Practical Secure Aggregation for Federated Learning on User-Held Data

Keith Bonawitz[*], **Vladimir Ivanov**[*], **Ben Kreuter**[*], **Antonio Marcedone**[†*], **H. Brendan McMahan**[*], **Sarvar Patel**[*], **Daniel Ramage**[*], **Aaron Segal**[*], **and Karn Seth**[*]
[*]`{bonawitz,vlivan,benkreuter,mcmahan,sarvar,dramage,asegal,karn}@google.com`
Google, Mountain View, California 94043
[†]`marcedone@cs.cornell.edu`
Cornell University, Ithaca, New York 14853

1 Introduction

Secure Aggregation is a class of Secure Multi-Party Computation algorithms wherein a group of mutually distrustful parties $u \in \mathcal{U}$ each hold a private value x_u and collaborate to compute an aggregate value, such as the sum $\sum_{u \in \mathcal{U}} x_u$, without revealing to one another any information about their private value except what is learnable from the aggregate value itself. In this work, we consider training a deep neural network in the *Federated Learning* model, using distributed gradient descent across user-held training data on mobile devices, using Secure Aggregation to protect the privacy of each user's model gradient. We identify a combination of efficiency and robustness requirements which, to the best of our knowledge, are unmet by existing algorithms in the literature. We proceed to design a novel, communication-efficient Secure Aggregation protocol for high-dimensional data that tolerates up to $1/3$ of users failing to complete the protocol. For 16-bit input values, our protocol offers $1.73\times$ communication expansion for 2^{10} users and 2^{20}-dimensional vectors, and $1.98\times$ expansion for 2^{14} users and 2^{24}-dimensional vectors.

2 Secure Aggregation for Federated Learning

Consider training a deep neural network to predict the next word that a user will type as she composes a text message to improve typing accuracy for a phone's on-screen keyboard [11]. A modeler may wish to train such a model on all text messages across a large population of users. However, text messages frequently contain sensitive information; users may be reluctant to upload a copy of them to the modeler's servers. Instead, we consider training such a model in a *Federated Learning* setting, wherein each user maintains a private database of her text messages securely on her own mobile device, and a shared global model is trained under the coordination of a central server based upon highly processed, minimally scoped, ephemeral updates from users [14, 17].

A neural network represents a function $f(x, \Theta) = y$ mapping an input x to an output y, where f is parameterized by a high-dimensional vector $\Theta \in \mathbb{R}^k$. For modeling text message composition, x might encode the words entered so far and y a probability distribution over the next word. A training example is an observed pair $\langle x, y \rangle$ and a training set is a collection $D = \{\langle x_i, y_i \rangle; i = 1, \ldots, m\}$. We define a loss on a training set $\mathcal{L}_f(D, \Theta) = \frac{1}{|D|} \sum_{\langle x_i, y_i \rangle \in D} \mathcal{L}_f(x_i, y_i, \Theta)$, where $\mathcal{L}_f(x, y, \Theta) = \ell(y, f(x, \Theta))$ for a loss function ℓ, e.g., $\ell(y, \hat{y}) = (y - \hat{y})^2$. Training consists of finding parameters Θ that achieve small $\mathcal{L}_f(D, \Theta)$, typically using a variant minibatch stochastic gradient descent [4, 10].

