UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Sprint: Secure and Fast Outsourced Machine Learning

Sprint: Sicheres und Schnelles Auslagern von Maschinellem Lernen

Masterarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Jonas Sebastian Sander

ausgegeben und betreut von
Prof. Dr.-Ing Thomas Eisenbarth

mit Unterstützung von
Ida Bruns,
Dr. rer. nat. Sebastian Berndt und
Prof. Dr. Esfandiar Mohammadi

Lübeck, den 22. September 2021

Abstract

Machine learning solutions are moving faster and faster into more and more parts of our society. Cloud providers such as Amazon Web Services, Microsoft Azure, and the Google Cloud Platform, driven by huge profit potentials, aggressively expand their Machine-Learning-as-a-Service offerings. There is a risk that data security and privacy increasingly fall behind.

We investigate the current state of secure outsourcing of ML workloads to the cloud, concentrating on deep convolutional neural networks. We find that the most common and performant mixed SMPC approaches based on homomorphic encryption, secret sharing, and garbled circuits underly a communication overhead that grows linearly in the depth of the neural network. We present Sprint, a scheme for fast and secure outsourcing of ML workloads to the cloud. Sprint is based purely on arithmetic garbled circuits, needs only a single communication round per inference step regardless of the depth of the neural network, and requires compared to recent schemes, a 22 times smaller communication volume for ResNet32.

Kurzfassung

Lösungen des Maschinellen Lernens erhalten immer schneller in immer weitere Teile unserer Gesellschaft Einzug. Cloud Provider wie Amazon Web Services, Microsoft Azure und die Google Cloud Platform bauen, getrieben durch immense Gewinnpotentiale, aggressiv ihre Machine-Learning-as-a-Service Angebote aus. Es besteht die Gefahr, dass Datensicherheit und Privatsphäre zunehmend ins Hintertreffen geraten.
Wir untersuchen den aktuellen Stand des sicheren Auslagerns von ML Workloads in die Cloud und konzentrieren uns insbesondere auf tiefe Neuronale Faltungsnetze. Dabei stellen wir fest, dass die verbreitetsten und performantesten gemischten SMPC Ansätze basierend auf Techniken der Homomorphen Verschlüsselung, Secret Sharing und Garbled Circuits einem in der Tiefe des Neuronalen Netzes linear wachsenden Kommunikations-Overhead unterliegen. Wir präsentieren Sprint, ein Framework zum schnellen und sicheren auslagern von ML Workloads in die Cloud. Sprint basiert auf einem reinen Arithmetischen Garbled Circuit Ansatz, benötigt unabhängig von der Tiefe des Neuronalen Netzes nur eine einzige Kommunikationsrunde pro Inferenz-Schritt und besitzt im Vergleich zu aktuellen Schemes ein 22-mal kleineres Kommunikationsvolumen für ResNet32.

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Lübeck, 22. September 2021

Acknowledgements

An dieser Stelle möchte ich mich zur aller erst bei meinen Eltern und meinen beiden kleinen Geschwistern bedanken. Danke, dass Ihr immer für mich da seid und mir in den paar Frust-Phasen des Studiums mit Rat und Ablenkung zur Seite standet. Das vielleicht wichtigste meines Studiums sind meine Kommilitonen. Ihr seid die Besten und ich möchte mich bei Euch für die unvergessliche Zeit und die vielen schönen und lustigen Erinnerungen bedanken.
Auch die Mitarbeiter des ITS haben mir immer das Gefühl gegeben, willkommen zu sein. Egal, wen ich um Rat gefragt habe, ich bin immer auf Hilfsbereitschaft gestoßen. Besonders möchte ich mich bei Prof. Eisenbarth und Prof. Mohammadi für die Möglichkeit bedanken, eine Masterarbeit mit meinen Lieblingsthemen des Studiums schreiben zu können. Danke Ida für deine Geduld bei der Betreuung meiner Arbeit und den vielen Tipps und Hinweisen zu meinen Texten. Auch möchte ich mich bei Sebastian für die Hilfe bei den Crypto Themen und dem Knobeln an den Half Gates bedanken. Danke Florian für die Hilfestellung bei meinen ersten praktischen Gehversuchen mit Intel SGX. Thanks, Okan, for the discussions about the half gates!

Contents

Nomenclature

Abbreviations

OML	Outsourced Machine Learning
GC	Garbled Circuit
HG	Half Gate
HE	Homomorphic Encryption
SS	Secret Sharing
ANN	Artificial Neuronal Network
TEE	Trusted Execution Environment
FPU	Fast Processing Unit
MRS	Mixed-Radix System
CPM	Composite Primorial Modulus
CRT	Chinese Remainder Theorem

Letters and Symbols

$\oplus$	Logical XOR
J	Error
L	Loss
$x^{(i)}$	i-th sample of a dataset
$X^{\{i\}}$	i-th batch of a dataset
$a^{[i]}$	ANN-Activations of layer i
sgn	Sign function
ReLU	Rectified Linear Unit function
sk_a^b	Secret key for value b on wire a
$\text{EN}_{sk}(C)$	Encrypt C using sk

1 Introduction

1.1 Motivation

The recent progress of machine learning (ML) with its many technical and practical innovations crosses almost all industries and government institutions while getting more and more applied to security and privacy sensitive domains like healthcare, law enforcement, finance, public administration, logistics, and many more. A large part of these applications require substantial computational resources and while cloud providers push their Machine-Learning-as-a-Service (MLaaS) offerings, the complexity of the hardware and software stacks is continually growing and exposes an expanding surface for potential attackers. A rapidly growing need for methods and techniques for secure and privacy-preserving outsourcing of ML applications arises, which moreover meets a political and legal necessity through the General Data Protection Regulation (GDPR) of the European Union and related laws worldwide. In this thesis, we introduce the cryptographic methods used, summarize the current developments in outsourced ML (OML) and research optimization approaches for OML, to further reduce the computational performance gap to conventional ML schemes.

1.2 Research Path

In recent years, artificial neuronal networks (ANNs) have proven to be highly flexible in their application and dominate large areas of the ML landscape. As this class of learning algorithms is highly computational intense, it is especially urgent to develop suitable OML methods. This thesis focuses on the secure outsourcing of ANNs.
We find that Tramèr and Boneh's scheme Slalom [TB19] (see also 3.1), which leverages a trusted execution environment (TEE) and outsources linear workloads to a fast processing unit (FPU) - a hardware accelerator faster than the CPU, is the most promising approach for secure and efficient outsourcing of ANNs. As Slalom's main weakness, we identified the significant communication overhead in the online phase, where after each layer of the outsourced ANN, the TEE and FPU communicate. The communication is necessary because the typical activation functions of ANNs are not homomorphic. The past has shown that in practice, particularly deep neural networks often deliver the best results and therefore incur a high communication overhead in Slalom. This thesis raises the re-

search question of how the communication between the layers can be reduced or avoided entirely.

Our research path starts in the area of mixed-SMPC OML-approaches for ANNs, which combine secret-sharing (SS), homomorphic encryption (HE), and garbled circuit (GC) based techniques. While HE and SS are efficient regarding linear operations, the non-linear activation functions of ANNs lead to large overheads. Therefore, a typical approach computes the linear components via HE or SS and the activation functions via GCs [CL01, BOP06, OPB07, BFL$^+$11, MZ17, LJLA17, RWT$^+$18, MR18, MLS$^+$20]. Unfortunately, all these approaches have a linear number of communication rounds in the depth of the ANN, and we were not able to switch between SMPC techniques without communicating between ANN-layers[1].

Following a line of work about OML purely based on GCs [SS08, RRK18, RSC$^+$19], we switched the perspective and considered the Slalom setting of a TEE with a co-located FPU from this view. In particular, we follow the work of Marshall Ball et al. on highly optimized arithmetic GCs [BMR16] and ANN-specific GC optimizations [BCM$^+$19]. In this way, the Slalom becomes a Sprint. The idea is to garble the whole ANN within the TEE and to accelerate the evaluation of the GC, respectively the ANN-inference, on the FPU. Thus, no communication between the layers of the ANN is needed. During the offline phase, the GC is transferred to the FPU. During the online, phase only the garbled input is sent to the FPU, and the garbled inference result from the FPU to the TEE.

We started by implementing our scheme in abstraction layers on the CPU, planning to port it to the GPU afterward. Unfortunately, with the CPU implementation finished, the time of the thesis was well advanced[2]. The bases for the mixed-radix system (MRS) determined to be performant by Ball et al. [BCM$^+$19] were not usable, so we had to implement our own search. Becoming familiar with Nvidia GPUs' and Intel SGXs' hardware characteristics, CUDA programming, the GPU and SGX drivers, and CUDA tools on Linux took considerably more time than planned. In order to develop an efficient and highly parallel graphical processing units (GPU) port of garbled ANNs, a new circuit representation is needed that exposes the parallelism of the gates within the gadgets and ANN layers and takes the memory, cache, and computation properties of the GPU into account. Therefore, we were unfortunately not able to complete the whole GPU port until the submission of the thesis. At the end of this thesis, it is not yet possible to make a concrete evaluation of our scheme Sprint as a whole. However, we show that our current implementation is com-

[1]In particular, we have explored the possibilities of SS-based OML approaches such as the GMW protocol and Beaver triples (see chapter 7.3).

[2]Implementing arithmetic GC schemes is an error-prone task since all intermediate values of their computations are properly encrypted, randomly permuted, and frequently switch between representations, which massively increase debugging times.

petitive and that Sprint clearly outperforms recent schemes in terms of the communication overhead in the online phase.

1.3 Organization

We start by introducing the necessary background knowledge. First, we introduce the basic concept of ANNs via linear and logistic regression in section 2.1. Then we give a short introduction to the field of secure multi-party computation, including its relation to OML in section 2.2 and explain in a functional way how the GC protocol works in section 2.3. Next, we introduce the practically most relevant optimizations for conventional binary GCs in section 2.4, including point-and-permute, garbled row-reduction 3, free-xor, and half gates. Finally, we briefly introduce the principles of TEEs, particularly Intel SGX in section 2.5 and CUDA-enabled GPUs including the concept of general-purpose computing on graphics processing units (GPGPU) in section 2.6. After introducing the background knowledge, we summarize a selection of important works on OML and evaluate them, especially concerning their performance in the online phase in chapter 3.
Based on the binary GCs and their optimizations, we introduce the garbling techniques for Sprint in chapter 4. We start by introducing the garbling gadgets for mixed-modulus arithmetic GCs in section 4.1, including free addition, free multiplication by a public constant, and a unary projection gate for arbitrary functions. Further, we describe in detail cryptographic and ANN specific optimizations for arithmetic GCs in section 4.2. In particular, these optimizations are used for accelerated sign and ReLU activation functions and, starting from the ReLU function, for max-pooling. The necessary gates include a half-gate generalization for arithmetic GCs, mixed-modulus half gates, mixed-radix addition, and an approximated sign gadget. We then discuss the use of fixed-key AES as a computational-wise optimization in section 4.3.
Afterward, we present our OML approach Sprint step by step along its three abstraction levels, outline our CPU implementation and discuss details of a GPU implementation in chapter 5. Finally, we present the result of the CPU implementation, including discussions about various performance-accuracy tradeoffs, Sprint's online communication overhead, and feature set in chapter 6. In the end, we summarize and present an outlook and ideas for future work in chapter 7.

2 Background

2.1 Machine Learning

In this section, we introduce the inner workings of ANNs as a generalization of *Linear Regression* (LiR) and *Logistic Regression* (LoR). We end the chapter with a complete description of the inference and training of a small example ANN.

2.1.1 Linear Regression

LiR is used in predictive modeling to learn the linear relation between m samples $X^{m \times n}$, or more specifically their n features, and a single continuous explanatory variable y^m of a given dataset (see figure 2.1a).[3] We leverage the *bias-trick* and introduce a weight w_0 and a constant input $x_{1:n,0} = 1$ to hide the bias inside the weight matrix w^{n+1}. One option to learn the parameters w^{n+1} of the model $f_w(x^{(i)}) = \hat{y}^{(i)} = \sum_{j=1}^{n} x_j^{(i)} w_j$ is to perform *gradient descent*[4] on the *mean squared error*[5] $J(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)})$, with $L(y^{(i)}, \hat{y}^{(i)}) = \frac{1}{2}(y^{(i)} - \hat{y}^{(i)})^2$. Gradient descent iteratively updates the model weights with respect to the loss on the training data. Each iteration consists of one update step $w = w - l \cdot \nabla_w J = w - \frac{l}{m} X^T (Xw - y)$, with l being the learning rate to control the process's convergence speed. In practice, it is common to use *stochastic gradient descent* (SGD), where the weights are updated in each iteration with respect to a random subset of the training set, called *mini-batch*. If the batch size is greater than one, the algorithm is called *mini-batch SGD*, which allows leveraging the vectorization capabilities of modern FPUs and speeds up training times.

2.1.2 Logistic Regression

LoR is a *classification algorithm*, which models the probability that a given sample relates to a specific class and outputs a discrete class value (see figure 2.1b).[6] To limit the prediction value between zero and one, meaning the sample is member of the positive or

[3]We focus on *simple LiR* with only one explanatory variable.

[4]It is also possible to find the optimal parameters for LiR in just one step, leveraging the so-called normal equations.

[5]The mean squared error can be derived via maximum likelihood estimation of w, assuming the errors per sample being independently gaussian distributed [Mur12].

[6]We focus on binary classification ($y \in \{0, 1\}$).

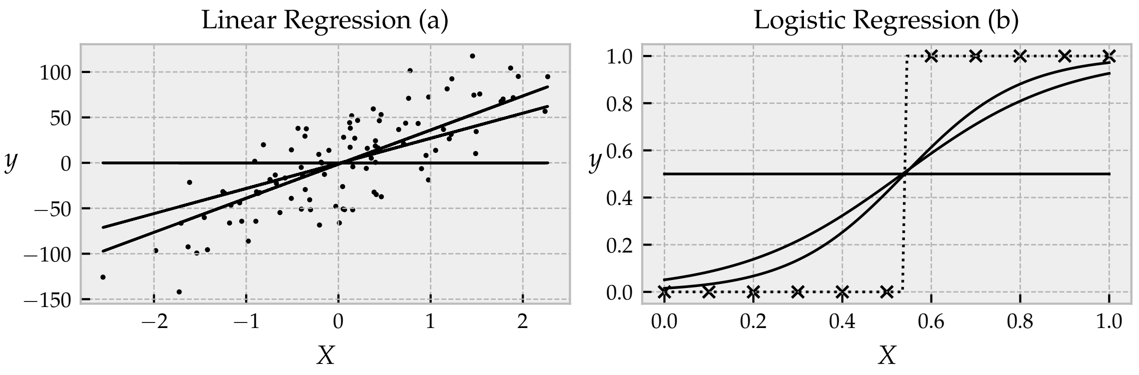


Figure 2.1: Training progress of a LiR and a LoR in the context of example datasets. (a) The LiR was trained over 100 epochs with a learning rate $l = 0.01$. The model parameters are plotted after initialization, 50 and 100 epochs. (b) The LoR was trained over 200 epochs with $l = 0.5$, and parameters are plotted after initialization, 100 and 200 epochs. The dotted line shows the final classification border.

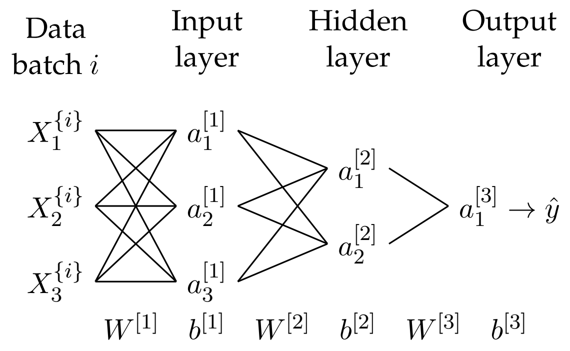
negative class, the output of the linear model f_w is passed through the *logistic activation function* $\sigma(x) = \frac{1}{1+e^{-x}}$; $g_w(x^{(i)}) = \hat{y}^{(i)} = \sigma(f_w(x^{(i)}))$. The final classification is then performed using a threshold, typically 0.5. To learn the parameters w, one can again use gradient descent. In case of LoR therefore the *binary cross-entropy*[7] $J(y, \hat{y}) = \frac{1}{m}\sum_{i=1}^{m} L$, with $L(y^{(i)}, \hat{y}^{(i)}) = -(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$, is used. This leads to a similar update step of gradient descent, as in LiR: $w = w - l \cdot \nabla_w J = w - \frac{l}{m} X^T(\sigma(Xw) - y)$.[8]

2.1.3 Artificial Neuronal Networks

Artificial neural networks (ANN) can be considered as a powerful generalization of LoR. They consist of layers of so-called neurons (see figure 2.2 for a small example), where each of the $n^{[i]}$ neurons $a_j^{[i]}$ of layer i computes a LoR on the outputs of all neurons of the previous layer $i - 1$: $a^{[i]} = \sigma(W^{[i]} a^{[i-1]} + b^{[i]})$, with $a^{[0]} = X$ being the input. Note that the bias-trick can also be applied to the training of ANNs, but we omit it here for clarity. Computing this step from the input to the output layer is also referenced as *forward-propagation* and is the only computation needed to perform predictions using a trained ANN. For a mini-batch $X^{\{i\}}$ of size B and the example ANN, forward-propagation is performed via the operations on the right side of figure 2.2. As one can see, the matrix shapes do not

[7]The binary cross-entropy can be derived via maximum likelihood estimation of w, assuming the samples being independently generated from a Bernoulli distribution [Mur12].

