FL deployment consists of a PQC Server and several PQC clients; these components act as a proxy between the FL application and the PQC encryption procedure. This solution offers a technology-agnostic approach regarding the FL framework used.

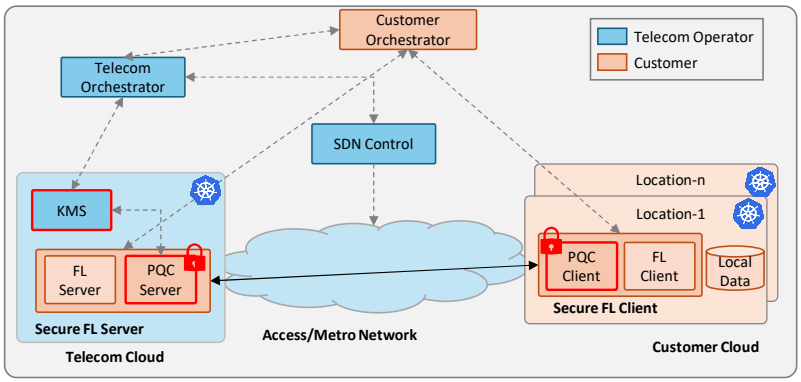

Figure 1. Proposed Architecture

The proposed deployment workflow is presented in Figure 2. The workflow starts when a customer requests the creation of a new FL deployment through the UI (see step 1 in Figure 2). This petition is handled by the Customer Orchestrator, where an FL deployment descriptor is generated (step 2). This step includes the creation of the configuration for each of the components involved in the FL setup. The Customer Orchestrator sends a request to the Telecom Orchestrator to deploy a new Secure FL Server (step 3). The Telecom Orchestrator deploys the pods hosting the FL Server on the Telecom Cloud K8s (step 4). Then, it registers the new FL deployment on the KMS, granting the FL Server the capability to retrieve keys from it (step 5). The Customer Orchestrator receives the response of the Telecom Cloud which includes the assigned endpoint for the FL Server to retrieve the keys from the KMS. Then, the Customer Orchestrator deploys the necessary software in the locations comprising the Customer Cloud; this includes the PQC components and the FL components (step 6).

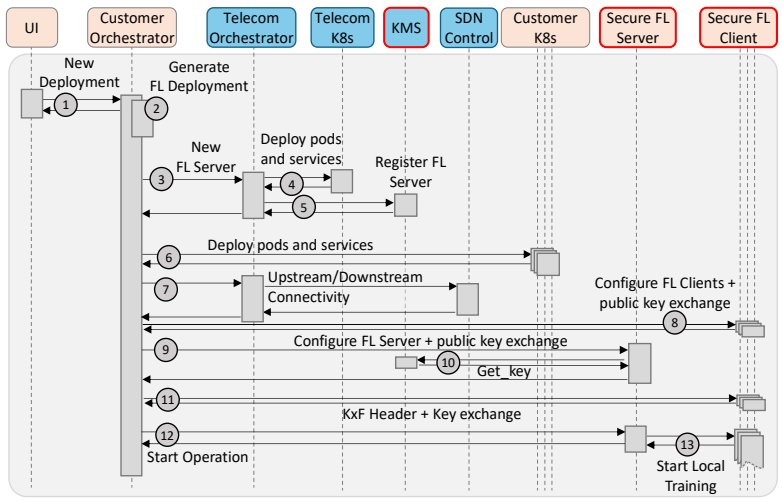

Figure 2 Deployment Workflow

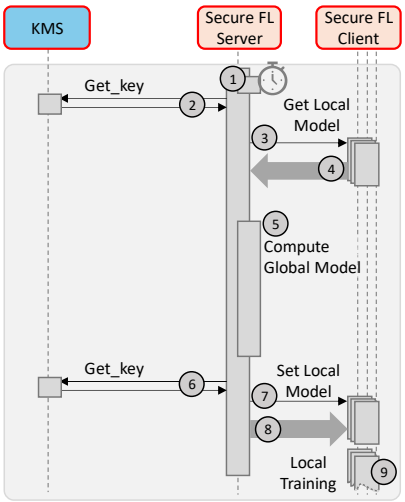

Figure 3 Operation Workflow

Then, the Customer Orchestrator requests the Telecom Orchestrator to establish the connectivity between the Telecom Cloud and the Customer Cloud locations through the SDN Controller (step 7). Next, the Customer Orchestrator configures the public key of the FL Clients to be used to generate the Key Exchange Frame (KxF) header pattern (step 8). This will be used by the FL Client to detect whenever a new key is sent by the FL Server. The public key is collected and sent back to the Customer Orchestrator. The Customer Orchestrator sends the FL Client public key to the FL Server (step 9) and asks it to retrieve a key from the KMS that will be used as a seed for data encryption (step 10). Specifically, the FL Server uses the retrieved seed to generate the session key and the ciphertext. The session key is used to encrypt a known bit stream and the ciphertext is added to the KxF header that is inserted periodically. After this, the generated KxF header pattern and the ciphertext are sent back to the Customer Orchestrator, which sends them to the FL Client (step 11). The ciphertext is then decapsulated using the private key of the FL Client to get the session key to decrypt the FL Data. Finally, the Customer Orchestrator sends a confirmation to the Secure FL Server and Secure FL Client to start FL operation (steps 12 and 13).

