PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École Normale Supérieure de Paris

Cryptography for Privacy-Preserving Machine Learning

Soutenue par
Théo Ryffel
Le 23 Juin 2022

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Aurélien Bellet INRIA Lille	*Rapporteur*
Yuval Ishai Technion	*Rapporteur*
Renaud Sirdey CEA	*Président du jury*
Mariya Georgieva Inpher	*Examinatrice*
Laurent Massoulié INRIA, ENS, PSL, Paris	*Examinateur*
Jonathan Passerat-Palmbach Imperial College London	*Examinateur*
David Pointcheval ENS, CNRS, PSL, Paris	*Directeur de thèse*
Francis Bach INRIA, ENS, CNRS, PSL, Paris	*Directeur de thèse*

1794 ENS ÉCOLE NORMALE SUPÉRIEURE

PSL RESEARCH UNIVERSITY PARIS

Résumé

L'usage sans précédent du machine learning (ML) ou *apprentissage automatique*, motivé par les possibilités qu'il apporte dans un grand nombre de secteurs, interroge de plus en plus en raison du caractère sensible des données qui doivent être utilisées et du manque de transparence sur la façon dont ces données sont collectées, croisées ou partagées. Aussi, un certain nombre de méthodes se développent pour réduire son intrusivité sur notre vie privée, afin d'en rendre son usage plus acceptable notamment dans des domaines tels que la santé, où son potentiel est encore très largement sous-exploité.

Cette thèse explore différentes méthodes issues de la cryptographie ou plus largement du monde de la sécurité et les applique au machine learning afin d'établir des garanties de confidentialité nouvelles pour les données utilisées et les modèles de ML.

Notre première contribution est le développement d'un socle technique pour implémenter et expérimenter de nouvelles approches au travers d'une librairie open-source nommée PySyft. Nous proposons une architecture modulaire qui permet à chacun et chacune d'utiliser les briques de confidentialité nécessaires selon son contexte d'étude, ou encore de développer et d'interfacer de nouvelles briques. Ce socle sert de base à l'ensemble des implémentations proposées dans cette thèse.

Notre seconde contribution consiste à mettre en lumière la vulnérabilité des modèles de ML en proposant une attaque qui exploite un modèle entraîné et permet de révéler des attributs confidentiels d'un individu. Cette attaque pourrait par exemple détourner un modèle qui reconnaît le sport fait par une personne à partir d'une image, pour détecter les origines raciales de cette personne. Nous proposons des pistes pour limiter l'impact de cette attaque.

Dans un troisième temps, nous nous intéressons à certains protocoles de cryptographie qui permettent de faire des calculs sur des données chiffrées. Une première étude propose un protocole de chiffrement fonctionnel qui permet de réaliser des prédictions grâce à un petit modèle de ML à partir de données chiffrées et de ne rendre public que la prédiction. Une seconde étude porte sur l'optimisation d'un protocole de partage de secret fonctionnel, qui permet d'entraîner ou d'évaluer un modèle de ML sur des données de façon privée, c'est à dire sans révéler à quiconque ni le modèle ni les données. Ce protocole offre des performances suffisantes pour utiliser des modèles qui ont une utilité pratique dans des tâches non triviales comme la détection de pathologies dans les radiographies de poumons.

Dans un dernier temps, nous intéressons à la confidentialité différentielle qui permet de limiter la vulnérabilité des modèles de ML et donc l'exposition des données qui sont utilisées lors de l'entraînement, en introduisant une perturbation contrôlée. Nous proposons un protocole et démontrons qu'il offre notamment la possibilité d'entraîner un modèle lisse et fortement convexe en garantissant un niveau de confidentialité indépendant du nombre d'accès aux données sensibles lors de l'entraînement.

Mots clés : Apprentissage Automatique, Apprentissage Fédéré, Chiffrement Fonctionnel, Calculs Multipartites, Partage de Secret Fonctionnel, Confidentialité Différentielle, Intégrité Contextuelle

Abstract

The ever growing use of machine learning (ML), motivated by the possibilities it brings to a large number of sectors, is increasingly raising questions because of the sensitive nature of the data that must be used and the lack of transparency on the way these data are collected, combined or shared. Therefore, a number of methods are being developed to reduce its impact on our privacy and make its use more acceptable, especially in areas such as healthcare where its potential is still largely under-exploited.

This thesis explores different methods from the fields of cryptography and security, and applies them to machine learning in order to establish new confidentiality guarantees for the data used and the ML models.

Our first contribution is the development of a technical foundation to facilitate experimentation of new approaches, through an open-source library named PySyft. We propose a modular architecture that allows one to pick the confidentiality blocks necessary for one's study, or to develop and easily integrate new blocks. This library is reused in all the implementations proposed in this thesis.

Our second contribution consists in highlighting the vulnerability of ML models by proposing an attack that exploits a trained model to reveal confidential attributes of an individual. This attack could, for example, subvert a model that recognizes a person's sport from an image, to detect the person's racial origins. We propose solutions to limit the impact of this attack.

In a third step, we focus on some cryptographic protocols that allow us to perform computations on encrypted data. A first study proposes a functional encryption protocol that allows to make predictions using a small ML model over encrypted data and to only make the predictions public. A second study focuses on optimizing a functional secret sharing protocol, which allows an ML model to be trained or evaluated on data privately, i.e. without revealing either the model or the data to anyone. This protocol provides sufficient performance to use models that have practical utility in non-trivial tasks such as pathology detection in lung X-rays.

Our final contribution is in differential privacy, a technique that limits the vulnerability of ML models and thus the exposure of the data used in training by introducing a controlled perturbation. We propose a new protocol and show that it offers the possibility to train a smooth and strongly convex model with a bounded privacy loss regardless of the number of calls to sensitive data during training.

Keywords: Machine Learning, Federated Learning, Functional Encryption, Multi-Party Computation, Function Secret Sharing, Differential Privacy, Contextual Integrity

Acknowledgments

Je me retourne avec un certain vertige sur ces trois années et demie de thèse qui s'achèvent, laissant derrière elles une densité de souvenirs et de rencontres que ces remerciements ne pourront probablement pas restituer à leur juste valeur. Pourtant, je mesure la chance immense qui m'a été accordée de pouvoir réaliser ce projet, secoué comme tant d'autres par la pandémie, et je souhaite vous remercier tous et toutes pour le soutien que vous m'avez apporté, de mille manières différentes mais toujours aussi précieux.

Je souhaite tout d'abord exprimer ma profonde reconnaissance à David Pointcheval et Francis Bach qui m'ont soutenu et guidé tout au long de cette thèse. Le souvenir de nos premiers échanges reste gravé dans ma mémoire, comme cette matinée à l'Inria où Francis il me semble avoir lu dans tes yeux « et puis au fond pourquoi pas » lorsque je te présentais mon projet iconoclaste. En plus du temps que vous avez chacun pris pour me former –moi qui avais soigneusement évité les cours de cryptographie–, de votre réactivité sans faille pour répondre à mes questions à toute heure du jour et de la nuit, vous m'avez ouvert de nombreuses portes, me présentant à de nombreux chercheurs et chercheuses, m'envoyant présenter mes travaux à plusieurs évènements et dans une revue ; cette confiance m'honore profondément. Je garde enfin un souvenir riant de ces décalages entre les exigences de la cryptographie et celles de l'apprentissage automatique qui animaient nos sessions de travail et j'espère pouvoir continuer ces débats avec vous.

Je remercie très spécialement Jonathan Palmbach-Passerat qui m'a encadré lors de ma thèse de master à Londres et a fait germer en moi la passion qui a été la mienne pour le sujet de ce manuscrit. Qui aurait cru que ce projet sobrement intitulé « Federated machine learning on medical data using blockchain » saurait me convertir à une thèse ? Je te remercie une seconde fois d'avoir accepté de prendre part à mon jury, bouclant ainsi la boucle commencée il y a quatre ans.

I would also like to thank Yuval Ishai and Aurélien Bellet who have accepted to be rapporteurs on this manuscript. Their comments and suggestions have been of a precious help and I hope the reading hasn't been too tedious, some chapters being much more cryptography focused and some others more centered around differential privacy or federated learning. Aurélien, I've been told that I have the privilege of your first thesis manuscript review, thank you for this! I also sincerely thank Renaud Sirdey, Laurent Massoulié and Mariya Georgieva who also compose the jury. Having your insights both from the academic and industry world is very precious to me.

Je remercie également l'ensemble des membres de l'équipe crypto de l'ENS avec qui j'ai eu le privilège de travailler, notamment Anca qui m'a montré que la cryptographie pouvait être aussi féerique qu'un livre de contes, Antoine, Aurélien qui a terminé sa thèse peu après mon arrivée me donnant une idée de à quoi ressemblait le produit fini, Azam, Balthazar, Baptiste, Brice, Céline, Chloé pour m'avoir transmis la passion des aquariums et pour les innombrables conseils notamment pour surmonter les différents écueils administratifs, Geoffroy pour tes précieux conseils en FSS, Georg, Guillaume, Hoeteck, Hugo, Huy, Huyen, Jérémy, Jianwei, Ky, Lénaïck, Léonard, Louiza, Mélissa, Michael, Michel pour le soutien indéfectible à mon addiction au café, Michele pour avoir partagé notre petit bureau et pour les expériences toujours troublantes à base de cartes de cantine, Paul, Paola pour avoir repris le flambeau de Michele (mais sans les cartes), Phong, Pierrick, Pooya, Quentin, Robert, Romain pour cette clématite qui n'était vraiment pas en forme et à laquelle j'ai prodigué tant de soins avant qu'elle n'expire, et Xiaxi. Je remercie aussi l'ensemble de l'équipe administrative qui ont tant oeuvré, souvent en coulisse, pour que nous puissions travailler dans d'excellentes conditions, notamment Lise-Marie Bivard, Linda Boulevart, Martine Girardot, Sophie Jaudon, Valerie Mongiat, Nathalie Gaudechoux pour sa bonne humeur inaltérable même au plus profond de l'hiver, ainsi que Jacques Beigbeder et Ludovic Ricardou au SPI.

I would also like to deeply thank the OpenMined community with which I have lived one of the biggest adventures of my life. From discovering the world of open-source, learning to code according to the best practices, discovering collaboration across 4 continents, 24/24 7/7, to managing a team of highly motivated volunteers, this journey has been rich in precious learning lessons. Thanks Andrew for fueling OpenMined with your amazing energy and your good vibes. Thanks Bobby, Jason, Morten, Patrick for putting up the first blocks together. And last but not least, a big thank to all cryptography team, including Pierre Tholoniat, George Muraru, Rasswanth S, Hrishikesh Kamath, Arturo Marquez, Yugandhar Tripathi, S P Sharan, Nicolas Remerscheid, Jason Paumier, Muhammed Abogazia, Alan Aboudib, Ayoub Benaissa, Sukhad Joshi and many others.

Je tiens à remercier l'ensemble d'Arkhn qui m'a accompagné depuis trois ans, et tout particulièrement Alexis, Corneliu et Emeric avec qui tout a commencé. J'ai conscience de la chance qui a été la mienne de pouvoir travailler sur mes problématiques de recherche tout en restant proche du terrain, notamment auprès des hôpitaux, et je sais que cela n'aurait pas été possible sans le travail acharné de notre formidable équipe. Vous êtes incroyables.

Je remercie également l'ensemble de mes amis et camarades de longue date, qui ont parfois essuyé les externalités inhérentes au travail doctoral : l'Orlins crew, le clan corse, la team Saint-Louis, le Binet Marais, l'équipée aventureuse du Styx, les fistons, les Bons Entendeurs, vous vous reconnaîtrez j'en suis certain.

Enfin, je tiens à remercier les plus grands artisans de cette thèse, ma famille, qui par une double action coordonnée de soutien et de persuasion, ont su me mettre sur les rails de la recherche. Sans sarcasme et sans moquerie : merci, ça en valait la peine ! Marianne, merci pour ton soutien sans faille depuis quatre ans, il me semble que tes forces sont sans limites lorsque les miennes viennent à manquer.

A celles et ceux que je n'ai pas mentionné, je suis un ingrat et vous méritez une bière.

Contents

Chapter

1 Introduction

Chapter content

1.1 Motivation

Whatever our opinion of machine learning, it is already part of our everyday life. When we write a message on our phone, an algorithm suggests the next word we might need. When we visit a website, moments before the page is displayed, a lightning auction takes place between different advertisers who analyze our user profile and determine the ad most likely to capture our attention. When watching a series, film or music video, the application makes its best recommendations for us to keep watching. Despite these chilling examples, there are also more socially beneficial applications, from helping to clean up pollution by identifying garbage using aerial images, to preventing infrastructure failures or leaks, to helping diagnose patients in hospitals. Let's look at this last example, because the healthcare sector is very specific. Indeed, while it is one of those where machine learning could have the greatest societal impact, it is nevertheless the one where it is the least present. Why this paradox? A diversion into the origins of machine learning will help us understand it better.

Machine learning actually refers to a large number of techniques and we will focus on deep learning, which is particularly widespread for image analysis. It was conceptualized in the 1950s by analogy with the human brain, and took its current form in the 1980s thanks to the work of John Hopfield [Hop82] and Geoffrey Hinton [RHW86]. The latter defined modern deep learning models, also known as neural networks. The idea of neural networks is to process information (should it be text, images, etc.) through a series of interconnected layers of neurons. It took almost 30 years from these early ideas for neural networks to become popular. Two factors explain this period of hibernation: computing power and available data. Indeed, to be properly trained, neural networks must learn from a large number of examples. The more data there is, the better the network learns; and the bigger the network is, the more complex tasks it can perform, which in turn requires a lot of computation.

It is precisely these two barriers - computing capacity and availability of data - that are being lifted around 2010 in many sectors. On the one hand, the continuous progress in the computational performance of computers, which doubled every two years on average between 1975 and 2015 (the so-called Moore's law), has made it possible to achieve sufficient power to train deep networks on non-trivial tasks. On the other hand, the rise of home computing and the Internet has made it possible to collect large amounts of data that can be used to perform complex learning tasks. For example, the ImageNet image database, resulting from the collection and annotation of more than 14 million images on the Internet, was published for the first time in 2009 [DDS+09] and has served as a reference for the development of many neural networks in the field of imaging. As a result, research work has multiplied and the sectors where a lot of data was available have also benefited from these advances by investing heavily in the analysis of their data. The AI market is estimated by IDC to be worth 340 billion US dollars in 2021, with an estimated growth of 19% for 2022.

However, other sectors such as healthcare have not followed this trajectory. In France for example, each healthcare institution is responsible for storing its health data and the IT system varies greatly from one institution to another. Access to health care data is also highly regulated, and must be done on a case-by-case basis, both for clinical research teams and for companies specializing in AI. This leads to the following paradox that one can rely on public databases containing millions of images to train a model detecting cat species. But to solve public health problems such as the study of depressive behavior, the cohorts available rarely exceed hundreds of individuals. However, the data exists, whether it be medical records, but also email content or bank statements that could be useful for this type of study, but the information is "siloed": it is not shared because the risks of malicious use are too high. From this point of view, the big data revolution in healthcare has not yet happened.

GDPR The General Data Protection Regulation (GDPR), which entered into effect in 2018 in the European Union (EU), redefines and extends the concept of health data. This includes not only medical history, diagnoses, exam results or treatments, but also any information that identifies a person for health purposes (such as a patient identifier). It requires the explicit consent of individuals to the collection and processing of their data for a reason that must be specified and legitimate: it cannot be re-used at a later date for other purposes. Finally, the RGPD indicates that any information system or database must take into account the notion of respect for privacy from the outset and restrict access to designated persons. It strengthens the power of national administrative authorities of EU members like the CNIL in France to impose sanctions on companies in the event of non-compliance, up to 4% of global turnover.

1.1.1 Privacy Through Contextual Integrity

The absolute necessity of protecting health data, as enshrined in the GDPR, is non-negotiable. It is therefore a question of finding a framework that ensures data security while allowing innovation, with the promise of being able to achieve unparalleled advances through the exploitation of data on a national scale. This observation is now shared by many actors, but it requires finding solutions adapted to such a sensitive subject. The controversies surrounding the French Health Data Hub illustrate this well: this is a government initiative that consists of centralizing and facilitating access to French healthcare data for research projects, and which is criticized [PU19, FH19], among other things, for having a scope that is too broad and for storing the data on a cloud belonging to a American company. In the light of this example, it is important not to oppose innovation and respect for privacy. One should not see innovation as a forced march for which necessary sacrifices on privacy are needed, or on the contrary consider that privacy consists of locking up all personal information.

To reconcile diverging visions, we can revisit privacy from the perspective of contextual integrity [Nis04] as conceived by Helen Nissenbaum. Privacy is respected when an information flow from one individual to another via a dedicated channel is appropriate, with respect to the person concerned, the sender, the recipient, the type of information and the transmission principle. For example, someone sending his own medical report to his doctor via a secure messaging system is probably an appropriate flow of information, which does not harm the patient's privacy. Replace now the doctor with the employer, or the secure messaging app with a public communication channel and the flow of information becomes much more problematic. In other words, according to Helen Nissenbaum, privacy is exercised not by locking up one's data but by sharing it appropriately, revealing only to legitimate actors the information they need to have. This approach disrupts the usual notion of privacy, as having channels to exchange appropriate information flows not only limits the negative effects of inappropriate information dissemination, it also enhances information sharing where it is legitimate. Through such channels, it becomes possible for institutions to collaborate by sharing only what is necessary, without having to expose all their data and risk it being copied or misused.

Our motivation throughout this thesis is to explore several privacy-preserving techniques in the context of machine learning and to link them with the notion of contextual integrity. More specifically, we detail three techniques that can be used separately or jointly depending on the context: federated learning, differential privacy and computation on encrypted data. To illustrate our discussion, we will take the example of a laboratory that wants to train a model to help detect pneumonia in patients from lung imaging data owned by several institutions. This example will accompany us throughout this introduction to relate to a concrete case.

1.1.2 Federated Learning

The standard method for this type of task is to import all the images to a central server to train the model. This poses several problems. The first is that, in relation to the desired task (detecting whether a patient has pneumonia), the information flow is not necessarily appropriate as it may not be useful to transfer all the lung imaging data. The second, more practical, is that it is complicated to transfer large amounts of data such as imaging, and the question of where to store it is a thorny one: should it be stored on a cloud? If so, a national cloud or not? If not, which institution should be responsible for storing the data?

Federated learning solves these questions by reversing the way things are done: it consists of leaving the data where it is hosted (in hospitals, for example) and sending the neural network there to be trained. The process works like this: say a model should be trained on data from three hospitals. A copy of the initial and untrained version of the model is first sent to each center to train locally on their data. After a certain learning period, each center sends back its model to a central server. Models are then aggregated to build a single neural network resulting from the training on the three centers, which is sent back to the centers. These two phases of local learning and aggregation alternate until sufficient performance is obtained for the aggregated model.

Federated learning [MMR+17] builds upon a long line of research is distributed learning [VBT17, BT97, MR03], but differs in the sense that it exploits computing resources that are distributed because the data is intrinsically distributed. It has developed since 2017 under the influence of technology giants who first wanted to apply it to phones. One main use case is the prediction of the next word when writing text messages, for which transferring personal data to central servers would be unacceptable from a privacy standpoint. Today, this type of learning is applied in very different contexts, depending on whether few or millions of parties possess the data as in the case of telephones. This is a subject that is evolving very quickly and is the subject of active academic and industrial research in France and other countries. We have been involved in developing open-source tools for federated learning, including the pioneering PySyft library that we inspect more in depth in Chapter 3. The main challenges of this technique are the impact on bandwidth linked to the transfer of large models and the standardization of data. To reduce bandwidth usage, it is possible to transfer only certain layers of the networks or to compress the model parameters, at the expense of the learning quality. Regarding standardization, it is necessary that the centers receiving the models have a uniform data model. Indeed, the model cannot adapt to the way the data are stored, which can be very different, as in the case of hospital information systems.

Let us now take up the notion of contextual integrity as defined above and analyze the notion of federated learning from this perspective. It is not necessary to expose the data directly because it is ultimately the model and its ability to aid diagnosis that we want to extract. The information flow is therefore composed of the versions of the model updated by the different stakeholders. The aggregation of these versions is done either by the parties directly or by a third party central server. The question is therefore whether it is acceptable for a third party or the parties involved in the learning process to have access to the final model. Behind this question lies another: is it possible for the model to reveal more information than it was designed for? In particular, can it reveal specific information about the data on which it was trained, the same data that it was decided not to expose? The answer is yes. Due to their very large size (several million of parameters), models can store specific information about certain training data, even if they have not been overexposed to this data (so-called overfitting). Studies have looked at the various pieces of information that can be extracted from these models, ranging from straightforward reconstruction of certain training data [FJR15] to more subtle manipulations such as identifying whether a piece of data was in the training set or not [SSSS17]. We also have formalized in Chapter 4 an attack that exploits trained models to infer specific private attributes about data provided for a given prediction task. Aware of the ability of some models to leak sensitive information, the acceptability of such leakage depends on the context of stakeholder collaboration. In our example, if we only consider hospitals that could have directly shared together their data, it is likely that such leaks would not compromise patient privacy, as long as the model is not made public or shared with other stakeholders or companies.

In some situations, the model needs to be widely shared or published, such as for a mobile melanoma detection application that would not export the images to servers. In this case, a second technique needs to be implemented to resist the extraction attacks just mentioned. This means taking data protection a step further by using differential privacy.

1.1.3 Differential Privacy

Differential privacy [DMNS06] in its current form was theorized in 2006 by Cynthia Dwork, Frank McSherry, Kobbi Nissim and Adam Smith. It can be formalized as follows: an algorithm achieves a certain level of differential privacy, if, given two arbitrary datasets as input that differ by only one individual, its output distribution differs by at most a certain margin. The intuition behind the formula they proposed is that adding or removing an individual from a dataset should not significantly alter the behavior of an algorithm that respects differential privacy. This means that an individual alone should not have any influence on the algorithm. In other words, the algorithm should reveal as little information as possible about a single individual.

The concept of differential privacy applies not only to machine learning but also to other fields such as statistics in general and to any aggregation process on databases of individuals. Techniques for obtaining differential privacy are usually based on the addition of random noise. Let's take the example of a sociological study on the religious practices of a population, which leverages randomized response [War65]. Each person is asked whether they are religious or not, but before they do so, they are asked to toss a coin twice without anyone else being able to see the result. If they get "heads" on the first toss, they must give their true answer, and if they get "tails", they should answer randomly according to the second toss. If the study is carried out on a large number of people, and we obtain for example 60% of people who say that they are believers, as we know that half of the people questioned answered randomly and the other half answered honestly, we deduce that the real proportion of believers is 70%. Thus the result of the study is preserved, but it is not possible to know whether a particular person is a believer or not. This notion is also known as plausible deniability.

In order to understand how differential privacy can be applied to deep learning, let's go back to the way neural networks are trained. To train a network given an objective and a loss function, we often use stochastic gradient descent, a method that updates the model parameters in the opposite direction of their contribution to the loss, with the aim of iteratively reducing the loss i.e. the gap between predictions and the objective. Differential privacy is applied during the parameters update, by adding some extra Gaussian noise [ACG$^+$16]. The variance of this noise is finely calibrated to obtain the right level of confidentiality, to ensure the required privacy level without perturbing too much the training procedure and hurting the model accuracy. The reduction in model performance is indeed one of the major drawbacks of this method, as it can make the model much less relevant in practice. We analyze in Chapter 6 one particular approach of training models with differential privacy based on Langevin diffusion and show that it provides promising trade-offs for simple tasks.

Confidentiality can be further increased by combining federated learning and differential privacy. For example, differential privacy can be used in the local training phase by each center. Returning to our healthcare data example, if the laboratory actually wants to re-use the model outside of the hospital centers that participated in the training, combining the two methods ensures that the training data cannot be compromised, either directly or indirectly by exploiting the neural network. One last point remains to be addressed: let us imagine that the laboratory does not wish to make the model visible to the institutions that contribute to training, for example for intellectual property reasons, or because it has already been pre-trained elsewhere and represents a strategic product on which it wishes to retain exclusivity. In this case, the contextual integrity of the model may be compromised: we would therefore like to be able to encrypt the model before sharing it with the parties and do the training in a fully encrypted way.

1.1.4 Encrypted Computation

There are three main techniques for computing over encrypted data that we have explored throughout this thesis and particularly in Chapter 5: homomorphic encryption, functional encryption and multi-party computation. Let's take a quick look at each of these techniques to illustrate how they work.

Homomorphic encryption is based on computational security, meaning that decrypting ciphertexts (i.e. the objects that encode the secrets) from this protocol would be equivalent to solving a cryptographic problem which is known to be difficult or intractable. The Paillier cryptosystem [Pai99] for example, proposed in 1999 by the cryptographer Pascal Paillier, is based on the difficulty of factoring a number N formed by the product of two large prime numbers p and q. It is additively homomorphic, which means that it is possible to add two private values only by manipulating their ciphertexts without learning anything about the underlying values. Evaluation of circuits including an arbitrary number of additions and of multiplications was first proposed by Craig Gentry [Gen09]. This fully homomorphic encryption (FHE) protocol implements circuits of unbounded length by using a bootstrapping operation. In practice, the scheme relies on noisy ciphertexts, whose noise increases at each multiplication until it becomes too large to allow decryption. Bootstrapping acts as a procedure to periodically refresh the ciphertext to remove the noise without revealing the underlying intermediate results.

The interesting characteristic about homomorphic encryption is that it allows us to delegate an entire calculation without having to intervene except for the final decryption. For example, in our example, we could imagine that the laboratory encrypts the parameters of the model and sends a copy to each hospital so that it can train it on its data. Or the other way around, if the hospital does not have the necessary computing power. Indeed, current implementations for fully homomorphic encryption remain quite costly in terms of compute time even if significant improvements such as [CGGI16] have been proposed upon the initial work [Gen09].

Second, functional encryption [BSW11] is also based on computational assumptions but it differs from homomorphic encryption since it combines the computation step with the decryption step. In other words, given an encrypted input, a function can be applied and the result automatically decrypted without requesting decryption to the input owner. This owner maintains its ownership by selecting the functions that can be applied on its input. One classic example of functional encryption is the spam filtering done by an email server. If granted the appropriate key, the server can apply a spam detection function on an encrypted email and only decrypt whether the email is likely a spam or not.

Last, multi-party computation [GMW87] consists of jointly computing a function between several parties while keeping the input of each party private. One family of multi-party computation techniques is based on secret sharing. The simplest sharing method is additive sharing and works as follows: given a value, say a salary, two completely random shares are constructed in a large space (such as the space of 64-bit encoded integers), so that their sum is equal to the initial value. Each share is distributed to a different party, which means that each party only receives a random number that reveals nothing of the secret value. To reconstruct it, the parties have to pool and add up their shares. This secret sharing method also makes it possible to calculate sums over secret values: it is simply a matter of each party calculating the sum of the shares of the values it owns. It is also possible to construct more than 2 shares, in order to share the secret between an arbitrary number of parties. Thus, it would be possible to easily calculate an average of n wages without revealing any particular wages. But the more parties are involved, the more likely it is that one of the parties will leave the protocol and make reconstruction impossible. Thus, protocols that resist defection by $n - k$ of the n parties involved have been developed. For example, the cryptographer Adi Shamir proposed the so-called Shamir secret sharing [Sha79], which consists of encoding a secret as the fixed coefficient of a polynomial of degree $k - 1$, and giving each party an evaluation of the polynomial at a different point. Thanks to this construction, k parts are sufficient to reconstruct the polynomial by Lagrangian interpolation and thus to identify its fixed coefficient. This protocol is detailed in the preliminary chapter 2.

Multiplication and comparison operators are crucial for modeling neural networks. However, they require the different parties to communicate with each other, making the number of exchanges needed an important criterion when choosing a secret sharing protocol. In this thesis, we have explored function secret sharing protocols that were based on the work of [BGI15]. Function secret sharing is a protocol that can be used to drastically reduce the number of interactions for private comparison. Indeed, we can accelerate the calculations using specialized hardware such as GPUs but we are limited in the exchanges by the speed of light which means that data will never transit between Paris and San Francisco in less than 30ms. In practice, the latency is even 150ms, which means that each interaction eliminated offers a directly measurable gain. Chapter 5 presents AriaNN, a framework that leverages function secret sharing to implement comparison with a single interaction, at the expense of larger computations that are accelerated using GPUs.

The choice of the cryptographic protocols to use is therefore once again a function of the context: it is not useful to apply excessive constraints if the risks can legitimately be considered limited, especially since the extra cost associated with certain protections may be prohibitive for learning tasks that are already computationally intensive in clear text. Finally, encryption can be used in isolation or as a complement to other techniques, and Chapter 6 illustrates how differential privacy and multi-party computation can be used in synergy.

However, despite all these available techniques, protecting data cannot be an end in itself, as it would then be in opposition to any scientific progress. Data protection must be exercised as a means to share data and to strengthen collaboration between all actors. Helen Nissenbaum's approach of analyzing privacy through the notion of contextual integrity is not the only way to address this issue, but she defends the vision that the development of privacy-preserving methods will not lead to a society of secrecy but to a society of cooperation.

1.2 Contributions

Following this goal of exploring several approaches to implementing privacy in machine learning, we have included in this thesis results that belong to quite different fields, but we expect that the prism of contextual integrity sheds light on how all these domains are facets of the same solution.

More specifically, the results in this manuscript are mainly issued from the following papers, which are already published or still in the reviewing process.

***PySyft: A Library for Easy Federated Learning* [ZTL+21].**

This article is more precisely a chapter of the Springer book "Federated Learning Systems" published in 2021. It builds upon two preliminary articles [RTD+18] and [HJC+21], respectively presented at the NeurIPS 2018 workshop on Privacy-Preserving Machine Learning and the ICLR 2021 workshop on Distributed and Private Machine Learning. In this paper, we provide insights on the implementation principles we chose when building the PySyft library, to be able to simultaneously use federated learning with encrypted computation and differential privacy. We propose a modular architecture centered around the notion of tensor and show how external open-source contributors can easily integrate future protocols into this library.

***Partially encrypted machine learning using functional encryption* [RPB+19].**

In this article, we propose a practical framework to perform partially encrypted and privacy-preserving inference which combines adversarial training and functional encryption.

We first present a new functional encryption scheme to efficiently compute quadratic functions so that the data owner controls what can be computed but is not involved in the calculation: it provides a decryption key which allows one to learn a specific function evaluation of some encrypted data. We show how to use it in machine learning to partially encrypt neural networks with quadratic activation functions and perform inference. We provide an analysis of the information leaks by studying indistinguishability of data items of the same label. To this aim, we introduce a new image dataset made of distorted print characters that can either be classified by font or by letter.

Second, we propose a training method to prevent selected sensitive features from leaking, which adversarially optimizes the network against an adversary trying to identify these features. Several existing works using partially encrypted machine learning cannot deal with the last thresholding operation used for classification, which could significantly reduce the leakage. For these works, this approach can be very interesting as it comes with little reduction on the model's accuracy and significantly improves data privacy.

This paper has been published in the proceedings of the NeurIPS conference in 2019.

***AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing* [RTPB22].**

In this article, we propose AriaNN, which is a framework for private neural network training and inference on sensitive data, whose particularity is to require very few interactions thanks to a new protocol for implementing comparison.

This protocol is semi-honest and involves 2 parties with a trusted dealer. It builds upon [BGI15] and leverages function secret sharing, a lightweight cryptographic protocol that allows for an efficient online phase. We design optimized primitives for the building blocks of neural networks such as ReLU, MaxPool and BatchNorm. For instance, we perform private comparison for ReLU operations with a single message of the size of the input during the online phase, and with preprocessing keys close to 4× smaller than previous work [BGI16].

We implement our framework in open-source, as an extensible system on top of PyTorch that leverages CPU and GPU hardware acceleration for cryptographic and machine learning operations. We evaluate our end-to-end system for private inference between distant servers on standard neural networks such as AlexNet, VGG16 or ResNet18, and for private training on smaller networks like LeNet. We show that computation rather than communication is the main bottleneck and that using GPUs together with reduced key size is a promising solution to overcome this barrier.

This paper has been published in the proceedings of the PoPETS 2022 conference.

Differential Privacy Guarantees for Stochastic Gradient Langevin Dynamics **[RBP22].** In this paper, we analyze the privacy leakage of noisy stochastic gradient descent by modeling Rényi divergence dynamics with Langevin diffusions.

Inspired by the recent work on non-stochastic gradient descent [CYS21], we derive similar desirable properties in the stochastic setting. In particular, we prove that the privacy loss converges exponentially fast for smooth and strongly convex objectives under constant step size, which is a very interesting property compared to previous DP-SGD analyses [ACG+16]. We also extend our analysis to arbitrary sequences of varying step sizes and derive new utility bounds.

We propose an open-source implementation of this protocol and our experiments on various datasets show the practical utility of our approach compared to classical DP-SGD libraries.

1.3 Outline

This thesis is organized into seven chapters as follows:

Chapter 1 is the present introduction.

Chapter 2 is a preliminary chapter. It introduces core concepts informally presented in this introduction, together with notations that will be used in the next chapters.

Chapter 3 presents our work on building a modular open-source library to implement privacy-preserving techniques for machine learning. It is based on the papers [RTD+18, HJC+21, ZTL+21]

Chapter 4 proposes a new attack against trained models which consists of exploiting the output signal to infer private attributes of the input data. It is based on the second part of the paper [RPB+19].

Chapter 5 is divided in two parts. First we present and implement a quadratic functional encryption scheme which can be used for partially encrypted machine learning inference. Second, we provide a new framework based on function secret sharing for private inference and training. They are respectively based on the first part of [RPB+19] and on [RTPB22].

Chapter 6 explores synergies between differential privacy and private computation protocols like the ones presented in the previous chapter. It proposes a new protocol for differentially private stochastic gradient descent. It is based on the paper [RBP22].

Chapter 7 comes back to the notion of contextual integrity and discusses how the techniques discussed in this thesis could apply to the domain of healthcare. It concludes the manuscript with some open questions.

Chapter

2 Preliminaries

Chapter content

This preliminary chapter aims to fix the notations and to recall the basic notions we will use throughout this thesis.

2.1 Definitions and Notations

Set: A set is a collection of elements. $a \in S$ denotes that the element a belongs to the set S. Two sets S and S' are said equal ($S = S'$) if they contain the same elements. We denote by $\mathbb{N}$ the set of non-negative integers, and by $\mathbb{Z}$ the set of integers. We denote $\mathbb{Z}_n$ the finite set of integers $\{k, k \in \mathbb{Z}, 0 \leq k < n\}$. For any finite set S, $|S|$ refers to the number of elements in the set S.

Cartesian Product: Let $S_1, \dots, S_n$ be n sets, with $n \geq 2$. For all k, x_k refers to an element of S_k. $(x_1, \dots, x_n)$ is a tuple of elements of $S_1, \dots, S_n$. We denote as $S_1 \times \cdots \times S_n$ the cartesian product of $S_1, \dots, S_n$, i.e. the set of all $(x_1, \dots, x_n)$.

Group: We denote by group $(G, *)$ a set equipped with a law of composition $*$, which satisfies the following: the law $*$ is associative and admits a neutral element, every element of G has an inverse. $(G, *)$ is said to be abelian if the law $*$ is commutative. For example, $(\mathbb{Z}, +)$ is an additive group. $(\mathbb{Z}_n, +)$ is also an additive group for the addition operation modulo n, where modulo n refers to the remainder of the Euclidean division by n.

Cyclic group: A cyclic group $(G, *)$ is a finite group which can be completely generated by composing one of its elements g by itself using the group law, namely $G = (1, g, g^2, g^{p-1})$ for some $p \in \mathbb{N}$. We say that g is a generator of G and that p is the order of the group. We write such a group $\mathbb{G}$ to indicate that it is cyclic.

Ring: We denote by ring $(G, +, \times)$ an abelian group $(G, +)$ equipped with a law $\times$ which is associative and distributive with respect to the law $+$, and admits a neutral element. For example, $(\mathbb{Z}_n, +, \times)$ is an abelian ring. With a slight abuse of notation, this ring will be referred to as $\mathbb{Z}_n$.

Bit string: By definition, $\{0,1\} = \mathbb{Z}_2$. $\{0,1\}^n$, the cartesian product of n sets $\mathbb{Z}_2$ with $n \in \mathbb{N}$, refers to the set of bit strings of length n. The bit decomposition of any element x of Z_{2^n} into a bit string of $\{0,1\}^n$ is a bijection between Z_{2^n} and $\{0,1\}^n$. In particular, we denote by $x[i]$ the i-th element of the bit decomposition of x, namely the i-th bit of x. When there is no confusion, we will use $\mathbb{Z}_{2^n}$ and $\{0,1\}^n$ interchangeably. Additionally, we denote by $\{0,1\}^*$ the set of all bit strings, which we can identify with $\mathbb{N}$ or $\mathbb{Z}$. The symbol $||$ is used to concatenate bit strings: if $x \in \{0,1\}^m$ and $y \in \{0,1\}^n$, then $x \,||\, y \in \{0,1\}^{m+n}$.

Vector, Matrix: A vector $(x_1, \ldots, x_n)$ of n elements of S is represented as $\mathbf{x}$ in bold, or sometimes x for more clarity. Similarly, a matrix of n rows and p columns is denoted $\mathbf{M} = (x_{i,j})_{\substack{i=1..n\\ j=1..p}}$. For any element g and vector $\mathbf{x}$ of n elements, we denote by $g^{\mathbf{x}}$ the vector $(g^{x_1}, \ldots, g^{x_n})$, and by $\mathbf{x} \cdot \mathbf{y}$ the inner product $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$.

Bilinear groups: Bilinear groups or pairings are used in cryptography like in functional encryption. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order p, respectively generated by g_1 and g_2, and $\mathbb{G}_T$ another cyclic group of prime order p. A pairing e is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ which is efficiently computable, non-degenerated and bilinear: $e(g_1^\alpha, g_2^\beta) = e(g_1, g_2)^{\alpha\beta}$ for any scalars $\alpha, \beta \in \mathbb{Z}_p^*$. We define $g_T := e(g_1, g_2)$ which spans the group $\mathbb{G}_T$ of prime order p.

2.2 Adversary Models

Cryptography protocols like the ones defined in this thesis describe procedures for parties to compute over private data without having access to this data. When parties follow the protocol, it is guaranteed that the private data is not revealed. However, for some protocols, parties that deviate from the protocol can succeed in revealing all or part of the data. For other protocols, any deviation from the instructions can be detected and the computation aborted. The security of the protocols is therefore conditioned on some assumptions on what the adversaries can do, that we call the adversary model. We present here three main models.

Semi-honest model: The semi-honest model [Gol09, Chapter 7.2.2] refers to adversaries that follow the protocol but will try to leverage any information gained as part of the protocol. That's why we often refer to it as the honest-but-curious model. This model makes the strongest assumptions on the adversaries, and there are situations where such assumptions may not hold in practice. However, it allows for more lightweight and efficient cryptography, and can be a first step towards designing protocols against stronger adversaries. We claim throughout this thesis that this model is valid for cooperation scenarios between institutions like public health institutions, that would in any case cooperate even without cryptography protocols and for which such semi-honest protocol represent already a decisive step forward.

Malicious model: In this model [Gol09, Chapter 7.2.3], the adversaries use any strategy available to gain insight on the private data, and may deviate arbitrarily from the protocol. Such a model often requires heavier cryptography primitives and results in a longer runtime. When several parties are involved in the protocol, like in the field of secure multi-party computation, we often consider that only a subset of the adversary are malicious and that the others are semi-honest, hence defining notions like honest-majority malicious secure protocols. In particular, a line of work [MR18, WTB+21] has developed protocols that can transition from semi-honest to honest-majority malicious security with limited impact on the runtime, which suggests that malicious security for privacy-preserving machine learning is becoming practical.

Covert security model: This model [AL10] is a compromise between the two others. Informally, it means that parties that deviate from the protocol can be caught with some non-negligible probability. This notion is particularly interesting for parties that can be associated with real world entities, and therefore care about their reputation: they may not want to take the risk of having their reputation damaged. We use this notion in this thesis for the third parties that provide cryptography primitives: it is usually simple for semi-honest parties to check that the primitives were generated correctly by a third party, and this party has a great interest in remaining trustworthy.

2.3 Functional Encryption

This section contains several notions of security and cryptography that are used primarily in our results in functional encryption in chapter 5.

2.3.1 Generic Notions

Probabilistic Polynomial-Time Algorithms. A probabilistic polynomial-time algorithm (or PPT algorithm in short) is a probabilistic algorithm, namely an algorithm that has access to some source of randomness, and that runs in time polynomial with respect to its input size. This notion of PPT algorithm is used to refer to an efficient algorithm, compared to an algorithm that would run for instance in time exponential compared to its input length, like a brute-force attack to uncover a password.

Security parameter. The security parameter denoted here by λ is a measure of the difficulty for an adversary to break a scheme. For example, a secret key encoded on a small number of bits could easily be guessed by an adversary while recovering a key encoded on say 128 bits is intractable for modern computers. Unless specified otherwise, we will use $\lambda = 128$. This parameter is often provided as input to algorithms as 1^λ. A PPT algorithm that take as input 1^λ and output a secret key is therefore expected to run in time polynomial to λ while outputting a key that would need $O(2^\lambda)$ iterations to break by an adversary.

Public-Key Encryption Scheme. A public-key encryption scheme is a composed of the following three PPT algorithms:

- $\mathsf{KeyGen}(1^\lambda)$: the key generation algorithm takes as input the security parameter λ and outputs two keys, the public key pk and the private key sk.

- $\mathsf{Enc}(\mathsf{pk}, m)$: the encryption algorithm uses the public key to convert a message m into a ciphertext c.

- $\mathsf{Dec}(\mathsf{sk}, c)$: the decryption algorithm uses the private key to decrypt a ciphertext c and recover the original message m

A correct public-key encryption scheme should satisfy that for any $(\mathsf{pk}, \mathsf{sk})$ output by KeyGen, and for any message m, we have $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$ except with negligible probability. Such scheme is widely used in practice because parties willing to exchange information only have to publicly share their pk instead of sharing a secret key in advance to the trusted recipients.

Example. The El Gamal Encryption Scheme [ElG85] is an example of public-key encryption scheme. Given a cyclic group $\mathbb{G}$ of prime order p, it is defined as follows:

- $\mathsf{KeyGen}(1^\lambda)$: Randomly choose $s \xleftarrow{\$} \mathbb{Z}_p$. Output the public key $\mathsf{pk} = g^{-s}$ and the private key $\mathsf{sk} = s$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: given a message $m \in \mathbb{G}$, sample $r \xleftarrow{\$} \mathbb{Z}_p$. Output $c = (c_1, c_2) = (m \cdot \mathsf{pk}^r, g^r)$.
- $\mathsf{Dec}(\mathsf{sk}, c)$: given the ciphertext c, recover the message by computing $c_1 \cdot c_2^{\mathsf{sk}}$.

Correctness of the scheme can be verified directly:

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = c_1 \cdot c_2^{\mathsf{sk}} = m \cdot \mathsf{pk}^r \cdot g^{-r\mathsf{sk}} = m \cdot g^{rs} \cdot g^{-rs} = m$$

Security of a public-key encryption scheme can be defined under several scenarios regarding the adversary.

2.3.2 Security

First, let us define two different kinds of security that we will use throughout this thesis.

Information-theoretic security. This notion of security also known as perfect secrecy [KL14, Chapter 2.1] relies on the fact that the adversary is not given the information required to break the encryption. This means that no matter the computational power an adversary has, it is not able to break a given scheme. This is typically what is achieved with secret sharing in multi-party computation.

Computational security. [KL14, Chapter 3.1] Computational security can be seen as a relaxation of perfect secrecy: it relies on the fact that an adversary has in practice a (potentially huge but) finite amount of computational power. Hence, the security relies on the hardness of a problem that cannot be plausibly solved by an adversary that runs for a feasible amount of time. Note that in this scenario, there is a negligible but not null probability that the adversary succeeds. For example, public-key encryption schemes like El Gamal or also functional encryption schemes rely on computational security.

We define security of protocols based on different scenarios where an adversary has some specific capabilities and tries to infer information about private input data given one or several ciphertexts. The argument of security relies on the fact that if a ciphertext is indistinguishable from random, then it does not disclose sensitive information. This is formalized using the notion of *advantage*, which informally means the advantage that an adversary has in distinguishing a ciphertext from randomness.

IND-CPA. IND-CPA [KL14, Chapter 3.4.2] stands for indistinguishability under chosen-plaintext attacks. It defines the following scenario or *game* between a PPT adversary and a challenger: the adversary can perform a polynomial number (in λ) of encryptions, and then chooses two plaintext different messages m_0 and m_1 and gives them to the challenger. The challenger chooses m_b with $b \xleftarrow{\$} \{0,1\}$ and returns $\mathsf{Enc}(\mathsf{pk}, m_b)$. The adversary can perform more encryptions (of messages distinct from m_0 and m_1) and then outputs a guess for the value of b. We measure its success against pure luck by defining the following advantage.

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND-CPA}}(1^\lambda) := \Pr\left[b' = b : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{Enc}}(c) \end{array} \right] - \frac{1}{2}$$

We say a schema is IND-CPA secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}(1^\lambda) = \mathrm{negl}(\lambda)$.

IND-CCA. IND-CCA [KL14, Chapter 3.7.1] stands for indistinguishability under chosen ciphertext attack. IND-CCA is a stronger notion of security than IND-CPA, as it provides an additional capability to the adversary. Indeed, the adversary is also given the ability to ask for decryption of any ciphertext, either up to the release of the challenge (IND-CCA1) or even after c is returned by the challenger (IND-CCA2), provided that decryption on c is not performed.

Similarly, we can define the following advantage for IND-CCA1:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND-CCA}}(1^\lambda) := \Pr\left[b' = b : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Dec}}(\mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{Enc}}(c) \end{array} \right] - \frac{1}{2}$$

Back to our example with the El Gamal encryption scheme, we will prove that it is IND-CPA secure. But such (computational) security can only be proven under some computational assumptions, that we use to state that breaking the schema would be equivalent to solving a problem that is believed to be difficult.

2.3.3 Computational Assumptions

We present here some computational assumptions that are useful for this thesis, but many others exist, including those based on integer factorization.

The Discrete Log assumption. This is a very standard computational assumption [KL14, Chapter 8.3.2] over the computation of discrete logarithms in cyclic groups. It says that given a generic cyclic group $\mathbb{G}$ of order p with a generator g, and given an element $h \in \mathbb{G}$, it is computationally intractable to find $x \in \mathbb{Z}_p$ such that $h = g^x$. The bigger the order p is, the more complex it is to solve this problem. Best known approaches for generic cyclic groups are computed in time $O(\sqrt{p})$ [Pol78], so p should be chosen to verify $\lambda \sim \log(p)$.

The Decisional Diffie–Hellman assumption (DDH). The DDH assumption [KL14, Chapter 8.3.2] is related to the discrete log assumption, and states that given $\mathbb{G}$, a cyclic group of order p (such that $\lambda \sim \log(p)$), g a generator of $\mathbb{G}$, and $a, b, c \xleftarrow{\$} \mathbb{Z}_p^3$ chosen by a challenger, the two following DDH triplets (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) are computationally indistinguishable for any PPT adversary.

We can use this assumption to prove that the El Gamal encryption scheme is IND-CPA secure. The IND-CPA game is the following: on input (m_0, m_1) provided by the adversary, the challenger chooses $b \xleftarrow{\$} \{0,1\}$ and outputs $c = (c_1, c_2) = (m_b \cdot \mathsf{pk}^r, g^r) = (m_b \cdot g^{-rs}, g^r)$. The challenger can compute ${m_0}^{-1}$ (since every element of a group admits an inverse by the composition law) and deduce $c_1 \cdot {m_0}^{-1}$ which equals g^{-rs} if and only if $b = 0$. In addition, it has access to g^r (this is c_2) and g^{-s} (this is pk). Hence, any adversary that can break the El Gamal scheme can distinguish between (g^r, g^{-s}, g^{-rs}) and (g^r, g^{-s}, g^c) for some $c \in \mathbb{Z}_p$ and hence break the DDH assumption.
We can also briefly show that the El Gamal scheme is not secure in the IND-CCA2 way. Say now that the adversary samples another message $m' \xleftarrow{\$} \mathbb{Z}_p$. It can craft a ciphertext $c' := (m' \cdot c_1, c_2)$ which is an admissible ciphertext for the message $m' \cdot m_b$ since $c' = ((m' \cdot m_b) \cdot \mathsf{pk}^r, g^r)$. It can then call the decryption oracle to recover $m' \cdot m_b$ and deduce m_b.

When using asymmetric pairings as we do for our functional encryption schemes, the DDH assumption is not a strong assumption. Indeed, using the notations of Section 2.1, distinguishing between (g_1^a, g_2^b, g_T^{ab}) and (g_1^a, g_2^b, g_T^c) is straightforward as the pairing function e can be used to compute $g_T^{ab} = e(g_1^a, g_2^b)$ and to compare it to g_T^c. Instead, we introduce a new assumption called co-DBDH.

The Decisional Co-Bilinear Diffie-Hellman assumption (co-DBDH). In the case of asymmetric pairing groups (i.e. when $\mathbb{G}_1 \neq \mathbb{G}_2$), we introduce the following assumption [BF01]. With the notations of Section 2.1, and given $(a, b, c, d) \xleftarrow{\$} \mathbb{Z}_p^4$ chosen by a challenger, the following triples $(g_1^a, g_1^b, g_2^b, g_2^c, g_T^{abc})$ and $(g_1^a, g_1^b, g_2^b, g_2^c, g_T^d)$ are computationally indistinguishable.

2.4 Machine Learning

Most of our work has consisted in developing new privacy-privacy techniques or improving existing ones to be practical in the context of machine learning. In particular, we have considered private training and evaluation of neural networks, which are commonly used for many real world applications. We recall here the main notions about neural networks.

Linear Neural layers. A neural layer [BN06, Chapter 5.1] is a function which acts on a signal based on its internal parameters (or weights) and outputs a modified signal. The most simple layer is the feed forward layer and is defined as such:

$$\begin{aligned} f &: \mathbf{x} \mapsto \mathbf{W}\mathbf{x} + \mathbf{b} \\ f &: \mathbb{R}^m \to \mathbb{R}^n \\ \mathbf{W} &\in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^n \end{aligned}$$

This layer can for example implement linear regression, but more complex layers exist like convolution and batch normalization. Convolution is mostly used for vision tasks and consists of extracting local features of an image. Batch normalization normalizes the signal and is particularly useful for very deep networks. Other layers like the ones used for text analysis are not described here.

Pooling layers. Pooling layers [LBBH98] do not have internal state: their role is to reduce the dimension of a signal by averaging or taking the maximum of neighbor components and hence outputting a signal with a smaller dimension. This is very helpful to reduce the complexity of a neural network.

Activation functions. An activation function [BN06, Chapter 5.1] usually transforms the signal to introduce a non-linearity and does not necessarily have internal weights. It is often combined with a neural layer like the feed forward or the convolutional layer. A popular activation function is the sigmoid function or the ReLU function, which is defined as such:

$$\begin{aligned} \sigma &: \mathbf{x} \mapsto \max(\mathbf{x}, \mathbf{0}) \\ \sigma &: \mathbb{R}^m \to \mathbb{R}^m \end{aligned}$$

In some work in privacy preserving machine learning, we sometimes also use the square function $\sigma : \mathbf{x} \mapsto \mathbf{x}^2$, which also introduces a non-linearity while being significantly simpler to compute than a sigmoid or a maximum function with the current cryptographic protocols. We also introduce the softmax activation function:

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} \ , \ \forall i = 1, \ldots, m$$

Neural Network. A neural network is a sequence of neural layers, pooling layers and activation functions. It can be composed of a single layer or dozens of them like ResNet152 [HZRS16] which is composed of 152 layers, depending on the complexity of the task considered. Neural networks are commonly used for regression and classification, and most of the experiments conducted throughout this thesis address classification tasks. Classification between C classes means in practice the neural network takes as input a sample $\mathbf{x}$ and outputs $\mathbf{y} \in \mathbb{R}^C$ where each component of $\mathbf{y}$ corresponds to a class. A softmax activation function is often added on top of this output to normalize the signal and squeeze it between 0 and 1, so that each component y_i can be viewed as the probability that $\mathbf{x}$ belongs to class i.

Back propagation. [**BN06, Chapter 5.3**] A neural network is trained by adapting the weights of its layers so that the output signal $\mathbf{y}$ corresponds to the expected one $\hat{\mathbf{y}}$. This is done by letting the network compute an output for some training samples and comparing it to the expected output using a loss function $\mathcal{L}$ like the mean square error. Then, the error observed is differentiated with respect to all the weights of the network to analyze which layer contributed most to the error. Given the feed forward structure of the network, the successive derivatives are computed using the chain rule, starting backward from the output layer. This is why this step is called back propagation of the gradients. Then, each weight is updated using the gradient descent method or a variant of it, which basically consists of updating the weight in the direction opposed to its contribution to the error:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$$

2.5 Multi-Party Computation

We now switch to notions that will be helpful to introduce our results in multi-party computation.

2.5.1 Generic Notions

One-Way Function. [**KL14, Chapter 7.1**] A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is a one-way function if it satisfies two key properties:

- For any $x \in \{0,1\}^*$, $f(x)$ can be efficiently computed, namely in time polynomial with respect to the input size.
- Given x chosen uniformly over $\{0,1\}^*$ and $y = f(x)$, finding a x' such that $f(x') = y$ is computationally infeasible for any PPT adversary.

One-way functions are used to prove the existence of pseudorandom generators and pseudorandom functions that we will use in our construction in function secret sharing in chapter 5.

Pseudorandom generator. [**KL14, Chapter 7.4**] Pseudorandomness is often seen as a computational relaxation of pure randomness in the same sense that indistinguishability is a relaxation of perfect secrecy: a PPT adversary is not able to distinguish between them. More formally, a pseudorandom generator $G : \{0,1\}^\lambda \mapsto \{0,1\}^\nu$ is a deterministic polynomial time algorithm, if for any PPT distinguisher D, i.e. a PPT adversary that returns 1 when it detects that the input is random and 0 otherwise, the following holds:

$$\left| \Pr\left[D(G(s)) = 1\right] - \Pr\left[D(r) = 1\right] \right| \leq \text{negl}(\lambda),$$

where probabilities are respectively taken uniformly over s and r. In addition, it is expected that the output bit string should be longer than the input, meaning $\nu > \lambda$.