In the Federated Learning setting, each user $u \in \mathcal{U}$ holds a private set D_u of training examples with $D = \bigcup_{u \in \mathcal{U}} D_u$. To run stochastic gradient descent, for each update we select data from a random subset $\mathcal{U}' \subset \mathcal{U}$ and form a (virtual) minibatch $B = \bigcup_{u \in \mathcal{U}'} D_u$ (in practice we might have say $|\mathcal{U}'| = 10^4$ while $|\mathcal{U}| = 10^7$; we might only consider a subset of each user's local dataset). The minibatch loss gradient $\nabla \mathcal{L}_f(B, \Theta)$ can be rewritten as a weighted average across users: $\nabla \mathcal{L}_f(B, \Theta) = \frac{1}{|B|} \sum_{u \in \mathcal{U}'} \delta_u^t$ where $\delta_u^t = |D_u| \nabla \mathcal{L}_f(D_u, \Theta^t)$. A user can thus share just $\langle |D_u|, \delta_u^t \rangle$ with the server, from which a gradient descent step $\Theta^{t+1} \leftarrow \Theta^t - \eta \frac{\sum_{u \in \mathcal{U}'} \delta_u^t}{\sum_{u \in \mathcal{U}'} |D_u|}$ may be taken.

Although each update $\langle |D_u|, \delta_u^t \rangle$ is ephemeral and contains less information then the raw D_u, a user might still wonder what information remains. There is evidence that a trained neural network's parameters sometimes allow reconstruction of training examples [8, 17, 1]; might the parameter updates be subject to similar attacks? For example, if the input x is a one-hot vocabulary-length vector encoding the most recently typed word, common neural network architectures will contain at least one parameter θ_w in Θ for each word w such that $\frac{\partial \mathcal{L}_f}{\partial \theta_w}$ is non-zero only when x encodes w. Thus, the set of recently typed words in D_u would be revealed by inspecting the non-zero entries of δ_u^t. The server does not need to inspect any individual user's update, however; it requires only the sums $\sum_{u \in \mathcal{U}} |D_u|$ and $\sum_{u \in \mathcal{U}} \delta_u^t$. Using a Secure Aggregation protocol would ensure that the server learns only that *one or more* users in $\mathcal{U}$ wrote the word w, but not *which* users.

Federated Learning systems face several practical challenges. Mobile devices have only sporadic access to power and network connectivity, so the set $\mathcal{U}$ participating in each update step is unpredictable and the system must be robust to users dropping out. Because Θ may contain millions of parameters, updates δ_u^t may be large, representing a direct cost to users on metered network plans. Mobile devices also generally cannot establish direct communications channels with other mobile devices (relying on a server or service provider to mediate such communication) nor can they natively authenticate other mobile devices. Thus, Federated Learning motivates a need for a Secure Aggregation protocol that: (1) operates on high-dimensional vectors, (2) is communication efficient, even with a novel set of users on each instantiation, (3) is robust to users dropping out, and (4) provides the strongest possible security under the constraints of a server-mediated, unauthenticated network model.

3 A Practical Secure Aggregation Protocol

In our protocol, there are two kinds of parties: a single server S and a collection of n users $\mathcal{U}$. Each user $u \in \mathcal{U}$ holds a private vector x_u of dimension k. We assume that all elements of both x_u and $\sum_{u \in \mathcal{U}} x_u$ are integers on the range $[0, R)$ for some known R[1]. Correctness requires that if all parties are honest, S learns $\bar{x} = \sum_{u \in \bar{\mathcal{U}}} x_u$ for some subset of users $\bar{\mathcal{U}} \subseteq \mathcal{U}$ where $|\bar{\mathcal{U}}| \geq \frac{n}{2}$. Security requires that (1) S learns nothing other than what is inferable from $\bar{x}$, and (2) each user $u \in \mathcal{U}$ learns nothing. We consider three different threat models. In all of them, all users follow the protocol honestly, but the server may attempt to learn extra information in different ways[2]:

(T1) The server is honest-but-curious, that is it follows the protocol honestly, but tries to learn as much as possible from messages it receives from users.

(T2) The server can lie to users about which other users have dropped out, including reporting dropouts inconsistently among different users.

(T3) The server can lie about who dropped out (as in T2) and also access the private memory of some limited number of users (who are following the protocol honestly themselves). (In this, the privacy requirement applies only to the inputs of the remaining users.)

