Université Jean Monnet

Master 2, DSC 2025-2026
Sécurité
Projet

Sécurité dans le Federated Learning

Élèves :
Mohamed MOUDJAHED

Professeur :
Pierre-Louis CAYREL

9 décembre 2025

Table des matières

Résumé

Dans le cadre de l'UE Sécurité, notre projet porte sur la cryptographie appliquée au **Federated Learning** [9]. Ce choix vise à combiner les enseignements des UE Deep Learning et Sécurité, afin de renforcer nos compétences dans ces domaines. Nous avons étudié et implémenté des mécanismes de protection des échanges de modèles, notamment la signature des poids via **RSA** [10] pour authentifier la provenance du modèle, l'agrégation sécurisée [2] (**Secure Aggregation**) via la génération de masques pairwise à l'aide du **protocole de Diffie-Hellman** [5]. Nos expériences, menées sur le jeu de données **MNIST** [8], illustrent la capacité de ces techniques à protéger les informations sensibles tout en préservant les performances des modèles fédérés, affichant de bons résultats comparables à une approche centralisée.

1 Introduction

1.1 Présentation

Le Federated Learning (FL), ou apprentissage fédéré, est un paradigme d'apprentissage collaboratif dans lequel plusieurs clients, appareils, organisations ou institutions, entraînent ensemble un modèle de machine learning sans jamais partager leurs données brutes. Contrairement au cadre centralisé, chaque participant conserve ses données localement et n'envoie au serveur que des mises à jour du modèle (poids, gradients ou autres paramètres). Le serveur joue le rôle d'orchestrateur, il agrège ces contributions pour produire un modèle global, qui est ensuite redistribué aux clients.
Ce mode d'apprentissage permet de répondre à plusieurs enjeux importants :

- **Confidentialité** : les données sensibles restent sur le site d'origine et ne sont pas transférées.
- **Conformité réglementaire** : le FL facilite le respect de contraintes légales (par exemple le RGPD).
- **Scalabilité** : l'entraînement distribué exploite les ressources de calcul locales sans surcharger un serveur central.
- **Robustesse** : l'agrégation d'informations provenant de sources non identiquement distribuées (non-IID) peut enrichir le modèle global.

Parmi les algorithmes d'agrégation, FedAvg [9] (Federated Averaging) est le plus utilisé, il calcule la moyenne pondérée des mises à jour envoyées par les clients.

1.2 Exemple d'utilisation

Cas d'usage : Diagnostic médical collaboratif

Considérons un réseau de quatre hôpitaux souhaitant entraîner conjointement un modèle de détection de tumeurs cérébrales à partir d'images IRM, sans partager les données médicales de leurs patients pour des raisons de confidentialité et de conformité au RGPD.

