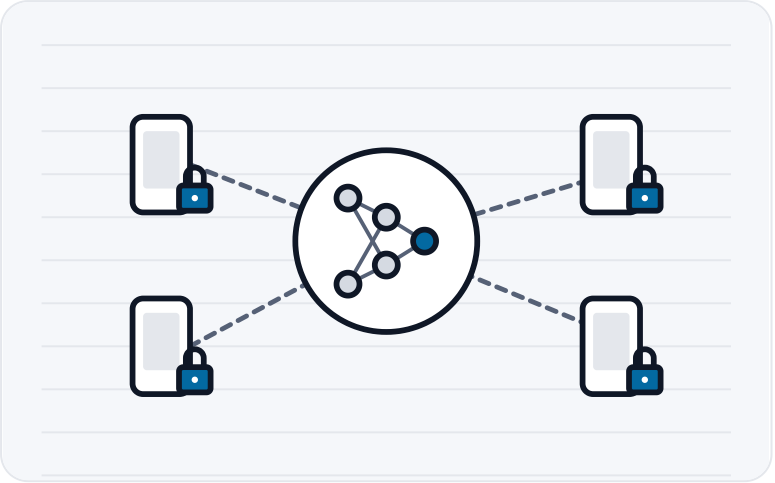
A=SIT Technology
E-Government Innovation Center

22 January 2024 Overview · Privacy & Crypto · Mobile Security · 18 min read

Evaluating the "Secure Aggregation" Aspect in Privacy-Preserving "Federated Machine Learning"

Karl W. Koch



Authors	Karl W. Koch
Version	2023-08-31
Source	https://technology2.a-sit.at/en/publications/secure-aggregation-in-federated-learning/
Note	Machine translation.

Key findings

- Basic federated learning does not protect privacy sufficiently: training data can partly be derived from model updates or from the global model.
- Instead of a complete graph, SecAgg+ uses a sparse random graph, which substantially reduces computation and communication effort.
- FastSecAgg uses the FastShare scheme to reduce the quadratic complexity of secret sharing to n*log(n) and needs 2.5 communication rounds.
- LightSecAgg simplifies the reconstruction of the masks, but like FastSecAgg it only secures against passive corruption.

- The protocols offer trade-offs: SecAgg+ has the best communication complexity, while LightSecAgg and FastSecAgg have lower computation complexity.

In this report, we look at the concept of federated machine learning ("Federated Machine Learning" / FL), and above all at its challenges with regard to the privacy of the participating users' data. In particular, we look at the privacy-preserving variant of FL (= PFL) by means of secure aggregation ("Secure Aggregation" / SAgg). This analysis aims to enable further ML use cases in federated scenarios.

PFL – with SAgg – enables the private learning of an ML model in federated scenarios (many decentralised/distributed users/devices, such as smartphone apps). A server/service provider/etc. learns only from a group of users, while the ML model update of an individual user remains private. One of the most promising SAgg techniques is the use of novel privacy-friendly cryptographic building blocks, in particular based on "Secure Multi-Party Computation" (MPC).

In 2017, Google presented what was effectively the first practically relevant PFL-SAgg protocol based on MPC: SecAgg. Several follow-up works followed. We examine three of these follow-up works more closely: SecAgg+, FastSecAgg and LightSecAgg. The different protocols offer different trade-offs in terms of computational and communication complexity. Finally, a conclusion is drawn on the MPC-based PFL-SAgg protocols examined, and potential follow-up work is outlined.

1. Introduction & Privacy-Related Challenges in Federated Machine Learning ("Federated Learning")

Increasing amounts of data are generated by the devices of individuals, such as smartphones or smartwatches. Examples are fitness and general activity data from trackers such as FitBit [1] or the Apple Watch [2]. Another example is how people use an application (app) on a smartphone. One such app is Google's Gboard [3] [4]: a virtual keyboard in which the app learns the user's input, primarily only locally on the smartphone (as with any other app). But with the (local) usage data, especially when it is combined with the data of other app users, it is possible to train a machine learning (ML) model. Such an ML model can, for example, improve the usability of the app. In the case of Gboard, this is the prediction of the next word or even the next sentence, which ultimately lets end users reply faster and communicate more efficiently. The main problem in such scenarios is protecting the privacy of the users.