Pseudorandom function. [KL14, Chapter 7.5] In a similar way, we can define keyed pseudorandom functions. A function $F : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \mapsto \{0,1\}^{n_3}$, where the first input space stands for the key and the second for the actual input, is said to be a keyed pseudorandom function if for any PPT distinguisher, the following holds:

$$|\Pr[D^{F(k,\cdot)}(1^\lambda) = 1] - \Pr[D^{f(\cdot)}(1^\lambda) = 1]| \leq \mathrm{negl}(\lambda),$$

where the probabilities are respectively taken uniformly for k on the key space and for f on the space of function whose input space is $\{0,1\}^{n_2}$ and output space is in $\{0,1\}^{n_3}$.

2.5.2 Secret Sharing

Secret sharing [KL14, Chapter 13.3.1] is widely used in multi-party computation to split and distribute a private input across several (say n) parties that are involved in some protocol. The general idea consists of building n shares from a secret x, such that t (with $t \leq n$) shares are required to reconstruct x. These shares should be distributed among the parties involved such that no party can recover the secret alone. As a consequence, no party should hold more than $t-1$ shares.

From now on, we will assume that private inputs and shares are defined in the ring $\mathbb{Z}_k$, with k typically equal to 32 or 64. This allows in practice to benefit from hardware optimization for respectively `int` and `long` types.

Additive Secret Sharing. The most common type of sharing is additive secret sharing. Given a secret $x \in \mathbb{Z}_k$, n are chosen at random such that $\sum_{i=1..n} x_i = n \mod 2^k$. In this situation, the view of any subset of shares $\{x_i, i \in J, J \subset [1,n], |J| < n\}$ provides no information at all about x, since each share is perfectly indistinguishable from random (in the information theoretic way). Note also that all arithmetic operations should be taken modulo 2^k. When each party is given exactly one share, we achieve n out of n secret-sharing, meaning that all parties should agree to recover a secret, which corresponds to fully shared governance. However, when the number of parties is high, it is to be expected that one party might drop from the computation, say a mobile phone that runs out of battery or goes offline. Hence, it is also possible to distribute more than one share to each party. This process is referred to as replicated secret sharing since $n-t+1$ copies are made for each share. It allows $n-t$ parties to drop from the computation without aborting the protocol. As a consequence, this implements t out of n sharing since t parties alone are guaranteed to have at least one copy of each share. In addition, such a replicated secret sharing scheme is very practical to achieve malicious security since several parties that share the same share can verify between them that no one is deviating from the protocol.

Shamir Secret Sharing. The Shamir Secret Sharing scheme [Sha79] is an interesting and classic k out of n secret sharing scheme that relies on polynomial interpolation. It is known that any polynomial P of degree $d \leq k-1$ can completely described by at most k evaluations of the polynomial on distinct inputs $(x_1, \ldots, x_k)$, using Lagrange interpolation for example:

$$\forall x, P(x) = \sum_{i=1}^{k} P(x_i) \prod_{j \neq i} \frac{x - x_i}{x_j - x_i}$$

Hence, given P, n shares can be made by evaluating the polynomial on n distinct inputs, and each party should receive exactly one share. The private value can be encoded as any of the polynomial coefficients, or even several values can be encoded depending on the convention chosen.

Both of these schemes are additively homomorphic, which mean that values secret shared can still be manipulated to compute additive operations without having to reveal them. Indeed, in the first scheme for example, if each party has received a share of some input x and a share of another input y, then by summing those two shares, each party will create a share of the value $x+y$.

We will denote by $\llbracket x \rrbracket^{\mathcal{P}^n}$ a private input x secret shared using a scheme $\mathcal{P}$ across n parties, or just $\llbracket x \rrbracket$ when there is no risk of confusion. The i-th share of $\llbracket x \rrbracket$ will be referred to as with $\llbracket x \rrbracket_i$. Hence, for n out of n additive secret sharing, that we use for example for our results in function secret sharing, $\llbracket x \rrbracket$ can be viewed as a tuple $(\llbracket x \rrbracket_0, \dots, \llbracket x \rrbracket_{n-1})$ distributed across parties. In addition, using this notation the homomorphic property writes $\llbracket x + y \rrbracket = \llbracket x \rrbracket + \llbracket y \rrbracket$.

In practice, most protocols only involve 2 up to 4 parties, because it reduces the number of interactions needed, especially when broadcasting some information to all parties, while offering already efficient schemes with various levels of security.

2.5.3 Preprocessing Model

The preprocessing model consists of pre-computing generic cryptographic material ahead of the interaction between the parties to make it more efficient. As a consequence, the protocol is divided into two sequences. First, the offline (or preprocessing) phase, where cryptographic primitives that are independent from the future inputs but are generally computation intensive, are generated, possibly exchanged, and stored. Second, the online phase, where the inputs for the computation are made available and where the cryptographic primitives are consumed to realize a given protocol. The online phase is generally the one where there are most constraints of efficiency, as it corresponds to the situation where one or several parties have provided their inputs and are waiting for the protocol to output the result, for example a physician sending an encrypted X-ray and waiting for a pre-diagnosis result. Therefore, offloading as much computation as possible in the offline phase is critical, as we show for example with our AriaNN framework in chapter 5.

In general, the preprocessing is agnostic of the input value but is circuit dependent [BGI19]. We can illustrate this with the example above and consider a scenario where a neural network is computing a prediction based on the X-ray provided, using some secret sharing protocol like SPDZ [DPSZ12] or a derived protocol. Such protocols are not only additively but also multiplicatively homomorphic and they rely on preprocessed Beaver triplets [Bea91] for multiplication. A Beaver triplet is a random triplet $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ which satisfies $c = a \cdot b$ and it is used when parties $(P_i)_{i=1..n}$ compute the product $\llbracket z \rrbracket = \llbracket x \cdot y \rrbracket$ from inputs $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$. Shares $\llbracket a \rrbracket$ and $\llbracket b \rrbracket$ are respectively used to mask $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$, each party P_i revealing $\llbracket x - a \rrbracket_i$ and $\llbracket y - b \rrbracket_i$ to reconstruct collectively $\delta = x - a$ and $\epsilon = y - b$. Randomness of a and b guarantees that δ and ϵ do not leak any information about x and y. Then, the parties use such δ and ϵ to compute additively $\llbracket z \rrbracket = \delta \cdot \llbracket b \rrbracket + \epsilon \cdot \llbracket a \rrbracket + \delta \cdot \epsilon + \llbracket c \rrbracket = \llbracket (x-a)b + (y-b)a + (x-a)(y-b) + ab \rrbracket = \llbracket x \cdot y \rrbracket$. Such approach with Beaver triplets also works for matrix product, provided that we consider matrix triplets $(\llbracket \mathbf{A} \rrbracket, \llbracket \mathbf{B} \rrbracket, \llbracket \mathbf{C} \rrbracket) \in (\mathbb{R}^{n_1 \times n_2}, \mathbb{R}^{n_2 \times n_3}, \mathbb{R}^{n_1 \times n_3})$, which is very convenient for neural networks to implement private fully connected layers $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$. However, the preprocessing is circuit dependent because (n_1, n_2, n_3) is directly linked to the size of the layer, and therefore the architecture of the model needs to be provided prior to the preprocessing phase, even if the weight values need not be provided.

The preprocessing material can be created jointly by the parties as it is done in [DPSZ12] or can be provided by a trusted third-party dealer, depending on the security model considered (malicious, semi-honest, or covert security).

2.6 Differential Privacy

Differential privacy [DKM+06, DR+14] is a system for publicly sharing insights derived from a dataset while keeping private information about individuals in the dataset. It provides a measurement of the privacy risk associated with publishing each particular result, by measuring the maximum leakage that each result can cause about the individuals' data. The role of differential privacy in compliance with new privacy regulations like GDPR is still being explored [CD18], alongside other privacy-enhancing techniques.

Differential privacy applied to training machine learning models is a promising technique to reduce exposure of training datasets when releasing machine learning models. The privacy leakage from these models can be quantified using either (ϵ, δ)-differential privacy [DR+14] or (α, ε)-Rényi differential privacy [Mir17], which are described below. The standard procedure consists of providing an upper bound of the leakage, modeled through the divergence of the distributions of two models trained on neighboring datasets, i.e. datasets that only differ in one item. The intuition behind is that a model whose behavior (in terms of distribution) is sensitive to the presence or absence of a single individual is likely to memorize information about specific individuals of the training set.

More formally, let $\mathcal{D} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in \mathbb{R}^{m \times n}$ and $\mathcal{D}' = (\mathbf{x}'_1, \ldots, \mathbf{x}'_n) \in \mathbb{R}^{m \times n}$ be two neighbouring datasets, meaning that there exists exactly one index i_0 such that $\mathbf{x}_{i_0} \neq \mathbf{x}'_{i_0}$.

Definition 1 ((ϵ, δ)-differential privacy)**.** A randomized algorithm $\mathcal{A} : \mathcal{D} \to \mathbb{R}^d$ satisfies (ϵ, δ)-differential privacy if for any neighboring datasets $\mathcal{D}$ and $\mathcal{D}'$, i.e. datasets that only differ in one item, and any subset $S \subset \mathbb{R}^d$, the distribution of $\mathcal{A}$ satisfies:

$$\mathrm{P}[\mathcal{A}(\mathcal{D}) \in S] \leq e^{\epsilon} \mathrm{P}[\mathcal{A}(\mathcal{D}') \in S] + \delta$$

An alternative notion, coined as (α, ε)-Rényi differential privacy has been proposed by [Mir17], which is more suited to studying composition mechanisms, but can be converted back to standard (ϵ, δ)-differential privacy.

Definition 2 (Rényi differential privacy)**.** A randomized algorithm $\mathcal{A} : \mathcal{D} \to \mathbb{R}^d$ satisfies (α, ε)-Rényi differential privacy if for any neighboring datasets $\mathcal{D}$ and $\mathcal{D}'$, the Rényi divergence of order $\alpha > 1$ satisfies $R_\alpha(\mathcal{A}(\mathcal{D}) \mid\mid \mathcal{A}(\mathcal{D}')) \leq \varepsilon$, where:

$$R_\alpha(\mathcal{A}(\mathcal{D}) \mid\mid \mathcal{A}(\mathcal{D}')) = \frac{1}{\alpha - 1} \log \mathop{\mathbb{E}}_{\theta \sim \mathcal{A}(\mathcal{D}')} \left[\left(\frac{\mu_{\mathcal{A}(\mathcal{D})}(\theta)}{\mu_{\mathcal{A}(\mathcal{D}')}(\theta)} \right)^{\alpha} \right]$$

and where $\mu_{\mathcal{A}}$ denotes the density $\mathcal{A}$.

Conversion from Rényi differential privacy to (ϵ, δ)-differential privacy is given by the following proposition:

Proposition 3 ([Mir17] From Rényi to (ϵ, δ)-differential privacy)**.** *If $\mathcal{A}$ satisfies (α, ε)-Rényi differential privacy, it also satisfies (ϵ, δ)-differential privacy for any $0 < \delta < 1$ with*

$$\epsilon = \varepsilon + \frac{\log(1/\delta)}{\alpha - 1}$$

Differential privacy often relies on the notion of *sensitivity* (or global sensitivity). The sensitivity of a function corresponds to the delta in its output when the input is modified. More formally:

Definition 4 (Sensitivity)**.** For a function $f : \mathcal{D} \to \mathbb{R}^d$, the global sensitivity of f is defined as follows:

$$S(f) = \max_{\mathcal{D},\mathcal{D}' : d(\mathcal{D},\mathcal{D}') \leq 1} \|f(x) - f(x')\|_2$$

where $d(\mathcal{D}, \mathcal{D}') \leq 1$ means that $\mathcal{D}$ and $\mathcal{D}'$ differ by at most one element and hence that they are neighbours.

In practice, differential privacy works through the addition of a controlled amount of statistical noise to obscure the contributions from each individual in the dataset in the global result. In machine learning, the most standard approaches to training neural networks with differential privacy are derived from [ACG+16]'s method of differentially private stochastic gradient descent (DP-SGD), which is described in Algorithm 1. DP-SGD is an attractive method as it closely mimics classic SGD training of neural networks and applies to almost all architectures. It therefore enjoys easy adoption from data scientists: at each batch update, Gaussian noise is carefully added to the gradient update to hide the contribution of the batch data items. This means that virtually any neural network can be trained using DP-SGD[1] and DP-SGD has been integrated in popular libraries like Opacus [YSS+21] and TensorFlow Privacy.

```
Input: Dataset $\mathcal{D} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, loss function $\ell$, step size $\{\eta_k\}_{k\geq 0}$, noise variance $\sigma^2$,
       gradient norm bound $C$ and initial random parameter $\theta_0 \in \mathcal{C}$
Output: $\theta_K$
for $k = 0, \ldots, K-1$ do
    Sample batch $\mathcal{B}_k$ of size $m$ from $\mathcal{D}$ with replacement
    Compute and clip $\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) = \frac{1}{m}\sum_{\mathbf{x}\in\mathcal{B}_k} \nabla\ell(\theta_k, \mathbf{x}) / \max\left(1, \frac{\|\nabla\ell(\theta_k,\mathbf{x})\|_2}{C}\right)$
    Add noise and update $\theta_{k+1} = \theta_k - \eta_k\left(\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}_d)\right)$
return $\theta_K$
```

Algorithm 1: [ACG+16] $\mathcal{A}_{\text{DP-SGD}}$: Differentially Private Stochastic Gradient Descent

As DP-SGD is a succession of differentially private steps, the differential privacy estimation of the whole training mechanism needs some composition rules to leverage each step. We recall here the notion of adaptive composition in differential privacy and two major composition theorems attached:

Definition 5 (Adaptive Composition)**.** A k-fold adaptive composition is a sequence of mechanisms $m_1, \ldots, m_k$ such that for $i \in [1, k]$:

- m_i can adapt to the execution of the previous mechanisms
- m_i takes as input not only the input dataset but also the output of the previous mechanisms.

Theorem 6 ([DRV10] Simple adaptive composition)**.** *Consider $K \in \mathbb{N}$ mechanisms, each of them satisfying (ϵ_k, δ_k)-differential privacy with $\epsilon_k > 0$, $\delta \geq 0$, then the adaptive composition of these mechanisms satisfy $(\tilde{\epsilon}, \tilde{\delta})$-differential privacy, where*

$$\tilde{\epsilon} = \sum_{k=1}^{K} \epsilon_k \qquad \tilde{\delta} = \sum_{k=1}^{K} \delta_k$$

[1] There are some slight adjustments to be made for some architectures, as for example BatchNorm is usually replaced with GroupNorm

Theorem 7 ([DRV10] Strong composition)**.** *For $\epsilon > 0$, $\delta \geq 0$, $\delta' > 0$, and $K \in \mathbb{N}$, the class of (ϵ, δ)-differentially private mechanisms is $(\tilde{\epsilon}, \tilde{\delta})$-differentially private under K-fold adaptative composition, where*

$$\tilde{\epsilon} = \sqrt{2K \log \frac{1}{\delta'}}\epsilon + K\epsilon\frac{e^{\epsilon} - 1}{e^{\epsilon} + 1} \qquad \tilde{\delta} = k\delta + \delta'$$

This states that the privacy leakage modeled through standard (ϵ, δ)-differential privacy grows approximately in $\sqrt{K}$ for high privacy regimes (i.e. when $\epsilon \ll 1$), where K is the number of iterations, which represents a significant improvement over the bound of the simple adaptive composition theorem.

Chapter

3 Initial Building Blocks

One motivation for this thesis was to provide the machine learning community with new practical tools to help it transition towards more privacy-preserving approaches when dealing with sensitive data. To this view, the main challenge was to provide libraries that fit in the data scientist toolbox while providing strong privacy guarantees. This chapter sheds the light on how we tried to address this challenge through the development of the open-source PySyft library as part of a global initiative of the OpenMined community. It is based on two implementation papers and one book chapter: [RTD+18] presented at the NeurIPS workshop on Privacy-Preserving Machine Learning in 2018, [HJC+21] presented the ICLR workshop on Distributed and Private Machine Learning in 2021, and [ZTL+21] which is part of the Springer book "Federated Learning Systems" published in 2021.

These papers can be seen as white papers, they represent above all the work of a community in its entirety, despite the fact that we were first author on the first article and co-authors on the two others. In this perspective, we reuse indistinguishably parts of the papers that explain best the parts of the PySyft library we have contributed to.

Chapter content

3.1 PySyft Core Privacy Features

3.1.1 Background

Modern machine learning models require large datasets to achieve state of the art performance, and even larger datasets to validate that trained models are fair, unbiased and robust. However, most real world datasets include confidential or private data which needs to be protected for regulatory, contractual, or ethical reasons. Additionally, much of this data is generated and stored in a decentralized fashion, for example on edge computing hardware, such as mobile phones or wearable health tracking devices.

Research using large, private and decentralized datasets requires novel technical solutions so that this data can be used securely. These technical solutions must enable training on data which is neither locally available nor directly accessible without leaking that data.

Decentralized computing techniques collectively referred to as federated learning (FL) allow training on non-local data. In federated learning, training is performed at the location where the data resides, and only the machine learning (ML) algorithm (or updates to it) are being transferred. Secure machine learning on the other hand represents a collection of techniques which allow ML models to be trained without direct access to the data while protecting the models themselves from theft, manipulation or misuse. Some examples of secure computation are encryption techniques such as homomorphic encryption (HE) or secure multi-party computation (SMPC). Last, differential privacy (DP) in machine learning aims to prevent trained models from inadvertently storing personally identifiable information about individuals from the dataset in the model itself.

For secure and private ML to attain widespread acceptance, the availability of well-maintained, high quality, easily deployed open-source code libraries incorporating state-of-the art techniques of the above-mentioned fields is essential. OpenMined, an open-source community, built the PySyft library to make these techniques as accessible and easy to implement as possible.

PySyft is an open-source, multi-language library which enables secure and private machine learning. PySyft aims to popularize privacy preserving techniques in machine learning by making them as accessible as possible via Python bindings and an interface reminiscent of common ML frameworks which are familiar to researchers and data scientists like PyTorch or Numpy. To ease future development in the field of privacy preserving machine learning, PySyft aims to be extensible such that new federated learning, multi-party computation, or differential privacy methods can be flexibly and simply implemented and integrated.

This chapter will introduce the methods available within the PySyft library that we have directly contributed to and will describe some implementations. We also review the use of PySyft in academic literature to date and discuss future use-cases.

3.1.2 Privacy and Federated Learning

Federated learning (FL) is a collection of computational techniques allowing the distributed training of algorithms on remote datasets. It relies on distributing copies of the learning algorithm to where the data is instead of centrally collecting the data itself. For example, in a healthcare setting, patient data can remain on the hospital's servers thus retaining data ownership while still allowing the training of algorithms on the data. FL can be set up in several ways, for example a common way is having decentralized nodes for training who send their updates back to a central server for aggregation.

There are some interesting distinctions that we can make about FL settings. First, we should distinguish between horizontal and vertical federated learning. In the horizontal federated learning setting, the dataset, seen as a collection of items $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ where each sample is a vector $\mathbf{x}_i \in \mathbb{R}^p$, is split by samples, meaning that the samples are distributed across several machines. This is the configuration of some companies that leverage data from mobile users to improve the prediction of the next word when writing messages: the samples are sentences and they are not necessarily distributed equally across several mobile phones. In this scenario, gathering more datasets allows to collect more samples and improve the model accuracy. In contrast, vertical federated learning, which is also referred to as heterogeneous federated learning, considers that the dataset is split in feature sets. Each sample $\mathbf{x}_i$ is composed of several feature sets

$\mathbf{x}_i = (\mathbf{x}_i^1, \ldots, \mathbf{x}_i^k)$, which are held by k different parties. This situation is typically encountered when a consumer profile is built up from the habits of a user across several websites that track different behaviors, or also when trying to train healthcare models on the data of a patient which is split across several software applications with different medical specialties. In this scenario, the sum of feature sets provides the big picture about the individual, allowing for more complex models to be trained.

Another interesting distinction is between model-centric and data-centric federated learning. Model-centric federated learning refers to a situation where the task is well defined and should be executed on designated data owners, as much as possible. This means that there is little flexibility on the design of the task like the type of model or the training procedure to use. This is typically what a company which runs mobile operating systems would do, defining a task and recruiting mobile phones that are available to do the training. In return, data-centric federated learning consists for a data owner to make its data available for federated tasks, should it be research studies or any kind of federated analytics. In this scenario, the data owner is more in control of its data but it is also responsible for providing high quality and well documented data, since the entity willing to perform federated learning might not know in advance how the data is structured and how to leverage it. This setting is probably the most promising in terms of perspective of global collaboration, because it holds in its core principles that data owners are empowered and well equipped to contribute their data.

As we see, these two distinctions are complementary, and a federated task like the one on the next word prediction on mobiles is both horizontal and model-centric.

Federated learning itself is not always a sufficient privacy mechanism. Deep learning models tend to unintentionally memorize aspects of their training data, i.e. storing dataset attributes in their parameters [WSZ+19]. This unintentional memorization makes federated learning settings without additional privacy preserving techniques sensitive to various attacks, such as membership inference attacks which are described more in depth in the next chapter. In addition to the various attack vectors, FL typically has substantial network transfer requirements as well as usually reduced algorithm performance compared to centralized training. Techniques for efficient training and for reducing network transfer requirements are being actively researched [KMA+19]. These concerns aside, federated learning is an important component of many privacy preserving machine learning systems.

3.1.3 Differential Privacy

Introduction

Differential privacy is a randomized system for publicly sharing meaningful information about a dataset by describing the patterns within the dataset while withholding information about individuals in the dataset.

Consider the example of a simple private database which stores the names and ages of the citizens of a small town. This dataset could be useful for various applications and help better the lives of the citizens of that town, but it is important that the database remains private and does not get into the wrong hands.

If we wanted to obtain statistics on the database publicly, such as the average, minimum and maximum age of the citizens, we need to be careful. If we report the true average, minimum or maximum, we might compromise the privacy of some specific citizens. So, we report a slightly noisy average, minimum and maximum publicly, so that no one single citizen's privacy is breached. This enables all citizens a certain degree of plausible deniability as to whether they were a part of the database.

(ϵ, δ)-differential privacy provides us with a universal privacy guarantee: given some small privacy budget modeled through ϵ and δ, we can ensure that the output of the process does not change more than a certain margin if we change or remove a single person from the database. In this perspective, differential privacy is more robust than naive anonymization since it quantifies the risk of data leakage and provides theoretical privacy guarantees within its scope of application. A formal definition of differential privacy is given 2.6.

Differentially Private Machine Learning

In the context of data-driven approaches like machine learning, differential privacy is very useful. Their objectives align, since both focus on recognizing general meaningful patterns instead of individual user data. However, it is quite challenging to generalize various DP techniques to the broad set of techniques in machine learning. Consider deep learning for example. Neural networks which are composed of a large number of layers and parameters are notoriously non-linear and can have unpredictable behaviors as illustrated by authors of [EEF+18] who show that small magnitude perturbations on the input can heavily mislead a prediction. Therefore, finding the query sensitivity for these algorithms is generally a very complex task.

However, differential privacy has also been observed to provide interesting capabilities to machine learning models, beyond increased resistance to privacy attacks like membership inference attacks. One observation is the ability of a model trained with differential privacy to generalize. It has been a surprising result that adding noise to ensure privacy actually improves the utility of the learning model, as it can generalize better on larger unseen data. Another observation is regarding the stability of the model, as adding differential privacy tends to improve the stability of the learning algorithm.

Differential Privacy in PySyft

PySyft has created an automatic differential privacy to provide the data users with strong privacy guarantees, independent of the machine learning architecture and the data itself. Automatic DP works by adding automatic query sensitivity tracking and privacy budgeting to the private tensor. The tensor is made up of a matrix of private scalar values, which are all bounded. This way, we can dynamically track the sensitivity of the query function as well as the amount of privacy budget we have spent. This helps us in employing the various techniques on making machine learning automatically differentially private with ease. Thus, PySyft is general enough to support any kind of deep learning architecture, as the key components of differential privacy are a part of PySyft's building blocks.

PySyft has also been combined with popular DP libraries like Opacus [YSS+21] which supports a large range of neural network architectures, to be able to leverage the implementation of the existing literature around the privacy budget estimation, including tools like the strong composition theorem or the moment accountants [ACG+16].

3.1.4 Secure Multi-Party Computation

Expressiveness for machine learning

Federated learning and differential privacy are not sufficient to prevent attacks against machine learning models or datasets by the participants in the training. For example it is very easy for a data owner to steal a model in classic federated learning as the model is sent in plain text.

Secure multi-party computation (SMPC) provides a framework to safe-keep both the data and the algorithms, while still permitting training and inference. It allows multiple parties to jointly perform computations over a set of inputs and to receive the resulting outputs without exposing any party's sensitive input. It thus allows models to be trained or applied to data without disclosing the training data items or the model's weights. SMPC methods implemented in PySyft rely on additive secret sharing, which is detailed in the preliminary chapter. Evaluating or training a model can be decomposed in basic computations for which SMPC protocols exist, which allows for flexible, end-to-end and secure procedures. In the case of ML, during training, the model's gradients can be secret shared, while in inference, the entire model function is secret shared.

SMPC as implemented in PySyft, which relies on semi-honest protocoles, incurs a significant communication overhead but it has the advantage that unless parties involved in the protocol are malicious or coordinate to reveal an input, the data will remain private even if sought after for unlimited time and resources. In addition, it conveys the notion of shared governance, in the sense that all (or a sufficient number of) owners of shares must agree to be able to reconstruct the secret.

As described in the preliminary chapter, additive secret sharing allows for efficient addition and multiplication. This schema appears quite simple, but it already permits numerous arithmetic and machine learning operations as combinations of additions and multiplications, like fully connected layers or convolutions. Some operations like the sigmoid can also be implemented using polynomial approximations. Boolean operations like equality tests, comparisons or the ReLU activation function receive a particular treatment, since some protocols have been developed on top of standard secret sharing to perform exact private comparison such as SecureNN [WGC19], which was first implemented in PySyft. The new protocol for efficient boolean comparison introduced in chapter 5 has later been integrated in PySyft in replacement of SecureNN.

Fixed precision encoding

These protocols are adequate for integers, covering the encryption of elements like the values of the pixels in images or the counts of entries in a database. However, the parameters of many machine learning models like neural networks are floats, so how can we use additive secret sharing in machine learning? We have introduced a new element in PySyft which allows us to approximately encode floats as integers in order to leverage the SMPC toolbox: fixed precision encoding. The intuition behind fixed precision encoding in base 10 is to fix a precision k, to round all float values up to the k-th decimal and to remove the decimal point.

Let's take an example, like 123.456. Say that we want to keep the first 2 decimals. In that case, we will store the integer 12345 and the fact that the fixed precision is set to $k = 2$ allows us to convert back when needed to the original value 123.45 up to a small approximation error. When we keep enough decimals, typically $k = 6$, we do not observe any noticeable degradation on the accuracy of the models trained in fixed precision, i.e. when the weights and the inputs are converted to fixed precision.

We can first make the following observation: the more precision you require, the smaller your range of values is. For example, if one encodes values on `int32`, only values in the range $[-2.147\,10^9, 2.147\,10^9]$ can be stored, so setting $k = 6$ with a base 10 means that only floats in the range $[-2.147\,10^3, 2.147\,10^3]$ can be encoded properly. At the same time, values smaller than 10^{-6} will be corrupted with the encoding and decoded to 0. Hence, there is a trade-off between the precision one can keep and the range of floats that can be supported.

Fixed precision is very practical for addition because one just has to sum the integers encoded and keep the same precision k. Multiplication is a bit different. Let's consider for example the multiplication of 1.2 and 2.0 encoded with a decimal precision of 2, so 120×10^{-2} and 200×10^{-2}. If we multiply the integer part, and keep the precision unchanged, we end up with 24000×10^{-2} so 240 instead of 2.4. We can either increase the precision from 2 to 4, but in the fixed precision setting we prefer the precision to remain constant (otherwise we will be limited in the number of sequential multiplication we can perform), or we can truncate the integer part, by removing the 2 least significant digits which correspond to the current precision of 2: $(24000//10^2) \times 10^{-2} = 240 \times 10^{-2} = 2.4$.

In practice, bit precision is preferred to decimal precision, i.e. rescaling numbers with powers of 2, because it is closer to the way numbers are encoded. Hence, 1.5 encoded in a bit precision of 3 would be 12×2^{-3}.

Such fixed precision encoding is directly supported in PySyft: any floating point tensor can be converted by calling the method `.fix_precision(base=..., precision=...)`.

Fixed precision encoding and additive secret sharing

In additive secret sharing, shares are not float values but integers in a sufficiently big space, for example in $\mathbb{Z}_{2^n}$, the space of the numbers encoded on n bits, with typically $n = 32$. Hence, if we want to secret share a float value, say a model parameter, we need to first convert it to fixed precision and then secret share the underlying integer value. As we recall that private machine learning using SMPC relies on decomposing model operations in basic operations, we first study how fixed precision impacts these two operations: addition and division by a public constant, which is needed for multiplication through the truncation operation.

For addition, nothing changes, each part only needs to sum its shares of each term. For division by a public constant, which is a very common operation for example when one computes a mean $\frac{1}{n}\sum_{i=1}^{n} x_i$, things get a bit more complicated. Indeed, the first idea would be to ask each party to compute the division on its share, so that when the shares are summed up, the result will be the original result divided by the public constant. We propose two examples to see how this naive method behaves:

A working public division. We consider shares to be in $\mathbb{Z}_{2^4}$ to simplify the computation. $\mathbb{Z}_{2^4}$ represents all the integers in the range $[-8, 7]$. Indeed, $2^4 = 16$ and we consider signed numbers, so instead of the range $[0, 15]$, we use $[-8, 7]$.

Let's consider $x = 4$, shared between Alice and Bob as such: $[x] = (x_{alice}, x_{bob}) = (-2, 6)$. If each party computes the division by 2 on its share, we obtain $(-1, 3)$ which reconstructs to 2 and is the correct result.

A non-working public division. Let's now consider $x = -4$, shared as such: $[x] = (x_{alice}, x_{bob}) = (6, 6)$. This sharing is correct since $6 + 6 = 12$ which maps to $12 - 16 = -4$ in $\mathbb{Z}_{2^4}$. We say that there is a *wrap around*, as we exceeded the maximum value of our space, 7. Dividing the shares by 2 like before leads to a sharing $(3, 3)$, which reconstructs to $6 \neq -4/2$. It's failing.

So what happened? Actually the shared result $(3, 3)$ is still valid, but no longer in $\mathbb{Z}_{2^4}$ but in the space of values divided by 2, namely $\mathbb{Z}_{2^3}$. Indeed, $\mathbb{Z}_{2^3}$ represents the range $[-4, 3]$, so $3 + 3 = 6$ should map to $6 - 8 = -2$ in $\mathbb{Z}_{2^3}$. However, we want to keep the same space for all the shares that we are using, so we need to find a solution.

The reason why it failed is because of the *wrap around*, which happens when we sum the shares of x but not when we sum the shares of $x/2$. It is not possible for any of the parties holding a single share of x to know if there is a wrap around, denoted by the boolean θ_x, because it would reveal information about the other parties shares. However, they can privately compute the wrap around and obtain a secret shared $[\![\theta_x]\!]$. Computing such $[\![\theta_x]\!]$ is non-trivial but protocols like [WTB+21] propose some implementations.

Having access to $[\![\theta_x]\!]$ helps to compute privately and correctly x/k in $\mathbb{Z}_{2^n}$ using the following formula:

$$[\![x/k]\!] = [\![x]\!]/k - [\![\theta_x]\!] \cdot 2^n/k$$

This means that each party divides naively its shares, and they jointly correct the result if there was a wrap around (i.e. if $\theta_x \neq 0$).

We can verify on the previous non-working example that this formula works. Indeed, we have $\theta_x = 1$ and $2^n/k = 16/2 = 8$. If we assume $[\![\theta_x]\!] = (1, 0)$, then the shares of the result would not be $(3, 3)$ but $(-5, 3)$, which sums up to -2.

As a side note, we mention the impact of numerical errors when using small fields. In the examples above, we have been using $\mathbb{Z}_{2^4}$. If we consider $x = 4$ to be shared in $\mathbb{Z}_{2^4}$ as such: $[\![x]\!] = (1, 3)$, and that we apply division per 2, even with the method involving θ_x that is 0 here, we will end up with shares (0, 1) (because 1 // 2 = 0 and 3 // 2 = 1), that sum up to $1 \neq 4/2$. The rounding error here causes the result to be completely incorrect, but for bigger spaces like $\mathbb{Z}_{2^{32}}$ that we usually consider, this error will be less important and will not significantly impact the computation.

3.2 PySyft Implementation

Since not all machine learning practitioners hold expertise in privacy and cryptography techniques, PySyft was built to make them easily available via the most popular machine learning tools that researchers and data scientists work with on a daily basis.

The central component in PySyft is the *Tensor*. Tensors usually follow standard APIs like the PyTorch or TensorFlow APIs, but also provide extra functionalities like the ability to send a tensor to a remote worker or to encode it to fix precision or to secret share it between several parties. This is done using *chains of tensors* where each tensor in the chain adds a specific capability to the chain, like the conversion of float values to fixed precision. This modularity is the key to implement federated learning, differential privacy, homomorphic encryption or secure multi-party computation protocols within the same framework. Another important concept is the concept of *Worker*: workers own tensors and are able to perform computation on them. They can communicate using *Messages* with other workers to send or get tensors, or to perform remote computations. Workers are composed of a client and a server: an interaction with a remote worker is always made through its client which defines the set of possible actions. Messages are serialized using protocol buffers which allows for cross-platform distributed execution: a worker in Python can communicate with a worker in JavaScript, Swift, Kotlin, etc. All remote computations are made through *Pointers*. The most popular pointer is the *PointerTensor* but there exists other types of pointers like the PointerDataset.

3.2.1 A Framework to Abstract Operations on Tensors

The chain structure

Performing transformations or sending tensors to other workers can be represented as a chain of operations, and each operation is embodied by a special class. To achieve this, we created an abstraction called the AbstractTensor. It gives default behaviors to Syft tensors, and implements actions that are independent from the underlying framework (PyTorch, Tensorflow, ..). All tensors living under the AbstractTensor are meant to represent a state or transformation of the data and can be chained together. The chain structure allows the transformations or states embodied by the different Syft tensors to be accessed downward using the `child` attribute and upward using the `parent` attribute.

Figure 3.1 presents the general structure of a tensor chain, where SyftTensors are a generic term for any concrete instance of AbstractTensor. All operations are first applied to a Wrapper tensor at the top of the chain which makes it possible to have the native PyTorch interface, and they are then transmitted through the chain by being forwarded to the child attribute.

Wrapper tensor is one of the two important subclasses of Tensor types. It wraps the new Syft tensor types with a native type so that the new tensor is compatible with the rest of the PyTorch or TensorFlow API. It's only role is to forward operation down to its child Syft tensor; it holds no value. This construction primarily addresses the lack of support in PyTorch and

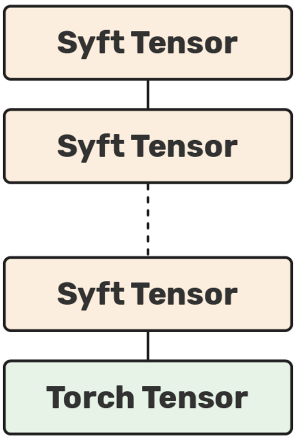

Figure 3.1: General structure of a Tensor chain. SyftTensors are a generic term for any concrete instance of AbstractTensor.

TensorFlow for creating arbitrary Tensor types (via subclassing `torch.Tensor` for instance in PyTorch). Ensuring this compatibility is critical in order for the new tensors to be usable with the existing ecosystem of layers and loss functions shipping with the native frameworks. All tensor chains begin with a Wrapper tensor, hence we did not report them on the figures.

PointerTensors are the second important class deriving from AbstractTensor. They mimic the entire API of a normal tensor, but instead of computing a tensor function locally (such as addition, subtraction, etc.) they forward the computation to another PySyft worker where the real tensor is held.

As such, PointerTensors should never exist by themselves: their role is to proxy API calls to an actual tensor located on a different worker (Virtual or actually remote) that it points to. From a data communication point of view, PointerTensors fully rely on the corresponding worker owning the target tensor to communicate the API commands it receives down the chain of tensors as shown in Figure 3.2.

From virtual to real context execution of federated learning

In order to simplify debugging complex chains of operations, PySyft develops the notion of Virtual Workers. Virtual Workers all live on the same machine and do not communicate over the network. They simply replicate the chain of commands, serialization operations, and expose the very same interface as the actual workers to communicate with each other.

Network-enabled workers in the context of Federated Learning have two implementations in the framework as of now: WebSocket workers and GridNodes. They both leverage WebSockets as their communication medium, thus ensuring a broad range of devices from IoT to web browsers to servers can participate in a PySyft network.

WebSocket workers allow multiple workers to be instantiated from within a browser, each within its own tab. This gives us another level of granularity when building federated learning applications before actually addressing remote workers which are not on the same machine. Web Socket workers are also a very good fit for the data science ecosystem revolving around browser-based notebooks.

The PyGrid project, also developed under the OpenMined umbrella, introduces its own

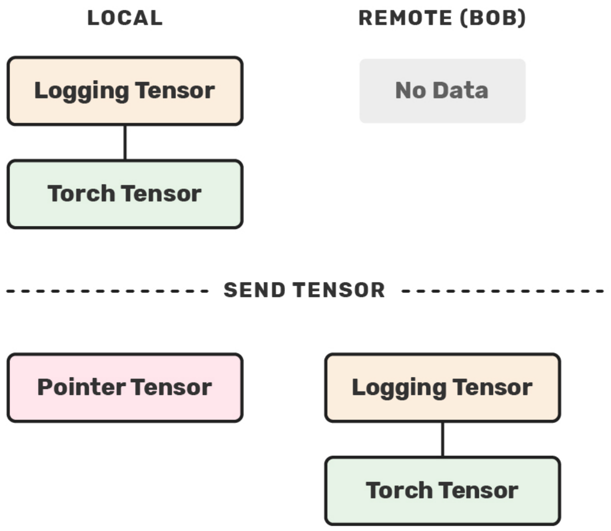

Figure 3.2: Impact of sending a tensor on the local and remote chains. The complete chain of tensors, including a `LoggingTensor` (which logs all operations requested) is sent to the remote worker and the local worker only keeps a pointer to the remote chain.

Worker known as GridNode. GridNodes communicate with their peers and clients over web socket, but contrary to Web Socket workers, they expose a traditional web socket server interface that then passes input commands and messages down to an embedded VirtualWorker.

3.2.2 Application to Additive Secret Sharing

The elements introduced in Section 3.2.1 form the building bricks necessary to create more advanced privacy-preserving tensors. The SMPC protocol detailed in Section 3.1.4 involves splitting and sending shares of an original tensor. These elements can be managed using a list of PointerTensors as described in Figure 3.3. This section describes the MPC toolbox proposed in PySyft, it provides in particular convenient abstractions over the SecureNN [WGC19] and AriaNN [RTPB22] protocols.

As explained above, additive secret sharing requires fixed precision to work on float values. Figure 3.3 shows the corresponding tensor chain associated with performing MPC with additive secret sharing. We can first observe the `FixedPrecisionTensor` which is in charge of transforming multiplications for example in a sequence of multiplication and truncation to keep the precision unchanged. This tensor also replaces some operations like the square root or the sigmoid for instance with a polynomial approximation, without any change for the end user experience. Down the chain, the operations are managed through the `AdditiveSharingTensor` that hold references through `PointerTensors` to each remote secret shares. This tensor leverages protocols like SecureNN or AriaNN to instruct remote workers on how to perform each basic operation like multiplication, public division, equality or comparisons. Last, secret shared values are actually stored on `LongTensors` i.e. 64 bits tensors, to be able to conciliate high fixed precision and a large range of possible values. `IntTensors` i.e. 32 bits tensors can also be used in specific contexts for increased computation performance.

Unlike standard MPC protocols like the ones proposed by [DPSZ12], players are not equal in

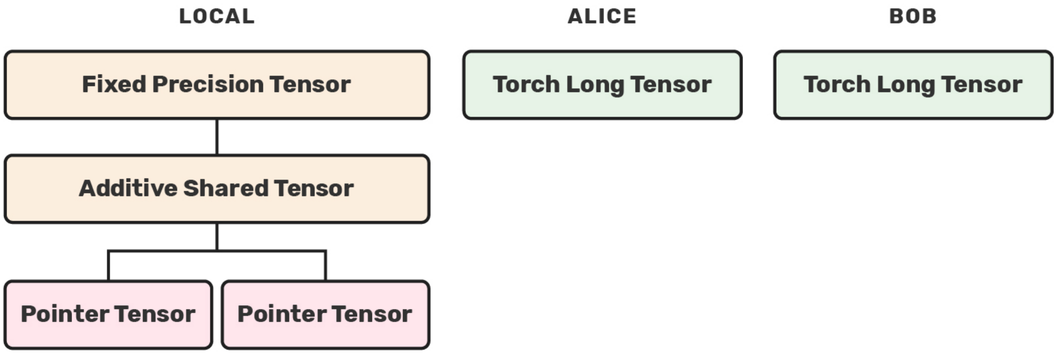

Figure 3.3: Chain structure for an additively shared tensor. Local worker holds through an `AdditiveSharingTensor` some references to secret shares owned by remote workers Alice and Bob.

our framework since one is the owner of the model (called the local worker). He acts as a leader by controlling the training procedure on all the other players (the remote workers) through the `AdditiveSharingTensor` interface. This asymmetry is aligned with the network structure of federated learning where a central node (or server) manages the training of a model on multiple nodes.

So far, the current implementation does not come with a mechanism to ensure that every player behaves honestly. An interesting improvement would be to implement MAC authentication of the secret shared value, as proposed by [DPSZ12].

3.2.3 Actions, Plans and Protocols

All operations in PySyft are called actions. Actions are categorized in two main classes. On the one hand, computation actions are actions which act on tensors and produce new tensors. For example, framework specific commands like `torch.add(x, y)` is a computation action, but syft specific command like `x.fix_precision()` is also one. On the other hand, communication actions are actions which involve moving tensors across workers. The command `pointer = x.send(alice)` which sends a tensor to a worker Alice is typically a communication action and `pointer.get()` which brings back a remote value is one as well. When a local worker wants to execute a computation action but on remote tensors, it will send a specific message to have the action computed remotely. If it wants to run n actions it will therefore need to send n messages, which can be very inefficient especially if there is some latency between the workers. Therefore, PySyft provides an object called a Plan which allows to batch together several computation actions. This Plan can be sent to any worker with a single message and be executed. One thing special about Plans is that they need to be built i.e. they need to be run once on dummy data to select and store the computation actions they will contain. However, Plans fall short when someone wants to include communication actions to ship them to other workers. That's why we introduced Protocols.

Protocols contain an arbitrary set of computation and communication actions which should involve a fixed number of workers, modeled by Roles. Like Plans, Protocols need to be built, and then are deployed across workers which all receive a Role and the actions in which this role is involved. This Protocol abstraction allows the design of complex, distributed and asynchronous computation graphs which are extremely suited for federated learning and secure multi-party computation.

More information about the implementation choices and the documentation can be found on the OpenMined GitHub[1].

3.3 PySyft Literature Review and Use Cases

PySyft is the first open-source federated learning framework for building secure and scalable ML models [YLC+19]. In recent review articles, PySyft has been described as a framework that has "made SMPC and Federated Learning intuitive and accessible to machine learning developers" [AMD20], and allows data scientists to "focus purely on building up the machine learning model without being burdened with how it will be deployed in local data repositories" [IVF19]. In a recent review [WNK20], authors have mentioned the PySyft project as an influential open-source library for encrypted computations.

To date, PySyft has been discussed as a novel and convenient approach to federated learning in several comprehensive surveys of existing literature on federated learning frameworks in healthcare, mobile edge devices, and more [QQBAF20] [SŞ20] [YLC+19] [LWH19] [LLH+20] [AMIA21] [AMD20]. In [MJ20], PySyft was mentioned for its functionality of applying differential privacy methods to federated learning and preventing the extraction of sensitive information from trained models. In this section, we summarize the use of PySyft in the literature and provide the most common use cases reported.

3.3.1 Comparisons with other Frameworks

In 2019, Li et al. [LWH19] noted the advantages of PySyft over other contemporary FL frameworks like TFF, PaddleFL and others. The authors explain that while PaddleFL provides algorithm level APIs for users to use directly, PySyft provides more detailed building blocks so that the developers can easily implement their FL processes. It is also mentioned that PySyft provides more privacy mechanisms compared to TFF, while covering all the listed features that TFF supports. Asad et al. presented in [AMIA21] a detailed comparison between different existing FL frameworks such as PySyft, TFF and LEAF based on their architecture, features and communication-efficiency. In addition, Boemer et al. in [BCD+20] compared their framework MP2ML to PySyft and reported that in terms of model privacy PySyft hides model weights but not the activation functions and model architecture which is a key feature of MP2ML. Last, [BTM+20] introduces Flower, a framework to facilitate FL research which is agnostic of the ML framework used. The main difference is that Flower is very opinionated on the computation workflow, which guarantees ease of use, while PySyft is intended to allow any remote computation, not only federated learning.

3.3.2 Use Case: Benchmarking and Standardizing Federated Learning Systems

Along with frameworks with cutting-edge tools, there are also new frameworks being introduced for benchmarking federated learning algorithms applied in different use-cases. As example, in [HLL+20] Hu et al. presented a benchmark suite for federated machine learning systems equipped with diverse data partitioning methods, cutting edge applications in image, text and structured data and use-cases with various learning complexity. They also implemented few available tools using existing FL frameworks as references to their benchmarking. They used PySyft for model training and implementing the encryption process for secure multy-party computation. On the other hand, He et al in [HLS+20] introduced an open research library called FedML for the development and benchmarking of new FL algorithms in different system environments including distributed training, mobile on-device training, and standalone simulation. Authors reported that compared to PySyft, FedML provides additional features like topology

[1] https://github.com/OpenMined/

customization, algorithm implementation of decentralized FL, FedNAS [HAA20] and vertical FL and standardized benchmarking of Deep Neural Networks. In an attempt to standardize Federated Learning based applications, Rodríguez-Barroso et al. in [RBSJL+20] proposed an open-source unified framework for federated learning and differential privacy including the standard methodology for adapting the machine learning paradigms and developing AI based services like classification and regression using federated learning and differential privacy. While analyzing different existing frameworks, they mentioned PySyft as a low level FL framework intended to be used by advanced developers with the lack of beginner-friendly support and mechanism and algorithms for differential privacy. In [RTPB22], we proposed a low-interaction SMPC protocol for private training and inference of standard deep neural networks on sensitive data based on function secret sharing. The framework offers a wide range of functions including ReLU, MaxPool and BatchNorm, and allows to use very deep neural networks like AlexNet and ResNet18. It uses PySyft for the communication layer between parties and for fixed precision tensors, and was integrated into the library.

3.3.3 Use Case: Federated Learning on Edge Devices

In a comprehensive survey on implementing federated learning on mobile edge networks in terms of the background, challenges and existing solutions [LLH+20], PySyft is mentioned as an existing open-source framework for performing encrypted, privacy-preserving deep learning and implementations of techniques like SMPC and differential privacy in untrusted environments while protecting data. In a research article [FYS+20], PySyft was used for Federated Learning in IoT Edge devices for training machine learning models in edge devices in a secure and resource-efficient way. The authors explain their use of PySyft as the FL framework as "it provides the network worker structure that enables the remote communication of the model and uses the WebSocket protocol to lower overhead, and facilitates real-time data transfer from and to the Server". The PySyft library is also used by Chaulwar et al. in their privacy preserving framework FedCollabNN [Cha20] for training machine learning models at edge, which is computationally efficient and robust against adversarial attacks. PySyft has also been discussed as a tool for training deep neural networks on the Raspberry Pi 4 boards as edge devices [DB19]. Finally, in [KXJ+20], Kang et al. proposed a blockchain-empowered secure, scalable and communication-efficient decentralized federated edge learning system with a hierarchical blockchain framework consisting of a main chain and subchains. Their proposed framework was implemented using Pytorch, PySyft, and a blockchain platform named EOSIO.

3.3.4 Use Case: Healthcare and Medical Research

The benefits of PySyft as an adaptable secure and private machine learning framework have been leveraged in several healthcare-related research works. Our recent work with Kaissis et al. [KMRB20] describes the availability of frameworks like PySyft as instrumental to the widespread application of secure, privacy-preserving and federated machine learning techniques in medical imaging and medicine in general, and a number of current review articles reference PySyft in relation to medical image workflows, amongst others Quayyum et al. [QQBAF20], Suzen et al. [SŞ20] and Li et al. [LWH19]. Furthermore, several concrete healthcare-related use cases have been proposed utilizing PySyft. Passerat-Palmbach et al. in their work [PPFM+19] present a combination of federated learning and blockchain for use by healthcare consortia. Their proposed system architecture leverages federated learning building blocks from the PySyft library. In a 2019 research paper by Gao et al. on federated learning on heterogeneous EEG data for human behavior and emotion recognition tasks [GJW+19], the PySyft framework was used for Horizontal Federated Learning. PySyft has also been used in a prototype platform of federated-learning-based quantitative structure–activity relationships modeling for collaborative drug discovery in a work by Chen et al. [CXC+21]. Last, the PriMIA (Private Medical Imaging

Analysis) library [ZPPR+20], which is a collaborative development between the Technical University of Munich, Imperial College London and OpenMined, builds upon PySyft and PyGrid to offer a single interface for federated training of algorithms and encrypted inference-as-a-service, which is showcased on a multi-institutional case study on pediatric X-ray image data.

3.3.5 Use Case: Finance, Business or Industry

In [HSKS20], Hiessl et al introduce an industrial federated learning system to enable collaboration of business partners on common ML problems. As a future work, they would like to evaluate the incorporation of FL open source frameworks including PySyft with respect to production readiness and support for concurrent communication and computation required by the FL cohorts they introduce. In [THL+19], Tang et al. designed and implemented a general scheme for privacy-preserving and fair deep learning inference service in a three-worker model under the setting of publicly verifiable covert security. The implementation for secure three-party computation is completely based on PySyft and PyTorch. They also noted a major advantage of PySyft over MiniONN on the basis of PySyft's ability to split and share data and model using PointerTensors as opposed to MiniONN's requirement of transforming an entire existing neural network to an oblivious neural network. In addition, a potential use-case of federated learning and privacy-preserving collaborative learning has been discussed by Kawa et al. in [KPN+19] for credit risk assessment where multiple participating institutes will be able to collaboratively learn a shared prediction model while keeping all the data within themselves. Last, in her master's thesis [JA19] Jansson presented a framework for training a credit card fraud detection model using Federated Averaging implemented in PySyft.

3.3.6 Use Case: Anomaly Detection

PySyft has also been relied on for building federated learning based anomaly detection models. In [SBPB21], Singh et al. implemented unsupervised anomaly detection using federated learning. They used a special type of neural network called autoencoder and used it's reconstruction loss as the basis of classifying anomalies. The autoencoder part was obtained from PyTorch library, whereas the federated learning and aggregation of models were implemented using PySyft tools. Authors reported results comparable to the baseline model which was implemented without federated learning. Apart from that, as mentioned in [LGN+20] by Liu et al., a communication-efficient on-device federated learning based deep anomaly detection framework for sensing time-series data in Industrial IoT was implemented entirely using PySyft libraries.

3.4 Conclusion

PySyft is an open-source library built and maintained by the OpenMined community, which allows data scientists to perform arbitrary federated learning or encrypted computation, and also supports differential privacy. Its flexibility comes from the tensor chain structure, thanks to which users can combine multiple functionalities on top of standard tensors.

We have contributed to this library throughout the duration of this thesis, by building the core tensor and federated architecture, and later on by adding new privacy-preserving blocks like the AriaNN function secret sharing protocol introduced in chapter 5.

Chapter

4 Privacy Attacks in Machine Learning

This chapter is based on the paper [RPB+19] published in the proceedings of Advances in Neural Information Processing Systems, NeurIPS 2019.

Chapter content

Machine learning models, especially deep neural networks, are known to be sensitive to several attacks that aim at subverting the original goal of a model (like performing image classification for a given set of classes), in order to perform several tasks breaching the privacy of training data items or disclosing sensitive attributes about evaluated data items. For example, reverse-engineering attacks [FJR15] try to analyze the model to recover part of the training dataset, or model inversion attacks [SSSS17] analyze the behavior of a trained model on some input to determine if this input was part of the training dataset. We first propose a panorama of classical privacy attacks against data used for training or inference, which underlines the importance of developing privacy enhancing machine learning approaches, and we then expose one attack developed as part of [RPB+19].

4.1 Data Privacy Attacks on Neural Networks

We present here attacks that aim at disclosing information about the sensitive data. In particular, we do not consider methods like adversarial examples [YHZL19], model extraction [TZJ+16] or backdoor attacks [BVH+20] which underline the weaknesses of machine learning models but do not directly break the privacy of the data.

4.1.1 Membership Inference Attacks

Membership inference attacks on machine learning models were first reported by [SSSS17], where the model to attack is provided as a black-box. A membership inference attack consists of exploiting a trained model to assess whether a data item was used during training or not. We usually consider two types of attacks: in the *white-box* setting, the model is provided to the attacker who can analyze its architecture and the parameters' value. It is the most permissive setting, but it is of particular interest in the context of federated learning, where each party contributing its data has a direct access to the model. In the *black-box* setting, the model is not provided directly but can be queried to get predictions on data items provided by the attacker, as studied in [SSSS17]. This setting is very interesting because it fits many real case scenarios where a model is only available through an API by a company and is trained on sensitive data, should it be company internal data or data from individuals like healthcare data. Say a hospital releases a model which analyzes side effects of chemotherapy. No matter what data is used to train this model, if an attacker is able to infer if one patient was part or not of the dataset, it can learn that this patient underwent chemotherapy. Therefore the privacy leakage is here associated with the *risk* of belonging to such or such dataset, which for healthcare directly relates to the consent of the patient to having their data included in a research study.

The black-box setting is appealing because virtually any model as a service is sensitive to such attacks. The way the attack is usually conducted is by training an attacker model that uses the output signal of the target model on some input to perform the binary classification of membership. Providing this attacker model with examples of items inside and outside the training dataset is not possible in real scenarios, since no precise information about how the target model was trained is generally available. To circumvent this, one builds several proxies to the target model by training models that are believed to be close in terms of architecture, on proxy datasets which are also believed to be comparable to the target dataset. The attacker model can then be trained on the output of this proxy model, and [SSSS17] shows that its predictions generalize well on the target models, providing a significant advantage in the membership inference over the random guess performance. [SSSS17] experiments are mostly based on vision datasets or on numerical datasets. [SS19] also studied black-box membership inference attacks but in the context of natural language processing on models like recurrent neural networks, where they provide evidence that such models are also sensitive to these attacks.

