A-SIT

Zentrum für sichere Informationstechnologie - Austria

Evaluierung des “Secure-Aggregation”-Aspekts in privatsphären-bewahrendem „Federated Machine Learning“

Evaluierung des “Secure-Aggregation”-Aspekts in privatsphären-bewahrendem „Federated Machine Learning“

Autor:
Karl W. Koch
Tel: +43 316 873 - 5517
Mail: karl.koch@iaik.tugraz.at

Datum: 31.08.2023

Kurzfassung:
In diesem Bericht betrachten wir das Konzept von föderiertem maschinellem Lernen („Federated Machine Learning“ / FL), und vor allem dessen Herausforderungen in Bezug auf die Privatsphäre der teilnehmenden Nutzer-Daten. Im Speziellen die privatsphären-bewahrende Variante von FL (= PFL) mittels sicherer Aggregation („Secure Aggregation“ / SAgg). Diese Analyse zielt darauf ab, weitere ML-Anwendungsfälle in föderierten Szenarien zu ermöglichen.

PFL - mit SAgg - ermöglicht das private Lernen eines ML-Modells in föderierten Szenarien (viele dezentralisierte/verteilte Nutzer/Geräte; wie Smartphone-Apps). Ein Server/Dienstanbieter/etc. lernt nur von einer Gruppe von Nutzern, während die Aktualisierung des ML-Modells eines einzelnen Nutzers privat bleibt. Eine der vielversprechendsten SAgg-Techniken ist die Verwendung neuartiger datenschutzfreundlicher kryptografischer Bausteine, insbesondere auf der Grundlage von „Secure Multi-Party Computation“ (MPC).

Google stellte 2017 das quasi-erste praktisch relevante PFL-SAgg-Protokoll auf Basis von MPC vor: **SecAgg**. Daraufhin folgten einige Folgearbeiten. Drei dieser Folgearbeiten sehen wir uns näher an: **SecAgg+**, **FastSecAgg** und **LightSecAgg**. Die verschiedenen Protokolle bieten unterschiedliche Trade-Offs in Bezug auf Berechnungs- und Kommunikations-Komplexität. Abschließend wird eine Conclusio der betrachtenden MPC-basierten PFL-SAgg-Protokolle gezogen und potentielle Folgearbeiten aufgezeigt.

Im Folgenden werden die unterschiedlichen Abschnitte des Berichts gelistet:

Inhaltsverzeichnis

1. Einleitung & Privatsphären-bezogene Herausforderungen beim Föderierten Maschinellen Lernen („Federated Learning")

Immer mehr Daten werden von Geräten einzelner Personen, wie Smartphones oder Smartwatches, erzeugt. Zum Beispiel Fitness- und generelle Aktivitätsdaten von entsprechenden Trackern wie FitBit[1] oder der AppleWatch[2]. Oder auch, wie Personen eine Anwendung (App) eines Smartphones nutzen. Ein Beispiel für eine solche App ist Googles **Gboard**[3][4]: eine virtuelle Tastatur, bei der die App die Eingaben des Nutzers lernt - in erster Linie nur lokal auf dem Smartphone (wie bei jeder anderen App auch). Aber mit den (lokalen) Nutzungsdaten, vor allem wenn sie mit den Daten anderer App-Nutzer zusammengeführt werden, ist es möglich, ein Modell für maschinelles Lernen (ML) zu trainieren. Solch ein ML-Modell kann z.B. die Bedienbarkeit der App verbessern. Bei **Goard** ist dies die Vorhersage des nächsten Wortes oder sogar des nächsten Satzes; womit End-Nutzer schlussendlich schneller antworten bzw. effizienter kommunizieren können. Das Hauptproblem in solchen Szenarien ist die Wahrung der Privatsphäre der Nutzer.

Beim klassischen ML stehen alle Trainingsdaten einer Instanz zur Verfügung, die das Training eines ML-Modells durchführt. Bei dieser Instanz handelt es sich in der Regel um einen zentralen Server oder ein Unternehmen, das die Trainingsdaten aus potenziell vielen verschiedenen Quellen bezieht. Wenn die Trainingsdaten jedoch (1) datenschutzrechtlich sensibel und/oder (2) umfangreich sind, wird diese zentrale Instanz zu einem interessanten Ziel für Cyberangriffe und/oder ist möglicherweise nicht in der Lage, all diese Daten zu speichern. Außerdem möchten die Teilnehmer ihre (datenschutzsensiblen) Daten möglicherweise gar nicht erst vollständig mit dieser zentralen Stelle teilen. Zum Beispiel medizinische Daten eines Nutzers, die für eine verbesserte Krebsvorhersage verwendet werden, oder getippte Wörter eines Nutzers, um nachfolgende Wörter oder sogar ganze Sätze besser vorhersagen zu können.