[8]It is also possible to find useful parameters for LoR, with many fewer iterations, using Fisher's scoring. If n is not too large, this approach converges faster than SGD.



$$z^{[1]} = W^{[1]} \cdot X^{\{i\}} + b^{[1]}, a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}, a^{[2]} = \sigma(z^{[2]})$$
$$z^{[3]} = W^{[3]} \cdot a^{[2]} + b^{[3]}, a^{[3]} = \sigma(z^{[3]})$$

$$(3, B) = (3, 3) \cdot (3, B) + (3, 1)$$
$$(2, B) = (2, 3) \cdot (3, B) + (2, 1)$$
$$(1, B) = (1, 2) \cdot (2, B) + (1, 1)$$

Figure 2.2: Architecture and calculations of example ANN. The left side shows the architecture of the example ANN consisting of three fully connected layers. It expects input samples with three features $(x_1, x_2, x_3)^T$ and outputs a binary verdict $\hat{y}$. The right side shows the calculations for inference as well as the corresponding matrix dimensions.

match when $B > 1$. To allow $B > 1$ for training and prediction, in practical ML, a technique called *broadcasting* is used that efficiently "broadcasts" operations such as addition over matrices so that their shapes match [HMvdW+20]. In the case of the bias, broadcasting would simulate a matrix of the form $(3, B)$ that repeats the original bias vector b times along the second axis and results in a valid matrix addition.

ANNs are usually trained with mini-batch SGD or related optimizers, such as Adam [KB15]. Here we focus on mini-batch SGD with the binary cross entropy as error function. To be able to update the weights $W^{[i]} = W^{[i]} - l\nabla_{W^{[i]}} J$, and biases $b^{[i]} = b^{[i]} - l\nabla_{b^{[i]}} J$, we have to propagate the error from the output layer via *backward-propagation* to layer i, resulting in $\frac{\partial L}{\partial W^{[i]}} = \frac{\partial L}{z^{[i]}} \cdot a^{[i-1]T}$, with $a^{[0]} = X^{\{i\}}$, and $\frac{\partial L}{\partial b^{[i]}} = \frac{\partial L}{z^{[i]}}$. For the example ANN, the weight updates are as follows (for simplicity matrices are shown without the mini-batch indices, $\odot$ note the element-wise multiplication)

$$\frac{\partial L}{\partial z^{[3]}} = a^{[3]} - y \qquad (1, B) = (1, B) - (1, B)$$
$$\frac{\partial J}{\partial W^{[3]}} = \frac{1}{B} \odot \frac{\partial L}{\partial z^{[3]}} \cdot a^{[2]T} \qquad (1, 2) = (1, 1) \odot (1, B) \cdot (B, 2)$$
$$\frac{\partial L}{\partial z^{[2]}} = W^{[3]T} \cdot \frac{\partial L}{\partial z^{[3]}} \cdot \sigma'(a^{[2]}) \qquad (2, B) = (2, 1) \cdot (1, B) \odot (2, B)$$
$$\frac{\partial J}{\partial W^{[2]}} = \frac{1}{B} \odot \frac{\partial L}{\partial z^{[2]}} \cdot a^{[1]T} \qquad (2, 3) = (1, 1) \odot (2, B) \cdot (B, 3)$$
$$\frac{\partial L}{\partial z^{[1]}} = W^{[2]T} \cdot \frac{\partial L}{\partial z^{[2]}} \cdot \sigma'(a^{[1]}) \qquad (3, B) = (3, 2) \cdot (2, B) \odot (3, B)$$
$$\frac{\partial J}{\partial W^{[3]}} = \frac{1}{B} \odot \frac{\partial L}{\partial z^{[1]}} \cdot X^T \qquad (3, 3) = (1, 1) \odot (3, B) \cdot (B, 3).$$

Before training, the biases are usually initialized to zero, while the weights are set using the *glorot initialization* $W^{[i]} \sim \sqrt{n^{1-i}}$ (for sigmoid activation) to address the problem of exploding or vanishing gradients [GB10]. In addition to the sigmoid function, a variety of other activations functions exist. Very widely used is the rectified linear unit, short ReLU function $f(x) = \max(0, x)$, because it can significantly accelerate the training in practice [KO11].
In addition to the fully connected layers described above, a large and dynamic ecosystem of additional layers and model architectures has evolved for a wide range of use cases such as image or natural language processing. While, in the first place, the algorithmic and hardware developments of recent years have made the rapid development in the field of deep learning possible, a selection of powerful frameworks such as TensorFlow [AAB+15] (with Keras [C+15]), PyTorch [PGM+19] and JAX [BFH+18] was developed by the open-source community and drive today's deep learning progress. Due to their auto-diff modules and performance optimizations, these frameworks enable particularly deep and complex networks to be implemented and trained with minimal effort. For training the models of our experiments, we use TensorFlow and Keras.

2.2 Secure Multiple-Party Computation and Outsourced Machine Learning

In this section, we introduce the field of secure multi-party computation (SMPC) and its relationship to OML. OML is strongly related to the cryptographic area of SMPC[9], which deals with a setting of two or more parties who jointly compute a given function without revealing their inputs to each other to ensure *privacy* while guaranteeing the *correctness* of the computed results. In contrast to classical cryptography, which often considers attackers from outside, SMPC always considers one or more parties involved in the collective computation of the given function to be *malicious* or at least *curious* about others' private inputs. In this way, the extensively studied and large SMPC research field provides a good theoretical foundation for considering ML outsourcing. Indeed, many SMPC techniques and, in particular, the SMPC attacker models are used to develop OML schemes.
If we neglect the massive computational resource requirements of ML, along with the need for specialized hardware and the increasing importance of side-channel resistance, OML can be viewed as a specialized form of the typical SMPC setting. Specially, because in contrast to SMPC, OML often additionally requires a predefined distribution of the computational workload among the participants. In our case, a client with limited computing capabilities in possession of some data would like to outsource expensive ANN-inference

[9]SMPC is also often called SFE: Secure Function Evaluation.

to a server without the server learning anything about the data inputs and possible also demand integrity protection over the computed function to prevent undetected cheating from a malicious server. Obviously, for OML, many different constellations of parties and security objectives are plausible, and we consider further examples in chapter 3.
Whereas many different settings for ML outsourcing schemes are thinkable and practically useful, besides the *feasibility* question, the protocols' *efficiency* is crucial. There are many surprising SMPC solutions for different settings that have been presented over the years, but many of these theoretically elegant solutions are by far not efficient enough to be of practical use. These efficiency difficulties are further amplified by the additional requirements in the OML setting and must be considered in the development and analysis of appropriate schemes.

2.2.1 Ideal/Real Simulation Paradigm

Before we model the capabilities of a potential attacker in the next section, we first define what properties a protocol must satisfy to be considered secure. Hazay et al. [HL10] list the following essential properties, which can be used to cover many if not all multi-party computation tasks and, in this way, also OML:

- *Privacy*: No party should be able to learn more than its outputs from the jointly computed function[10].
- *Correctness*: The output of each party is correct, meaning the function evaluation can not be tampered from an attacker.
- *Independence of Inputs*: Corrupted parties choose the input independent of the input of honest parties.
- *Guaranteed Output Delivery*: Corrupted parties should not be able to intercept the outputs of honest parties.
- *Fairness*: Corrupted parties get their outputs only when the honest parties get their outputs as well.

While the above list gives a good idea about the rough requirements for an SMPC scheme, it does not provide a security definition. The modern formalization, the *Ideal/Real Simulation Paradigm*, was introduced (not in that terminology) by Goldwasser [GM84] and is

[10]That the jointly computed output does not reveal more information than intended is not considered in the SMPC definition and must be ensured separately. In the case of non-trivial functions such as machine learning models, corresponding privacy guarantees can be ensured via the framework of differential privacy [Dwo08, ACG$^+$16].

derived via a mental experiment in which an ideal function evaluation via a trusted third party (TTP) is considered. This party performs the computations that should otherwise be computed using the SMPC protocol distributed over all participants. For this purpose, the SMPC protocol participants send their inputs via secure channels to the third party, which in turn performs the necessary computations and sends the appropriate outputs to the corresponding parties.
Obviously, this protocol meets all the security requirements described above, but in the real function-evaluation does not exist a TTP. The simulation paradigm states that a real protocol is secure if no attacker can do more damage in the real function evaluation than in the ideal function evaluation. In other words, a secure protocol should simulate the ideal function evaluation (same input/output distributions). If an attacker can now successfully execute an attack in the real setting, this attacker is also successful in the ideal setting. However, we know that no attacks are possible in the ideal setting and can conclude that the real function evaluation must also be secure. In many settings, as in ours, where half or more of the participants are dishonest, we often can not achieve the output guarantee and fairness requirements and relax the security definition by dropping them.

2.2.2 Attacker Models

To truly comprehend the security achieved by a protocol (or carry out rigorous proofs), it is important to model, in addition to the security requirements, the capabilities of a potential attacker. It must be determined whether benign parties can become dishonest adaptive during protocol execution, known as *adaptive corruption*, or whether they are fixed before the protocol starts, known as *static corruption*. In the OML settings discussed below, we consider only static distributions of dishonest parties and neglect the adaptive case.
It must also be determined whether a dishonest party can deviate from the protocol or not. Therefore one distinguishes between *semi-honest* and *malicious* attackers[11]. The former attacker is weaker and follows the protocol, but receives all internal information from all corrupted parties and tries to learn private information from benign parties. The malicious attacker, on the other hand, can arbitrarily deviate from the protocol to undermine the security requirements of the protocol, and efficiently achieving security in this setting is often much more difficult.
It is natural to question the usefulness of the semi-honest model. One might think, what is the benefit of a security model that requires a potential attacker to be honest and follow the protocol. On the one hand, this model ensures that there are no unintentional

[11]Semi-honest attackers are also called honest-but-curious or passive and malicious attackers are also called active or byzantine.

information leaks among honest parties. On the other hand, there are protocols, such as the garbled circuits presented later, for which conversions to the malicious model exist [GMW87, LP07], and the semi-honest model often provides a good stepping stone for such developments.

Last, we also need to model the computational capabilities of the potential attacker. Therefore, we distinguish between the *information-theoretic model* (also known as unconditional or statistical security), in which the attacker has unlimited computational resources and all parties communicate via perfect private channels, and the *computational model*, in which the attacker performs computations in polynomial time and the classical cryptographic assumptions hold. The following discussions refer exclusively to the computational model.

2.3 Garbled Circuits

Before we get to the optimizations for GCs, we introduce the Oblivious Transfer and the most basic form of the GC protocol in this chapter.

2.3.1 Oblivious Transfer

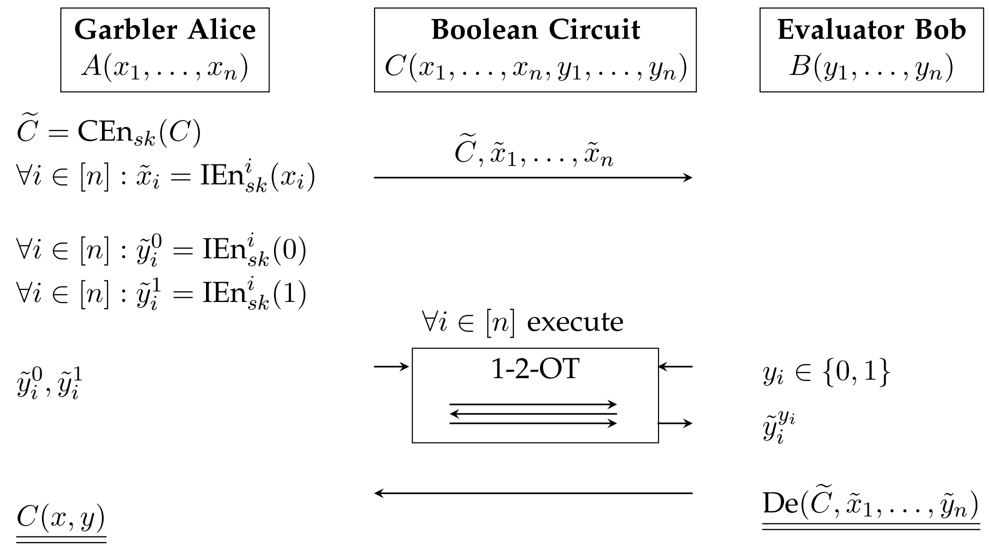
Rabin's Oblivious Transfer (OT) is a protocol between two parties, the sender and the receiver [Rab05]. The sender transfers two bits to the receiver, and the receiver learns only one of them with a probability of $1/2$. The other bit becomes oblivious, meaning the receiver can not infer it from the protocol transcript. Besides, the sender cannot recognize which bit the receiver has learned. The *1-out-of-2 Oblivious Transfer* (1-2 OT) [EGL82] behaves the same way, except that the receiver can choose which of the bits he wants to learn. Indeed, Claude Crépeau [Cré87] has shown that both variants are equivalent. Naor and Pinkas [NP99] have shown that the 1-2 OT generalizes efficiently to a 1-n OT. In the large and well-studied SMPC field, the Oblivious Transfer has established itself as one of the most important primitives and plays a central role in the protocols presented below.

2.3.2 Garbled Circuit Protocol

Garbled circuits (GC) were introduced by Andrew Yao [Yao86] and allow SMPC computations of binary circuits in the two party setting under a semi-honest attacker. Figure 2.3 schematically shows the garbled circuit protocol procedure between a *garbler* Alice and an *evaluator* Bob. Here, the public circuit C should be computed under Alice's private inputs x_1 to x_n and Bob's private inputs y_1 to y_n.

In the first step, Alice encrypts the circuit C and her inputs x_1 to x_n with her secret key[12]

[12]The keys used to encrypt wire values of GCs are often also referred to as labels.



Garbler Alice
$A(x_1, \ldots, x_n)$
Boolean Circuit
$C(x_1, \ldots, x_n, y_1, \ldots, y_n)$
Evaluator Bob
$B(y_1, \ldots, y_n)$
$\widetilde{C} = \text{CEn}_{sk}(C)$
$\forall i \in [n] : \tilde{x}_i = \text{IEn}^i_{sk}(x_i)$
$\widetilde{C}, \tilde{x}_1, \ldots, \tilde{x}_n$
$\forall i \in [n] : \tilde{y}^0_i = \text{IEn}^i_{sk}(0)$
$\forall i \in [n] : \tilde{y}^1_i = \text{IEn}^i_{sk}(1)$
$\forall i \in [n]$ execute
1-2-OT
$\tilde{y}^0_i, \tilde{y}^1_i$
$y_i \in \{0, 1\}$
$\tilde{y}^{y_i}_i$
$\text{De}(\widetilde{C}, \tilde{x}_1, \ldots, \tilde{y}_n)$
$C(x, y)$

Figure 2.3: High-level procedure of the garbled circuit protocol.

sk and sends them to Bob. Then she encrypts all possible inputs from Bob and transfers the inputs Bob requested to Bob using n-times the 1-out-of-2 oblivious transfer. Bob can now evaluate the circuit under the garbled inputs using the evaluation function and sends the result to Alice.

Figure 2.4 shows an example of how the garbling process on circuits works. Every logic gate gets two secret keys sk^0 and sk^1 per input, which correspond to the input of the 0 and 1-bit. Based on the gate's function, the encryption is then performed such that if the evaluator decrypts a gate with its known keys, it receives as output the key that corresponds to the function of the gate. For example, in the case of the first NAND gate at the top left in figure 2.4, the evaluator should only receive the key sk^0_5 if he knows the keys sk^1_1 and sk^1_2. In addition to the garbled gates, the garbled circuit contains an *output translation table* that assigns the keys at the output of the last gate in the circuit to their bit values. With this table, the evaluator can translate the output keys into their corresponding bit values when evaluating a circuit. The garbling of the inputs runs accordingly and for the circuit from figure 2.4, for example, the following encoding would be a valid garbling: $\text{IEn}_{sk}(1, 1, 0, 0) = (sk^1_1, sk^1_2, sk^0_3, sk^0_4)$.

The evaluator should only learn the correct key per wire to protect the garbler's inputs, and the protocol achieves this via the oblivious transfer. Besides, the evaluator should not be able to distinguish between the keys for the 0 and 1-bit. For this reason, a garbled circuit can only be used with one garbled input combination. Besides, the encryptions per gate must be shuffled. A canonical ordering as in figure 2.4 would expose the key-bit-mapping. To allow the evaluator to still choose the right encryption for its keys from each gate's

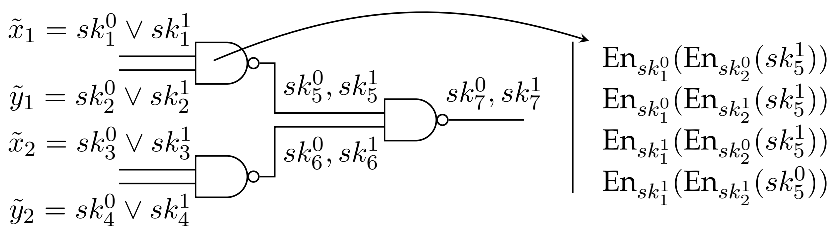

$\tilde{x}_1 = sk_1^0 \vee sk_1^1$
$\tilde{y}_1 = sk_2^0 \vee sk_2^1$
$\tilde{x}_2 = sk_3^0 \vee sk_3^1$
$\tilde{y}_2 = sk_4^0 \vee sk_4^1$
sk_5^0, sk_5^1
sk_6^0, sk_6^1
sk_7^0, sk_7^1
$\text{En}_{sk_1^0}(\text{En}_{sk_2^0}(sk_5^1))$
$\text{En}_{sk_1^0}(\text{En}_{sk_2^1}(sk_5^1))$
$\text{En}_{sk_1^1}(\text{En}_{sk_2^0}(sk_5^1))$
$\text{En}_{sk_1^1}(\text{En}_{sk_2^1}(sk_5^0))$

Figure 2.4: Example of garbling a small circuit of NAND gates.

Table 2.1: GC optimizations with two-input gates. Size is the number of ciphertexts needed per garbled gate (times k bits, with k being the security parameter). The number of ciphertexts per XOR-gate in fleXOR are dependent on the environment in the circuit. H is a key-derivation function (see also 4.3). Table is taken from [ZRE15].