Recently, authors of [CCN+22] have proposed a new approach to quantify how successful a membership attack is. Instead of relying on average-case success metrics such as the accuracy, they argue that a better metric should be able to quantify the success of an attack when only a small fraction of the population is targeted. As such, they propose to measure the true-positive rate at low false-positive rates, thus reproducing standard practices from computer security [HSJ+17].

4.1.2 Model Inversion

The power of model inversion attacks, also known as reconstruction attacks, have been brought in light by [FJR15] which has shown that trained models, either provided in the white-box or in the black-box setting, can be exploited to recover sensitive attributes about individuals present in the training dataset. One particular example provided in the study is through two different attacks on a facial recognition model, which associates a name from a pre-defined list to a facial image. The first attack consists of exploiting the model to produce an image given a name in the list. The success of the attack is measured through the ability to match the produced image with one in the set of faces presented to the attacker, composed of one face per name. The second attack consists of providing an attacker with a blurred image for which the model is unable to perform its classification, and the attacker exploits the model to try to deblur the image. The success is then measured in a similar way by trying to match the deblurred image with a representative set.

[FJR15] used a simple method to perform this attack which is based gradient descent. The main idea is to update an input $\mathbf{x}$ of class $c \in [0, C-1]$ so as to maximize the output of the model $f(\theta, \mathbf{x}) \in \mathbb{R}^C$ on its c-th component, where this output is modelling a probability to belong to each of the C classes and is typically implemented using a softmax activation function. Gradient descent, as commonly used to train models parametrized with $\theta \in \mathbb{R}^p$, consists of iteratively computing $\theta_{k+1} = \theta_k - \eta \nabla_\theta f(\theta_k, \mathbf{x})$. Here, as the model parameters are freezed and $\mathbf{x}$ is being optimized, it writes instead $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla_{\mathbf{x}} f(\theta, \mathbf{x}_k)$. In the white-box setting, $\nabla_x f$ can easily be computed, but in the black-box setting, the gradient needs to be computed approximately based on the observed output $f(\mathbf{x})$. As they show and as we illustrate in a different setting in our work on collateral learning, the precision with which the output probabilities are provided directly conditions the success of the attack, as low precision compromises the ability to numerically estimate the gradient. This is an interesting observation as the black-box setting usually corresponds to machine learning as a service, a scenario where the user of the service is mainly interested by the predicted class than the exactly probability distribution output by the model, and even when this distribution is required by the user, it needs not be provided with great precision. Hence, providing the output of the model with low precision is en efficient countermeasure to model inversion attacks in the black-box setting.

Federated learning allows for a new kind of attack based on the inversion of gradients during training instead of the fully trained model and has been explored by [GBDM20]. Indeed, since the users usually send the updates of the model through the gradients of its parameters, such gradients can be exploited by the central server to try to reconstruct the user batch data. The authors demonstrate how successful this attack can be, especially but not solely for fully connected networks.

Membership inference and model inversion attacks can be formalized through the advantage of an adversary $\mathcal{A}$:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MI}}(1^\lambda) := \Pr\left[b' = b : \begin{array}{l} \mathbf{x} \xleftarrow{\$} \mathbf{\Omega} \\ b \leftarrow \tau(\mathbf{x}) \\ c \leftarrow \mathsf{Gen}(\mathcal{M}, \mathcal{D}, \mathbf{x}) \\ b' \leftarrow \mathcal{A}^{\mathcal{M}(\mathcal{D})}(c) \end{array} \right] - \Pr\left[b' = b : \begin{array}{l} \mathbf{x} \xleftarrow{\$} \mathbf{\Omega} \\ b \leftarrow \tau(\mathbf{x}) \\ c \leftarrow \mathsf{Gen}(\mathcal{M}, \mathcal{D}, \mathbf{x}) \\ b' \leftarrow \mathcal{A}^{*}(c) \end{array} \right]$$

Where τ is the objective function which maps any $\mathbf{x}$ from the sample space $\mathbf{\Omega}$ to some objective space $\{0,1\}^*$. The advantage corresponds to the difference between the probability of success for adversary $\mathcal{A}$ to predict the output of τ on $\mathbf{x}$ given some auxiliary information c derived from $\mathbf{x}$ using Gen, when the adversary has some access to the trained model $\mathcal{M}(\mathcal{D})$ or not. This access can be limited as in the black-box setting or not, like as in the white-box setting.

For both attacks, $\mathsf{Gen}(\mathcal{M}, \mathcal{D}, \mathbf{x}) := h(\mathcal{M}(\mathcal{D})(\mathbf{x}))$, where h is a post-processing function over the output of the model over $\mathbf{x}$. For membership inference attacks, the objective function τ is defined as such:

$$\tau(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in \mathcal{D} \\ 0 & \mathbf{x} \notin \mathcal{D} \end{cases}$$

and for model inversion attacks, τ is the identity function:

$$\tau(\mathbf{x}) = \mathbf{x}$$

.

4.2 Collateral Learning

As part of our work [RPB+19], we introduce a new functional encryption scheme which is described more in depth in the following chapter and we show how to use it to build privacy preserving neural networks that perform well on simple image classification problems. Functional encryption (FE) [BSW11, O'N10] allows users to receive in plaintext evaluations on a function over encrypted data: for a function f, a functional decryption key is generated such that, given any ciphertext with underlying plaintext $\mathbf{x}$, a user can use this key to obtain $f(\mathbf{x})$ without learning $\mathbf{x}$ or any other information than $f(\mathbf{x})$.

An interesting question that emerges as part of this work is: how much information does $f(\mathbf{x})$ reveal about $\mathbf{x}$? In the case of this FE scheme which only allows building small neural networks with a single hidden layer and a quadratic activation function, this question is particularly important. We develop an attack that we coin *collateral learning* which does not consist in reconstructing the original input as in model inversion, but instead in inferring sensitive *collateral* attributes of the input which were not part of the learning task. These attacks can be classified in the family of property inference attacks which is formalized in [RG20] as the family of attacks that extract input properties which were not features and were not correlated to the learning task. In addition, we present an adversarial training technique to process a model in order to improve privacy, so that its output which is in plaintext cannot be exploited by adversaries to recover specific sensitive information at test time.

Our contributions

We first provide a thorough analysis of the information leaks from the output of a model, based on indistinguishability of data items of the same label. We show how the leaks materialize using a privacy attack against some input properties. Second, we propose a training method to prevent selected sensitive features from leaking, which adversarially optimizes the network against an adversary trying to identify these features. This is interesting since many encryption schemes cannot deal with the thresholding operation commonly used for classification to select the most probable class, and it comes with little reduction on the model's accuracy while significantly improving data privacy.

4.2.1 Background

Quadratic and Polynomial Neural Networks

Polynomial neural networks are a class of networks which only use linear elements, like fully connected linear layers, convolutions but with average pooling, and model activation functions with polynomial approximations when not simply the square function. Despite these simplifications, they have proved themselves satisfactorily accurate for relatively simple tasks ([DGBL+16] learns on MNIST and [BCL+18] on CIFAR10 [Kri12]). The simplicity of the operations they build on guarantees good efficiency, especially for the gradient computations, and works like [LSSS14] have shown that they can achieve convergence rates similar to those of networks with non-linear activations.

In particular, they have been used for several early stage implementations in cryptography [DGBL+16, CGGI16, BMMP18] to demonstrate the usability of new protocols for machine learning. However, the `argmax` or other thresholding functions present at the output of a classifier network to select the predicted class among the output neurons cannot be conveniently handled, so several protocol implementations (among which ours) run polynomial networks on encrypted inputs, but take the `argmax` over the decrypted output of the network. This results in potential information leakage which can be maliciously exploited.

Indistinguishability

We analyze how sensitive the output $f(\mathbf{x})$ is with respect to the private input $\mathbf{x}$, using a relaxed version of indistinguishability. More precisely, we assume that our input data can be used to predict public labels but also sensitive private ones, respectively y_{pub} and y_{priv}. Our quadratic FE scheme denoted by q aims at predicting y_{pub} and an adversary would rather like to infer y_{priv}. In this case, the security game consists in the adversary providing two inputs $(\mathbf{x}_0, \mathbf{x}_1)$ labeled with the same y_{pub} but a different y_{priv} and then trying to distinguish which one was selected by the challenger, given its output $q(\mathbf{x}_b)$, $b \in \{0, 1\}$. One way to do this is to measure the ability of an adversary to predict y_{priv} for items which all belong to the same y_{pub} class, given only outputs of q.

In particular, note that we do not consider approaches based on input reconstruction (as done by [CFLS18]) because in many cases, the adversary is not interested in reconstructing the whole input, but rather wants to get insights into specific properties of the data.

Another way to see this problem is that we want the sensitive label y_{priv} to be independent from the decrypted output $q(\mathbf{x})$ given the true public label y_{pub}. This independence notion is known as *separation* and is used as a fairness criterion in [BHN18] if the sensitive features can be misused for discrimination.

4.2.2 Private Inference Setting

Classifying in two directions

We are interested in specific types of datasets $(\mathbf{x}_i)_{i=1,\dots,n}$ which have public labels y_{pub} but also private ones y_{priv}. Moreover, these different types of labels should be entangled, meaning that they should not be easily separable, unlike the color and the shape of an object in an image for example which can be simply separated. For example, in the spam filtering use case mentioned above, y_{pub} would be a spam flag, and y_{priv} would be some marketing information highlighting areas of interest of the email recipient like technology, culture, etc. In addition, to simplify our analysis, we assume that classes are balanced for all types of labels, and that labels are independent from each other given the input: $\forall \mathbf{x}, P(y_{\text{pub}}, y_{\text{priv}}|\mathbf{x}) = P(y_{\text{pub}}|\mathbf{x})P(y_{\text{priv}}|\mathbf{x})$. To illustrate our approach in the case of image recognition, we propose a synthetic dataset inspired by MNIST which consists of 60 000 gray scaled images of 28×28 pixels representing digits using two fonts and some distortion, as shown in Figure 4.1. Here, the public label y_{pub} is the digit on the image and the private one y_{priv} is the font used to draw it.

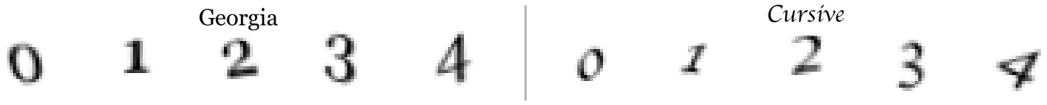

Figure 4.1: Artificial dataset inspired from MNIST with two types of labels.

We define two tasks: a *main* task which tries to predict y_{pub} using a partially-encrypted polynomial neural network with functional encryption, and a *collateral* task which is performed by an adversary who tries to leverage the output of the FE encrypted network at test time to predict y_{priv}. Our goal is to perform the main task with high accuracy while making the collateral one as bad as random predictions. In terms of indistinguishability, given a dataset with the same digit drawn, it should be infeasible to detect the used font.

Threat of Collateral Learning

A typical adversary would have read access to the main task classification process. It would leverage the output of the quadratic network to try to learn the font used on ciphered images. To do this, all that is needed is to train another network on top of the quadratic network so that it learns to predict the font, assuming some access to labeled samples (which is the case if the adversary encrypts itself images and provides them to the main task at evaluation time). Note that in this case the private network is not updated by the collateral network as we assume it is only provided as a black-box after the main task is trained. Figure 4.3 summarizes the setting.

We implemented this scenario using as adversary a neural network composed of a first layer acting as a decompression step where we increase the number of neurons from 10 back to 28×28 and add on top of it a classical convolutional neural network[1] convolutional. This

[1] Described here: `https://github.com/pytorch/examples/blob/master/mnist/main.py`

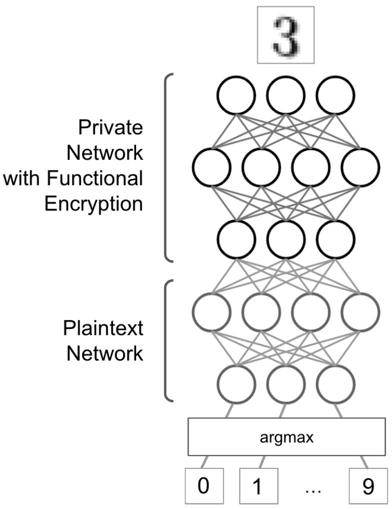

Figure 4.2: Semi-encrypted network using quadratic FE.

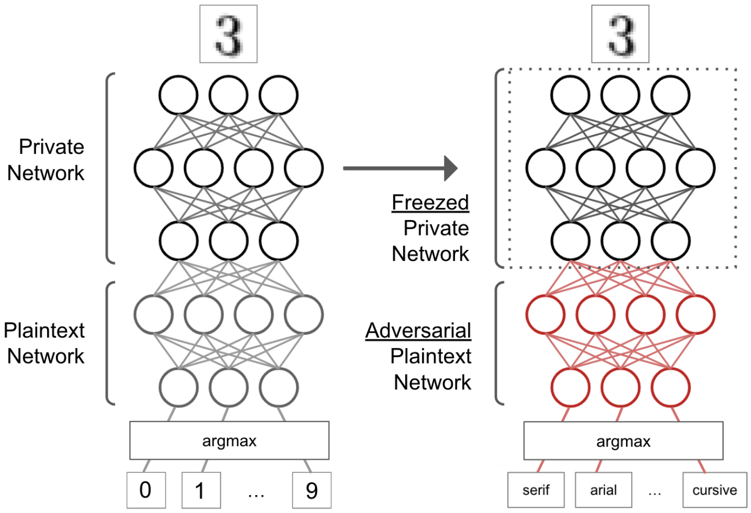

Figure 4.3: Semi-encrypted network with an adversary trying to recover private labels from the private network.

structure is reminiscent of autoencoders [VLBM08] where the bottleneck is the public output of the private net and the challenge of this autoencoder is to correctly memorize the public label while forgetting the private one. What we observed is striking: in less than 10 epochs, the collateral network leverages the 10 public neurons output and achieves 93.5% accuracy for the font prediction. As expected, it gets even worse when the adversary is assessed with the indistinguishability criterion because in that case the adversary can work on a dataset where only a specific digit is represented: this reduces the variability of the samples and makes it easier to distinguish the font; the probability of success is indeed of 96.9%.

We call *collateral learning* this phenomenon of learning unexpected features and we show in the next section how to implement counter-measures to this threat in order to improve privacy.

4.2.3 Defeating Collateral Learning

Reducing information leakage

Our first approach is based on the observation that we leak many bits of information. We first investigate whether we can reduce the number of outputs of the privately-evaluated network, as adding extra layers on top of the private network makes it no longer necessary to keep 10 of them. The intuition is that if the information that is relevant to the main task can fit in less than 10 neurons, then the extra neurons would leak unnecessary information. We have therefore a trade-off between reducing too much and losing desired information or keeping a too large output and having an important leakage. We can observe this through the respective accuracies as it is shown in Figure 4.4, where the main and adversarial networks are CNNs as in Section 4.2.2 with 10 epochs of training using 7-fold cross validation. What we observe here is interesting: the main task accuracy does not drop significantly even when we reach an output size of 3 neurons, where it reaches 97.1% which is still very good although 2% under the best accuracy. In return, the collateral accuracy starts to significantly decrease when output size is below 7. At size 4, it is only 76.4% on average so 18% less than the baseline. We will keep an output size of 3 or 4 for the next experiments to keep the main accuracy almost unchanged.

Another hyperparameter that we can consider is the weight compression: how many bits do we need to represent the weights on the private networks layers? This is of interest for the FE scheme as we need to convert all weights to integers and those integers will be low provided that the compression rate is high. Small weight integers mean that the output of the private

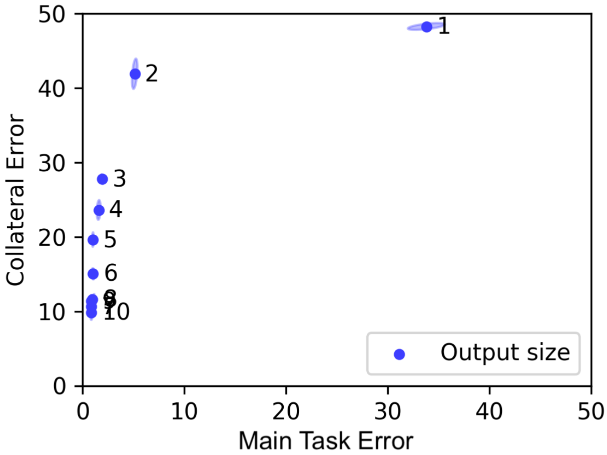

Figure 4.4: Trade-off between main and collateral accuracies depending on the private output size.

network has a relatively low amplitude and can be therefore efficiently decrypted using discrete logarithm. We managed to express all weights and even the input image using 4 bit values with limited impact on the main accuracy and almost none on the collateral one. Details about compression can be found in Appendix A.1.2.

Adversarial training

We propose a new approach to actively adapt against collateral learning. The main idea is to simulate adversaries and to try to defeat them. To do this, we use semi-adversarial training and simultaneously optimize the main classification objective and the opposite of the collateral objective of a given simulated adversary. The function that we want to minimize at each iteration step can be written:

$$\min_{\theta_q}[\min_{\theta_{\text{pub}}} \mathcal{L}_{\text{pub}}(\theta_q, \theta_{\text{pub}}) - \alpha \min_{\theta_{\text{priv}}} \mathcal{L}_{\text{priv}}(\theta_q, \theta_{\text{priv}})].$$

This approach is inspired from [LKC17] where the authors train some objective against nuisances parameters to build a classifier independent from these nuisances. Private features leaking in our scheme can indeed be considered to be a nuisance. However, our approach goes one step further as we do not just stack a network above another; our global network structure is fork-like: the common basis is the private network and the two forks are the main and collateral classifiers. This allows us to have a better classifier for the main task which is not as sensitive to adversarial training as the scheme exposed by [LKC17, Figure 1]. One other difference is that the collateral classifier is a specific modeling of an adversary, and we will discuss this in detail in the next section. We define in Figure 4.5 the 3-step procedure used to implement this semi-adversarial training using partial back-propagation.

4.2.4 Experimental Results

Accurate main task and poor collateral results. In Figures 4.6 and 4.7 we show that the output size has an important influence on the two tasks' performances. For this experiment, we use $\alpha = 1.7$ as detailed in Appendix A.1.2, the adversary uses the same CNN as stated above and the main network is a simple feed forward network (FFN) with 4 layers. We observe that both networks behave better when the output size increases, but the improvement is not synchronous which makes it possible to have a main task with high accuracy while the collateral task is still very inaccurate. In our example, this corresponds to an output size between 3 and

Pre-training: *Initial phase where both tasks learn and strengthen before the joint optimization*
Minimize $\mathcal{L}_{\text{pub}}(\theta_q, \theta_{\text{pub}})$
Minimize $\mathcal{L}_{\text{priv}}(\mathsf{Frozen}(\theta_q), \theta_{\text{priv}})$

Semi-adversarial training: *The joint optimization phase, where θ_{pub} and θ_{priv} are updated depending on the variations of θ_q and θ_q is optimized to reduce the loss $\mathcal{L} = \mathcal{L}_{\text{pub}} - \alpha\mathcal{L}_{\text{priv}}$*
Minimize $\mathcal{L}_{\text{pub}}(\mathsf{Frozen}(\theta_q), \theta_{\text{pub}})$
Minimize $\mathcal{L}_{\text{priv}}(\mathsf{Frozen}(\theta_q), \theta_{\text{priv}})$
Minimize $\mathcal{L} = \mathcal{L}_{\text{pub}}(\theta_q, \mathsf{Frozen}(\theta_{\text{pub}})) - \alpha\mathcal{L}_{\text{priv}}(\theta_q, \mathsf{Frozen}(\theta_{\text{priv}}))$

Recover phase: *Both tasks recover from the perturbations induced by the adversarial phase, θ_q does not change anymore*
Minimize $\mathcal{L}_{\text{pub}}(\mathsf{Frozen}(\theta_q), \theta_{\text{pub}})$
Minimize $\mathcal{L}_{\text{priv}}(\mathsf{Frozen}(\theta_q), \theta_{\text{priv}})$

Figure 4.5: Our semi-adversarial training scheme.

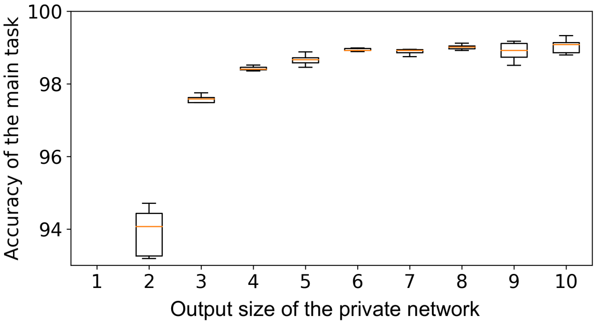

Figure 4.6: Influence of the output size on the main task accuracy with adversarial training.

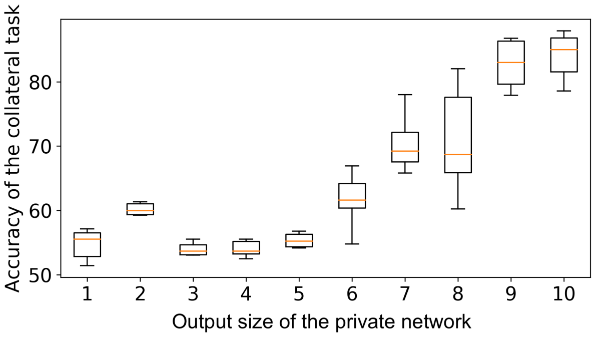

Figure 4.7: Influence of the output size on the collateral task accuracy with adversarial training.

5. Note that the collateral result is the accuracy at the distinction task, i.e., the digit is fixed for the adversary which trains to distinguish two fonts during a 50 epoch *recover phase* using 7-fold cross validation, after 50 epochs of *semi-adversarial training* have been spent to reduce leakage from the private network.

Generalizing resistance against multiple adversaries. In practice, it is very likely that the adversary will use a different model than the one against which the protection has been built. We have therefore investigated how building resistance against a model $\mathcal{M}$ can provide resistance against other models. Our empirical results tend to show that models with less parameters than $\mathcal{M}$ do not perform well. In return, models with more parameters can behave better, provided that the complexity does not get excessive for the considered task, because it would not provide any additional advantage and would just lead to learning noise. In particular, the CNN already mentioned above seems to be a sufficiently complex model to resist against a wide range of feed forward (FFN) and convolutional networks, as illustrated in Figure 4.8 where the measure used is indistinguishability of the font for a fixed digit. This study is not exhaustive as the adversary can change the activation function (here we use ReLU) or even the training parameters (optimizer, batch size, dropout, etc.), but these do not seem to provide any decisive advantage.

We also assessed the resistance to a large range of other models from the sklearn library [PVG+11] and report the collateral accuracy in Figure 4.9. As can be observed, some models such as k-nearest neighbors or random forests perform better compared to neural networks, even if their accuracy remains relatively low. One reason can be that they operate in a very different manner compared to the model on which the adversarial training is performed: k-nearest neighbors for example just considers distances between points.

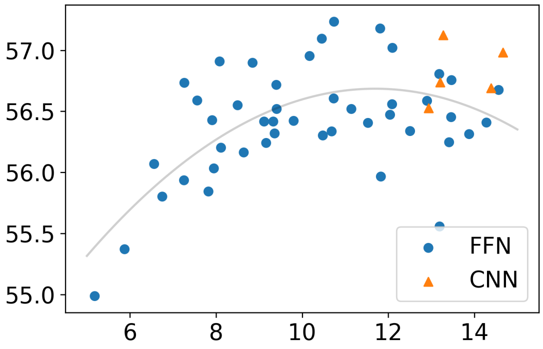

Figure 4.8: Collateral accuracy depending of the adversarial network complexity seen as the log of the number of parameters.

Linear Ridge Regression	53.5 ± 0.5%
Logistic Regression	52.5 ± 0.6%
Quad. Discriminant Analysis	54.9 ± 0.3%
SVM (RBF kernel)	57.9 ± 0.4%
Gaussian Process Classifier	53.8 ± 0.3%
Gaussian Naive Bayes	53.2 ± 0.5%
K-Neighbors Classifier	58.1 ± 0.7%
Decision Tree Classifier	56.8 ± 0.4%
Random Forest Classifier	58.9 ± 0.2%
Gradient Boosting Classifier	58.9 ± 0.2%

Figure 4.9: Accuracy on the distinction task for different adversarial learning models.

Chapter

5 Private Evaluation and Training of Neural Networks

This chapter is based on the article [RPB+19] published in the proceedings of Advances in Neural Information Processing Systems, NeurIPS 2019 and on the article [RTPB22] published in Proceedings on Privacy Enhancing Technologies Symposium, PoPETS 2022.

Chapter content

Beyond the analysis of federated systems, the development of new types of attacks against trained models together with some countermeasures, we turn ourselves towards another problem which can be summed up as such: can we have an algorithm interact with some data without exposing any of those? One typical example would be a regulation authority that builds a model to analyze financial fraud using sensitive data from banks. On the one hand, the authority does not want to expose its model to the banks to avoid any form of optimization against that model in order to slip under the radar. On the other hand, banks are not willing to expose their customers' data, even to a regulation authority. Cryptography protocols are a common solution to this problem, since they allow to encrypt or hide the data before sharing it to other parties. For a computation to be possible, such as a model evaluation or training using the data, we require the protocol to satisfy homomorphic properties, which roughly means that applying operations on encrypted inputs should be equivalent to applying the operation on the inputs and then encrypting the result. Numerous protocols have been proposed but we have investigated two specific directions. The first one goes somewhat beyond standard scenarios for homomorphic encryption and combines the computation step and the decryption step together, which is very interesting in situations where we want to avoid an extra decryption request after the computation was made. It is called *functional encryption* (FE) and we explore new capabilities for FE algorithms. In particular, we show that our algorithm can be used to evaluate small neural networks in a semi-encrypted way. The second direction we explore is the family of SMPC protocols based on additive secret sharing, which have been studied more intensively in the context of private machine learning. Existing results are promising in terms of the flexibility they offer to support various neural layers but they require a prohibitive number of interaction rounds which make them impractical for real world scenarios with high latency. We build upon a cryptographic primitive known as *function secret sharing* (FSS) to propose a new low-interaction protocol for comparison, which is a core primitive of neural network non linearities and a bottleneck in private ML in terms of computation time. We show that we can train and evaluate real world neural networks in a fully private fashion, and we have integrated our implementation as part of the PySyft library to enlarge the privacy-preserving toolbox of privacy aware data scientists.

5.1 Partially Encrypted Machine Learning using Functional Encryption

As both public opinion and regulators are becoming increasingly aware of issues of data privacy, the area of privacy-preserving machine learning has emerged with the aim of reshaping the way machine learning deals with private data. Breakthroughs in fully homomorphic encryption (FHE) [BMMP18, CGGI16] and secure multi-party computation (SMPC) [DPSZ12, WGC19] have made computation on encrypted data practical and implementations of neural networks to do encrypted predictions have flourished [RWT$^+$18, RTD$^+$18, SEA19, BFL$^+$11, BPTG15].

However, these protocols require the data owner encrypting the inputs and the parties performing the computations to interact and communicate in order to get decrypted results, which we would like to avoid in some cases, like spam filtering, for example, where the email receiver should not need to be online for the email server to classify incoming email as spam or not. Functional encryption (FE) [BSW11, O'N10] in return does not need interaction to compute over encrypted data: it allows users to receive in plaintext specific functional evaluations of encrypted data: for a function f, a functional decryption key can be generated such that, given any ciphertext with underlying plaintext $\mathbf{x}$, a user can use this key to obtain $f(\mathbf{x})$ without learning $\mathbf{x}$ or any other information than $f(\mathbf{x})$. It stands in between traditional public key encryption, where data can only be directly revealed, and FHE, where data can be manipulated but cannot be revealed: it allows the user to tightly control what is disclosed about his data.

Use cases

Spam filtering. Consider the following scenario: Alice uses a secure email protocol which makes use of functional encryption. Bob uses Alice's public key to send her an email, which lands on Alice's email provider's server. Alice gave the server keys that enable it to process the email and take a predefined set of appropriate actions without her being online. The server could learn how urgent the email is and decide accordingly whether to alert Alice. It could also detect whether the message is spam and store it in the spam box right away.

Privacy-preserving enforcement of content policies Another use case could be to enable platforms, such as messaging apps, to maintain user privacy through end-to-end encryption, while filtering out content that is illegal or does not adhere to policies the site may have regarding, for instance, abusive speech or explicit images.

These applications are not currently feasible within a reasonable computing time, as the construction of FE for all kinds of circuits is essentially equivalent to *indistinguishable obfuscation* [BGI$^+$01, GGH$^+$13], concrete instances of which have been shown insecure, let alone efficient. However, there exist practical FE schemes for the inner-product functionality [ABDP15, ALS16] and more recently for quadratic computations [BCFG17], that are usable for practical applications.

Our contributions

We introduce a new FE scheme to compute quadratic forms which outperforms that of Baltico et al. [BCFG17] in terms of complexity, and provide an efficient implementation of this scheme. We show how to use it to build privacy preserving neural networks, which perform well on simple image classification problems. Specifically, we show that the first layers of a polynomial network can be run on encrypted inputs using this quadratic scheme.

We demonstrate the practicality of our approach using a dataset inspired from MNIST [LC10], which is made of images of digits written using two different fonts. We show how to perform classification of the encrypted digit images in less than 3 seconds with over 97.7% accuracy.

This paper builds on a preliminary version available on the Cryptology ePrint Archive. All code and implementations can be found online[1].

5.1.1 Background on Functional Encryption

Functional encryption extends the notion of public key encryption where one uses a public key pk and a secret key sk to respectively encrypt and decrypt some data. More precisely, pk is still used to encrypt data, but for a given function f, sk can be used to derive a functional decryption key dk_f which will be shared to users so that, given a ciphertext of x where x is an integer from a set $\mathcal{X}$, they can decrypt $f(x)$ but not x. In particular, someone having access to dk_f cannot learn anything about x other than $f(x)$. Note also that functions cannot be composed, since the decryption happens within the function evaluation. Hence, only single quadratic functions can be securely evaluated. We provide below a more formal definition, together with notions of correctness and security.

5.1.1.1 Formal Definition of Functional Encryption

Functional encryption relies on a pair of keys like in public key encryption: a master secret key msk and a public key pk. The public key pk can be shared and is used to encrypt the data, while the master secret key msk is used to build functional decryption keys dk_f for $f \in \mathcal{F}$. A user having access to c an encryption of x with pk and to dk_f can learn $f(x)$ but cannot learn anything else about x.

We give the definition of Functional Encryption, originally defined in [BSW11, O'N10].

Definition 8 (Functional Encryption)**.** A *functional encryption* scheme FE for a set of functions $\mathcal{F} \subseteq \mathcal{X} \to \mathcal{Y}$ is a tuple of PPT algorithms $\mathsf{FE} = (\mathsf{SetUp}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows.

$\mathsf{SetUp}(1^\lambda, \mathcal{F})$ takes as input a security parameter 1^λ, the set of functions $\mathcal{F}$, and outputs a master secret key msk and a public key pk.

$\mathsf{KeyGen}(\mathsf{msk}, f)$ takes as input the master secret key and a function $f \in \mathcal{F}$, and outputs a functional decryption key dk_f.

$\mathsf{Enc}(\mathsf{pk}, x)$ takes as input the public key pk and a message $x \in \mathcal{X}$, and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{dk}_f, \mathsf{ct})$ takes as input a functional decryption key dk_f and a ciphertext ct, and returns an output $y \in \mathcal{Y} \cup \{\perp\}$, where $\perp$ is a special rejection symbol.

[1] Available at https://github.com/LaRiffle/collateral-learning and https://github.com/edufoursans/reading-in-the-dark

5.1.1.2 Correctness

Perfect correctness. Perfect correctness is achieved in functional encryption: $\forall x \in \mathcal{X}$, $f \in \mathcal{F}$, $\Pr[\mathsf{Dec}(\mathsf{dk}_f, \mathsf{ct}) = f(x)] = 1$, where $\mathsf{dk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$. Note that this property is a very strict condition, which is not satisfied by existing fully homomorphic encryption schemes (FHE), such as [BV11, GSW13].

5.1.1.3 IND-CPA Security

To assess the security of our framework, we first consider the FE scheme security and make sure that we cannot learn anything more than what the function is supposed to output given an encryption of x. We will rely on *indistinguishability* [GM84], a classical security notion which can be summed up in the following game: an adversary provides two input items to the challenger (here our FE algorithm), and the challenger chooses one item to be encrypted, runs encryption on it before returning the output. The adversary should not be able to detect which input was used. This is known as IND-CPA security and is formally defined in the preliminary chapter in section 2.3.2.

With notations of above, for any stateful adversary $\mathcal{A}$ and any functional encryption scheme FE, IND-CPA security translates in the following advantage:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FE}}(\lambda) := \Pr\left[\beta' = \beta : \begin{array}{l} (\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\lambda, \mathcal{F}) \\ (x_0, x_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk}) \\ \beta \xleftarrow{\$} \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x_\beta) \\ \beta' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) \end{array}\right] - \frac{1}{2},$$

with the restriction that all queries f that $\mathcal{A}$ makes to key generation algorithm $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ must satisfy $f(x_0) = f(x_1)$.

We say FE is IND-CPA secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FE}}(\lambda) = \mathrm{negl}(\lambda)$.

5.1.1.4 Bilinear Groups

Our FE scheme uses bilinear (or *pairing*) groups, whose use in cryptography has been introduced by [BF03, Jou04] and which are defined in the preliminary chapter in section 2.1. Given λ a security parameter, let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order p (for a 2λ-bit prime p) and g_1 and g_2 their generators, respectively. The application $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a pairing if it is efficiently computable, non-degenerated, and bilinear: $e(g_1^\alpha, g_2^\beta) = e(g_1, g_2)^{\alpha\beta}$ for any $\alpha, \beta \in \mathbb{Z}_p$. Additionally, we define $g_T := e(g_1, g_2)$ which spans the group $\mathbb{G}_T$ of prime order p.

We will denote by GGen a probabilistic polynomial-time (PPT) algorithm that on input 1^λ returns a description $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, p, g_1, g_2, e)$ of an asymmetric bilinear group. For convenience, given $s = 1$, 2 or T, $n \in \mathbb{N}$ and vectors $\mathbf{u} := (u_1 \dots u_n) \in \mathbb{Z}_p^n$, $\mathbf{v} \in \mathbb{Z}_p^n$, we denote by $g_s^{\mathbf{u}} := (g_s^{u_1} \dots g_s^{u_n}) \in \mathbb{G}_s^n$ and $e(g_1^{\mathbf{u}}, g_2^{\mathbf{v}}) = \prod_{i=1} e(g_1, g_2)^{u_i \cdot v_i} = e(g_1, g_2)^{\mathbf{u}\cdot\mathbf{v}} \in \mathbb{G}_T$, where $\mathbf{u} \cdot \mathbf{v}$ is the inner product, i.e. $\mathbf{u} \cdot \mathbf{v} := \sum_{i=1}^n u_i v_i$.

Last, GL_2 denotes the general linear group of degree 2, i.e. the multiplicative group of 2×2 invertible matrices.

5.1.2 Our Context for Private Inference

5.1.2.1 Functional Encryption Scheme

We now introduce our new framework for quadratic functional encryption and show that it can be used to partially encrypt a polynomial network.

We build an efficient FE scheme for the set of quadratic functions defined as $\mathcal{F}_{n,B_x,B_y,B_q} \subset \{q : [-B_x, B_x]^n \times [-B_y, B_y]^n \to \mathbb{Z}\}$, where q is described as a set of bounded coefficients $\{q_{i,j} \in [-B_q, B_q]\}_{i,j \in [n]}$ and for all vectors $(\mathbf{x}, \mathbf{y})$, we have $q(\mathbf{x}, \mathbf{y}) = \sum_{i,j \in [n]} q_{i,j} x_i y_j$.

A complete version of our scheme is given in Figure 5.1, but here are the main ideas and notations. A pair of vectors $(\mathbf{s}, \mathbf{t})$ is first selected and constitutes the private key msk, while the public key is $(g_1^{\mathbf{s}}, g_2^{\mathbf{t}})$. Encrypting $(\mathbf{x}, \mathbf{y})$ roughly consists of masking $g_1^{\mathbf{x}}$ and $g_2^{\mathbf{y}}$ with $g_1^{\mathbf{s}}$ and $g_2^{\mathbf{t}}$, which allows any user to compute $g_T^{q(\mathbf{x},\mathbf{y})-q(\mathbf{s},\mathbf{t})}$ with for any quadratic function q, using the pairing. The functional decryption key for a specific q is $g_T^{q(\mathbf{s},\mathbf{t})}$ which allows to get $g_T^{q(\mathbf{x},\mathbf{y})}$. Last, taking the discrete logarithm gives access to $q(\mathbf{x}, \mathbf{y})$ (discrete logarithm for small exponents is easy). Security uses the fact that it is hard to compute msk from pk (discrete logarithm for large exponents $\mathbf{s}, \mathbf{t}$ is hard to compute). More details are given in Appendix A.1.1[2]

```
$\mathsf{SetUp}(1^\lambda, \mathcal{F}_{n,B_x,B_y,B_f})$:
$\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, g_1, g_2, e) \leftarrow \mathsf{GGen}(1^\lambda)$, $\mathbf{s}, \mathbf{t} \xleftarrow{\$} \mathbb{Z}_p^n$, $\mathsf{msk} := (\mathbf{s}, \mathbf{t})$, $\mathsf{pk} := (\mathcal{PG}, g_1^{\mathbf{s}}, g_2^{\mathbf{t}})$
Return $(\mathsf{pk}, \mathsf{msk})$.

$\mathsf{Enc}(\mathsf{pk}, (\mathbf{x}, \mathbf{y}))$:
$\gamma \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{W} \xleftarrow{\$} \mathsf{GL}_2$, for all $i \in [n]$, $\boldsymbol{a}_i := (\mathbf{W}^{-1})^\top \begin{pmatrix} x_i \\ \gamma s_i \end{pmatrix}$, $\boldsymbol{b}_i := \mathbf{W} \begin{pmatrix} y_i \\ -t_i \end{pmatrix}$
Return $\mathsf{ct} := \left(g_1^\gamma, \{g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_i}\}_{i \in [n]}\right) \in \mathbb{G}_1 \times (\mathbb{G}_1^2 \times \mathbb{G}_2^2)^n$

$\mathsf{KeyGen}(\mathsf{msk}, q)$:
Return $\mathsf{dk}_f := \left(g_2^{q(\mathbf{s},\mathbf{t})}, q\right) \in \mathbb{G}_2 \times \mathcal{F}_{n,B_x,B_y,B_q}$.

$\mathsf{Dec}\left(\mathsf{pk}, \mathsf{ct} := \left(g_1^\gamma, \{g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_i}\}_{i \in [n]}\right), \mathsf{dk}_q := \left(g_2^{q(\mathbf{s},\mathbf{t})}, q\right)\right)$:
$out := e(g_1^\gamma, g_2^{q(\mathbf{s},\mathbf{t})}) \cdot \prod_{i,j \in [n]} e(g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_j})^{q_{i,j}}$
Return $\log(out) \in \mathbb{Z}$.
```

Figure 5.1: Our functional encryption scheme for quadratic polynomials.

Theorem 9 (Security, correctness and complexity)**.** *The FE scheme provided in Figure 5.1:*

- *is IND-CPA secure in the Generic Bilinear Group Model,*
- *verifies* $\log(out) = q(\mathbf{x}, \mathbf{y})$ *and satisfies perfect correctness,*
- *has a overall decryption complexity of* $2n^2(E + P) + P + D$,

where E, P and D respectively denote exponentiation, pairing and discrete logarithm complexities.

Our scheme outperforms previous schemes for quadratic FE with the same security assumption, like the one from [BCFG17, Sec. 4] which achieves $3n^2(E + P) + 2P + D$ complexity and uses larger ciphertexts and decryption keys. Note that the efficiency of the decryption can even be further optimized for those quadratic polynomials used that are relevant to our application (see Section 5.1.2.2).

Computing the discrete logarithm for decryption. Our decryption requires computing discrete logarithms of group elements in base g_T, but contrary to previous works like [KLM+18] it is independent of the ciphertext and the functional decryption key used to decrypt. This allows pre-computing values and dramatically speeds-up decryption.

[2] Note that we only present a simplified scheme here. In particular, the actual encryption is randomized, which is necessary to achieve IND-CPA security.

5.1.2.2 Equivalence of the FE Scheme with a Quadratic Network

We classify data which can be represented as a vector $\mathbf{x} \in [0, B]^n$ (in our case, the size $B = 255$, and the dimension $n = 28 \times 28 = 784$) and we first build models $(q_i)_{i \in [\ell]}$ for each public label $i \in [\ell]$, such that our prediction y_{pub} for $\mathbf{x}$ is $\text{argmax}_{i \in [\ell]}\, q_i(\mathbf{x})$.

Quadratic polynomial on $\mathbb{R}^n$. The most straightforward way to use our FE scheme would be for us to learn a model $(\mathbf{Q}_i)_{i \in [\ell]} \in \left(\mathbb{R}^{n \times n}\right)^{\ell}$, which we would then round onto integers, such that $q_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{Q}_i \mathbf{x}$, $\forall i \in [\ell]$. This is an unnecessarily powerful model in the case of MNIST as it has ℓn^2 parameters ($n = 784$), and the resulting number of pairings to compute would be unreasonably large.

Linear homomorphism. The encryption algorithm of our FE scheme is linearly homomorphic with respect to the plaintext: given an encryption of $(\mathbf{x}, \mathbf{y})$ under the secret key $\mathsf{msk} := (\mathbf{s}, \mathbf{t})$, one can efficiently compute an encryption of $(\mathbf{u}^\top \mathbf{x}, \mathbf{v}^\top \mathbf{y})$ under the secret key $\mathsf{msk}' := (\mathbf{u}^\top \mathbf{s}, \mathbf{v}^\top \mathbf{t})$ for any linear combination $\mathbf{u}, \mathbf{v}$ (see proof in Appendix A.1.1). Any vector $\mathbf{v}$ is a column, and $\mathbf{v}^\top$ is a row.

Therefore, if q can be written $q(\mathbf{x}, \mathbf{y}) = (\mathbf{U}\mathbf{x})^\top \mathbf{M} (\mathbf{V}\mathbf{y})$ for all $(\mathbf{x},\, \mathbf{y})$, with $\mathbf{U}, \mathbf{V} \in \mathbb{Z}_p^{d \times n}$ projection matrices and $\mathbf{M} \in \mathbb{Z}_p^{d \times d}$, it is more efficient to first compute the encryption of $(\mathbf{U}\mathbf{x}, \mathbf{V}\mathbf{y})$ from the encryption of $(\mathbf{x}, \mathbf{y})$, and then to apply the functional decryption on these ciphertexts, because their underlying plaintexts are of reduced dimension $d < n$. This reduces the number of exponentiations from $2n^2$ to $2dn$ and the number of pairing computations from $2n^2$ to $2d^2$ for a single q_i. This is a major efficiency improvement for small d, as pairings are the main bottleneck in the computation.

Projection and quadratic polynomial on $\mathbb{R}^d$. We can use this and apply the quadratic polynomials on projected vectors: we learn $\mathbf{P} \in \mathbb{R}^{n \times d}$ and $(\mathbf{Q}_i)_{i \in [\ell]} \in \left(\mathbb{R}^{d \times d}\right)^{\ell}$, and our model is $q_i(\mathbf{x}) = (\mathbf{P}\mathbf{x})^\top \mathbf{Q}_i (\mathbf{P}\mathbf{x})$, $\forall i \in [\ell]$. We only need $2\ell d^2$ pairings and since the same $\mathbf{P}$ is used for all q_i, we only compute once the encryption of $\mathbf{P}\mathbf{x}$ from the encryption of $\mathbf{x}$. Better yet, we can also perform the pairings only once, and then compute the scores by exponentiating with different coefficients the same results of the pairings, thus only requiring $2d^2$ pairing evaluations, independently of ℓ.

Degree 2 polynomial network, with one hidden layer. To further reduce the number of pairings, we actually limit ourselves to diagonal matrices, and thus rename $\mathbf{Q}_i$ to $\mathbf{D}_i$. We find that the gain in efficiency associated with only computing $2d$ pairings is worth the small drop in accuracy. The resulting model is actually a polynomial network of degree 2 with one hidden layer of d neurons and the activation function is the square. In the following experiments we take $d = 40$.

Our final encrypted model can thus be written as $q_i(\mathbf{x}) = (\mathbf{P}\mathbf{x})^\top \mathbf{D}_i (\mathbf{P}\mathbf{x}), \forall i \in [\ell]$, where we add a bias term to $\mathbf{x}$ by replacing it with $\mathbf{x} = (1\ x_1 \ldots x_n)$.

Full network. The result of the quadratic $(q_i(\mathbf{x}))_{i \in [\ell]}$ (i.e., of the private quadratic network) is now visible in clear. As mentioned above, we cannot compose this block several times as it contains decryption, so this is currently the best that we can have as an encrypted computation with FE. Instead of simply applying the argmax to the cleartext output of this privately-evaluated quadratic network to get the label, we observe that adding more plaintext layers on top of it helps improve the overall accuracy of the main task. We have therefore a neural network composed of a private and a public part, as illustrated in Figure 4.2.

5.1.3 Experimental Results

We have assessed the performance of this network on the image dataset presented in the previous chapter about collateral learning, which can be assimilated to the MNIST dataset in terms of image format ($60,000$ gray scale 28×28 pixel images).

Accuracy. We do not report precise accuracy results on this dataset since it is non standard and hence does not have established benchmarks, but experiments conducted as part of the attacks in the previous chapter report an accuracy around 98% for the quadratic network alone, which is not altered by encryption and conversion to fixed precision.

Runtime. The runtime during the test phase is dominated by the FE scheme part which can be broken down to 4 steps: functional key generation, encryption of the input, evaluation of the function and discrete logarithm. Regarding encryption and evaluation, the main overhead comes from the exponentiations and pairings which are implemented in the crypto library charm [AGM+13]. In return, the discrete logarithm is very efficient thanks to the reduction of the weights amplitude detailed in Section 4.2.3.

Functional key generation	94 ± 5ms		Evaluation time	2.97 ± 0.07s
Encryption time	12.1 ± 0.3 s		Discrete logarithms time	24 ± 9ms

Table 5.1: Average runtime for the FE scheme using a 2,7 GHz Intel Core i7 and 16GB of RAM.

Table 5.1 shows that encryption time is longer than evaluation time, but a single encryption can be used with several decryption keys dk_{q_i} to perform multiple evaluation tasks.

5.1.4 Conclusion

Our algorithm based on quadratic functional encryption can successfully be used for evaluation of a small quadratic neural network and displays reasonable performance even on a laptop. However, to be able to use more complex neural network architectures or to study model training scenarios, functional encryption does not yet offer enough flexibility.

As a consequence, we consider another cryptographic protocol, function secret sharing, that belongs to the SMPC family and we show that it can efficiently be used for private training and inference of neural networks on sensitive data.

5.2 AriaNN: Low-Interaction Privacy-Preserving Deep Learning via Function Secret Sharing

We introduce AriaNN, a low-interaction additive secret sharing framework for private neural network training and inference. Our semi-honest 2-party computation protocol (with a trusted dealer) leverages function secret sharing [BGI15], a lightweight cryptographic protocol that allows us to achieve an efficient online phase. We design optimized primitives for the building blocks of neural networks such as ReLU, MaxPool and BatchNorm. For instance, we perform private comparison for ReLU operations with a single message of the size of the input during the online phase, and with preprocessing keys close to 4× smaller than previous work. Furthermore, we propose an extension to support n-party private federated learning.

We implement our protocol as an extensible framework based on PyTorch, that leverages CPU and GPU hardware acceleration for cryptographic and machine learning operations, and integrate it into the PySyft library. We evaluate our end-to-end system for private inference between distant servers on standard neural networks such as AlexNet, VGG16 or ResNet18, and for private training on smaller networks like LeNet. We show that computation rather than communication is the main bottleneck and that using GPUs together with reduced key size is a promising solution to overcome this barrier.

5.2.1 Introduction

With the massive improvements of cryptography techniques for secure computation over sensitive data [DPSZ12, CGGI16, KPR18], privacy-preserving machine learning [SS15, ARC19] has become practical for concrete use cases, thus encouraging public authorities to use them to protect citizens' data especially in healthcare applications [KKB18, DZ16].

However, tools are lacking to provide end-to-end solutions for institutions that have little expertise in cryptography while facing critical data privacy challenges. A striking example is hospitals, which handle large amounts of data while having relatively constrained technical teams. SMPC is a promising technique that can be efficiently integrated into machine learning workflows to ensure data and model privacy, while allowing multiple parties or institutions to participate in a joint project. In particular, it provides intrinsic shared governance: because data is secret-shared, none of the parties can decide alone to reconstruct it.

Use case

The main use case driving our work is the collaboration between a healthcare institution and an AI company. The healthcare institution, a hospital for example, acts as the data owner and the AI company as the model owner. The collaboration consists of either training the model with labelled data or using a pre-trained model to analyze unlabelled data. Training can possibly involve several data owners, as detailed in Section 5.2.5. Since the model can be a sensitive asset (in terms of intellectual property, strategic asset or regulatory and privacy issues), it cannot be trained directly on the data owner(s) machines using techniques like federated learning [KMY$^+$16, BEG$^+$19]: it could be stolen or reverse-engineered [HAPC17, FJR15].

We will assume that the parties involved in the computation are located in different regions, and that they can communicate large amounts of information over the network with a reasonable latency (70ms for example). This corresponds to the *Wide Area Network (WAN)* setting, as opposed to the *Local Area Network (LAN)* setting where parties are typically located in the same data center and communicate with low latency (typically <1ms). Second, parties are *honest-but-curious*, [Gol09, Chapter 7.2.2] and care about their reputation. Hence, they have little incentive to deviate from the original protocol, but they will use any information available in their own interest.

Contributions

By leveraging function secret sharing (FSS) [BGI15, BGI16], we propose a low-interaction framework for private deep learning which drastically reduces communication to a single round for basic machine learning operations, and achieves the first private evaluation benchmark on ResNet18 using GPUs.

- We improve upon existing work of [BGI16] on function secret sharing to design compact and ready-to-implement algorithms for tensor private comparison, which is a building block for neural networks and can be run with a single round of communication. In particular, given n the number of bits on which values are encoded, we significantly reduce the key size from roughly $n(4\lambda + n)$ to $n(\lambda + 2n)$, which is a crucial parameter as the computation time is linear in the key size.
- We show how function secret sharing can be used in machine learning and provide privacy-preserving implementations of classical layers, including ReLU, MaxPool and BatchNorm, to allow secure evaluation and training of arbitrary models on private data.
- Last, we provide a GPU implementation and a hardware-accelerated CPU implementation of our private comparison protocol[3]. As AriaNN is built over PyTorch for other tensor operations, it can run either completely on the GPU or on the CPU. We show its practicality both in LAN and WAN settings by running private inference on CIFAR-10 and Tiny Imagenet with models such as AlexNet [KSH12], VGG16 [SZ14] and ResNet18 [HZRS16], and private training on MNIST using models like LeNet.

Related work

Related work in privacy-preserving machine learning encompasses SMPC and fully homomorphic encryption (FHE) techniques.

FHE only needs a single round of interaction but does not support efficient non-linearities. For example, nGraph-HE [BLCW19] and its extensions [BCCW19] build on the SEAL library [SEA19] and provide a framework for secure evaluation that greatly improves on the CryptoNet seminal work [GBDL+16], but it resorts to polynomials (like the square) for activation functions.

SMPC frameworks usually provide faster implementations using lightweight cryptography. MiniONN [LJLA17], DeepSecure [RRK18] and XONN [RSC+19] use optimized garbled circuits [Yao86] that allow very few communication rounds, but they do not support training and alter the neural network structure to speed up execution. Other frameworks such as ShareMind [BLW08], SecureML [MZ17], SecureNN [WGC19], QUOTIENT [ASSKG19] or more recently FALCON [WTB+21] rely on additive secret sharing and allow secure model evaluation and training. They use simpler and more efficient primitives, but require a large number of rounds of communication, such as 11 in [WGC19] or $5+\log_2(n)$ in [WTB+21] (typically 10 with $n = 32$) for ReLU. ABY [DSZ15], Chameleon [RWT+18] and more recently ABY^3 [MR18], CrypTFlow [KRC+20] and [DEK21] mix garbled circuits, additive or binary secret sharing based on what is most efficient for the operations considered. However, conversion between those can be expensive and they do not support training except ABY^3. There is a current line of work including BLAZE [PS20], Trident [CRS20] and FLASH [BCPS20] which improves over ABY^3 to reduce communication overheads: BLAZE and Trident achieve for example 4 rounds of communication for ReLU.

Last, works like Gazelle [JVC18] combine FHE and SMPC to make the most of both, but conversion can also be costly.

Works on trusted execution environments are left out of the scope of this work as they require access to dedicated and expensive hardware [HSS+18].

A concurrent work from Boyle et al. [BCG+21] was made public shortly after ours. Their

[3] The code is available at `https://github.com/LaRiffle/AriaNN`.

approach also provides improvement over previous algorithms for private comparison using function secret sharing, and their implementation results in the same number of rounds than ours and similar key size (approximately $n(\lambda+n)$, where n is the number of bits to encode the value, it accounts for correctness and is typically set to 32, and λ is the security parameter and usually equals 128). However, [BCG+21] is not intended for machine learning: they only provide an implementation of ReLU, but not of MaxPool, BatchNorm, Argmax or other classic machine learning components. In addition, as they do not provide experimental benchmarks or an implementation of their private comparison, we are not able to compare it to ours in our private ML framework. They avoid the negligible error rate that we study in Section 3, which has no impact in the context of machine learning as we show.