Föderiertes maschinelles Lernen (FL). Eine vielversprechende Lösung für das genannte Datenschutz- und Speicherkapazitäts-Problem bietet das föderierte maschinelle Lernen bzw. „Federated Machine Learning" (FL), welches von Google auf der AISTATS 2017 vorgestellt wurde (McMahan, Moore, Ramage, Hampson, & Arcas, 2017). *Bei FL trainiert jeder Nutzer lokal ein ML-Modell und sendet nur die Aktualisierung des ML-Modells an die zentrale Instanz:* Die zentrale Instanz sendet ein ML-Modell an die teilnehmenden Nutzer (= Clients), und die Clients aktualisieren das Modell lokal auf der Grundlage ihrer (lokalen) Daten. Anschließend senden die Clients nur die Aktualisierung des ML-Modells an die zentrale Instanz. Schließlich berechnet die zentrale Einheit eine gewichtete Summe aller empfangenen Aktualisierungen des ML-Modells und aktualisiert so das globale ML-Modell. Bei FL hat die zentrale Instanz also weder Zugriff auf die (lokalen) Trainingsdaten der Clients noch muss sie diese Daten speichern. Ein solcher Durchgang – *Instanz->Clients->Instanz* - wird als Epoche bezeichnet. Um die Genauigkeit eines ML-Modells zu erhöhen, kann die zentrale Instanz zahlreiche Epochen durchführen.

Bonawitz et al. (von Google) unterteilten FL weiter in geräte-übergreifendes („cross-device") FL und silo-übergreifendes („cross-silo") FL (Bonawitz, Kairouz, McMahan, & Ramage, 2022). Geräte-übergreifendes FL bezeichnet das Szenario, an dem viele Clients (z.B. 1 Million) teilnehmen, die eine relativ geringe Rechenleistung haben, und bei dem in der Regel nur eine Teilmenge von Clients Teil einer Epoche ist. Ein Beispiel für geräte-übergreifende FL ist das – oben genannte Beispiel vom - Lernen von Vorhersagen für das nächste Wort oder den nächsten Satz auf der Tastatur eines Smartphones (**Gboard**). Silo-übergreifendes FL bezeichnet das Szenario, an dem weniger Clients (z.B. 100) teilnehmen, die eine relativ große Rechenleistung haben, und an dem normalerweise alle Clients Teil einer Epoche sind. Ein Beispiel für silo-übergreifendes FL ist das Lernen von Patientendiagnosen zwischen mehreren Spitälern.

[1] fitbit.com/global/at/home
[2] apple.com/apple-watch-series-9/why-apple-watch/
[3] wikipedia.org/wiki/Gboard
[4] aclanthology.org/2023.acl-industry.60 *(Federated Learning of Gboard Language Models with Differential Privacy)*

Herausforderungen für den Datenschutz bei FL (→ Richtung PFL). Dieses Basisprotokoll von FL ist zwar ein guter Schritt zum Schutz der Privatsphäre der Clients, reicht aber nicht aus. Seit der Einführung von FL wurden einige praktische Angriffe auf Basis der ML-Modell-Aktualisierungen der Clients gezeigt (Nasr, Shokri, & Houmansadr, 2019), (Fredrikson, Jha, & Ristenpart, 2015), (Ganju, Wang, Yang, Gunter, & Borisov, 2018), (Carlini, Liu, Kos, & Song, 2019), (Shokri, Stronati, Song, & Shmatikov, 2017). So kann es – erstens - möglich sein, das Trainingsset (teilweise) eines Clients allein durch dessen Aktualisierung wiederherzustellen. Zweitens, eine weitere Angriffsfläche ist das ML-Modell selbst. Die Trainingsdaten eines Clients könnten durch das globale ML-Modell selbst abgeleitet werden, was insbesondere bei Ausreißern („Outliers") vorkommen kann. So zeigte Google im April 2022, dass die Texteingabe eines Clients mit einem ML-Modell zur Vorhersage des nächsten Wortes abgeleitet werden kann (Bonawitz, Kairouz, McMahan, & Ramage, 2022). Abbildung 1 zeigt dieses Problem der Inferenz unter Verwendung des gelernten globalen ML-Modells. D.h. es gibt grundlegend Angriffe auf die zwei Hauptphasen von FL: (1) beim Training (Aktualisieren des ML-Modells durch neue Client-Parameter), und (2) bei der Inferenz (Evaluieren des ML-Modells).