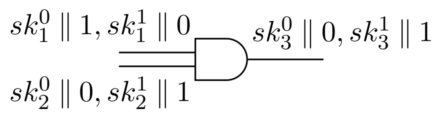
Optimization	Size per Gate		Calls to H per Gate			
			Generator		Evaluator	
	XOR	AND	XOR	AND	XOR	AND
Classical [Yao86]	4	4	4	4	4	4
Point-and-Permute [BMR90]	4	4	4	4	1	1
Free-XOR [KS08]	0	4	0	4	0	1
GRR3 [NPS99] + Free-XOR	0	3	0	4	0	1
GRR2 [PSSW09]	2	2	4	4	1	1
fleXOR [KMR14]	{0,1,2}	2	{0,2,4}	4	{0,1,2}	1
Half Gates [ZRE15]	0	2	0	4	0	2
Garbled Gadgets [BMR16]	2	2	3	3	1	1

shuffled ciphertexts, it must be possible for the evaluator to detect the wrong ciphertexts. Such an extension of ciphers is easily possible. For example, one can concatenate the plaintext with a random string before encrypting and publishing it with the ciphers.

GCs are very well studied, improved since their introduction, and besides the simple protocol described here, they have found many different applications, including previously presented OML schemes. For more details and a formal description of GCs see [BHR12].

2.4 Garbled Circuit Optimizations

Possible optimizations for GCs focus on the size, the computational complexity, and the hardness assumptions taken to give appropriate security guarantees. As described below and shown in table 2.1, most efforts in recent years have focused on optimizing the size of GCs to reduce the communication complexity between the garbler and the evaluator.


$sk_1^0 \parallel 1, sk_1^1 \parallel 0$
$sk_2^0 \parallel 0, sk_2^1 \parallel 1$
$sk_3^0 \parallel 0, sk_3^1 \parallel 1$

$$\begin{array}{|l}
\mathrm{En}_{sk_1^1}(\mathrm{En}_{sk_2^0}(sk_3^0 \parallel 0)) \\
\mathrm{En}_{sk_1^1}(\mathrm{En}_{sk_2^1}(sk_3^1 \parallel 1)) \\
\mathrm{En}_{sk_1^0}(\mathrm{En}_{sk_2^0}(sk_3^0 \parallel 0)) \\
\mathrm{En}_{sk_1^0}(\mathrm{En}_{sk_2^1}(sk_3^0 \parallel 0))
\end{array}$$

Figure 2.5: Garbling with point-and-permute optimization.

2.4.1 Point-and-Permute

In the classical garbling scheme, in the worst case, the evaluator has to decrypt all four ciphertexts per gate to obtain the correct output label. By introducing the *point-and-permute* optimization in 1990, Beaver et al. [BMR90] reduce the number of necessary decryptions to one ciphertext per gate. To do this, point-and-permute appends a pointer pair $(p_i, \overline{p_i})$ with $p_i \in_r \{0,1\}$ to each key pair $(sk_i^0 \parallel p_i, sk_i^1 \parallel \overline{p_i})$ and sorts the ciphertexts of the gate in descending order based on the pointer bits[13] (see figure 2.5 for an example). This creates a canonical ordering of the ciphertexts that allows the evaluator to determine the correct gate output with only one decryption without learning any additional information about the input.

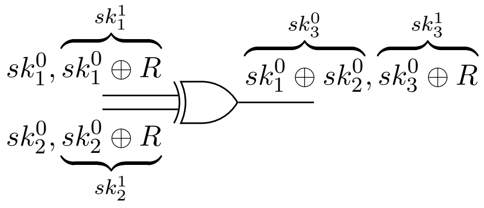
2.4.2 Garbled Row-Reduction 3

In 1999, Naor et al. [NPS99] presented the *garbled row reduction 3* (GRR3) optimization, which reduces the number of necessary ciphertexts per gate to three. In the classical GC scheme, the keys k_i are randomly generated, whereas GRR3 introduces for the first ciphertext per gate (e.g., $sk_3^0 \parallel 1$ in figure 2.5) the condition: $sk_3^0 = (\mathrm{EN}_{sk_1^1}(\mathrm{EN}_{sk_2^0}(0^{k+1})))^{-1}$. The garbler chooses sk_3^0 such that the corresponding ciphertext equals 0^{k+1}. When the evaluator receives an input with a pointer to the first ciphertext, it can conclude that the plaintext must be the zero-string 0^{k+1}. Therefore, only the last three ciphertexts per gate must be transferred to the evaluator.

2.4.3 Free-XOR

Free-XOR was introduced by Kolesnikov and Schneider [KS08] and allows the evaluation of GCs without having to perform cryptographic operations or to transmit ciphertexts for XOR gates[14]. For this the garbler chooses sk_i^0 randomly and $sk_i^1 = sk_i^0 \oplus R$, with $R \in_r \{0,1\}^k$ being a circuit-wide constant. Besides, he sets the output key $sk_3^0 = sk_1^0 \oplus sk_2^0$ (see figure 2.6 for an overview). Now the output of an XOR gate is the XOR of the two

[13]These bits often also called color bits.
[14]The use of Free-XOR is restricted by a patent [KS13].



sk_1^1
$sk_1^0, sk_1^0 \oplus R$
$sk_2^0, sk_2^0 \oplus R$
sk_2^1
sk_3^0
sk_3^1
$sk_1^0 \oplus sk_2^0, sk_3^0 \oplus R$

Figure 2.6: Garbling with Free-XOR optimization.

input keys:

$$\begin{aligned}
&sk_1^0 \oplus sk_2^0 = sk_3^0 \\
&sk_1^0 \oplus sk_2^1 = sk_1^0 \oplus (sk_2^0 \oplus R) = (sk_1^0 \oplus sk_2^0) \oplus R = sk_3^0 \oplus R = sk_3^1 \\
&sk_1^1 \oplus sk_2^0 = (sk_1^0 \oplus R) \oplus sk_2^0 = (sk_1^0 \oplus sk_2^0) \oplus R = sk_3^0 \oplus R = sk_3^1 \\
&sk_1^1 \oplus sk_2^1 = (sk_1^0 \oplus R) \oplus (sk_2^0 \oplus R) = (sk_1^0 \oplus sk_2^0) \oplus (R \oplus R) = sk_1^0 \oplus sk_2^0 = sk_3^0
\end{aligned}$$

Note that Free-XOR introduces a new optimization criterion for GCs. In a circuit consisting of AND, NOT and XOR gates, the number of AND gates should be minimized to obtain an efficient function representation[15].

2.4.4 Half Gates

Garbled row reduction 2 is an efficient optimization for AND gates needing just two ciphertexts per gate, while Free-XOR allows garbling XOR gates without a single ciphertext. Both optimizations are not compatible with each other. Leveraging Free-XOR and the GRR3 approach, the *Half Gates* (HG) optimization introduced by Zahur et al. in 2015 [ZRE15] accomplish to garble AND gates with only two ciphertexts per gate, while XOR gates remain free[16]. The optimization divides an AND gate into two HGs with two ciphertexts each. Then GRR3 allows to remove one ciphertext per HG.

The application of an HG is shown in figure 2.7. Note that additionally, GRR3 must be applied per HG, such that only two ciphertexts are necessary to garble the whole AND gate. The core idea of the HG optimization is to reformulate the AND operation such that

[15]NOT gates are free by incorporating them trough inverted wire-label semantics into their neighboring AND and OR gates.

[16]During the thesis, a new optimization was published which beats the HG optimization's lower bound for communication complexity at the price of a higher computational cost [RR21]. In this optimization, only $1.5k + 5$ (with k being the security parameter) bits are needed for garbling an AND gate. If and to what extent this optimization applies to our use case is an interesting question for future work.

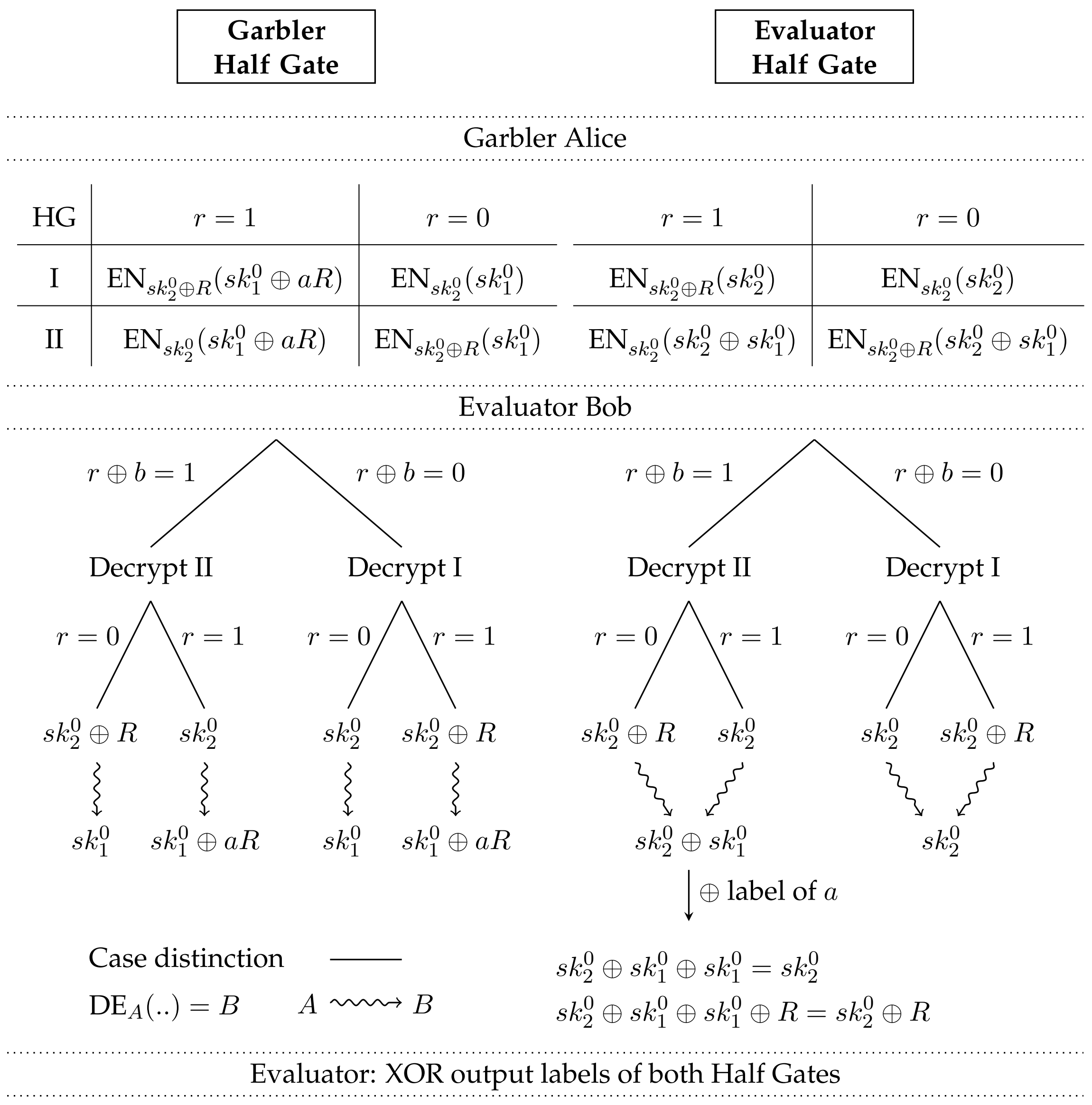
one input is known for each HG:

$$\begin{aligned} a \wedge b &= a \wedge (b \oplus r \oplus r) \\ &= \underbrace{(a \wedge r)}_{\text{Garbler Half Gate}} \oplus \underbrace{(a \wedge (b \oplus r))}_{\text{Evaluator Half Gate}} \end{aligned}$$

In this reformulation r is a bit randomly chosen by the garbler. For efficiency, one of the point-and-permute bits can be used. In the following, r is the point-and-permute bit of sk_2^0. Due to the reformulation, the garbler knows one input of the garbler HG (r), and the evaluator knows one input of the evaluator HG $(b \oplus r)$ - the point-and-permute bit of sk_2^b. This way, the semantics of the garbler HG simplify to an unary gate. If r is false, the output is false, and if r is true, the output is a. The ciphertexts of the HG are computed accordingly (see table of the garbler HG in figure 2.7).
Garbling the evaluator HG is more involved in the sense that if $b \oplus r$ is true the evaluator must be able to introduce the truth value a to the HG output. Therefore the garbler sends two ciphertexts $\text{EN}_{sk_2^0 \oplus R}(sk_2^0)$ and $\text{EN}_{sk_2^0}(sk_2^0 \oplus sk_1^0)$ if $r = 1$ or $\text{EN}_{sk_2^0}(sk_2^0)$ and $\text{EN}_{sk_2^0 \oplus R}(sk_2^0 \oplus sk_1^0)$ if $r = 0$.
The evaluator selects the ciphertexts to be decrypted from the two HGs based on $r \oplus b$ (see bottom of figure 2.7). If $r \oplus b = 1$, the evaluator XORes the output of the evaluator HG with sk_0^a. Finally, the AND gate result is obtained by XORing (via Free-XOR) the two HG outputs.

2.5 Trusted Execution Environments

TEEs such as Intel Software Guard Extensions (SGX)[MAB+13, CD16, Int21], AMD Secure Encrypted Virtualization (SEV) [Kap16], Sanctum [CLD16], and ARM TrustZone [PS19] allow programs, containers, or whole VMs to be executed securely in hardware-sealed *enclaves*. The hardware isolates the enclave from all other programs on the host, regardless of the privilege level or CPU mode, so it is protected even from a compromised operating system.
The key feature of TEEs besides the enclave isolation is *attestation*, a process that guarantees that a specific enclave, specific in the sense of the code and data it contains, has been deployed on a specific system. One distinguishes between local and remote attestation. With local attestation, two enclaves on the same host authenticate each other. In remote attestation, an enclave authenticates itself to a remote third party. This makes TEEs particularly attractive for use in the typical cloud computing offerings like MLaaS. The cloud provider creates an enclave to which the cloud customer establishes a secure connection and authenticates the enclave via remote attestation.



HG	$r = 1$	$r = 0$	$r = 1$	$r = 0$
I	$\text{EN}_{sk_2^0 \oplus R}(sk_1^0 \oplus aR)$	$\text{EN}_{sk_2^0}(sk_1^0)$	$\text{EN}_{sk_2^0 \oplus R}(sk_2^0)$	$\text{EN}_{sk_2^0}(sk_2^0)$
II	$\text{EN}_{sk_2^0}(sk_1^0 \oplus aR)$	$\text{EN}_{sk_2^0 \oplus R}(sk_1^0)$	$\text{EN}_{sk_2^0}(sk_2^0 \oplus sk_1^0)$	$\text{EN}_{sk_2^0 \oplus R}(sk_2^0 \oplus sk_1^0)$

Figure 2.7: The Half Gate optimization procedure. a and b are the truth values on the input wires of the AND gate. r is the point-and-permute bit of sk_2^0. To achieve full efficiency, the HGs must be reduced with GRR3, and the XOR of the HG outputs must be performed via Free-XOR.

Like Slalom, we use Intel SGX over AMD SEV to minimize the trusted computing base. SGX applications must be divided by the developer into a trusted and untrusted component. The trusted component runs inside the enclave and communicates with the untrusted part outside the enclave through an interface defined by the developer. The trusted part of the application and the communication between the two components should be kept small to reduce the potential attack surface and improve performance.
After Intel SGX was introduced in Intel Core CPUs in 2015, Intel's current CPU lineup only supports it in 3rd Gen Intel Xeon Scalable server CPUs. While SGX-enabled CPUs in the past only supported 128MB of *Processor Reserved Memory* (PRM), the current Scalable Xeon CPUs can instantiate enclaves up to 1TB in size. This postpones the paging overhead that occurs when the available memory is exceeded. SGX's enclave memory is encrypted, and a memory management engine decrypts data before loading it into registers.
While SGX provides good protection against typical software-based attacks, its design basically does not consider hardware-based side-channel attacks. Over the years, the research community has demonstrated a variety of different side-channel attacks on SGX [BMD+17, GESM17, LSG+17, WCP+17, SCNS16]. However, these hardware-based attacks are usually much more demanding than typical software vulnerabilities. Therefore, SGX, combined with traditional software-based efforts, can significantly raise the bar for successful attacks.

2.6 Graphical Processing Units

Today, *graphical processing units* (GPUs) are mainly sold by Intel, AMD, and Nvidia and were initially designed for rendering real-time 3D graphics. While *central processing units* (CPUs) enable accelerated sequential execution through branch prediction, speculative execution, and large caches, GPUs are designed for massively parallel execution of many similar operations. This concept is also known as *single-instruction-multiple-data* (SIMD). One operation is applied to multiple data elements simultaneously. In the context of Nvidia's *compute unified device architecture* (CUDA) [NBGS08, HP17], it is often also referred to as *single-instruction-multiple-thread* (SIMT). The omission of complex control flow mechanisms and large caches saves chip space and allows thousands of SIMT cores per GPU. Therefore, GPUs offer significantly larger instruction throughput and memory bandwidth compared to CPUs.

2.6.1 Memory Spaces

From now on, we will look in particular at CUDA-enabled Nvidia GPUs with the so-called CUDA cores. Each GPU has multiple *streaming multiprocessors* (SMs) with multiple CUDA

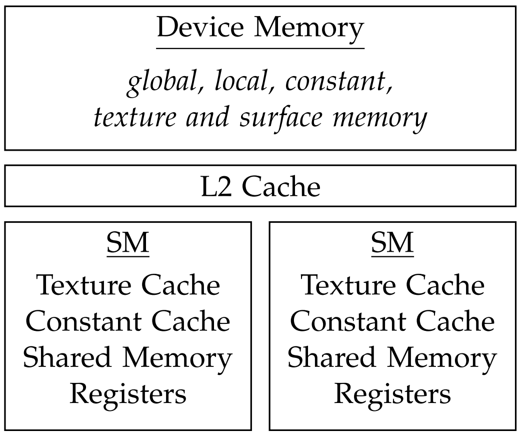


Figure 2.8: Memory hierarchy of CUDA-enabled Nvidia GPUs.

cores. In general, appropriate memory access patterns are the most critical component for the performance of CUDA applications. Therefore, Nvidia's memory hierarchy (see figure 2.8) offers different memory spaces for various access patterns. The *global memory* resides in the *device memory*, which is large but underlies high latency. It is read- and writeable by all threads above all SMs and persistent across kernel launches. A *kernel* is a top-level device function launched by the host and executed by multiple CUDA threads in parallel. In order to achieve the highest possible throughput, the alignment of the data and the *compute capability* (CC) of the GPU must be taken into account in addition to the access pattern when selecting the appropriate memory space (for details, see [Nvi21]). Each GPU has a CC that describes its software and hardware features.