5.2.2 Background

Notations. All values are encoded on n bits and live in $\mathbb{Z}_{2^n}$. The bit decomposition of any element x of $\mathbb{Z}_{2^n}$ into a bit string of $\{0,1\}^n$ is a bijection between $\mathbb{Z}_{2^n}$ and $\{0,1\}^n$. Therefore, bit strings generated by a pseudo random generator G are implicitly mapped to $\mathbb{Z}_{2^n}$. In addition, we interpret the most significant bit as a sign bit to map them in $[-2^{n-1}, 2^{n-1}-1]$, notably in Algorithms 2, 3, 4, 5, 6, where the modulo operation makes the conversion between n bit strings and signed integers explicit.

The notation $[\![x]\!]$ denotes 2-party additive secret sharing of x, i.e., $[\![x]\!] = ([\![x]\!]_0, [\![x]\!]_1)$ where the shares $[\![x]\!]_j$ are random in $\mathbb{Z}_{2^n}$, are held by distinct parties and verify $x = [\![x]\!]_0 + [\![x]\!]_1 \bmod 2^n$. We say indistinctly that $[\![x]\!]$ is private or secret-shared. In return, $x[i]$ refers to the i-th bit of x. The comparison operator $\leq$ is taken over the natural embedding of $\mathbb{Z}_{2^n}$ into $\mathbb{Z}$.

5.2.2.1 Function Secret Sharing

Unlike classical data secret sharing, where a shared input $[\![x]\!]$ is applied on a public f, function secret sharing applies a public input x on a private shared function $[\![f]\!]$. Shares or *keys* $([\![f]\!]_0, [\![f]\!]_1)$ of a function f satisfy $f(x) = [\![f]\!]_0(x) + [\![f]\!]_1(x) \bmod 2^n$ and they can be provided by a semi-trusted dealer. Both approaches output a secret shared result.

Let us take an example: say Alice and Bob respectively have secret shares $[\![y]\!]_0$ and $[\![y]\!]_1$ of a private input $[\![y]\!]$, and they want to compute $[\![y \leq 0]\!]$. They first mask their shares using a secret shared random mask $[\![\alpha]\!]$, by computing respectively $[\![y]\!]_0 + [\![\alpha]\!]_0$ and $[\![y]\!]_1 + [\![\alpha]\!]_1$, and then reveal these values to reconstruct $x = y + \alpha$. Next, they apply this public x on their function shares $[\![f_\alpha]\!]_j$ of $f_\alpha : x \to (x \leq \alpha)$, to obtain a secret shared output $([\![f_\alpha]\!]_0(x), [\![f_\alpha]\!]_1(x)) = [\![f_\alpha(y+\alpha)]\!] = [\![(y+\alpha) \leq \alpha]\!] = [\![y \leq 0]\!]$. They can then reveal their shares to publicly reconstruct $y \leq 0$ or use them as a private input for a subsequent computation. [BGI15, BGI16] have shown the existence of such function shares for comparison which perfectly hide y and the result. From now on, to be consistent with the existing literature, we will denote the function keys $(\mathsf{k}_0, \mathsf{k}_1) := ([\![f]\!]_0, [\![f]\!]_1)$.

Note that for a perfect comparison, the value $y + \alpha$ should not wrap around and become negative. Because typically values of y used in practice in machine learning are small compared to the n-bit encoding amplitude with typically $n = 32$, the failure rate is less than one comparison in a million, as detailed in Section 5.2.3.2.

5.2.2.2 2-Party Computation in the Preprocessing Model

Preprocessing is performed during an offline phase by a trusted third party that builds and distributes the function keys to the 2 parties involved in future computation. This is standard in function secret sharing, and as mentioned by [BGI19], in the absence of such a trusted dealer, the keys can alternatively be generated via an interactive secure protocol that is executed offline, before the inputs are known. This setup can also be found in other privacy-preserving machine learning frameworks including SecureML [MZ17]. This trusted dealer is not active during the online phase, and he is unaware of the computation the 2 parties intend to execute. In particular, as we are in the honest-but-curious model, it is assumed that no party colludes with the dealer. In practice, such a third party would typically be an institution concerned about its reputation, and it could be easy to check that preprocessed material is correct using a *cut-and-choose* technique [ZHKS16]. For example, the third party produces n keys for private comparison. The 2 parties willing to do the private computation randomly check some of them: they extract from their keys s_0, s_1 and also reconstruct α from $[\![\alpha]\!]_j$, $j \in \{0, 1\}$. They can then derive the computations of KeyGen and verify that the correlated randomness of the keys was correct. They can then use the remaining keys for the private computation.

5.2.2.3 Security Model of the Function Secret Sharing Protocol

We consider security against *honest-but-curious* adversaries, i.e., parties following the protocol but trying to infer as much information as possible about others' input or function share. This is a standard security model in many SMPC frameworks [BLW08, BELO16, RWT$^+$18, WGC19] and is aligned with our main use case: parties that would not follow the protocol would face major backlash for their reputation if they got caught. The security of our protocols relies on indistinguishability of the function shares, which informally means that the shares received by each party are computationally indistinguishable from random strings. More formally, we introduce the following definitions from [BGI16].

Definition 10 (FSS: Syntax)**.** A (2-party) *function secret sharing (FSS) scheme* is a pair of algorithms (KeyGen, Eval) with the following syntax:

- $\mathsf{KeyGen}(1^\lambda, \hat{f})$ is a PPT *key generation* algorithm, which on input 1^λ (security parameter) and $\hat{f} \in \{0, 1\}^*$, description of a function $f : \mathbb{Z}_{2^n} \to \mathbb{Z}_{2^n}$, outputs a pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$.
- $\mathsf{Eval}(i, \mathsf{k}_i, x)$ is a polynomial-time *evaluation* algorithm, which on input $i \in \{0, 1\}$ (party index), k_i (the i-th function key) and $x \in \mathbb{Z}_{2^n}$, outputs $[\![f]\!]_i(x) \in \mathbb{Z}_{2^n}$ (the i-th share of $f(x)$).

Definition 11 (FSS: Correctness and Security)**.** We say that (KeyGen, Eval) as in Definition 10 is a *FSS scheme for a family of function* $\mathcal{F}$ if it satisfies the following requirements:

- **Correctness:** For all $f : \mathbb{Z}_{2^n} \to \mathbb{Z}_{2^n} \in \mathcal{F}$, $\hat{f}$ a description of f, and $x \in \mathbb{Z}_{2^n}$, if $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{KeyGen}(1^\lambda, \hat{f})$ then $\Pr[\mathsf{Eval}(0, \mathsf{k}_0, x) + \mathsf{Eval}(1, \mathsf{k}_1, x) = f(x)] = 1$.
- **Security:** For each $i \in \{0, 1\}$, there is a PPT algorithm Sim_i (simulator), such that for every infinite sequence $(\hat{f}_j)_{j\in\mathbb{N}}$ of descriptions of functions from $\mathcal{F}$ and polynomial size input sequence x_j for f_j, the outputs of the following experiments Real and Ideal are computationally indistinguishable:
  - $\mathsf{Real}_j : (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{KeyGen}(1^\lambda, \hat{f}_j)$; Output k_i
  - Ideal_j : Output $\mathsf{Sim}_i(1^\lambda)$

[BGI16] has proved the existence of efficient FSS schemes in particular for equality. Such protocols and the ones that we derive from this work are proved to be secure against semi-honest adversaries, and as mentioned by [BGI19], they could be extended to guarantee *security with abort* against malicious adversaries using MAC authentication [DPSZ12], which means that the protocol would abort if parties deviated from it.

5.2.2.4 General Security Guarantees and Threats

The 2-party interaction for private inference, i.e. when the model is already trained, is an example of Encrypted Machine Learning as a Service (EMLaaS). In this scenario, as stated above, even a malicious model owner could not disclose information about the private inputs or predictions. However, it could use a different model where the weights have been modified to make poor or biased predictions. It is difficult for the data owner to realize that the model owner is misbehaving or using a model whose performance is inferior to what it claims, and this is an issue users also have with standard Machine Learning as a Service (MLaaS). Proving that the computation corresponds to a *certified* given model would require to commit the model and would be costly. On the other hand, the information obtained by the data owner about the model (i.e. the prediction on a given input) is the same as in MLaaS. Model inversion techniques [ZJP+20] can leverage multiple calls to the model to try to build a new model with similar performance. There are not many defenses against this, except limiting access to the model, which is usually the case in MLaaS where data owners are given a quota of requests. Also, attacks like membership inference [SSSS17] or reverse-engineering [FJR15, HAPC17] methods could be used to unveil information about the dataset on which the model was originally trained. Using differential privacy [DR+14, ACG+16] during the initial training of the model can provide some guarantees [RRL+18] against these threats, but it has a trade-off between privacy and utility as differentially private models usually have poorer performance.

Beyond evaluation, the case of fully-encrypted training can also expose the parties to some threats. The most common one is *data poisoning* and consists of the data owner undermining the training by providing irrelevant data or labels that are wrong or biased [BVH+20]. This attack however does not affect privacy. In return, if the model owner gets the final model in plaintext at the end of the training, the privacy of the data owner is at risk because the model owner could use the aforementioned techniques to get information about the training data. Using differential privacy during the private training is important to mitigate this privacy leakage, and should also be applied in a n-party training setting.

All these threats must be taken seriously when building production-ready systems. However, they are independent of the function secret sharing protocol and can be addressed separately by combining our work with differential privacy libraries for deep learning.

5.2.3 Function Secret Sharing Primitives

Our algorithms for private equality and comparison are built on top of the work of [BGI16], so the security assumptions are the same as in this article. We first present an algorithm for equality which is very close to the one of [BGI16] but which is used as a basis to build the comparison protocol. We then describe the private comparison protocol, which improves over the work of [BGI16] on Distributed Interval Functions (DIF) by specializing on the operations needed for neural network evaluation or training. In particular, we are able to reduce the function key size from roughly $n(4\lambda + n)$ to $n(\lambda + 2n)$.

5.2.3.1 Equality Test

We start by describing private equality as introduced by [BGI16], which is slightly simpler than comparison and gives useful hints about how comparison works. The equality test consists in comparing a public input x to a private value α. Evaluating the input using the function keys can be viewed as walking a binary tree of depth n, where n is the number of bits of the input (typically 32). Among all the possible paths, the path from the root down to α is called the *special path*. Figure 5.2 illustrates this tree and provides a compact representation which is used by our protocol, where we do not detail branches for which all leaves are 0. Evaluation goes as follows: two evaluators are each given a function key which includes a distinct initial random

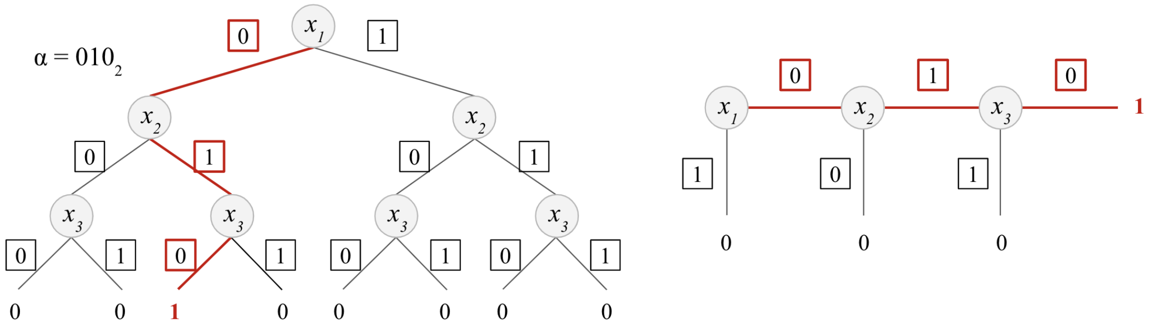

Figure 5.2: *(Left)* Binary decision tree with the special path for $n = 3$. Given an input $x = x[1] \dots x[n]$, at each level i, one should take the path labeled by the value in the square equal to the bit value $x[i]$. *(Right)* Flat representation of the tree.

state $(s, t) \in \{0,1\}^\lambda \times \{0,1\}$. Each evaluator starts from the root, at each step i goes down one node in the tree and updates his state depending on the bit $x[i]$ using a common *correction word* $CW^{(i)} \in \{0,1\}^{2(\lambda+1)}$ from the function key. At the end of the computation, each evaluator outputs t. As long as $x[i] = \alpha[i]$, the evaluators stay on the special path and because the input x is public and common to them, they both follow the same path. If a bit $x[i] \neq \alpha[i]$ is met, they leave the special path and should output 0 ; else, they stay on it all the way down, which means that $x = \alpha$ and they should output 1.

Intuition. The main idea is that while they are on the special path, evaluators should have states (s_0, t_0) and (s_1, t_1) respectively, such that s_0 and s_1 are i.i.d. and $t_0 \oplus t_1 = 1$. When they leave it, the correction word should act to have $s_0 = s_1$ but still indistinguishable from random and $t_0 = t_1$, which ensures $t_0 \oplus t_1 = 0$. To reconstruct the result in plaintext, each evaluator should output its t_j and the result will be given by $t_0 \oplus t_1$. The formal description of the protocol is given below and is composed of two parts: first, in Algorithm 2, the KeyGen algorithm consists of a preprocessing step to generate the functions keys, and then, in Algorithm 3, Eval is run by two evaluators to perform the equality test. It takes as input the private share held by each evaluator and the function key that they have received. They use $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$, a pseudorandom generator (PRG), where the output set is $\{0,1\}^{\lambda+1} \times \{0,1\}^{\lambda+1}$, and operations modulo 2^n implicitly convert back and forth n-bit strings into integers.

```
Initialisation: Sample random $\alpha \xleftarrow{\$} \mathbb{Z}_{2^n}$
Sample random $s_j^{(1)} \xleftarrow{\$} \{0,1\}^\lambda$ and set $t_j^{(1)} \leftarrow j$, for $j = 0, 1$
for $i = 1..n$ do
    $(s_j^{\mathsf{L}} \,||\, t_j^{\mathsf{L}}, s_j^{\mathsf{R}} \,||\, t_j^{\mathsf{R}}) \leftarrow G(s_j^{(i)}) \in \{0,1\}^{\lambda+1} \times \{0,1\}^{\lambda+1}$, for $j = 0, 1$
    if $\alpha[i]$ then $cw^{(i)} \leftarrow (0^\lambda \,||\, 0, s_0^{\mathsf{L}} \oplus s_1^{\mathsf{L}} \,||\, 1)$ else $cw^{(i)} \leftarrow (s_0^{\mathsf{R}} \oplus s_1^{\mathsf{R}} \,||\, 1, 0^\lambda \,||\, 0)$ ;
    $CW^{(i)} \leftarrow cw^{(i)} \oplus G(s_0^{(i)}) \oplus G(s_1^{(i)}) \in \{0,1\}^{\lambda+1} \times \{0,1\}^{\lambda+1}$
    $state_j \leftarrow G(s_j^{(i)}) \oplus (t_j^{(i)} \cdot CW^{(i)}) = (state_{j,0}, state_{j,1})$, for $j = 0, 1$
    Parse $s_j^{(i+1)} \,||\, t_j^{(i+1)} = state_{j,\alpha[i]} \in \{0,1\}^{\lambda+1}$, for $j = 0, 1$
$CW^{(n+1)} \leftarrow (-1)^{t_1^{(n+1)}} \cdot (1 - s_0^{(n+1)} + s_1^{(n+1)}) \bmod 2^n$
return $\mathsf{k}_j \leftarrow [\![\alpha]\!]_j \,||\, s_j^{(1)} \,||\, CW^{(1)} \,||\, \cdots \,||\, CW^{(n+1)}$, for $j = 0, 1$
```

Algorithm 2: KeyGen: function key generation for equality (from [BGI16])

Correctness. Intuitively, the correction words $CW^{(i)}$ are built from the expected state of each evaluator on the special path, i.e., the state that each should have at each node i if it is on

```
Input: $(j, \mathsf{k}_j, [\![y]\!]_j)$ where $j \in \{0,1\}$ refers to the evaluator id
Parse $\mathsf{k}_j$ as $[\![\alpha]\!]_j \mid\mid s^{(1)} \mid\mid CW^{(1)} \mid\mid \cdots \mid\mid CW^{(n+1)}$
Publish $[\![\alpha]\!]_j + [\![y]\!]_j \bmod 2^n$ and get revealed $x = \alpha + y \bmod 2^n$
Let $t^{(1)} \leftarrow j$
for $i = 1..n$ do
    $state \leftarrow G(s^{(i)}) \oplus (t^{(i)} \cdot CW^{(i)}) = (state_0, state_1)$
    Parse $s^{(i+1)} \mid\mid t^{(i+1)} = state_{x[i]}$
return $[\![T]\!]_j \leftarrow (-1)^j \cdot \left(t^{(n+1)} \cdot CW^{(n+1)} + s^{(n+1)}\right) \bmod 2^n$
```

Algorithm 3: Eval: evaluation of the function key for the equality test $y = 0$ (from [BGI16])

the special path given some initial state. During evaluation, a correction word is applied by an evaluator only when it has $t = 1$. Hence, on the special path, the correction is applied only by one evaluator at each bit. If at step i, the evaluator stays on the special path, the correction word compensates the current states of both evaluators by xor-ing them with themselves and re-introduces a pseudorandom value s (either $s_0^{\mathsf{R}} \oplus s_1^{\mathsf{R}}$ or $s_0^{\mathsf{L}} \oplus s_1^{\mathsf{L}}$), which means the xor of their states is now $(s, 1)$ but those states are still indistinguishable from random.

On the other hand, if $x[i] \neq \alpha[i]$, the new state takes the other half of the correction word, so that the xor of the two evaluators states is (0, 0). From there, they have the same states and both have either $t = 0$ or $t = 1$. They will continue to apply the same corrections at each step and their states will remain the same, meaning that $t_0 \oplus t_1 = 0$. A final computation is performed to obtain a shared $[\![T]\!]$ modulo 2^n of the result bit $t = t_0 \oplus t_1 \in \{0,1\}$.

Security. From the privacy point of view, when the seed s is random, $G(s)$ is indistinguishable from random (this is a pseudorandom bit-string). Each half is used either in the cw or in the next state, but not both. Therefore, the correction words $CW^{(i)}$ do not contain information about the expected states and for $j = 0, 1$, the output k_j is independently uniformly distributed with respect to α and $s_{1-j}^{(1)}$, in a computational way. As a consequence, at the end of the evaluation, for $j = 0, 1$, $[\![T]\!]_j$ also follows a distribution independent of α. Until the shared values are reconstructed, even a malicious adversary cannot learn anything about α nor the inputs of the other player.

Implementation. Function keys should be computed by a third party dealer and sent to the evaluators in advance, which requires one extra communication of the size of the keys. We use the trick of [BGI16] to reduce the size of each correction word in the keys, from $2(1+\lambda)$ to $(2+\lambda)$ by reusing the pseudo-random λ-bit string dedicated to the state used when leaving the special path for the state used for staying onto it, since for the latter state the only constraint is the pseudo-randomness of the bitstring. Regarding the PRG, we use a Matyas-Meyer-Oseas one-way compression function with an AES block cipher, as in [KOS16] or [WYG$^+$17]. We concatenate several fixed key block ciphers to achieve the desired output length: $G(x) = E_{k_1}(x) \oplus x \mid\mid E_{k_2}(x) \oplus x \mid\mid \ldots$. Using AES helps us to benefit from hardware acceleration: we used the `aesni` Rust library for CPU execution and the `csprng` library of PyTorch for GPU. More details about implementation can be found in Appendix A.2.2.

5.2.3.2 Comparison

Our main contribution to the function secret sharing scheme is for the comparison function, which is intensively used in neural network to build non-polynomial activation functions like ReLU: we build on the idea of the equality test to provide a synthetic and efficient protocol whose structure is very close to the previous one, and improves upon the former DIF scheme of [BGI16] by significantly reducing the key size.

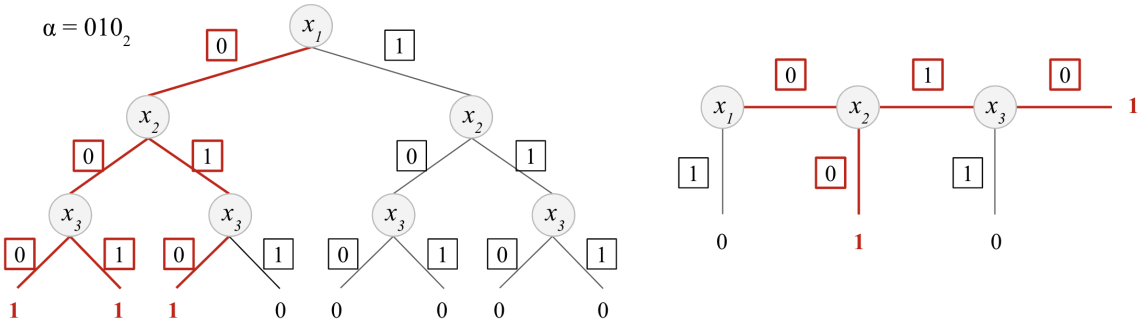

Figure 5.3: *(Left)* Binary decision tree with all the paths corresponding to $x \leq \alpha$ for n=3. *(Right)* Flat representation of the tree.

Intuition. Instead of seeing the special path as a simple path, we can see it as a frontier for the zone in the tree where $x \leq \alpha$. To evaluate $x \leq \alpha$, we could evaluate all the paths on the left of the special path and then sum up the results, but this is highly inefficient as it requires exponentially many evaluations. The key idea here is to evaluate all these paths at the same time, noting that each time one leaves the special path, it either falls on the left side (i.e., $x < \alpha$) or on the right side (i.e., $x > \alpha$). Hence, we only need to add an extra step at each node of the evaluation, where depending on the bit value $x[i]$, we output a leaf label which is 1 only if $x[i] < \alpha[i]$ and all previous bits are identical. Only one label between the final label (which corresponds to $x = \alpha$) and the leaf labels can be equal to one, because only a single path can be taken. Therefore, evaluators will return the sum of all the labels to get the final output.

Correctness of the comparison protocol. Consider (k_0, k_1) generated by KeyGen (Algorithm 4) with a random offset $\alpha \in \mathbb{Z}_{2^n}$. Consider a public input $x \in \mathbb{Z}_{2^n}$. Let us show that $\mathsf{Eval}(0, k_0, x) + \mathsf{Eval}(1, k_1, x) = (x \leq \alpha) \mod 2^n$, where $(x \leq \alpha) \in \{0, 1\}$. We add a subscript 0 or 1 to the variables of Algorithm 5 to identify the party to which they belong.

Consider $i \in [1, n]$ such that the evaluators remained on the special path until i (*i.e.* $\forall j < i, x[j] = \alpha[j]$). In particular, $G(s_0^{(i)}) \oplus G(t_0^{(i)} \cdot CW^{(i)}) \oplus (s_1^{(i)}) \oplus (t_1^{(i)} \cdot CW^{(i)}) = cw^{(i)}$. Let us study the 4 possible cases and show that 1) $(out_{i,0} + out_{i,1} \mod 2^n) \in \{0, 1\}$; 2) $out_{i,0} + out_{i,1} = 1 \mod 2^n \iff x[i] < \alpha[i]$; and 3) the evaluators stay on the special path if and only if $x[i] = \alpha[i]$.

- If $x[i] = 0$, we keep the left part of *state*′ at line 4.
  - If $\alpha[i] = 1$, we have $\tau_0^{(i+1)} \oplus \tau_1^{(i+1)} = 1$. Thanks to line 10 of KeyGen, we have $out_{i,0} + out_{i,1} = (\tau_0^{(i+1)} - \tau_1^{(i+1)}) \cdot CW_{leaf}^{(i)} + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = (1 - 2\tau_1^{(i+1)})(-1)^{\tau_1^{(i+1)}}(\sigma_1^{(i+1)} - \sigma_0^{(i+1)} + 1) + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = 1 \mod 2^n$. We also have $t_0^{(i+1)} \oplus t_1^{(i+1)} = 0$ and $s_0^{(i+1)} \oplus s_1^{(i+1)} = 0$, so the evaluators leave the special path.
  - If $\alpha[i] = 0$, we use line 5 of KeyGen to generate $cw^{(i)}$, so $\sigma_0^{(i+1)} \oplus \sigma_1^{(i+1)} = 0$ and $\tau_0^{(i+1)} \oplus \tau_1^{(i+1)} = 0$. Hence, $out_{i,0} + out_{i,1} = (\tau_0^{(i+1)} - \tau_1^{(i+1)}) \cdot CW_{leaf}^{(i)} + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = 0 \mod 2^n$. We also have $t_0^{(i+1)} \oplus t_1^{(i+1)} = 1$ and $s_0^{(i+1)}, s_1^{(i+1)}$ stay on the special path.
- If $x[i] = 1$, we keep the right part of *state*′ at line 4.
  - If $\alpha[i] = 1$, we use line 4 of KeyGen to generate $cw^{(i)}$, so $\sigma_0^{(i+1)} \oplus \sigma_1^{(i+1)} = 0$ and $\tau_0^{(i+1)} \oplus \tau_1^{(i+1)} = 0$. Hence, similarly as the case where $(x[i], \alpha[i]) = (0, 0)$, we have $out_{i,0} + out_{i,1} = 0 \mod 2^n$ and the evaluators stay on the special path.

- If $\alpha[i] = 0$, we use line 5 of KeyGen and get $cw^{(i)} = ((s_0^{\mathsf{R}} \oplus s_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0), (0^\lambda \mid\mid 0, \sigma_0^{\mathsf{L}} \oplus \sigma_1^{\mathsf{L}} \mid\mid 1))$. We keep the right part of $state'$ at line 9 of KeyGen for $CW_{leaf}^{(i)}$. We use the same right part at line 5 of Eval, so we have $\tau_0^{(i+1)} \oplus \tau_1^{(i+1)} = 1$. Finally, $out_{i,0} + out_{i,1} = (\tau_0^{(i+1)} - \tau_1^{(i+1)}) \cdot CW_{leaf}^{(i)} + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = (1 - 2\tau_1^{(i+1)})(-1)^{\tau_1^{(i+1)}}(\sigma_1^{(i+1)} - \sigma_0^{(i+1)} + 0) + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = 0 \mod 2^n$. We also have $t_0^{(i+1)} \oplus t_1^{(i+1)} = 0$ and the evaluators leave the special path.

If the evaluators leave the special path at step i, their bistrings remain equal until the end of the evaluation: $\forall j \in [i, n+1], s_0^{(j)} = s_1^{(j)}$ and $\sigma_0^{(j)} = \sigma_1^{(j)}$, so $\forall j \in [i, n+1], out_{j,0} + out_{j,1} = 0 \mod 2^n$.

Finally, if the evaluators never leave the special path (*i.e.* $x = \alpha$), we have $\forall j \in [1, n], out_{j,0} + out_{j,1} = 0 \mod 2^n$, and $out_{n+1,0} + out_{n+1,1} = 1$. Indeed, step $n+1$ is identical to the equality case (Algorithm 3).

In the end, the sum $[\![T]\!]_j$ of the out_i's is a share of 1 either if out_{n+1} was a share of 1 (i.e. $x = \alpha$) or if one of the other out_i was a share of 1, which is possible only if $\alpha[i] = 1$ and $x[i] < \alpha[i]$ (i.e. $x < \alpha$). Otherwise (i.e. $x > \alpha$), $[\![T_j]\!]$ is a share of 0.

```
   Initialisation: Sample random $\alpha \xleftarrow{\$} \mathbb{Z}_{2^n}$
   Sample random $s_j^{(1)} \xleftarrow{\$} \{0,1\}^\lambda$ and set $t_j^{(1)} \leftarrow j$, for $j = 0, 1$
1  for $i = 1..n$ do
2      for $j = 0, 1$ do
3          $((s_j^{\mathsf{L}} \mid\mid t_j^{\mathsf{L}}, s_j^{\mathsf{R}} \mid\mid t_j^{\mathsf{R}}), (\sigma_j^{\mathsf{L}} \mid\mid \tau_j^{\mathsf{L}}, \sigma_j^{\mathsf{R}} \mid\mid \tau_j^{\mathsf{R}})) \leftarrow G(s_j^{(i)}) \in$
             $\{0,1\}^{\lambda+1} \times \{0,1\}^{\lambda+1} \times \{0,1\}^{n+1} \times \{0,1\}^{n+1}$
4      if $\alpha[i]$ then $cw^{(i)} \leftarrow ((0^\lambda \mid\mid 0, s_0^{\mathsf{L}} \oplus s_1^{\mathsf{L}} \mid\mid 1), (\sigma_0^{\mathsf{R}} \oplus \sigma_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0))$
5      else $cw^{(i)} \leftarrow ((s_0^{\mathsf{R}} \oplus s_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0), (0^\lambda \mid\mid 0, \sigma_0^{\mathsf{L}} \oplus \sigma_1^{\mathsf{L}} \mid\mid 1))$;
6      $CW^{(i)} \leftarrow cw^{(i)} \oplus G(s_0^{(i)}) \oplus G(s_1^{(i)})$
7      for $j = 0, 1$ do
8          $state_j \leftarrow G(s_j^{(i)}) \oplus (t_j^{(i)} \cdot CW^{(i)}) = ((state_{j,0}, state_{j,1}), (state'_{j,0}, state'_{j,1}))$
9          Parse $s_j^{(i+1)} \mid\mid t_j^{(i+1)} = state_{j,\alpha[i]}$ and $\sigma_j^{(i+1)} \mid\mid \tau_j^{(i+1)} = state'_{j,1-\alpha[i]}$
10     $CW_{leaf}^{(i)} \leftarrow (-1)^{\tau_1^{(i+1)}} \cdot \left(\alpha[i] - \sigma_0^{(i+1)} + \sigma_1^{(i+1)}\right) \mod 2^n$
11 $CW_{leaf}^{(n+1)} \leftarrow (-1)^{t_1^{(n+1)}} \cdot \left(1 - s_0^{(n+1)} + s_1^{(n+1)}\right) \mod 2^n$
12 return $\mathsf{k_j} \leftarrow [\![\alpha]\!]_j \mid\mid s_j^{(1)} \mid\mid (CW^{(i)})_{i=1..n} \mid\mid (CW_{leaf}^{(i)})_{i=1..n+1}$, for $j = 0, 1$
```

Algorithm 4: KeyGen: function key generation for comparison $x \leq \alpha$ (new)

Failure rate of the sign protocol. Algorithm 6 details how we build a sign protocol thanks to our comparison primitive (Algorithm 5), following the secret sharing workflow introduced in Section 5.2.2.1. Our sign protocol can fail if $y + \alpha$ wraps around and becomes negative. We cannot act on α because it must be completely random to act as a perfect mask and to make sure the revealed $x = y + \alpha \mod 2^n$ does not leak any information about y, but the smaller y is, the lower the error probability will be. [BGI19] suggests a method which uses 2 invocations of the protocol to guarantee perfect correctness but because it incurs an important runtime overhead, we rather show that the failure rate of our comparison protocol is very small and is reasonable in contexts that tolerate a few mistakes, as in machine learning. Consider $y \in [-2^{n-1}, 2^{n-1} - 1]$, a pair of comparison keys (k_0, k_1), and note $\widehat{\mathsf{Sign}}(y) := \mathsf{Sign}(0, k_0, [\![y]\!]_0) + \mathsf{Sign}(1, k_1, [\![y]\!]_1) \mod 2^n$ the reconstructed result of the sign protocol. We have $\Pr\left[\widehat{\mathsf{Sign}}(y) \neq \mathbb{1}[y \leq 0]\right] = |y|/2^n \leq Y/2^n$ where Y is the maximum amplitude for $|y|$.

```
Input: $(j, \mathsf{k}_j, x)$ where $j \in \{0,1\}$ refers to the evaluator id
Parse $\mathsf{k}_j$ as $[\![\alpha]\!]_j \;||\; s^{(1)} \;||\; (CW^{(i)})_{i=1..n} \;||\; (CW^{(i)}_{leaf})_{i=1..n+1}$
Let $t^{(1)} \leftarrow j$
for $i = 1..n$ do
    $state \leftarrow G(s^{(i)}) \oplus (t^{(i)} \cdot CW^{(i)}) = ((state_0, state_1), (state'_0, state'_1))$
    Parse $s^{(i+1)} \;||\; t^{(i+1)} = state_{x[i]}$ and $\sigma^{(i+1)} \;||\; \tau^{(i+1)} = state'_{x[i]}$
    $out_i \leftarrow (-1)^j \cdot \left(\tau^{(i+1)} \cdot CW^{(i)}_{leaf} + \sigma^{(i+1)}\right) \bmod 2^n$
$out_{n+1} \leftarrow (-1)^j \cdot \left(t^{(n+1)} \cdot CW^{(n+1)}_{leaf} + s^{(n+1)}\right) \bmod 2^n$
return $[\![T]\!]_j \leftarrow \sum_i out_i \bmod 2^n$
```

Algorithm 5: Eval: evaluation of the function key for comparison $x \leq \alpha$ (new)

```
Input: $(j, \mathsf{k}_j, [\![y]\!]_j)$ where $j \in \{0,1\}$ refers to the evaluator id
Parse the first $n$ bits of $\mathsf{k}_j$ as $[\![\alpha]\!]_j$
Publish $[\![\alpha]\!]_j + [\![y]\!]_j$ and get revealed $x = \alpha + y \bmod 2^n$
return $[\![T]\!]_j \leftarrow \mathsf{Eval}(j, k_j, x)$
```

Algorithm 6: Sign: protocol for $\mathsf{sign}([\![y]\!])$

We quantify this failure rate on real world examples, namely on Network-2 and on the 64×64 Tiny Imagenet version of VGG16, with a fixed precision of 3 decimals, and find respective failure rates of 1 in 4 millions comparisons and 1 in 100 millions comparisons, which is low compared to the number of comparisons needed for an evaluation, respectively $\sim$ 10K and $\sim$ 1M. In practice, such error rates do not affect the model accuracy, as Table 5.5 shows.

Security. We prove the security of the protocol by following the same process than [BGI16]. We first prove that each party's key k_j is pseudorandom. This is done via a sequence of hybrid distributions, where in each step we replace two correction words $CW^{(i)}$ and $CW^{(i)}_{leaf}$ within the key from being honestly generated to being random. In the initial game, all the correction words are as in the real distribution, and in the last game, they are all random. As every gaps are indistinguishable for any polynomially-bounded adversary, the real distribution is indistinguishable from random: this proves the pseudo-randomness of the keys.

The high-level argument for security will go as follows. Each party $j \in \{0,1\}$ begins with a share $[\![\alpha]\!]_j$ and a random seed $s_j^{(1)}$ that are completely unknown to the other party. In each level of key generation (for $i = 1$ to n), the parties apply a PRG to their seed $s_j^{(i)}$ to generate 8 items: namely, 2 seeds $s_j^{\mathsf{L}}, s_j^{\mathsf{R}}$, 2 bits $t_j^{\mathsf{L}}, t_j^{\mathsf{R}}$, 2 n-bits values $\sigma_j^{\mathsf{L}}, \sigma_j^{\mathsf{R}}$ and 2 other bits $\tau_j^{\mathsf{L}}, \tau_j^{\mathsf{R}}$. This process is always done on a seed which appears completely random given the view of the other party. Hence, the security of the PRG guarantees that the 8 resulting values appear similarly random given the view of the other party. The ith level correction word $CW^{(i)}$ will "use up" the secret randomness of 3 of the 4 first pieces: the two bits $t_j^{\mathsf{L}}, t_j^{\mathsf{R}}$, and the seed s_j^{Lose} corresponding to the direction exiting the special path i.e. $\mathsf{Lose} = \mathsf{L}$ if $\alpha[i] = 1$ and $\mathsf{Lose} = \mathsf{R}$ if $\alpha[i] = 0$. However, given this $CW^{(i)}$, the remaining seed s_j^{Keep} for $\mathsf{Keep} \neq \mathsf{Lose}$ is still unpredictable to the other party, as it is kept hidden. Similarly, the ith level correction word $CW^{(i)}_{leaf}$ uses up the secret randomness of the 4 last pieces, $\sigma_j^{\mathsf{L}}, \sigma_j^{\mathsf{R}}$ and $\tau_j^{\mathsf{L}}, \tau_j^{\mathsf{R}}$, and appears random given the view of the other party. The argument is then continued in similar fashion to the next level, which uses s_j^{Keep} as an input to the PRG.

For each $i \in \{0, 1, \ldots, n+1\}$, we will consider a hybrid distribution Hyb_i defined roughly as follows, for $j \in \{0,1\}$:

1. $s_j^{(1)} \xleftarrow{\$} \{0,1\}^\lambda$ chosen at random (honestly), and $t_j^{(1)} = j$.
2. $CW^{(1)}, \ldots, CW^{(i)} \leftarrow \{0,1\}^{2(\lambda+n+2)}$ and $CW_{leaf}^{(1)}, \ldots, CW_{leaf}^{(i)} \leftarrow \{0,1\}^n$ chosen at random.
3. For $k < i$, $s_j^{(k+1)} || t_j^{(k+1)}, \sigma_j^{(k+1)} || \tau_j^{(k+1)}$ computed honestly, as a function of $s_j^{(0)} \mid\mid t_j^{(0)}$ and $CW^{(1)}, \ldots, CW^{(k)}$.
4. For i, the other party's seed $s_{1-j}^{(i)} \leftarrow \{0,1\}^\lambda$ is chosen at random, $t_{1-j}^{(i)} = 1 - t_j^{(i)}$, $\sigma_{1-j}^{(i)} = \sigma_j^{(i)}$, and $\tau_{1-j}^{(i)} = \tau_j^{(i)}$.
5. For $k \geq i$: the remaining values $s_j^{(k+1)} \mid\mid t_j^{(k+1)}$, $s_{1-j}^{(k+1)} \mid\mid t_{1-j}^{(k+1)}$, $CW^{(k)}$, $\sigma_j^{(k+1)} \mid\mid \tau_j^{(k+1)}$, $\sigma_{1-j}^{(k+1)} \mid\mid \tau_{1-j}^{(k+1)}$, $CW_{leaf}^{(k)}$ all computed honestly, as a function of the previously chosen values.
6. The output of the experiment is $\mathsf{k}_j := [\![\alpha]\!]_j \mid\mid s_j^{(1)} \mid\mid (CW^{(i)})_{i=1..n} \mid\mid (CW_{leaf}^{(i)})_{i=1..n+1}$.

Hyb_i is formally described in Algorithm 7. When $i = 0$, the algorithm corresponds to the honest key generation, while when $i = n+1$, it generates a completely random key. We only need to prove that for any $i \in \{1, \ldots n+1\}$, Hyb_{i-1} and Hyb_i are indistinguishable based on the security of our PRG.

More precisely, let us first consider $i \leq n$.

Claim 5.2.1. *There exists a polynomial p' such that for any (T, ϵ_{PRG})-secure pseudorandom generator G, then for every $i \leq n$, $j \in \{0,1\}$, and every non-uniform adversary $\mathcal{A}$ running in time $T' \leq T - p'(\lambda)$, it holds that*

$$|P[\mathsf{k}_j \leftarrow \mathsf{Hyb}_{i-1}(1^\lambda, j); c \leftarrow \mathcal{A}(1^\lambda, \mathsf{k}_j) : c = 1] \ - P[\mathsf{k}_j \leftarrow \mathsf{Hyb}_i(1^\lambda, j); c \leftarrow \mathcal{A}(1^\lambda, \mathsf{k}_j) : c = 1]| < \epsilon_{PRG}$$

Proof. Let's fix $i \in \{1, \ldots n\}$, $j \in \{0,1\}$. Let $\mathcal{A}$ be a Hyb-distinguishing adversary with advantage ϵ for these values. We use $\mathcal{A}$ to construct a corresponding PRG adversary $\mathcal{B}$. Recall that in the PRG challenge for G, the adversary $\mathcal{B}$ is given a value r that is either computed by sampling a seed $s \leftarrow \{0,1\}^\lambda$ and computing $r = G(s)$, or is sampled truly at random $r \leftarrow \{0,1\}^{2(\lambda+n+2)}$. Algorithm 8 describes the PRG challenge of $\mathcal{B}$ embedded in the Hyb-distinguishing challenge of $\mathcal{A}$.

Now, consider $\mathcal{B}$'s success in the PRG challenge as a function of $\mathcal{A}$'s success in distinguishing Hyb_{i-1} from Hyb_i. This means that if $\mathcal{A}$ succeeds, then $\mathcal{B}$ will succeed at its challenge, which implies Claim 5.2.1. If, in Algorithm 8, r is computed *pseudorandomly* using the PRG, then it is clear the generated $\mathsf{k_j}$ is distributed as $\mathsf{Hyb}_{i-1}(1^\lambda, j)$.

It remains to show that if r was sampled at random then the generated $\mathsf{k_j}$ is distributed as $\mathsf{Hyb}_i(1^\lambda, j)$. That is, if r is random, then the corresponding computed values of $s_{1-j}^{(i+1)}$, $CW^{(i)}$ and $CW_{leaf}^{(i)}$ are distributed *randomly* conditioned on the values of $s_j^{(1)} \mid\mid t_j^{(1)} \mid\mid (CW^{(i)})_{i=1..i-1} \mid\mid (CW_{leaf}^{(i)})_{i=1..i-1}$, and the value of $t_{1-j}^{(i)}$ is given by $1 - t_j^{(i)}$. Note that all remaining values (for $k > i$) are computed as a function of the values computed up to step i.
First, consider $CW^{(i)}$, which is computed as such:

$$CW^{(i)} = cw^{(i)} \oplus G(s_j^{(i)}) \oplus r$$

In particular, when $\alpha[i] = 1$:

$$\begin{aligned}
cw^{(i)} \oplus r =& ((0^\lambda \mid\mid 0, s_0^{\mathsf{L}} \oplus s_1^{\mathsf{L}} \mid\mid 1), (\sigma_0^{\mathsf{R}} \oplus \sigma_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0)) \\
& \oplus ((s_{1-j}^{\mathsf{L}} \mid\mid t_{1-j}^{\mathsf{L}}, s_{1-j}^{\mathsf{R}} \mid\mid t_{1-j}^{\mathsf{R}}), (\sigma_{1-j}^{\mathsf{L}} \mid\mid \tau_{1-j}^{\mathsf{L}}, \sigma_{1-j}^{\mathsf{R}} \mid\mid \tau_{1-j}^{\mathsf{R}})) \\
=& ((0^\lambda \mid\mid 0, s_j^{\mathsf{L}} \mid\mid 1), (\sigma_j^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0)) \\
& \oplus ((s_{1-j}^{\mathsf{L}} \mid\mid t_{1-j}^{\mathsf{L}}, s_{1-j}^{\mathsf{L}} \oplus s_{1-j}^{\mathsf{R}} \mid\mid t_{1-j}^{\mathsf{R}}), (\sigma_{1-j}^{\mathsf{L}} \mid\mid \tau_{1-j}^{\mathsf{L}}, \sigma_{1-j}^{\mathsf{L}} \oplus \sigma_{1-j}^{\mathsf{R}} \mid\mid \tau_{1-j}^{\mathsf{R}}))
\end{aligned}$$

```
Input: $(1^\lambda, i, j)$
Initialisation: Sample random $\alpha \xleftarrow{\$} \mathbb{Z}_{2^n}$
Sample random $s_0^{(1)}, s_1^{(1)} \xleftarrow{\$} \{0,1\}^\lambda$ and set $t_0^{(1)} = 0,\ t_1^{(1)} = 1$
for $k = 1..n$ do
    $((s_j^{\mathsf{L}} \mid\mid t_j^{\mathsf{L}}, s_j^{\mathsf{R}} \mid\mid t_j^{\mathsf{R}}), (\sigma_j^{\mathsf{L}} \mid\mid \tau_j^{\mathsf{L}}, \sigma_j^{\mathsf{R}} \mid\mid \tau_j^{\mathsf{R}})) \leftarrow G(s_j^{(k)})$
    if $k < i$ then
        $CW^{(k)} \xleftarrow{\$} \{0,1\}^{2(\lambda+n+2)}$
    else
        if $k = i$ then $s_{1-j}^{(i)} \xleftarrow{\$} \{0,1\}^\lambda$ and $t_{1-j}^{(i)} = 1 - t_j^{(i)}$ ;
        $((s_{1-j}^{\mathsf{L}} \mid\mid t_{1-j}^{\mathsf{L}}, s_{1-j}^{\mathsf{R}} \mid\mid t_{1-j}^{\mathsf{R}}), (\sigma_{1-j}^{\mathsf{L}} \mid\mid \tau_{1-j}^{\mathsf{L}}, \sigma_{1-j}^{\mathsf{R}} \mid\mid \tau_{1-j}^{\mathsf{R}})) \leftarrow G(s_{1-j}^{(k)})$
        if $\alpha[k]$ then
            $cw^{(k)} \leftarrow ((0^\lambda \mid\mid 0, s_0^{\mathsf{L}} \oplus s_1^{\mathsf{L}} \mid\mid 1), (\sigma_0^{\mathsf{R}} \oplus \sigma_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0))$
        else
            $cw^{(k)} \leftarrow ((s_0^{\mathsf{R}} \oplus s_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0), (0^\lambda \mid\mid 0, \sigma_0^{\mathsf{L}} \oplus \sigma_1^{\mathsf{L}} \mid\mid 1))$
        $CW^k \leftarrow cw^{(k)} \oplus G(s_0^{(k)}) \oplus G(s_1^{(k)})$
        $state_{1-j} \leftarrow G(s_{1-j}^{(k)}) \oplus (t_{1-j}^{(k)} \cdot CW^k) =$
          $((state_{1-j,0}, state_{1-j,1}), (state'_{1-j,0}, state'_{1-j,1}))$
        Parse $s_{1-j}^{(k+1)} \mid\mid t_{1-j}^{(k+1)} = state_{1-j,\alpha[k]}$ and $\sigma_{1-j}^{(k+1)} \mid\mid \tau_{1-j}^{(k+1)} = state'_{1-j,1-\alpha[k]}$
    $state_j \leftarrow G(s_j^{(k)}) \oplus (t_j^{(k)} \cdot CW^k) = ((state_{j,0}, state_{j,1}), (state'_{j,0}, state'_{j,1}))$
    Parse $s_j^{(k+1)} \mid\mid t_j^{(k+1)} = state_{j,\alpha[k]}$ and $\sigma_j^{(k+1)} \mid\mid \tau_j^{(k+1)} = state'_{j,1-\alpha[k]}$
    if $k < i$ then
        $CW_{leaf}^k \xleftarrow{\$} \{0,1\}^n$
    else
        $CW_{leaf}^k \leftarrow (-1)^{\tau_1^{(k+1)}} \cdot \left(\sigma_1^{(k+1)} - \sigma_0^{(k+1)} + \alpha[k]\right) \bmod 2^n$ if $k < i$ else $\{0,1\}^n$
$CW_{leaf}^{(n+1)} \leftarrow (-1)^{t_1^{(n+1)}} \cdot \left(1 - s_0^{(n+1)} + s_1^{(n+1)}\right) \bmod 2^n$ if $i \leq n$ else $\{0,1\}^n$
return $\mathsf{k_j} \leftarrow [\![\alpha]\!]_j \mid\mid s_j^{(1)} \mid\mid (CW^{(i)})_{i=1..n} \mid\mid (CW_{leaf}^{(i)})_{i=1..n+1}$
```

Algorithm 7: Hyb_i: Hybrid distribution i, in which the first i correction words are sampled completely at random, and the remaining correction words are computed honestly.

```
Input: $(1^\lambda, (i,j), r)$
Sample random $\alpha \xleftarrow{\$} \mathbb{Z}_{2^n}$
Sample $s_j^{(1)} \xleftarrow{\$} \{0,1\}^\lambda$ and set $t_j^{(1)} \leftarrow j$
for $k = 1..i-1$ do
    $CW^{(k)} \xleftarrow{\$} \{0,1\}^{2(\lambda+n+2)}$
    $CW_{leaf}^{(k)} \leftarrow \{0,1\}^n$
    $state_j \leftarrow G(s_j^{(k)}) \oplus (t_j^{(k)} \cdot CW^{(k)} = ((state_{j,0}, state_{j,1}), (state'_{j,0}, state'_{j,1}))$
    Parse $s_j^{(k+1)} \mid\mid t_j^{(k+1)} = state_{j,\alpha[k]}$ and $\sigma_j^{(k+1)} \mid\mid \tau_j^{(k+1)} = state'_{j,1-\alpha[k]}$
    Take $t_{1-j}^{(k+1)} = 1 - t_j^{(k+1)}$
$((s_j^{\mathsf{L}} \mid\mid t_j^{\mathsf{L}}, s_j^{\mathsf{R}} \mid\mid t_j^{\mathsf{R}}), (\sigma_j^{\mathsf{L}} \mid\mid \tau_j^{\mathsf{L}}, \sigma_j^{\mathsf{R}} \mid\mid \tau_j^{\mathsf{R}})) \leftarrow G(s_j^{(i)})$
$((s_{1-j}^{\mathsf{L}} \mid\mid t_{1-j}^{\mathsf{L}}, s_{1-j}^{\mathsf{R}} \mid\mid t_{1-j}^{\mathsf{R}}), (\sigma_{1-j}^{\mathsf{L}} \mid\mid \tau_{1-j}^{\mathsf{L}}, \sigma_{1-j}^{\mathsf{R}} \mid\mid \tau_{1-j}^{\mathsf{R}})) \leftarrow r$ ;    // The PRG challenge
if $\alpha[i]$ then $cw^{(i)} \leftarrow ((0^\lambda \mid\mid 0, s_0^{\mathsf{L}} \oplus s_1^{\mathsf{L}} \mid\mid 1), (\sigma_0^{\mathsf{R}} \oplus \sigma_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0))$
else $cw^{(i)} \leftarrow ((s_0^{\mathsf{R}} \oplus s_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0), (0^\lambda \mid\mid 0, \sigma_0^{\mathsf{L}} \oplus \sigma_1^{\mathsf{L}} \mid\mid 1))$;
$CW^{(i)} \leftarrow cw^{(i)} \oplus G(s_j^{(i)}) \oplus r$
for $x = 0, 1$ do
    $state_x \leftarrow G(s_x^{(i)}) \oplus (t_x^{(i)} \cdot CW^{(i)})$ if $x = j$ else $r \oplus (t_x^{(i)} \cdot CW^{(i)})$
    $state_x = ((state_{x,0}, state_{x,1}), (state'_{x,0}, state'_{x,1}))$
    Parse $s_x^{(i+1)} \mid\mid t_x^{(i+1)} = state_{x,\alpha[i]}$ and $\sigma_x^{(i+1)} \mid\mid \tau_x^{(i+1)} = state'_{x,1-\alpha[i]}$
$CW_{leaf}^{(i)} \leftarrow (-1)^{\tau_1^{(i+1)}} \cdot \left(\sigma_1^{(i+1)} - \sigma_0^{(i+1)} + \alpha[i]\right) \bmod 2^n$

for $k = i+1..n$ do
    for $x = 0, 1$ do
        $((s_x^{\mathsf{L}} \mid\mid t_x^{\mathsf{L}}, s_x^{\mathsf{R}} \mid\mid t_x^{\mathsf{R}}), (\sigma_x^{\mathsf{L}} \mid\mid \tau_x^{\mathsf{L}}, \sigma_x^{\mathsf{R}} \mid\mid \tau_x^{\mathsf{R}})) \leftarrow G(s_x^{(k)})$
    if $\alpha[k]$ then $cw^{(k)} \leftarrow ((0^\lambda \mid\mid 0, s_0^{\mathsf{L}} \oplus s_1^{\mathsf{L}} \mid\mid 1), (\sigma_0^{\mathsf{R}} \oplus \sigma_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0))$
    else $cw^{(k)} \leftarrow ((s_0^{\mathsf{R}} \oplus s_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0), (0^\lambda \mid\mid 0, \sigma_0^{\mathsf{L}} \oplus \sigma_1^{\mathsf{L}} \mid\mid 1))$;
    $CW^{(k)} \leftarrow cw^{(k)} \oplus G(s_0^{(k)}) \oplus G(s_1^{(k)})$
    for $x = 0, 1$ do
        $state_x \leftarrow G(s_x^{(k)}) \oplus (t_x^{(k)} \cdot CW^{(k)}) = ((state_{x,0}, state_{x,1}), (state'_{x,0}, state'_{x,1}))$
        Parse $s_x^{(k+1)} \mid\mid t_x^{(k+1)} = state_{x,\alpha[k]}$ and $\sigma_x^{(k+1)} \mid\mid \tau_x^{(k+1)} = state'_{x,1-\alpha[k]}$
    $CW_{leaf}^{(k)} \leftarrow (-1)^{\tau_1^{(k+1)}} \cdot \left(\sigma_1^{(k+1)} - \sigma_0^{(k+1)} + \alpha[k]\right) \bmod 2^n$
$CW_{leaf}^{(n+1)} \leftarrow (-1)^{t_1^{(n+1)}} \cdot \left(1 - s_0^{(n+1)} + s_1^{(n+1)}\right) \bmod 2^n$
return $\mathsf{k_j} = [\![\alpha]\!]_j \mid\mid s_j^{(1)} \mid\mid (CW^{(i)})_{i=1..n} \mid\mid (CW_{leaf}^{(i)})_{i=1..n+1}$
```

Algorithm 8: PRG Challenge for adversary $\mathcal{B}$

When $\alpha[i] = 0$:

$$\begin{aligned} cw^{(i)} \oplus r =& ((s_0^{\mathsf{R}} \oplus s_1^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0), (0^\lambda \mid\mid 0, \sigma_0^{\mathsf{L}} \oplus \sigma_1^{\mathsf{L}} \mid\mid 1)) \\ &\oplus((s_{1-j}^{\mathsf{L}} \mid\mid t_{1-j}^{\mathsf{L}}, s_{1-j}^{\mathsf{R}} \mid\mid t_{1-j}^{\mathsf{R}}), (\sigma_{1-j}^{\mathsf{L}} \mid\mid \tau_{1-j}^{\mathsf{L}}, \sigma_{1-j}^{\mathsf{R}} \mid\mid \tau_{1-j}^{\mathsf{R}})) \\ =&((s_j^{\mathsf{R}} \mid\mid 1, 0^\lambda \mid\mid 0), (0^\lambda \mid\mid 0, \sigma_j^{\mathsf{L}} \mid\mid 1)) \\ &\oplus((s_{1-j}^{\mathsf{L}} \oplus s_{1-j}^{\mathsf{R}} \mid\mid t_{1-j}^{\mathsf{L}}, s_{1-j}^{\mathsf{R}} \mid\mid t_{1-j}^{\mathsf{R}}), (\sigma_{1-j}^{\mathsf{L}} \mid\mid \tau_{1-j}^{\mathsf{L}}, \sigma_{1-j}^{\mathsf{L}} \oplus \sigma_{1-j}^{\mathsf{R}} \mid\mid \tau_{1-j}^{\mathsf{R}})) \end{aligned}$$

Independently of the value of $\alpha[i]$, when r is random, the right hand side of the $\oplus$ acts as a perfect one-time pad, and so $CW^{(i)}$ is distributed uniformly.