Abbildung 1. Ein Beispiel wo das gelernte globale ML-Modell, für die Vorhersage der nächsten Wörter, sensitive Informationen preisgibt. Quelle: https://xkcd.com/2169

Aus diesem Grund wurde FL mit privatsphären-bewahrenden Bausteinen erweitert (= Privatsphären-bewahrendes FL (PFL)). In der Trainings-Phase durch die sogenannte Sichere Aggregation (SAgg / „Secure Aggregation"). Und in der Inferenz-Phase durch eine Art Maskierung der Daten bzw. Hinzufügen von Rauschen.

Bei SAgg werden die neuen ML-Modell-Parameter der Clients so zusammengefasst, dass die zentrale Instanz die neuen Parameter aller Clients auf einmal erhält (= die Summe aller Parameter), aber nicht auf die Parameter einzelner Clients rückschließen kann. Sofern – natürlich – die Anzahl der teilnehmenden Clients groß genug ist. Beim Hinzufügen von Rauschen, auf z.B. die Eingabedaten oder die resultierenden ML-Modell-Parameter, werden die Daten bewusst (statistisch) verzerrt (= mit Rauschen versehen), damit keine bzw. kaum Rückschlüsse auf einzelne Clients (primär) bei der Inferenz-Phase möglich sind.

Bonawitz et al. bezeichnen diese beiden Techniken als Praxis der (1) Datenminimierung und (2) Datenanonymisierung, insbesondere der anonymen Aggregation (Bonawitz, Kairouz, McMahan, & Ramage, 2022). Darüber hinaus führen Bonawitz et al. noch einen dritten Schlüsselaspekt im Hinblick auf die Herausforderungen des Datenschutzes in FL ein: die Zustimmung und Transparenz in Bezug auf die Anwendung/des Dienstes selbst. Mit Zustimmung und Transparenz ist im Wesentlichen gemeint, dass die Clients wissen, was (mit ihren Daten) geschieht, und dass sie dem zustimmen. Dieser dritte Schlüsselaspekt kann nicht - im Gegensatz zu den ersten beiden Schlüsselaspekten – mit technischen Maßnahmen gelöst werden.

Summa summarum. Um die Privatsphäre der Clients – idealerweise endgültig - zu schützen, muss ein PFL-Protokoll (1) die ML-Modell-Aktualisierungen der Clients schützen und (2) sicherstellen, dass die Trainingsdaten der Clients nicht vom ML-Modell selbst abgeleitet werden können, was insbesondere bei Ausreißern („Outliers") vorkommen kann. Die erste Herausforderung (1) kann – bzw. wird primär - mit kryptographischen Primitiven wie „Secure Multi-Party-Computation" (MPC) oder der homomorphen Verschlüsselung gelöst werden und wird als Sichere Aggregation (SAgg / „Secure Aggregation") bezeichnet. Die zweite Herausforderung (2) kann durch Hinzufügen von (statistischem) Rauschen zu den Trainingsdaten oder der neuen ML-Modell-Parameter der Clients gelöst werden und wird üblicherweise mittels der Differentiellen Privatsphäre (DP / „Differential Privacy") umgesetzt.

Wenn beide Aspekte bei PFL berücksichtigt werden, sprechen wir von Ende-zu-ende Privatsphären-bewahrendem Föderiertem (maschinellen) Lernen (**EPFL**). Als ersten Schritt in Richtung EPFL, konzentrieren wir uns in diesem Bericht auf den ersten Aspekt: *PFL mit MPC-basierter SAgg*.