The *local memory* also resides in the device memory and is local to the thread that declares it. Furthermore, the *constant memory* is located in the device memory and is cached SM-wise. Constant memory is read-only within kernels, but the host can write data with the CUDA runtime API. The last memory space in the device memory is the *texture and surface memory*, which is also cached SM-wise. This memory space, or rather its cache, is in particular characterized by spatial 2D-locality.

The *shared memory* is fast on-chip read- and write-memory located in each SM. It is much faster than global and local memory and has much higher bandwidth. To achieve high bandwidth, the shared memory is divided into banks. To prevent bank conflicts, the requests must be scheduled according to the CC of the GPU. For more Nvidia GPU-specific memory features and optimizations, as well as instruction throughput optimizations, see [Nvi21].

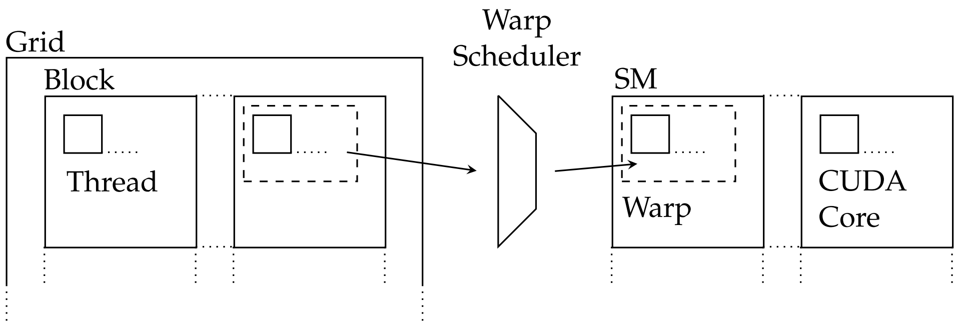


Figure 2.9: Execution model of CUDA-enabled Nvidia GPUs.

2.6.2 General-Purpose Computing

When GPUs are used beyond their original purpose of rendering computer graphics, it is called *general-purpose computing on graphics processing units* (GPGPU). The beginnings of GPGPU were to outsource parallelizable workloads from the CPU to the GPU using graphical primitives. The CUDA toolkit[17] makes it possible to process data without needing a graphical representation via high-performance computing primitives, which are optimized for the Nvidia GPU architecture.

In the typical CUDA workflow, the host generates data. Next, the host copies the data to the device and starts the kernel functions that process the data in parallel with multiple CUDA threads. Usually, one thread processes one data element via one CUDA core. Each thread has a unique identifier for memory-addressing and program flow control. Once the parallelized execution is finished, the results are copied back to the host. Next to explicit functions for host-device data transfers, CUDA also supports the concept of *unified memory*. Unified memory introduces the same view of the memory for the CPU and the GPU and always migrates the data to the processor that needs it.

In CUDA's execution model (see figure 2.9), threads are grouped into so-called *blocks*, and blocks are grouped into so-called *grids*. All blocks in a grid have the same number of threads, and blocks and grids can have up to three dimensions. This partitioning of threads into blocks and grids must be defined by the host when a kernel is started. Threads from the same block share resources such as shared memory and can synchronize and communicate.

CUDA assigns each block to an SM. If there are more blocks than SMs, the remaining blocks must wait. Each block is executed in so-called *warps* of 32 threads. All threads in a warp are executed simultaneously on one SM. The execution context (register values, shared memory, and caches) of a warp remains on the SM for the whole lifetime

[17] A popular open-source alternative to CUDA is OpenCL [SGS10].

of the warp. Therefore the warp scheduler can cost-effectively switch between warps to hide memory latency. When the executions of all threads of a block are finished, the SM becomes free, and CUDA schedules a new block until all blocks of the grid have been executed.
Since all threads in a warp share the same program counter, so-called *warp convergence* arises when threads have different program flows, e.g., due to an if condition. In warp convergence, threads with different program flows are executed sequentially and therefore reduce parallelism and throughput. Besides avoiding warp convergence, the so-called *occupancy* is an essential indicator for utilizing the available CUDA cores. The occupancy is the ratio of active warps to the maximum number of warps that the device supports. Generally, the higher the occupancy, the higher the utilization of the SMs and the higher the throughput. CUDA contains an API for calculating the block size for a maximum occupancy for a given kernel.
Besides the CUDA cores, the current Nvidia GPU generations contain *Tensor cores*, which are particularly useful for accelerating linear algebra workloads. To what extent these special cores can be leveraged for accelerated evaluation of our garbled gates and gadgets is an interesting question for future work [Nvi21].

3 Related Work

Here we summarize important and recent work on OML from a high-level perspective and focus on their key performance aspects regarding the online phase. For each scheme we give an overview, a security analysis, a description of the used methods, and point out the limitations. The input-privacy and model-privacy terms used in the following descriptions refer to privacy in the SMPC setting. Note that without further precautions, an attacker can exploit the prediction services as black box oracles. This enables him to perform model extraction attacks [CJM20, PMG$^+$17, TZJ$^+$16] to extract model parameters and model inversion [FLJ$^+$14, FJR15] or membership inference attacks [SSSS17, LBW$^+$18, NSH19] to learn training data points or their features. As protection against such attacks, the server could limit possible requests or use the differential privacy framework. These countermeasures are not considered in the OML setting discussed here.

3.1 Slalom

Slalom[18] is an OML scheme introduced in 2019 by Tramèr and Boneh [TB19] for verified and private inference on ANNs (with fully connected layers, convolutions, separable convolutions, pooling layers, residual blocks, and activation functions). The scheme is designed for the one-server setting, where the server must feature a CPU with a TEE and an FPU. The one-server setting corresponds to the typical MLaaS case where a remote client wants to outsource expensive ML computations to the cloud. Several schemes for this case which leverage TEEs have been introduced [OSF$^+$16, TGS$^+$18, HSS$^+$18, HZG$^+$18, LLP$^+$19, ZHC$^+$20]. However, Slalom takes a step further and outsources the expensive linear operations of ANN inference from the TEE to an FPU to improve trough-output. To further increase the performance, the protocol is divided into an offline and an online phase. In the offline phase, parts of the integrity check for the computations outsourced to the FPU are pre-computed. If input-privacy with respect to the FPU is required, masking and unmasking bits can also be pre-computed. Slalom's techniques for outsourcing matrix multiplications to an FPU can be easily applied to other ML algorithms.

Security. Slalom is secure in the malicious attacker model in terms of the integrity of the computations and optionally the confidentiality of the inputs under the assumption that

[18]The name comes from how the TEE and FPU exchange messages for each layer.

the security of the TEE cannot be broken. Also, a commit-and-prove scheme can be used to ensure model-privacy concerning the remote client.

Methods. To verify the matrix multiplications outsourced to the FPU, Slalom uses Freivald's algorithm [Fre77]. Thereby Slalom repeatedly reuses the same random vector for the same input samples to minimize the memory usage in the TEE. To achieve optional input-privacy, Slalom uses a simple stream cipher that additively masks all input bits. Compared to the integrity-only approach, the stream encryption generates a significant overhead.

Limitations. Since Slalom requires communication between the TEE and FPU after each layer of the ANN, the communication overhead increases with the depth of the ANN. As described by the authors, if Intel SGX is used, the framework is rendered useless at matrix dimensions greater than 4096 due to SGX's DRAM limitation. This limitation drops with the new Intel Xeon Scalable processors, as already described in section 2.5. Compared to purely cryptographic solutions, the use of TEEs widens the surface for possible attacks.

3.2 SecureML

SecureML is a 2017 OML scheme by Mohassel and Zhang [MZ17] that allows inference and training (including adaptive learning rate decay and early stopping) on LiR, LoR, and ANNs. The scheme is designed within the two-server model. Here, the data owners distribute their private data to two untrusted but non-colluding servers. These two servers then train the desired model using 2-party SMPC techniques. The protocol is divided into an offline phase, in which multiplication triples are generated, and an online phase for training or inference. Next to SecureML exist a more recent variant called ABY^3 [MR18] which works in the three-server model. Patra et al. [PSSY20] optimize the ABY framework used in SecureML and achieve a significant speedup.

Security. SecureML is secure in the semi-honest attacker model under the assumption that the attacker can corrupt any subset of clients, but at most one server. If the clients should be included in the pre-computation of the multiplication triples to accelerate the offline phase, the attacker model changes such that the clients cannot collude with a server. Secure means that the inputs of all data providers remain confidential during and after the computation.

Methods. SecureML replaces the SMPC unfriendly activation functions sigmoid and softmax with more efficient and ML-friendly approximations to speed up inference and learning. While all computations are performed under arithmetic sharing, SecureML uses GCs to speed up the computation of the activation functions. To compute the GCs starting from an arithmetic sharing, SecureML uses a technique from the ABY framework [DSZ15] that allows efficient switching between arithmetic and Yao sharing. For further speedup, SecureML uses vectorization techniques and applies the same multiplication triplets to the same inputs in each epoch.

Limitations. All benchmarks in the publication were performed at Amazon Web Services. In a LAN scenario, both servers were positioned in the same region and in a WAN scenario in different regions. The runtimes of these two scenarios differ considerably. To what extent the LAN setting is realistic is certainly questionable. Once the two servers for SecureML are rented from the same cloud provider and no further security measures are implemented, the assumption that these two servers will not collude is extremely strong or even unrealistic. Even though the authors argue that there are scenarios in which geographically separated servers provide a high-speed link, this assumption greatly limits the scheme's practical utility. In addition, the evaluated ANN is very small (2 hidden layers with 128 neurons each) and larger ANNs would result in a huge overhead, making the framework practically unusable.

3.3 Oblivious Neuronal Networks via MiniONN Transformations

MiniONN[19] is an approach for secure inference on ANNs in the one-server setting (with convolutions, dropout, dropconnect, batch normalization, ReLU, leaky ReLU, maxout, tanh, sigmoid, max, and mean pooling) and was introduced by Liu et al. in 2017 [LJLA17]. The protocol consists of an offline phase for the pre-computation of dot-product triples (similar to beaver triples; see also section 7.3) and an online phase for prediction.

Security. The framework is secure in the semi-honest attacker model under the assumption that either the client or the server is corrupted. It ensures input-privacy w.r.t. the server and model-privacy w.r.t. the client (and calls this property obliviousness).

Methods. Utilizing the pre-computed dot-product triples, the server computes the convolution, dropout, dropconnect, and batch normalization layers (all are commonly implemented solely with matrix operations) without additional communication, so that both parties hold valid secret shares after the computation. To compute the activation functions

[19]The name means minimizing the overhead for oblivious neuronal network transformations and is inspired by a well-known animation movie.

and max pooling layers, the framework uses GCs and the ABY framework to efficiently switch between arithmetic and Yao sharing. To compute smooth activation functions such as sigmoid, instead of using high-dimensional polynomials that require many SMPC unfriendly multiplications, the framework uses interval-wise approximations with multiple low-degree polynomials. For efficient pre-computation of the dot-product triples, a similar technique as commonly used for pre-computation of Beaver Triples is used.

Limitations. Compared to SecureML, the framework is also practical for deeper networks, but compared to Slalom, it does not support FPUs. The evaluation shows that deeper ANNs with seven activation layers or more lead to a large overhead. The authors point out that the prediction accuracy can also increase with increasing model complexity, but saturates above a certain complexity level. They propose an accuracy-overhead tradeoff by considering the number of activation functions used. This tradeoff allows to design deeper models obliviously, but beyond a certain model depth even this approach reaches its limit.

3.4 Faster CryptoNets

Faster CryptoNets [CBL+18] is a scheme for encrypted inference in the one-server setting. According to the authors Chou et al., it is based on the typical MLaaS setting in which users send inputs to a third-party provider that offers a prediction service on its machine learning models. The scheme supports convolution and fully connected layers, as well as activation functions (ReLU, square, Swish, softplus), scaled average pooling, and batch normalization.

Security. The scheme is secure in the malicious attacker model and protects the confidentiality of inputs. The authors refer to this property as oblivious inference.

Methods. The methodology of the scheme follows the two previously proposed CryptoNets [GDL+16, XBF+14] approaches and uses leveled homomorphic encryption (LHE) [RAD+78] to perform ANN-inference over encrypted inputs. Following the requirements of ANN-inference, the encryption scheme supports additive and multiplicative homomorphisms. Since the available arithmetic operations prevent typical activation functions, they leverage efficient quantized polynomial approximations. To reduce the number of expensive multiplications, they prune and quantize the ANN.
The practicable multiplicative depth of the LHE scheme limits the ANN to three layers using activation functions. The authors suggest leveraging transfer learning [Ben12, HS06]

and computing the inference on the pre-trained layers on the client and the inference on the fine-tuned layers encrypted on the server. They call this approach delegated feature extraction (DFE). To also enable fine-tuning on the client part, they proposed to leverage differential private stochastic gradient descent and improve the accuracy of their scheme noticeably.

Limitations. The approximated and quantized activation functions approximate the original activation functions best in the interval [-1,1]. Outside this interval, more significant errors occur. The authors suggest using batch normalization before the activation functions to normalize the inputs to zero mean and unit variance. This works well according to the authors' experiments but limits the model architecture space.
Compared to Slalom, inference times are orders of magnitude slower. The scheme scales relatively poorly, and the DFE approach is interesting but contradicts the MLaaS setting if the goal is to outsource expensive ANN-inference workloads. The authors propose a medical smartphone application as a use case for their system. Whether this is practicable for an inference step with message sizes of several hundred GBs to TBs is questionable.

3.5 Delphi

Delphi [MLS+20] is an OML scheme for outsourced ANN-inference based on the techniques of Gazelle [JVC18], which leverages LHE for linear and GCs for non-linear ANN layers. It works in the typical MLaaS setting in which a user sends his private input to the prediction API of a cloud provider, which classifies this input using its private ANN model.

Security. Delphi is secure in the two-party semi-honest setting, where an adversary corrupts one party. A corrupted client can only learn the output of the inference and the architecture of the ANN. The Planner (see methods) used to replace ReLU activations with approximations leads in an l-layer ANN to a maximal leakage of l-bit training data concerning the client. The authors consider this leakage to be negligible considering the amount of data processed in an ANN. To mitigate this leakage, the authors propose to leverage DP training methods. The server is not able to learn anything about the private inputs of the client.

Methods. Delphi is a mixed-SMPC approach with an offline and an online phase. Compared to Gazelle, Delphi reduces the cost of the linear layers by moving the expensive LHE computations via additive SS to the offline phase. Using a planner based on neural network architecture search [EMH19, WRP19], Delphi replaces suitable ReLU activation functions with approximations while retaining non-approximated ReLUs relevant for accuracy (performance-accuracy tradeoff). Subsequently, hyperparameter optimization is used for the new architecture. Delphi computes ReLU activations via GCs (OT in the offline phase) and ReLU approximations with quadratic polynomials via Beaver Triples (see section 7.3) (precomputed with linear HE in the offline phase). Delphi utilizes standard GPU libraries for the acceleration of linear layers.

Limitations. The use of non-standard activation functions complicates training and limits the space of possible model architectures. The neural network architecture search leads to a significant training overhead. ANNs cannot always be re-trained, for example, the training data may no longer be available, or the training cost may be too high.

4 Garbling Techniques for Sprint

In this chapter, we introduce the garbling techniques for Sprint based on the binary GCs an their optimizations introduced in section 2.3 and section 2.4.

4.1 Garbling Gadgets for Arithmetic Circuits

Implementing arithmetic operations via conventional binary GCs is expensive, especially compared to other SMPC approaches like secret-sharing-based SMPC. Ball, Malkin, and Rosulek [BMR16] introduced garbling gadgets for efficient garbling of arithmetic circuits over large finite fields. Considering our use case, their gadgets allow, in particular, free addition $(\mathrm{mod}\ m)$, free multiplication with a constant $(\mathrm{mod}\ m)$, and efficient projection gates for arbitrary unary functions $(\mathrm{mod}\ m)$. Starting from Free-XOR, Ball et al. consider wire-labels as vectors of components from $\mathbb{Z}_m$. The encoding of a value $x \in \mathbb{Z}_m$ is given through $sk^x = sk^0 + xR_m$, with R_m being a circuit-wide constant vector of random elements from $\mathbb{Z}_m$[20]. The construction considers wires with different moduli, called *mixed moduli circuits*, and leverages different offsets R_m for each module m - but wires of one modulus always share the same offset value. Point-and-permute generalizes by using an element from $\mathbb{Z}_m$ instead of a single bit and choosing $1 \in \mathbb{Z}_m$ as the point-and-permute component of R_m.

A mixed modulus circuit consists of an acyclic structure of wires together with their moduli and gates constructed as follows (we limit the description to gates relevant for our use case):

- **Addition** $(x + y) \bmod m$ (unbounded fan-in with equal modulus):
  $sk_1^x + sk_2^y \equiv (sk_1^0 + sk_2^0) + (x + y)R_m \bmod m$.

- **Multiplication by a public constant** $xc \bmod m$ (unary gate):
  $c \cdot sk^x \equiv c \cdot sk^0 + cxR_m \bmod m$ with c being coprime to modulus m (needed for technical reasons in the security proof, see also [BMR16]).

- **Unary projection gate for arbitrary functions** $\varphi : \mathbb{Z}_m \to \mathbb{Z}_n$:
  A garbled projection gate consists of m ciphertexts of the form $\mathrm{En}_{sk^x}(sk^{\varphi(x)})$. Leveraging GRR3, one ciphertext can be removed.

[20]This Free-XOR generalization together with the free addition gate was shown before, e.g. in [MPS15].