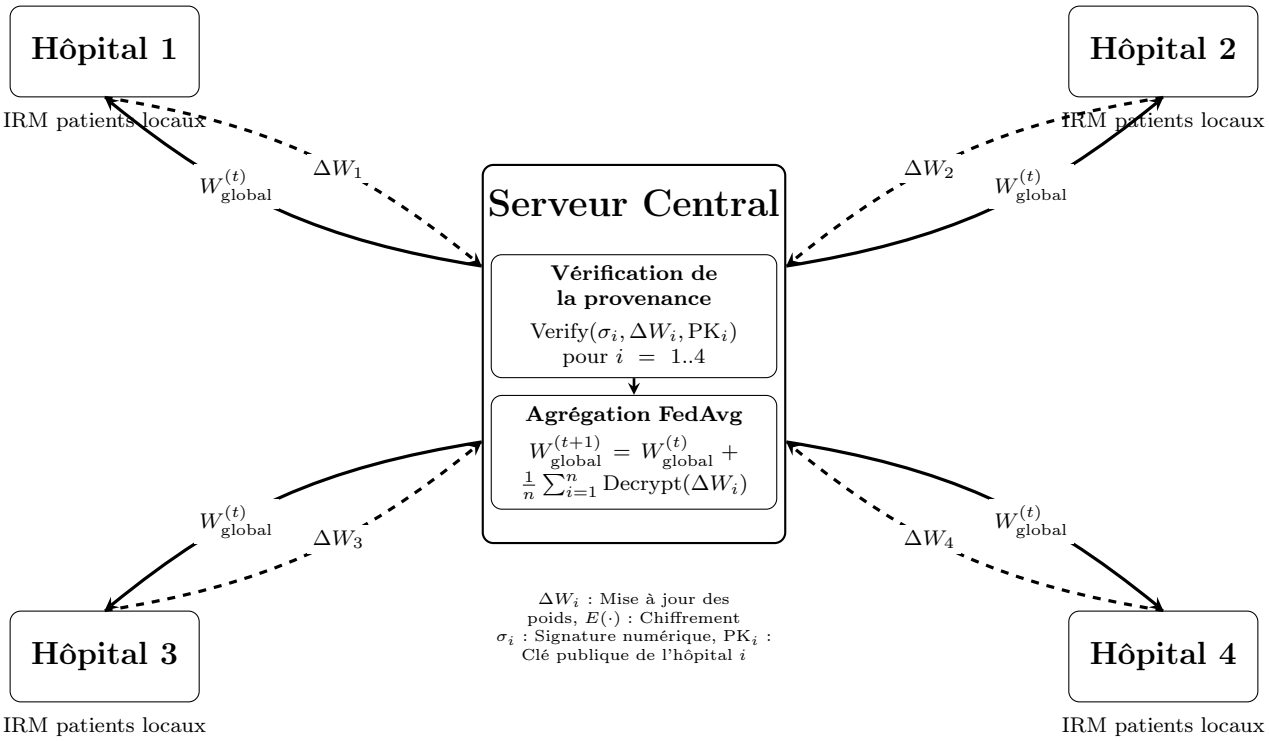

FIGURE 1 – Architecture Federated Learning sécurisé pour le diagnostic médical : les hôpitaux entraînent localement sur leurs IRM, envoient les mises à jour de poids chiffrées et signées, le serveur vérifie l'authenticité puis agrège via FedAvg

Déroulement du processus

Étape 1 : Le serveur central initialise un modèle de réseau de neurones et l'envoie aux quatre hôpitaux.

Étape 2 : Chaque hôpital entraîne le modèle localement sur ses propres images IRM pendant plusieurs époques. Les données médicales restent confinées dans l'infrastructure de chaque établissement.

Étape 3 : À la fin de l'entraînement local, chaque hôpital calcule les mises à jour de poids ΔW_i (différence entre le modèle initial et le modèle entraîné localement).

Étape 4 : Les hôpitaux transmettent uniquement ces mises à jour au serveur central, après les avoir chiffrées et signées. Aucune image IRM ni information patient n'est échangée.

Étape 5 : Le serveur vérifie les signatures pour s'assurer de l'authenticité et de l'intégrité des contributions, puis agrège les quatre contributions selon l'algorithme FedAvg :

$$W^{(t+1)}_{\text{global}} = W^{(t)}_{\text{global}} + \frac{1}{4} \sum_{i \in \{A,B,C,D\}} \Delta W_i$$

Étape 6 : Le nouveau modèle global est redistribué aux hôpitaux, et le cycle recommence.

2 Sécurité & attaque en FL

2.1 Protection cryptographique en FL

La cryptologie joue un rôle important dans la protection des informations échangées en FL, qu'il s'agisse des poids ou des gradients. Dans cette étude, nous nous concentrons sur les poids, même si les mêmes risques s'appliqueraient également aux gradients si ceux-ci étaient transmis au serveur. Lors de la transmission, le chiffrement empêche un attaquant ou même le serveur d'accéder aux mises à jour individuelles. Sans ces protections, un serveur malveillant pourrait reconstruire des images via inversion de modèle [6], analyser la distribution des classes, estimer la taille du dataset, identifier des cas rares ou suivre l'évolution temporelle des données d'un client. L'agrégation sécurisée permet justement de contrer ces menaces, elle masque chaque mise à jour individuelle et ne révèle au serveur que la moyenne globale, ce qui rend impossible toute analyse ciblée d'un client spécifique. De plus, il est possible de signer cryptographiquement les modèles ou les mises à jour de poids, assurant ainsi l'authenticité et l'intégrité des contributions.