2. MPC-basierte Sichere Aggregation beim Föderierten Maschinellen Lernen

Eine der vielversprechendsten Ansätze für die Sichere Aggregation (SAgg / „Secure Aggregation") von Client-Aktualisierungen der ML-Modell-Parameter in PFL ist auf Basis von MPC – im Speziellen der Aspekt des Geheimen Teilens (SeSh / „Secret Sharing").

Quasi die erste als praktisch geltende MPC-basierte SAgg wurde auf der CCS 2017 von Google und Cornell Tech vorgestellt (Bonawitz et al.) (Bonawitz, et al., 2017). Ihr Ansatz wird üblicherweise als **SecAgg** bezeichnet und besteht aus maximal fünf Runden plus einer Einrichtungsphase:

0) Einrichtung / „Setup"
   Der Server sendet jedem Client die Parameter für dieses Protokoll, insbesondere den Sicherheitsparameter κ, die Anzahl der Clients n, den Schwellenwert t und finites Feld $\mathbb{F}$ für SeSh und den verwendeten Zahlenraum $\mathbb{Z}_R^m$. Außerdem richtet jeder Client einen sicheren und privaten authentifizierten Kanal mit dem Server ein (z.B. TLS mit gepinntem Zertifikat).

1) Berechnung und Austausch von (öffentlichen) DH-Schlüsseln
   Jeder Client generiert 2 Diffie-Hellman (DH)-Schlüsselpaare und teilt deren öffentlichen Teil.

2) Aufteilung (SeSh) und Austausch von maskenbezogenen Werten
   Jeder Client u teilt und sendet die jeweiligen Werte, welche für die (Ent-)Maskierung benötigt werden:

   a) den geheimen DH-Schlüssel s_u^{SK} für die gemeinsamen Masken → $s_{u,v}$

   b) Zufälliges Element für die einzelne Maske (nur u) → b_u

   u führt ein *Shamir-t-aus-n-Schwellenwert*-SeSH für diese beiden Werte durch. D.h. die Werte werden in n Teile aufgeteilt, wobei man mindestens t Teile benötigt um auf den Ursprungswert zurückzurechnen (Shamir, 1979).

3) Maskieren & Aktualisieren der neuen ML-Modell-Parameter
   Basierend auf einem Pseudozufallsgenerator (PRG) berechnet jeder Client die jeweiligen Maskenwerte und sendet die neuen maskierten ML-Modell-Parameter an den Server: $y_u = x_u + m_u + \sum_v m_{u,v}$. m_u basiert auf b_u. $m_{u,v}$ basiert auf $s_{u,v}$. Und jeder Wert in der Gleichung ist ein Vektor; die Länge entspricht der Anzahl der ML-Modell-Parameter.

4) Gewährleistung der Konsistenz (aktiver Korruptions-Modus / „Malicious Security")
   Da in jeder Runde (1) bestimmte Clients aufhören könnten, am Protokoll teilzunehmen, oder (2) manche Clients korrupt sind/werden und keine korrekten Daten senden: signiert jeder Client zusätzlich die Liste der teilnehmenden Clients, die eine Aktualisierung des ML-Modells gesendet haben. Diese Liste wird für einen möglichen Wiederherstellungsprozess während der Demaskierung (= Berechnung der SAgg-Summe) benötigt.

5) Sichere Rekonstruktion der aggregierten neuen ML-Modell-Parameter
   Jeder Client sendet seine jeweils empfangenen Anteile an den Server:

   a) Anteil von b_v für jeden Client, der eine ML-Modell-Aktualisierung gesendet hat

   b) Anteil von s_v^{SK} für jeden Client, der *keine* ML-Modell-Aktualisierung gesendet hat, aber in der vorangegangenen Runde maskenbezogene Werte geteilt hat (andernfalls würde sich die entsprechend addierte Maske der "aktiven" Clients in der endgültigen Gleichung nicht aufheben)

   Das heißt, Clients, die eine Aktualisierung des ML-Modells gesendet haben, werden als „aktive" Clients bezeichnet. Nun, für jeden Client in a), wird b_v rekonstruiert und die *einzelne* Maske m_v berechnet. Für jeden Client in b), wird s_u^{SK} rekonstruiert und die gemeinsamen Masken $m_{u,v}$ berechnet (für jeden „aktiven" Client v). Schließlich summiert der Server für jeden "aktiven" Client v jedes empfangene y_v auf und addiert/subtrahiert die entsprechenden Masken: $x = \Sigma_u y_u - \Sigma_u m_u + \Sigma_{u,v} m_{u,v}$

SecAgg besteht also aus 4 Runden für den passiven Korruptions-Modus, und aus 5 Runden für den aktiven Korruptions-Modus. Und in jeder Runde benötigt der

Server mindestens *t* Client-Antworten; andernfalls können die erforderlichen geheimen Anteile – des SeSh - zur Berechnung der entsprechenden Maskenwerte nicht rekonstruiert werden.