While the first two gate types are free, the projection gate is not practical for larger moduli m. BMR, therefore, propose to use a *composite primal modulus* (CPM) $P_k = 2 \cdot 3 \cdot \ldots \cdot p_k$, the product of the first k primes and to leverage the Chinese remainder theorem to represent the wire values in a *residue representation*[21]:

$$[\![x]\!]_{\text{crt}} = ([x]_2, [x]_3, \ldots, [x]_{p_k}), \text{ where } [x]_m \text{ represents } x \bmod m.$$

From the circuit perspective, every value of a conventional wire is now represented by a bundle of wires, with each of the wires in a bundle applying one factor of the CPM. BMR show how to transfer various operations efficiently into the domain of the residue representation. We will return to the residue representation in the next section and show operations that capitalize on it. At this point, we are only interested in how the addition and multiplication operations described above remain free:

- **Addition** $\mathrm{mod}\ m$:

$$\begin{aligned} [\![x]\!]_{\text{crt}} + [\![y]\!]_{\text{crt}} &= ([x]_2, [x]_3, \ldots, [x]_{pk}) + ([y]_2, [y]_3, \ldots, [y]_{pk}) \\ &= ([x]_2 + [y]_2, [x]_3 + [y]_3, \ldots, [x]_{pk} + [y]_{pk}) \\ &= [\![x + y]\!]_{\text{crt}} \end{aligned}$$

- **Multiplication by a public constant** $c \bmod m$: $c[\![x]\!]_{\text{crt}} = (c[x]_2, c[x]_3, \ldots, c[x]_{pk})$, for $c \neq 0$. For $c = 0$, BMR include one global (the same for each modulus) zero-wire to represent the zero-product.

4.2 Garbled Neuronal Networks

By introducing new cryptographic and ANN specific GC optimizations in an also implementation-wise heavily tweaked library called fancy-garbling, Ball et al. demonstrate the practicability of garbled neuronal network inference [BCM+19]. Via a new *mixed-modulus multiplication gate* and a fast *approximated sign function*, their optimizations especially target

- the ReLU activation function ($\mathsf{ReLU}(x) = \mathsf{sgn}(x) \cdot x$),
- and the max-pooling operation ($\mathsf{max}(x, y) = x + \mathsf{ReLU}(y - x)$).

Here, we describe the GC optimizations, followed by a brief description of fancy-garbling's implementation-level optimizations.

[21]Leveraging CRT-representations for optimizations in GCs was proposed before, e.g. in [AIK11].

4.2.1 Half-Gate Generalization

Starting from the BMR scheme for arithmetic circuits, Ball et al. [BCM+19] introduce a mixed-modulus multiplication gate by generalizing the HG optimization for AND gates in binary GCs. Before we describe the mixed-modulus HG, we introduce the HG generalization of Malkin et al. [MPS15].

As in the BMR scheme, the wire-labels are vectors consisting of elements from $\mathbb{Z}_p$, and the wire-label encoding of x is given by $sk^0 + xR$. Also, the free addition $\bmod p$ gate and the point-and-permute generalization are the same. Like in the HG optimization, we denote the point-and-permute value of sk_2^0 as r, and the point-and-permute value of sk_2^y is given by $(y + r) \bmod p$. Similar to the HG optimization, the key idea is to reformulate the multiplication such that the garbler and the evaluator each know one input to the generalized HGs:

$$x \cdot y \equiv x \cdot (y + r - r) \equiv (\underbrace{x \cdot (y + r)}_{\text{Evaluator HG}} - \underbrace{x \cdot r}_{\text{Garbler HG}}) \bmod p$$

Since r is the point-and-permute value of the zero-label of the y-wire, the garbler knows r at garbling time, and the evaluator learns $y + r$ at evaluation time. That way, the garbler HG can be garbled using an unary projection gate $(x \mapsto rx)$ with $p - 1$ ciphertexts. Concretely, the garbler selects a random zero-label sk_3^0 for the garbler HG and performs encryptions for all possible input values $\boldsymbol{x} \in \mathbb{Z}_p$:

$$\mathrm{En}_{sk_1^0 + \boldsymbol{x}R}(sk_3^0 + \boldsymbol{x}rR)$$

For the evaluator HG, the garbler selects a random zero-label sk_4^0 and performs encryptions for all possible input values $(\boldsymbol{y} + \boldsymbol{r}) \in \mathbb{Z}_p$ (results in $p - 1$ ciphertexts):

$$\mathrm{EN}_{sk_2^0 + (\boldsymbol{y}+\boldsymbol{r})R}(sk_4^0 - (\boldsymbol{y} + \boldsymbol{r}) \cdot sk_1^0)$$

The evaluator knows the labels of both inputs and decrypts two of the above ciphers accordingly (leveraging point-and-permute). Knowing $y + r$ and $sk_1^0 + xR$, he can induce the semantics of the input x into the output of the evaluator HG (similar to the evaluator HG in the binary domain):

$$sk_4^0 - (y + r)sk_1^0 + (y + r)(sk_1^0 + xR) = sk_4^0 + (y + r)xR$$

Considering $sk_3^0 - sk_4^0$ as the zero-label of the generalized HG, the product-output can be derived via a single free subtraction:

$$sk_4^0 + (y+r)xR - (sk_3^0 + xrR) = sk_4^0 - sk_3^0 + yxR$$

4.2.2 Mixed-Modulus Half-Gate

To leverage the generalized HG for the ReLU activation function, we must cast the $\mathsf{sgn}(x)$-bit via a projection gate to a value from $\mathbb{Z}_{p_i}$. Garbling a cast operation $\mathbb{Z}_q \to \mathbb{Z}_{p_i}$ (with $q = 2$ in our use case), results in $q - 1$ ciphertexts. Together with the generalized HG, garbling the multiplication $\mathsf{sgn}(x) \cdot x$, results in $(q-1) + (2p-2)$ ciphertexts. By introducing a new mixed modulus HG, Ball et al. [BCM+19] show how the multiplication of two wire-values $x \in \mathbb{Z}_{p_i}$ and $y \in \mathbb{Z}_q$ with different moduli can be garbled with roughly $q+p-1$ ciphertexts. The garbling process of the garbler HG remains unchanged, except that we chose $r \in \mathbb{Z}_{p_i}$ to be the color-value of the zero-label of x, respectively the color-value of sk_1^0 (this slightly differs from Ball et al., who introduce a new random value called "virtual wire"). To encrypt the evaluator HG, the garbler uses an input-label $sk_2^0 + yR'$, with sk_2^0 being a vector of $\mathbb{Z}_q$-elements and an offset value $R' \in \mathbb{Z}_q$. Instead of $p-1$ ciphertexts, the garbled evaluator HG results now in $q - 1$ ciphertexts. Since $y \in \mathbb{Z}_q$, also the corresponding color-value is from $\mathbb{Z}_q$. To leverage the same trick as above and split the multiplication into two HGs with one known input each (namely the color-value of the associated wire), we have to preserve the known input $a = y + r$ (with $a \in \mathbb{Z}_p$) of the evaluator HG. Therefore Ball et al. propose to leverage a single projection gate of $q-1$ really short ciphertexts, encrypting a for all possible y. In the case of the multiplication $\mathsf{sgn}(x) \cdot x$, we are able to pack these short ciphertexts in 128-Bit, the length of a "usual" ciphertext. Hence, the total cost of a mixed-modulus HG is $q + p - 1$. More concretely, garbling the evaluator HG results for every possible $\boldsymbol{y} \in \mathbb{Z}_q$ in the following ciphertexts:

$$\mathrm{EN}_{sk_2^0 + \boldsymbol{y}R'}(sk_4^0 - (r + \boldsymbol{y}) \cdot sk_1^0)$$
$$\mathrm{EN}_{sk_2^0 + \boldsymbol{y}R'}(r + y)$$

4.2.3 Mixed-Radix Addition

For use in the approximated garbled sign function (see below), Ball et al. [BCM+19] introduced a fast *mixed-radix addition*. Consider the summation of $k = 3$ values represented in mixed-radix representation $\mathbb{Z}_{D_n} \cong (\mathbb{Z}_{d_1} \times \ldots \times \mathbb{Z}_{d_n})$ (associated with the integers $\{0, \ldots, D_n - 1 = (\prod_i d_i) - 1\}$ and d_1 being the most significant digit). The operation proceeds from the least to the most significant digit as follows:

1. Compute the sum $s = x + y + z + c_i^{\mathtt{in}}$, where x, y, z are the values of the $\mathbb{Z}_{d_i}$-wires,

respectively the digits of the summands in mixed-radix representation and $c_i^{\mathtt{in}}$ the carry-input. For the least significant digit with $c_i^{\mathtt{in}} = 0$, leverage D_n circuit-wide constant zero-wires, one for each possible wire modulus.

2. Cast $x, y, z, c_i^{\mathtt{in}}$ using four projection gates to $\mathbb{Z}_{3d_i + c_i^{\mathtt{max}} - 1}$, where $c_i^{\mathtt{max}}$ describes the maximal possible value of $c_i^{\mathtt{in}}$.

3. Add all input values $\bmod 3d_i + c_i^{\mathtt{max}} - 1$.

4. Compute the carry-out (the carry-in of the next digit) $c_{\mathtt{out}} = \left\lfloor \frac{x+y+z+c_i^{\mathtt{in}}}{d_i} \right\rfloor$ using an unary gate. Note that the carry-out modulus must match the modulus of the next digit's sum computation in step 1.

Leveraging the addition gadget of the BMR scheme, both additions are free. The cast operations via projection gates cost $4(d_i - 1)$ and the unary gate in the last step $(3d_i + c_i^{\mathtt{max}} - 2)$ ciphertexts per digit $\mathbb{Z}_{d_i}$. This procedure generalizes to arbitrary many summands.
Since the sign function requires just the most significant digit of the sum, we do not need the final carry-out. The costs for computing the most significant digit of a k summands mixed-radix addition are as follows. For all but the most significant digit, the cast operation (step 2) costs $k(d-1) + c_i^{\mathtt{max}}$ ciphertexts. The carry-out (step 4) is computed via an unary gate resulting in $k(d-1) + c_i^{\mathtt{max}}$ ciphertexts. The additions in steps 1 and 3 are free. Thus, the overall cost is bound by:

$$\left[2k \sum_{i=2}^{n} (d_i - 1) + 2 \sum_{i=2}^{n} c_i^{max}\right] \leq \left[2k \sum_{i=2}^{n} (d_i - 1) + 2n(k-1)\right]$$

4.2.4 Approximated Garbled Sign

The garbled sign function $\mathsf{sgn} : \mathbb{Z}_{P_k} \to \{0, 1\}$ of Ball et al. [BCM+19] expects $\mathbb{Z}_{P_k}$-values and interprets the first half of the ring as negative and the other half as positive numbers:

$$\mathsf{sgn}(x) = \begin{cases} 0 \text{ if } x < P_k/2 \\ 1 \text{ if } x \geq P_k/2 \end{cases}$$

The concept of the construction[22] is based on the Chinese remainder theorem, which describes the reconstruction of the value $x \in P_k$ from its residue representation $[\![x]\!]_{\mathsf{crt}} =$

[22]This general approach also appears in earlier works like [HP94].

$(x_1, \ldots, x_k)$:

$$\begin{aligned} x &\equiv \sum_{i=1}^{k} A_i^{-1} \cdot A_i \cdot x_i \bmod P_k, \text{ with } A_i = \frac{P_k}{p_i} \\ &\equiv \sum_{i=1}^{k} \alpha_i \cdot x_i \bmod P_k \end{aligned}$$

For some integer q, we can write:

$$\begin{aligned} &\left[x = q \cdot P_k + \sum_{i=1}^{k} \alpha_i x_i \right] \\ \iff &\left[\frac{x}{P_k} = q + \sum_{i=1}^{n} \frac{\alpha_i x_i}{P_k} \right] \\ \implies &\left[\text{fractional part of } \frac{x}{P_k} = \text{fractional part of } \sum_{i=1}^{n} \frac{\alpha_i x_i}{P_k} \right] \end{aligned}$$

Hence, the sign function can be computed just regarding the fractional part of the last summand:

$$\begin{aligned} \mathsf{sgn}(x) = 1 &\iff x \geq P_k/2 \\ &\iff \frac{x}{P_k} \geq 1/2 \\ &\iff \text{fractional part of } \sum_{i=1}^{n} \frac{\alpha_i x_i}{P_k} \geq 1/2 \end{aligned}$$

This concept leads to the computation of the sign operation:

1. Compute $\alpha_i x_i / P_k$ for all residues x_i and round to $1/M$, with M being a discretization level. This fixed-point approximation d/M can be represented as $\mathbb{Z}_m$-wire value (as described later, M is represented in the mixed-radix system using a bundle of wires instead of a single $\mathbb{Z}_m$-wire). d can be pre-computed as lookup table.

2. Add all numerators of the fixed-point approximation $\bmod M$ (approximation of the fractional part of $\sum_{i=1}^{n} \alpha_i x_i / P_k$).

3. Compare the result to $M/2$.

The value d/M approximates $\alpha_i x_i / P_k$ within an error margin of $1/2M$. Therefore, the error of a k term sum is limited by $k/2M$, and if $k/2M < 1/P_k$, the sign computation is correct. This means $M > kP_k/2$ guarantees correct results. But note, that smaller values

of M improve the computational efficiency and can also be correct. Ball et al. [BCM$^+$19] (see figure 1 in their publication) performed an exhaustive search for $k \leq 11$ primes for three different approaches of choosing M and its mixed-radix representation. Compared to the exact BMR sign gadget, their approach is slightly more efficient for $k \leq 11$.
The above sign function becomes interesting in use cases, such as ANNs, where full correctness is not required. In these situations, M can be seen as a trade-off parameter between the function precision and the garbling cost. Ball et al. [BCM$^+$19] (see figure 2 in their publication) showed that even a minimal loss in precision could lead to significant efficiency enhancements.
Now, we describe the garbled sign function for values represented in a mixed-radix representation $\mathbb{Z}_{m_1} \times \ldots \times \mathbb{Z}_{m_t}$, with $\prod_{i=1}^{t} m_i = M$:

1. Approximate x_i by $d \in \mathbb{Z}_M$ using t projection gates per prime p_i and one for each digit m_i of the mixed-radix representation. This results in $t \sum_{i=1}^{k}(p_i - 1)$ ciphertexts.

2. Add all k values represented in mixed-radix representation, using the mixed-radix addition from above. The summation results in maximal $2k \sum_{i=2}^{t}(m_i - 1) + (k-1)^2$ ciphertexts.

3. Compare the sum against $M/2$ by checking whether the most significant digit is greater or equal to $m_1/2$. The comparison is made via a single projection gate resulting in $m_1 - 1$ ciphertexts. To reduce the comparison to the most significant digit requires m_1 to be even.

To balance the garbling cost across the three steps, Ball et al. [BCM$^+$19] suggest to chose $M = m_1 \cdot \ldots \cdot m_t$, with m_1 being relatively large (larger than 50) and $m_2, \ldots, m_t$ being relatively small.

4.2.5 Fancy-Garbling

Since GCs representing ANNs are very large, potentially larger than the memory available, fancy-garbling implements *circuit-streaming*. The garbler and the evaluator form a pipeline and immediately execute computations on the level of addition, multiplication, and projection gates without buffering. Besides, fancy-garbling parallelizes the GC evaluation using an additional postman thread (the technique is not described in detail, but multiple threads compute one ANN layer). All values of the ANN are encoded in CRT representation and with the minimal number of prime residues necessary to preserve the accuracy of the ANN.
In the case of public weights, the scalar multiplication of input values with the weights is free using the BMR scheme. In the case of private weights, the scalar multiplication is per-

formed via projection gates, resulting in $d_i - 1$ ciphertexts. Fancy-garbling is implemented in Rust and available as open source[23].

4.3 Efficient Garbling with Random Permutations via Fixed-Key AES

Most optimizations described so far focus on the communication complexity of GC-based protocols. A line of work also introduces computational improvements for the circuit garbling and evaluation [NPS99, LPS08, HEKM11, KSS12]. The state-of-the-art was presented by Bellare et al. [BHKR13]. They propose to encrypt the wire-labels using a cryptographic permutation instantiated by fixed-key AES_c, with c being a fixed and public key. By instantiating AES with a fixed key, the scheme must perform just a single key-derivation for the whole circuit compared to one key-derivation per label in [KSS12]. While skipping the key-derivation heavily improves computation times, it comes at the cost of introducing non-standard assumptions about AES to enable a security-proof in the random-permutation model (for a further discussion see [GLNP15, GKWY20]).
At the same time, Bellare et al. [BHKR13] presented JustGarble, an implementation of a highly optimized general-purpose garbling tool. To instantiate the fixed-key AES approach described above, they used hardware-accelerated AES NI instructions together with SEE4 to access 128-bit registers for label manipulation. In this implementation, using the permutation instead of a block cipher, namely AES256 as in [KSS12], leads to a 2.5-fold improvement during evaluation time and a 3-fold improvement during garbling time. Using the permutation instead of a cryptographic hash function (SHA1 as in [HEKM11]) leads to a 6.7-times improvement in the evaluation and a 10-times improvement during garbling.
While the speedup introduced by using the permutation is significant, JustGarble demonstrates the importance of common implementation-level optimizations and a sleek circuit representation - using indices and arrays instead of objects. Even in the disadvantageous case of circuits larger than the cache size, JustGarble achieves a 70-times speedup against [KSS12].

[23]https://github.com/GaloisInc/fancy-garbling

5 Sprint: Secure and Fast Outsourced Machine Learning

This chapter describes our approach Sprint for the one server setting (the same as in Slalom [TB19]) based on the work of Ball et al. [BMR16, BCM$^+$19] introduced in sections 4.1 and 4.2. The description also mentions outsourcing the GC evaluation to a GPU, while the implementation does not include this feature. Our system allows a data owner to outsource secure ANN-inference to a remote server controlled by a third party. Therefore, we garble the ANN within a TEE on the remote server and outsource the evaluation or the actual inference step to a co-located FPU. To keep the size of the projection gates of our garbled ANN in practical dimensions, we follow the approach of Ball et al. and transfer the inputs into the CRT space. In the following, we describe the garbling procedure step by step from the perspective of the individual residues of the CRT representation, the CRT encoded inputs, and the ANN operations. We stick to the notation introduced earlier and avoid repetitive descriptions. Finally, we describe our CPU-based implementation and the challenges of outsourcing the GC evaluation to a GPU.