Cependant, le chiffrement et la signature ont aussi leurs limites. Ils ne protègent pas contre les attaques qui surviennent avant le chiffrement (par exemple un client compromis), n'empêchent pas l'envoi de contributions malveillantes, et ne couvrent pas non plus les attaques portant sur le modèle final. Ce dernier peut encore être analysé, mais uniquement sur des données moyennées, comme dans FedAvg [9].

2.2 Attaque en FL

2.2.1 Attaque par inversion de modèle

Une **attaque par inversion de modèle** [6] (Model Inversion Attack) cherche à reconstruire des informations sensibles à partir d'un modèle entraîné, en exploitant uniquement ses paramètres (dans notre cas les poids). Nous présentons ci-dessous le principe et les étapes courantes de cette attaque.

Principe Partant d'un modèle f_θ entraîné sur des données privées, l'attaquant tente de générer une entrée x qui maximise la probabilité d'une classe cible y :

$$x^* = \arg\max_x f_\theta(x)_y,$$

où $f_\theta(x)_y$ désigne la probabilité prédite pour la classe y. En pratique, on résout une optimisation en minimisant la perte :

$$\mathcal{L}(x) = -\log f_\theta(x)_y + \lambda\, R(x),$$

avec $R(x)$ (par exemple une régularisation L_2) pour contraindre x à rester naturel, et λ le coefficient de régularisation.

Étapes de l'attaque

1. **Initialisation.** L'attaquant initialise une entrée artificielle x_0, souvent générée aléatoirement en l'absence des données d'entraînement.

2. **Choix de la fonction de perte.** On fixe la cible y et la fonction $\mathcal{L}(x)$, qui combine l'objectif de forcer le modèle à prédire la classe souhaitée et une contrainte de régularisation sur x.
3. **Optimisation.** La mise à jour de x se fait par itérations (par exemple via descente de gradient) afin de réduire la perte jusqu'à convergence.
4. **Reconstruction.** la solution optimale obtenue x^* représente une approximation des données privées liées à la classe ciblée. Elle ne correspond pas nécessairement à un échantillon réel, mais révèle néanmoins des caractéristiques sensibles apprises par le modèle.

3 Signature RSA

3.1 Fonctionnement en FL

En FL, pour garantir l'authenticité des mises à jour de modèles envoyées par les clients, on peut utiliser des méthodes de **signatures** [4] comme **RSA** [10].

Principe général Chaque client signe ses mises à jour ΔW avec sa clé privée (d, n) pour produire une signature S :

$$S = H(\Delta W)^d \mod n$$

où $H(\cdot)$ est une fonction de hachage cryptographique qui transforme les poids en une empreinte de taille fixe avant signature.
Le serveur, possédant la clé publique (e, n) du client, peut vérifier que la signature correspond bien aux mises à jour reçues en recalculant :

$$S^e \mod n \stackrel{?}{=} H(\Delta W)$$

Si la vérification est correcte, le serveur est certain que les mises à jour proviennent du client attendu et n'ont pas été altérées pendant le transfert.

Moment de la signature dans le processus FL La signature RSA intervient **après l'entraînement local** et **avant l'envoi au serveur**, selon le flux suivant :