2.1. PFL-Protokoll SecAgg+ („Secure Single-Server Aggregation with (Poly)Logarithmic Overhead") (Bell, Bonawitz, Gascón, Lepoint, & Raykova, 2020)

SecAgg+ (Bell, Bonawitz, Gascón, Lepoint, & Raykova, 2020) baut auf **SecAgg** auf; daher auch der Name/die Abkürzung. Während **SecAgg** einen vollständigen Graphen für die Maskierungen verwendet, verwendet **SecAgg+** einen „dünnbesetzten Zufallsgraphen" („Sparse Random Graph") vom Grad *k*, wobei $k = O(\log(n))$. In dem vollständigen Graphen fügt jeder Client die gemeinsame Maske für jeden anderen Client hinzu (wie in Abschnitt 2 gezeigt). Im Graphen vom Grad *k* fügt jeder Client die gemeinsame Maske "nur" für *k* Nachbar-Clients hinzu. Diese Verringerung der Berechnungs- und Kommunikationskomplexität macht **SecAgg+** (wesentlich) effizienter. So sagen Bell et al. (Bell, Bonawitz, Gascón, Lepoint, & Raykova, 2020) in ihrem Protokoll für passive Korruption, dass **SecAgg+** für ~1 Milliarde Clients die gleiche Effizienz pro Client bietet wie **SecAgg** für ~1.000 Clients; das ist eine Verbesserung um ~6 Größenordnungen. Grundlegend bietet **SecAgg+** drei Beiträge im Bereich von PFL:

1) MPC-basierte SAgg in PFL im
   a. Passiven Korruptions-Modell („semi-ehrlich" / „semi honest"),
   b. Halb-aktiven Korruptions-Modell („semi-malicious"), und
2) (das erste) praktisch instanziierte sichere Mischen („Shuffling") für das „Shuffling-Modell" von „Differential Privacy".

SecAgg+'s Variante mit dem passiven Korruptions-Modell funktioniert im Wesentlichen genauso wie die Variante mit dem passiven Korruptions-Model von **SecAgg**. Allerdings wird zur Berechnung und Kommunikation ein ungerichteter Grad-*k*-Graph verwendet (geheimes Teilen (= SeSh) der jeweiligen Maskenwerte und Berechnung der gemeinsamen Maskenwerte "nur" für *k* Nachbar-Clients). Dieser Graph wird vom Server erstellt und dann an die Clients übermittelt. Insgesamt besteht die Variante mit dem passiven Korruptions-Modell aus 3,5 Kommunikationsrunden.

Bei **SecAgg+**'s Variante mit dem halb-aktiven Korruptions-Modell wird dem Server vertraut, dass er die Einrichtungsphase korrekt durchführt. Außerdem wird zu Beginn des Protokolls die Anzahl der korrumpierten Clients als feststehend betrachtet; daher bietet **SecAgg+** keine Sicherheit gegen adaptive Korruption. Weiters besteht **SecAgg+**'s Variante mit dem halb-aktiven Korruptions-Modell im Wesentlichen aus denselben Schritten wie die Variante mit dem passiven Korruptions-Modell, enthält jedoch Änderungen zum Schutz vor einem korrupten Server und Clients:

0) Einrichtungsphase / „Setup"
   Obwohl der Server in der Einrichtungsphase als semi-ehrlich gilt (passiver Korruptions-Modus) - die Clients erzeugen die beiden DH-Schlüsselpaare und senden die öffentlichen Teile an den Server - bindet der Server die empfangenen öffentlichen DH-Schlüssel in einen Merkle-Baum ein.