Consider now $CW_{leaf}^{(i)}$, computed as such:

$$CW_{leaf}^{(i)} \leftarrow (-1)^{\tau_1^{(i+1)}} \cdot \left(\sigma_1^{(i+1)} - \sigma_0^{(i+1)} + \alpha[i]\right) \bmod 2^n$$

Since $\sigma_{1-j}^{(i+1)}$ is distributed randomly conditioned on the previous values computed, it acts as a one-time pad, which ensures that $CW_{leaf}^{(i)}$ is distributed uniformly.

Now, condition on $CW^{(i)}$ as well, and consider the value of $s_{1-j}^{(i+1)}$. $s_{1-j}^{(i+1)}$ is extracted from $r \oplus t_{1-j}^{(i)} \cdot CW^{(i)}$, see Line 15. If $t_{1-j}^{(i)} = 0$, $s_{1-j}^{(i+1)}$ is immediately uniformly distributed. If $t_{1-j}^{(i)} = 1$, $r \oplus (t_{1-j}^{(i)} \cdot CW^{(i)}) = r \oplus CW^{(i)} = r \oplus cw^{(i)} \oplus G(s_j^{(i)}) \oplus r = cw^{(i)} \oplus G(s_j^{(i)})$. When $\alpha[i] = 1$, the part of $cw^{(i)}$ contributing to $s_{1-j}^{(i+1)}$ is $s_j^{\mathsf{L}} \oplus s_{1-j}^{\mathsf{L}}$. s_{1-j}^{L} is random and hence acts as a perfect one-time pad, so $s_{1-j}^{(i+1)}$ is uniformly distributed. When $\alpha[i] = 0$, the same result is derived using s_{1-j}^{R}. Finally, consider the value of $t_{1-j}^{(i+1)}$. We show that $t_{1-j}^{(i+1)} = 1 - t_j^{(i)}$. By construction,

$$t_{1-j}^{(i+1)} = t_{1-j}^{\mathsf{Keep}} \oplus t_{1-j}^{(i)} \cdot t_{CW}^{\mathsf{Keep}}$$

where $\mathsf{Keep} = \mathsf{L}$ if $\alpha[i] = 0$ else R, and where $t_{CW}^{\mathsf{Keep}} = 1 \oplus t_0^{\mathsf{Keep}} \oplus t_1^{\mathsf{Keep}}$. Furthermore, by noting that $t_{1-j}^{(i)}$ was set to $1 - t_j^{(i)}$,

$$\begin{aligned} & t_j^{(i+1)} \oplus t_{1-j}^{(i+1)} \\ &= (t_j^{\mathsf{Keep}} \oplus t_j^{(i)} \cdot t_{CW}^{\mathsf{Keep}}) \oplus (t_{1-j}^{\mathsf{Keep}} \oplus t_{1-j}^{(i)} \cdot t_{CW}^{\mathsf{Keep}}) \\ &= t_j^{\mathsf{Keep}} \oplus t_{1-j}^{\mathsf{Keep}} \oplus (t_j^{(i)} \oplus t_{1-j}^{(i)}) \cdot t_{CW}^{\mathsf{Keep}}) \\ &= t_j^{\mathsf{Keep}} \oplus t_{1-j}^{\mathsf{Keep}} \oplus 1 \cdot (1 \oplus t_0^{\mathsf{Keep}} \oplus t_1^{\mathsf{Keep}}) \\ &= 1 \end{aligned}$$

Combining these pieces, we have that in the case of a random PRG challenge r, the resulting distribution of $\mathsf{k_j}$ as generated by $\mathcal{B}$ is precisely distributed as is $\mathsf{Hyb}_i(1^\lambda, j)$. Thus, the advantage of $\mathcal{B}$ in the PRG challenge experiment is equivalent to the advantage ϵ of $\mathcal{A}$ in distinguishing $\mathsf{Hyb}_{i-1}(1^\lambda, j)$ from $\mathsf{Hyb}_i(1^\lambda, j)$. The runtime of $\mathcal{B}$ is equal to the runtime of $\mathcal{A}$ plus a fixed polynomial $p'(\lambda)$. Thus for any $T' \leq T - p'(\lambda)$, it must be that the distinguishing advantage ϵ of $\mathcal{A}$ is bounded by ϵ_{PRG}, which concludes the proof of Claim 5.2.1. □

Combining all the steps, for $i \in \{1, \dots n\}$, we have thus proven that $\mathsf{Hyb}_0(1^\lambda, j)$ and $\mathsf{Hyb}_n(1^\lambda, j)$ are computationally indistinguishable for any adversary, for $j \in \{0, 1\}$. On the other hand, we can also prove:

Claim 5.2.2.

$$\mathsf{Hyb}_n(1^\lambda, j) = \mathsf{Hyb}_{n+1}(1^\lambda, j)$$

Proof. In Hyb_{n+1}, $CW_{leaf}^{(n+1)} \xleftarrow{\$} \{0,1\}^n$. In Hyb_n, $CW_{leaf}^{(n+1)} = (-1)^{t_1^{(n+1)}} \cdot (1 - s_0^{(n+1)} + s_1^{(n+1)}) \bmod 2^n$, with $s_{1-j}^{(n+1)}$ distributed randomly conditioned on the previous values computed. $s_{1-j}^{(n+1)}$ acts as a one-time pad which perfectly hides other values. Hence, $CW_{leaf}^{(n+1)}$ is also uniformly distributed in this case. □

This concludes the proof of security of our FSS comparison protocol.

Implementation and Communication Complexity. In all these computations modulo 2^n, the bitstrings $s_j^{(i)}$ and $\sigma_j^{(i)}$ are respectively in $\{0,1\}^\lambda$ and $\{0,1\}^n$, where we have typically $\lambda = 128$ and $n = 32$. The PRG used here is $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)+2(n+1)}$ where the output is seen as a pair of pairs of elements in $\Big(\{0,1\}^{\lambda+1} \times \{0,1\}^{\lambda+1}\Big) \times \big(\{0,1\}^{n+1} \times \{0,1\}^{n+1}\big)$. For the right-hand part, we only need n bits instead of λ bits since the $\sigma^{(i)}$ deriving from the PRG are not used for anything other than masking the n-bit output. This allows us to use fewer AES block ciphers in our PRG implementation and hence to achieve faster computation. In addition, because our comparison protocol works very similarly to the equality protocol, we can reuse the trick that consists of reusing randomness of the state corresponding of leaving the special area for the state corresponding of staying into it, as it does not compromise the fact that this state only needs to be pseudo-random. Thanks to this, we almost divide by 2 the size of the $CW^{(i)}$ from $2(\lambda+1)+2(n+1)$ to $\lambda+2+n+2$. Compared to the previous Distributed Interval Function (DIF) protocol of [BGI16], our algorithm is not only much simpler as it does not require inspecting binary trees and searching for paths, but it also reduces significantly the key size from roughly $n(4\lambda+n)$ to $n(\lambda+2n+4)+\lambda+2n$ bits. This allows for faster transmission of keys over the network to the parties doing the evaluation.

5.2.4 Application to Deep Learning

We now apply these primitives to a private deep learning setup in which a model owner interacts with a data owner. The data and the model parameters are sensitive and are secret shared to be kept private. The shape of the input and the architecture of the model are however public, which is a standard assumption in secure deep learning [LJLA17, MZ17].

5.2.4.1 Additive Sharing Workflow with Preprocessing

All our operations are modular and follow this additive sharing workflow: inputs are provided secret shared and are masked with random values before being revealed. This disclosed value is then consumed with preprocessed function keys to produce a secret shared output. Each operation is independent of all surrounding operations, which is known as *circuit-independent preprocessing* [BGI19] and implies that key generation can be fully outsourced without having to know the model architecture. This results in a fast runtime execution with a very efficient online communication, with a single round of communication and a message size equal to the input size for comparison.

Additionally, values need to be converted from float to fixed point precision before being secret shared. The fixed point representation allows one to store decimal values with some approximation using n-bits integers. For example, when using a fixed precision of 3 in base 10, a decimal value x is stored as $\lfloor x \cdot 10^3 \rfloor$ in $\mathbb{Z}_{2^n}$. Fixed precision is used to simplify operations like addition because the inputs can be summed up directly in $\mathbb{Z}_{2^n}$.

5.2.4.2 Common Machine Learning Operations

ReLU activation function is supported as a direct application of our comparison protocol, which we combine with a point wise multiplication. As mentioned in Section 5.2.2, this construction is not exact and is associated with an error rate which is below 1 in a million for typical ML computations. The comparison made in Table 5.5 between fixed point and private evaluation of pre-trained models shows that this error rate does not affect model accuracy.

Matrix Multiplication (MatMul), as mentioned by [BGI19], fits in this additive sharing workflow. We use Beaver triples [Bea91] to compute $[\![z]\!] = [\![x \cdot y]\!]$ from $[\![x]\!]$, $[\![y]\!]$ and using a triple

```
  Input: $[\![x_1]\!], \dots, [\![x_m]\!]$
  Output: $\arg\max_{i \in [1,m]} x_i$
1 for $j \in [1, m]$ do
2     $[\![s_j]\!] \leftarrow \sum_{i \neq j} [\![x_i - x_j \leq 0]\!]$
3 for $j \in [1, m]$ do
4     $[\![\delta_j]\!] \leftarrow [\![s_j = m - 1]\!]$
5 return $[\![\delta_1]\!], \dots, [\![\delta_m]\!]$
```

Algorithm 9: Argmax functionality using FSS

$([\![a]\!], [\![b]\!], [\![c]\!] := [\![a \cdot b]\!])$, where all values are secret shared in $\mathbb{Z}_{2^n}$. The mask is here $[\![(-a, -b)]\!]$ and is used to reveal $(\delta, \epsilon) := (x - a, y - b)$. The functional keys are the shares of $([\![a]\!], [\![b]\!], [\![c]\!])$ and are used to compute $\delta \cdot [\![b]\!] + \epsilon \cdot [\![a]\!] + \delta \cdot \epsilon + [\![c]\!] = [\![z]\!]$. Matrix multiplication is identical but uses matrix Beaver triples [MZ17].

Convolution can also be computed using Beaver triples. Using the previous notations, we can now consider y to be the convolution kernel, and the operation $\cdot$ now stands for the convolution operator. We use this method for the CPU and GPU implementations, which enables us to use the PyTorch Conv2d function to compute the $\cdot$ operation. Note that convolution can also be computed as a matrix multiplication using an unrolling technique as described in [CPS06], but it incurs an overhead in terms of communication because the unrolled matrix is bigger than the original one when the stride is smaller than the kernel size. More details about unrolling can be found in Appendix A.2.2 with Figure A.5.

Argmax is used to determine the predicted label for classification tasks (i.e. compute the index of the highest value of the last layer). Algorithm 9 shows how to compute this operator in a constant number of rounds using pairwise comparisons, in a fashion similar to [HKSvdM19]. This algorithm outputs the indices in the one-hot format, meaning that the output vector is of a similar shape to the input, and contains 1 where the maximum was found and 0 elsewhere. This protocol does not guarantee one-hot output: if the last layer outputs two identical maximum values, both will be retrieved. This sounds acceptable for machine learning evaluation as it informs that the model cannot choose between two classes. For training, the output signal only needs to be normalized. Probabilistic techniques are available to break ties, which only require an additional comparison.

In our algorithm, the first loop (line 2) requires $m(m-1)$ parallel comparisons, and the second loop (line 4) requires m equality checks. Hence, the argmax uses 2 rounds of communication and sends $O(m^2)$ values over the network. This is reasonable for a neural network where the number of outputs m does not exceed 100.

MaxPool can be implemented by combining the ideas of the unrolling-based convolution and the argmax: the matrix is first unrolled like in Figure A.5 and the argmax of each row is then computed using parallel pairwise comparisons. This argmax is then multiplied with the row to get the maximum value, and the matrix is rolled back. These steps are illustrated in Figure A.6 in Appendix A.2.2 and are formally described in Algorithm 10. It requires 3 rounds of communication, but we also provide an optimization when the kernel size k equals 2, which reduces the computation complexity by a factor 4× but uses an additional round of communication, and is very useful for some deep models such as VGG16.

BatchNorm is implemented using Newton's method as in [WTB$^+$21] to implement the square inverse of the variance, as computing batch normalization exactly in a private way is very costly [WGC19]. Given an input $\mathbf{x} = (x_0, \dots, x_{m-1})$ with mean μ and variance σ^2, we return $\gamma \cdot \hat{\theta} \cdot (\boldsymbol{x} - \mu) + \beta$. Variables γ and β are learnable parameters and $\hat{\theta}$ is the estimate inverse of $\sqrt{\sigma^2 + \epsilon}$ with $\epsilon \ll 1$ and is computed iteratively as such:

```
Input: $[\![\mathbf{X}]\!] = ([\![x_{i,j}]\!])_{i,j=1 \dots m}$
Output: $[\![\mathsf{MaxPool}(\mathbf{X}, k)]\!]$
Set $n = \lfloor (m-k)/s + 1 \rfloor$
Define $[\![\mathbf{X}^{\mathsf{unrolled}}]\!]$ of shape $n^2 \times k^2$
Define $[\![\boldsymbol{y}]\!]$ of size $n^2$
for $i, j \in [0, s, 2s, \dots, m-k]$ do
    $[\![x^{\mathsf{unrolled}}_{i,j}]\!] = ([\![x_{i,j}]\!], \dots, [\![x_{i+k,j+k}]\!])$
for $i \in [0, n^2 - 1]$ do
    $[\![y_i]\!] \leftarrow \langle [\![x_i]\!], \mathsf{Argmax}([\![\boldsymbol{x}_i]\!]) \rangle$
    where $[\![\boldsymbol{x}_i]\!] = [\![x_{i,0}]\!], \dots, [\![x_{i,k^2}]\!]$
return $\boldsymbol{y}$ reshaped as a $n \times n$ matrix.
```

Algorithm 10: MaxPool functionality using FSS. k is the kernel size, the stride is fixed to 2, padding to 0 and dilation to 1.

$$\theta_{i+1} = \theta_i \cdot \frac{(C+1) - (\sigma^2 + \epsilon) \cdot \theta_i^2}{C}$$

Compared to [WTB+21], we do not make any costly initial approximation, therefore instead of $C = 2$ which corresponds to the classic Newton's method, we use higher values of C (like $C = 6$ for the intermediate layers) which can reduce the convergence speed of the method but spares the initialisation cost.

The requirements on the approximation depend whether we are doing training or evaluation. If we are evaluating a pre-trained secret-shared neural network, having a very precise approximation is crucial, especially if the model is deep like ResNet18. Indeed, the deeper the model is, the more errors in the BatchNorm layers will propagate in the model and make it unusable. However, if the model has a running mean and variance which is the default for PyTorch, we only need to compute once the square inverse of the running variance at the beginning of the computation.

For training however, we can use less precise approximations, since the goal of the batch normalization layer is to normalize the signal and this does not need to be done exactly as we show. We have found it very useful to reuse the result of the computation on the previous batch as an initial guess for the next batch. Moreover, we observe that for deep networks such as ResNet18, we can reduce the number of iterations of the Newton method from 4 to only 3 compared to [WTB+21], except of the first batch (which does not have a proper initialisation), and for the initial and last BatchNorm layers. For those layers, which either suffer from a too high or too low variance, we increase the number of iterations. For all layers, typical relative error never exceeds 5% and moderately affects learning capabilities, as our analysis on ResNet18 shows in Table 5.2. We train the model on the Hymenoptera binary classification task[4] using different approximated BatchNorm layers for which we report the associated number of rounds per layer when computed in a private way. More details about our experiments on ResNet18 can be found in Appendix 5.

Table 5.3 summarizes the online communication cost of each ML operation presented above, and shows that basic operations such as comparison have very efficient online communication. We also report results from [MR18] and [WTB+21] which achieve good experimental performance. Note that ReLU has two communication rounds: one for the binary comparison and one for multiplying the comparison with the initial input signal, in order to implement $(x \geq 0) \cdot x$.

[4] https://download.pytorch.org/tutorial/hymenoptera_data.zip

Table 5.2: Accuracy of training ResNet18 on the Hymenoptera classification task with exact or approximated BatchNorm (BN)

BatchNorm	init. with previous batch	Nb of Newton iterations	Accuracy (%)	Average comm. rounds per BN
Exact	-	-	93.59	-
Approximate	True	3	89.15	9
Approximate	False	20	88.24	60
Approximate	False	10	84.97	30
Approximate	False	3	60.13	9

Table 5.3: Theoretical online communication complexity of our protocols. Input sizes into brackets are those of the layers' parameters, where k stands for the kernel size and s for the stride. Communication is given in the number of values transmitted, and should be multiplied by their size (typically 4 bytes). Missing entries mean that data was not available.

Protocol	Input size	Rounds			Online Communication		
		Ours	FALCON	ABY^3	Ours	FALCON	ABY^3
			[WTB+21]	[MR18]		[WTB+21]	[MR18]
Equality	m	1	-	2	m	-	$\sim \lambda m$
Comparison	m	1	7	2	m	$2m$	$\sim \lambda m$
MatMul	$m_1 \times m_2, m_2 \times m_3$	1	1	1	$m_1 m_2 + m_2 m_3$	$m_1 m_3$	$m_1 m_3$
Linear	$m_1 \times m_2, \{m_2 \times m_3\}$	1	1	-	$m_1 m_2 + m_2 m_3$	$m_1 m_3$	-
Convolution	$m \times m, \{k, s\}$	1	1	-	$((m-k)/s+1)^2 k^2 + k^2$	$\sim m^2 k^2$	-
ReLU	m	2	10	-	$3m$	$4m$	-
Argmax	m	2	-	-	m^2	-	-
MaxPool	$m \times m, \{k, s\}$	3	$12(k^2-1)$	-	$((m-k)/s+1)^2(k^4+2)$	$\sim 5m^2$	-
BatchNorm	$m \times m$	9	335	-	$18m^2$	$\sim 56m^2$	-

5.2.4.3 Training Phase using Autograd

These operations are sufficient to evaluate real world models in a fully private way. To also support private training of these models, we need to perform backpropagation privately. As we overload operations such as convolutions or activation functions, we cannot use the built-in autograd functionality of PyTorch. Therefore, we have used the custom autograd functionality of the PySyft library [RTD+18], where it should be specified how to compute the derivatives of the operations that we have overloaded. Backpropagation also uses the same basic blocks than those used in the forward pass, including our private comparison protocol. Therefore, the training procedure Train described in Algorithm 11 closely follows the steps of plaintext training, except that the interactions between the secret shared data and model parameters use the protocols we have described in Section 5.2.4.2.

```
  Input: $[\![x]\!], [\![y_{\text{real}}]\!], [\![\theta]\!]$
  Output: $[\![\hat{\theta}]\!]$
1 $\mathsf{opt} = \mathsf{Optim}([\![\theta]\!])$
2 $[\![y_{\text{pred}}]\!] = \mathsf{Forward}([\![\theta]\!], [\![x]\!])$
3 $\ell = \mathcal{L}([\![y_{\text{pred}}]\!], [\![y_{\text{real}}]\!])$
4 $[\![\nabla\theta]\!] = \mathsf{Backward}(\ell, [\![\theta]\!])$
5 $[\![\theta]\!] = \mathsf{opt}([\![\nabla\theta]\!], [\![\theta]\!])$
6 return $[\![\theta]\!]$
```

Algorithm 11: Train procedure that uses data x to update the model θ. Lines $2-5$ often run on batches extracted from $[\![x]\!]$ and hence are iterated until $[\![x]\!]$ has been completely used. Optim refers here to the optimizer that implements (stochastic) gradient descent. Forward and Backward are the forward and backward passes of the model θ. $\mathcal{L}$ is the loss function (mean square error or cross entropy).

5.2.5 Extension to Private Federated Learning

This 2-party protocol between a model owner and a data owner can be extended to an n-party federated learning protocol where several clients contribute their data to a model owned by an orchestrator server. We assume that the clients have the same set of features but have different samples in their data sets. This approach is sometimes called Horizontal Federated Learning and is used widely as in secure aggregation [BIK+17]. The idea is that the server sends a version of the model to all clients, so that all clients start training the same model in parallel using their own data. With a frequency that varies depending on the settings, the server aggregates the models produced by each client and sends back the aggregated version to be further trained by all clients. This way, clients federate their effort to train a global model, without sharing their data. Compared to secure aggregation [BIK+17], we are less concerned with parties dropping before the end of the protocol (we consider institutions rather than phones), and we do not reveal the updated model at each aggregation or at any stage, hence providing better privacy.

Algorithm 12 shows one possible implementation of fully private federated learning using 2-party function secret sharing. It prevents collusion between at most k out of n clients, the threat being that a client receiving the share of another client during the aggregation phase could collude with the server to help reconstruct the model contributed by this client, and infer information about its private data. This aggregation requires extra communication rounds but this is in practice negligible compared to the training procedure Train initiated between a server and a client. Note that other aggregation mechanisms could be used, including using n-party MPC protocols or homomorphic encryption, but we proposed masking as this is quite in line with the concept of FSS where we mask the private input with α.

```
Input: Model on the server $S$, initialized with parameters $\theta$
Output: Model trained using the data from the clients $(C_i)_{i=1..n}$
Initialisation $S$ secret shares $\theta$ with the clients.
$[\![\theta]\!] \leftarrow_{\mathsf{share}} \theta$
for $i \in [1, n]$ do
    $[\![\theta_i]\!] \leftarrow_{\mathsf{copy}} [\![\theta]\!]$
    S stores $[\![\theta_i]\!]_0$ and sends $[\![\theta_i]\!]_1$ to $C_i$
Training $S$ runs in parallel $n$ training procedures.
for $i \in [1, n]$ do
    $[\![\hat{\theta}_i]\!] \leftarrow \mathsf{Train}([\![x_i]\!], [\![y_i]\!], [\![\theta_i]\!])$, where $[\![x_i]\!]$ and $[\![y_i]\!]$ are the data and corresponding
      labels from $C_i$
Aggregation All updated models are aggregated with a scheme secure against
  collusion between the server and $k$ clients.
$S$ computes $[\![\hat{\theta}]\!]_0 := \sum_{i=1..n} [\![\hat{\theta}_i]\!]_0$
$C^* \xleftarrow{\$} (C_i)_{i=1..n}$
for $i \in [1, n]$ do
    $C_i$ generates $k$ seeds and sends them to $C_{i+1 \mod n}, \dots, C_{i+k \mod n}$
    $C_i$ receives $k$ seeds from $C_{i-k \mod n}, \dots, C_{i-1 \mod n}$
    $C_i$ derives $k$ random masks $(m_j)_{j=1..k}$ from its own seeds
    $C_i$ derives $k$ random masks $(\hat{m}_j)_{j=1..k}$ from the seeds received
    $C_i$ builds a global mask $\mu_i = \sum_{j=1..k} m_j - \hat{m}_j$
    $C_i$ sends $[\![\theta_i]\!]_1 + \mu_i$ to $C^*$
$C^*$ receives $[\![\hat{\theta}]\!]_1 := \sum_{i=1..n} [\![\theta_i]\!]_1 + \sum_{i=1..n} \mu_i = \sum_{i=1..n} [\![\theta_i]\!]_1$.
$C^*$ broadcasts $[\![\hat{\theta}]\!]_1$ to all clients.
Iterate Training and Aggregation using $[\![\hat{\theta}]\!]$ until the training is complete.
return $[\![\hat{\theta}]\!]$
```

Algorithm 12: Federated Learning algorithm using 2-party Function Secret Sharing

5.2.6 Experiments

In order to simplify comparison with existing work, we follow a setup very close to the work of [WTB+21]. The reason why we compare our work to [WTB+21] is that it provides the most extensive experiments of private training and evaluation we are aware of. We are aware that [WTB+21] also provides honest-majority malicious security, but we only report their results in the honest but curious setting (where they obtain the best runtimes). We assess private inference of several networks on the datasets MNIST [LC10], CIFAR-10 [KNH14], 64×64 Tiny Imagenet [WZX17, RDS+15] and 224×224 Hymenoptera which is a subset of Imagenet, and we also benchmark private training on MNIST. More details about the datasets used can be found in Appendix A.2.4. More precisely, we assess 6 networks: a 3 layers fully-connected network (Network-1), a small convolutional network with maxpool (Network-2), LeNet [LBBH98], AlexNet [KSH12], VGG16 [SZ14] and ResNet18 [HZRS16] which to the best of our knowledge has never been studied before in private deep learning. The description of these networks is available in Appendix A.2.4.

Our implementation provides a Python interface and is tightly coupled with PyTorch to provide both the ease of use and the expressiveness of this library. To use our protocols that only work in finite groups like $\mathbb{Z}_{2^{32}}$, we convert our input values and model parameters to fixed precision. To do so, we rely on the PySyft library [RTD+18] which extends common deep learning frameworks including PyTorch with a communication layer for federated learning and supports fixed precision. The experiments are run on Amazon EC2 using m5d.4xlarge machines

Table 5.4: Comparison of the inference time between secure frameworks on several popular neural network architectures. FALCON, SecureNN, and ABY^3 are 3-party protocols. XONN and Gazelle are 2-party protocols. All protocols are evaluated in the honest-but-curious setting. For each network we report in order the runtime for computing the preprocessing (if any) using CPUs, the runtime for the online phase in LAN using CPUs, the runtime for the online phase in LAN using GPUs, the runtime for the online phase in WAN using CPUs, and the communication needed during the online phase. Runtime is given in seconds and communication in MB. Missing entries mean that data was not available.

		Network-1					Network-2					LeNet				
		LAN	LAN	LAN	WAN		LAN	LAN	LAN	WAN		LAN	LAN	LAN	WAN	
Framework	Dataset	Prep.	CPU	GPU	CPU	Comm	Prep.	CPU	GPU	CPU	Comm	Prep.	CPU	GPU	CPU	Comm
AriaNN	MNIST	0.002	0.004	0.002	0.043	0.022	0.028	0.041	0.024	0.133	0.28	0.041	0.055	0.035	0.143	0.43
FALCON	MNIST	-	0.011	-	0.990	0.012	-	0.009	-	0.760	0.049	-	0.047	-	3.06	0.74
SecureNN	MNIST	-	0.043	-	2.43	2.1	-	0.130	-	3.93	8.86	-	-	-	-	-
XONN	MNIST	-	0.130	-	-	4.29	-	0.150	-	-	32.1	-	-	-	-	-
Gazelle	MNIST	0	0.030	-	-	0.5	0.481	0.330	-	-	22.5	-	-	-	-	-
ABY^3	MNIST	0.005	0.003	-	-	0.5	-	-	-	-	-	-	-	-	-	-
CrypTFlow	MNIST	-	0.008	-	-	-	-	0.034	-	-	-	-	0.058	-	-	-

		AlexNet					VGG16					ResNet18				
		LAN	LAN	LAN	WAN		LAN	LAN	LAN	WAN		LAN	LAN	LAN	WAN	
Framework	Dataset	Prep.	CPU	GPU	CPU	Comm.	Prep.	CPU	GPU	CPU	Comm.	Prep.	CPU	GPU	CPU	Comm.
AriaNN	CIFAR-10	0.091	0.15	0.078	0.34	0.95	0.94	1.75	1.55	1.99	12.59	-	-	-	-	-
FALCON	CIFAR-10	-	0.043	-	0.13	1.35	-	0.79	-	1.27	13.51	-	-	-	-	-
AriaNN	64×64 ImageNet	0.27	0.33	0.20	0.48	1.75	3.42	7.51	6.83	8.00	53.11	-	-	-	-	-
FALCON	64×64 ImageNet	-	1.81	-	2.43	19.21	-	3.15	-	4.67	52.56	-	-	-	-	-
AriaNN	224×224 Hymenoptera	-	-	-	-	-	-	-	-	-	-	10.02	19.88	13.90	24.07	148

for CPU benchmarks and g4dn.4xlarge for GPU, both with 16 cores and 64GB of CPU RAM, and we report our results both in the LAN and in the WAN setting. Latency is of 70ms for the WAN setting and is considered negligible in the LAN setting. Last, all values are encoded on 32 bits.

5.2.6.1 Inference Time and Communication

Comparison of experimental runtimes should be taken with caution, as different implementations and hardware may result in significant differences even for the same protocol. We report our online inference runtimes in Table 5.4 and show that they compare favourably with existing work including [LJLA17, MR18, MZ17, WGC19, WTB+21]. For example, our CPU implementation of Network-1 outperforms all other studied frameworks by at least a factor 2× in the LAN setting and even more in the WAN setting. For larger networks such as AlexNet and VGG16, we have an execution time which is slightly higher than [WTB+21]. One reason for this can be that we use a Python interface to serialize messages and communicate between parties, while [WTB+21] uses exclusively C code. However, we are more communication-efficient than [WTB+21] for models starting from LeNet, with a typical gain of 7% to 30% on CIFAR-10. Regarding the high advantage we have on AlexNet and 64×64 Tiny Imagenet, this is explained by the fact that [WTB+21] uses a modified and more complex AlexNet while we use the one from PyTorch. Details about our networks architecture is given in Appendix A.2.4.

Results are given for a batched evaluation with a default batch size of 128 to amortize the communication cost, as in other works compared here. For larger networks, we reduce the batch size to have the preprocessing material (including the function keys) fitting into RAM, which reduces the benefit of amortization. The exact values chosen are available in Appendix A.2.3.

We have also added the results of our GPU implementation, which offers a clear speed-up

Table 5.5: Accuracy of pre-trained neural network architectures, evaluated over several datasets in plaintext, fixed precision and privately using FSS. Time for private evaluation in the LAN setting is also reported.

		LAN	Accuracy		
Model	Dataset	time (h)	Private	Fix prec.	Public
Network-1	MNIST	0.01	98.2	98.2	98.2
Network-2	MNIST	0.18	99.0	99.0	99.0
LeNet	MNIST	0.24	99.2	99.3	99.3
AlexNet	CIFAR-10	0.60	70.3	70.3	70.3
AlexNet	64×64 ImageNet	0.48	38.3	38.6	38.6
VGG16	CIFAR-10	5.19	87.4	87.4	87.4
VGG16	64×64 ImageNet	9.97	55.2	56.0	55.9
ResNet18	Hymenoptera	0.95	94.7	94.7	95.3

Table 5.6: Accuracy of neural network architectures trained over several datasets in plaintext, fixed precision and privately using FSS. Time for private training in the LAN setting is given in hours per epoch.

		LAN time		Accuracy		
Model	Dataset	per epoch (h)	Private	Fix prec.	Public	Epochs
Network-1	MNIST	0.78	98.0	98.0	98.2	15
Network-2	MNIST	2.8	98.3	99.0	99.0	10
LeNet	MNIST	4.2	99.2	99.2	99.3	10

compared to CPU with an execution which is between 10% and 100% faster. While this already shows the usefulness of using GPUs, one could expect a greater speed-up. One reason is that classic GPUs currently offer 16GB of RAM, which is a clear limitation for our work where we store keys in RAM. Storing the keys on the CPU would come at a marginal cost of importing them on the GPU during the online phase but would allow to use bigger batches and hence better amortize the computation.

5.2.6.2 Test Accuracy

Thanks to the flexibility of our framework, we can train each of these networks in plaintext and need only one line of code to turn them into private networks where all parameters are secret shared, or to fixed precision networks where all parameters are converted to fixed precision but computation is still in plaintext. Comparing the performance of private models with their fixed precision version helps us to understand if fixed precision by itself reduces the accuracy of the model, and gives an estimate of the loss that is related to using secret shared computation.

We compare the accuracy of several pre-trained networks in these 3 modes in Table 5.5 by running a private evaluation with FSS, a fixed precision using only PySyft and a public evaluation where the model is not modified. We observe that accuracy is well preserved overall and that converting to fixed precision has no impact on the accuracy of the model. We have a small reduction in accuracy for the two private models evaluated on 64×64 ImageNet but it remains close to the plaintext baseline. This gap can be explained by the fact that PySyft uses a basic and approximate private truncation after multiplication where truncation is directly applied on the shares, and by the error rate of our FSS comparison protocol. The drop in accuracy on ResNet18 is also minor and corresponds to a single mislabeled item.

If we degrade the encoding precision which by default considers values in $\mathbb{Z}_{2^{32}}$, or the fixed precision which is by default of 4 decimals, performance degrades as shown in Appendix A.2.1.

5.2.6.3 Training Accuracy

We have also assessed the ability of training neural networks from scratch in a private way using AriaNN. Private training is an end-to-end private procedure, which means the model or the gradients are never accessible in plaintext. We use stochastic gradient descent (SGD) with momentum, a simple but popular optimizer, and support several losses such as mean square error (used for Network-1) and cross entropy (used for Network-2 and LeNet). We report the runtime and accuracy obtained by training from scratch and evaluating several networks in Table 5.6, in plaintext, in fixed precision and in a fully private way, just as we did for inference. Note that because of the training setting, accuracy might not match best known results, but the training procedure is the same for all training modes which allows for fair comparison.

We observe that the training is done almost perfectly both in fixed precision and private mode compared to the plaintext counterpart. The only noticeable difference we observe is for Network-2, where the privately-trained model achieves 98.3% while 99.0% is expected. The fixed-precision accuracy which is 99.0% suggests that our autograd functionality is working properly, so the difference must be explained by the small failure rate of FSS. Training profiles show that the accuracy starts decreasing roughly after 3 epochs, while it is supposed to keep increasing smoothly up to the 10th epoch. Instability caused by some FSS failures could account for this behaviour. However, training on LeNet did not suffer from the same phenomenon.

Recently, [DEK21] also reports accuracy results when training securely Network-1 on MNIST, using a 3-party semi-honest protocol that mixes [MR18] and [AFL+16]. They achieve 97.8 % of accuracy in 15 epochs with a runtime of only 33.8s per epoch in the LAN setting. However, they do not provide a detailed comparison between the accuracy achieved with private training, and cleartext training. One major difference with our work is that we are more communication efficient. We only require 10.3MB of communication during the online phase while they use 33.8MB per epoch. In addition, they rely on ABY^3, which means they use much more interaction rounds, which could be costly in the WAN setting although this is not monitored by this work.

Training did not complete in reasonable time in our experiments for larger networks such as VGG16, which in practice might be fine-tuned rather than trained from scratch. Note that training time includes the time spent building the preprocessing material, as it is too large to be fully processed and stored in RAM in advance.

5.2.6.4 Computation and Communication Analysis

We have provided in Table 5.7 a small analysis of how the compute time can be decomposed. We use AlexNet on the Tiny Imagenet dataset as it is the biggest network on which we could use a batch size higher than 64 both on CPU and GPU and hence amortize the serialization and communication cost.

Table 5.7: Distribution of the compute time during inference of AlexNet on the Tiny Imagenet dataset using either CPUs or GPUs.

		MatMul and	Serialization	
Processor	FSS	Convolution	and Deser.	Other
CPU	16%	72%	4%	8%
	53ms	238ms	13ms	26ms
GPU	51%	39%	8%	2%
	102ms	78ms	16ms	4ms

Thanks to the efficiency of our Rust implementation, function secret sharing only accounts for 16% for the online runtime when we use CPUs, and most of the time is spent doing matrix multiplications and convolutions. This last part uses the underlying PyTorch functions on integers which are significantly slower than when they run on floats. This motivates us to use GPUs for which such operations are far more efficient. In the GPU setting, the distribution of time is indeed much more balanced, and having function secret sharing directly running on GPUs avoids going back and forth between the CPU and the GPU.

Regarding the trade-off between computation and communication time, we show in Table 5.8 that in the WAN setting and using CPUs, computation appears to be the main bottleneck especially for bigger models. This also encourages us to further improve the GPU implementation, as any optimization of the computation efficiency will have an important impact on the overall runtime.

Table 5.8: Proportion of the overall runtime spent on computation versus communication, in the WAN setting using CPUs

Model	Dataset	Computation (%)	Comm. (%)
Network-1	MNIST	9	91
Network-2	MNIST	31	69
LeNet	MNIST	38	62
AlexNet	CIFAR-10	44	56
AlexNet	64×64 ImageNet	69	31
VGG16	CIFAR-10	88	12
VGG16	64×64 ImageNet	93	7
ResNet18	Hymenoptera	83	17

5.2.6.5 Discussion

Regarding experiments on larger networks, we could not use batches of size 128. This is mainly due to the size of the comparison function keys, which is currently proportional to the size of the input tensor, with a multiplicative factor of $n\lambda$ where $n = 32$ and $\lambda = 128$. Optimizing the function secret sharing protocol to reduce the size of those keys would allow to better amortize batched computations and would also reduce the runtime as we would manipulate smaller arrays during the private comparison. An interesting other improvement would be to run experiments on $n = 16$ bits instead of 32. Classic ML frameworks like PyTorch or TensorFlow now support 16 bits encoding both on CPU and GPU.

We have proposed a first implementation of FSS on GPU, which can still be improved to reduce the memory footprint of the keys. Further efforts could be made to decrease it roughly by 50% to match the theoretical key size. In addition, and as the small difference between the LAN and the WAN runtime shows, especially for bigger networks, most of the time is now spent on computation. Therefore, optimizing computation on GPUs will have a direct impact on the overall efficiency of the inference or the training.

We have shown the relevance of using FSS for private training and evaluation of models in machine learning. Compared to concurrent works like [BCG+21], we have shown that we have very competitive protocols, and that the failure rate of the comparison protocol has little impact for machine learning applications. We have reused this protocol in our work [KZPP+21] where it was applied to the field of medical imaging on chest X-rays.

5.2.7 Conclusion

In this work, we improve over the best known protocols for private comparison using function secret sharing by reducing the keys size by almost a factor ×4. We show how this new algorithm helps us implement efficient machine learning components and we provide constructions for ReLU and MaxPool with only 2 and 3 rounds of communication. Additionally, we show that AriaNN can implement a large diversity of neural networks, from convolutional networks to ResNet18, which are very competitive in terms of runtime and communication compared to existing work. Last, we provide an implementation of AriaNN which can run both on CPU and GPU, providing promising runtime improvements for the next generation of hardware accelerated privacy-preserving machine learning models.

Chapter

6 Differential Privacy Guarantees for Stochastic Gradient Langevin Dynamics

This chapter is based on the article [RBP22] (unpublished).

Chapter content

End-to-end private training using SMPC techniques like function secret sharing is very powerful because it allows both the data and the model to stay completely secret until the training completes. However, no matter whether the trained model is published or kept secret after training, it is likely that external users will be granted access to query it and recover the responses in clear text. In this respect, it is as susceptible to black-box attacks as if it had been trained in a non-private way. This is why one might be tempted to add differential privacy during the private training step. Unlike conventional training with differential privacy, there is one important fact that must be noted in this setting: the intermediate model updates are never released and thus need not be protected. To leverage this interesting characteristic, we build upon a recent framework [CYS21] that uses a similar assumption. We analyze the privacy leakage of noisy stochastic gradient descent by modeling Rényi divergence dynamics with Langevin diffusions. We derive similar desirable properties than [CYS21] but in the stochastic setting which more closely follows machine learning standard practice. In particular, we prove that the privacy loss converges exponentially fast for smooth and strongly convex objectives under constant step size, which is a significant improvement over previous DP-SGD analyzes. We also extend our analysis to arbitrary sequences of varying step sizes and derive new utility bounds. Last, we propose an implementation and our experiments show the practical utility of our approach compared to classical DP-SGD libraries.

6.1 Introduction

Differential privacy [DMNS06] for machine learning is a promising approach to reduce exposure of training datasets when releasing machine learning models. The privacy leakage from these models can be quantified using Rényi differential privacy [Mir17] which models it through the divergence of the distributions of two models trained on datasets that only differ in one item. The intuition behind is that a model whose behavior is sensitive to the presence or absence of a single individual is likely to memorize information about specific individuals, which can then be uncovered using several types of attacks like membership inference attacks [SSSS17].

The most standard approaches to training neural networks with differential privacy are derived from [BST14]'s method of differentially private stochastic gradient descent (DP-SGD) which leverages subsampling to improve privacy bounds. A line of work including [ACG⁺16, BBG18, WBK19] have provided tighter analyses of the privacy amplification by subsampling. DP-SGD is an attractive method as it closely mimics classic SGD training of neural networks, and applies to almost all architectures. It therefore enjoys easy adoption from data scientists and has been integrated in popular libraries like Opacus [YSS⁺21]. The privacy leakage modeled through standard (ϵ, δ)-differential privacy grows approximately in $\sqrt{K}$ for high privacy regimes [ACG⁺16, DRV10], where K is the number of iterations, which is a limitation of DP-SGD in real world applications especially for the tasks with slow convergence.

Another close line of work has explored privacy amplification by iteration [FMTT18, BBGG19], and rely on the hypothesis that the intermediate versions of the model should not be released. In particular, authors of [CYS21] have proposed a novel analysis of the differential privacy dynamics of Langevin diffusion and have applied it to a noisy gradient descent algorithm (DP-GLD). This method notably guarantees that under strongly convex and smooth objectives, the privacy leakage can be bounded by a constant, which is particularly suited for slow convergence scenarios. However, it only addresses full gradient descent (DP-GLD), which is impractical for large datasets.

Contributions. We provide a privacy analysis of the *stochastic* noisy gradient descent algorithm (DP-SGLD), which is also based on Langevin diffusions. We prove that DP-SGLD achieves similar privacy and utility guarantees than DP-GLD, including exponential convergence of the privacy bound, and we extend the analysis to the case where the step size is not constant. More specifically:

- We introduce DP-SGLD, a stochastic version of the DP-GLD algorithm studied in [CYS21], and we show that it achieves the same privacy guarantees including exponentially fast convergence.

- We show that DP-SGLD achieves similar utility than DP-GLD up to a term due to using stochastic estimates of the gradient. We also relax assumptions on the step size η in utility theorems of [CYS21] to only verify $\eta \leq \frac{1}{2\beta}$ instead of $\eta \leq \frac{\lambda}{2\beta^2}$, where β is the smoothness constant and λ the strong-convexity parameter, thus obtaining the classical scaling from convex optimization.

- We extend our analysis of DP-SGLD to non-constant step sizes and derive utility bounds when the step size is parametrized as $\eta_k = \frac{1}{2\beta+\lambda k/2}$, and removes the term due to stochastic training.

- Last, we provide an implementation of DP-SGLD[1] and an experimental evaluation on differentially private logistic regression. We show that DP-SGLD experimentally outperforms DP-SGD on several tasks, and that it significantly reduces the gap with non-private training.

[1] The code is provided in the supplementary material.

Note that authors of [WT11] also analyze this algorithm, but in order to construct samples of the posterior distribution but instead of deriving privacy guarantees. [YS22] provides a concurrent stochastic version and analysis of [CYS21].

6.2 Preliminaries

Preliminaries including the definition of Rényi differential privacy can be found in the preliminary chapter 2.

6.3 Privacy Analysis of Noisy Stochastic Gradient Descent

We use the same notations as in [CYS21]. Let $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ be a dataset of size n, with $\mathbf{x}_i \in \mathbb{R}^p$. Let $\ell(\theta, \mathbf{x})$ be the loss function of a learning algorithm parametrized by $\theta \in \mathcal{C}$ on an input x, where $\mathcal{C}$ is a closed convex set of $\mathbb{R}^d$. $\Pi_{\mathcal{C}}$ denotes the orthogonal projection onto $\mathcal{C}$. We denote by $\mathcal{L}_{\mathcal{D}}(\theta)$ the global empirical loss of the model, and by $\mathcal{L}_{\mathcal{B}_k}(\theta)$ the estimated empirical loss computed on the batch $\mathcal{B}_k$ of size $|\mathcal{B}_k| = m$:

$$\mathcal{L}_{\mathcal{D}}(\theta) = \frac{1}{n} \sum_{\mathbf{x} \in \mathcal{D}} \ell(\theta, \mathbf{x}), \qquad \mathcal{L}_{\mathcal{B}_k}(\theta) = \frac{1}{m} \sum_{\mathbf{x} \in \mathcal{B}_k} \ell(\theta, \mathbf{x}).$$

We analyse the privacy loss of the DP-SGLD algorithm given in Algorithm 13 which implements noisy stochastic gradient descent.

```
Initialisation: Dataset $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$, loss function $\ell$, step size $\{\eta_k\}_{k \geq 0}$, noise
                variance $\sigma^2$ and initial parameter $\theta_0 \in \mathcal{C}$
for $k = 0, \dots, K-1$ do
    Sample batch $\mathcal{B}_k$ from $\mathcal{D}$ with replacement
    Compute $\nabla \mathcal{L}_{\mathcal{B}_k}(\theta_k) = \frac{1}{m} \sum_{\mathbf{x} \in \mathcal{B}_k} \nabla \ell(\theta_k, \mathbf{x})$
    $\theta_{k+1} = \Pi_{\mathcal{C}}(\theta_k - \eta_k \nabla \mathcal{L}_{\mathcal{B}_k}(\theta_k) + \sqrt{2\eta_k}\, \mathcal{N}(0, \sigma^2 \mathbf{I}_d))$
return $\theta_K$
```

Algorithm 13: $\mathcal{A}_{\text{DP-SGLD}}$: Noisy Stochastic Gradient Descent

Let θ_k and θ'_k denote the parameters at the k-th iteration of $\mathcal{A}_{\text{DP-SGLD}}$ on neighboring datasets $\mathcal{D}$ and $\mathcal{D}'$, respectively. Stating that $\mathcal{D}$ and $\mathcal{D}'$ are *neighbors* means that they only differ by one $\mathbf{x}_{i_0}$, for some index i_0. Batch $\mathcal{B}_k$ is sampled with replacement from $\mathcal{D}$, meaning that a subset of size m is chosen at random from $\mathcal{D}$ and then replaced at the end of the k-th iteration. Hence, $\mathbf{x}_{i_0}$ can only appear once in $\mathcal{B}_k$, with probability m/n. We denote by Θ_{t_k} and Θ'_{t_k} the corresponding random variables associated with θ_k and θ'_k. We abuse notation to also denote their probability density functions by Θ_{t_k} and Θ'_{t_k}. Our objective is to model and analyze the dynamics of differential privacy of this algorithm, and to compare it to the ones of the DP-GLD algorithm described in [CYS21], which implements full noisy gradient descent. To this aim, we use the theoretical framework and constructions they provide to analyze the privacy loss of releasing the output θ_K of the algorithm, assuming private internal states (i.e., $\theta_1, \dots, \theta_{K-1}$).

More precisely, the proof strategy goes as follows: we first model the transition from any step k to the next step $k+1$ in DP-SGLD using a diffusion process, and derive the evolution equation of the distribution Θ_t during $k < t < k+1$. We use the theoretical results of [CYS21] to establish the evolution of the Rényi divergence of two distributions based on neighboring datasets during $k < t < k+1$. Finally, we compute a bound on the global Rényi divergence for the K-step DP-SGLD process.

6.3.1 Tracing Diffusion for DP-SGLD

To analyze the privacy loss of DP-SGLD, which is a discrete-time stochastic process, we first interpolate each discrete update from θ_k to θ_{k+1} with a piecewise continuously differentiable diffusion process. Given step size $\{\eta_k\}_{k\geq 0}$, variance noise σ^2 and initial parameter vector θ_0, the k-th discrete update in Algorithm 13 is:

$$\theta_{k+1} = \Pi_{\mathcal{C}}(\theta_k - \eta_k \nabla \mathcal{L}_{\mathcal{B}_k}(\theta_k) + \sqrt{2\eta_k\sigma^2}\mathbf{Z}_K), \quad \text{with } \mathbf{Z}_k \sim \mathcal{N}(0, \mathbf{I}_d), \mathcal{B}_k \overset{\$}{\leftarrow} \mathcal{D}, \tag{6.1}$$

where $\mathcal{B}_k \overset{\$}{\leftarrow} \mathcal{D}$ means that batch $\mathcal{B}_k$ is sampled with replacement from $\mathcal{D}$, and where the loss $\mathcal{L}_{\mathcal{B}_k}(\theta_k)$ is the estimated empirical loss function over batch $\mathcal{B}_k$. This discrete jump can be interpolated with the following random process $\{\Theta_t\}_{t_k \leq t \leq t_{k+1}}$, where $t_k = \sum_{i=1}^k \eta_i$,

$$\begin{cases} \Theta_{t_k} = \theta_k \\ \Theta_{t_k + \Delta t} - \left(\Theta_{t_k} - \eta_k \sum_{i=1, i\neq i_0}^m \dfrac{\nabla \ell(\Theta_{t_k}, \mathbf{x}_i)}{m}\right) \\ \qquad = -\Delta t \dfrac{\nabla \ell(\Theta_{t_k}, \mathbf{x}_{i_0})}{m} + \sqrt{2\Delta t\, \sigma^2}\mathbf{Z}_k,\ 0 < \Delta t < \eta_k \\ \Theta_{t_{k+1}} = \Pi_{\mathcal{C}}(\lim_{\Delta t \to \eta_k} \Theta_{t_k + \Delta t}),\ 0 < \Delta t < \eta_k, \end{cases} \tag{6.2}$$

where $\mathbf{Z}_k \sim \mathcal{N}(0, \mathbf{I}_d)$, $\mathcal{B}_k$ is sampled with replacement from $\mathcal{D}$, i_0 is chosen as follows: if the data item which differs between $\mathcal{D}$ and $\mathcal{D}'$ is part of $\mathcal{B}_k$, then i_0 refers to its index, else i_0 is chosen at random.

We can compute from (6.2) $\lim_{\Delta t \to \eta_k} \Theta_{t_k + \Delta t} = \theta_k - \eta_k \nabla \mathcal{L}_{\mathcal{B}_k}(\theta_k) + \sqrt{2\eta_k \sigma^2}\mathbf{Z}_K$. Therefore by the update equation (6.1), we see that the random variable $\Theta_{t_{k+1}} = \Pi_{\mathcal{C}}(\lim_{\Delta t \to \eta_k} \Theta_{t_k + \Delta t})$ has the same distribution as the parameter θ_{k+1} at the $(k+1)^{\text{th}}$ step of DP-SGLD.

We now differentiate (6.2) over time t for $\{\Theta_t\}_{t_k < t < t_{k+1}}$ and we derive the following stochastic differential equation

$$d\Theta_t = -\tfrac{1}{m} \nabla \ell(\Theta_{t_k}, \mathbf{x}_{i_0})dt + \sqrt{2\sigma^2} d\mathbf{W}_t, \tag{6.3}$$

where $d\mathbf{W}_t \sim \sqrt{t}\mathcal{N}(0, \mathbf{I}_d)$ describes the Wiener process on $\mathbb{R}^d$, and i_0 is chosen as such:

$$\begin{cases} \{\mathbf{x}_{i_0}\} = \{\mathbf{x}_i, \mathbf{x}_i \in \mathcal{B}_k, \mathbf{x}_i \notin \mathcal{B}'_k\} & \text{if } \mathcal{B}_k \neq \mathcal{B}'_k \\ \mathbf{x}_{i_0} \overset{\$}{\leftarrow} \mathcal{B}_k & \text{if } \mathcal{B}_k = \mathcal{B}'_k, \end{cases}$$

where we assume without loss of generality that two neighboring batches $\mathcal{B}_k$ and $\mathcal{B}'_k$ are indexed so as to be equal on all indices but one.
This shows that $\{\Theta_t\}_{t_k < t < t_{k+1}}$ is a diffusion process and we repeat the construction for $k = 0, 1, \ldots$ to define a piecewise continuous diffusion process $\{\Theta_t\}_{t\geq 0}$ whose distribution at time $t = t_k$ is consistent with θ_k. We refer to this process as the tracing diffusion for DP-SGLD.

Definition 12 (Coupled tracing diffusions [CYS21])**.** Let $\Theta_0 = \Theta'_0$ be two identically distributed random variables. We refer to the stochastic processes $\{\Theta_t\}_{t\geq 0}$ and $\{\Theta'_t\}_{t\geq 0}$ defined by (6.2) as coupled tracing diffusions processes for DP-SGLD under loss function $\ell(\theta, \mathbf{x})$ on neighboring datasets $\mathcal{D}$, $\mathcal{D}'$ differing in i_0-th data point.

The Rényi divergence $R_\alpha(\Theta_{t_K} \,||\, \Theta'_{t_K})$ reflects the Rényi privacy loss of Algorithm 13 with K steps. Conditioned on observing θ_k and on sampling $\mathcal{B}_k$ from $\mathcal{D}$, the process $\{\Theta_t\}_{t_k < t < t_{k+1}}$ is a Langevin diffusion along a constant vector field $\nabla \ell(\theta_k, \mathbf{x}_{i_0})$ for duration η_k. Therefore, the conditional probability distribution $p_{t|t_k}(\theta|\theta_k)$ follows the following Fokker-Planck equation, where the notation $p_{t|t'}(\theta|\theta')$ represents the conditional probability density function $p(\Theta_t = \theta | \Theta_{t'} = \theta')$:

$$\frac{\partial p_{t|t_k}(\theta|\theta_k)}{\partial t} = \nabla \cdot \left(p_{t|t_k}(\theta|\theta_k) \frac{1}{m} \nabla \ell(\theta_k, \mathbf{x}_{i_0})\right) + \sigma^2 \Delta p_{t|t_k}(\theta|\theta_k).$$

By taking expectations over the distribution $p_{t_k}(\theta_k)$ on both sides, we get the partial differential equation that models the evolution of probability distribution $p_t(\theta)$ in the tracing diffusion.

Lemma 13. *For the SDE (6.3), the equivalent Fokker-Planck equation at time $t_k < t < t_{k+1}$ is*

$$\frac{\partial p_t(\theta)}{\partial t} = \nabla \cdot \Big(p_t(\theta) \mathop{\mathbb{E}}_{\theta_k \sim p_{t_k|t}} \Big[\frac{1}{m} \nabla \ell(\theta_k, \mathbf{x}_{i_0}) \Big| \theta, \mathcal{B}_k \Big] \Big) + \sigma^2 \Delta p_t(\theta).$$

Using this distribution evolution equation, we model the privacy dynamics in the tracing diffusion process. This process is similar to Langevin diffusion under conditionally expected loss function $\nabla \mathcal{L}_{\mathcal{B}_k}(\theta) = \mathbb{E}_{\theta_k \sim p_{t_k|t}} \big[\frac{1}{m} \nabla \ell(\theta_k, \mathbf{x}_{i_0}) | \theta, \mathcal{B}_k \big]$.