Protocol 0: Masking with One-Time Pads We develop our protocol in a series of refinements. We begin by assuming that all parties complete the protocol and possess pair-wise secure communication channels with ample bandwidth. Each pair of users first agree on a matched pair of input perturbations. That is, user u samples a vector $s_{u,v}$ uniformly from $[0, R)^k$ for each other user v. Users u and v exchange $s_{u,v}$ and $s_{v,u}$ over their secure channel and compute perturbations $p_{u,v} = s_{u,v} - s_{v,u} \pmod{R}$, noting that $p_{u,v} = -p_{v,u} \pmod{R}$ and taking $p_{u,v} = 0$ when $u = v$. Each user sends to the server: $y_u = x_u + \sum_{v \in \mathcal{U}} p_{u,v} \pmod{R}$. The server simply sums the perturbed values: $\bar{x} = \sum_{u \in \mathcal{U}} y_u \pmod{R}$. Correctness is guaranteed because the paired perturbations in y_u cancel:

$$\bar{x} = \sum_{u \in \mathcal{U}} x_u + \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{U}} p_{u,v} = \sum_{u \in \mathcal{U}} x_u + \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{U}} s_{u,v} - \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{U}} s_{v,u} = \sum_{u \in \mathcal{U}} x_u \pmod{R}.$$

Protocol 0 guarantees perfect privacy for the users; because the $s_{u,v}$ factors that users add are uniformly sampled, the y_u values appear uniformly random to the server, subject to the constraint that $\bar{x} = \sum_{u \in \mathcal{U}} y_u \pmod{R}$. In fact, even if the server can access the memory of some users, privacy holds for those remaining. [3]

[1]Federated Learning updates $\delta_u \in \mathbb{R}^k$ can be mapped to $[0, R)^k$ through a combination of clipping/scaling, linear transform, and (stochastic) quantization.

[2]We do not analyze security against arbitrarily malicious servers and users that may collude. We defer this case and a more formal security analysis to the full version.

[3]A more complete and formal argument is deferred to the full version of this paper.

Protocol 1: Dropped User Recovery using Secret Sharing Unfortunately, Protocol 0 fails several of our design criteria, including robustness: if any user u fails to complete the protocol by sending her y_u to the server, the resulting sum will be masked by the perturbations that y_u would have cancelled. To achieve robustness, we first add an initial round to the protocol in which user u generates a public/private keypair, and broadcasts the public key over the pairwise channels. All future messages from u to v will be intermediated by the server but encrypted with v's public key, and signed by u, simulating a secure authenticated channel. This allows the server to maintain a consistent view of which users have successfully passed each round of the protocol. (We assume here, temporarily, that the server faithfully delivers all messages between users.)

We also add a secret-sharing round between users after $s_{u,v}$ values have been selected. In this round, each user computes n shares of each perturbation $p_{u,v}$ using a (t, n)-threshold scheme [4], such as Shamir's Secret Sharing [16], for some $t > \frac{n}{2}$. For each secret user u holds, she encrypts one share with each user v's public key, then delivers all of these shares to the server. The server gathers shares from a subset of the users $\mathcal{U}_1 \subseteq \mathcal{U}$ of size at least t (e.g. by waiting a for a fixed period), then considers all other users dropped. The server delivers to each user $v \in \mathcal{U}_1$ the secret shares that were encrypted for that user; all the users in $\mathcal{U}_1$ now infer a consistent view of the surviving user set $\mathcal{U}_1$ from the set of received shares. When a user computes y_u, she only includes those perturbations related to surviving users; that is, $y_u = x_u + \sum_{v\in\mathcal{U}_1} p_{u,v} \pmod R$.