1) Graph-Erzeugungsphase
   Jeder Client *i* wählt zufällig *k* Nachbarn (ausgehende Nachbarn von *i*). Dann erhält *i* die Liste der Clients, die *i* als Nachbarn gewählt haben (eingehende Nachbarn von *i*), zusammen mit den jeweiligen öffentlichen DH-Schlüsseln. Um die Korrektheit der Schlüssel zu überprüfen, stellt der Server die erforderlichen Zwischen-Hashes des Merkle-Baums bereit. Außerdem akzeptiert *i* "nur" maximal $3*k$ eingehende Nachbarn.

2) Phase der Geheimen Anteile (= SeSh)
   Jeder Client *i* gibt die beiden geheimen Anteile – vom SeSh - an seine ausgehenden Nachbarn weiter (*(1)* den Samen („Seed") der Einzelmaske und *(2)* den geheimen DH-Schlüssel, der zur Berechnung des Seeds der gemeinsamen Maske verwendet wird)

3) Masken-Eingabephase
   Jeder Client *i* berechnet die maskierte Aktualisierung des ML-Modells. Für die gemeinsamen Masken

unterzeichnet *i* zusätzlich eine Nachricht, dass er die gemeinsame Maske *(i, j)* einbezogen hat (= die "beigefügte" Signatur).

4) Summierungsphase
Der Server stellt jedem Client *i* zwei Listen seiner eingehenden Nachbarn zur Verfügung: *(a)* Clients, die eine maskierte ML-Modell-Aktualisierung beigetragen haben (= noch aktiv sind) und *(b)* Clients, die keine Aktualisierung beigetragen haben (= ausgeschieden sind). Zusätzlich validiert jeder Client seine empfangenen "beigefügten" Signaturen; sowie, dass die Schnittmenge von *(a)* und *(b)* eine leere Menge ist und dass alle diese Clients tatsächliche Nachbarn von *i* sind. Dann signiert *i* eine Nachricht für jeden Client in *(a)* – eingehender aktiver Client *j* - dass "*i* weiß, dass *j* noch aktiv ist" (= die "ACK"-Signatur). Nun erwartet *j* "genug" solcher gültigen "ACK"-Signaturen von seinen ausgehenden Nachbarn *i*. Erst dann stellen die Clients dem Server die entsprechenden Anteile zur Verfügung (Anteil der einzelnen Maske für die Clients in *(a)* und Anteil des DH-Geheimschlüssels der gemeinsamen Maske für die Clients in *(b)*).

2.2. Verwandte PFL-Protokolle

Auf der Grundlage des SecAgg-Ansatzes folgten weitere (MPC-basierte) Protokolle. Zwei dieser Ansätze sind "**FastSecAgg**: Scalable Secure Aggregation for Privacy- Preserving Federated Learning" (Kadhe, Rajaraman, Koyluoglu, & Ramchandran, 2020) und "**LightSecAgg**: a Lightweight and Versatile Design for Secure Aggregation in Federated Learning" (So, et al., 2022).

FastSecAgg führt ein neues Verfahren für Geheimes Teilen (= SeSh) ein: **FastShare**. Während der De-facto-Standard für SeSh (bis jetzt), Shamir's *t-aus-n-Schwellenwert*-SeSH, eine quadratische Komplexität pro Client verursacht, reduziert **FastShare** diese auf *n*log(n)*. Basierend auf **FastShare** führt **FastSecAgg** SAgg in 3 Runden (2,5 Runden Kommunikation) durch. Anstatt eine Maske hinzuzufügen, erstellt jeder Client *n* Anteile seiner ML-Modell-Aktualisierung und sendet jedem anderen Client einen Anteil. Dann summieren die Clients lokal ihre (empfangenen) Anteile auf und senden diese Summe an den Server. Schließlich addiert der Server die Summen aller Clients, um die *globale* aggregierte ML-Modell-Aktualisierung zu erhalten. Das Protokoll ist zwar (nur) für passive Korruption sicher, erlaubt dies aber adaptiv während des Protokolllaufs; daher werden die korrupten Clients nicht als feststehend betrachtet, bevor das Protokoll beginnt (wie z. B. bei **SecAgg+**).