5.1 Residue Level

We initialize the garbling procedure (see algorithm 1) by creating random offset vectors R_{m_i} for all possible wire-moduli $m_1, \ldots, m_k$ in our circuit. The last component of the offset vector serves as the color value and is a constant one. For all circuit inputs, we create random input wire labels with matching modulus. Again, the last component serves as a color value but now is chosen randomly to hide the input semantics in the point-and-permute manner. The number of components of our labels results from the corresponding wire modulus, such that each wire label contributes 128 bits of entropy to the AES-based fixed-key permutation.

Algorithm 1: Initialize

```
  Result: $R_{m_1}, \ldots, R_{m_k}, sk_1^0, \ldots, sk_n^0$
1 for all $k$ possible wire-moduli $m_i$ do
2     $c \leftarrow \left\lfloor \frac{128}{\log_2 m_i} \right\rfloor$            // Number of components per label
3     $R_{m_i} \xleftarrow{r} \mathbb{Z}_{m_i}^{c-1} \| 1$       // Last label-component serves as color value
4 for all $n$ circuit inputs $i$ with modulus $m$ do
5     $sk_i^0 \xleftarrow{r} \mathbb{Z}_m^c$
```

The garbling process (see algorithm 2) follows the topology of the ANN from the input- to the output layer. The garbling of the individual gate types proceeds analogously to the method of Ball et al. introduced in section 4.1. The garbling results include the ciphertexts of the projection gates and the zero-input wire labels sk^0 of all gates. While the ciphertexts are transferred to the FPU for the GC evaluation, the zero-input wire labels stay inside the TEE and are required to generate the decoding information and encode the inputs.

Algorithm 2: Garble

```
   Result: $\tilde{C}$, $sk_i^0$ for all gate ids
1  for all gates in circuit-wise topological order with output id $i$ do
2      $m \leftarrow$ input-modulus of the gate
3      $n \leftarrow$ output-modulus of the gate
4      $J \leftarrow$ indices of all gate inputs
5      switch gate type do
           // Gate operations are performed component-wise
6          case addition do
7              $sk_i^0 \leftarrow \sum_{j \in J} sk_j^0 \mod m$
8          case multiplication-by-constant $w$ do
9              $sk_i^0 \leftarrow w \cdot sk_{J_0}^0 \mod m$
10         case unary projection gate with functionality $\varphi : \mathbb{Z}_m \mapsto \mathbb{Z}_n$ do
11             $r \leftarrow$ last component of $R_m$
12             $sk_i^0 \leftarrow 0$
13             for all possible input values $0 \leq x < m$ do
14                 $C_{r+x} \leftarrow \mathrm{EN}(sk_{J_0}^0 + x \cdot R_m \mod m) \oplus (sk_i^0 + \varphi(x) \cdot R_n \mod n)$
15             $C^i \leftarrow \{C_0, \ldots, C_{m-1}\}$
16 $\tilde{C} = \{C^i \mid$ for all projection gates with output-id $i\}$
```

We generate the decoding information (see algorithm 3) for all circuit outputs and possible circuit output values based on the zero-wire label inputs of the circuit output gates. The initialization, the garbling of the ANN, and the generation of the decoding information are performed offline.

Algorithm 3: Generate Decoding Information

```
   Result: $\tilde{D}$
1  for all circuit-output ids $i$ do
2      $m \leftarrow$ wire modulus of $i$
3      for all possible output values $0 \leq x < m$ do
4          $D_x^i \leftarrow \mathrm{EN}(sk_i^0 + x \cdot R_m)$
5      $D^i \leftarrow \{D_0^i, \ldots, D_{m-1}^i\}$
6  $\tilde{D} = \{D^i \mid$ for all circuit-output-ids $i\}$
```

The online phase starts as soon as an input for the ANN arrives in the TEE. First, Sprint encodes the inputs (see algorithm 4) using the zero-input wire labels of the circuit inputs and transfers them to the FPU. The FPU evaluates the garbled inputs on the circuit (see

Algorithm 4: Encode Inputs

```
Result: $\tilde{X}$
1 for all circuit-input ids $i$ and input values $x$ with modulus $m$ do
2     $X^i \leftarrow sk_i^0 + x \cdot R_m$
3 $\tilde{X} \leftarrow \{X^i \mid$ for all circuit input ids $i\}$
```

algorithm 5) already obtained during the offline phase. The evaluation also runs in the same way as described in section 4.1. After the evaluation is complete, the FPU sends the garbled outputs back to the TEE. In the final step of the inference, the TEE decodes the

Algorithm 5: Evaluate

```
Result: $\tilde{Y}$
1 for all gates in circuit-wise topological order with output ids $i$ do
2     $n \leftarrow$ output-modulus of the gate
3     $J \leftarrow$ indices of all gate inputs
4     switch gate type do
5         case addition do
6             $sk_i \leftarrow \sum_{j \in J} X_j \mod n$
7         case multiplication-by-constant $w$ do
8             $sk_i \leftarrow w \cdot X_{J_0} \mod n$
9         case unary projection gate do
10            $r \leftarrow$ last component of $X_{J_0}$
11            $sk_i \leftarrow C_r^i \oplus \text{EN}(X_{J_0})$
12 $\tilde{Y} \leftarrow \{sk_i \mid$ for circuit-output ids $i\}$
```

garbled outputs (see algorithm 6) and sends them back to the data owner over a secure channel.

Algorithm 6: Decode Outputs

```
Result: $Y$
1 for all circuit-outputs ids i do
2     $n \leftarrow$ modulus of the circuit-output
3     for all possible output values $0 \leq y < n$ do
4         if $EN(sk_i) = D_x^i$ then
5             $y_i \leftarrow x$
6 $Y \leftarrow \{y_i \mid$ for all circuit-output ids $i\}$
```

5.2 CRT Level

The CRT level perspective on Sprint is straightforward and mainly described in section 4.1. Essentially, we transition all inputs to their CRT representation and proceed for each residue as described in the previous section. When the data owner receives the garbled outputs, he converts the CRT representation to its initial form using the Chinese remainder theorem. To prevent overflows, we choose the minimum CPM for the CRT representation that can represent all values in the circuit.

5.3 ANN Level

While most ML algorithms use decimal arithmetics, the most common SMPC and OML protocols and thus the GCs in Sprint operate over integers. Therefore, we quantize the weights, biases, and inputs via division by a small quantisation constant and rounding to the nearest integer. Subsequently, we map all positive integers, including the zero, to the lower half of the finite ring $\mathbb{Z}_{P_k}$ encoding our inputs. We map all negative numbers to the second half of the ring. For example, the inputs-sequence $\langle 0, 1, \ldots, 4, -5, -4, \ldots, -1 \rangle$ would be mapped to $\langle 0, 1, \ldots, 9 \rangle$ for $P_k = 10$.
Based on the CRT level, Sprint leverages the approximated garbled sign gadget of Ball et al. to garble the sign and ReLU activation functions. Moreover, as shown in section 4.2, the ReLU function can be used to garble a max-pooling layer. To maintain the encapsulation of the abstraction levels, Sprint garbles these operations with inputs over all residual classes of the CRT representation in separate GCs. During the evaluation, Sprint interrupts when such an operation occurs, transfers the output from the residue level to the GC of the operation, and transfers the operations's output back to the residue level. To correctly represent the semantic of our encoding in the sign function, we modify the final comparison of the sum x of the mixed-radix addition with the most significant mixed-radix base m_1

$$\mathsf{sign}(x) = \begin{cases} 1 & \text{if } x < \frac{m_1}{2} \\ 0 & \text{else.} \end{cases}$$

Besides these ANN-specific gadgets, which cannot be garbled at the CRT level due to their non-linearity, Sprint can implement operations such as vector-matrix multiplication at the CRT level, for example, to abstract them at the ANN level as a dense layer. Using the bias trick, the dense layer and the convolution operation can be implemented as a garbled matrix multiplication on the ANN level.

5.4 Security

Sprint can operate in two modes. In the more efficient variant, the ANN weights are public, and their multiplication by the input is free. In this mode, Sprint is secure in the honest-but-curious attacker model and protects the confidentiality of the input and the result of the inference. The confidentiality of the input follows directly from the oblivious property of GCs [BMR16]. Given the GC and garbled inputs, an attacker is unable to learn anything about the plain input. Moreover, without the decoding information, nothing can be learned about the plain output.
In the second mode, the weights are secret and are modeled with projection gates[24]. In this mode, Sprint, like Slalom, is secure in the malicious attacker model and protects the confidentiality of the inputs as well as outputs and the integrity of the computations. Unlike the typical GC protocol that proceeds between a garbler and an evaluator, which both contribute inputs, in our case, only the garbler introduces inputs to the computation. An attack in which the garbler reveals the inputs of the evaluator through a manipulated circuit is therefore not relevant. Together with the oblivious property of GCs follows the confidentiality of the inputs. The integrity of the computations follows directly from the authenticity property of GCs [BMR16]. An attacker cannot generate a manipulated garbled output whose decoding yields a valid plain output given the GC and garbled inputs.

5.5 Implementation

For the implementation of Sprint, we follow JustGarble (see section 4.3) and use hardware-accelerated AES NI instructions. We use arrays and indices exclusively instead of objects to avoid the associated administrative overhead and fully implement Sprint in pure C. For high-quality randomness in the wire label generation, we use Intel's Digital Random Number Generator (DRNG). We implemented Sprint in the three abstraction levels described above to facilitate maintenance, extension, and debugging. For the determination of the maximal intermediate value in the inference process and further debugging purposes, we implemented a non-garbled Sprint version. We use JSON to transfer the ANN architecture, including the weights, biases, shapes, layer-types, and activation functions from the TensorFlow model to Sprint.
To send the wire labels made of $\mathbb{Z}_p$ elements to the AES-based fixed-key permutation, they have to be packed into 128-bit strings. Like Ball et al. [BMR16], we use the Horner method for packing. Therefore, we add component by component each value of the label

[24]Note that it is also possible to model the private weights using garbled inputs, but at a higher cost. Compared to the $p-1$ ciphertexts per projection gate, in this case a multiplication with $2p-2$ ciphertexts is needed [BCM+19].

to the packed representation and multiply for each component by the wire modulus. For unpacking, we implemented the optimized method of Ball et al. [BMR16] based on a lookup table. However, for CRT representations of our sizes ($p_k \leq 15$), we did not see any improvement over the naive approach of dividing stepwise for each wire-label with the wire modulus. Therefore, we choose the naive unpacking method for Sprint.

5.6 GPGPU Specifics

Efficient memory management is typically one of the most critical requirements for efficient GPGPU applications. The arithmetic circuits in Sprint are neither strictly hierarchical nor uniform due to the global circuit wires and the modulus-based adaptive label length. This makes memory coalescing difficult since the labels of the gates executed in parallel are not necessarily consecutively arranged in the global memory. To what extent this affects the instruction throughput of a GPU port has to be evaluated.
In general, arithmetic garbled circuits are not suitable for the evaluation on CUDA-enabled GPUs since they do fit not into the concept of single-instruction-multiple-data and the parallel evaluation of different gates causes warp convergence. ANNs or CNNs typically have homogeneous layers consisting of many similar operations. Therefore, the particular case of garbled ANNs with their layered structure of homogeneous gates and layer-wise data dependencies fits CUDA's execution model. To schedule the gates accordingly, the garbled ANN should be represented by a circuit with suitable layer information. Depending on the size of the garbled ANN and the utilization of the GPU, batch-wise inference can be realized by evaluating multiple circuits in parallel.
While current CPUs provide the AES NI instruction set for hardware-accelerated AES, no comparable primitives exist on NVIDIA GPUs. A series of works investigates software optimized implementations of AES on GPUs [LPK16, WC19, NAI17, MCXS17]. These works focus on the parallel processing of multiple plaintext- or ciphertext blocks. However, in our use case as a fixed-key permutation in an arithmetic GC, we only require the encryption of a single label or plaintext block per gate. Using the above software-optimized AES implementations warp-wise could be an approach to still benefit from their optimizations.

6 Performance and Accuracy Evaluation

In this chapter, we evaluate our implementation and compare it with the work of Ball et al. [BCM^{+}19][25]. We run our experiments on a machine with an Intel Core i7-11700 CPU and a base clock of 2.5GHz. We train our ANNs using TensorFlow [AAB^{+}15] with Keras [C^{+}15] on the MNIST dataset [Den12]. This dataset consists of 70.000 black and white images of handwritten digits from zero to nine. 60.000 images serve as training data, and 10.000 images serve as test data. Each image is represented by 28×28 integer pixel values from the range [0,255].
We trained all models over ten epochs using the Adam Optimizer [KB15], and the loss function Sparse Categorical Crossentropy. We quantize all weights and biases by dividing by a *quantization constant* α reported below and then rounding to the nearest integer value. For future comparisons, we document the number of clock cycles besides the pure runtime in seconds. Otherwise, fair comparisons using different hardware are not possible.

6.1 CRT Representation and the Quantization Constant

To transfer a trained ANN into its GC representation, its weights and biases must be quantized to integer values. Depending on the size of the target representation, some learned information will be lost, and the predictive power of the ANN will suffer. The smaller the target representation of the ANN becomes, the more the range of the intermediate values computed during the inference (see also figure 6.1 and table 6.1) decreases. As this range decreases, a smaller CPM can be chosen and the inference computation becomes lighter. In this way, the quantization constant forms a trade-off between a garbled ANN's performance and prediction capability.
We evaluate the influence of the quantization constant on Model-A, a simple ANN consisting of a single dense layer with ten neurons. As shown in table 6.1, three CPMs, P_6, P_7, and P_8 can be considered for garbling Model-A. The most accurate model is obtained by $\alpha = 10.0$ and P_7. Compared to the plain Model-A with an accuracy of 92.82%, the garbled Model-A loses less than 2% accuracy. Larger quantization constants and CPMs do not promise higher accuracy. The performance of the model can only be increased with a quantization constant of 1.7. In this way, the offline phase and the online phase are accel-

[25]Note that the authors also evaluated their implementation on a single CPU without an oblivious transfer and corresponding communication.

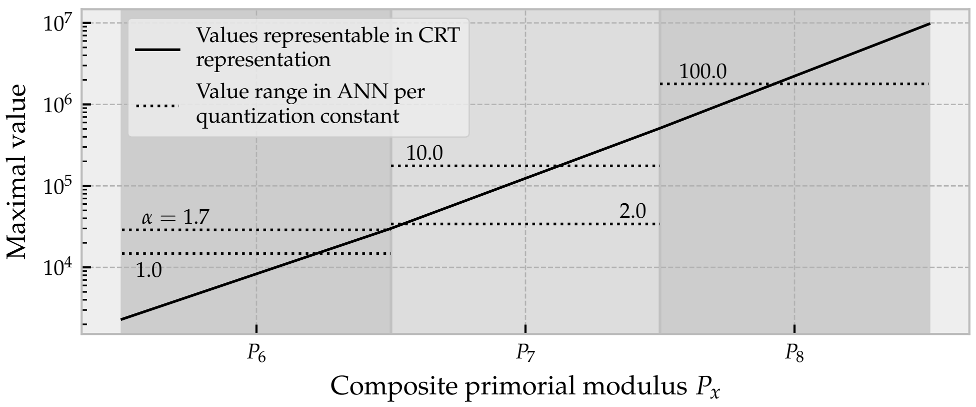


Figure 6.1: Relation between the quantization constant α and the CPM of the CRT representation leveraged in the garbled Model-A.

Table 6.1: Influence of the quantization value and CPM on the performance and accuracy of garbled Model-A. The reported values are means regarding the MNIST test set with a relative standard derivation below 2%.

Quantization constant		1.0	1.7	2.0	10.0	100.0
Size of value range		14.812	28.817	34.060	175.897	1.785.814
CPM		P_6	P_6	P_7	P_7	P_8
Clock cycles	offline	1.216E+05	1.214E+05	1.333E+05	1.333E+05	1.449E+05
	online	2.108E+04	2.110E+04	2.307E+04	2.310E+04	2.500E+04
Runtime (s)	offline	0.122	0.121	0.133	0.133	0.145
	online	0.021	0.021	0.023	0.023	0.025
Acc. after encoding		48.61%	71.30%	80.52%	90.11%	90.02%
Garbled acc.		46.45%	71.32%	80.50%	90.88%	90.02%

erated by about 9%. However, this speedup costs about 20% accuracy. If the quantization does not divide by a quantization constant, the model's accuracy suffers even more, and a speedup cannot be achieved since the CPM P_k does not fall below $k = 6$. The significant impact that the new parameters α and the CPM already have on the performance and accuracy of a small garbled ANN emphasizes their importance and demonstrates the need for careful consideration for garbling larger and deeper ANNs.

6.2 Searching Mixed-Radix Bases and the Approximated Sign Function

Ball et al. [BCM+19] present MRS-bases for the approximated sign computation for the CPMs P_4 to P_{11} for different correctness levels (above 99% accuracy). In our evaluation, we could not reproduce these correctness levels for the presented MRS bases. For exam-

Table 6.2: Accuracy of the garbled Model-B for different input layer sizes and mixed-radix bases. The reported values are means regarding the MNIST test set with a relative standard derivation below 2%.

Model		Model-B30		Model-B100	
MRS Bases		{292, 8 }	{78, 7, 5, 4 }	{292, 8 }	{78, 7, 5, 4 }
Size of value range		2.044.579 $\in [P_8]$		2.401.846 $\in [P_8]$	
Clock cycles	offline	2.811E+05	2.821E+05	7.017E+05	7.076E+05
	online	7.531E+04	7.633E+04	2.472E+05	2.502E+05
Runtime (s)	offline	0.281	0.282	0.702	0.708
	online	0.075	0.076	0.247	0.250
Acc. after encoding		95.92%		97.05%	
Garbled acc.		89.77%	32.71%	93.65%	42.00%

ple, the MRS bases {78, 7, 5, 4} for P_8 have correctness of over 99.99%, according to Ball et al., whereas we could only observe correctness of 98.20%. All other bases presented for P_8 performed even worse. Therefore, we implemented our own search. The search space is relatively large, so we restricted our search range relatively strongly. Nevertheless, we were able to find better bases, e.g. {292, 8}. These bases achieve correctness of 99.43%.