In classical ML, all training data is available to one entity that performs the training of an ML model. This entity is usually a central server or a company that obtains the training data from potentially many different sources. However, if the training data is (1) sensitive under data protection law and/or (2) large, this central entity becomes an attractive target for cyber attacks and/or may not be able to store all of this data. In addition, participants may not want to share their (privacy-sensitive) data fully with this central body in the first place. Examples are a user's medical data used for improved cancer prediction, or a user's typed words used to better predict subsequent words or even whole sentences.

Federated Machine Learning (FL). A promising solution to the privacy and storage capacity problem described above is federated machine learning, or "Federated Machine Learning" (FL), which Google presented at AISTATS 2017 (McMahan, Moore, Ramage, Hampson, & Arcas, 2017). In FL, each user trains an ML model locally and sends only the update of the ML model to the central entity: The central entity sends an ML model to the participating users (= clients), and the clients update the model locally based on their (local) data. The clients then send only the ML model update to the central entity. Finally, the central entity computes a weighted sum of all received ML model updates and thereby updates the global ML model. In FL, the central entity therefore neither has access to the clients' (local) training data nor has to store this data. One such cycle – entity→clients→entity – is called an epoch. To increase the accuracy of an ML model, the central entity can run numerous epochs.

Bonawitz et al. (of Google) further divided FL into cross-device FL and cross-silo FL (Bonawitz, Kairouz, McMahan, & Ramage, 2022). Cross-device FL refers to the scenario in which many clients (e.g. 1 million) with relatively low computing power participate, and in which usually only a subset of the clients takes part in an epoch. An example of cross-device FL is the example mentioned above of learning next-word or next-sentence predictions on a smartphone keyboard (Gboard). Cross-silo FL refers to the scenario in which fewer clients (e.g. 100) with relatively large computing power participate, and in which normally all clients take part in an epoch. An example of cross-silo FL is learning patient diagnoses across several hospitals.

Privacy Challenges in FL (à towards PFL). This basic FL protocol is a good step towards protecting the clients' privacy, but it is not sufficient. Since the introduction of FL, several practical attacks based on the clients' ML model updates have been demonstrated (Nasr, Shokri, & Houmansadr, 2019), (Fredrikson, Jha, & Ristenpart, 2015), (Ganju, Wang, Yang, Gunter, & Borisov, 2018), (Carlini, Liu, Kos, & Song, 2019), (Shokri, Stronati, Song, & Shmatikov, 2017). First, it may be possible to (partially) reconstruct a client's training set from its update alone. Second, another attack surface is the ML model itself. A client's training data could be inferred from the global ML model itself, which can occur in particular with outliers. For example, in April 2022 Google showed that a client's text input can be inferred from an ML model for next-word prediction (Bonawitz, Kairouz, McMahan, & Ramage, 2022). Figure 1 shows this problem of inference using the learned global ML model. That is, there are fundamentally attacks on the two main phases of FL: (1) during training (updating the ML model with new client parameters), and (2) during inference (evaluating the ML model).

With SAgg, the clients' new ML model parameters are combined in such a way that the central entity receives the new parameters of all clients at once (= the sum of all parameters), but cannot draw conclusions about the parameters of individual clients. Provided, of course, that the number of participating clients is large enough. When noise is added, for example to the input data or to the resulting ML model parameters, the data is deliberately distorted (statistically) (= noised), so that no or hardly any conclusions about individual clients are possible (primarily) in the inference phase.

Bonawitz et al. call these two techniques the practice of (1) data minimisation and (2) data anonymisation, in particular anonymous aggregation (Bonawitz, Kairouz, McMahan, & Ramage, 2022). In addition, Bonawitz et al. introduce a third key aspect regarding the privacy challenges in FL: consent and transparency with regard to the application/service itself. Consent and transparency essentially mean that the clients know what happens (with their data) and that they agree to it. Unlike the first two key aspects, this third key aspect cannot be solved with technical measures.