After the server has received y_u from at least t users $\mathcal{U}_2 \subseteq \mathcal{U}_1$, it proceeds to a new unmasking round, considering all other users to be dropped. From the remaining users in $\mathcal{U}_2$, the server requests all shares of secrets generated by the dropped users in $\mathcal{U}_1 \setminus \mathcal{U}_2$. As long as $|\mathcal{U}_2| > t$, each user will respond with those shares. Once the server receives shares from at least t users, it reconstructs the perturbations for $\mathcal{U}_1 \setminus \mathcal{U}_2$ and computes the aggregate value: $\bar{x} = \sum_{u\in\mathcal{U}_2} y_u - \sum_{u\in\mathcal{U}_2} \sum_{v\in\mathcal{U}_1\setminus\mathcal{U}_2} p_{u,v}$ $\pmod R$. Correctness is guaranteed for $\bar{\mathcal{U}} = \mathcal{U}_2$ as long as at least t users complete the protocol. In this case, the sum $\bar{x}$ includes the values of at least $t > \frac{n}{2}$ users, and all perturbations cancel out:

$$\bar{x} = \left(\sum_{u\in\mathcal{U}_2} x_u + \sum_{u\in\mathcal{U}_2}\sum_{v\in\mathcal{U}_1} p_{u,v}\right) - \sum_{u\in\mathcal{U}_2}\sum_{v\in\mathcal{U}_1\setminus\mathcal{U}_2} p_{u,v} = \sum_{u\in\mathcal{U}_2} x_u + \sum_{u\in\mathcal{U}_2}\sum_{v\in\mathcal{U}_2} p_{u,v} = \sum_{u\in\mathcal{U}_2} x_u \pmod R.$$

However, security has been lost: if a server incorrectly omits u from $\mathcal{U}_2$, either inadvertently (e.g. y_u arrives slightly too late) or by malicious intent, the honest users in $\mathcal{U}_2$ will supply the server with all the secret shares needed to remove all the perturbations that masked x_u in y_u. This means we cannot guarantee security even against honest-but-curious servers (Threat Model T1).

Protocol 2: Double-Masking to Thwart a Malicious Server To guarantee security, we introduce a double-masking structure that protects x_u even when the server can reconstruct u's perturbations. First, each user u samples an additional random value b_u uniformly from $[0, R)^k$ during the same round as the generation of the $s_{u,v}$ values. During the secret sharing round, the user also generates and distributes shares of b_u to each of the other users. When generating y_u, users also add this secondary mask: $y_u = x_u + b_u + \sum_{v\in\mathcal{U}_1} p_{u,v} \pmod R$. During the unmasking round, the server must make an explicit choice with respect to each user $u \in \mathcal{U}_1$: from each surviving member $v \in \mathcal{U}_2$, the server can request *either* a share of the $p_{u,v}$ perturbations associated with u *or* a share of the b_u for u; an honest user v will only respond if $|\mathcal{U}_2| > t$, and will never reveal both kinds of shares for the same user. After gathering at least t shares of $p_{u,v}$ for all $u \in \mathcal{U}_1 \setminus \mathcal{U}_2$ and t shares of b_u for all $u \in \mathcal{U}_2$, the server reconstructs the secrets and computes the aggregate value: $\bar{x} = \sum_{u\in\mathcal{U}_2} y_u - \sum_{u\in\mathcal{U}_2} b_u - \sum_{u\in\mathcal{U}_2}\sum_{v\in\mathcal{U}_1\setminus\mathcal{U}_2} p_{u,v} \pmod R$.

We can now guarantee security in Threat Model T1 for $t > \frac{n}{2}$, since x_u always remains masked by either $p_{u,v}$s or by b_us. It can be shown that in Threat Models T2 and T3 the thresholds must be raised to $\frac{2n}{3}$ and $\frac{4n}{5}$ correspondingly. We defer the detailed analysis, as well as the case of arbitrarily malicious and colluding servers and users, to the full version[5].

Protocol 3: Exchanging Secrets Efficiently While Protocol 2 is robust and secure with the right choice of t, it requires $O(kn^2)$ communication, which we address in this refinement of the protocol.