We have garbled Model-Bx with two dense layers and with the bases of Ball et al. and ours. The first layer has x neurons with ReLU activations, and the second layer has ten neurons. As shown in table 6.2, even small changes in the accuracy of the approximation of the sign function have an enormous impact on the accuracy of a garbled ANN. While we obtain an accuracy loss of ~5% respectively ~3% with our bases compared to the accuracy after encoding, the bases of Ball et al. led to a total failure with accuracy drops of well over 50%. In addition, our smaller bases achieve a slight performance advantage.

We are confident that we can further improve the accuracy of our garbled ANNs with the extension of the search space for MRS bases. To handle this search space, we propose to parallelize the search and port it to the GPU. In addition, it might be interesting to search for MRS bases that perform particularly well for the input domains that are relevant to us. As shown in figure 6.2, the errors of the approximated sign function occur highly locally restricted in the range of the input. In addition, it could be interesting to include besides the accuracy the optimization goal of the performance and to search for small bases.

6.3 Computational Workload in Comparison

Next, we garble MNIST Model A of Ball et al. to compare the computational workload with fancy-garbling. This comparison is possible despite the differences because neither our evaluation nor the evaluation of fancy-garbling considers the communication over-

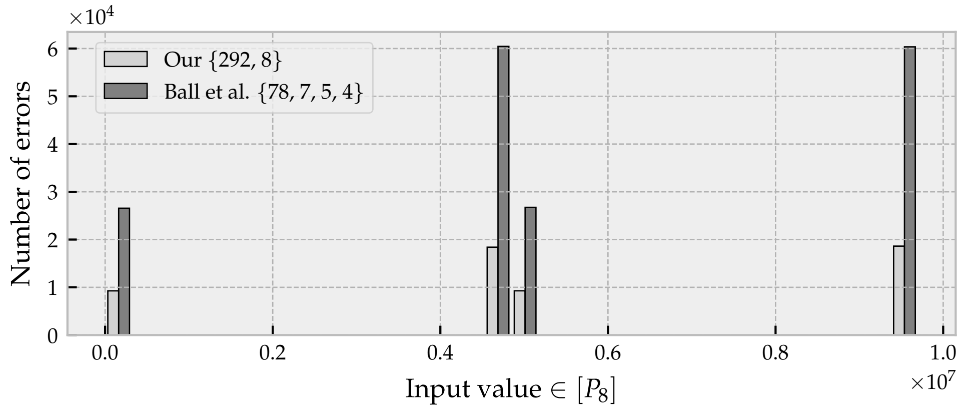


Figure 6.2: Error distribution of the approximated sign function by input range and selected MRS-bases for CPM P_8. Light gray bars show the performance of our chosen MRS-bases, and dark gray bars show the performance of the best MRS-bases from Ball et al. [BCM$^+$19] evaluated by the target function of our MRS-bases search.

head of the schemes and is completely executed on one CPU. MNIST Model A consists of three dense layers with 128, 128, and 10 neurons. The neurons of the first two layers use ReLU activations.

As shown in table 6.3, the whole fancy-garbling scheme runs about 16-17 times faster than our scheme. However, we must consider that fancy-garbling distributes the scheme over eight cores (presumably 16 threads), executes garbling and evaluation in parallel, evaluates on a significantly higher clocked CPU (2.5GHz vs. 3.5GHz), and does not encode structural information into the circuit[26]. We estimate that we evaluate a similar number of gates per thread. Thus the computational workload as expected barely differs. Our CPU implementation can also be parallelized well on the CPU due to the abstraction layers if we evaluate all gates per CRT residue in one thread. Presumably, we can achieve a speedup similar to fancy-garbling for a CPU-only implementation.

Furthermore, as shown in Table 6.3, the now comparatively low accuracy achieved by our MRS bases leads to a significant accuracy drop for this ANN. The two consecutive layers with ReLU activations require a more accurate sign approximation. We can expand the search space for our MRS bases here as well, which is a promising approach to reaching the same accuracy as Ball et al.

[26]Fancy-garbling can garble and evaluate gates in parallel and does not require any structure information about the circuit since it streams the circuit gate-wise. Streaming is not possible in our scheme due to our division into an offline and online phase.

Table 6.3: Comparison of the computational workload and the accuracy of our approach with Ball et al. Note that this comparison neglects the communication costs. The reported values are means regarding the MNIST test set with a relative standard derivation below 1%. Blank values (-) are unknown.

Approach		Our (single-thread)	Ball et al. [BCM+19] (multi-thread)
Size of value range		$8.960.678 \in [P_8]$	$- \in [P_8]$
Clock cycles	offline	1.289E+06	-
	online	3.748E+05	-
Runtime (s)	offline	1.289	-
	online	0.375	-
	total	(1.664)	(0.1)
Approx. sgn acc.		99.43%	$\geq$ 99.99%
Acc. after encoding		97.23%	-
Garbled acc.		86.31%	95.70%

Table 6.4: Comparison of Sprint's communication volume (CV) and communication rounds (CRs) in the online phase with recent OML schemes for deep CNNs. ResNet32 is trained on CIFAR-100, and all other CNNs are trained on the ImageNet dataset from 2012. The scheme authors have documented entries with asterisks, and all other entries were calculated based on the scheme descriptions.

Scheme	Sprint		Slalom [TB19]		Delphi [MLS+20]		MiniONN [LJLA17]
Measure	CV	CRs	CV	CRs	CV	CRs	CV
VGG16	20.8MB	1	>50MB*	17	-	17	-
ResNet32	50.8kB	1	1.16MB	31	60MB*	31	-
ResNet50	20.8MB	1	36.9MB	50	-	50	-
ResNet101	20.8MB	1	57.8MB	101	-	101	-
ResNet152	20.8MB	1	62.6MB	152	-	152	789.2GB*

6.4 Online-Communication in Comparison

In the this step, we evaluate the communication overhead in the online phase of our scheme when evaluating deep CNNs (VGG16 [SZ15] and ResNet [HZRS16]) and compare with recent schemes for outsourced ANN inference. All CNNs were trained with the ImageNet dataset [DDS+09] except ResNet32 which was trained with CIFAR-100 [KH+09]. We compare both the communication volume (CV) and the number of communication rounds (CR). In the case of Sprint and Slalom [TB19], we consider the communication between the client and the TEE and between the TEE and the FPU. In the case of Delphi [MLS+20] and MiniONN [LJLA17], we consider the communication between the client and the server.

As shown in table 6.4, our scheme dominates this comparison with significantly smaller

Table 6.5: Comparison of Sprint's features with recent OML schemes. The comparison considers ANNs with more than 32 layers to be deep. Pure activation functions are not modified to be suitable for SMPC techniques. Off-the-shelf models are conventionally trained models without special tuning for the OML setting. Model privacy refers to the weights and biases of the model.

Scheme	TEE	FPU	Deep ANNs	Pure activation fun.	Off-the-shelf models	Constant CRs	Constant CV	Malicious security	Model privacy
Sprint	●	●	●	◐	◐	●	●	◐	◐
Slalom [TB19]	●	○	●	●	●	-	-	●	◐
SecureML [MZ17]	-	-	-	-	●	-	-	-	●
MiniONN [LJLA17]	-	-	○	-	●	-	-	-	●
CryptoNets [CBL+18]	-	-	-	◐	◐	●	●	●	-
Delphi [MLS+20]	-	○	●	-	◐	-	-	-	●

● Full, ◐ Optional, ○ Limited, − No support

CVs and requires only a single CR regardless of the number of non-linear layers in the ANN. MiniONN is not competitive due to its huge CV. In the case of Delphi, as in the case of Slalom, the number of CRs grow linearly in the number of non-linear layers of the outsourced ANN. For linear layers, both schemes do not require CRs. Sprint has a CV up to 22 times smaller than Slalom and up to 1181 times smaller than Delphi and is thus clearly the most efficient scheme regarding the communication.
Note that the XONN [RSC+19] and DeepSecure [RRK18] schemes not listed here, which are also based on pure GC approaches, require only a constant number of CRs. However, these schemes are not suitable for general ANNs or depth CNNs, and we have refrained from including them in the comparison. SecureML [MZ17] and the CryptoNets [XBF+14, GDL+16, CBL+18] are also not practical for deep CNNs and therefore not considered. Fancy-garbling is not directly comparable because it does not distinguish between an online and offline phase and streams the GC from the garbler to the evaluator. This stream is orders of magnitude larger than the online cost of Sprint.

6.5 Feature Set in Comparison

In this last part of the evaluation, we briefly compare the feature set and properties of Sprint with other current OML schemes for secure outsourcing of ANNs (see also chapter 3). Table 6.5 shows that Sprint outperforms other schemes also in terms of the provided feature set. In particular, Sprint stands out with full FPU support. Both linear and non-linear layers can be accelerated on an FPU without breaking the underlying security guarantees. Another unique feature is that Sprint offers both semi-honest and malicious security modes. Sprint is one of the few schemes that can securely outsource practically used deep ANNs. Overall, this comparison further illustrates the competitiveness of our scheme. In the future, it would be interesting to deepen and further expand this comparison.

7 Conclusion & Outlook

In this final chapter, we summarize the thesis results, provide an outlook on the next steps until the full implementation of our schema Sprint and discuss the wide range of opportunities for future work.

7.1 Summary

The area of OML is diverse, an exciting research field, growing rapidly, and promises to solve the practical pressing problem of protecting sensitive data in the field of ML. In the past, mixed-SMPC approaches that combine HE, SS, and GCs were considered to pave the most promising road to practical OML schemes for ANNs. We observed that these schemes all need to communicate when switching between SMPC techniques. Since different techniques are used depending on the operation and layer of the ANN to be outsourced, this communication overhead grows linearly in the depth of the ANN. This is problematic since deep ANNs have proven to be especially powerful in many domains. We have identified the Slalom scheme [TB19] as particularly promising, but unfortunately, even this scheme suffers from a communication overhead that grows linearly in the depth of the ANN. However, Slalom differentiates itself from other approaches by alleviating this communication overhead through the use of a TEE and a co-located FPU that communicate via a high-speed interface. Slalom uses the TEE for sensitive computations and the FPU for simple cryptographically protected and accelerated computations. Based on the trend of mixed-SMPC approaches and Slalom, we raised the research question of whether we can omit the communication between the layers of an outsourced ANN and reduce the massive communication overhead in the online phase. Like Slalom, we remain in the typical MLaaS case where a client wants to outsource computational intense inference on ANNs to a powerful server.

After a deep dive into highly optimized arithmetic GCs, including ANN specific optimized gadgets in chapter 4, we presented our new scheme Sprint in chapter 5. Sprint is based entirely on arithmetic GCs and requires only minimal communication in the online phase at the start and end of the inference process. During the computation of the inference, and in particular, between each layer of the ANN, Sprint does not require any communication, thus answering our research question positively. Like Slalom, Sprint uses an FPU co-located to a TEE and is divided into an offline and online phase. In the offline phase, the TEE quantizes and encodes the weights and biases of the outsourced ANN. Then the TEE converts them to their CRT representation for garbling and transmission to the FPU. When the TEE receives an input for inference, it starts the online phase, quantizes and encodes the inputs, transfers them to their CRT representation, and garbles them for transmission to the FPU. The FPU evaluates the garbled input on the GC received in the offline phase and sends the garbled output back to the TEE. The TEE decodes the garbled output and sends it back to the client over a secure channel.

Compared to the typical mixed-SMPC approaches and Slalom, Sprint massively reduces the communication overhead during the online phase. In addition, Sprint is particularly versatile since GCs allow the computation of arbitrary functions. This way, Sprint is ready for future ANN layers and allows pre-processing like normalization and scaling or post-processing like majority voting of model orchestras and much more. In addition, Sprint offers two modes of operation, the faster one with public ANN weights and the malicious secure one with secret weights.

Unfortunately, for reasons already described, it was not possible to complete the implementation of our scheme before the submission of the thesis. However, we completed the full CPU implementation, including searching for suitable MRS bases for the approximated sign gadget and evaluating it concerning computational performance, accuracy and communication cost in the online phase. We show the competitiveness of our CPU implementation and gain deeper insights into the search and error structure of MRS-bases for the garbled sign approximation.

7.2 Outlook

Three implementation steps are missing before the completion of Sprint. The most complex one is to finish the implementation of the GC evaluation on a CUDA-enabled GPU. From this part of the implementation, we expect a significant performance boost. The details that have to be considered for this part of the scheme are described in section 2.6 and section 5.6. The second step involves isolating the sensitive computations and intermediate results belonging to the garbling procedure in an SGX enclave. Since the size limitation of the SGX enclaves, which existed at the beginning of the thesis, has been raised considerably, this step should be smoothly implementable. The last missing step concerns the search for suitable MRS bases for the approximated sign gadget. This search can be well parallelized and potentially accelerated on a GPU. From this step, we expect a significant accuracy improvement for deeper and more complex ANNs.

7.3 Future Work

While the OML domain offers countless possibilities for future work, we see four interesting optimization areas concerning our scheme, Sprint, and its use case. The first area of GC optimizations is the smallest, as we already did extensive literature research and implemented many optimizations. However, it might be exciting to take a closer look at binary GCs and binarized ANNs. The work on the scheme XONN [RSC+19] deals with the binary case and can serve as a good starting point.

The second area concerns the setting of our scheme. Since Sprint significantly optimizes the communication in the online phase, it would be interesting to research use cases where the communication is comparatively expensive. This does not apply to the case described in the thesis since the communication between the TEE, and the co-located FPU is comparatively cheap. For example, a scenario in which no TEE is available on the server or the security guarantee of a TEE is not sufficient would be thinkable. Then the client must send the garbled ANN in the offline phase and the garbled inputs in the online phase to the server. Subsequently the server answers after the GC evaluation with the garbled outputs.

The third area regards the implementation of our scheme. First of all, it would be interesting to exploit further optimization possibilities like Intel SSE, AVX, or multi-threading and observe their influence on the performance. Besides, a performance comparison of Intel SGX with AMD SEV would be exciting. While not directly related to the implementation, several algorithmic optimizations exist for the matrix multiplication (like the prominent one of Strassen [Str69]), which account for a large part of our computational workload. It would be interesting to study to what extent these optimizations are applicable and valuable in arithmetic GCs.

The last area concerns the ANNs to be garbled and is quite broad since we did not perform any optimizations of this kind so far. First, we could use quantization-aware training techniques [JKC+18, WJZ+20] to make the ANNs robust to accuracy losses. Besides quantization, it would also be interesting to consider the approximation of the ReLU function already during training. For example, it would be reasonable to shift pre-activation values in the error range of the input of the approximated sign function (see also figure 6.2) into error-free ranges or to use the approximated ReLU function on the forward-pass. We expect a significant accuracy or since smaller MRS bases can be used, performance boost by such an approach. To optimize the communication during the offline phase, all standard methods for model reduction, e.g., pruning [MS88, Kar90], are interesting to study. Most of the work in the field of OML deals exclusively with inference, and we are not aware of any scheme that allows outsourcing the training of practically relevant deep ANNs.

Therefore, it would be very exciting to explore how our arithmetic GC approach could be extended by training methods.

References

[Citing pages are listed after each reference.]

[AAB^{+}15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. [Pages 8 and 43.]

[ACG^{+}16] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 308–318. ACM, 2016. [Page 9.]

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 120–129. IEEE Computer Society, 2011. [Page 30.]

[BCM^{+}19] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. Garbled neural networks are practical. *IACR Cryptol. ePrint Arch.*, 2019:338, 2019. [Pages 2, 30, 31, 32, 33, 35, 37, 41, 43, 44, 46, and 47.]

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991. [Page 65.]

[Ben12] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham W. Taylor, and Daniel L. Silver, editors, *Unsupervised and Transfer Learning - Workshop held at ICML 2011, Bellevue, Washington, USA, July 2, 2011*, volume 27 of *JMLR Proceedings*, pages 17–36. JMLR.org, 2012. [Page 26.]

[BFH^{+}18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. [Page 8.]

[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 Trans. Inf. Forensics Secur.*, 6(2):452–468, 2011. [Page 2.]

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 478–492. IEEE Computer Society, 2013. [Page 36.]

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796. ACM, 2012. [Page 13.]

References

[BMD+17] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. In William Enck and Collin Mulliner, editors, *11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017*. USENIX Association, 2017. [Page 18.]

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513. ACM, 1990. [Pages 13 and 14.]

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 565–577. ACM, 2016. [Pages 2, 13, 29, 37, 41, and 42.]

[BOP06] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In Sviatoslav Voloshynovskiy, Jana Dittmann, and Jessica J. Fridrich, editors, *Proceedings of the 8th workshop on Multimedia & Security, MM&Sec 2006, Geneva, Switzerland, September 26-27, 2006*, pages 146–151. ACM, 2006. [Page 2.]

[C+15] François Chollet et al. Keras. `https://keras.io`, 2015. [Pages 8 and 43.]

[CBL+18] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *CoRR*, abs/1811.09953, 2018. [Pages 26 and 48.]

[CD16] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016. [Page 16.]

[CJM20] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 189–218. Springer, 2020. [Page 23.]

[CL01] Yan-Cheng Chang and Chi-Jen Lu. Oblivious polynomial evaluation and oblivious neural learning. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 369–384. Springer, 2001. [Page 2.]

[CLD16] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 857–874. USENIX Association, 2016. [Page 16.]

[Cré87] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 350–354. Springer, 1987. [Page 11.]

[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 Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009. [Page 47.]

[Den12] Li Deng. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process. Mag.*, 29(6):141–142, 2012. [Page 43.]

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015. [Page 25.]

[Dwo08] Cynthia Dwork. Differential privacy: A survey of results. In Manindra Agrawal, Ding-Zhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation, 5th International Conference, TAMC 2008, Xi'an, China, April 25-29, 2008. Proceedings*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2008. [Page 9.]

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, pages 205–210. Plenum Press, New York, 1982. [Page 11.]

[EMH19] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20:55:1–55:21, 2019. [Page 28.]

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1322–1333. ACM, 2015. [Page 23.]

[FLJ$^+$14] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon M. Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 17–32. USENIX Association, 2014. [Page 23.]

[Fre77] Rusins Freivalds. Probabilistic machines can use less running time. In Bruce Gilchrist, editor, *Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto, Canada, August 8-12, 1977*, pages 839–842. North-Holland, 1977. [Page 24.]