6.3.2 Privacy Erosion in Tracing (Langevin) Diffusion

The analysis of the privacy erosion in tracing Langevin diffusion is detailed in [CYS21] but we provide here the key results that we will apply to our algorithm. Privacy erosion refers to the continuous increase of the privacy loss over time as more data is accessed to compute the gradient of the loss in the update equation.

Consider a Langevin diffusion process modelled through the following Fokker-Planck equation:

$$\frac{\partial p_t(\theta)}{\partial t} = \nabla \cdot (p_t(\theta) \nabla \mathcal{L}_{\mathcal{B}_k}(\theta)) + \sigma^2 \Delta p_t(\theta).$$

Definition 14 (Log-Sobolev inequality [Gro75])**.** The distribution of a variable Θ satisfies the Log-Sobolev Inequality for a constant c, or is c-LSI, if for all functions $f : \mathbb{R}^d \to \mathbb{R}$ with continuous ∇f and $\mathbb{E}[f(\Theta)^2] < \infty$, it satisfies

$$\mathbb{E}[f(\Theta)^2 \log f(\Theta)^2] - \mathbb{E}[f(\Theta)^2] \log \mathbb{E}[f(\Theta)^2] \leq \frac{2}{c} \mathbb{E}[\|\nabla f(\Theta)\|_2^2].$$

Lemma 15 (Dynamics for Rényi privacy loss under c-LSI [CYS21])**.** *Assuming that Θ is c-LSI and S_g is the sensitivity of the loss gradient, the dynamics of the Rényi privacy loss can be modelled as such, where $R(\alpha, t)$ represents the α Rényi divergence $R_\alpha(\Theta_t \,||\, \Theta'_t)$:*

$$\frac{\partial R(\alpha,t)}{\partial t} \leq \frac{1}{\gamma} \frac{\alpha S_g^2}{4\sigma^2} - 2(1-\gamma)\sigma^2 c \left[\frac{R(\alpha,t)}{\alpha} + (\alpha - 1) \frac{\partial R(\alpha,t)}{\partial \alpha} \right], \tag{6.4}$$

for some γ which can be arbitrarily fixed.

The initial privacy loss satisfies $R(\alpha, 0) = 0$ as $\Theta_0 = \Theta'_0$. The solution $R(\alpha, t)$ for this equation increases with time $t \geq 0$, which models the erosion of Rényi privacy loss in Langevin diffusion. Intuitively, the c-LSI condition which provides the negative dependence $\frac{\partial R(\alpha,t)}{\partial t}$ with respect to $R(\alpha, t)$, can be interpreted as a sufficient condition to ensure that the the Rényi privacy loss is bounded, which is further formalized in Theorem 17.

6.3.3 Privacy Guarantee for DP-SGLD

We now extend these results to the tracing diffusion for DP-SGLD. In addition, we do not make the assumption that the step size is constant and use a sequence $\{\eta_k\}_{k \geq 0}$ instead. We first bound the gradient sensitivity of the conditionally expected loss $\nabla \mathcal{L}_{\mathcal{B}_k}(\theta) = \mathbb{E}_{\theta_k \sim p_{t_k|t}} \Big[\frac{1}{m} \nabla \ell(\theta_k, \mathbf{x}_{i_0}) | \theta, \mathcal{B}_k \Big]$.

Lemma 16 (Sensitivity)**.** *Let $\ell(\theta, \mathbf{x})$ be an L-Lipschitz loss function on closed convex set $\mathcal{C}$, then for coupled tracing diffusions $\{\Theta_t\}_{t \geq 0}$ and $\{\Theta'_t\}_{t \geq 0}$ for DP-SGLD on neighboring datasets $\mathcal{D}$ and $\mathcal{D}'$, and noise variance σ^2, the gradient sensitivity of conditionally expected loss $\nabla \mathcal{L}_{\mathcal{B}_k}(\theta) = \mathbb{E}_{\theta_k \sim p_{t_k|t}} \Big[\frac{1}{m} \nabla \ell(\theta_k, \mathbf{x}_{i_0}) \,|\, \theta, \mathcal{B}_k \Big]$ is bounded by:*

$$\Big\| \mathop{\mathbb{E}}_{\theta_k \sim p_{t_k|t}} \Big[\frac{1}{m} \nabla \ell(\theta_k, \mathbf{x}_{i_0}) \Big| \theta, \mathcal{B}_k \Big] - \mathop{\mathbb{E}}_{\theta'_k \sim p'_{t_k|t}} \Big[\frac{1}{m} \nabla \ell(\theta'_k, \mathbf{x}'_{i_0}) \Big| \theta, \mathcal{B}_k \Big] \Big\|_2 \leq \frac{2L}{n},$$

where n is the size of the dataset $\mathcal{D}$ and $\mathcal{D}'$, and m is the size of the batch $\mathcal{B}_k$ and $\mathcal{B}'_k$.

Proof. We first distinguish on the events $\mathcal{B}_k = \mathcal{B}'_k$ and $\mathcal{B}_k \neq \mathcal{B}'_k$ and then use the triangle inequality:

$$\begin{aligned}
&\left\| \mathbb{E}_{\theta_k \sim p_{t_k|t}} \left[\frac{\nabla \ell(\theta_k; \mathbf{x}_{i_o})}{m} | \theta, \mathcal{B}_k \right] \right. \\
&\qquad \left. - \mathbb{E}_{\theta'_k \sim p'_{t_k|t}} \left[\frac{\nabla \ell(\theta'_k; \mathbf{x}'_{i_o})}{m} | \theta, \mathcal{B}_k \right] \right\|_2 \\
&= \mathbb{P}[\mathcal{B}_k \neq \mathcal{B}'_k] \cdot \left\| \mathbb{E} \left[\mathop{\mathbb{E}}_{\theta_k \sim p_{t_k|t}} \left[\frac{\nabla \ell(\theta_k; \mathbf{x}_{i_o})}{m} | \theta \right] \right. \right. \\
&\qquad \left. \left. - \mathop{\mathbb{E}}_{\theta'_k \sim p'_{t_k|t}} \left[\frac{\nabla \ell(\theta'_k; \mathbf{x}'_{i_o})}{m} | \theta \right] | \, \mathcal{B}_k \neq \mathcal{B}'_k \right] \right\|_2 \\
&\leq \frac{m}{n} \cdot \frac{2L}{m} = \frac{2L}{n}
\end{aligned}$$

□

We now substitute the sensitivity S_g with $\frac{2L}{n}$ in equation (6.4) modelling the Rényi privacy loss dynamics of tracing diffusion at $t_k \leq t < t_{k+1}$ under c-LSI condition:

$$\frac{\partial R(\alpha,t)}{\partial t} \leq \frac{1}{\gamma} \frac{\alpha L^2}{n^2 \sigma^2} - 2(1-\gamma)\sigma^2 c \left[\frac{R(\alpha,t)}{\alpha} + (\alpha-1) \frac{\partial R(\alpha,t)}{\partial \alpha} \right]. \tag{6.5}$$

Following the methodology from [CYS21], we solve this PDE under $\gamma = \frac{1}{2}$ and derive the RDP guarantee for the DP-SGLD algorithm.

Theorem 17 (RDP for DP-SGLD under c-LSI)**.** *Let $\ell(\theta, \mathbf{x})$ be an L-Lipschitz loss function on a closed convex set $\mathcal{C}$. Let $\{\Theta_t\}_{t \geq 0}$ be the tracing diffusion for $\mathcal{A}_{\mathrm{DP-SGLD}}$ under loss function $\ell(\theta, \mathbf{x})$ on dataset $\mathcal{D}$. If Θ_t satisfies c-LSI throughout $0 \leq t \leq \sum_{k=1}^{K} \eta_k$, then $\mathcal{A}_{\mathrm{DP-SGLD}}$ satisfies (α, ε)-Rényi Differential Privacy for*

$$\varepsilon = \frac{2\alpha L^2}{c n^2 \sigma^4} (1 - e^{-\sigma^2 c \sum_{k=1}^{K} \eta_k}).$$

Proof. At each step k, the Rényi privacy follows the evolution equation defined in (6.5) (since it is a tracing diffusion) when $t_k < t < t_{k+1}$, where $t_k = \sum_{i=1}^{k} \eta_i$:

$$\frac{\partial R(\alpha,t)}{\partial t} \leq \frac{1}{\gamma} \frac{\alpha L^2}{\sigma^2 n^2} - 2(1-\gamma)\sigma^2 c \left[\frac{R(\alpha,t)}{\alpha} + (\alpha-1) \frac{\partial R(\alpha,t)}{\partial \alpha} \right]$$

Let's $a_1 = 2(1-\gamma)\sigma^2 c$, $a_2 = \frac{1}{\gamma} \frac{\alpha L^2}{\sigma^2 n^2}$ and

$$u(t,y) = \begin{cases} \frac{R(\alpha,t)}{\alpha} - \frac{a_2}{a_1} & t_k < t < t_{k+1} \\ \frac{R(\alpha, \lim_{t \to t_k^+} t)}{\alpha} - \frac{a_2}{a_1} & t = t_k \end{cases}$$

We can rewrite the evolution equation as such:

$$\frac{\partial u}{\partial t} + a_1 u + a_1 \frac{\partial u}{\partial y} \leq 0 \ , \ t_k < t < t_{k+1}$$

with initial condition $u(t_k, y) = \frac{R(\alpha, \lim_{t \to t_k^+} t)}{\alpha} - \frac{a_2}{a_1}$.

Let now introduce $\tau = t$ and $z = t - \frac{1}{a_1} y$ and write $v(\tau, z) = u(t, y)$. The equation now writes:

$$\frac{\partial v}{\partial \tau} + a_1 v < 0$$

with initial condition $v(t_k, z) = u(t_k, -a_1(z - t_k))$

The solution of this equation is:

$$v(\tau, z) \le v(t_k, z)e^{-a_1(\tau - t_k)}$$

which also writes

$$u(t, y) \le u(t_k, y - a_1(t - t_k))e^{-a_1(t-t_k)}$$

or in terms of $R(\alpha, t)$:

$$R(\alpha, t) - \frac{a_2}{a_1}\alpha \le (R(\alpha, \lim_{t \to t_k^+}) - \frac{a_2}{a_1}\alpha)e^{-a_1(t-t_k)}$$

for $t_k < t < t_{k+1}$.

Taking the limit $t \to t_{k+1}^-$, we have

$$R(\alpha, \lim_{t \to t_{k+1}^-} t) - \frac{a_2}{a_1}\alpha \le (R(\alpha, \lim_{t \to t_k^+} t) - \frac{a_2}{a_1}\alpha)e^{-a_1\eta_{k+1}}$$

In addition, by the tracing diffusion process described in (6.2), we have

$$\lim_{t \to t_k^+} \Theta_t = \phi(\Pi_{\mathcal{C}}(\lim_{t \to t_k^-} \Theta_t))$$
$$\lim_{t \to t_k^+} \Theta'_t = \phi(\Pi_{\mathcal{C}}(\lim_{t \to t_k^-} \Theta'_t))$$

where $\phi(\theta) = \theta - \eta \sum_{i=1, i \ne i_0}^{m} \dfrac{\nabla \ell(\theta, \mathbf{x}_i)}{m}$ is a mapping which is the same for both processes, as the batches $\mathcal{B}_k$ for each distribution only differ at most on $\mathbf{x}_{i_0}$. Hence, by post-processing of Rényi divergence, we have

$$R(\alpha, \lim_{t \to t_k^+} t) \le R(\alpha, \lim_{t \to t_k^-} t)$$

Combining the above two inequalities, we derive the following recursive equation

$$R(\alpha, \lim_{t \to t_{k+1}^-} t) - \frac{a_2}{a_1}\alpha \le (R(\alpha, \lim_{t \to t_k^-}) - \frac{a_2}{a_1}\alpha)e^{-a_1\eta_{k+1}}$$

Repeating this step for $k = 0, \ldots, K-1$ we have

$$R(\alpha, \lim_{t \to t_K^-} t) - \frac{a_2}{a_1}\alpha \le (R(\alpha, \lim_{t \to 0^-} t) - \frac{a_2}{a_1}\alpha)e^{-a_1 t_K}$$

where $t_K = \sum_{i=1}^{K} \eta_i$. Because coupled tracing diffusions have the same start parameter, we have $R(\alpha, \lim_{t \to 0^-} t) = 0$. Moreover, since projection is a post-processing mapping we have $R(\alpha, t_K) \le R(\alpha, \lim_{t \to t_K^-} t)$. Putting back the values of a_1 and a_2 we have:

$$R(\alpha, t_K) \le \frac{\alpha L^2}{2\gamma(1-\gamma)c\sigma^4 n^2}(1 - e^{-2(1-\gamma)\sigma^2 c t_K})$$

Setting $\gamma = \frac{1}{2}$ suffices to prove the Rényi privacy loss bound in the theorem.

□

Sketch of proof. We introduce $t_k = \sum_{i=1}^{k} \eta_i$ for $k \ge 0$. The idea of the proof is to bound $R(\alpha, t_K)$, the Rényi divergence after K updates in DP-SGLD, with a function of $R(\alpha, t_0)$. This is first done by considering the k-th update, and proving the following equations for some constants a_1, a_2:

$$R(\alpha, \lim_{t \to t_{k+1}^-} t) - \frac{a_2}{a_1}\alpha \le (R(\alpha, \lim_{t \to t_k^+} t) - \frac{a_2}{a_1}\alpha)e^{-a_1\eta_{k+1}} \quad \text{and} \quad R(\alpha, \lim_{t \to t_k^+} t) \le R(\alpha, \lim_{t \to t_k^-} t).$$

This allows to bound $R(\alpha, \lim_{t\to t_{k+1}^-} t)$ with a function of $R(\alpha, \lim_{t\to t_k^-} t)$ and the final bound follows by recursivity, by noting that $R(\alpha, t_0^-) = 0$ since coupled tracing diffusions have the same start parameter.

This theorem guarantees that under the c-LSI condition, the privacy loss converges during the noisy SGD process if $\lim_{K\to\infty} \sum_{k=1}^K \eta_k = \infty$. In particular, the case where the step size is constant is straightforward:

Corollary 18 (RDP for DP-SGLD under c-LSI with constant step-size)**.** *Let Θ_t be defined as in Theorem 17. If $\mathcal{A}_{\text{DP-SGLD}}$ has constant step size η and if Θ_t satisfies c-LSI throughout $0 \le t \le \eta K$, then $\mathcal{A}_{\text{DP-SGLD}}$ satisfies (α, ε) Rényi Differential Privacy for*

$$\varepsilon = \frac{2\alpha L^2}{cn^2\sigma^4}(1 - e^{-c\sigma^2\eta K}).$$

In addition, [CYS21] show that the c-LSI condition is satisfied in DP-GLD with constant step size, for loss functions that are Lipschitz-continuous, strongly convex and smooth, with appropriate conditions on the algorithm parameters and initialization. We derive an equivalent lemma for DP-SGLD with varying step size.

Lemma 19 (LSI for DP-SGLD)**.** *If loss function $\ell(\theta, \mathbf{x})$ is λ-strongly convex and β-smooth over a closed convex set $\mathcal{C}$, then the coupled tracing diffusion processes $\{\Theta_t\}_{t\ge 0}$ and $\{\Theta'_t\}_{t\ge 0}$ for DP-SGLD with step size $\{\eta_k\}_{k\ge 0}$ satisfying $\eta_k < \frac{1}{\beta}$ for $k \ge 0$, and with initial distribution $\Theta_0 \sim \Pi_{\mathcal{C}}(\mathcal{N}(0, \frac{2\sigma^2}{\lambda}\mathbf{I}_d))$, satisfy c-LSI for $t \ge 0$ with $c = \frac{\lambda}{2\sigma^2}$.*

The proof of this lemma is exactly the same as Lemma 5 in [CYS21], where n needs to be replaced with the batch size m and η with η_k. We immediately derive the following bound on the Rényi privacy loss for DP-SGLD.

Theorem 20 (Privacy guarantee for DP-SGLD)**.** *Let $\ell(\theta, \mathbf{x})$ be an L-Lipschitz, λ-strongly convex and β-smooth loss function on closed convex set $\mathcal{C}$, then $\mathcal{A}_{\text{DP-SGLD}}$ with start parameter $\theta_0 \sim \Pi_{\mathcal{C}}(\mathcal{N}(0, \frac{2\sigma^2}{\lambda}\mathbf{I}_d))$ and step-size $\eta < \frac{1}{\beta}$ satisfies (α, ε)-Rényi differential privacy with*

$$\varepsilon = \frac{4\alpha L^2}{\lambda n^2\sigma^2}(1 - e^{-\frac{\lambda}{2}\sum_{k=1}^K \eta_k}).$$

The case where the step size is constant follows:

Corollary 21 (Privacy Guarantee for DP-SGLD with constant step-size)**.** *With $\ell(\theta, \mathbf{x})$ and $\mathcal{A}_{\text{DP-SGLD}}$ defined as in Theorem 20, and with constant step size $\eta < \frac{1}{\beta}$, $\mathcal{A}_{\text{DP-SGLD}}$ satisfies (α, ε)-Rényi Differential Privacy with*

$$\varepsilon = \frac{4\alpha L^2}{\lambda n^2\sigma^2}(1 - e^{-\lambda\eta K/2}).$$

The case where the step size decreases as $\eta_k = \frac{1}{2\beta + \lambda k/2}$ is further analyzed in the next section:

Corollary 22 (Privacy Guarantee for DP-SGLD with decreasing step-size)**.** *With $\ell(\theta, \mathbf{x})$ and $\mathcal{A}_{\text{DP-SGLD}}$ defined as in Theorem 20, and with step size $\eta_k = \frac{1}{2\beta + \lambda k/2}$, $\mathcal{A}_{\text{DP-SGLD}}$ satisfies (α, ε)-Rényi Differential Privacy with*

$$\varepsilon = \frac{4\alpha L^2}{\lambda\sigma^2 n^2}(1 - e^{-\log(1+\frac{\lambda K}{4\beta})}) = \frac{4\alpha L^2}{\lambda n^2\sigma^2}\frac{\lambda K}{4\beta + \lambda K}.$$

Discussion. Let us consider $\eta = \frac{1}{2\beta}$, as set in the utility analysis in the next section. In the fast convergence setting, i.e. when K is small compared to $\frac{\beta}{\lambda}$, Corollary 21 and Corollary 22 have equivalent evolutions on ε:

$$(21)\ \varepsilon = \frac{4\alpha L^2}{\lambda n^2\sigma^2}(1 - e^{-\lambda K/4\beta}) \sim_{K \ll \frac{\beta}{\lambda}} \frac{\alpha L^2 K}{\beta n^2 \sigma^2} \quad (22)\ \varepsilon = \frac{4\alpha L^2}{\lambda n^2\sigma^2}\frac{\lambda K}{4\beta + \lambda K} \sim_{K \ll \frac{\beta}{\lambda}} \frac{\alpha L^2 K}{\beta n^2 \sigma^2}.$$

This bound is equivalent (up to a factor 2) to the one in [ACG+16]: $\varepsilon' = \frac{\alpha L^2}{n^2\sigma^2} \cdot \eta K = \frac{\alpha L^2 K}{2\beta n^2 \sigma^2}$.

In the slow convergence regime, i.e. where K is sufficiently large compared to $\frac{\beta}{\lambda}$, both corollaries converge to the same bound on ε and outperform [ACG+16]:

$$\varepsilon \sim_{K \gg \frac{\beta}{\lambda}} \frac{4\alpha L^2}{\lambda n^2 \sigma^2}.$$

As a side note, we notice that we can consider the unconstrained regularized version of the problem: $\widetilde{\mathcal{L}_{\mathcal{D}}}(\theta) = \mathcal{L}_{\mathcal{D}}(\theta) + \frac{\lambda}{2}\left\|\theta\right\|_2^2$ and derive equivalent properties. In this scenario, we no longer need the strong convexity assumption on $\mathcal{L}_{\mathcal{D}}(\theta)$. In addition we can use the optimality of θ^* for $\widetilde{\mathcal{L}_{\mathcal{D}}}$ to derive two equations:

$$\mathcal{L}_{\mathcal{D}}(\theta^*) + \frac{\lambda}{2}\left\|\theta^*\right\|_2^2 \leq \mathcal{L}_{\mathcal{D}}(0), \qquad c\nabla\mathcal{L}_{\mathcal{D}}(\theta^*) + \lambda\theta^* = 0.$$

Each of these provides a bound on $\left\|\theta^*\right\|_2$, by using respectively the positivity of $\mathcal{L}_{\mathcal{D}}$ and the lipschitzness of $\mathcal{L}_{\mathcal{D}}$,

$$\left\|\theta^*\right\|_2 \leq \left(\frac{2\mathcal{L}_{\mathcal{D}}(0)}{\lambda}\right)^{1/2}, \quad \left\|\theta^*\right\|_2 \leq \frac{L}{\lambda},$$

which can be used as bounds for the radius of the convex $\mathcal{C}$ we project onto, so that we still actually end up solving the unconstrained problem.

6.4 Utility Analysis for Noisy Stochastic Gradient Descent

Differential privacy is known for setting a trade-off between privacy and utility. To assess the utility of the noisy stochastic gradient descent algorithm $\mathcal{A}_{\mathrm{DP-SGLD}}$, we measure two quantities, the worst case excess empirical risk

$$\max_{\mathcal{D}\in\mathcal{X}^n} \mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K) - \mathcal{L}_{\mathcal{D}}(\theta^*)],$$

and the worst case average empirical risk

$$\max_{\mathcal{D}\in\mathcal{X}^n} \mathbb{E}[\frac{1}{K}\sum_{k=1}^{K}\mathcal{L}_{\mathcal{D}}(\theta_k) - \mathcal{L}_{\mathcal{D}}(\theta^*)], \tag{6.6}$$

where θ_K is the output of the randomized algorithm $\mathcal{A}_{\mathrm{DP-SGLD}}$ on $\mathcal{D}$ during K iterations, θ^* is the solution to the standard non-noisy GD algorithm and the expectation is taken over the randomness of the algorithm.

6.4.1 Fixed Step Size η

We propose a bound on the worst case excess empirical risk when the learning rate is fixed and satisfies $\eta < \frac{1}{2\beta}$.

Lemma 23 (Empirical risk for smooth and strongly convex loss)**.** *Let $\ell(\theta, \mathbf{x})$ be an L-Lipschitz, λ-strongly convex and β-smooth loss function on closed convex set $\mathcal{C}$, $\mathcal{A}_{\mathrm{DP-SGLD}}$ be parameterized* *with step-size $\eta < \frac{1}{2\beta}$ and start parameter $\theta_0 \sim \Pi_{\mathcal{C}}(\mathcal{N}(0, \frac{2\sigma^2}{\lambda}\mathbf{I}_d))$, then the empirical risk of $\mathcal{A}_{\mathrm{DP-SGLD}}$ is bounded by*

$$\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K) - \mathcal{L}_{\mathcal{D}}(\theta^*)] \leq \frac{\beta}{2}\,\mathbb{E}[\|\theta_0 - \theta^*\|_2^2]e^{-\lambda\eta K} + \frac{\beta\eta\xi^2}{2\lambda} + \frac{\beta d\sigma^2}{\lambda}, \tag{6.7}$$

where θ^* is the minimizer of $\mathcal{L}_{\mathcal{D}}(\theta)$ in $\mathcal{C}$ and $\xi^2 = \mathbb{E}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2]$.

Sketch of proof. First, we recursively bound $\|\theta_{k+1} - \theta^*\|_2^2$ as a function of $\|\theta_k - \theta^*\|_2^2$, using the definition of θ_{k+1}. Then, we take the expectation with respect to $\mathcal{B}_k$ and use co-coercivity of the gradients, and take the expectation again to derive a recursive relationship between $\mathbb{E}[\|\theta_{k+1} - \theta^*\|_2^2]$ and $\mathbb{E}[\|\theta_k - \theta^*\|_2^2]$. Last, we express the empirical risk $\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K) - \mathcal{L}_{\mathcal{D}}(\theta^*)]$ as a function of $\mathbb{E}[\|\theta_K - \theta^*\|_2^2]$.

Proof. Let's recall the noisy SGD update equation:

$$\theta_{k+1} = \Pi_{\mathcal{C}}(\theta_k - \eta\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) + \sqrt{2\eta\sigma^2}\mathcal{N}(0, \mathbf{I}_d))$$

From the definitions of $\Pi_{\mathcal{C}}(\cdot)$ and θ^*, we have:

$$\Pi_{\mathcal{C}}(\theta^*) = \theta^*$$

Combining this facts and using the contractivity of the projection $\Pi_{\mathcal{C}}(\cdot)$, we derive:

$$\begin{aligned}
\|\theta_{k+1} - \theta^*\|_2^2 &= \left\|\Pi_{\mathcal{C}}(\theta_k - \eta\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) + \sqrt{2\eta\sigma^2}\mathcal{N}(0, \mathbf{I}_d)) - \theta^*\right\|_2^2 \\
&= \left\|\Pi_{\mathcal{C}}(\theta_k - \eta\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) + \sqrt{2\eta\sigma^2}\mathcal{N}(0, \mathbf{I}_d)) - \Pi_{\mathcal{C}}(\theta^*)\right\|_2^2 \\
&\leq \left\|\theta_k - \eta\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) + \sqrt{2\eta\sigma^2}\mathcal{N}(0, \mathbf{I}_d) - \theta^*\right\|_2^2 \\
&= \left\|\theta_k - \theta^* - \eta\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) + \sqrt{2\eta\sigma^2}\mathcal{N}(0, \mathbf{I}_d)\right\|_2^2 \\
&= \|\theta_k - \theta^*\|_2^2 + \eta^2\,\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k)\|_2^2 + 2\eta\sigma^2\,\|\mathcal{N}(0, \mathbf{I}_d)\|_2^2 \\
&\quad + 2\langle\theta_k - \theta^*, \sqrt{2\eta\sigma^2}\mathcal{N}(0, \mathbf{I}_d)\rangle \\
&\quad - 2\eta\langle\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k), \sqrt{2\eta\sigma^2}\mathcal{N}(0, \mathbf{I}_d)\rangle \\
&\quad - 2\eta\langle\theta_k - \theta^*, \nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k)\rangle
\end{aligned}$$

We now take the expectation with respect to the random variable $\mathcal{B}_k$ sampled from $\mathcal{D}$, and we note that $\mathbb{E}_{\mathcal{D}}[\mathcal{L}_{\mathcal{B}_k}(\theta_k)] = \mathcal{L}_{\mathcal{D}}(\theta_k)$ since $\mathcal{L}_{\mathcal{B}_k}$ is an unbiased estimate of $\mathcal{L}_{\mathcal{D}}$.

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}[\|\theta_{k+1} - \theta^*\|_2^2] = \|\theta_k - \theta^*\|_2^2 &+ \eta^2\,\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k)\|_2^2] \\
&+ 2\eta\sigma^2\,\mathbb{E}_{\mathcal{D}}[\|\mathcal{N}(0, \mathbf{I}_d)\|_2^2] \\
&+ 2\langle\theta_k - \theta^*, \sqrt{2\eta\sigma^2}\mathcal{N}(0, \mathbf{I}_d)\rangle \\
&- 2\eta\langle\nabla\mathcal{L}_{\mathcal{D}}(\theta_k), \sqrt{2\eta\sigma^2}\mathcal{N}(0, \mathbf{I}_d)\rangle \\
&- 2\eta\langle\theta_k - \theta^*, \nabla\mathcal{L}_{\mathcal{D}}(\theta_k)\rangle
\end{aligned}$$

Using the classic inequality $(a+b)^2 \leq 2(a^2 + b^2)$, we derive:

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k)\|_2^2] &= \mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) - \nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*) + \nabla\mathcal{L}_{\mathcal{B}_k}(\theta_*)\|_2^2] \\
&\leq 2\,\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) - \nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2] + 2\,\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2]
\end{aligned}$$

Furthermore, by β-smoothness of $\nabla\mathcal{L}_{\mathcal{B}_k}$, we can use the property of co-coercivity of the gradients:

$$\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) - \nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2 \leq \beta\langle\theta_k - \theta^*, \nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) - \nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\rangle \tag{6.8}$$

Taking expectation over $\mathcal{B}_k$ and using optimality of θ^*:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k) - \nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2] \\
&\leq \beta\langle\theta_k - \theta^*, \nabla\mathcal{L}_{\mathcal{D}}(\theta_k) - \nabla\mathcal{L}_{\mathcal{D}}(\theta^*)\rangle \\
&= \beta\langle\theta_k - \theta^*, \nabla\mathcal{L}_{\mathcal{D}}(\theta_k)\rangle
\end{aligned} \tag{6.9}$$

Combining these elements, we derive:

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[\|\theta_{k+1}-\theta^*\|_2^2] = &\|\theta_k-\theta^*\|_2^2 + 2\eta^2\,\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta_k)-\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2]\\ &+2\eta^2\,\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2] + 2\eta\sigma^2\,\mathbb{E}_{\mathcal{D}}[\|\mathcal{N}(0,\mathbf{I}_d)\|_2^2]\\ &+2\langle\theta_k-\theta^*,\sqrt{2\eta\sigma^2}\mathcal{N}(0,\mathbf{I}_d)\rangle\\ &-2\eta\langle\nabla\mathcal{L}_{\mathcal{D}}(\theta_k),\sqrt{2\eta\sigma^2}\mathcal{N}(0,\mathbf{I}_d)\rangle\\ &-2\eta\langle\theta_k-\theta^*,\nabla\mathcal{L}_{\mathcal{D}}(\theta_k)\rangle\\ \le\ &\|\theta_k-\theta^*\|_2^2 + 2\eta(\eta\beta-1)\langle\theta_k-\theta^*,\nabla\mathcal{L}_{\mathcal{D}}(\theta_k)\rangle\\ &+2\eta^2\,\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2] + 2\eta\sigma^2\,\mathbb{E}_{\mathcal{D}}[\|\mathcal{N}(0,\mathbf{I}_d)\|_2^2]\\ &+2\langle\theta_k-\theta^*,\sqrt{2\eta\sigma^2}\mathcal{N}(0,\mathbf{I}_d)\rangle\\ &-2\eta\langle\nabla\mathcal{L}_{\mathcal{D}}(\theta_k),\sqrt{2\eta\sigma^2}\mathcal{N}(0,\mathbf{I}_d)\rangle\\ &\text{using (6.8) and (6.9)}\end{aligned}$$

Let's assume that $\eta \le \frac{1}{2\beta}$. Therefore, $2\eta(\eta\beta-1) \le -\eta$. By optimality of θ^* and strong convexity arguments on $\mathcal{L}_{\mathcal{D}}$, we deduce the following inequality:

$$\begin{aligned}-\langle\theta_k-\theta^*,\nabla\mathcal{L}_{\mathcal{D}}(\theta_k)\rangle &\le \mathcal{L}_{\mathcal{D}}(\theta_k)-\mathcal{L}_{\mathcal{D}}(\theta^*)+\frac{\lambda}{2}\|\theta_k-\theta^*\|_2^2\\ &\le \lambda\,\|\theta_k-\theta^*\|_2^2\end{aligned}$$

Plugging this together, we have:

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[\|\theta_{k+1}-\theta^*\|_2^2] \le\ &(1-\eta\lambda)\,\|\theta_k-\theta^*\|_2^2 + \eta^2\,\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2]\\ &+2\eta\sigma^2\,\mathbb{E}_{\mathcal{D}}[\|\mathcal{N}(0,\mathbf{I}_d)\|_2^2]\\ &+2\langle\theta_k-\theta^*,\sqrt{2\eta\sigma^2}\mathcal{N}(0,\mathbf{I}_d)\rangle\\ &-2\eta\langle\nabla\mathcal{L}_{\mathcal{D}}(\theta_k),\sqrt{2\eta\sigma^2}\mathcal{N}(0,\mathbf{I}_d)\rangle\end{aligned}$$

We take the expectation again:

$$\begin{aligned}\mathbb{E}[\|\theta_{k+1}-\theta^*\|_2^2] \le\ &(1-\eta\lambda)\,\mathbb{E}[\|\theta_k-\theta^*\|_2^2]\\ &+\eta^2\,\mathbb{E}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2] + 2\eta d\sigma^2\end{aligned} \tag{6.10}$$

Now, by β-smoothness of $\mathcal{L}_{\mathcal{D}}(\cdot)$, we have:

$$\mathcal{L}_{\mathcal{D}}(\theta_K)-\mathcal{L}_{\mathcal{D}}(\theta^*) \le \langle\nabla\mathcal{L}_{\mathcal{D}}(\theta^*),\theta_K-\theta^*\rangle + \frac{\beta}{2}\,\|\theta_K-\theta^*\|_2^2$$

Second, by optimality of θ^* in $\mathcal{C}$ and the fact that $\theta_K \in \mathcal{C}$, we have

$$\langle\nabla\mathcal{L}_{\mathcal{D}}(\theta^*),\theta_K-\theta^*\rangle = 0$$

Combining these two we get:

$$\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K)-\mathcal{L}_{\mathcal{D}}(\theta^*)] \le \frac{\beta}{2}\,\mathbb{E}[\|\theta_K-\theta^*\|_2^2]$$

Using the recursive equation (6.10) repeatedly for $k=0,\dots,K-1$, we have:

$$\begin{aligned}\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K)-\mathcal{L}_{\mathcal{D}}(\theta^*)] \le\ &\frac{\beta}{2}(1-\eta\lambda)^K\,\mathbb{E}[\|\theta_0-\theta^*\|_2^2]\\ &+\frac{\beta}{2}(\eta^2\xi^2+2\eta d\sigma^2)\textstyle\sum_{k=0}^{K-1}(1-\eta\lambda)^k\\ \le\ &\frac{\beta}{2}e^{-\lambda\eta K}\,\mathbb{E}[\|\theta_0-\theta^*\|_2^2] + \frac{\beta\eta\xi^2}{2\lambda} + \frac{\beta d\sigma^2}{\lambda}\\ &\text{where } \xi^2 = \mathbb{E}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2]\end{aligned}$$

□

This shows that under Lipschitz smooth strongly convex loss function, the empirical risk of $\mathcal{A}_{\mathrm{DP-SGLD}}$ decreases as the iterations increase, and reaches a constant factor which is the sum of a term directly related to the variance of the noise σ^2 added at each iteration and another term which represents the error due to the SGD process, which decreases with the learning rate η.

Lemma 24 (Empirical risk for smooth and strongly convex loss, independent of θ)**.** *Let ℓ and $\mathcal{A}_{\mathrm{DP-SGLD}}$ be defined as in Lemma 23, then the empirical risk is bounded by*

$$\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K) - \mathcal{L}_{\mathcal{D}}(\theta^*)] \leq \frac{2\beta L^2}{\lambda^2} e^{-\lambda\eta K} + \frac{\beta\eta\xi^2}{2\lambda} + \frac{\beta d\sigma^2}{\lambda},$$

where $\xi^2 = \mathbb{E}[\|\nabla \mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2]$.

Proof. Since we have $\|\mathcal{C}\| \leq \frac{2L}{\lambda}$, we can bound $\mathbb{E}[\|\theta_0 - \theta^*\|_2^2] \leq \frac{4L^2}{\lambda^2}$, as $\theta_0, \theta^* \in \mathcal{C}$. □

Combining now Lemma 24 and Theorem 20, we derive the utility of $\mathcal{A}_{\mathrm{DP-SGLD}}$ under (α, ε)-Rényi differential privacy.

Theorem 25 (Utility bound for (α, ε)-Rényi differential privacy)**.** *Let $\ell(\theta, \mathbf{x})$ be an L-Lipschitz, λ-strongly convex and β-smooth loss function on closed convex set $\mathcal{C}$, then $\mathcal{A}_{\mathrm{DP-SGLD}}$ with start parameter $\theta_0 \sim \Pi_{\mathcal{C}}(\mathcal{N}(0, \frac{2\sigma^2}{\lambda}\mathbf{I}_d))$ and constant step-size $\eta = \frac{1}{2\beta}$, satisfies (α, ε) Rényi differential privacy and*

$$\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K) - \mathcal{L}_{\mathcal{D}}(\theta^*)] = O\left(\frac{\alpha\beta dL^2}{\varepsilon\lambda^2 n^2}\right) + \frac{\xi^2}{4\lambda},$$

where σ^2 and K are set as such:

$$\sigma^2 = \frac{4\alpha L^2}{\varepsilon\lambda n^2}, \quad K = \frac{2\beta}{\lambda}\log\left(\frac{\varepsilon n^2}{\alpha d}\right).$$

Proof. First, as $\theta_0, \theta^* \in \mathcal{C}$ and $\|C\|_2 \leq \frac{2L}{\lambda}$, we have $\mathbb{E}[\|\theta_0 - \theta^*\|_2^2] \leq \frac{4L^2}{\lambda^2}$. Then, we plug the values of σ^2 and K in Lemma 23:

$$\begin{aligned}\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K) - \mathcal{L}_{\mathcal{D}}(\theta^*)] &\leq \frac{\beta}{2}\frac{4L^2}{\lambda^2} e^{-\lambda\eta K} + \frac{\beta d\sigma^2}{\lambda} + \frac{\beta\eta\xi^2}{2\lambda} \\ &\leq \frac{2\beta L^2}{\lambda^2}\frac{\alpha d}{\varepsilon n^2} + \frac{\beta d}{\lambda}\frac{4\alpha L^2}{\lambda\varepsilon n^2} + \frac{\beta\eta\xi^2}{2\lambda} \\ &\leq \frac{6\alpha\beta dL^2}{\varepsilon\lambda^2 n^2} + \frac{\xi^2}{4\lambda} \text{ with } \eta = \frac{1}{2\beta}\end{aligned}$$

□

Theorem 26 (Utility bound for (ϵ, δ)-differential privacy)**.** *With the same conditions as is Theorem 25, for $\epsilon \leq 2\log(1/\delta)$ and $\delta > 0$, $\mathcal{A}_{\mathrm{DP-SGLD}}$ satisfies (ϵ, δ) differential privacy and*

$$\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K) - \mathcal{L}_{\mathcal{D}}(\theta^*)] = O\left(\frac{\beta dL^2\log(1/\delta)}{\epsilon^2\lambda^2 n^2}\right) + \frac{\xi^2}{4\lambda},$$

where σ^2 and K are set as such:

$$\begin{aligned}\sigma^2 &= \frac{8L^2(\epsilon + 2\log(1/\delta)}{\epsilon^2\lambda n^2} \\ K &= \frac{2\beta}{\lambda}\log\left(\frac{\epsilon^2 n^2}{4\log(1/\delta)d}\right).\end{aligned}$$

Proof. By setting $\varepsilon = \frac{\epsilon}{2}$, we derive from Proposition 3:

$$\alpha = 1 + \frac{2}{\epsilon}\log(1/\delta)$$

.

We use this to rewrite the results from Theorem 25:

$$\begin{aligned}
\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K) - \mathcal{L}_{\mathcal{D}}(\theta^*)] &\leq \frac{6\alpha\beta d L^2}{\varepsilon\lambda^2 n^2} + \frac{\xi^2}{4\lambda} \\
&= \frac{6\beta d L^2}{\lambda^2 n^2}\frac{1+\frac{2}{\epsilon}\log(1/\delta)}{\frac{\epsilon}{2}} + \frac{\xi^2}{4\lambda} \\
&= \frac{6\beta d L^2}{\lambda^2 n^2}\frac{2\epsilon + 4\log(1/\delta)}{\epsilon^2} + \frac{\xi^2}{4\lambda} \\
&\leq \frac{6\beta d L^2}{\lambda^2 n^2}\frac{8\log(1/\delta)}{\epsilon^2} + \frac{\xi^2}{4\lambda} \\
&\text{using } \epsilon \leq 2\log(1/\delta)
\end{aligned}$$

Similarly,

$$\begin{aligned}
\sigma^2 &= \frac{4L^2\alpha}{\lambda n^2 \varepsilon} \\
&= \frac{4L^2}{\lambda n^2}\frac{1+\frac{2}{\epsilon}\log(1/\delta)}{\frac{\epsilon}{2}} \\
&= \frac{8L^2(\epsilon + 2\log(1/\delta)}{\epsilon^2\lambda n^2} \\
K &= \frac{2\beta}{\lambda}\log\left(\frac{n^2}{d}\frac{\varepsilon}{\alpha}\right) \\
&= \frac{2\beta}{\lambda}\log\left(\frac{n^2}{d}\frac{\frac{\epsilon}{2}}{1+\frac{2}{\epsilon}\log(1/\delta)}\right) \\
&= \frac{2\beta}{\lambda}\log\left(\frac{n^2}{d}\frac{\epsilon^2}{2\epsilon + 4\log(1/\delta)}\right) \\
&\leq \frac{2\beta}{\lambda}\log\left(\frac{\epsilon^2 n^2}{4\log(1/\delta)d}\right)
\end{aligned}$$

□

As a side note, arguments of the proof of Lemma 23 (like the co-coercivity of the gradients) can be reused to increase the upper bound on η from $\frac{\lambda}{2\beta^2}$ to $\frac{1}{2\beta}$ in Theorem 4 on the utility of DP-GLD in [CYS21]. We provide experimental evidence in Table 6.4 that this factor $\frac{\beta}{\lambda}$ is non-negligible.

6.4.2 Decreasing Step Size η_k

To remove the $\frac{\xi^2}{4\lambda}$ term which is due to using stochastic gradient descent, we follow the approach from [RSB12] and propose to bound the worst case average empirical risk (6.6) when the step size decreases as such:

$$\eta_k = \frac{1}{2\beta + \frac{\lambda k}{2}}, \; k \geq 0.$$

Lemma 27 (Empirical risk for smooth and strongly convex loss with decreasing learning rate)**.** *Let $\ell(\theta, \mathbf{x})$ be an L-Lipschitz, λ-strongly convex and β-smooth loss function on closed convex set $\mathcal{C}$, $\mathcal{A}_{\text{DP-SGLD}}$ be parameterized with* decreasing *step-size $\eta_k = \frac{1}{2\beta+\lambda k/2}$ and start parameter $\theta_0 \sim \Pi_{\mathcal{C}}(\mathcal{N}(0, \frac{2\sigma^2}{\lambda}\mathbf{I}_d))$, then the average empirical risk of $\mathcal{A}_{\text{DP-SGLD}}$ is bounded by*

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K}\mathcal{L}_{\mathcal{D}}(\theta_k) - \mathcal{L}_{\mathcal{D}}(\theta^*)\right] \leq \frac{2\beta}{K}\mathbb{E}[\|\theta_0 - \theta^*\|_2^2] + \frac{4\xi^2}{K\lambda}\log\left(1 + \frac{\lambda K}{4\beta}\right) + 2d\sigma^2, \tag{6.11}$$

where θ^* is the minimizer of $\mathcal{L}_{\mathcal{D}}(\theta)$ in $\mathcal{C}$.

Proof. We define η_k as such

$$\eta_k = \frac{1}{2\beta + \frac{\lambda k}{2}}.$$

Note that in particular that for $k \geq 0$, $\eta_k \leq \frac{1}{2\beta}$ and hence $2\eta_k(\eta_k\beta - 1) \leq -\eta_k$.
As detailed in the proof of Lemma 23 we have the following:

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[\|\theta_{k+1} - \theta^*\|_2^2] \leq \; & \|\theta_k - \theta^*\|_2^2 + 2\eta_{k+1}(\eta_{k+1}\beta - 1)\langle\theta_k - \theta^*, \nabla\mathcal{L}_{\mathcal{D}}(\theta_k)\rangle \\ & +2\eta_{k+1}^2\,\mathbb{E}_{\mathcal{D}}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2] + 2\eta_{k+1}\sigma^2\,\mathbb{E}_{\mathcal{D}}[\|\mathcal{N}(0, \mathbf{I}_d)\|_2^2] \\ & +2\langle\theta_k - \theta^*, \sqrt{2\eta_{k+1}\sigma^2}\mathcal{N}(0, \mathbf{I}_d)\rangle \\ & -2\eta_{k+1}\langle\nabla\mathcal{L}_{\mathcal{D}}(\theta_k), \sqrt{2\eta_{k+1}\sigma^2}\mathcal{N}(0, \mathbf{I}_d)\rangle \\ & \text{ using (6.8) and (6.9)}\end{aligned}$$

By taking expectation again we derive

$$\begin{aligned}\mathbb{E}[\|\theta_{k+1} - \theta^*\|_2^2] \leq \; & \mathbb{E}[\|\theta_k - \theta^*\|_2^2] \\ & +2\eta_{k+1}(\eta_{k+1}\beta - 1)\,\mathbb{E}[\langle\nabla\mathcal{L}_{\mathcal{D}}(\theta_k), \theta_k - \theta^*\rangle] \\ & +2\eta_{k+1}^2\,\mathbb{E}[\|\nabla\mathcal{L}_{\mathcal{B}_k}(\theta^*)\|_2^2] + 2\eta_{k+1}\sigma^2 d \\ \leq \; & \mathbb{E}[\|\theta_k - \theta^*\|_2^2] \\ & -\eta_{k+1}\,\mathbb{E}[\langle\nabla\mathcal{L}_{\mathcal{D}}(\theta_k), \theta_k - \theta^*\rangle] \\ & +2\eta_{k+1}(\eta_{k+1}\xi^2 + d\sigma^2)\end{aligned}$$

By optimality of θ^* and strong convexity arguments on $\mathcal{L}_{\mathcal{D}}$, we have

$$-\langle\theta_k - \theta^*, \nabla\mathcal{L}_{\mathcal{D}}(\theta_k)\rangle \leq \mathcal{L}_{\mathcal{D}}(\theta_k) - \mathcal{L}_{\mathcal{D}}(\theta^*) + \frac{\lambda}{2}\|\theta_k - \theta^*\|_2^2$$

Hence we have

$$\begin{aligned}\mathbb{E}[\|\theta_{k+1} - \theta^*\|_2^2] \leq \; & (1 - \frac{\eta_{k+1}\lambda}{2})\,\mathbb{E}[\|\theta_k - \theta^*\|_2^2] \\ & -\eta_{k+1}\,\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_k) - \mathcal{L}_{\mathcal{D}}(\theta^*)] \\ & +2\eta_{k+1}(\eta_{k+1}\xi^2 + d\sigma^2)\end{aligned}$$

Equivalently,

$$\begin{aligned}\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_k) - \mathcal{L}_{\mathcal{D}}(\theta^*)] \leq \; & -\frac{1}{\eta_{k+1}}\,\mathbb{E}[\|\theta_{k+1} - \theta^*\|_2^2] \\ & +(\frac{1}{\eta_{k+1}} - \frac{\lambda}{2})\,\mathbb{E}[\|\theta_k - \theta^*\|_2^2] \\ & +2(\eta_{k+1}\xi^2 + d\sigma^2)\end{aligned}$$

Plugging in the definition of η_k we have

$$\begin{aligned}\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_k) - \mathcal{L}_{\mathcal{D}}(\theta^*)] \leq \; & -(2\beta + \frac{\lambda}{2}(k+1))\,\mathbb{E}[\|\theta_{k+1} - \theta^*\|_2^2] \\ & +(2\beta + \frac{\lambda}{2}k)\,\mathbb{E}[\|\theta_k - \theta^*\|_2^2] \\ & +2(\eta_{k+1}\xi^2 + d\sigma^2)\end{aligned}$$

We derive the average empirical risk

$$\begin{aligned}
&\mathbb{E}[\tfrac{1}{K}\textstyle\sum_{k=0}^{K-1}\mathcal{L}_{\mathcal{D}}(\theta_k)-\mathcal{L}_{\mathcal{D}}(\theta^*)] \\
&= \frac{1}{K}\textstyle\sum_{k=0}^{K-1}\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_k)-\mathcal{L}_{\mathcal{D}}(\theta^*)] \\
&= -\frac{1}{K}(2\beta+\frac{\lambda}{2}K)\,\mathbb{E}[\|\theta_K-\theta^*\|_2^2] \\
&\quad+\frac{2\beta}{K}\,\mathbb{E}[\|\theta_0-\theta^*\|_2^2]+2d\sigma^2+\frac{2\xi^2}{K}\sum_{k=1}^{K}\eta_k \\
&\leq \frac{2\beta}{K}\,\mathbb{E}[\|\theta_0-\theta^*\|_2^2]+2d\sigma^2+\frac{2\xi^2}{K}\sum_{k=1}^{K}\frac{1}{2\beta+\frac{\lambda k}{2}} \\
&\leq \frac{2\beta}{K}\,\mathbb{E}[\|\theta_0-\theta^*\|_2^2]+2d\sigma^2+\frac{2\xi^2}{K}\int_0^K\frac{1}{2\beta+\frac{\lambda t}{2}}dt \\
&\leq \frac{2\beta}{K}\,\mathbb{E}[\|\theta_0-\theta^*\|_2^2]+2d\sigma^2+\frac{4\xi^2}{K\lambda}\log\left(1+\frac{\lambda K}{4\beta}\right)
\end{aligned}$$

□

The stochastic term $\frac{\xi^2}{\lambda}$ decreases roughly in $\frac{1}{K}\log(K)$. The term $d\sigma^2$ still appears as in Lemma 24 but without the $\frac{\beta}{\lambda}$ factor.

We then use this lemma to derive the following utility bound under (α,ε)-Rényi differential privacy:

Theorem 28 (Utility bound for (α,ε)-Rényi differential privacy with decreasing learning rate)**.** *Let $\ell(\theta,\mathbf{x})$ be an L-Lipschitz, λ-strongly convex and β-smooth loss function on closed convex set $\mathcal{C}$, then $\mathcal{A}_{\mathrm{DP-SGLD}}$ with start parameter $\theta_0\sim\Pi_{\mathcal{C}}(\mathcal{N}(0,\frac{2\sigma^2}{\lambda}\mathbf{I}_d))$ and* decreasing *step-size $\eta_k=\frac{1}{2\beta+\lambda k/2}$, satisfies (α,ε) Rényi differential privacy and*

$$\mathbb{E}[\frac{1}{K}\sum_{k=1}^{K}\mathcal{L}_{\mathcal{D}}(\theta_k)-\mathcal{L}_{\mathcal{D}}(\theta^*)]=O\left(\frac{\alpha dL^2}{\varepsilon\lambda n^2}\right),$$

where σ^2 and K are set as such:

$$\sigma^2=\frac{4\alpha L^2}{\varepsilon\lambda n^2},\quad K=\max\left(\frac{\beta}{\lambda}\frac{\varepsilon n^2}{\alpha d},\frac{\lambda}{\beta}\left(\frac{\varepsilon n^2}{\alpha d}\right)^2\right).$$

Proof. Since $\theta_0,\theta^*\in\mathcal{C}$ and $\|C\|_2\leq\frac{2L}{\lambda}$, we have $\mathbb{E}[\|\theta_0-\theta^*\|_2^2]\leq\frac{4L^2}{\lambda^2}$. Additionally, by Lipschitzness of the loss we have $\xi^2\leq L^2$.
We can use this and plug the value of σ^2 to rewrite Lemma 27:

$$\begin{aligned}
\mathbb{E}[\tfrac{1}{K}\textstyle\sum_{k=1}^{K}\mathcal{L}_{\mathcal{D}}(\theta_k)-\mathcal{L}_{\mathcal{D}}(\theta^*)] \leq{}& \frac{8\alpha dL^2}{\varepsilon\lambda n^2}+\frac{8\beta L^2}{K\lambda^2} \\
&+\frac{L^2}{\beta}\frac{4\beta}{K\lambda}\log(1+\frac{K\lambda}{4\beta})
\end{aligned}$$

Then, we use the following bound on the logarithmic function from [MV70]:

$$\forall x>-1,\frac{1}{x}\log(1+x)\leq\frac{1}{\sqrt{1+x}}$$

to derive:

$$\frac{L^2}{\beta}\frac{4\beta}{K\lambda}\log(1+\frac{K\lambda}{4\beta})\leq\frac{L^2}{\beta}\frac{1}{\sqrt{1+\frac{K\lambda}{4\beta}}}$$

We can now plug the value of K.

$$\begin{aligned}\mathbb{E}[\tfrac{1}{K}\textstyle\sum_{k=1}^{K}\mathcal{L}_{\mathcal{D}}(\theta_k)-\mathcal{L}_{\mathcal{D}}(\theta^*)] \leq &\frac{8\alpha d L^2}{\varepsilon\lambda n^2}+\frac{8\beta L^2}{K\lambda^2}\\ &+\frac{L^2}{\beta}\sqrt{\frac{4\beta}{\lambda}\frac{\beta}{\lambda}\left(\frac{\alpha d}{\varepsilon n^2}\right)^2}\\ \leq &\frac{18\alpha d L^2}{\varepsilon\lambda n^2}\end{aligned}$$

□

Compared to previous Theorem 25, we improve the utility bound by a factor $\frac{\beta}{\lambda}$ which is non negligible in practice. However, the number of iterations K can now evolve either in n^2 or in n^4 in the regime where $\frac{\varepsilon n^2}{\alpha d} > (\frac{\beta}{\lambda})^2$. Typical values from our experiments, which are reported in Table 6.4, show that K evolves in the n^2 regime.

Theorem 29 (Utility bound for (ϵ,δ)-differential privacy with decreasing learning rate)**.** *With the same conditions as is Theorem 28, for $\epsilon \leq 2\log(1/\delta)$ and $\delta > 0$, $\mathcal{A}_{\text{DP-SGLD}}$ satisfies (ϵ,δ) differential privacy and*

$$\mathbb{E}[\mathcal{L}_{\mathcal{D}}(\theta_K)-\mathcal{L}_{\mathcal{D}}(\theta^*)] = O\left(\frac{dL^2\log(1/\delta)}{\epsilon^2\lambda n^2}\right),$$

where σ^2 and K are set as such:

$$\begin{aligned}\sigma^2 &= \frac{8L^2(\epsilon+2\log(1/\delta)}{\epsilon^2\lambda n^2}\\ K &= \max\left(\frac{\beta}{\lambda}\frac{\epsilon^2 n^2}{4\log(1/\delta)d},\frac{\lambda}{\beta}\left(\frac{\epsilon^2 n^2}{4\log(1/\delta)d}\right)^2\right).\end{aligned}$$

Proof. By setting $\varepsilon = \frac{\epsilon}{2}$, we derive from Proposition 3:

$$\alpha = 1+\frac{2}{\epsilon}\log(1/\delta)$$

.

