Flamingo: Multi-Round Single-Server Secure Aggregation with Applications to Private Federated Learning

Yiping Ma\textsuperscript{1} Jess Woods\textsuperscript{1} Sebastian Angel\textsuperscript{1,2} Antigoni Polychroniadou\textsuperscript{3} Tal Rabin\textsuperscript{1}

\textsuperscript{1}University of Pennsylvania
\textsuperscript{2}Microsoft Research
\textsuperscript{3}J.P. Morgan AI Research & AlgoCRYPT CoE
Data-driven applications nowadays

Service providers collect and analyze user data in order to provide customized functionalities.
Data-driven applications nowadays

Simply put, to protect users at scale, we rely on machine learning powered by user feedback to catch spam and help us identify patterns in large data sets—making it easier to adapt quickly to ever-changing spam tactics. Gmail employs a number of AI-driven filters that determine what gets marked as spam. These filters look at a variety of signals, including characteristics of the IP address, domains/subdomains, whether bulk senders are authenticated, and user input. User feedback, such as when a user marks a certain email as spam or signals they want a sender’s emails in their inbox, is key to this filtering process, and our filters learn from user actions.
Building a machine learning model

Step 1. Collect data from many users.

(“hotel”, not spam) ("malicious", spam) ("hacked", spam)
Building a machine learning model

Step 2. Run a training algorithm on the collected data.
Building a machine learning model

Step 3. Deploy the model in their services.
Centralized vs. decentralized training

Centralized

Decentralized

Federated learning [McMahan et al. in 2016]

Many clients (users) collaboratively train a model under the orchestration of a central server (service provider).

Data never leaves user devices!
Centralized vs. decentralized training

**Centralized**

- **Train**:
  - ("hacked", spam)
  - ("hotel", not spam)
  - ("malicious", spam)

**Decentralized**

- **Update**:
  - ("hacked", spam)
  - ("malicious", spam)
  - ("hotel", not spam)

**Local weights**
Centralized vs. decentralized training

Centralized

\[
\text{Global weights} = \text{Train}\left(\begin{array}{l}
\text{ (“hacked”, spam)} \\
\text{ (“hotel”, not spam)} \\
\text{ (“malicious”, spam)} \\
\end{array}\right)
\]

Decentralized

\[
\begin{align*}
\text{Global weights} & = \text{Update}\left(\text{ (“hacked”, spam)}\right) \\
& + \text{Update}\left(\text{ (“malicious”, spam)}\right) \\
& + \text{Update}\left(\text{ (“hotel”, not spam)}\right)
\end{align*}
\]
Centralized vs. decentralized training

Centralized

A few hundreds to a few thousands of clients

Decentralized

A few hundreds to a few thousands of clients

Train

(“hacked”, spam)
(“hotel”, not spam)
(“malicious”, spam)
(“apple”, not spam)
(“dog”, not spam)
(“lottery”, spam)
(“random”, spam)
Centralized vs. decentralized training

**Centralized**

\[ \text{Train}(\text{"hacked"}, \text{spam}) \]
\[ \text{Train}(\text{"hotel"}, \text{not spam}) \]
\[ \text{Train}(\text{"malicious"}, \text{spam}) \]
\[ \text{Train}(\text{"apple"}, \text{not spam}) \]
\[ \text{Train}(\text{"dog"}, \text{not spam}) \]
\[ \text{Train}(\text{"lottery"}, \text{spam}) \]
\[ \text{Train}(\text{"random"}, \text{spam}) \]

**Decentralized**

\[ = + + \]
Centralized vs. decentralized training

Centralized

Centralized

Train

= \left( \begin{array}{l}
(\text{“hacked”, spam}) \\
(\text{“hotel”, not spam}) \\
(\text{“malicious”, spam}) \\
(\text{“apple”, not spam}) \\
(\text{“dog”, not spam}) \\
(\text{“lottery”, spam}) \\
(\text{“random”, spam}) \\
\end{array} \right)

Decentralized

Decentralized

= + +
Federated learning: steps forward

Only the first step to address privacy issues

- **Inference attacks**
  Infer information about individual data from local weights [Zhu et al. 2019]
- **Poisoning attacks**
  Contribute bogus training data [Biggio et al. 2012] or bogus model weights to undermine the performance of trained models [Bagdasaryan et al. 2018]
Federated learning: steps forward

- Inference attacks: Infer information about individual data from local weights [Zhu et al. 2019]
- Poisoning attacks: Contribute bogus training data [Biggio et al. 2012] or bogus model weights to undermine the performance of trained models [Bagdasaryan et al. 2018]

Our work deals with this problem
Secure aggregation for federated learning

- **Secure aggregation** (A special case of MPC [Yao 1986])

- Only learns $x_1 + x_2 + x_3$

Many works under different communication models, cryptographic assumptions, etc.