[4]A (t, n) secret-sharing scheme allows splitting a secret into n shares, such that any subset of t shares is sufficient to recover the secret, but given any subset of fewer than t shares the secret remains completely hidden.

[5]The security argument involves bounding the number of shares the server can recover by forging dropouts.

computation	
User	$O(n^2 + kn)$
Server[6]	$O(kn^2)$
communication	
User	$O(n + k)$
Server	$O(n^2 + kn)$
storage	
User	$O(n + k)$
Server	$O(n^2 + k)$

Table 1: Protocol 4 Cost Summary (derivations deferred to the full paper).

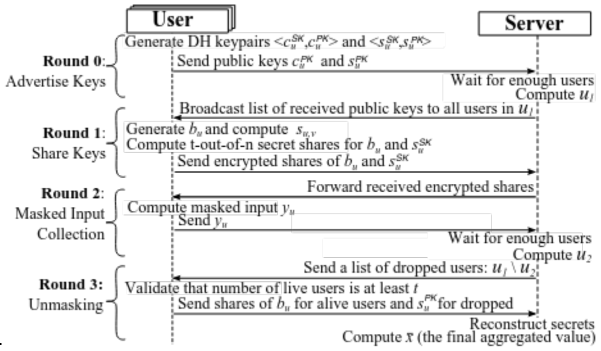
User
Server
Round 0: Advertise Keys
Generate DH keypairs $<c_u^{SK}, c_u^{PK}>$ and $<s_u^{SK}, s_u^{PK}>$
Send public keys c_u^{PK} and s_u^{PK}
Wait for enough users
Compute $\mathcal{U}_1$
Broadcast list of received public keys to all users in $\mathcal{U}_1$
Round 1: Share Keys
Generate b_u and compute $s_{u,v}$
Compute t-out-of-n secret shares for b_u and s_u^{SK}
Send encrypted shares of b_u and s_u^{SK}
Forward received encrypted shares
Round 2: Masked Input Collection
Compute masked input y_u
Send y_u
Wait for enough users
Compute $\mathcal{U}_2$
Send a list of dropped users: $\mathcal{U}_1 \setminus \mathcal{U}_2$
Round 3: Unmasking
Validate that number of live users is at least t
Send shares of b_u for alive users and s_u^{PK} for dropped
Reconstruct secrets
Compute $\bar{x}$ (the final aggregated value)

Figure 1: Protocol 4 Communication Diagram

Observe that a single secret value may be expanded to a vector of pseudorandom values by using it to seed a cryptographically secure pseudorandom generator (PRG) [2, 9]. Thus we can generate just scalar seeds $s_{u,v}$ and b_u and expand them to k-element vectors. Still, each user has $(n-1)$ secrets $s_{u,v}$ with other users and must publish shares of all these secrets. We use key agreement to establish these secrets more efficiently. Each user generates a Diffie-Hellman secret key s^{SK} and public key s^{PK}. Users send their public keys to the server (authenticated as per Protocol 1); the server then broadcasts all public keys to all users, retaining a copy for itself. Each pair of users u, v can now agree on a secret $s_{u,v} = s_{v,u} = \textsc{Agree}(s_u^{SK}, s_v^{PK}) = \textsc{Agree}(s_v^{SK}, s_u^{PK})$. To construct perturbations, we assume a total ordering on $\mathcal{U}$ and take $p_{u,v} = \mathrm{PRG}(s_{u,v})$ for $u < v$, $p_{u,v} = -\mathrm{PRG}(s_{u,v})$ for $u > v$, and $p_{u,v} = 0$ for $u = v$ (as before). The server now only needs to learn s_u^{SK} to reconstruct all of u's perturbations; therefore u need only distribute shares of s_u^{SK} and b_u during the secret sharing round. The security of Protocol 3 can be shown to be essentially identical to that of Protocol 2 in each of the different threat models.