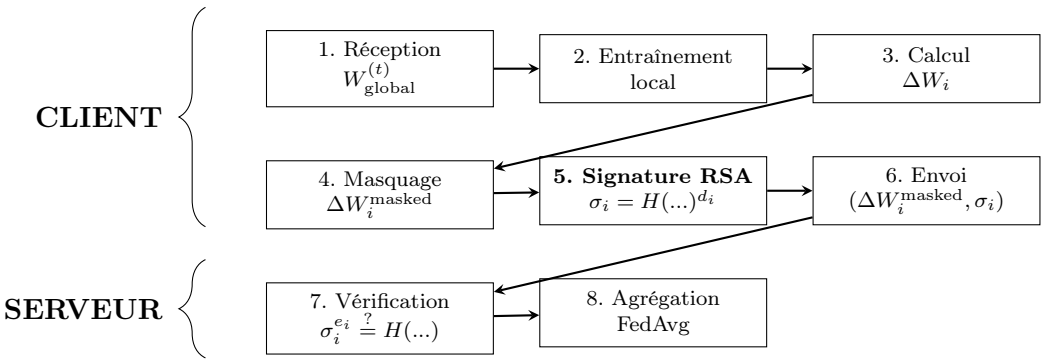

FIGURE 2 – Flux du processus FL avec signature RSA : la signature (étape 5) intervient juste avant la transmission

4 Secure Aggregation

4.1 Base du fonctionnement

L'idée clé est la suivante :

> Chaque paire de clients se met d'accord sur un masque secret. Chaque client ajoute certains masques et en soustrait d'autres, de sorte que, une fois les modèles envoyés au serveur et agrégés, tous les masques s'annulent.

Ainsi :

- Le serveur ne voit jamais les poids individuels.
- À la fin, seule la somme ou la moyenne réelle apparaît, car tous les masques se compensent.
- Le serveur obtient ce dont il a besoin c'est à dire, l'agrégation du modèle, sans aucune information privée.

4.2 Exemple illustratif

Supposons trois clients A, B et C. Chaque paire génère un masque secret partagé :

$$(A, B), \quad (A, C), \quad (B, C)$$

Chaque client possède un modèle à deux poids (ou matrices 2×2). Par exemple pour le client A :

$$W_A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Similairement :

$$W_B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, \qquad W_C = \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix}$$

Masquage pairwise

Exemple de masque pour la paire (A, B) :

$$M_{AB} = \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \end{bmatrix}$$

Client A **ajoute** ce masque, client B **soustrait** le même masque.
Ainsi :

$$W'_A = W_A + M_{AB} = \begin{bmatrix} 1.5 & 1.5 \\ 2.5 & 4.5 \end{bmatrix}$$

$$W'_B = W_B - M_{AB} = \begin{bmatrix} 4.5 & 6.5 \\ 7.5 & 7.5 \end{bmatrix}$$

Chaque client fait de même avec les masques (A, C) et (B, C).

Envoi

Les clients envoient W'_A, W'_B, W'_C au serveur. Les masques sont invisibles pour lui.

Agrégation

Le serveur calcule la moyenne élément-par-élément :

$$\bar{W} = \frac{W'_A + W'_B + W'_C}{3}$$

Tous les masques se compensent exactement :

$$(+M_{AB} + M_{AC} - M_{AB} - M_{BC} + M_{BC} - M_{AC}) = 0$$

4.3 Protocole Diffie-Hellman

Comment deux clients peuvent-ils générer et utiliser le même masque, alors qu'ils ne communiquent jamais directement entre eux, que toutes les communications passent par le serveur, et qu'ils ne doivent surtout pas révéler ce masque au serveur ? C'est là qu'intervient le protocole de Diffie–Hellman [5]. Pour que les masques partagés existent, chaque paire de clients doit d'abord se mettre d'accord sur un secret commun, sans jamais le transmettre explicitement.

Idée générale

Diffie–Hellman permet à deux clients (par exemple A et B) de produire ensemble un nombre secret :

$$K_{AB}$$

Ce secret sera le même pour les deux, mais restera complètement inconnu du serveur, même si celui-ci observe toutes les communications. Ce secret partagé servira ensuite à fabriquer un **masque pairwise**.

Étapes simplifiées

Voici comment A et B fabriquent leur secret commun :

1. Chaque client choisit un nombre secret :

   $$a \quad \text{pour A}, \qquad b \quad \text{pour B}$$

2. Ils échangent publiquement des valeurs dérivées de ces secrets :

   $$g^a \quad \leftrightarrow \quad g^b$$

   Le serveur peut voir ces valeurs publiques, mais elles ne révèlent pas les secrets.

3. Grâce à ces valeurs, chacun peut calculer le **même** secret final :

   $$K_{AB} = (g^b)^a = (g^a)^b$$

Ainsi, A et B possèdent désormais exactement le même nombre, sans jamais l'avoir envoyé directement.