LightSecAgg baut auf **SecAgg** und **SecAgg+** auf. Da die Rekonstruktion der Masken einer der Engpässe des **SecAgg/+**-Protokolls ist, zielt **LightSecAgg** darauf ab, dies zu vereinfachen. Statt (1) eine einzelne Maske und (2) eine paarweise gemeinsame Maske mit jedem anderen Client hinzuzufügen, maskiert ein Client in **LightSecAgg** die Eingabe nur mit einem Zufallsvektor (= eine "einzelne Maske"). Dann wird diese "Einzelmaske" partitioniert und mit einer *T*-privaten Maximum-Distanz-Code-Matrix (MDS / „Maximum-Distance-Separable") kodiert; und jeder andere Client erhält eine andere Kodierung dieser Matrix. In der Summierungsphase summiert jeder Client seine (empfangenen) kodierten Masken-Anteile der noch aktiven Clients und sendet diese Client-Aggregation an den Server. Der Server kann nun alle Client-Aggregationen aufsummieren, um die *globale* aggregierte Maske zu erhalten. Das heißt der Server (1) rekonstruiert diese aggregierte Maske in einem einzigen Schritt und (2) rekonstruiert dann die *globale* aggregierte ML-Modell-Aktualisierung mit dieser aggregierten Maske. Insgesamt besteht **LightSecAgg** aus 3 + 1 Runden (2,5 + 1 Kommunikationsrunden). In ihrem Papier gehen So-He-Yang et al. davon aus, dass die Einrichtungsphase (= Schlüsselaustausch usw.) gegeben ist, daher die +1. **LightSecAgg** berücksichtigt jedoch nur passive Korruption.

Tabelle 1 gibt einen Überblick über die asymptotischen Laufzeiten von **SecAgg**, **SecAgg+**, **FastSecAgg**, **LightSecAgg** und der einfachen Aggregation (**„Plain"**).

Approach	Server		Client	
	Computation	*Communication*	*Computation*	*Communication*
SecAgg	$O(n^2 + n^2 l)$	$O(n^2 + nl)$	$O(n^2 + nl)$	$O(n + l)$
SecAgg+	$O(n \log^2(n) + nl \log(n))$	$O(n \log(n) + nl)$	$O(\log^2(n) + l \log(n))$	$O(\log(n) + l)$
FastSecAgg	$O(\log(n) l)$	$O(nl + n^2)$	$O(\log(n) l)$	$O(l + n)$
LightSecAgg	$O(l \frac{U \log(U)}{U-T})$	$O(ln + l \frac{U}{U-T})$	$O(l \frac{n \log(n)}{U-T})$	$O(l + \frac{l}{U-T})$
Plain Aggregation	$O(nl)$	$O(n)$	$O(l)$	$O(l)$

*Tabelle 1. Asymptotischer Überblick über den maximalen Berechnungs- bzw. Kommunikationsaufwand zwischen den PFL-Protokollen **SecAgg, SecAgg+, FastSecAgg, LightSecAgg** und der Aggregation in **„Plain“** (wo die resultierenden ML-Modell-Parameter im Klartext an den Aggregator gesendet werden). Für **LightSecAgg** bezeichnet **U** die minimale Anzahl von Nutzern, die zur Rekonstruktion der Masken der Teilnehmer erforderlich sind, und **T** die maximale Anzahl von korrupten Teilnehmern.*

3. Fazit & Weiterführende Arbeiten

Seit der Einführung des (praktischen) privatsphären-bewahrendem föderierten maschinellen Lernens (PFL) im Jahr 2017 (**SecAgg**) sind einige Folgearbeiten erschienen. In diesem Bericht wurden drei Folgearbeiten näher beleuchtet: **SecAgg+**, **FastSecAgg** und **LightSecAgg**. Im Wesentlichen bieten alle genannten Arbeiten bestimmte Kompromisse. So bietet **SecAgg+** die beste Kommunikationskomplexität, hat aber eine relativ hohe Rechenkomplexität. **LightSecAgg** und **FastSecAgg** hingegen bieten beide eine vielversprechende Berechnungskomplexität, haben aber eine relativ höhere Kommunikationskomplexität. **LightSecAgg** und **FastSecAgg** garantieren jedoch nur Sicherheit für passive Korruptionen, während **SecAgg** und **SecAgg+** Varianten für (halb-)aktive Korruption bieten. Darüber hinaus führt **FastSecAgg** ein neues Schema für Geheimes Teilen (= SeSh) ein - **FastShare** - das in zukünftigen Arbeiten für SeSh-Szenarien von unabhängigem Interesse sein könnte (wie z.B. im SPDZ-Protokoll für generelles MPC (Smart, Pastro, Damgård, & Zakarias, 2012)).