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010. [Page 8.]

[GDL$^+$16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016. [Pages 26 and 48.]

[GESM17] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel SGX. In Cristiano Giuffrida and Angelos Stavrou, editors, *Proceedings of the 10th European Workshop on Systems Security, EUROSEC 2017, Belgrade, Serbia, April 23, 2017*, pages 2:1–2:6. ACM, 2017. [Page 18.]

[GKWY20] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 825–841. IEEE, 2020. [Page 36.]

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 567–578. ACM, 2015. [Page 36.]

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984. [Page 9.]

References

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987. [Pages 11 and 65.]

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011. [Page 36.]

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010. [Page 9.]

[HMvdW$^+$20] Charles R. Harris, K. Jarrod Millman, Stéfan van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with numpy. *Nat.*, 585:357–362, 2020. [Page 7.]

[HP94] CY Hung and B Parhami. An approximate sign detection method for residue numbers and its application to rns division. *Computers & Mathematics with Applications*, 27(4):23–35, 1994. [Page 33.]

[HP17] John L. Hennessy and David A. Patterson. *Computer Architecture, Sixth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 6th edition, 2017. [Page 18.]

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006. [Page 26.]

[HSS$^+$18] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *CoRR*, abs/1803.05961, 2018. [Page 23.]

[HZG$^+$18] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Max Augustin, Michael Backes, and Mario Fritz. Mlcapsule: Guarded offline deployment of machine learning as a service. *CoRR*, abs/1808.00590, 2018. [Page 23.]

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. [Page 47.]

[Int21] Intel. Intel software guard extensions developer reference for linux* os, 2021. [Page 16.]

[JKC$^+$18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2704–2713. Computer Vision Foundation / IEEE Computer Society, 2018. [Page 53.]

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1651–1669. USENIX Association, 2018. [Page 27.]

[Kap16] David Kaplan. {AMD} x86 memory encryption technologies. 2016. [Page 16.]

[Kar90] Ehud D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Trans. Neural Networks*, 1(2):239–242, 1990. [Page 53.]

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Pages 7 and 43.]

[KH$^+$09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. [Page 47.]

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for XOR gates that beats free-xor. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457. Springer, 2014. [Page 13.]

[KO11] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011. [Page 8.]

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008. [Pages 13 and 14.]

[KS13] Vladimir Kolesnikov and Thomas Schneider. Secure function evaluation techniques for circuits containing xor gates with applications to universal circuits, May 14 2013. US Patent 8,443,205. [Page 14.]

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 285–300. USENIX Association, 2012. [Page 36.]

[LBW$^+$18] Yunhui Long, Vincent Bindschaedler, Lei Wang, Diyue Bu, Xiaofeng Wang, Haixu Tang, Carl A. Gunter, and Kai Chen. Understanding membership inferences on well-generalized learning models. *CoRR*, abs/1802.04889, 2018. [Page 23.]

[LJLA17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 619–631. ACM, 2017. [Pages 2, 25, 47, and 48.]

[LLP$^+$19] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using SGX. In Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos, editors, *The 25th Annual International Conference on Mobile Computing and Networking, MobiCom 2019, Los Cabos, Mexico, October 21-25, 2019*, pages 46:1–46:17. ACM, 2019. [Page 23.]

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, 2007. [Page 11.]

[LPK16] Rone Kwei Lim, Linda R. Petzold, and Çetin Kaya Koç. Bitsliced high-performance AES-ECB on gpus. In Peter Y. A. Ryan, David Naccache, and Jean-Jacques Quisquater, editors, *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, volume 9100 of *Lecture Notes in Computer Science*, pages 125–133. Springer, 2016. [Page 42.]

References

[LPS08] Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *Security and Cryptography for Networks, 6th International Conference, SCN 2008, Amalfi, Italy, September 10-12, 2008. Proceedings*, volume 5229 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 2008. [Page 36.]

[LSG+17] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 557–574. USENIX Association, 2017. [Page 18.]

[MAB+13] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In Ruby B. Lee and Weidong Shi, editors, *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013*, page 10. ACM, 2013. [Page 16.]

[MCXS17] Jianwei Ma, Xiaojun Chen, Rui Xu, and Jinqiao Shi. Implementation and evaluation of different parallel designs of AES using CUDA. In *Second IEEE International Conference on Data Science in Cyberspace, DSC 2017, Shenzhen, China, June 26-29, 2017*, pages 606–614. IEEE Computer Society, 2017. [Page 42.]

[MLS+20] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2505–2522. USENIX Association, 2020. [Pages 2, 27, 47, and 48.]

[MPS15] T Malkin, V Pastro, and A Shelat. The whole is greater than the sum of its parts: Linear garbling and applications. In *Workshop talk at Securing Computation Workshop in Berkley*, 2015. [Pages 29 and 31.]

[MR18] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018. [Pages 2 and 24.]

[MS88] Michael Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*, pages 107–115. Morgan Kaufmann, 1988. [Page 53.]

[Mur12] Kevin P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012. [Pages 5 and 6.]

[MZ17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38. IEEE Computer Society, 2017. [Pages 2, 24, and 48.]

[NAI17] Naoki Nishikawa, Hideharu Amano, and Keisuke Iwai. Implementation of bitsliced AES encryption on cuda-enabled GPU. In Zheng Yan, Refik Molva, Wojciech Mazurczyk, and Raimo Kantola, editors, *Network and System Security - 11th International Conference, NSS 2017, Helsinki, Finland, August 21-23, 2017, Proceedings*, volume 10394 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2017. [Page 42.]

[NBGS08] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. In *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2008, Los Angeles, California, USA, August 11-15, 2008, Classes*, pages 16:1–16:14. ACM, 2008. [Page 18.]

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 245–254. ACM, 1999. [Page 11.]

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999. [Pages 13, 14, and 36.]

[NSH19] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 739–753. IEEE, 2019. [Page 23.]

[Nvi21] Nvidia. Cuda toolkit documentation, 2021. [Pages 19 and 21.]

[OPB07] Claudio Orlandi, Alessandro Piva, and Mauro Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP J. Inf. Secur.*, 2007, 2007. [Page 2.]

[OSF+16] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 619–636. USENIX Association, 2016. [Page 23.]

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. [Page 8.]

[PMG+17] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 506–519. ACM, 2017. [Page 23.]

[PS19] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Comput. Surv.*, 51(6):130:1–130:36, 2019. [Page 16.]

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009. [Page 13.]

[PSSY20] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: improved mixed-protocol secure two-party computation. *IACR Cryptol. ePrint Arch.*, 2020:1225, 2020. [Page 24.]

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2005:187, 2005. [Page 11.]

[RAD+78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978. [Page 26.]

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. Cryptology ePrint Archive, Report 2021/749, 2021. `https://eprint.iacr.org/2021/749`. [Page 15.]

[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, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 2:1–2:6. ACM, 2018. [Pages 2 and 48.]

References

[RSC+19] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: xnor-based oblivious deep neural network inference. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 1501–1518. USENIX Association, 2019. [Pages 2, 48, and 53.]

[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 Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 707–721. ACM, 2018. [Page 2.]

[SCNS16] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. Preventing page faults from telling your secrets. In Xiaofeng Chen, XiaoFeng Wang, and Xinyi Huang, editors, *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*, pages 317–328. ACM, 2016. [Page 18.]

[SGS10] John E. Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Comput. Sci. Eng.*, 12(3):66–73, 2010. [Page 20.]

[SS08] Ahmad-Reza Sadeghi and Thomas Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers*, volume 5461 of *Lecture Notes in Computer Science*, pages 336–353. Springer, 2008. [Page 2.]

[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 2017, San Jose, CA, USA, May 22-26, 2017*, pages 3–18. IEEE Computer Society, 2017. [Page 23.]

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969. [Page 53.]

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Page 47.]

[TB19] Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Pages 1, 23, 37, 47, 48, and 51.]

[TGS+18] Shruti Tople, Karan Grover, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. Privado: Practical and secure DNN inference. *CoRR*, abs/1810.00602, 2018. [Page 23.]

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 601–618. USENIX Association, 2016. [Page 23.]

[WC19] Canhui Wang and Xiaowen Chu. GPU accelerated AES algorithm. *CoRR*, abs/1902.05234, 2019. [Page 42.]

[WCP+17] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2421–2434. ACM, 2017. [Page 18.]

[WJZ+20] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *CoRR*, abs/2004.09602, 2020. [Page 53.]

[WRP19] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *CoRR*, abs/1905.01392, 2019. [Page 28.]

[XBF+14] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig. Crypto-nets: Neural networks over encrypted data. *CoRR*, abs/1412.6181, 2014. [Pages 26 and 48.]

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986. [Pages 11 and 13.]

[ZHC+20] Fan Zhang, Warren He, Raymond Cheng, Jernej Kos, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. The ekiden platform for confidentiality-preserving, trustworthy, and performant smart contracts. *IEEE Secur. Priv.*, 18(3):17–27, 2020. [Page 23.]

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer, 2015. [Pages 13 and 15.]

Appendix

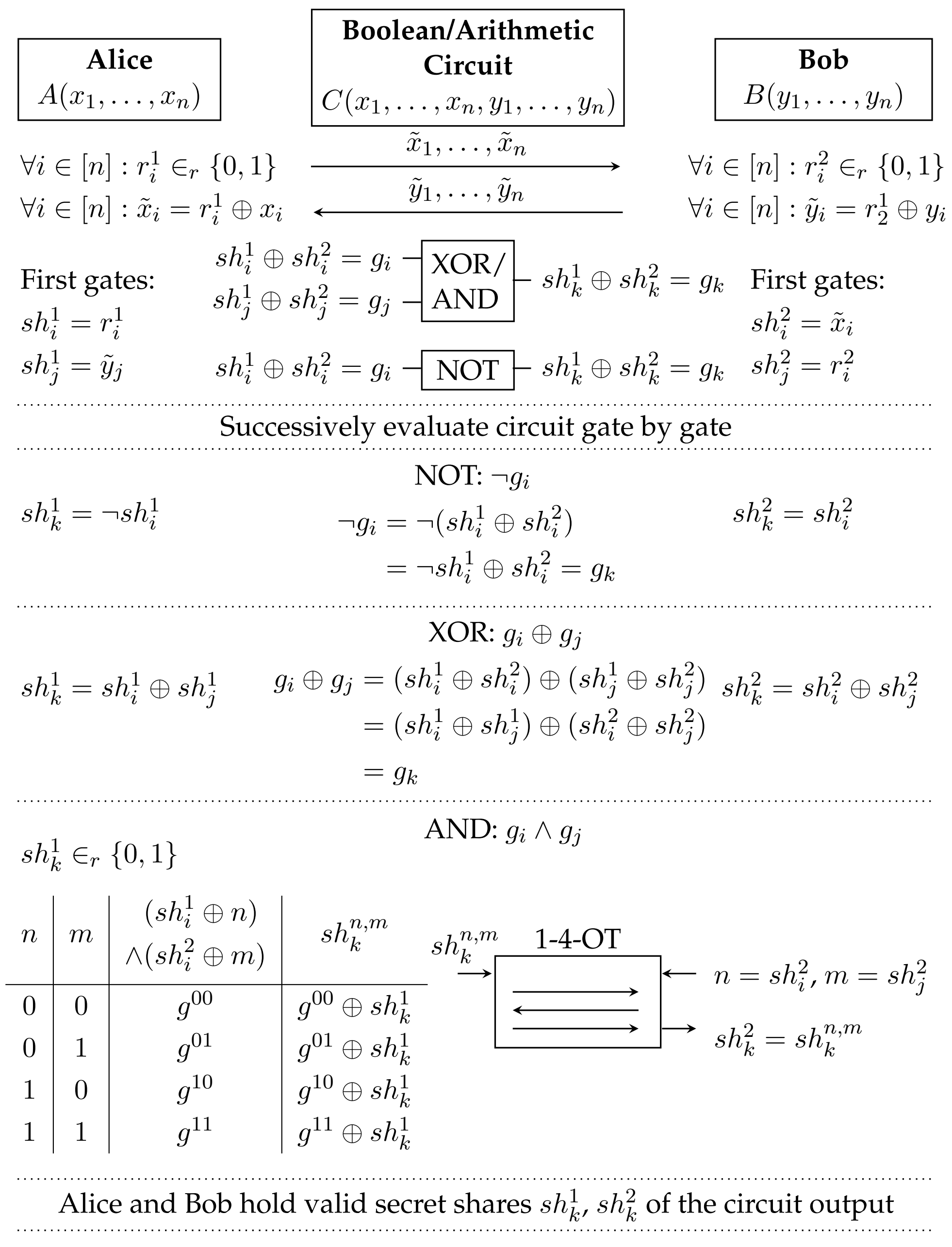
GMW Protocol

The GMW protocol was published in 1987 and is an additive SMPC protocol for boolean and arithmetic circuits named after its authors Goldreich, Micali, and Wigderson [GMW87]. While figure 1 shows the procedure of the protocol for the two parties Alice with inputs $x_1, \ldots, x_n$ and Bob with inputs $y_1, \ldots, y_n$, the protocol generalizes nicely to the general SMPC setting (see [GMW87]).
In the first step, Alice and Bob mask their inputs with random bits r_i and send the result to each other. In this way, the generated masks and the received masked values form a secret sharing of the inputs. Using the shared inputs, the gates of the circuit can now be successively evaluated without any party learning the other's inputs. In figure 1, we consider a boolean circuit consisting of XOR, AND, and NOT gates[27].
To evaluate the NOT gate, it is sufficient for one party to negate its input, as Alice does in figure 1. Then both parties hold a valid sharing of the output of the NOT-gate. The evaluation of the XOR gate is just as straight-forward: Both parties xore their shares.
While the evaluation of NOT and XOR gates does not require any communication between the parties, the evaluation of the AND gate requires a 1-out-of-4 oblivious transfer. In the first step, a party chooses a random bit sh_k^1 as its share of the output. Then, it computes all four possible evaluations of the AND gate $g^{n,m}$ given its input sh_i^1, sh_j^1 and the unknown input of the other party and masks them with the random bit sh_k^1. Next, it transfers the masked output value that arises under the other party's correct input to the other party using the oblivious transfer. In the end, both parties hold a valid share sh_k^1, sh_k^2 of the output of the AND gate without learning the input of the other party.

Beaver Triples

With the introduction of *beaver triples* (BT), also called multiplication triples, Beaver [Bea91] shows how to transfer a large part of the communication overhead of SMPC protocols to an input-independent offline phase. At the same time, BTs naturally transfer the additive sharing approach of the GMW protocol to the arithmetic input-domain. Like NOT and XOR, addition, and multiplication by a constant are trivial and require no

[27]These gates are sufficient to model arbitrary functions.


Alice
$A(x_1, \ldots, x_n)$
Boolean/Arithmetic Circuit
$C(x_1, \ldots, x_n, y_1, \ldots, y_n)$
Bob
$B(y_1, \ldots, y_n)$
$\forall i \in [n] : r_i^1 \in_r \{0,1\}$
$\forall i \in [n] : \tilde{x}_i = r_i^1 \oplus x_i$
$\tilde{x}_1, \ldots, \tilde{x}_n$
$\tilde{y}_1, \ldots, \tilde{y}_n$
$\forall i \in [n] : r_i^2 \in_r \{0,1\}$
$\forall i \in [n] : \tilde{y}_i = r_2^1 \oplus y_i$
First gates:
$sh_i^1 = r_i^1$
$sh_j^1 = \tilde{y}_j$
$sh_i^1 \oplus sh_i^2 = g_i$
$sh_j^1 \oplus sh_j^2 = g_j$
XOR/ AND
$sh_k^1 \oplus sh_k^2 = g_k$
$sh_i^1 \oplus sh_i^2 = g_i$
NOT
$sh_k^1 \oplus sh_k^2 = g_k$
First gates:
$sh_i^2 = \tilde{x}_i$
$sh_j^2 = r_i^2$
Successively evaluate circuit gate by gate
NOT: $\neg g_i$
$sh_k^1 = \neg sh_i^1$
$\neg g_i = \neg(sh_i^1 \oplus sh_i^2) = \neg sh_i^1 \oplus sh_i^2 = g_k$
$sh_k^2 = sh_i^2$
XOR: $g_i \oplus g_j$
$sh_k^1 = sh_i^1 \oplus sh_j^1$
$g_i \oplus g_j = (sh_i^1 \oplus sh_i^2) \oplus (sh_j^1 \oplus sh_j^2) = (sh_i^1 \oplus sh_j^1) \oplus (sh_i^2 \oplus sh_j^2) = g_k$
$sh_k^2 = sh_i^2 \oplus sh_j^2$
AND: $g_i \wedge g_j$
$sh_k^1 \in_r \{0,1\}$
$sh_k^{n,m}$
1-4-OT
$n = sh_i^2, m = sh_j^2$
$sh_k^2 = sh_k^{n,m}$
Alice and Bob hold valid secret shares sh_k^1, sh_k^2 of the circuit output

n	m	$(sh_i^1 \oplus n) \wedge (sh_i^2 \oplus m)$	$sh_k^{n,m}$
0	0	g^{00}	$g^{00} \oplus sh_k^1$
0	1	g^{01}	$g^{01} \oplus sh_k^1$
1	0	g^{10}	$g^{10} \oplus sh_k^1$
1	1	g^{11}	$g^{11} \oplus sh_k^1$

Figure 1: Procedure of the GMW protocol.

communication. For a multiplication, Beaver's approach produces correlated value pairs offline and later leverages them in an elegant way requiring only the communication of two variable-openings per multiplication. Using BTs, the arithmetic GMW protocol is secure against $n-1$ corrupted parties in the semi-honest attacker model.
Consider the case where Alice and Bob have an additive sharing $[x]$, $[y]$ of the inputs of a multiplication gate and want to compute the output $[z] = [xy]$. In addition, they determined an additive sharing (the BTs) $[a]$, $[b]$, and $[c]$ with $c = a \cdot b$ in an offline phase. To compute the multiplication, during the online phase, both parties first compute locally $[d] = [x - a]$, $[e] = [y - b]$, then open d and e and finally compute $[z] = [xy] = de + d[b] + e[a] + [c]$ locally. Now Alice and Bob hold valid shares of the product z.