4.4 Transformation du secret Diffie-Hellman en masque exploitable

Le secret K_{AB} obtenu via Diffie-Hellman est un grand nombre entier. Pour l'utiliser comme masque en FL, nous devons le transformer en une **matrice de même dimension que les poids du modèle** (qui peut contenir des millions de paramètres). Cette transformation se fait en deux étapes :

4.4.1 Étape 1 : Hachage cryptographique pour créer une graine

Le secret K_{AB} est passé dans une **fonction de hachage cryptographique** SHA-256 [1] :

$$S_{AB} = \text{SHA-256}(K_{AB})$$

Intérêt pour la génération de masques :

- SHA-256 produit toujours une sortie de 256 bits, créant une graine de taille fixe quel que soit le secret initial.
- La même valeur K_{AB} produit toujours la même graine S_{AB}, assurant que les deux clients généreront le même masque.
- Si le masque était partiellement observable, le hachage empêcherait de remonter au secret Diffie-Hellman.

4.4.2 Étape 2 : Expansion par générateur pseudo-aléatoire (PRG)

La graine S_{AB} est utilisée par un PRG pour produire le masque complet :

$$M_{AB} = \text{PRG}(S_{AB})$$

À partir de 256 bits de graine, le PRG génère autant de valeurs que nécessaire pour couvrir tous les poids du modèle (millions, voire milliards de paramètres). Les deux clients, utilisant la même graine S_{AB}, génèrent exactement la même séquence, donc le même masque M_{AB}.

4.4.3 Application des masques en FL

Une fois le masque M_{AB} généré à partir du secret Diffie-Hellman, les clients l'appliquent selon cette convention :

- **Client** A : $W'_A = W_A + M_{AB}$
- **Client** B : $W'_B = W_B - M_{AB}$

Lors de l'agrégation par le serveur :

$$W'_A + W'_B = (W_A + M_{AB}) + (W_B - M_{AB}) = W_A + W_B$$

Le masque s'annule, permettant l'agrégation correcte sans révéler les poids individuels.

4.5 Résumé

$$\boxed{\text{Diffie-Hellman}} \rightarrow \boxed{\text{Secret } K_{AB}} \xrightarrow{\text{SHA-256}} \boxed{\text{Graine } S_{AB}} \xrightarrow{\text{PRG}} \boxed{\text{Masque } M_{AB}}$$

1. **Diffie-Hellman** établit le secret partagé sans canal sécurisé direct entre clients.
2. **SHA-256** convertit ce secret en une graine cryptographiquement robuste et standardisée.
3. **PRG** étend cette graine pour créer un masque de la taille exacte du modèle.
4. **Résultat** : Chaque paire de clients dispose d'un masque identique, imprévisible pour le serveur, permettant l'annulation lors de l'agrégation.

5 Implémentation Python

5.1 Jeu de données expérimental

Pour nos expérimentations, nous utilisons le jeu de données labellisé MNIST [8]. Il contient 70 000 images de chiffres manuscrits (de 0 à 9), dont 60 000 pour l'entraînement et 10 000 pour le test. Chaque image est une matrice 28×28 pixels en niveaux de gris, centrée et normalisée afin de faciliter la classification. Chaque image est associée à une étiquette indiquant le chiffre qu'elle représente, ce qui permet un apprentissage supervisé.

5.2 Attaque d'Inversion de Modèle

5.2.1 Modèle Cible

Nous avons illustré un client se faisant attaquer par inversion de modèle à l'aide d'un réseau de neurones convolutif (CNN) sous Python avec TensorFlow, volontairement rendu vulnérable par surapprentissage (overfitting).