Darüber hinaus wären folgende Schritte interessant:

- Die unterschiedlichen Trade-Offs näher beleuchten (z.B. Effizienz (Berechnung/Speicher) / ML-Modell-Genauigkeit / Robustheit / Vertrauenswürdigkeit).
- Weitere Entwicklungen von einerseits MPC-basierter Sicherer Aggregation (= SAgg) inspizieren, und andererseits auch im Bereich der Homomorphen Verschlüsselung *(als Alternative bzw. Ergänzung zu MPC-SAgg)*.
- Etwaige Ergänzungen mit „Daten-Rausch"-Techniken *(„noise")* inspizieren; z.B. mittels differenzierter Privatsphäre *(„Differential Privacy")*, wo beispielsweise bei den Eingabedaten oder den ML-Modell-Parametern Rauschen hinzugefügt wird, um die Privatsphäre bei der Inferenz zu bewahren.
- Gemeinsamkeiten und Unterschiede zwischen den generellen PFL-Konzepten „X-Gerät" *(viele berechnungs-schwache Teilnehmer (z.B. Smartphones))* und „X-Silo" *(wenige berechnungs-starke Teilnehmer (z.B. große Organisationen wie Spitäler))* PFL herausarbeiten.

Literaturverzeichnis

McMahan, Moore, Ramage, Hampson, & Arcas. (2017). Communication-Efficient Learning of Deep Net- works from Decentralized Data. *AISTATS.* Fort Lauderdale: https://proceedings.mlr.press/v54/mcmahan17a.html.

Bonawitz, Kairouz, McMahan, & Ramage. (2022). Federated learning and privacy. *Commun. ACM.* https://dl.acm.org/doi/10.1145/3500240.

Nasr, Shokri, & Houmansadr. (2019). Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. *S&P.* https://ieeexplore.ieee.org/document/8835245.

Ganju, Wang, Yang, Gunter, & Borisov. (2018). Property Inference Attacks on Fully Connected Neural Networks using Permutation Invariant Representations. *CCS.* Toronto: https://dl.acm.org/doi/10.1145/3243734.3243834 .

Fredrikson, Jha, & Ristenpart. (2015). Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. *CCS.* Denver: https://dl.acm.org/doi/10.1145/2810103.2813677.

Carlini, Liu, E., Kos, & Song. (2019). The secret sharer: Evaluating and testing unintended memorization in neural networks. *USENIX Security.* Santa Clara: https://www.usenix.org/conference/usenixsecurity19/presentation/carlini.

Shokri, Stronati, Song, & Shmatikov. (2017). Membership Inference Attacks Against Machine Learning Models. *S&P.* San Jose: https://ieeexplore.ieee.org/document/7958568.

Bonawitz, Ivanov, Kreuter, Marcedone, McMahan, Patel, . . . Seth. (2017). Practical Secure Aggregation for Privacy-Preserving Machine Learning. *CCS.* New York: https://dl.acm.org/doi/10.1145/3133956.3133982.

Shamir. (1979). How to Share a Secret. *Commun. ACM.* https://dl.acm.org/doi/10.1145/359168.359176.

Bell, Bonawitz, Gascón, Lepoint, & Raykova. (2020). Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. *CCS.* New York: https://dl.acm.org/doi/10.1145/3372297.3417885.

Kadhe, Rajaraman, Koyluoglu, & Ramchandran. (2020). FastSecAgg: Scalable Secure Aggregation for Privacy-Preserving Federated Learning. *ICML-FL & CCS-PPML.* Virtual: http://arxiv.org/abs/2009.11248.

So, He, Yang, Li, Yu, Ali, . . . Avestimehr. (2022). LightSecAgg: a Lightweight and Versatile Design for Secure Aggregation in Federated Learning. *MLSys.* Santa Clara, CA, USA: https://proceedings.mlsys.org/paper_files/paper/2022/hash/6c44dc73014d66ba49b28d483a8f8b0d-Abstract.html.

Smart, Pastro, Damgård, & Zakarias. (2012). Multiparty Computation from Somewhat Homomorphic Encryption. *Crypto.* Santa Barbara, CA, USA: https://link.springer.com/chapter/10.1007/978-3-642-32009-5_38.