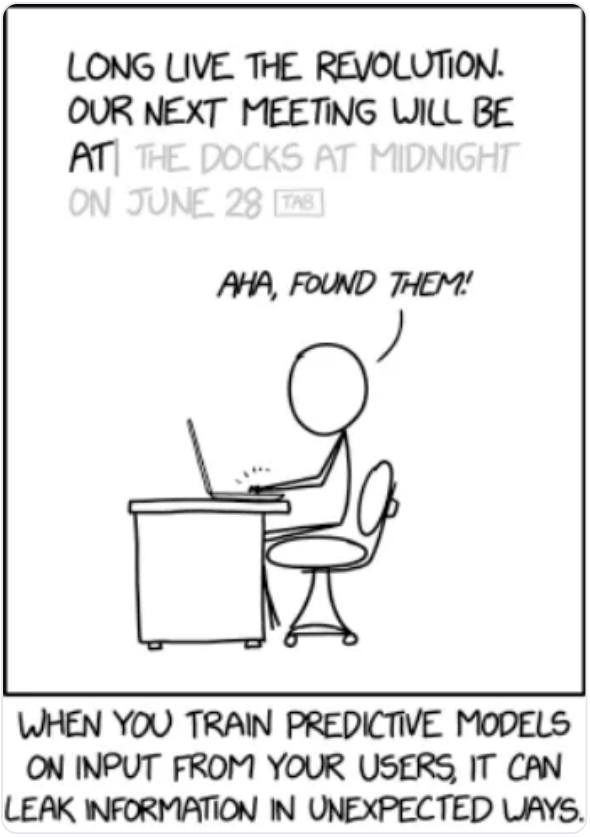


Figure 1. An example where the learned global ML model, for the prediction of the next words, reveals sensitive information. Source: https://xkcd.com/2169

For this reason, FL was extended with privacy-preserving building blocks (= Privacy-Preserving FL (PFL)). In the training phase through so-called Secure Aggregation (SAgg / "Secure Aggregation"). And in the inference phase through a kind of masking of the data or the addition of noise.

In summary. To protect the clients' privacy – ideally conclusively – a PFL protocol must (1) protect the clients' ML model updates and (2) ensure that the clients' training data cannot be inferred from the ML model itself, which can occur in particular with outliers. The first challenge (1) can be – or primarily is – solved with cryptographic primitives such as Secure Multi-Party Computation (MPC) or homomorphic encryption and is called Secure Aggregation (SAgg / "Secure Aggregation"). The second challenge (2) can be solved by adding (statistical) noise to the training data or to the clients' new ML model parameters and is usually implemented by means of Differential Privacy (DP / "Differential Privacy").

If both aspects are taken into account in PFL, we speak of End-to-End Privacy-Preserving Federated (Machine) Learning (EPFL). As a first step towards EPFL, this report concentrates on the first aspect: PFL with MPC-based SAgg.

2. MPC-Based Secure Aggregation in Federated Machine Learning

One of the most promising approaches for Secure Aggregation (SAgg / "Secure Aggregation") of client updates of the ML model parameters in PFL is based on MPC – specifically on the aspect of Secret Sharing (SeSh / "Secret Sharing").

The first MPC-based SAgg considered practical was presented at CCS 2017 by Google and Cornell Tech (Bonawitz et al.) (Bonawitz, et al., 2017). Their approach is usually called SecAgg and consists of at most five rounds plus a setup phase:

0. Setup The server sends each client the parameters for this protocol, in particular the security parameter κ, the number of clients n, the threshold t and the finite field $\mathbb{F}$ for SeSh and the number space used $\mathbb{Z}$"! . In addition, each client sets up a secure and private authenticated channel with the server (e.g. TLS with a pinned certificate).

1. Computation and exchange of (public) DH keys Each client generates 2 Diffie-Hellman (DH) key pairs and shares their public parts.

2. Splitting (SeSh) and exchange of mask-related values Each client u splits and sends the respective values needed for (un)masking:

a) the secret DH key s#$% for the shared masks → su,v

b) a random element for the individual mask (only u) → bu

u performs a Shamir t-out-of-n threshold SeSH for these two values. That is, the values are split into n parts, and at least t parts are needed to recompute the original value (Shamir, 1979).

3. Masking & updating the new ML model parameters Based on a pseudorandom generator (PRG), each client computes the respective mask values and sends the new masked ML model parameters to the server: $y\# = x\# + m\# + \sum' m\#,'$. mu is based on bu. mu,v is based on su,v. And each value in the equation is a vector; its length corresponds to the number of ML model parameters.

4. Ensuring consistency (active corruption mode / “malicious security”) Since in each round (1) certain clients could stop participating in the protocol, or (2) some clients are or become corrupt and do not send correct data: each client additionally signs the list of participating clients that sent an update of the ML model. This list is needed for a possible recovery process during unmasking (= computing the SAgg sum).