— **Architecture** : Conv2D (32) → MaxPool → Conv2D (64) → MaxPool → Conv2D (128) → Dense (128) → Dense (10)

— **Dataset d'entraînement** : 100 échantillons MNIST

— **Nombre d'epochs** : 10

— **Régularisation** : Aucune (pas de dropout, pas de weight decay)

Cette configuration a été choisie afin de maximiser la vulnérabilité du modèle à des fins pédagogiques. L'utilisation d'un petit jeu de données, d'un entraînement prolongé et l'absence de régularisation poussent le modèle à mémoriser les données d'entraînement plutôt qu'à généraliser. Les poids du modèle encodent ainsi directement les motifs des images d'entraînement, ce qui facilite leur reconstruction lors d'une attaque par inversion.

5.2.2 Paramètres de l'Attaque

L'attaque par inversion de modèle utilise une optimisation par descente de gradient pour reconstruire des images. L'attaquant initialise une entrée artificielle x_0 suivant une distribution normale (bruit gaussien), puis suit les étapes décrites en section 2.2.1 avec les hyperparamètres suivants : learning rate de 0.1, 3000 itérations et un coefficient de régularisation $\lambda = 0.01$..

5.2.3 Résultats de l'Attaque

L'attaque a été réalisée pour les 10 classes de chiffres (0-9) avec 10 tentatives de reconstruction par classe :

Métrique	Valeur
MAE minimum moyen	**0.0580**
MAE moyen global	0.1309
Confiance moyenne du modèle	0.9561 (95.61%)
Nombre total de reconstructions	100 (10 classes × 10 tentatives)

TABLE 1 – Résultats de l'attaque par inversion

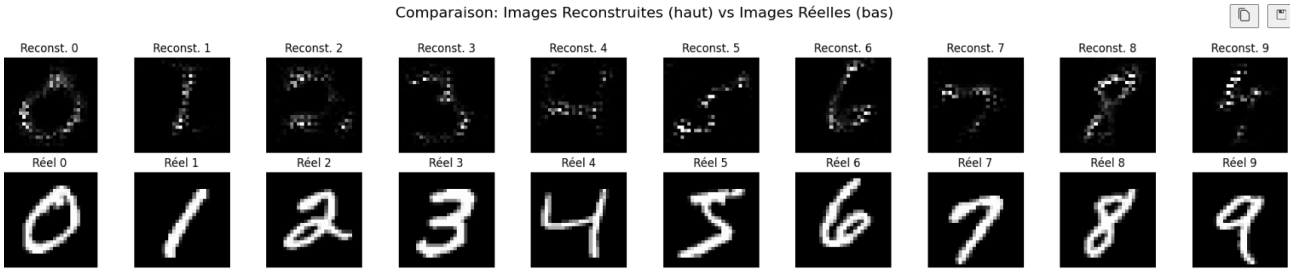

FIGURE 3 – Comparaison : Images Reconstruites (haut) vs Images Réelles (bas)

Comme l'illustre la figure 3, les reconstructions sont visuellement très proches des originaux avec une MAE de seulement 0.058 (Table 1). Ces résultats démontrent l'importance pour les clients de sécuriser leurs données et de protéger l'accès aux poids de leurs modèles.

5.3 FL avec Secure Aggregation et Signature RSA

5.3.1 Configuration Expérimentale

Pour cette expérimentation, une configuration de 20 clients a été mise en place. Afin de mettre en évidence l'intérêt du Federated Learning, trois approches ont été comparées :

1. **Apprentissage centralisé** : Entraînement avec 100% des données sur un serveur central
2. **Client unique** : Entraînement avec seulement 5% des données
3. **Federated Learning** : Entraînement distribué avec 20 clients utilisant Secure Aggregation et signatures RSA

5.3.2 Modèle et Sécurité

L'architecture utilisée est un réseau de neurones convolutif (CNN) : Conv2D (32) → MaxPool → Conv2D (64) → MaxPool → Flatten → Dropout (0.5) → Dense (128) → Dense (10).
L'implémentation suit l'algorithme décrit en annexe (Algorithme 1), qui combine :

— **Secure Aggregation** (Section 4) : Masquage des mises à jour locales pour préserver la confidentialité

— **Signatures RSA** (Section 3) : Authentification des clients avec des clés de 2048 bits pour garantir l'intégrité des communications

5.4 Résultats

TABLE 2 – Comparaison des performances selon différentes approches

Approche	Accuracy	F1-Score
1. Centralisé (100% données)	0.9916	0.9916
2. Un seul client (5.00% données)	0.9576	0.9577
3. FL + Secure Aggregation + RSA (20 clients)	0.9786	0.9786

Les résultats (Table 2) montrent que le Federated Learning avec 20 clients permet d'atteindre des performances proches de l'apprentissage centralisé, tout en étant significativement supérieur à un client isolé avec seulement 5% des données. Cela montre l'intérêt de l'apprentissage fédéré, c'est à dire améliorer le modèle grâce au partage des connaissances entre plusieurs clients, sans exposer leurs données privées.