- Secret sharing
  [KRKR 2020], [DSQG+ 2022], ...
- Threshold homomorphic encryption
  [SGA 2021], [SHYL+ 2022], ...
- Pairwise masking
  [BIKM+ 2017], [BBGLR 2020], ...
- Other styles
  [SSVR+ 2022], [GPSBB 2022], ...
Federated learning has complex setting

- Clients are mobile devices
  - Limited computation power
  - Different network conditions

The federation property

Machine learning

- Inputs are large (e.g., 8K in MNIST)
- Many participants (e.g., 500 clients per sum)
- Many iterations (e.g., 200)
We are aiming for...

- Clients are mobile devices
  - Limited computation power
  - Different network conditions

  {Lightweight client computation
   Tolerate dropouts at any point}

- Inputs are large (e.g., 8K in MNIST)
- Many participants (e.g., 500 clients per sum)
- Many iterations (e.g., 200)

{How to handle these large system parameters?}
We are aiming for...

- Clients are mobile devices
  - limited computation power
  - different network conditions
- Inputs are large (e.g., 8K in MNIST)
- Many participants (e.g., 500 clients per sum)
- Many iterations (e.g., 200)

Lightweight client computation
Tolerate dropouts at any point

Work well for one summation:
E.g., [BBGLR 2020]

But not for many summations!
We are aiming for...

• Clients are mobile devices
  limited computation power
  different network conditions

• Inputs are large (e.g., 8K in MNIST)
• Many participants (e.g., 500 clients per sum)
• Many iterations (e.g., 200)

Lightweight client computation

Tolerate dropouts at any point

Work well for one summation:
E.g., [BBGLR 2020]

One summation: multiple round trips, some of which are expensive
We are aiming for...

- Clients are mobile devices with limited computation power and different network conditions.
  - Lightweight client computation
  - Tolerate dropouts at any point
- Inputs are large (e.g., 8K in MNIST)
- Many participants (e.g., 500 clients per summation)
- Many iterations (e.g., 200)

One summation: multiple round trips, some of which are expensive.

Reduce the number of round trips → Efficiently handle a full training

E.g., [BBGLR20]
Having fewer round trips is important

• Each iteration (one sum): several round trips
  Reduce run time
  Improve quality and reduce bias

• For all iterations
  Can we do better than just repeating the single-sum protocol?
Having fewer round trips is important

• Each iteration (one sum): several round trips
  Reduce run time
  Improve quality and reduce bias

• For all iterations

Can we do better than just repeating the single-sum protocol?
Having fewer round trips is important

- Each iteration (one sum): several round trips
  - Reduce run time
  - Improve quality and reduce bias

- For all iterations
  - Can we do better than just repeating the single-sum protocol?
We propose Flamingo

- Clients are mobile devices with limited computation power and different network conditions
- Lightweight client computation
- Tolerate dropouts at any point
- Inputs are large (e.g., 8K in MNIST)
- Many participants (e.g., 500 clients per sum)
- Many iterations (e.g., 200)
- Handle a full training session

Without compromising the security guarantees in prior works
Flamingo has two key ideas

• A fault-tolerant private sum protocol based on pairwise secrets and threshold decryption

• A way to reuse pairwise secrets over many iterations
A fault-tolerant private sum protocol

**Pairwise secrets**

BIKM+ 2017, BBGLR 2020

Take some cost to set them up

\[ \vec{v}_1 = \vec{x}_1 + \text{PRG}(s_{12}) + \text{PRG}(s_{13}) \]
A fault-tolerant private sum protocol

Pairwise secrets

BIKM+ 2017, BBGLR 2020

Efficient despite large inputs

$\overrightarrow{v_1} = \overrightarrow{x_1} + \text{PRG}(s_{12}) + \text{PRG}(s_{13})$

$\overrightarrow{v_2} = \overrightarrow{x_2} - \text{PRG}(s_{12}) + \text{PRG}(s_{23})$

$\overrightarrow{v_3} = \overrightarrow{x_3} - \text{PRG}(s_{13}) - \text{PRG}(s_{23})$

$$\sum_{i=1}^{3} \overrightarrow{v_i} = \sum_{i=1}^{3} \overrightarrow{x_i}$$
A fault-tolerant private sum protocol