5. Secure reconstruction of the aggregated new ML model parameters Each client sends the shares it received to the server:

a) share of bv for each client that sent an ML model update

b) share of s′$% for each client that did not send an ML model update but shared mask-related values in the previous round (otherwise the correspondingly added mask of the “active” clients would not cancel out in the final equation)

That is, clients that have sent an update of the ML model are called “active” clients. Now, for each client in a), bv is reconstructed and the individual mask mv is computed. For each client in b), s#$% is reconstructed and the shared masks mu,v are computed (for each “active” client v). Finally, for each “active” client v, the server sums up each received yv and adds/subtracts the corresponding masks: $x = \Sigma\, uyu - \Sigma\, umu + \Sigma\, u,vmu,v$ SecAgg thus consists of 4 rounds for the passive corruption mode, and of 5 rounds for the active corruption mode. And in each round the server needs at least t client responses; otherwise the required secret shares – of the SeSh – cannot be reconstructed to compute the corresponding mask values.

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

SecAgg+ (Bell, Bonawitz, Gascón, Lepoint, & Raykova, 2020) builds on SecAgg; hence the name/abbreviation. While SecAgg uses a complete graph for the masking, SecAgg+ uses a "sparse random graph" of degree k, where k = O(log(n)). In the complete graph, each client adds the shared mask for every other client (as shown in Section 2). In the graph of degree k, each client adds the shared mask "only" for k neighbouring clients. This reduction in computational and communication complexity makes SecAgg+ (considerably) more efficient. Bell et al. (Bell, Bonawitz, Gascón, Lepoint, & Raykova, 2020) state for their protocol for passive corruption that for ~1 billion clients SecAgg+ offers the same efficiency per client as SecAgg does for ~1,000 clients; this is an improvement of ~6 orders of magnitude. Fundamentally, SecAgg+ makes three contributions in the area of PFL:

1. MPC-based SAgg in PFL in the a. passive corruption model ("semi-honest"), b. semi-active corruption model ("semi-malicious"), and 2) (the first) practically instantiated secure shuffling for the "shuffling model" of "Differential Privacy".

SecAgg+ 's variant with the passive corruption model works essentially the same as the variant of SecAgg with the passive corruption model. However, an undirected graph of degree k is used for computation and communication (secret sharing (= SeSh) of the respective mask values and computation of the shared mask values "only" for k neighbouring clients). This graph is created by the server and then transmitted to the clients. Overall, the variant with the passive corruption model consists of 3.5 communication rounds.

In SecAgg+ 's variant with the semi-active corruption model, the server is trusted to perform the setup phase correctly. In addition, the number of corrupted clients is considered fixed at the start of the protocol; SecAgg+ therefore offers no security against adaptive corruption. Furthermore, SecAgg+ 's variant with the semi-active corruption model consists essentially of the same steps as the variant with the passive corruption model, but contains changes to protect against a corrupt server and clients:

0. Setup phase In the setup phase the server is considered semi-honest (passive corruption mode) – the clients generate the two DH key pairs and send the public parts to the server – but the server incorporates the received public DH keys into a Merkle tree.

1. Graph generation phase Each client i randomly chooses k neighbours (outgoing neighbours of i). Then i receives the list of clients that have chosen i as a neighbour (incoming neighbours of i), together with the respective public DH keys. To verify the correctness of the keys, the server provides the required intermediate hashes of the Merkle tree. In addition, i accepts "only" at most 3∗k incoming neighbours.

2. Secret shares phase (= SeSh) Each client i passes the two secret shares – of the SeSh – on to its outgoing neighbours ((1) the seed of the individual mask and (2) the secret DH key used to compute the seed of the shared mask)

3. Mask input phase Each client i computes the masked update of the ML model. For the shared masks, i additionally signs a message that it has included the shared mask (i, j) (= the "attached" signature).

4. Summation phase The server provides each client i with two lists of its incoming neighbours: (a) clients that contributed a masked ML model update (= are still active) and (b) clients that did not contribute an update (= have dropped out). In addition, each client validates its received “attached” signatures; as well as that the intersection of (a) and (b) is an empty set and that all these clients are actual neighbours of i. Then i signs a message for each client in (a) – incoming active client j – that “i knows that j is still active” (= the “ACK” signature). Now j expects “enough” such valid “ACK” signatures from its outgoing neighbours i. Only then do the clients provide the server with the corresponding shares (share of the individual mask for the clients in (a) and share of the secret DH key of the shared mask for the clients in (b)).