6 Conclusion

Ce projet nous a permis de plonger au cœur des questions de sécurité liées au Federated Learning, en mêlant théorie et mise en pratique. On y montre que, même si le FL préserve déjà la confidentialité en évitant de centraliser les données, il reste quand même exposé à des attaques si l'on ne met pas en place des protections cryptographiques appropriées.

Nos expériences ont mis en lumière deux points clés. D'un côté, l'attaque par inversion de modèle que nous avons implémentée illustre le risque, quand un modèle circule sans protection, on peut reconstruire des images d'entraînement. De l'autre, la mise en place d'un système FL, avec une agrégation sécurisée via Diffie–Hellman et une authentification par signatures RSA, montre qu'on peut sécuriser les échanges de modèles tout en conservant des performances élevées.

Avec 20 clients, notre système FL sécurisé atteint une accuracy de **97,86 %**. C'est bien mieux qu'un client isolé disposant de peu de données, et c'est comparable à une approche centralisée. Le Federated Learning permet donc à plusieurs clients de s'améliorer collectivement sans sacrifier la confidentialité des données locales.

Au-delà des mécanismes cryptographiques, ce travail souligne qu'il faut une vision globale de la sécurité en FL. Le chiffrement et les signatures sont indispensables, mais ne suffisent pas à eux seuls. Pour la suite, plusieurs pistes sont intéressantes, comme détecter et neutraliser des clients malveillants qui chercheraient à empoisonner le modèle [11], intégrer du bruit pour garantir la confidentialité, [7], ou encore optimiser les communications pour des scénarios à grande échelle (mille à milliers de participants) [3].

A Annexe

Métriques d'Évaluation

Mean Absolute Error (MAE)

La MAE mesure l'erreur absolue moyenne entre les valeurs prédites et les valeurs réelles :

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

où n est le nombre d'échantillons, y_i la valeur réelle et $\hat{y}_i$ la valeur prédite.

Accuracy

L'accuracy représente la proportion de prédictions correctes parmi l'ensemble des prédictions :

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

où :

- TP : vrais positifs
- TN : vrais négatifs
- FP : faux positifs
- FN : faux négatifs

F1-Score

Le F1-score est la moyenne harmonique de la précision et du rappel :

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

où :

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Algorithm 1 Federated Learning avec Secure Aggregation et RSA

Require:
N clients avec ensembles de données $\mathcal{D}_1, \ldots, \mathcal{D}_N$
Nombre de rounds T, époques locales E, taux d'apprentissage η
Modèle global initial $\mathbf{w}^{(0)}$
Secret partagé s pour Diffie-Hellman
Ensure: Modèle global entraîné $\mathbf{w}^{(T)}$

Initialisation - Génération des clés RSA :
for $i = 1$ à N **do**
  $(sk_i, pk_i) \leftarrow$ RSA-KeyGen(2048)
  Client i garde sk_i ; Serveur reçoit pk_i
end for

Entraînement Fédéré :
for round $t = 1$ à T **do**

  Phase 1 - Distribution :
  Serveur envoie $\mathbf{w}^{(t-1)}$ à tous les clients

  Phase 2 - Entraînement Local (en parallèle) :
  for chaque client $i = 1, \ldots, N$ **do**
    $\mathbf{w}_i^{(t)} \leftarrow \text{EntraînementLocal}(\mathbf{w}^{(t-1)}, \mathcal{D}_i, E)$
    $\Delta\mathbf{w}_i^{(t)} \leftarrow \mathbf{w}_i^{(t)} - \mathbf{w}^{(t-1)}$ {Calcul de la mise à jour}
  end for