We use this to rewrite the results from Theorem 28:

$$\begin{aligned}\mathbb{E}[\tfrac{1}{K}\textstyle\sum_{k=1}^{K}\mathcal{L}_{\mathcal{D}}(\theta_k)-\mathcal{L}_{\mathcal{D}}(\theta^*)] &\leq \frac{18\alpha d L^2}{\varepsilon\lambda n^2}\\ &= \frac{18dL^2}{\lambda n^2}\frac{1+\frac{2}{\epsilon}\log(1/\delta)}{\frac{\epsilon}{2}}\\ &= \frac{18dL^2}{\lambda n^2}\frac{2\epsilon+4\log(1/\delta)}{\epsilon^2}\\ &\leq \frac{18dL^2}{\lambda n^2}\frac{8\log(1/\delta)}{\epsilon^2}\\ &\quad \text{using } \epsilon \leq 2\log(1/\delta)\end{aligned}$$

Similarly,

$$\begin{aligned}\sigma^2 &= \frac{4L^2\alpha}{\lambda n^2\varepsilon}\\ &= \frac{4L^2}{\lambda n^2}\frac{1+\frac{2}{\epsilon}\log(1/\delta)}{\frac{\epsilon}{2}}\\ &= \frac{8L^2(\epsilon+2\log(1/\delta)}{\epsilon^2\lambda n^2}\end{aligned}$$

$$
\begin{aligned}
\frac{\epsilon n^2}{\alpha d} &= \frac{n^2}{d}\frac{\frac{\epsilon}{2}}{1+\frac{2}{\epsilon}\log(1/\delta)} \\
&= \frac{n^2}{d}\frac{\epsilon^2}{2\epsilon + 4\log(1/\delta)} \\
&\leq \frac{\epsilon^2 n^2}{4\log(1/\delta)d}
\end{aligned}
$$

$$
\begin{aligned}
K &= \max\left(\frac{\beta}{\lambda}\frac{\epsilon n^2}{\alpha d}, \frac{\lambda}{\beta}\left(\frac{\epsilon n^2}{\alpha d}\right)^2\right) \\
&\leq \max\left(\frac{\beta}{\lambda}\frac{\epsilon^2 n^2}{4\log(1/\delta)d}, \frac{\lambda}{\beta}\left(\frac{\epsilon^2 n^2}{4\log(1/\delta)d}\right)^2\right)
\end{aligned}
$$

□

6.5 Experiments: Application to Logistic Regression

We now propose an experimental evaluation of DP-SGLD and compare it to DP-SGD on a classification task using logistic regression on two vision datasets, CIFAR10 and Pneumonia, a dataset of chest X-ray images of pediatric pneumonia published in [KGC+18]. Details about the datasets and models are available in Appendix B.2.

The task consists of pre-training a model (here AlexNet or ResNet18) on a dataset that will be considered *public* (here we take CIFAR100 and Imagenet). Then, all layers of the model are frozen except for the last one which is retrained from scratch using a softmax loss function on a *private* dataset (here CIFAR10 or Pneumonia). This corresponds to multinomial logistic regression and some regularization is added to guarantee strong convexity. A formal description of multinomial logistic regression and an analysis of its smoothness and convexity constants are provided in Appendix B.1. Pre-training provides generic feature maps learned on a public dataset which improves the task accuracy without any compromise on privacy.

We compare our DP-SGLD algorithm with the standard DP-SGD from [ACG+16] implemented in Opacus and with the baseline SGD without DP on several vision tasks. In particular, we study the case where the step size is constant $\eta = \frac{1}{2\beta}$ and where it is decreasing as follows: $\eta_k = \frac{1}{2\beta+\lambda k/2}$. To be able to provide somewhat comparable results, all methods (DP-SGLD, DP-SGD and No-DP) use the same step size, number of training epochs, privacy budget $(\epsilon, \delta) = (1.0, 10^{-5})$ when applicable and use no momentum. Other hyperparameters are tuned to provide optimal accuracy for each method and are provided in the source code which is available online[2].

Results are given in Table 6.1 for constant step size and in Table 6.2 for decreasing step size. The model indicated is the feature extraction model, which is pre-trained on CIFAR100 when the task is on CIFAR10 and on Imagenet when the task is on Pneumonia. Only its last layer is re-trained using logistic regression. As the tables show, DP-SGLD outperforms standard DP-SGD for the tasks considered and considerably reduces the gap in accuracy compared to SGD without differential privacy. However, such results need to be taken cautiously before drawing conclusions since this task is strongly convex and smooth while DP-SGD also applies to non-convex tasks.

To better understand the effect of clamping DP-SGD to smooth and strongly convex tasks, we repeat the first experiment of Table 6.1, but instead of leveraging only the last layer for DP-SGD, we also unfreeze the last and fourth block of the ResNet18 architecture, composed notably of 5 convolutional layers. Results for DP-SGD and SGD without DP provided in Table 6.3 show that while SGD without DP benefits from this fine-tuning and increases accuracy

[2] The code is available at: `https://github.com/LaRiffle/langevin`

Table 6.1: Accuracy (in %) of logistic regression using SGD with a constant learning rate.

Method	Dataset	Model	Epochs	ϵ	Accuracy
DP-SGLD	CIFAR10	Resnet18	30	1.0	70.3 ± 0.0
DP-SGD	CIFAR10	Resnet18	30	1.0	68.0 ± 0.2
No DP	CIFAR10	Resnet18	30	-	70.7 ± 0.1
DP-SGLD	CIFAR10	Alexnet	30	1.0	57.5 ± 0.1
DP-SGD	CIFAR10	Alexnet	30	1.0	56.4 ± 0.1
No DP	CIFAR10	Alexnet	30	-	57.7 ± 0.1
DP-SGLD	Pneumonia	Resnet18	50	1.0	58.8 ± 0.5
DP-SGD	Pneumonia	Resnet18	50	1.0	58.8 ± 0.3
No DP	Pneumonia	Resnet18	50	-	59.3 ± 0.6

Table 6.2: Accuracy (in %) of logistic regression using SGD with a decreasing learning rate.

Method	Dataset	Model	Epochs	ϵ	Accuracy
DP-SGLD	CIFAR10	Resnet18	30	1.0	70.1 ± 0.1
DP-SGD	CIFAR10	Resnet18	30	1.0	68.1 ± 0.2
No DP	CIFAR10	Resnet18	30	-	70.2 ± 0.1
DP-SGLD	CIFAR10	Alexnet	30	1.0	57.3 ± 0.1
DP-SGD	CIFAR10	Alexnet	30	1.0	56.4 ± 0.1
No DP	CIFAR10	Alexnet	30	-	57.6 ± 0.1
DP-SGLD	Pneumonia	Resnet18	50	1.0	58.8 ± 0.5
DP-SGD	Pneumonia	Resnet18	50	1.0	58.8 ± 0.3
No DP	Pneumonia	Resnet18	50	-	59.3 ± 0.6

from 70.7% to 77.0%, DP-SGD does not improve and accuracy even decreases marginally from 68.0% to 67.8%. Such observation aligns with those from [TB21] in the sense that basic models like logistic regression currently are competitive compared to deeper models when trained with differential privacy, which underlines the importance of studying classical tasks like training smooth and strongly convex objectives.

Last, we provide in Table 6.4 the experimental value of some parameters, in light with comments made after Lemma 27 about the value of $\frac{\beta}{\lambda}$ and after Theorem 28 about the dependence in n^2 or n^4 of K, depending of the ratio $\frac{\varepsilon n^2}{\alpha d}/(\frac{\beta}{\lambda})^2$. As we show, this ratio is of magnitude 10^{-3} or less which shows that the evolution of K is quadratic in n.

Table 6.3: Accuracy (in %) when fine-tuning ResNet18.

Method	Dataset	Model	Epochs	ϵ	Accuracy
DP-SGD	CIFAR10	Resnet18	30	1.00	67.8 ± 0.2
No DP	CIFAR10	Resnet18	30	-	77.0 ± 0.2

Table 6.4: Value of some parameters used for DP-SGLD.

Dataset	Model	β	$\frac{\beta}{\lambda}$	$\frac{\varepsilon n^2}{\alpha d}/\left(\frac{\beta}{\lambda}\right)^2$
CIFAR10	Resnet18	55	5.5×10^4	3.2×10^{-3}
CIFAR10	Alexnet	259	2.6×10^5	1.5×10^{-4}
Pneumonia	Resnet18	354	7.1×10^4	6.8×10^{-5}

6.6 Conclusion

We have extended the theoretical framework from [CYS21] to provide a differential privacy analysis of noisy stochastic gradient descent based on Langevin diffusion (DP-SGLD) with arbitrary step size. Although our experiments already show the practical utility of our results, relaxing the smoothness and strong convexity hypothesis remains an open challenge and would pave the way for wide adoption by data scientists.

This work is one example of the promising interactions between different privacy-preserving methods, and showcases that differential privacy when combined with private computation such as SMPC can deliver more robust guarantees. In a similar spirit, works like [GDD⁺21, KLS21] explore combinations of differential privacy with federated learning and also discover interesting synergies like the ability to sample contributions from clients to improve privacy guarantees.

Chapter

7 Conclusion

7.1 Summary of the Results

We have opened the manuscript with the notion of contextual integrity which proposes a new vision of privacy, based on appropriate information disclosure instead of blind concealment. This notion is a leitmotiv of our work, since the techniques explored fit gently in this paradigm and bring complementary guarantees to privacy of sensitive machine learning tasks.

In chapter 3, we have first explored design architectures for privacy-preserving machine learning and in particular federated learning. The modularity of the PySyft library allows to easily intertwine remote execution, encryption of secret data and other mechanisms, which makes it a very interesting tool box for researchers at the crossroads of privacy-enhancing technologies. This library has served as a foundation for our implementations in the following chapters, and some of our protocols like AriaNN have been directly integrated into the library.

Then, in chapter 4, we have proposed a new type of privacy attack by leveraging one common weakness of fully encrypted networks which is that they directly disclose the output signal without applying a filter (like an argmax to only output the predicted class) or any other quantization mechanism that would reduce the information leakage. Since most standard vision datasets only classify on one set of features, we introduced a new dataset which allows to perform classification on two orthogonal sets of features, one being the intended task to perform and the other being private features that should not be exposed. We show that by default models do not respect this orthogonality and that classification on one set informs about the other set of features, a phenomenon that we call collateral learning. We develop adversarial training methods to reduce considerably collateral learning without degrading significantly the accuracy. This attack and the others mentioned in the chapter illustrate that models in machine learning are not neutral and that they can be exploited to reveal more information than expected, either on the training data or on data used at inference time.

Next, chapter 5 gathers several approaches for fully private inference or training of neural networks. We first leverage a new functional encryption scheme to perform semi-encrypted inference of small networks and we show that these networks are sensitive to the collateral learning attack described in the previous chapter. Then we propose a new function secret sharing protocol for comparison and we show that it allows us to do fully private training and inference for a large variety of neural networks. In particular, we show that this protocol requires less interactions than other MPC alternatives, which makes it very competitive in scenarios where latency between parties is high. We provide an implementation which can run on GPUs and show that models like ResNet18 that perform better than humans on some medical tasks [KZPP+21] can perform private inference in a couple of seconds.

Last, we analyze in chapter 6 an interesting synergy between differential privacy and encrypted training, i.e. a setting where the model is not visible to any party until the end of the training. We build upon an existing framework which analyzes the evolution of Rényi divergence using Langevin diffusion, and we propose a new stochastic gradient algorithm for the class of smooth strongly convex functions which notably provides convergence of the privacy loss regardless of the number of training iterations. We show on concrete learning tasks that building good features for differential privacy is key to reaching a good accuracy, but that training deeper networks can be of little interest on vision tasks when a high privacy level is required. This last chapter shows that privacy-enhancing techniques can be combined together and achieve higher guarantees than the sum of the guarantees if taken separately.

7.2 Applications in Healthcare

We have been interested in applying privacy-enhancing technologies (PETs) to a concrete domain, and have chosen healthcare where we believe such techniques could foster collaboration and benefit the largest number.

Healthcare is indeed a prime example when it comes to illustrating privacy methods, together with finance as well which also appears regularly. Everyone cares about protecting its health data, so much so that this area has not waited for PETs to develop strong regulation that provides a rigorous framework on how medical data should be collected, transformed, stored or reused. Interestingly as well, this area is probably one where the deployment of PETs might be the most complicated. Here are three main barriers that we believe hinders their adoption.

7.2.1 Technical: Heterogeneity of Healthcare Data

Let's distinguish (at least) three main categories of data: images (like MRI, X-rays, CT-scans, etc), structured data (measures, dates, names, etc) and unstructured text (like medical records, prescriptions, etc).

Images are a special category, since they benefit from longstanding standards among which DICOM that was created back in 1985. Hence, most of the servers hosting medical images are compatible with DICOM, which makes data retrieval easier. Images also have inherently a structured data type, which makes their processing simpler. For example, we have been able to use quite easily the dataset of chest X-ray images of pediatric pneumonia from [KGC+18] in our work [KZPP+21] to experiment encrypted inference using a ResNet18, or also in [RBP22] to benchmark our differential privacy algorithm DP-SGLD. The one but not least challenge with medical images is the size of the data, since one complete exam generates around 1GB of data. As a consequence, image servers can easily be overloaded, which means that building large training datasets without impacting the medical teams' work is a real challenge.

The second type of data is structured data, which are typically added into the hospital IT system by the care team using medical software or by sensors that also transmit measurements back to some software. Each medical software stores its data in a proprietary and unique data format, which does not follow any standard practice. This makes it very hard for any data scientist to retrieve the relevant data, since it requires that the user should be granted access to all relevant databases where the data might be siloed, together with sufficient information (or time) to understand the way the data is organized. New initiatives to change this paradigm exist in several countries including in France. They aim to aggregate all the data generated in the hospital in a single standardized data warehouse, so that the hospital recovers full sovereignty on the data it collects and can more easily provide tailored access to data scientists or research studies. Such initiatives rely on longstanding standard formats, like FHIR (2014) that enables the modeling of all medico-administrative data from health care institutions, or OMOP (2009) that is primarily intended for clinical research. Beyond syntaxic standardization, some concepts like medication, treatments or diseases can also be standardized semantically using medical ontologies like ICD-10 (1994), ICD-11 (2022) or SNOMED CT (2007) which was very recently adopted in France. Such standardization of data formats and concepts is very helpful to lower the entry barrier for data scientists in the general setting, but it is a necessary prerequisite for PETs like (horizontal) federated learning that requires a single model to train on several datasets in different hospitals, implying that these datasets should share together the same format and comparable concepts. Furthermore, for fully encrypted learning where the data is not visible by the analyst, such strong guarantees on the data quality are also necessary as preprocessing capabilities can be very limited.

Last, unstructured text like medical records. This type of data is probably the most complicated to leverage using PETs. Indeed, standard models used in NLP are typically based on Transformer architectures [VSP+17] which are composed of hundred of million of parameters, which make them poor candidates both for federated learning as they should be sent over the network multiple times and for encrypted computation, which is significantly slower than plaintext computation, which in turns already requires GPUs to run in a reasonable time. In addition, the specificity of the medical jargon adds substantial difficulty and limits the direct reuse of pretrained models, even in english [LYK+20]. In this situation, prior structuration in clear text by the healthcare facility or a partner can help consolidate and make available important information contained in the medical records, so that they can then be analyzed for different use cases using standard non-NLP learning techniques.

7.2.2 Regulatory: a Robust but Rigid Framework

We will address this barrier specifically for France, as we are more familiar with this case. Using or gathering medical data for research purposes is strictly regulated in France. Such regulation, including GDPR (2018) or the recent "Référentiel Entrepôt de Données de Santé (EDS)" (2021) issued by the French National Commission on Informatics and Liberty (CNIL), gives a clear definition on how medical data should be collected, stored and reused. In particular, the Référentiel EDS states that all data to be used for (external) research purpose should be pseudonymized, which can be extremely challenging on unstructured text like medical records for example, where the history of the disease of a patient can provide sufficient details to re-identify the subject.

Such regulation provides guarantees on the patient's privacy and sovereignty over its data, and at the same time provides a path for research to be conducted. However, it also limits fundamentally the scope of research and collaboration which can plausibly be attained, because it makes the strong assumption that the data is transmitted through stakeholders

At least in France, even more established privacy-preserving technologies like federated learning do not have any recognized status, meaning that they should fit in the same frame as non-private approaches. In the case of federated learning, it means that the data should be collected and pseudonymized just like if it were to be disclosed directly to researchers. Note that models, which are actually the only elements directly accessible in federated learning, are also not compliant to GDPR according to [CD18]. This is in line with the vulnerability to attacks that we have illustrated in the previous chapters and calls for additional measures including differential privacy. In the case of differential privacy specifically, except for a very small number of examples including its adoption by the US Census Bureau [Abo18], it has not yet received attention from political nor regulatory institutions, and hence does not benefit from the recognition or the convenient framework necessary to encourage its adoption even in very limited areas of healthcare like the monitoring of care indicators for example. In a sense, differential privacy still lacks its "Référentiel".

Several reasons account for this situation. First, such techniques are relatively new and regulation does not follow the pace of technology breakthroughs. This is a very crucial issue since every major breakthrough can potentially benefit from a time frame where no regulation controls the actors that emerge. In this regard, the position that consists of being very conservative by default on medical data analysis is easily understandable. The technicality of these breakthroughs and the non negligible probability that they might not actually be exploited is a challenge for lawmakers that should be able to prioritize and understand in depth the topics that will affect most public health. Second, even when a topic is identified as critical and when appropriate dissemination is available for non-specialists, creating the appropriate regulation can be extremely challenging, because it requires to set the proper frontier between what should be allowed but could also misused through attacks or abuse and what should be prohibited but could also help extending public knowledge and make progress. The example of differential privacy is very illustrative of this, because this technology provides quantitative estimation of the privacy leakage together with the ability to understand the trade-off between privacy and utility, which makes it a very practical tool. However, setting a numerical threshold on the privacy leakage that is acceptable is a responsibility that no regulatory authority is willing to take. On the one hand, everyone has in mind the dramatic impact that would have successful attacks against systems that would have followed a threshold guideline which turns out to be too permissive. On the other hand, thresholds that are too conservative would be impossible to follow for systems that need a high level of utility and have additional constraints like limited access to data. Another characteristic with differential privacy is that data is no longer imperishable: once the privacy budget has been spent for a data item, it should not be used anymore. This vision stands in contrast with the long running idea that personal data is a gold mine: in the world of differential privacy, personal data is not a renewable resource. Hence, it must be used wisely, and this raises prominent questions as a user about how I control and give access to my personal data. Setting clear regulation around this and defining a status for depleted data seems also incredibly challenging.

To sum up on this regulatory aspect, since PETs are inherently more difficult to use (not only because the tooling is not as developed than standard machine learning but also because data is by default not accessible directly) and since they do not benefit from regulatory accommodation, it is unlikely that they should be preferred over standard machine learning, especially in the medical context that is by itself complicated enough, except probably for reputational reasons.

7.2.3 Cross Domain Expertise

The last barrier is also very specific to healthcare. To be able to successfully and securely engage into a data analysis in a medical context, one should first have sufficient medical knowledge to be able to have a critical opinion over the data available, since data in a hospital is collected through software that scarcely check its coherence and consistency (it is common to find two conflicting information about the same patient in two different databases in the same hospital). Second, as mentioned above, data is siloed in different databases that all have different formats, so for all institutions which are not yet equipped with data warehouses, one should also be an experienced data analyst. Third, to be able to use PETs wisely, meaning being able to realize that some techniques might be overworked for a given context or conversely that such techniques might not be sufficient, one should be familiar with security and cryptography. Last, and surprisingly least important, one should have basic knowledge of data science, since most of the models are available through very user friendly libraries.

Providing tools and educational support that reduce the minimal set of knowledge a user should have in these cardinal domains is an important challenge as a community of researchers. In particular, all areas (except the medical one probably) could all become as user friendly as data science, provided that appropriate software or libraries are developed and made open-source. It is one of the goals of the OpenMined community to which we have contributed throughout this thesis.

Those three barriers illustrate on the specific example of healthcare the difficulties of transitioning from theoretical models to real life deployment. At each step of this process, all players should be aware that their domain might not be easily understood by other areas and do their best in facilitating coordination. Researchers can communicate their code and implementations, or provide additional support to onboard new comers on complex topics ; open-source developers can put a premium on code quality when building implementations; actors that access real life data can try to build benchmark datasets and advertise their use cases; etc. Such awareness and commitment are a crucial factor to accelerate adoption of new technologies by the greatest number and for the best interests.

7.3 Perspectives

We have explored separately several directions for privacy-preserving machine learning and have started to analyze interactions between them in chapter 6 where we combine encrypted computation and differential privacy. This interaction was also studied in other works like [YSMN21]. Recently as well, other interactions have received some attention like the one between differential privacy and federated or decentralized learning [BGTT18, GDD$^+$21], which is very promising since differential privacy needs an order of magnitude more data to keep a good utility / privacy tradeoff [TB21]. The other very interesting interaction which combines federated learning, differential privacy and a simple version of encrypted computation (only for the secure aggregation phase) was studied in [KLS21].

It is likely that these interactions will continue to be increasingly studied as each individual field reaches a certain level of maturity, which is reflected by the emergence of well supported open-source libraries. These libraries, including PySyft [RTD$^+$18], Crypten [KVH$^+$21], Opacus [YSS$^+$21], TensorFlow Privacy or TensorFlow Federated are almost up to date with the major breakthroughs in their respective areas and provide scientists with practical tools to use these technologies and focus on the next privacy challenges.

Last, as the last section about applications to healthcare suggests, privacy even considered through the prism of contextual integrity is not only a technical issue but also a social, political, legal and even economic one. Finding solutions to such an important challenge, which impacts society as a whole, requires far more cross domain fertilizations in order to share awareness between the different stakeholders who build tomorrow's privacy standards.

Appendix

A Private evaluation and training of neural networks

A.1 Partially Encrypted Machine Learning using Functional Encryption

A.1.1 Our Quadratic Functional Encryption Scheme - Proofs

Proofs of IND-CPA Security

To prove security of our scheme, we use the Generic Bilinear Group Model, which captures the fact that no attacks can make use of the representation of group elements. For convenience, we use Maurer's model [Mau05], where a third party implements the group and gives access to the adversary via handles, providing also equality checking. This is an alternative, but equivalent, formulation of the Generic Group Model, as originally introduced in [Nec94, Sho97].

We prove security in two steps: first, we use a master theorem from [BCFG17] that relates the security in the Generic Bilinear Group model to a security in a symbolic model. Second, we prove security in the symbolic model. Let us now explain the symbolic model (the next paragraph is taken verbatim from [ABGW17]).

In the symbolic model, the third party does not implement an actual group, but keeps track of abstract expressions. For example, consider an experiment where values x, y are sampled from $\mathbb{Z}_p$ and the adversary gets handles to g^x and g^y. In the generic model, the third party will choose a group of order p, for example $(\mathbb{Z}_p, +)$, will sample values $x, y \leftarrow_R \mathbb{Z}_p$ and will give handles to x and y. On the other hand, in the symbolic model the sampling will not be performed and the third party will output handles to X and Y, where X and Y are abstract variables. Now, if the adversary asks for equality of the elements associated to the two handles, the answer will be negative in the symbolic model, since abstract variable X is different from abstract variable Y, but there is a small chance the equality check succeeds in the generic model (only when the sampling of x and y coincides).

To apply the master theorem, we first need to change the distribution of the security game to ensure that the public key, challenge ciphertext, and functional decryption keys only contain group elements whose exponent is a polynomial evaluated on uniformly random values in $\mathbb{Z}_p$ (this is called polynomially induced distributions in [BCFG17, Definition 10], and previously in [BFF$^+$14]). We show that this is possible with only a negligible statistical change in the distribution of the adversary view.

After applying the master theorem from [BCFG17], we prove the security in the symbolic model (cf. Appendix 31), which simply consists of checking that an algebraic condition on the scheme is satisfied.

Theorem 30 (IND-CPA Security in the Generic Bilinear Group Model). *For any PPT adversary $\mathcal{A}$ that performs at most Q group operations against the functional encryption scheme described on 5.1, we have, in the generic bilinear group model:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FE}}(\lambda) \leq \frac{12 \cdot (6n + 3 + Q + Q')^2 + 1}{p},$$

where Q' is the number of queries to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$.

The proof of this result is quite technical and can be found in the dedicated Appendix A.1.3.

Proof of Correctness

For all $i, j \in [n]$, we have:

$$e(g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_j}) = g_T^{\boldsymbol{a}_i \cdot \boldsymbol{b}_j} = g_T^{x_i y_j - \gamma s_i t_j}$$

since

$$\begin{aligned}\boldsymbol{a}_i \cdot \boldsymbol{b}_j &= \left((\mathbf{W}^{-1})^\top \begin{pmatrix} x_i \\ \gamma s_i \end{pmatrix} \right)^\top \cdot \left(\mathbf{W} \begin{pmatrix} y_j \\ -t_j \end{pmatrix} \right) \\ &= \begin{pmatrix} x_i \\ \gamma s_i \end{pmatrix}^\top \mathbf{W}^{-1} \mathbf{W} \begin{pmatrix} y_j \\ -t_j \end{pmatrix} = x_i y_j - \gamma s_i t_j.\end{aligned}$$

Therefore we have:

$$\begin{aligned}out = e(g_1^\gamma, g_2^{q(\boldsymbol{s},\boldsymbol{t})}) \cdot \prod_{i,j} e(g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_i})^{q_{i,j}} &= g_T^{\gamma q(\boldsymbol{s},\boldsymbol{t})} \cdot g_T^{\sum_{i,j} q_{i,j} x_i y_j - \gamma q_{i,j} s_i t_j} \\ = g_T^{\gamma q(\boldsymbol{s},\boldsymbol{t})} \cdot g_T^{q(\boldsymbol{x},\boldsymbol{y}) - \gamma q(\boldsymbol{s},\boldsymbol{t})} = g_T^{q(\boldsymbol{x},\boldsymbol{y})}.\end{aligned}$$

Proof of Complexity

The complexity can be inferred from the decryption phase as detailed in Figure 5.1 and we compare this with previous quadratic FE schemes in Figure A.1.

FE scheme	*ct*	dk_f	Dec	Assumption
[BCFG17, Sec. 3]	$\mathbb{G}_1^{6n+1} \times \mathbb{G}_2^{6n+1}$	$\mathbb{G}_1 \times \mathbb{G}_2$	$6n^2(E_1 + P) + 2P$	SXDH, 3PDDH
[BCFG17, Sec. 4]	$\mathbb{G}_1^{2n+1} \times \mathbb{G}_2^{2n+1}$	$\mathbb{G}_1^2$	$3n^2(E_1 + P) + 2P$	GGM
Ours	$\mathbb{G}_1^{2n+1} \times \mathbb{G}_2^{2n}$	$\mathbb{G}_2$	$2n^2(E_1 + P) + P$	GGM

Figure A.1: Performance comparison of FE for quadratic polynomials. E_1 and P denote exponentiation in $\mathbb{G}_1$ and pairing evaluation, respectively. Decryption additionally requires solving a discrete logarithm but this computational overhead is the same for all schemes and is therefore omitted here.

Detailed Equivalence of the FE Scheme with a Neural Network

Proof of Linear Homomorphism

For all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^n$, and $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^n$, given an encryption of $(\boldsymbol{x}, \boldsymbol{y})$ under the public key $\mathsf{pk} := (g_1^{\boldsymbol{s}}, g_2^{\boldsymbol{t}})$, one can efficiently compute an encryption of $(\boldsymbol{u}^\top \boldsymbol{x}, \boldsymbol{v}^\top \boldsymbol{y})$ under the public key $\mathsf{pk}' := (g_1^{\boldsymbol{u}^\top \boldsymbol{s}}, g_2^{\boldsymbol{v}^\top \boldsymbol{t}})$. Indeed, given

$$\mathsf{Enc}(\mathsf{pk}, (\boldsymbol{x}, \boldsymbol{y})) := (g_1^\gamma, \{g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_i}\}_{i \in [n]}),$$

and $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}_p^n$, one can efficiently compute:

$$(g_1^\gamma, g_1^{\sum_{i \in [n]} u_i \cdot \boldsymbol{a}_i}, g_2^{\sum_{i \in [n]} v_i \cdot \boldsymbol{b}_i}),$$

which is $\mathsf{Enc}(\mathsf{pk}', (\boldsymbol{u}^\top \boldsymbol{x}, \boldsymbol{v}^\top \boldsymbol{y}))$, since:

$$\sum_{i\in[n]} u_i \cdot \boldsymbol{a}_i = \sum_{i\in[n]} u_i \cdot (\mathbf{W}^{-1})^\top \begin{pmatrix} x_i \\ \gamma s_i \end{pmatrix} = (\mathbf{W}^{-1})^\top \begin{pmatrix} \sum_{i\in[n]} u_i \cdot x_i \\ \gamma \sum_{i\in[n]} u_i \cdot s_i \end{pmatrix}$$
$$= (\mathbf{W}^{-1})^\top \begin{pmatrix} \boldsymbol{u}^\top \boldsymbol{x} \\ \gamma \boldsymbol{u}^\top \boldsymbol{s} \end{pmatrix}.$$

Similarly, we have:

$$\sum_{i\in[n]} v_i \cdot \boldsymbol{b}_i = \sum_{i\in[n]} v_i \cdot \mathbf{W} \begin{pmatrix} y_i \\ -t_i \end{pmatrix} = \mathbf{W} \begin{pmatrix} \boldsymbol{v}^\top \boldsymbol{y} \\ -\boldsymbol{v}^\top \boldsymbol{t} \end{pmatrix}.$$

A.1.2 Additional Results

Influence of Weight Compression on the Network Performance

We show here that we can manage to compress significantly the network weights in order to have a very fast discrete logarithm without modifying the results and conclusions made throughout the article. The main and collateral model follow the same CNN structure as stated above, and the collateral accuracy is reported after 10 epochs of training.

Main accuracy with compression	97.72 ± 0.30 %
Collateral accuracy with compression	55.27 ± 0.41 %

Table A.1: Impact of weight compression on the main and collateral accuracies

Influence of α during Adversarial Training

To choose the best value for α, we have chosen an output size of 4 which allows us to keep a very high main accuracy while significantly reducing the collateral one, as shown in Figure 4.4. We observe that the semi-adversarial training does not affect significantly the main accuracy for a large range of values for α, while its impact on the collateral accuracy is decisive. Figure A.2 illustrates the role of α and justify our choice of $\alpha = 1.7$. For this experiment, we have chosen for both networks a simple feed forward with a hidden layer of 32 neurons.

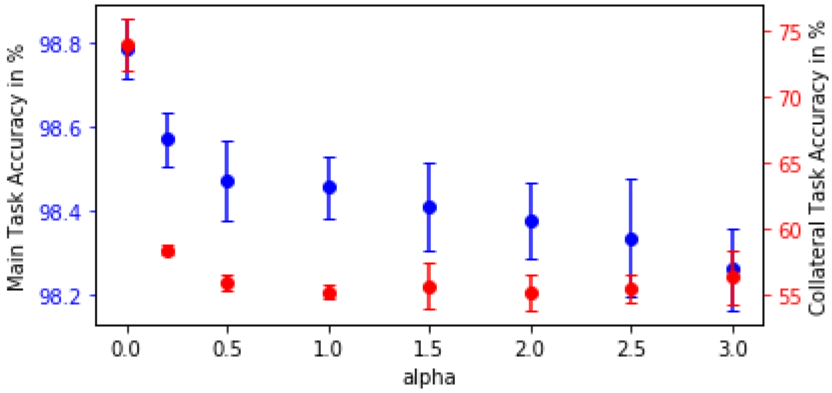

Figure A.2: Trade-off between the main and collateral tasks accuracies as a function of α

```
$\mathsf{Exp}_1(1^\lambda, \mathcal{A})$:
$(\mathbb{G}_1, \mathbb{G}_2, p, g_1, g_2, e) \leftarrow \mathsf{GGen}(1^\lambda)$, $\boldsymbol{s}, \boldsymbol{t} \xleftarrow{\$} \mathbb{Z}_p^n$
$a, b, c, d \xleftarrow{\$} \mathbb{Z}_p$, set $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, g_1^{ad-bc}, g_2, e)$
$\mathsf{msk} := (\boldsymbol{s}, \boldsymbol{t})$, $\mathsf{pk} := \left(\mathcal{PG}, g_1^{(ad-bc)\boldsymbol{s}}, g_2^{\boldsymbol{t}}\right)$
$\left((\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}), (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})\right) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}, \cdot)}(\mathsf{pk})$
$\beta \xleftarrow{\$} \{0,1\}$, $\gamma \xleftarrow{\$} \mathbb{Z}_p$
for all $i \in [n]$, $\boldsymbol{a}_i := \begin{pmatrix} d & -c \\ -b & a \end{pmatrix} \begin{pmatrix} x_i^{(\beta)} \\ \gamma s_i \end{pmatrix}$, $\boldsymbol{b}_i := \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} y_i^{(\beta)} \\ -t_j \end{pmatrix}$
$ct =: (g_1^{\gamma(ad-bc)}, \{g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_i}\}_{i \in [n]})$
$\beta' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}, \cdot)}(\mathsf{pk}, ct)$
Return 1 if $\beta' = \beta$ and for all queried $f$, $f(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}) = f(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})$.

$\mathsf{KeyGen}(\mathsf{msk}, f)$:
return $(g_2^{f(\boldsymbol{s},\boldsymbol{t})}, f)$.
```

Figure A.3: Experiment Exp_1, for the proof of Theorem 30.

A.1.3 Security Proof of our FE Scheme

Proof. For any experiment Exp, adversary $\mathcal{A}$, and security parameter $\lambda \in \mathbb{N}$, we use the notation: $\mathsf{Adv}_{\mathsf{Exp}}(\mathcal{A}) := \Pr[1 \leftarrow \mathsf{Exp}(1^\lambda, \mathcal{A})]$, where the probability is taken over the random coins of Exp and $\mathcal{A}$.

While we want to prove the security result in the real experiment Exp_0, in which the adversary has to guess β, we slightly modify it into the hybrid experiment Exp_1, described in A.3: we write the matrix $\mathbf{W} \xleftarrow{\$} \mathsf{GL}_2$ used in the challenge ciphertext as $\mathbf{W} := \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, chosen from the beginning. Then $\mathbf{W}^{-1} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$.

The only difference with the IND-CPA security game as defined in Appendix 5.1.1.3, is that we change the generator $g_1 \xleftarrow{\$} \mathbb{G}_1^*$ into g_1^{ad-bc} for $a, b, c, d \xleftarrow{\$} \mathbb{Z}_p$, which only changes the distribution of the game by a statistical distance of at most $\frac{3}{p}$ (this is obtained by computing the probability that $ad - bc = 0$ when $a, b, c, d \xleftarrow{\$} \mathbb{Z}_p$). Thus,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FE}}(\lambda) = \mathsf{Adv}_0(\mathcal{A}) \leq \mathsf{Adv}_1(\mathcal{A}) + \frac{3}{p}.$$

Note that in Exp_1, the public key, the challenge ciphertext and the functional decryption keys only contain group elements whose exponents are polynomials evaluated on random inputs (as opposed to $g_1^{\mathbf{W}^{-1}}$, for instance). This is going to be helpful for the next step of the proof, which uses the generic bilinear group model.

Next, we make the generic bilinear group model assumption, which intuitively says that no PPT adversary can exploit the structure of the bilinear group to perform better attacks than generic adversaries. That is, we have (with Exp_2 defined in A.4):

$$\max_{\text{PPT } \mathcal{A}} \Big(\mathsf{Adv}_1(\mathcal{A})\Big) = \max_{\text{PPT } \mathcal{A}} \Big(\mathsf{Adv}_2(\mathcal{A})\Big).$$

In this experiment, we denote by $\emptyset$ the empty list, by $\mathsf{append}(L, x)$ the addition of an element x to the list L, and for any $i \in \mathbb{N}$, we denote by $L[i]$ the i'th element of the list L if it exists (lists are indexed from index 1 on), or $\perp$ otherwise.

Thus, it suffices to show that for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_2(\mathcal{A})$ is negligible in λ. The experiment Exp_2 defined in Figure A.4 falls into the general class of simple interactive decisional problems from [BCFG17, Definition 14]. Thus, we can use their master theorem [BCFG17,

```
$\underline{\mathsf{Exp}_2(1^\lambda, \mathcal{A})\text{:}}$
$L_1 = L_2 = L_T := \emptyset$, $Q_{\mathsf{sk}} := \emptyset$, $\boldsymbol{s}, \boldsymbol{t} \overset{\$}{\leftarrow} \mathbb{Z}_p^n$, $a, b, c, d \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\mathsf{append}(L_1, (ad - bc) \cdot \boldsymbol{s})$, $\mathsf{append}(L_2, \boldsymbol{t})$, $\beta \overset{\$}{\leftarrow} \{0, 1\}$
$\Big((\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}), (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})\Big) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{add}}, \mathcal{O}_{\mathsf{pair}}, \mathcal{O}_{\mathsf{sk}}, \mathcal{O}_{\mathsf{eq}}}(1^\lambda, p)$
$\mathcal{O}_{\mathsf{chal}}\Big((\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}), (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})\Big)$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{add}}, \mathcal{O}_{\mathsf{pair}}, \mathcal{O}_{\mathsf{sk}}, \mathcal{O}_{\mathsf{eq}}}(1^\lambda, p)$
If $\beta = \beta'$, and for all $f \in Q_{\mathsf{sk}}$, $f(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}) = f(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})$, output 1. Otherwise, output 0.

$\underline{\mathcal{O}_{\mathsf{add}}(s \in \{1, 2, T\}, i, j \in \mathbb{N})\text{:}}$
$\mathsf{append}(L_s, L_s[i] + L_s[j])$.

$\underline{\mathcal{O}_{\mathsf{pair}}(i, j \in \mathbb{N})\text{:}}$
$\mathsf{append}(L_T, L_1[i] \cdot L_2[j])$.

$\underline{\mathcal{O}_{\mathsf{chal}}\Big((\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}), (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})\Big)\text{:}}$
$\gamma \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\mathsf{append}(L_1, \gamma(ad - bc))$
for all $i \in [n]$, $\boldsymbol{a}_i := \begin{pmatrix} d & -c \\ -b & a \end{pmatrix} \begin{pmatrix} x_i^{(\beta)} \\ \gamma s_i \end{pmatrix}$, $\mathsf{append}(L_1, \boldsymbol{a}_i)$, $\boldsymbol{b}_i := \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} y_i^{(\beta)} \\ -t_i \end{pmatrix}$, $\mathsf{append}(L_2, \boldsymbol{b}_i)$.

$\underline{\mathcal{O}_{\mathsf{sk}}(f \in \mathcal{F}_{n, B_x, B_y, B_f})\text{:}}$
$\mathsf{append}(L_2, f(\boldsymbol{s}, \boldsymbol{t}))$, $Q_{\mathsf{sk}} := Q_{\mathsf{sk}} \cup \{f\}$.

$\underline{\mathcal{O}_{\mathsf{eq}}(s \in \{1, 2, T\}, i, j \in \mathbb{N})\text{:}}$
Output 1 if $L_s[i] = L_s[j]$, 0 otherwise
```

Figure A.4: Experiment Exp_2. Wlog. we assume no query contains indices $i, j \in \mathbb{N}$ that exceed the size of the involved lists.

Theorem 7], which, for our particular case (setting the public key size $N := 2n+2$, the key size $c = 1$, the ciphertext size $c^* := 4n+1$, and degree $d = 6$ in [BCFG17, Theorem 7]) states that:

$$\mathsf{Adv}_2(\mathcal{A}) \leq \frac{12 \cdot (6n+3+Q+Q')^2}{p},$$

where Q' is the number of queries to $\mathcal{O}_{\mathsf{sk}}$, and Q is the number of group operations, that is, the number of calls to oracles $\mathcal{O}_{\mathsf{add}}$ and $\mathcal{O}_{\mathsf{pair}}$, provided the following algebraic condition is satisfied:

$$\begin{aligned} &\{\mathbf{M} \in \mathbb{Z}_p^{(3n+2)\times(3n+Q'+1)} : \mathsf{Eq}_0(\mathbf{M})\} \\ &= \{\mathbf{M} \in \mathbb{Z}_p^{(3n+2)\times(3n+Q'+1)} : \mathsf{Eq}_1(\mathbf{M})\}, \end{aligned}$$

where for all $\mathbf{M}$, $b \in \{0,1\}$,

$$\mathsf{Eq}_b(\mathbf{M}) : \begin{pmatrix} 1 \\ (AD-BC)\boldsymbol{S} \\ (AD-BC)\mathit{\Gamma} \\ D\boldsymbol{x}^{(b)} - \mathit{\Gamma} C\boldsymbol{S} \\ -B\boldsymbol{x}^{(b)} + \mathit{\Gamma} A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 1 \\ \boldsymbol{T} \\ A\boldsymbol{y}^{(b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(b)} - D\boldsymbol{T} \\ (f(\boldsymbol{S},\boldsymbol{T}))_{f\in Q_{\mathsf{sk}}} \end{pmatrix} = 0,$$

where the equality is taken in the ring $\mathbb{Z}_p[\boldsymbol{S},\boldsymbol{T},A,B,C,D,\mathit{\Gamma}]$, and 0 denotes the zero polynomial. Intuitively, this condition captures the security at a symbolic level: it holds for schemes that are not trivially broken. The latter means that computing a linear combination in the exponents of target group elements that can be obtained from pk, the challenge ciphertext, and functional decryption keys, does not break the security of the scheme. We prove this condition is satisfied in 31 below. □

Lemma 31 (Symbolic Security). *For any $(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)}),(\boldsymbol{x}^{(1)},\boldsymbol{y}^{(1)}) \in Z_p^{2n}$, and any set $Q_{\mathsf{sk}} \subseteq \mathcal{F}_{n,B_x,B_y,B_f}$ such that for all $f \in Q_{\mathsf{sk}}$, $f(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)}) = f(\boldsymbol{x}^{(1)},\boldsymbol{y}^{(1)})$, we have:*

$$\begin{aligned} &\{\mathbf{M} \in \mathbb{Z}_p^{(3n+2)\times(3n+Q'+1)} : \mathsf{Eq}_0(\mathbf{M})\} \\ &= \{\mathbf{M} \in \mathbb{Z}_p^{(3n+2)\times(3n+Q'+1)} : \mathsf{Eq}_1(\mathbf{M})\}, \end{aligned}$$

where for all $\mathbf{M}$, $b \in \{0,1\}$,

$$\mathsf{Eq}_b(\mathbf{M}) : \begin{pmatrix} 1 \\ (AD-BC)\boldsymbol{S} \\ (AD-BC)\mathit{\Gamma} \\ D\boldsymbol{x}^{(b)} - \mathit{\Gamma} C\boldsymbol{S} \\ -B\boldsymbol{x}^{(b)} + \mathit{\Gamma} A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 1 \\ \boldsymbol{T} \\ A\boldsymbol{y}^{(b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(b)} - D\boldsymbol{T} \\ (f(\boldsymbol{S},\boldsymbol{T}))_{f\in Q_{\mathsf{sk}}} \end{pmatrix} = 0,$$

where the equality is taken in the ring $\mathbb{Z}_p[\boldsymbol{S},\boldsymbol{T},A,B,C,D,\mathit{\Gamma}]$, and 0 denotes the zero polynomial.

Proof. Let $b \in \{0,1\}$, and $\mathbf{M} \in \mathbb{Z}_p^{(3n+2)\times(3n+Q'+1)}$ that satisfies $\mathsf{Eq}_b(\mathbf{M})$. We prove it also satisfies $\mathsf{Eq}_{1-b}(\mathbf{M})$. To do so, we use the following rules:

Rule 1 : for all $P,Q,R \in \mathbb{Z}_p[\boldsymbol{S},\boldsymbol{T},A,B,C,D,\mathit{\Gamma}]$, with $\deg(P) \geq 1$, if $P \cdot Q + R = 0$ and R is not a multiple of P, then $Q = 0$ and $R = 0$.

Rule 2 : for all $P \in \mathbb{Z}_p[\boldsymbol{S},\boldsymbol{T},A,B,C,D,\mathit{\Gamma}]$, any variable X among the set $\{\boldsymbol{S},\boldsymbol{T},A,B,C,D,\mathit{\Gamma}\}$, and any $x \in \mathbb{Z}_p$, $P = 0$ implies $P(X := x) = 0$, where $P(X := x)$ denotes the polynomial P evaluated on $X = x$.

Evaluating $\mathsf{Eq}_b(\mathbf{M})$ on $B = D = 0$ (using **Rule 2**), then using **Rule 1** on $P = C\mathit{\Gamma} S_i T_j$ for all $i, j \in [n]$, we obtain that:

$$\mathbf{M}_{n+2+i} \begin{pmatrix} 0 \\ \boldsymbol{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\boldsymbol{S}, \boldsymbol{T}))_{f \in Q_{\mathsf{sk}}} \end{pmatrix} = 0,$$

where $\mathbf{M}_{n+2+i}$ denotes the $n+2+i$'th row of $\mathbf{M}$.

Similarly, using **Rule 1** on $P = \mathit{\Gamma} A S_i T_j$ for all $i, j \in [n]$, we obtain that:

$$\mathbf{M}_{2n+2+i} \begin{pmatrix} 0 \\ \boldsymbol{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\boldsymbol{S}, \boldsymbol{T}))_{f \in Q_{\mathsf{sk}}} \end{pmatrix} = 0.$$

Thus, we have:

$$\forall \beta \in \{0,1\} : \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\boldsymbol{x}^{(\beta)} - \mathit{\Gamma} C \boldsymbol{S} \\ -B\boldsymbol{x}^{(\beta)} + \mathit{\Gamma} A \boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 0 \\ \boldsymbol{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\boldsymbol{S}, \boldsymbol{T}))_{f \in Q_{\mathsf{sk}}} \end{pmatrix} = 0. \tag{A.1}$$

Using **Rule 1** on $P = (AD - BC)S_i B T_j$ for all $i, j \in [n]$ in the equation $\mathsf{Eq}_b(\mathbf{M})$, we get that the coefficient $M_{i+1,n+1+j} = 0$ for all $i, j \in [n]$. Similarly, using **Rule 1** on $P = (AD - BC)S_i D T_j$ for all $i, j \in [n]$, we get $M_{i+1,2n+1+j} = 0$ for all $i, j \in [n]$. Then, using **Rule 1** on $P = (AD - BC)\mathit{\Gamma} B T_j$ for all $j \in [n]$, we get $M_{n+2,n+1+j} = 0$ for all $j \in [n]$. Finally, using **Rule 1** on $P = (AD - BC)\mathit{\Gamma} D T_j$ for all $j \in [n]$, we get $M_{n+2,2n+1+j} = 0$ for all $j \in [n]$. Overall, we obtain:

$$\forall \beta \in \{0,1\} : \begin{pmatrix} 0 \\ (AD - BC)\boldsymbol{S} \\ (AD - BC)\mathit{\Gamma} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\boldsymbol{y}^{(\beta)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(\beta)} - D\boldsymbol{T} \\ \mathbf{0} \end{pmatrix} = 0. \tag{A.2}$$

We write:

$$\begin{aligned} &\begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\boldsymbol{x}^{(b)} - \mathit{\Gamma} C \boldsymbol{S} \\ -B\boldsymbol{x}^{(b)} + \mathit{\Gamma} A \boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\boldsymbol{y}^{(b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(b)} - D\boldsymbol{T} \\ \mathbf{0} \end{pmatrix} \\ &= \sum_{i,j \in [n]} \begin{pmatrix} D x_i^{(b)} - \mathit{\Gamma} C S_i \\ -B x_i^{(b)} + \mathit{\Gamma} A S_i \end{pmatrix}^{\top} \\ &\times \left(m_{i,j}^{(1)} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + m_{i,j}^{(2)} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + m_{i,j}^{(3)} \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + m_{i,j}^{(4)} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \right) \\ &\times \begin{pmatrix} A y_j^{(b)} - B T_j \\ C y_j^{(b)} - D T_j \end{pmatrix} \end{aligned}$$

Evaluating the equation $\mathsf{Eq}_b(\mathbf{M})$ on $C = D = 0$ (by **Rule 2**), then using **Rule 1** on $P = \Gamma ABS_iT_j$ for all $i, j \in [n]$, we obtain $m^{(3)}_{i,j} = 0$ for all $i, j \in [n]$. Evaluating the equation $\mathsf{Eq}_b(\mathbf{M})$ on $A = B = 0$ (by **Rule 2**), then using **Rule 1** on $P = \Gamma CDS_iT_j$ for all $i, j \in [n]$, we obtain $m^{(4)}_{i,j} = 0$ for all $i, j \in [n]$. Evaluating the equation $\mathsf{Eq}_b(\mathbf{M})$ on $A = B = C = D = 1$ (using **Rule 2**), then using **Rule 1** on $P = \Gamma S_iT_j$ for all $i, j \in [n]$, using the fact that $m^{(3)}_{i,j} = m^{(4)}_{i,j} = 0$ and (A.1), we obtain $m^{(2)}_{i,j} = 0$ for all $i, j \in [n]$. Using **Rule 1** on $P = \Gamma(AD - BC)S_iT_j$ for all $i, j \in [n]$ in the equation $\mathsf{Eq}_b(\mathbf{M})$, we obtain that for all $i, j \in [n]$,

$$m^{(1)}_{i,j} = \mathbf{M}_{n+2} \begin{pmatrix} 0 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ (f_{i,j})_{f \in Q_{\mathsf{sk}}} \end{pmatrix},$$

where $\mathbf{M}_{n+2}$ is the $n+2$'th row of $\mathbf{M}$.

Putting everything together, can write

$$\begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\boldsymbol{x}^{(b)} - \Gamma C\boldsymbol{S} \\ -B\boldsymbol{x}^{(b)} + \Gamma A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\boldsymbol{y}^{(b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(b)} - D\boldsymbol{T} \\ \mathbf{0} \end{pmatrix}$$

as

$$\begin{aligned} &(AD - BC)\mathbf{M}_{n+2} \begin{pmatrix} 0 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \left(f(\boldsymbol{x}^{(b)}, \boldsymbol{y}^{(b)}) - \Gamma f(\boldsymbol{s}, \boldsymbol{t})\right)_{f \in Q_{\mathsf{sk}}} \end{pmatrix} \\ &= (AD - BC)\mathbf{M}_{n+2} \begin{pmatrix} 0 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \left(f(\boldsymbol{x}^{(1-b)}, \boldsymbol{y}^{(1-b)}) - \Gamma f(\boldsymbol{s}, \boldsymbol{t})\right)_{f \in Q_{\mathsf{sk}}} \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\boldsymbol{x}^{(1-b)} - \Gamma C\boldsymbol{S} \\ -B\boldsymbol{x}^{(1-b)} + \Gamma A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\boldsymbol{y}^{(b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(b)} - D\boldsymbol{T} \\ \mathbf{0} \end{pmatrix} \end{aligned} \tag{A.3}$$

where we use the fact that for all $f \in Q_{\mathsf{sk}}$, we have the equality $f(\boldsymbol{x}^{(b)}, \boldsymbol{y}^{(b)}) = f(\boldsymbol{x}^{(1-b)}, \boldsymbol{y}^{(1-b)})$.

Evaluating equation $\mathsf{Eq}_b(\mathbf{M})$ on $A = B = D = 0$ (by **Rule 2**), then using **Rule 1** on ΓS_iC for all $i \in [n]$, and using (A.1) and (A.3), we obtain that the coefficient $M_{n+2+i,1} = 0$ for all $i \in [n]$. Evaluating $\mathsf{Eq}_b(\mathbf{M})$ on $B = C = D = 0$ (by **Rule 2**), then using **Rule 1** on ΓS_iA for all $i \in [n]$, and using (A.1) and (A.3), we obtain that the coefficient $M_{2n+2+i,1} = 0$ for all $i \in [n]$. Thus, we have:

$$\forall \beta \in \{0,1\} : \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\boldsymbol{x}^{(\beta)} - \Gamma C\boldsymbol{S} \\ -B\boldsymbol{x}^{(\beta)} + \Gamma A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 1 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} = 0. \tag{A.4}$$

Evaluating equation $\mathsf{Eq}_b(\mathbf{M})$ on $A = C = D = 0$ (by **Rule 2**), then using **Rule 1** on BT_j for all $i \in [n]$, and using (A.3), we obtain that the coefficient $M_{1,n+1+j} = 0$ for all $j \in [n]$. Evaluating $\mathsf{Eq}_b(\mathbf{M})$ on $A = B = C = 0$ (by **Rule 2**), then using **Rule 1** on DT_j for all $j \in [n]$, and using (A.3), we obtain that the coefficient $M_{1,2n+1+j} = 0$ for all $j \in [n]$. Thus, we have:

$$\forall \beta \in \{0,1\} : \begin{pmatrix} 1 \\ \mathbf{0} \\ 0 \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\boldsymbol{y}^{(\beta)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(\beta)} - D\boldsymbol{T} \\ \mathbf{0} \end{pmatrix} = 0. \tag{A.5}$$

Overall, we have:

$$\mathsf{Eq}_b(\mathbf{M}) : \begin{pmatrix} 1 \\ (AD - BC)\boldsymbol{S} \\ (AD - BC)\varGamma \\ D\boldsymbol{x}^{(b)} - \varGamma C\boldsymbol{S} \\ -B\boldsymbol{x}^{(b)} + \varGamma A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 1 \\ \boldsymbol{T} \\ A\boldsymbol{y}^{(b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(b)} - D\boldsymbol{T} \\ (f(\boldsymbol{S}, \boldsymbol{T}))_{f \in Q_{\mathsf{sk}}} \end{pmatrix} = 0$$

which implies the following relation, under (A.1),(A.2),(A.4),(A.5)

$$\begin{pmatrix} 1 \\ (AD - BC)\boldsymbol{S} \\ (AD - BC)\varGamma \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 1 \\ \boldsymbol{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\boldsymbol{S}, \boldsymbol{T}))_{f \in Q_{\mathsf{sk}}} \end{pmatrix}$$

$$+ \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\boldsymbol{x}^{(b)} - \varGamma C\boldsymbol{S} \\ -B\boldsymbol{x}^{(b)} + \varGamma A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\boldsymbol{y}^{(b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(b)} - D\boldsymbol{T} \\ \mathbf{0} \end{pmatrix} = 0$$

and then, under (A.3)

$$\begin{pmatrix} 1 \\ (AD - BC)\boldsymbol{S} \\ (AD - BC)\varGamma \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 1 \\ \boldsymbol{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\boldsymbol{S}, \boldsymbol{T}))_{f \in Q_{\mathsf{sk}}} \end{pmatrix}$$

$$+ \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\boldsymbol{x}^{(1-b)} - \varGamma C\boldsymbol{S} \\ -B\boldsymbol{x}^{(1-b)} + \varGamma A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\boldsymbol{y}^{(1-b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(1-b)} - D\boldsymbol{T} \\ \mathbf{0} \end{pmatrix} = 0.$$

Under (A.1),(A.2),(A.4),(A.5), this implies

$$\mathsf{Eq}_{1-b}(\mathbf{M}) : \begin{pmatrix} 1 \\ (AD - BC)\boldsymbol{S} \\ (AD - BC)\varGamma \\ D\boldsymbol{x}^{(1-b)} - \varGamma C\boldsymbol{S} \\ -B\boldsymbol{x}^{(1-b)} + \varGamma A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 1 \\ \boldsymbol{T} \\ A\boldsymbol{y}^{(1-b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(1-b)} - D\boldsymbol{T} \\ (f(\boldsymbol{S}, \boldsymbol{T}))_{f \in Q_{\mathsf{sk}}} \end{pmatrix} = 0$$

□

A.2 AriaNN: Low-Interaction Privacy-Preserving Deep Learning via Function Secret Sharing

A.2.1 Encoding Precision

We have studied the impact of lowering the encoding space of the input to our function secret sharing protocol from $\mathbb{Z}_{2^{32}}$ to $\mathbb{Z}_{2^k}$ with $k < 32$. Finding the lowest k guaranteeing good performance is an interesting challenge as the function keys size is directly proportional to it. This has to be done together with reducing fixed precision from 3 decimals down to 1 decimal to ensure private values are not too big, which would result in higher failure rate in our private comparison protocol. We have reported in Table A.2 our findings on Network-1, which is pre-trained and then evaluated in a private fashion.