The operation workflow is proposed in Figure 3. Periodically, the FL Server resumes its execution to compute and update the global model (step 1 in Figure 3). The FL Server retrieves a new key from the KMS (step 2) that

will be used as the current session key. The FL Server sends a request to the FL Clients to transfer their local models; this petition includes the new key (step 3). The FL Clients collect and encrypt the last version of their local model and transfer it to the FL Server (step 4). The FL Server computes a new global model using the data sent by the FL Clients (step 5). Before updating the FL Clients with the new global model, the FL Server retrieves a new key from the KMS to be used as session key (step 6). The FL Server notifies the FL Clients that the new global model is ready and will be transferred (step 7). The FL Server transfers the new global model to the FL Clients (step 8) and the FL Clients resume their local training with the updated model (step 9). This operation is repeated until the FL training session finishes.

3. Experimental Evaluation and Results

The evaluation setup for the proposed solution is hosted in an on-premises OpenStack virtualized environment, where three distinct locations each run a Minikube Kubernetes cluster [6]. Minikube was selected because it provides most production-level Kubernetes features while remaining simple to deploy and maintain, and its use of the latest API versions ensures a seamless migration to classic Kubernetes clusters. Each location is hosted in a separate Virtual Machine (VM), while the Customer Orchestrator, Telecom Orchestrator, and PQC components are implemented in Python and deployed in their own dedicated VMs. The evaluation employs the Flower framework [7] to handle client-server communication and the orchestration of the Federated Learning (FL) training sessions. Because Flower is agnostic to the specific ML technology, it enables the definition of custom clients and servers while offering predefined strategies for aggregating data at the central server. A KMS has been developed in Python, integrating an emulated QRNG. However, in the presence of real quantum equipment, the code can be easily adapted to retrieve the keys using the standard ETSI GS QKD 014 interface.

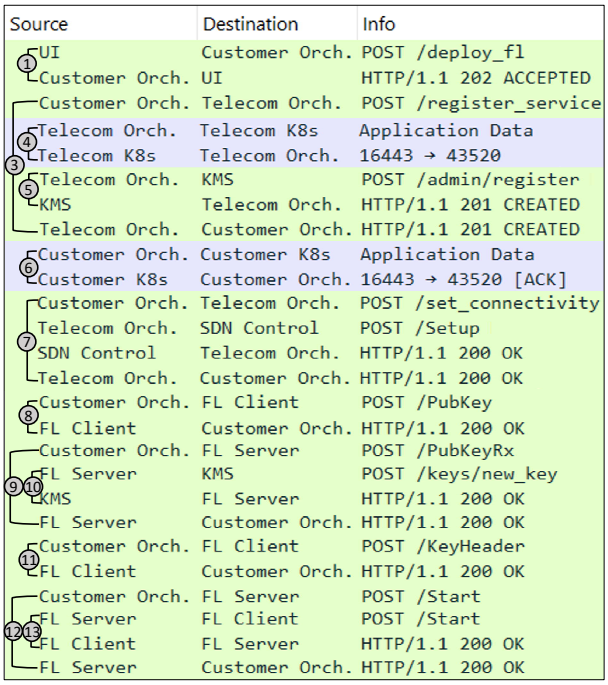

Figure 4 Deployment Workflow

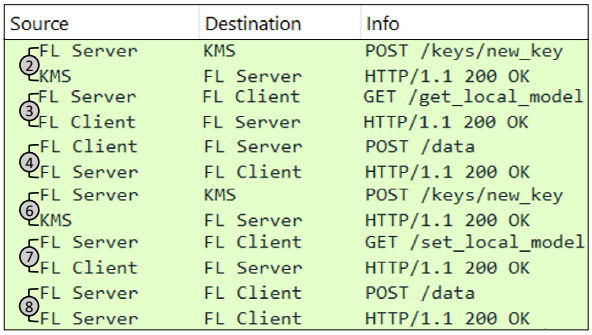

Figure 5 Operation Workflow

Figure 4 shows a capture of the messages exchanged during the deployment workflow, following the same numbering convention as in Figure 2. The deployment procedure is triggered when the customer submits a request through the Web Dashboard to instantiate a new FL service. This request contains the service identifier as well as a detailed specification of the logical names and physical locations where the FL Server and the FL Clients are deployed. After this request, the workflow is executed asynchronously, enabling the user to monitor the deployment status in real time without blocking the interface. The request is first sent to the Customer Orchestrator (message 1), which generates the corresponding FL deployment descriptor. This descriptor defines the configuration of the Kubernetes pods hosting the PQC Clients, PQC Server, Flower Clients, and Flower Server, including their resource requirements and networking parameters. To guarantee connectivity within and outside the Kubernetes cluster, multiple services are instantiated. For intra-cluster communication between PQC and Flower components, standard ClusterIP services are deployed. For external communication between the PQC components and the Customer Orchestrator, each pod exposes a NodePort service. Next, the Customer Orchestrator requests the Telecom Orchestrator to provision a new Secure FL Server (message 3). The Telecom Orchestrator allocates the required compute, storage, and networking resources in the Telecom Cloud to deploy and configure both the FL Server and the PQC Server (message 4). Once instantiated, the FL Server is registered with the KMS using its unique service identifier (message 5), authorizing it to retrieve keys to be used for establishing the secure communication. After finishing the server-side deployment, the Customer Orchestrator coordinates with each selected customer location to deploy the client-side components (message 6), including the pods running the FL Client and PQC Client software. The Customer Orchestrator sends a connectivity request to