Protocol 4: Minimizing Trust in Practice Protocol 3 is not practically deployable for mobile devices because they lack pairwise secure communication and authentication. We propose to bootstrap the communication protocol by replacing the exchange of public/private keys described in Protocol 1 with a server-mediated key agreement, where each user generates a Diffie-Hellman secret key c^{SK} and public key c^{PK} and advertises the latter together with s^{PK}[7]. We note immediately that the server may now conduct man-in-the-middle attacks, but argue that this is tolerable for several reasons. First, it is essentially inevitable for users that lack authentication mechanisms or a pre-existing public-key infrastructure. Relying only on the non-maliciousness of the bootstrapping round also constitutes minimization of trust: the code implementing this stage is small and could be publicly audited, outsourced to a trusted third party, or implemented via a trusted compute platform offering a remote attestation capability [7, 6, 18]. Moreover, the protocol meaningfully increases security (by protecting against anything less than an actively malicious attack by the server) and provides forward secrecy (compromising the server at any time after the key exchange provides no benefit to the attacker, even if all data and communications had been fully logged).

We summarize the protocol's performance in Table 1. Taking that key agreement public keys and encrypted secret shares are 256 bits and that users' inputs are all on the same range[8] $[0, R_U - 1]$, each user transfers $\frac{256(7n-4)+k\lceil\log_2(n(R_U-1)+1)\rceil+n}{k\lceil\log_2 R_U\rceil}$ more data than if she sent a raw vector.

4 Related work

The restricted case of secure aggregation in which all users but one have an input 0 can be expressed as a dining cryptographers network (DC-net), which provide anonymity by using pairwise blinding of inputs [3, 9], allowing to untraceably learn each user's input. Recent research has examined the communication efficiencly and operation in the presence of malicious users [5]. However, if even one user aborts too early, existing protocols must restart from scratch, which can be very expensive [13]. Pairwise blinding in a modulo addition-based encryption scheme has been explored, but existing schemes are neither efficient for vectors nor robust to even single failure [2, 12]. Other schemes (e.g. based on Paillier cryptosystem [15]) are very computationally expensive.

[6]We reconstruct n secrets from aligned (t, n)-Shamir shares in $O(t^2 + nt)$ by caching Lagrange coefficients.

[7]This can be viewed as bootstrapping a SSL/TLS connection between each pair of users

[8]Taking $R = n(R_U - 1) + 1$ to ensure no overflow

References

[1] Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. *arXiv preprint arXiv:1607.00133*, 2016.

[2] Gergely Ács and Claude Castelluccia. I have a DREAM! (DiffeRentially privatE smArt Metering). In *International Workshop on Information Hiding*, pages 118–132. Springer, 2011.

[3] David Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[4] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. In *ICLR Workshop Track*, 2016. URL `https://arxiv.org/abs/1604.00981`.

[5] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In *Proceedings of the 22nd USENIX Conference on Security*, pages 147–162. USENIX Association, 2013.

[6] Victor Costan and Srinivas Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, 2016. `http://eprint.iacr.org/2016/086`.

[7] Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. Technical report, Cryptology ePrint Archive, Report 2015/564, 201 5. http://eprint. iacr. org.

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

[9] Philippe Golle and Ari Juels. Dining cryptographers revisited. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 456–473. Springer, 2004.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[11] Joshua Goodman, Gina Venolia, Keith Steury, and Chauncey Parker. Language modeling for soft keyboards. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 194–195. ACM, 2002.

[12] Slawomir Goryczka and Li Xiong. A comprehensive comparison of multiparty secure additions with differential privacy. 2015.

[13] Young Hyun Kwon. *Riffle: An efficient communication system with strong anonymity*. PhD thesis, Massachusetts Institute of Technology, 2015.

[14] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[15] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 735–746. ACM, 2010.

[16] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[17] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1310–1321. ACM, 2015.

[18] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten Van Dijk, and Srinivas Devadas. Aegis: architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th annual international conference on Supercomputing*, pages 160–171. ACM, 2003.