**Threshold decryption**

\[
\begin{align*}
\overrightarrow{v_1} &= \overrightarrow{x_1} + \text{PRG}(s_{12}) + \text{PRG}(s_{13}) \\
\overrightarrow{v_2} &= \overrightarrow{x_2} - \text{PRG}(s_{12}) + \text{PRG}(s_{23})
\end{align*}
\]

Went offline…

Reveal the secrets!
A fault-tolerant private sum protocol

Threshold decryption

How are they chosen?
How is $SK$ shared among them?
A fault-tolerant private sum protocol

Threshold decryption

\[
\begin{align*}
\vec{v}_1 &= \vec{x}_1 + \text{PRG}(s_{12}) + \text{PRG}(s_{13}) \\
\vec{v}_2 &= \vec{x}_2 - \text{PRG}(s_{12}) + \text{PRG}(s_{23}) \\
\vec{v}_3 &= \vec{x}_3 - \text{PRG}(s_{13}) - \text{PRG}(s_{23}) \\
\end{align*}
\]

\[
\vec{v}_1 + \vec{v}_2 - \text{PRG}(s_{13}) - \text{PRG}(s_{23}) = \vec{x}_1 + \vec{x}_2
\]
A fault-tolerant private sum protocol

Threshold decryption

\[ \text{Enc}(\mathcal{PK}, s_{13}) \]  
\[ \text{Enc}(\mathcal{PK}, s_{23}) \]

Recovery is lightweight
Reusing the secrets

Essentially OTP

Iteration $t$: $s_{12}^t = \text{PRF}(s_{12}, t)$

Simple idea, but cannot work for [BBGLR 2020] due to a crucial difference in fault tolerance design
With the two key ideas →

Do the costly setup once, and run the lightweight sum many times
More details in the paper

• How *decryptors* work
  • Selection
  • Sharing of *SK*
  • Switching decryptors over time

• How *setup* is done

• How to achieve *malicious security*

• Efficient instantiation of cryptographic primitives
Evaluation results

• A factor that was underestimated
  • Server computation was the focus of most of the prior work
  • When server computation is made cheap, what matters is the “waiting time”
Evaluation results

• Feasibility of training a neural network on CIFAR100

We used ABIDES simulator: [https://github.com/abides-sim/abides](https://github.com/abides-sim/abides)
Summary

• This work: A secure aggregation system that handles real-world federated training tasks

• Many interesting future directions
  • Validation of client inputs
  • Stronger security, e.g., adaptive adversary

Thanks!

Code available at https://github.com/eniac/flamingo
Backup Slides
Handling a malicious server

Alice

Bob

Charlie

Decryptors
Handling a malicious server

Alice

Bob

Charlie

Decryptors

$\text{Enc}(PK, s_{12})$

$\text{Enc}(PK, s_{13})$

$\text{Enc}(PK, s_{23})$

$\overrightarrow{x_1}$

$\overrightarrow{x_2}$

$\overrightarrow{x_3}$

$s_{12}$

$s_{13}$

$s_{23}$

$SK_1$

$SK_2$

$SK_3$
Cross check

Key idea:
Honest decryptors agree on what to decrypt
Private sum with 3 rounds

\[
\begin{align*}
&x_1 \quad x_2 \quad x_3 \\
&v_1^T \quad v_2^T \quad v_3^T
\end{align*}
\]

\[
\begin{align*}
\text{Waiting...} & \quad \text{Process...} \quad \vec{v}_1 + \vec{v}_2 \\
\text{Enc}(PK, s_{12}^t) & \quad \text{Enc}(PK, s_{13}^t)
\end{align*}
\]

\[
\begin{align*}
\text{Process...} & \quad \text{Waiting...} \\
\text{Partial decryptions} & \quad \vec{v}_1 + \vec{v}_2
\end{align*}
\]

\[
\begin{align*}
\text{Waiting...} & \quad \text{Process...} \\
\text{Process...} & \quad \vec{v}_1 + \vec{v}_2
\end{align*}
\]

\[
\begin{align*}
\text{Process...} & \quad \vec{v}_1 + \vec{v}_2 \\
\text{−PRG}(s_{12}^t) & \quad \text{−PRG}(s_{13}^t)
\end{align*}
\]

\[
\begin{align*}
\text{Process...} & \quad \overline{x}_1 + \overline{x}_2
\end{align*}
\]