the Telecom Orchestrator, which forwards it to the SDN controller for network configuration (message 7). This request specifies the IP addresses and ports associated with all exposed endpoints. To enable the initial secure communication between the FL components, the Customer Orchestrator performs a Diffie–Hellman key exchange using pre-generated key pairs (messages 8 and 9). The FL Server then requests a new symmetric key from the KMS, specifying the required key length in bytes (message 10). A unique synchronization header is generated, encrypted together with the newly obtained key using the previously established shared secret, and forwarded to the FL Client via the Customer Orchestrator (message 11). Once this exchange is completed, secure communication between FL components can start. Finally, the Customer Orchestrator sends a confirmation message to the FL Server (message 12), which forwards the confirmation to all FL Clients (message 13).

Figure 5 shows a capture of the messages exchanged during the operational workflow, following the same numbering scheme introduced in Figure 3. Unlike the deployment phase, this workflow is executed periodically and corresponds to the iterative training process of the FL deployment. At each predefined interval, the FL Server resumes execution and initiates a new training round. Before requesting the local updates, the FL Server retrieves a new key from the KMS (message 2). This key is used as a seed to derive the session key that will protect all communications during the current round. Once the key is obtained, the FL Server sends a control message to the FL Clients requesting them to perform local training and submit their updated models (message 3). This message includes the last retrieved key, allowing the clients to derive the corresponding session key for secure communication. On the client side, the PQC Client exposes a TCP socket that monitors outbound traffic generated by the FL Client. Any captured data, in particular, the serialized local model parameters, is encrypted by the PQC Client and forwarded to the PQC Server. The PQC Server decrypts the payload and conveys it to the FL Server (message 4), ensuring confidentiality between the two endpoints. After receiving all local model updates, the FL Server performs the aggregation step to compute the new global model. Before distributing this model, it retrieves another fresh key from the KMS (message 6), which will serve as the session key for the downstream transmission phase. The FL Server then notifies the FL Clients that a new global model is available (message 7). This notification includes the last generated key, enabling the clients to decrypt the incoming data. Then, the FL Server transmits the updated global model. The PQC Server, which maintains a TCP socket listening to outgoing traffic from the FL Server, intercepts the transmitted data, encrypts it, and forwards it to the appropriate PQC Clients. Upon reception, each PQC Client decrypts the payload and sends it to its corresponding FL Client, which updates its local model (message 8). This process is executed periodically until the decommission of the FL service or reaching the maximum number of training rounds defined by the user.

4. Conclusions

To conclude, a secure PQC FL Orchestration architecture has been demonstrated on a setup with three different locations, one of them including an emulated QRNG residing in the security perimeter. The solution has been tested with a well-known FL framework that has been integrated within an existing PQC solution. The experimental results confirm that the proposed orchestration workflow successfully deploys and operates FL services while protecting model exchanges with quantum-derived session keys encapsulated through PQC mechanisms. Moreover, the modular and technology-agnostic design of the architecture enables its integration with different FL frameworks and cloud environments, requiring only one site to reside within the quantum security perimeter.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union's Horizon Europe research and innovation programme under G.A. No. 101092766 (ALLEGRO), the project PID2024-157824OB-I00 funded by MICIU/ AEI/10.13039/501100011033 /FEDER, UE and from ICREA.

REFERENCES

[1] M. Arbaoui *et al.*, “Federated Learning Survey: A Multi-Level Taxonomy of Aggregation Techniques, Experimental Insights, and Future Frontiers,” ACM Transactions on Intelligent Systems and Technology, vol. 15, pp. 1-69, 2024.
[2] J.-M. Parra-Ullauri *et al.*, “kubeFlower: A privacy-preserving framework for Kubernetes-based federated learning in cloud–edge environments,” Future Generation Computer Systems, vol. 157, pp. 558-572, 2024.
[3] P. Liu *et al.*, “Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives,” Cybersecurity, vol. 5, 2022.
[4] D.-T. Dam *et al.*, “A Survey of Post-Quantum Cryptography: Start of a New Race,” Cryptography, vol. 7, 2023.
[5] L. Velasco *et al*., “Scenarios for Optical Encryption Using Quantum Keys,” Sensors, vol. 24, 2024.
[6] Minikube [On-line] https://minikube.sigs.k8s.io/docs/
[7] Flower FL Framework [On-line] https://flower.ai/