2.2. Related PFL Protocols

Further (MPC-based) protocols followed on the basis of the SecAgg approach. Two of these approaches are " FastSecAgg : Scalable Secure Aggregation for Privacy- Preserving Federated Learning " (Kadhe, Rajaraman, Koyluoglu, & Ramchandran, 2020) and " LightSecAgg : a Lightweight and Versatile Design for Secure Aggregation in Federated Learning " (So, et al., 2022).

FastSecAgg introduces a new method for Secret Sharing (= SeSh): FastShare . While the de facto standard for SeSh (so far), Shamir’s t-out-of-n threshold SeSH, incurs quadratic complexity per client, FastShare reduces this to n*log(n). Based on FastShare, FastSecAgg performs SAgg in 3 rounds (2.5 rounds of communication). Instead of adding a mask, each client creates n shares of its ML model update and sends one share to each other client. The clients then locally sum up their (received) shares and send this sum to the server. Finally, the server adds up the sums of all clients to obtain the global aggregated ML model update. The protocol is secure (only) against passive corruption, but allows this adaptively during the protocol run; the corrupt clients are therefore not considered fixed before the protocol starts (as, for example, in SecAgg+).

LightSecAgg builds on SecAgg and SecAgg+. Since the reconstruction of the masks is one of the bottlenecks of the SecAgg/+ protocol, LightSecAgg aims to simplify this. Instead of adding (1) an individual mask and (2) a pairwise shared mask with every other client, in LightSecAgg a client masks the input only with a random vector (= an “individual mask”). This “individual mask” is then partitioned and encoded with a T-private maximum distance separable (MDS) code matrix; and each other client receives a different encoding of this matrix. In the summation phase, each client sums up its (received) encoded mask shares of the still active clients and sends this client aggregation to the server. The server can now sum up all client aggregations to obtain the global aggregated mask. That is, the server (1) reconstructs this aggregated mask in a single step and (2) then reconstructs the global aggregated ML model update using this aggregated mask. Overall, LightSecAgg consists of 3 + 1 rounds (2.5 + 1 communication rounds). In their paper, So-He-Yang et al. assume that the setup phase (= key exchange, etc.) is given, hence the +1. However, LightSecAgg considers only passive corruption.

Table 1 gives an overview of the asymptotic running times of SecAgg, SecAgg+, FastSecAgg, LightSecAgg and plain 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	$O(nl)$	$O(n)$	$O(l)$	$O(l)$

Table 1. Asymptotic overview of the maximum computation and communication costs of the PFL protocols SecAgg, SecAgg+, FastSecAgg, LightSecAgg and of aggregation in "Plain" form (where the resulting ML model parameters are sent to the aggregator in plaintext). For LightSecAgg, U denotes the minimum number of users required to reconstruct the participants' masks, and T the maximum number of corrupt participants.

3. Conclusion & Further Work

Since the introduction of (practical) privacy-preserving federated machine learning (PFL) in 2017 (SecAgg), several follow-up works have appeared. This report examined three follow-up works in more detail: SecAgg+, FastSecAgg and LightSecAgg . Essentially, all the works mentioned offer certain trade-offs. SecAgg+ offers the best communication complexity, but has a relatively high computational complexity. LightSecAgg and FastSecAgg, in contrast, both offer a promising computational complexity, but have a relatively higher communication complexity. However, LightSecAgg and FastSecAgg guarantee security only against passive corruption, whereas SecAgg and SecAgg+ offer variants for (semi-)active corruption. In addition, FastSecAgg introduces a new Secret Sharing (= SeSh) scheme – FastShare – which could be of independent interest in future work for SeSh scenarios (such as in the SPDZ protocol for general MPC (Smart, Pastro, Damgård, & Zakarias, 2012)).

Furthermore, the following steps would be of interest:

- Examine the different trade-offs more closely (e.g. efficiency (computation/memory) / ML model accuracy / robustness / trustworthiness).
- Examine further developments of MPC-based Secure Aggregation (= SAgg) on the one hand, and also in the area of Homomorphic Encryption on the other (as an alternative or complement to MPC-SAgg).
- Examine possible additions with "noise" techniques; e.g. by means of Differential Privacy, where noise is added, for example, to the input data or the ML model parameters in order to preserve privacy during inference.

- Work out commonalities and differences between the general PFL concepts “X-device” (many participants with weak computing power (e.g. smartphones)) and “X-silo” (few participants with strong computing power (e.g. large organisations such as hospitals)) PFL.

Bibliography

McMahan, Moore, Ramage, Hampson, & Arcas. (2017). Communication-Efficient Learning of Deep Networks 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.

---

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)* ↩