  Phase 3 - Masquage et Signature (en parallèle) :
  for chaque client $i = 1, \ldots, N$ **do**
    $\widetilde{\Delta\mathbf{w}}_i^{(t)} \leftarrow \text{MaskUpdate}(\Delta\mathbf{w}_i^{(t)}, i, N, s)$
    $\sigma_i \leftarrow \text{SignUpdate}(\widetilde{\Delta\mathbf{w}}_i^{(t)}, sk_i)$
    Envoyer $(\widetilde{\Delta\mathbf{w}}_i^{(t)}, \sigma_i)$ au serveur
  end for

  Phase 4 - Authentification :
  $\mathcal{V} \leftarrow \emptyset$ {Ensemble des clients vérifiés}
  for chaque client $i = 1, \ldots, N$ **do**
    Recevoir $(\widetilde{\Delta\mathbf{w}}_i^{(t)}, \sigma_i)$
    if $\text{VerifySignature}(\widetilde{\Delta\mathbf{w}}_i^{(t)}, \sigma_i, pk_i) = \text{True}$ **then**
      $\mathcal{V} \leftarrow \mathcal{V} \cup \{i\}$
    end if
  end for

  Phase 5 - Agrégation Sécurisée :
  if $|\mathcal{V}| > 0$ **then**
    $\overline{\Delta\mathbf{w}}^{(t)} \leftarrow \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \widetilde{\Delta\mathbf{w}}_i^{(t)}$ {Masques s'annulent}
    $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} + \overline{\Delta\mathbf{w}}^{(t)}$
  else
    $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)}$
  end if
end for

return $\mathbf{w}^{(T)}$

Notations

— N : Nombre de clients
— T : Nombre de rounds d'entraînement
— E : Nombre d'époques locales par round
— η : Taux d'apprentissage
— $\mathbf{w}^{(t)}$: Poids du modèle global au round t
— $\mathbf{w}_i^{(t)}$: Poids du modèle local du client i au round t
— $\Delta\mathbf{w}_i^{(t)}$: Mise à jour du client i au round t (non masquée)
— $\widetilde{\Delta\mathbf{w}}_i^{(t)}$: Mise à jour masquée du client i
— $\overline{\Delta\mathbf{w}}^{(t)}$: Moyenne des mises à jour agrégées
— $\mathcal{D}_i$: Ensemble de données du client i
— $\mathcal{B}$: Mini-batch de données
— $L(\mathbf{w}; \mathcal{B})$: Fonction de perte sur le batch $\mathcal{B}$
— sk_i, pk_i : Clé privée et publique RSA du client i
— σ_i : Signature RSA du client i
— s : Secret partagé pour la génération des masques
— $\mathbf{m}_{ij}^{(\ell)}$: Masque pseudo-aléatoire entre clients i et j pour la couche ℓ
— h_i : Hash SHA-256 de la mise à jour masquée
— $\mathcal{V}$: Ensemble des clients dont la signature a été vérifiée

Références

[1] Secure hash standard (shs). Technical Report FIPS PUB 180-4, National Institute of Standards and Technology (NIST), 2015. Accessed : 2025-11-30.

[2] Keith Bonawitz, Vladimir Ivanov, Benjamin Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. 2017.

[3] L. Chen and Z. Wang. Sparsification under siege : Defending against poisoning attacks in communication-efficient federated learning. 2023.

[4] Zao Chen. Federated learning based on homomorphic encryption and digital signatures. *Highlights in Science, Engineering and Technology*, 39, 2023. Dans le cadre de la conférence CMLAI 2023.

[5] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6) :644–654, 1976.

[6] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333. ACM, 2015.

[7] J. Kim, S. Park, and H. Lee. Precad : Privacy-preserving and robust federated learning via crypto-aided differential privacy. 2021.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

[9] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282, 2017.

[10] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2) :120–126, 1978.

[11] X. Zhu, Y. Liu, and Q. Wang. Fldetector : Defending federated learning against model poisoning attacks. 2022.