Decimals	$\mathbb{Z}_{2^{12}}$	$\mathbb{Z}_{2^{16}}$	$\mathbb{Z}_{2^{20}}$	$\mathbb{Z}_{2^{24}}$	$\mathbb{Z}_{2^{28}}$	$\mathbb{Z}_{2^{32}}$
1	-	-	-	-	-	9.5
2	69.4	96.0	97.9	98.1	98.0	98.1
3	10.4	76.2	96.9	98.1	98.2	98.1
4	9.7	14.3	83.5	97.4	98.1	98.2

Table A.2: Accuracy (in %) of Network-1 given different precision and encoding spaces

What we observe is that 3 decimals of precision is the most appropriate setting to have an optimal precision while allowing to slightly reduce the encoding space down to $\mathbb{Z}_{2^{24}}$ or $\mathbb{Z}_{2^{28}}$. Because this is not a massive gain and in order to keep the failure rate in comparison very low, we have kept $\mathbb{Z}_{2^{32}}$ for all our experiments.

A.2.2 Implementation Details

Pseudo-Random Generator

The PRG is implemented using a Matyas-Meyer-Oseas one-way compression function as in [WYG+17], with an AES block cipher. We concatenate several fixed key block ciphers to achieve the desired output length: $G(x) = E_{k_1}(x) \oplus x \mid\mid E_{k_2}(x) \oplus x \mid\mid \dots$. Those keys are fixed and hard-coded. We set $\lambda = 127$ to be able to use only 2 blocks for equality and 4 blocks for comparison. Note that for comparison we would theoretically only need 3 blocks, although our current implementation uses 4.

Unrolling Convolutions

Figure A.5 illustrates how to transform a convolution operation into a single matrix multiplication.

MaxPool and Optimisation

Figure A.6 illustrates how MaxPool uses ideas from matrix unrolling and argmax computation. Notations present in the figure are consistent with the explanation of argmax using pairwise comparison in Section 5.2.4.2. The $m \times m$ matrix is first unrolled to a roughly $(m/s)^2 \times k^2$ matrix. It is then expanded on k^2 layers, each of which is shifted by a step of 1. Next, $(m/s)^2 k^2 (k^2 - 1)$ pairwise comparisons are then applied simultaneously between the first layer and the other ones, and for each x_i we sum the result of its $k - 1$ comparison and check if it equals $k - 1$. We multiply this boolean by x_i and sum up along a line (like x_0 to x_3 in the figure). Last, we restructure the matrix back to its initial structure.

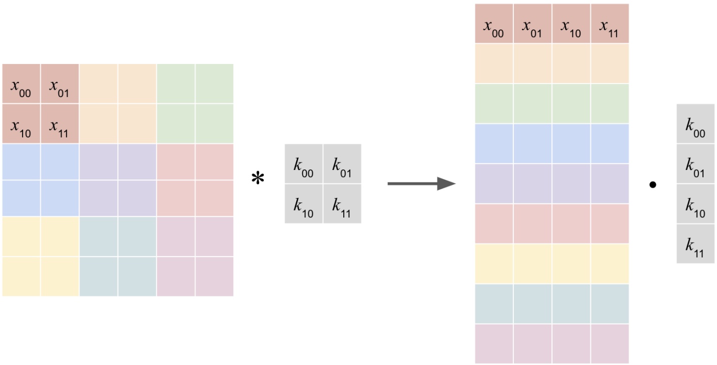

Figure A.5: Illustration of unrolling a convolution with kernel size $k = 2$ and stride $s = 2$.

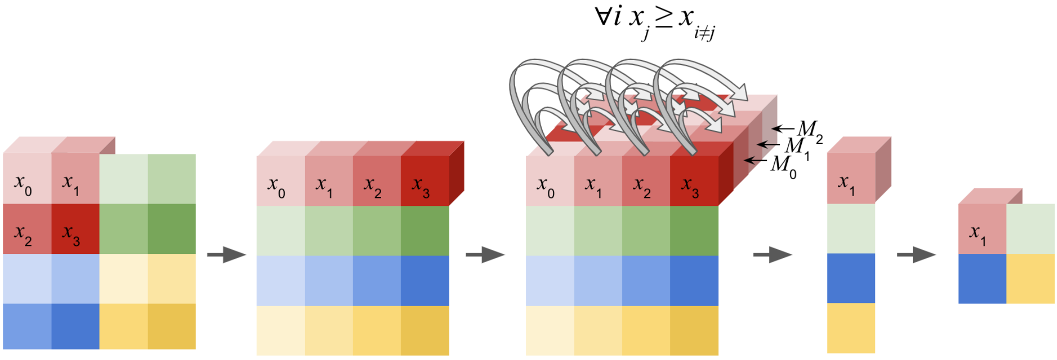

Figure A.6: Illustration of MaxPool with kernel size $k = 2$ and stride $s = 2$.

In addition, when the kernel size k is 2, rows are only of length 4 and it can be more efficient to use a binary tree approach instead, i.e. compute the maximum of columns 0 and 1, 2 and 3 and the max of the result: it requires $2 \log_2(k^2) = 4$ rounds of communication but only approximately $(k^2 - 1)(m/s)^2$ private comparisons, compared to a fixed 3 rounds and approximately $k^4(m/s)^2$. We found in practice that this 4× speed-up factor in computation is worth an additional communication round.

Interestingly, average pooling can be computed locally on the shares without interaction because it only includes mean operations, but we did not replace MaxPool operations with average pooling to avoid distorting existing neural networks architecture.

Breaking Ties in Argmax

Algorithm 14 provides a probabilistic method to break ties in the argmax output, which can be used on secret shared input as well.

BatchNorm Approximation

The BatchNorm layer is the only one in our implementation which requires a polynomial approximation during training. We have therefore experimented how this approximation can alter the behaviour of a deep network such as ResNet18 on a simple dataset like the hymenoptera dataset[1]. We follow the PyTorch transfer learning tutorial[2] and use a pretrained version of ResNet18. We retrain all the layers for 25 epochs, but we replace the BatchNorm layers with our approximated version which behaves as such: as a general rule the inverse for the variance

[1] https://download.pytorch.org/tutorial/hymenoptera_data.zip
[2] https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

```
**Input:** $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m)$, with $\forall i, \delta_i \in \{0, 1\}$
**Output:** $\delta_j$ with $j \xleftarrow{\$} \{i, \delta_i = 1\}$
$\boldsymbol{x} = \mathsf{CumSum}(\boldsymbol{\delta})$
$r \xleftarrow{\$} [0, \boldsymbol{x}[-1][$
$\boldsymbol{c} = \boldsymbol{x} > r$
Compute $\boldsymbol{c}_{\gg}$ by shifting $\boldsymbol{c}$ by one number on the right and padding with 0
**return** $\boldsymbol{c} - \boldsymbol{c}_{\gg}$
```

Algorithm 14: BreakTies algorithm to guarantee one-hot output

is computed using 3 iterations of the Newton methods and we use the result of the previous batch as an initial approximation. For the first batch of the epoch where no approximation is available, we use instead 50 iterations. For the first BatchNorm layer, since the variance can change significantly because of the input diversity, we systematically use more iterations, up to 60. For the last layer, as the variance is very small, its inverse can have a large amplitude. Therefore we do not use the result of the previous batch and perform 10 iterations instead. We report in Figure A.7 the evolution of the error as the model train on more batches. As one can see, the error dramatically shrinks after 10 batches, which shows how beneficial it is to use previous batch computations. It almost stays below 5%, except for the error of the first layer, for which we would need even more iterations to have significant improvements. The average error per layer is reported in Figure A.8, and is around 2%.

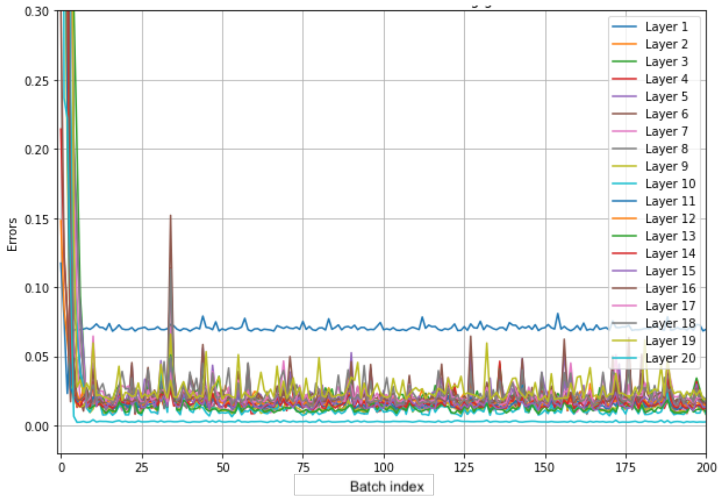

Figure A.7: BatchNorm relative error as training goes on

This is of course an experimental result, but it shows that the BatchNorm approximation can be tailored for specific models to allow for efficient training with little round overhead. Table 5.2 shows that we can indeed achieve a high accuracy, especially using the initial guess of the previous batch, although it remains below the accuracy of a model trained with an exact BatchNorm.

A.2.3 Extended Results about Private Inference

We provide additional results about our inference experiment in Table A.3.

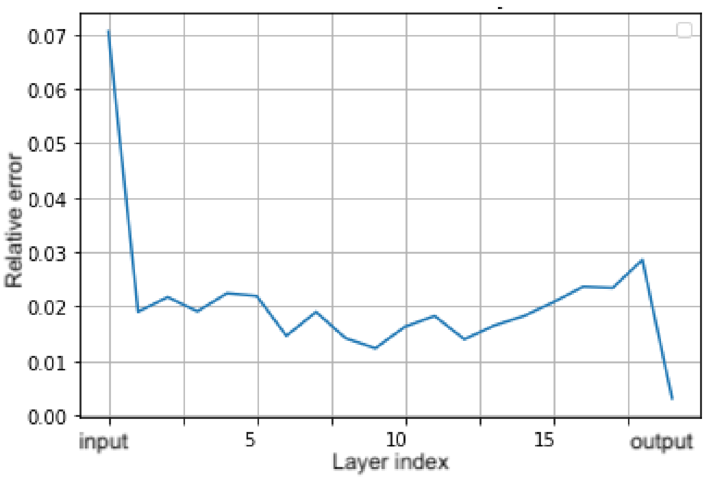

Figure A.8: Relative error of the BatchNorm layers across the model

Table A.3: Comparison of the inference time between secure frameworks on several popular neural network architectures. For each network we report in order the batch size used and the size of the preprocessing material in MB for the CPU setting.

		Network-1			Network-2			LeNet		
		Batch Size		Preprocessing	Batch Size		Preprocessing	Batch Size		Preprocessing
Framework	Dataset	CPU	GPU	Comm. (MB)	CPU	GPU	Comm. (MB)	CPU	GPU	Comm. (MB)
AriaNN	MNIST	128	128	0.36	128	128	9.98	128	128	14.7

		AlexNet			VGG16			ResNet18		
		Batch Size		Preprocessing	Batch Size		Preprocessing	Batch Size		Preprocessing
Framework	Dataset	CPU	GPU	Comm. (MB)	CPU	GPU	Comm. (MB)	CPU	GPU	Comm. (MB)
AriaNN	CIFAR-10	128	128	24.6	64	14	277	-	-	-
AriaNN	64×64 ImageNet	128	64	88.4	16	3	1124	-	-	-
AriaNN	224×224 Hymenoptera	-	-	-	-	-	-	8	1	3254

A.2.4 Datasets and Networks Architecture

Datasets

This section is taken almost verbatim from [WTB+21].

We select 4 datasets popularly used for training image classification models: MNIST [LC10], CIFAR-10 [KNH14], 64×64 Tiny Imagenet [WZX17] and Hymenoptera, a subset of the Imagenet dataset [RDS+15] composed 224×224 pixel images.

MNIST MNIST [LC10] is a collection of handwritten digits dataset. It consists of 60,000 images in the training set and 10,000 in the test set. Each image is a 28×28 pixel image of a handwritten digit along wit a label between 0 and 9. We evaluate Network-1, Network-2, and the LeNet network on this dataset.

CIFAR-10 CIFAR-10 [KNH14] consists of 50,000 images in the training set and 10,000 in the test set. It is composed of 10 different classes (such as airplanes, dogs, horses, etc.) and there are 6,000 images of each class with each image consisting of a colored 32×32 image. We perform private training of AlexNet and inference of VGG16 on this dataset.

Tiny ImageNet Tiny ImageNet [WZX17] consists of two datasets of 100,000 training samples and 10,000 test samples with 200 different classes. The first dataset is composed of colored 64×64 images and we use it with AlexNet and VGG16. The second is composed of colored 224×224 images and is used with ResNet18.

Hymenoptera Hymenoptera is a dataset extracted from the ImageNet database. It is composed of 245 training and 153 test colored 224×224 images, and was first proposed as a transfer learning task.

Model Description

We have selected 6 models for our experimentations. Description on the first 5 models is taken verbatim from [WTB+21].

Network-1 A 3-layered fully-connected network with ReLU used in SecureML [MZ17].

Network-2 A 4-layered network selected in MiniONN [LJLA17] with 2 convolutional and 2 fully-connected layers, which uses MaxPool in addition to ReLU activation.

LeNet This network, first proposed by LeCun et al. [LBBH98], was used in automated detection of zip codes and digit recognition. The network contains 2 convolutional layers and 2 fully connected layers.

AlexNet AlexNet is the famous winner of the 2012 ImageNet ILSVRC-2012 competition [KSH12]. It has 5 convolutional layers and 3 fully connected layers and it can use batch normalization layers for stability and efficient training.

VGG16 VGG16 is the runner-up of the ILSVRC-2014 competition [SZ14]. VGG16 has 16 layers and has about 138M parameters.

ResNet18 ResNet18 [HZRS16] is the runner-up of the ILSVRC-2015 competition. It is a convolutional neural network that is 18 layers deep, and has 11.7M parameters. It uses batch normalisation and we are the first private deep learning framework to evaluate this network.

Models Architecture

Unless otherwise specified, our models follow their standard architecture as provided by the `torchvision` library (version 0.5), except for smaller models such as Network-1, Network-2 and LeNet which are respectively detailed in [MZ17], [LJLA17] and [LBBH98].

For the CIFAR-10 version of AlexNet, we follow the architecture of [WTB+21] which includes BatchNorm layers and is available on the FALCON GitHub. For the 64×64 Tiny Imagenet version of AlexNet however, we used the standard architecture from PyTorch since it allows us to have a pretrained network and the version of FALCON seemed non-standard.

We have adapted the classifier parts of AlexNet, VGG16 and ResNet18 to the different datasets we use.

- For AlexNet on Tiny Imagenet, we use 3 fully connected layers with respectively 1024, 1024 and 200 neurons, and ReLU activations between them.
- For VGG16 we use 3 fully connected layers with respectively 4096, 4096 and 10 or 200 neurons for the last layer depending on the dataset used. We also use ReLU activations between them.
- For ResNet18, we use a single fully connected layer to map the 512 output logits to the appropriate number of classes.

Note also that we permute ReLU and Maxpool where applicable like in [WTB+21], as this is strictly equivalent in terms of output for the network and reduces the number of comparisons. More generally, we do not proceed to any alteration of the network behaviour except with the approximation on BatchNorm. This improves the usability of our framework as it allows us to use pre-trained neural networks from standard deep learning libraries like PyTorch and to encrypt them with a single line of code.

Appendix

B Differential Privacy Guarantees for Stochastic Gradient Langevin Dynamics

B.1 Smoothness and Convexity of Multinomial Logistic Regression

For clarity, we replace the generic parameter θ with the single matrix $\mathbf{W}$ that it represents for multinomial logistic regression.

The loss with regularization writes $\ell(\mathbf{W}, \mathbf{x}) = \log\left(\boldsymbol{\sigma}(\mathbf{W}\mathbf{x})\right)_y + \lambda \left\|\mathbf{W}\right\|_2^2$ where C is the number of classes, $y \in [1, C]$ is the label of sample $\mathbf{x}$, $\mathbf{W} \in \mathbb{R}^{C \times p}$ and $\boldsymbol{\sigma} : \mathbb{R}^C \mapsto \mathbb{R}^C$ is the sigmoid function (not to be confused with the noise variance σ):

$$\left(\boldsymbol{\sigma}(\mathbf{z})\right)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}},\ \forall i = 1, .., C\ , \mathbf{z} \in \mathbb{R}^C.$$

Lemma 32 (Convexity and smoothness constants for regularized multinomial logistic regression)**.** *Let $\ell(\mathbf{W}, \mathbf{x})$ be defined as above. Then ℓ is λ-strongly convex and β-smooth, with*

$$\beta = \frac{1}{2}\lambda_{\mathsf{max}}\Big(\frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top\Big) + \lambda$$

where λ_{max} refers to the maximum eigenvalue.

Proof. By plugging in the definition of the sigmoid, the loss ℓ also writes:

$$\ell(\mathbf{W}, \mathbf{x}) = \log \frac{\exp\left((\mathbf{W}\mathbf{x})_y\right)}{\sum_{i=1}^{C} \exp\left((\mathbf{W}\mathbf{x})_i\right)} + \lambda \left\|\mathbf{W}\right\|_2^2,$$

where $\mathbf{W} \in \mathbb{R}^{C \times p}$ also writes $[\mathbf{w}_1^\top, \ldots, \mathbf{w}_C^\top]^\top$.
Let's define for i = 1, ..., C

$$\begin{aligned} p_i = P(y_i = 1 | \mathbf{x}, \mathbf{W}) &= \frac{\exp\left((\mathbf{W}\mathbf{x})_i\right)}{\sum_{c=1}^{C} \exp\left((\mathbf{W}\mathbf{x})_c\right)} \\ &= \frac{\exp(\mathbf{w}_i^\top \mathbf{x})}{\sum_{c=1}^{C} \exp(\mathbf{w}_c^\top \mathbf{x})} \end{aligned}$$

We can rewrite the loss ℓ as such

$$\ell(\mathbf{W}, \mathbf{x}) = \log \prod_{i=1}^{C} p_i^{y_i} + \lambda \left\|\mathbf{W}\right\|_2^2$$

where the label y is now one-hot encoded: $y = (y_0, \ldots, y_C)$.

Following the work of [Böh92], we have:

$$\nabla^2 \ell(\mathbf{W}, \mathbf{x}) = (\mathbf{D}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top) \otimes \mathbf{x}\mathbf{x}^\top + \lambda \mathbf{I}_C$$

where $\mathbf{D}(\mathbf{p}) = \mathbf{I}_C \mathbf{p}$.
We derive

$$\nabla^2 \mathcal{L}_{\mathcal{D}}(\mathbf{W}) = \frac{1}{n} \sum_{j=1}^{n} (\mathbf{D}(\mathbf{p}_j) - \mathbf{p}_j \mathbf{p}_j^\top) \otimes \mathbf{x}_j \mathbf{x}_j^\top + \lambda \mathbf{I}_C$$

where $\mathbf{p}_j$ corresponds to $\mathbf{p}$ conditioned with $\mathbf{x}_j$.
As shown in [KCFH05], $\nabla^2 \mathcal{L}_{\mathcal{D}}(\mathbf{W})$ satisfies the following

$$\lambda \mathbf{I}_C \preceq \nabla^2 \mathcal{L}_{\mathcal{D}}(\mathbf{W}) \preceq \frac{1}{2}(\mathbf{I}_C - \frac{1}{C}\mathbf{1}_C \mathbf{1}_C^\top) \otimes \frac{1}{n}\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_C$$

where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times d}$. In particular, we deduce:

$$\begin{aligned}
\beta &= \lambda_{\mathsf{max}}(\nabla^2 \mathcal{L}_{\mathcal{D}}(\mathbf{W})) \\
&\leq \lambda_{\mathsf{max}} \left(\frac{1}{2}(\mathbf{I}_C - \frac{1}{C}\mathbf{1}_C \mathbf{1}_C^\top) \otimes \frac{1}{n}\mathbf{X}\mathbf{X}^\top + \right) + \lambda \\
&= \max_{\lambda_{\mathsf{eig}}, \lambda'_{\mathsf{eig}}} \lambda_{\mathsf{eig}} \left(\frac{1}{2}(\mathbf{I}_C - \frac{1}{C}\mathbf{1}_C \mathbf{1}_C^\top) \right) \lambda'_{\mathsf{eig}} \left(\frac{1}{n}\mathbf{X}\mathbf{X}^\top \right) + \lambda \\
&\leq \frac{1}{2n} \lambda_{\mathsf{max}} \left(\mathbf{X}\mathbf{X}^\top \right) + \lambda
\end{aligned}$$

□

B.2 Datasets and Models

Datasets

We have selected for our experiments two datasets commonly used for training image classification models: CIFAR-10 and CIFAR-100 [KNH14], and also a dataset with healthcare data which can more closely mimic a scenario where we care about training a model on sensitive data: Pneumonia [KGC+18].

CIFAR CIFAR-10 and CIFAR-100 [KNH14] both consist of 50,000 images in the training set and 10,000 in the test set. They are respectively composed of 10 and 100 different balanced classes (such as airplanes, dogs, horses, etc.) and each image consists of a colored 32×32 image. The datasets are disjoints, which allows us to pretrain our models AlexNet and Resnet18 on CIFAR-100 and consider it *public* pre-training before performing logistic regression on CIFAR10.

Pneumonia Pneumonia is a dataset of chest X-ray images of pediatric pneumonia that was published by [KGC+18]. It is composed of 5163 training and 624 test non-colored images of varying sizes. Images are divided in 3 classes: bacterial (26%), normal (48%) and viral (26%). It provides an interesting use case as it is a relatively small dataset and is composed of healthcare data.

Models

We have selected 2 models for our experimentations.

AlexNet AlexNet is the famous winner of the 2012 ImageNet ILSVRC-2012 competition [KSH12]. It has 5 convolutional layers and 3 fully connected layers and it can use batch normalization layers for stability and efficient training.

ResNet18 ResNet18 [HZRS16] is the runner-up of the ILSVRC-2015 competition. It is a convolutional neural network that is 18 layers deep, and has 11.7M parameters. It uses batch normalisation layers, but as only the last layer is retrained with differential privacy, we need not replace those layers with group normalisation.

Bibliography

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March / April 2015. *Cited on page 48.*

[ABGW17] Miguel Ambrona, Gilles Barthe, Romain Gay, and Hoeteck Wee. Attribute-based encryption in the generic group model: Automated proofs and new constructions. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 647–664. ACM Press, October / November 2017. *Cited on page 105.*

[Abo18] John M Abowd. The us census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2867–2867, 2018. *Cited on page 101.*

[ACG$^+$16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016. *Cited on pages 5, 8, 21, 26, 58, 80, 87, and 95.*

[AFL$^+$16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 805–817, 2016. *Cited on page 76.*

[AGM$^+$13] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013. *Cited on page 53.*

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010. *Cited on page 13.*

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016. *Cited on page 48.*

[AMD20] Shakkeel Ahmed, Ravi S. Mula, and Soma S. Dhavala. A framework for democratizing ai. *ArXiv*, abs/2001.00818, 2020. *Cited on page 33.*

[AMIA21] Muhammad Asad, Ahmed Moustafa, Takayuki Ito, and Muhammad Aslam. Evaluating the communication efficiency in federated learning algorithms. In *2021*

IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pages 552–557. IEEE, 2021. *Cited on page 33.*

[ARC19] Mohammad Al-Rubaie and J. Morris Chang. Privacy-preserving machine learning: Threats and solutions. *IEEE Security & Privacy*, 17(2):49–58, 2019. *Cited on page 54.*

[ASSKG19] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. Quotient: two-party secure neural network training and prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1231–1247, 2019. *Cited on page 55.*

[BBG18] Borja Balle, Gilles Barthe, and Marco Gaboardi. Privacy amplification by subsampling: Tight analyses via couplings and divergences. *Advances in Neural Information Processing Systems*, 31, 2018. *Cited on page 80.*

[BBGG19] Borja Balle, Gilles Barthe, Marco Gaboardi, and Joseph Geumlek. Privacy amplification by mixing and diffusion mechanisms. *Advances in neural information processing systems*, 32, 2019. *Cited on page 80.*

[BCCW19] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 45–56, 2019. *Cited on page 55.*

[BCD$^+$20] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. Mp2ml: A mixed-protocol machine learning framework for private inference. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–10, 2020. *Cited on page 33.*

[BCFG17] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 67–98. Springer, Heidelberg, August 2017. *Cited on pages 48, 49, 51, 105, 106, 108, and 110.*

[BCG$^+$21] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 871–900. Springer, 2021. *Cited on pages 55, 56, and 77.*

[BCL$^+$18] Ahmad Al Badawi, Jin Chao, Jie Lin, Chan Fook Mun, Jun Jie Sim, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekhar. The alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus. Cryptology ePrint Archive, Report 2018/1056, 2018. *Cited on page 40.*

[BCPS20] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. Flash: fast and robust framework for privacy-preserving machine learning. *Proceedings on Privacy Enhancing Technologies*, 2020(2):459–480, 2020. *Cited on page 55.*

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991. *Cited on pages 19 and 68.*

[BEG^{+}19] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388, 2019. *Cited on page 54.*

[BELO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *Proceedings of the Conference on Computer and Communications Security*, pages 578–590, 2016. *Cited on page 57.*

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001. *Cited on page 16.*

[BF03] Dan Boneh and Matthew K. Franklin. Identity based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003. *Cited on page 50.*

[BFF^{+}14] Gilles Barthe, Edvard Fagerholm, Dario Fiore, John C. Mitchell, Andre Scedrov, and Benedikt Schmidt. Automated analysis of cryptographic assumptions in generic group models. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 95–112. Springer, Heidelberg, August 2014. *Cited on page 105.*

[BFL^{+}11] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-preserving ecg classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security*, 6(2):452–468, 2011. *Cited on page 48.*

[BGI^{+}01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001. *Cited on page 48.*

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 337–367. Springer, 2015. *Cited on pages 6, 8, 54, 55, and 56.*

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016. *Cited on pages 8, 55, 56, 57, 58, 59, 60, 63, and 68.*

[BGI19] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2019. *Cited on pages 19, 56, 57, 62, and 68.*

[BGTT18] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. Personalized and private peer-to-peer machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 473–481. PMLR, 2018. *Cited on page 103.*

[BHN18] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning*. fairmlbook.org, 2018. *Cited on page 41.*

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the Conference on Computer and Communications Security*, pages 1175–1191, 2017. *Cited on page 72.*

[BLCW19] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 3–13, 2019. *Cited on page 55.*

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008. *Cited on pages 55 and 57.*

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, pages 483–512, 2018. *Cited on pages 40 and 48.*

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006. *Cited on pages 16 and 17.*

[Böh92] Dankmar Böhning. Multinomial logistic regression algorithm. *Annals of the institute of Statistical Mathematics*, 44(1):197–200, 1992. *Cited on page 120.*

[BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015. *Cited on page 48.*

[BST14] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 464–473. IEEE, 2014. *Cited on page 80.*

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011. *Cited on pages 6, 39, 48, and 49.*

[BT97] Dimitri Bertsekas and John Tsitsiklis. *Parallel and distributed computation: numerical methods*. Athena Scientific, 1997. *Cited on page 3.*

[BTM+20] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2020. *Cited on page 33.*

[BV11] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Oct 2011. *Cited on page 50.*

[BVH+20] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020. *Cited on pages 37 and 58.*

[CCN+22] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *Symposium on Security and Privacy (SP)*. IEEE, 2022. *Cited on page 38.*

[CD18] Rachel Cummings and Deven Desai. The role of differential privacy in gdpr compliance. In *FAT'18: Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2018. *Cited on pages 20 and 101.*

[CFLS18] Sergiu Carpov, Caroline Fontaine, Damien Ligier, and Renaud Sirdey. Illuminating the dark or how to recover what should not be seen. *IACR Cryptology ePrint Archive*, 2018:1001, 2018. *Cited on page 41.*

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. *Cited on pages 6, 40, 48, and 54.*

[Cha20] Amit Chaulwar. Private dataset generation using privacy preserving collaborative learning. *ArXiv*, abs/2004.13598, 2020. *Cited on page 34.*

[CPS06] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *International Workshop on Frontiers in Handwriting Recognition*, 2006. *Cited on page 69.*

[CRS20] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. In *27th Annual Network and Distributed System Security Symposium, NDSS*, pages 23–26, 2020. *Cited on page 55.*

[CXC+21] Shaoqi Chen, Dongyu Xue, Guohui Chuai, Qiang Yang, and Qi Liu. Fl-qsar: a federated learning-based qsar prototype for collaborative drug discovery. *Bioinformatics*, 36(22-23):5492–5498, 2021. *Cited on page 34.*

[CYS21] Rishav Chourasia, Jiayuan Ye, and Reza Shokri. Differential privacy dynamics of langevin diffusion and noisy gradient descent. *Advances in Neural Information Processing Systems*, 2021. *Cited on pages 8, 79, 80, 81, 82, 83, 84, 86, 91, and 97.*

[DB19] Anirban Das and Thomas Brunschwiler. Privacy is what we care about: Experimental investigation of federated learning on edge devices. In *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, pages 39–42, 2019. *Cited on page 34.*

[DDS$^+$09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. *Cited on page 2.*

[DEK21] Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2183–2200, 2021. *Cited on pages 55 and 76.*

[DGBL$^+$16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. Technical report, February 2016. *Cited on page 40.*

[DKM$^+$06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006. *Cited on page 20.*

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006. *Cited on pages 4 and 80.*

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. *Cited on pages 19, 31, 32, 48, 54, and 57.*

[DR$^+$14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014. *Cited on pages 20 and 58.*

[DRV10] Cynthia Dwork, Guy N Rothblum, and Salil Vadhan. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60. IEEE, 2010. *Cited on pages 21, 22, and 80.*

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015. *Cited on page 55.*

[DZ16] Tamara Dugan and Xukai Zou. A survey of secure multiparty computation protocols for privacy preserving genetic tests. In *International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 173–182. IEEE, 2016. *Cited on page 54.*

[EEF$^+$18] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018. *Cited on page 26.*

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985. *Cited on page 14.*

[FH19] Stéphane Foucart and Stéphane Horel. Données de santé : conflit d'intérêts au cœur de la nouvelle plate-forme. *Le Monde*, 2019. *Cited on page 2.*

[FJR15] 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, 2015. *Cited on pages 4, 37, 38, 54, and 58.*

[FMTT18] Vitaly Feldman, Ilya Mironov, Kunal Talwar, and Abhradeep Thakurta. Privacy amplification by iteration. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 521–532. IEEE, 2018. *Cited on page 80.*

[FYS$^+$20] Angelo Feraudo, Poonam Yadav, Vadim Safronov, Diana Andreea Popescu, Richard Mortier, Shiqiang Wang, Paolo Bellavista, and Jon Crowcroft. Colearn: enabling federated learning in mud-compliant iot edge networks. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pages 25–30, 2020. *Cited on page 34.*

[GBDL$^+$16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016. *Cited on page 55.*

[GBDM20] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems*, 33:16937–16947, 2020. *Cited on page 39.*

[GDD$^+$21] Antonious Girgis, Deepesh Data, Suhas Diggavi, Peter Kairouz, and Ananda Theertha Suresh. Shuffled model of differential privacy in federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2521–2529. PMLR, 2021. *Cited on pages 97 and 103.*

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009. *Cited on pages 5 and 6.*

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. *Cited on page 48.*

[GJW$^+$19] Dashan Gao, Ce Ju, Xiguang Wei, Yang Liu, Tianjian Chen, and Qiang Yang. Hhhfl: Hierarchical heterogeneous horizontal federated learning for electroencephalography. *ArXiv*, abs/1909.05784, 2019. *Cited on page 34.*

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. *Cited on page 50.*

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, pages 218–229. ACM, 1987. *Cited on page 6.*

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009. *Cited on pages 12 and 54.*

[Gro75] Leonard Gross. Logarithmic Sobolev inequalities. *American Journal of Mathematics*, 97(4):1061–1083, 1975. *Cited on page 83.*

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013. *Cited on page 50.*

[HAA20] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Fednas: Federated deep learning via neural architecture search, 2020. *Cited on page 34.*

[HAPC17] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618, 2017. *Cited on pages 54 and 58.*

[HJC$^+$21] Adam James Hall, Madhava Jay, Tudor Cebere, Bogdan Cebere, Koen Lennart van der Veen, George Muraru, Tongye Xu, Patrick Cason, William Abramson, Ayoub Benaissa, et al. Syft 0.5: A platform for universally deployable structured transparency. *arXiv preprint arXiv:2104.12385*, 2021. *Cited on pages 7, 8, and 23.*

[HKSvdM19] Awni Hannun, Brian Knott, Shubho Sengupta, and Laurens van der Maaten. Privacy-preserving contextual bandits. *arXiv preprint arXiv:1910.05299*, 2019. *Cited on page 69.*

[HLL$^+$20] Sixu Hu, Yuan Li, Xu Liu, Qinbin Li, Zhaomin Wu, and Bingsheng He. The oarf benchmark suite: Characterization and implications for federated learning systems, 2020. *Cited on page 33.*

[HLS$^+$20] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning, 2020. *Cited on page 33.*

[Hop82] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982. *Cited on page 1.*

[HSJ$^+$17] Grant Ho, Aashish Sharma, Mobin Javed, Vern Paxson, and David Wagner. Detecting credential spearphishing in enterprise settings. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 469–485. USENIX, 2017. *Cited on page 38.*

[HSKS20] Thomas Hiessl, Daniel Schall, Jana Kemnitz, and Stefan Schulte. Industrial federated learning–requirements and system design. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 42–53. Springer, 2020. *Cited on page 35.*

[HSS$^+$18] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018. *Cited on page 55.*

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. *Cited on pages 16, 55, 73, 118, and 120.*

[IVF19] Selim Ickin, Konstantinos Vandikas, and Markus Fiedler. Privacy preserving qoe modeling using collaborative learning. In *Proceedings of the 4th Internet-QoE Workshop on QoE-Based Analysis and Management of Data Communication Networks*, pages 13–18, 2019. *Cited on page 33.*

[JA19] Madeleine Jansson and Måns Axelsson. Federated learning used to detect credit card fraud. Master's thesis, 2020-06-19. *Cited on page 35.*

[Jou04] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, September 2004. *Cited on page 50.*

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *USENIX Security Symposium 18*, pages 1651–1669, 2018. *Cited on page 55.*

[KCFH05] Balaji Krishnapuram, Lawrence Carin, Mário AT Figueiredo, and Alexander J Hartemink. Sparse multinomial logistic regression: Fast algorithms and generalization bounds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):957–968, 2005. *Cited on page 120.*

[KGC$^+$18] Daniel S Kermany, Michael Goldbaum, Wenjia Cai, Carolina CS Valentim, Huiying Liang, Sally L Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5):1122–1131, 2018. *Cited on pages 95, 100, and 120.*

[KKB18] Harmanjeet Kaur, Neeraj Kumar, and Shalini Batra. An efficient multi-party scheme for privacy preserving collaborative filtering for healthcare recommender system. *Future Generation Computer Systems*, 86:297–307, 2018. *Cited on page 54.*

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (2nd ed.)*. CRC press, 2014. *Cited on pages 14, 15, 17, and 18.*

[KLM$^+$18] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J. Wu. Function-hiding inner product encryption is practical. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 544–562. Springer, Heidelberg, September 2018. *Cited on page 51.*

[KLS21] Peter Kairouz, Ziyu Liu, and Thomas Steinke. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning*, pages 5201–5212. PMLR, 2021. *Cited on pages 97 and 103.*

[KMA$^+$19] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David

Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and Open Problems in Federated Learning. *arXiv:1912.04977 [cs, stat]*, December 2019. arXiv: 1912.04977. *Cited on page 25.*

[KMRB20] Georgios A Kaissis, Marcus R Makowski, Daniel Rückert, and Rickmer F Braren. Secure, privacy-preserving and federated machine learning in medical imaging. *Nature Machine Intelligence*, pages 1–7, 2020. *Cited on page 34.*

[KMY$^+$16] Jakub Konečnỳ, H Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016. *Cited on page 54.*

[KNH14] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014. *Cited on pages 73, 117, and 120.*

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016. *Cited on page 60.*

[KPN$^+$19] Deep Kawa, Sunaina Punyani, Priya Nayak, Arpita Karkera, and Varshapriya Jyotinagar. Credit risk assessment from combined bank records using federated learning. *International Research Journal of Engineering and Technology (IRJET)*, 6(4):1355–1358, 2019. *Cited on page 35.*

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018. *Cited on page 54.*

[KRC$^+$20] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 336–353. IEEE, 2020. *Cited on page 55.*

[Kri12] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012. *Cited on page 40.*

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. *Cited on pages 55, 73, 118, and 120.*

[KVH$^+$21] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 2021. *Cited on page 103.*

[KXJ$^+$20] Jiawen Kang, Zehui Xiong, Chunxiao Jiang, Yi Liu, Song Guo, Yang Zhang, Dusit Niyato, Cyril Leung, and Chunyan Miao. Scalable and communication-efficient decentralized federated edge learning with multi-blockchain framework, 2020. *Cited on page 34.*

[KZPP$^+$21] Georgios Kaissis, Alexander Ziller, Jonathan Passerat-Palmbach, Théo Ryffel, Dmitrii Usynin, Andrew Trask, Ionésio Lima, Jason Mancuso, Friederike Jungmann, Marc-Matthias Steinborn, et al. End-to-end privacy preserving deep learning on multi-institutional medical imaging. *Nature Machine Intelligence*, pages 1–12, 2021. *Cited on pages 77, 99, and 100.*

[LBBH98] 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. *Cited on pages 16, 73, and 118.*

[LC10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. *Cited on pages 49, 73, and 117.*

[LGN$^+$20] Yi Liu, Sahil Garg, Jiangtian Nie, Yang Zhang, Zehui Xiong, Jiawen Kang, and M. Shamim Hossain. Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach. *IEEE Internet of Things Journal*, page 1–1, 2020. *Cited on page 35.*

[LJLA17] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the Conference on Computer and Communications Security*, pages 619–631, 2017. *Cited on pages 55, 68, 74, and 118.*

[LKC17] Gilles Louppe, Michael Kagan, and Kyle Cranmer. Learning to pivot with adversarial networks. *Advances in neural information processing systems*, 30, 2017. *Cited on page 43.*

[LLH$^+$20] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020. *Cited on pages 33 and 34.*

[LSSS14] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. *Advances in neural information processing systems*, 27, 2014. *Cited on page 40.*

[LWH19] Qinbin Li, Zeyi Wen, and Bingsheng He. Federated learning systems: Vision, hype and reality for data privacy and protection. *ArXiv*, abs/1907.09693, 2019. *Cited on pages 33 and 34.*

[LYK$^+$20] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020. *Cited on page 101.*

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005. *Cited on page 105.*

[Mir17] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pages 263–275. IEEE, 2017. *Cited on pages 20 and 80.*

[MJ20] Ruben Mayer and Hans-Arno Jacobsen. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Comput. Surv.*, 53(1), February 2020. *Cited on page 33.*

[MMR$^+$17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. *Cited on page 3.*

[MR03] Ciamac C Moallemi and Benjamin Roy. Distributed optimization in adaptive networks. *Advances in Neural Information Processing Systems*, 16, 2003. *Cited on page 3.*

[MR18] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the Conference on Computer and Communications Security*, pages 35–52, 2018. *Cited on pages 13, 55, 70, 71, 74, and 76.*

[MV70] Dragoslav S. Mitrinovic and Petar M. Vasic. *Analytic Inequalities*, volume 1. Springer, 1970. *Cited on page 93.*

[MZ17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017. *Cited on pages 55, 56, 68, 69, 74, and 118.*

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994. *Cited on page 105.*

[Nis04] Helen Nissenbaum. Privacy as contextual integrity. *Wash. L. Rev.*, 79:119, 2004. *Cited on page 2.*

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. *Cited on pages 39, 48, and 49.*

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999. *Cited on page 5.*

[Pol78] John M Pollard. Monte carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978. *Cited on page 15.*

[PPFM$^+$19] Jonathan Passerat-Palmbach, Tyler Farnan, Robert Miller, Marielle S Gross, Heather Leigh Flannery, and Bill Gleim. A blockchain-orchestrated federated learning architecture for healthcare consortia. *arXiv preprint arXiv:1910.12603*, 2019. *Cited on page 34.*

[PS20] Arpita Patra and Ajith Suresh. Blaze: Blazing fast privacy-preserving machine learning. *arXiv preprint arXiv:2005.09042*, 2020. *Cited on page 55.*

[PU19] Alexandre Piquard and Martin Untersinger. Données de santé : la plate-forme de la discorde. *Le Monde*, 2019. *Cited on page 2.*

[PVG^{+}11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. *Cited on page 44.*

[QQBAF20] Adnan Qayyum, Junaid Qadir, Muhammad Bilal, and Ala Al-Fuqaha. Secure and robust machine learning for healthcare: A survey. *IEEE Reviews in Biomedical Engineering*, 14:156–180, 2020. *Cited on pages 33 and 34.*

[RBP22] Théo Ryffel, Francis Bach, and David Pointcheval. Differential privacy guarantees for stochastic gradient langevin dynamics, 2022. *Cited on pages 8, 79, and 100.*

[RBSJL^{+}20] Nuria Rodríguez-Barroso, Goran Stipcich, Daniel Jiménez-López, José Antonio Ruiz-Millán, Eugenio Martínez-Cámara, Gerardo González-Seco, M. Victoria Luzón, Miguel Angel Veganzones, and Francisco Herrera. Federated learning and differential privacy: Software tools analysis, the sherpa.ai fl framework and methodological guidelines for preserving data privacy. *Information Fusion*, 64:270 – 292, 2020. *Cited on page 34.*

[RDS^{+}15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, and Michael Bernstein. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. *Cited on pages 73 and 117.*

[RG20] Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning. *arXiv preprint arXiv:2007.07646*, 2020. *Cited on page 40.*

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. *Cited on page 1.*

[RPB^{+}19] Théo Ryffel, David Pointcheval, Francis Bach, Edouard Dufour-Sans, and Romain Gay. Partially encrypted deep learning using functional encryption. *Advances in Neural Information Processing Systems*, 32, 2019. *Cited on pages 7, 8, 37, 39, and 47.*

[RRK18] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018. *Cited on page 55.*

[RRL^{+}18] Md Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.*, 11(1):61–79, 2018. *Cited on page 58.*

[RSB12] Nicolas Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 3167–3175, 2012. *Cited on page 91.*

[RSC^{+}19] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. {XONN}: Xnor-based oblivious deep neural network inference. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1501–1518, 2019. *Cited on page 55.*

[RTD$^+$18] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. In *NeurIPS 2018 Workshop Privacy-Preserving Machine Learning*, 2018. *Cited on pages 7, 8, 23, 48, 72, 73, and 103.*

[RTPB22] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis Bach. Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 2022. *Cited on pages 7, 8, 31, 34, and 47.*

[RWT$^+$18] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pages 707–721, New York, NY, USA, 2018. ACM. *Cited on pages 48, 55, and 57.*

[SBPB21] Shubham Singh, Shantanu Bhardwaj, Hemlatha Pandey, and Gunjan Beniwal. Anomaly Detection Using Federated Learning. In Poonam Bansal, Meena Tushir, Valentina Emilia Balas, and Rajeev Srivastava, editors, *Proceedings of International Conference on Artificial Intelligence and Applications*, volume 1164, pages 141–148. Springer Singapore, Singapore, 2021. *Cited on page 35.*

[SEA19] Microsoft SEAL (release 3.2). `https://github.com/Microsoft/SEAL`, February 2019. Microsoft Research, Redmond, WA. *Cited on pages 48 and 55.*

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. *Cited on pages 6 and 18.*

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. *Cited on page 105.*

[SS15] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the Conference on Computer and Communications Security*, pages 1310–1321, 2015. *Cited on page 54.*

[SS19] Congzheng Song and Vitaly Shmatikov. Auditing data provenance in text-generation models. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 196–206, 2019. *Cited on page 38.*

[SŞ20] Ahmet Ali SÜZEN and Mehmet Ali ŞİMŞEK. A novel approach to machine learning application to protection privacy data in healthcare: Federated learning. *Namık Kemal Tıp Dergisi*, 8(1):22–30, 2020. *Cited on pages 33 and 34.*

[SSSS17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017. *Cited on pages 4, 37, 38, 58, and 80.*

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. *Cited on pages 55, 73, and 118.*

[TB21] Florian Tramèr and Dan Boneh. Differentially private learning needs better features (or much more data). In *9th International Conference on Learning Representations, ICLR*, 2021. *Cited on pages 96 and 103.*

[THL+19] Fengyi Tang, Jialu Hao, Jian Liu, Huimei Wang, and Ming Xian. Pfdlis: privacy-preserving and fair deep learning inference service under publicly verifiable covert security setting. *Electronics*, 8(12):1488, 2019. *Cited on page 35.*

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016. *Cited on page 37.*

[VBT17] Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. Decentralized collaborative learning of personalized models over networks. In *Artificial Intelligence and Statistics*, pages 509–517. PMLR, 2017. *Cited on page 3.*

[VLBM08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML ’08, pages 1096–1103, New York, NY, USA, 2008. ACM. *Cited on page 42.*

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. *Cited on page 101.*

[War65] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965. *Cited on page 4.*

[WBK19] Yu-Xiang Wang, Borja Balle, and Shiva Prasad Kasiviswanathan. Subsampled rényi differential privacy and analytical moments accountant. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1226–1235. PMLR, 2019. *Cited on page 80.*

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: Efficient and private neural network training. (PETS 2019), February 2019. *Cited on pages 27, 31, 48, 55, 57, 69, and 74.*

[WNK20] Alexander Wood, Kayvan Najarian, and Delaram Kahrobaei. Homomorphic encryption for machine learning in medicine and bioinformatics. *ACM Computing Surveys (CSUR)*, 53(4):1–35, 2020. *Cited on page 33.*

[WSZ+19] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2512–2520. IEEE, 2019. *Cited on page 25.*

[WT11] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the International conference on machine learning*, pages 681–688. Citeseer, 2011. *Cited on page 81.*

[WTB$^+$21] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *Proceedings on Privacy Enhancing Technologies*, 2021. *Cited on pages 13, 28, 55, 69, 70, 71, 73, 74, 117, and 118.*

[WYG$^+$17] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 299–313, 2017. *Cited on pages 60 and 114.*

[WZX17] Jiayu Wu, Qixiang Zhang, and Guoxi Xu. Tiny imagenet challenge. Technical report, Available: http://cs231n.stanford.edu/reports/2017/pdfs/930.pdf, 2017. *Cited on pages 73 and 117.*

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986. *Cited on page 55.*

[YHZL19] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019. *Cited on page 37.*

[YLC$^+$19] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. Federated learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 13(3):1–207, 2019. *Cited on page 33.*

[YS22] Jiayuan Ye and Reza Shokri. Differentially private learning needs hidden state (or much faster convergence). *arXiv preprint arXiv:2203.05363*, 2022. *Cited on page 81.*

[YSMN21] Sen Yuan, Milan Shen, Ilya Mironov, and Anderson Nascimento. Label private deep learning training based on secure multiparty computation and differential privacy. In *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021. *Cited on page 103.*

[YSS$^+$21] A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, G. Cormode, and I. Mironov. Opacus: User-friendly differential privacy library in pytorch. *arXiv preprint arXiv:2109.12298*, 2021. *Cited on pages 21, 26, 80, and 103.*

[ZHKS16] Ruiyu Zhu, Yan Huang, Jonathan Katz, and Abhi Shelat. The cut-and-choose game and its application to cryptographic protocols. In *USENIX Security Symposium Security 16*, pages 1085–1100, 2016. *Cited on page 57.*

[ZJP$^+$20] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. The secret revealer: generative model-inversion attacks against deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 253–261, 2020. *Cited on page 58.*

[ZPPR$^+$20] Alexander Ziller, Jonathan Passerat-Palmbach, Théo Ryffel, Dmitrii Usynin, Andrew Trask, Ionésio Da Lima Costa Junior, Jason Mancuso, Marcus Makowski, Daniel Rueckert, Rickmer Braren, et al. Privacy-preserving medical image analysis. *arXiv preprint arXiv:2012.06354*, 2020. *Cited on page 35.*

[ZTL$^+$21] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, et al. Pysyft: A library for easy federated learning. In *Federated Learning Systems*, pages 111–139. Springer, 2021. *Cited on pages 7, 8, and 23.*

RÉSUMÉ

L'usage sans précédent du machine learning (ML) ou *apprentissage automatique*, motivé par les possibilités qu'il apporte dans un grand nombre de secteurs, interroge de plus en plus en raison du caractère sensible des données qui doivent être utilisées et du manque de transparence sur la façon dont ces données sont collectées, croisées ou partagées. Aussi, un certain nombre de méthodes se développent pour réduire son intrusivité sur notre vie privée, afin d'en rendre son usage plus acceptable notamment dans des domaines tels que la santé, où son potentiel est encore très largement sous-exploité. Cette thèse explore différentes méthodes issues de la cryptographie ou plus largement du monde de la sécurité et les applique au machine learning afin d'établir des garanties de confidentialité nouvelles pour les données utilisées et les modèles de ML.

Notre première contribution est le développement d'un socle technique pour implémenter et expérimenter de nouvelles approches au travers d'une librairie open-source nommée PySyft. Nous proposons une architecture modulaire qui facilite l'utilisation des briques de confidentialité ainsi que le développement et l'intégration de nouvelles briques. Ce socle sert de base à l'ensemble des implémentations proposées dans cette thèse.

Notre seconde contribution consiste à mettre en lumière la vulnérabilité des modèles de ML en proposant une attaque qui exploite un modèle entraîné et permet de révéler des attributs confidentiels d'un individu. Cette attaque pourrait par exemple détourner un modèle qui reconnaît le sport fait par une personne à partir d'une image, pour détecter les origines raciales de cette personne. Nous proposons des pistes pour limiter l'impact de cette attaque.

Dans un troisième temps, nous nous intéressons à certains protocoles de cryptographie qui permettent de faire des calculs sur des données chiffrées. Nous proposons un protocole de chiffrement fonctionnel qui permet de réaliser des prédictions sur des données chiffrées et de ne rendre public que la prédiction. Par ailleurs, nous optimisons un protocole de partage de secret fonctionnel, qui permet d'entraîner ou d'évaluer un modèle de ML sur des données de façon privée, c'est à dire sans révéler à quiconque ni le modèle ni les données. Ce protocole offre des performances suffisantes pour la réalisation de tâches non triviales comme la détection de pathologies dans les radiographies de poumons.

Enfin, nous intéressons à la confidentialité différentielle qui permet de limiter la vulnérabilité des modèles de ML et donc l'exposition des données utilisées lors de l'entraînement, en introduisant une perturbation contrôlée. Nous proposons un protocole qui offre notamment la possibilité d'entraîner un modèle lisse et fortement convexe en garantissant un niveau de confidentialité indépendant du nombre d'accès aux données sensibles lors de l'entraînement.

MOTS CLÉS

Intelligence Artificielle, Apprentissage Fédéré, Chiffrement Fonctionnel, Calculs Multipartites, Partage de Secret Fonctionnel, Confidentialité Différentielle, Intégrité Contextuelle

ABSTRACT

The ever growing use of machine learning (ML), motivated by the possibilities it brings to a large number of sectors, is increasingly raising questions because of the sensitive nature of the data that must be used and the lack of transparency on the way these data are collected, combined or shared. Therefore, a number of methods are being developed to reduce its impact on our privacy and make its use more acceptable, especially in areas such as healthcare where its potential is still largely under-exploited. This thesis explores different methods from the fields of cryptography and security, and applies them to machine learning in order to establish new confidentiality guarantees for the data used and the ML models.

Our first contribution is the development of a technical foundation to facilitate experimentation of new approaches, through an open-source library named PySyft. We propose a modular architecture that facilitates the use of privacy blocks, or the development and integration of new blocks. This library is reused in all the implementations proposed in this thesis.

Our second contribution consists in highlighting the vulnerability of ML models by proposing an attack that exploits a trained model to reveal confidential data. This attack could, for example, subvert a model that recognizes a person's sport from an image, to detect the person's racial origins. We propose solutions to limit the impact of this attack.

In a third step, we focus on some cryptographic protocols that allow us to perform computations on encrypted data. A first study proposes a functional encryption protocol that allows to make predictions using a small ML model over encrypted data and to only make the predictions public. A second study focuses on optimizing a functional secret sharing protocol, which allows an ML model to be trained or evaluated on data privately, i.e. without revealing either the model or the data to anyone. This protocol provides sufficient performance to use models that have practical utility in non-trivial tasks such as pathology detection in lung X-rays.

Our final contribution is in differential privacy, a technique that limits the vulnerability of ML models and thus the exposure of the data used in training by introducing a controlled perturbation. We propose a new protocol and show that it offers the possibility to train a smooth and strongly convex model with a bounded privacy loss regardless of the number of calls to sensitive data during training.

KEYWORDS

Artificial Intelligence, Federated Learning, Functional Encryption, Multi-Party Computation, Function Secret Sharing, Differential Privacy, Contextual Integrity