Privacy-Preserving Verifiable CNNs

Nuttapong Attrapadung[1(✉)], Goichiro Hanaoaka[1], Ryo Hiromasa[2], Yoshihiro Koseki[2], Takahiro Matsuda[1], Yutaro Nishida[2], Yusuke Sakai[1], Jacob C. N. Schuldt[1], and Satoshi Yasuda[2]

[1] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
{n.attrapadung,hanaoka-goichiro,t-matsuda,yusuke.sakai,
jacob.schuldt}@aist.go.jp
[2] Mitsubishi Electric, Kamakura, Japan
Hiromasa.Ryo@aj.MitsubishiElectric.co.jp,
Koseki.Yoshihiro@ak.MitsubishiElectric.co.jp,
Nishida.Yutaro@da.MitsubishiElectric.co.jp,
Yasuda.Satoshi@ea.MitsubishiElectric.co.jp

Abstract. Convolutional neural networks (CNNs) have emerged as one of the most successful deep learning approaches to image recognition and classification. A recent line of research, which includes zkCNN (ACM CCS '21), vCNN (Cryptology ePrint Archive), and ZEN (Cryptology ePrint Archive), aims at protecting the privacy of CNN models by developing publicly verifiable proofs of correct classification which do not leak any information about the underlying CNN models themselves. A shared feature of these schemes is that they require the entity constructing the proof to have access to both the model and the input in the clear. In other words, a client holding a potentially sensitive input is required to reveal this input to the entity holding the CNN model, thereby sacrificing his privacy, to be able to obtain a verifiable proof of correct classification. This is in contrast to the security guarantees provided by secure classification considered in privacy-preserving machine learning, which does not require the client to reveal his input to obtain a (non-verifiable) classification.

In this paper, we propose a privacy-preserving verifiable CNN scheme that overcomes this limitation of the previous schemes by allowing the client to obtain a classification proof without having to reveal his input. The obtained proof allows the client to selectively reveal properties of the obtained classification and his input, which will be verifiable to any third-party verifier. Our scheme is based on the recent notion of collaborative zk-SNARKs by Ozdemir and Boneh (USENIX '22). Specifically, we construct a new collaborative zk-SNARK based on Bulletproofs achieving an efficient maliciously secure proof generation protocol. Based on this, we then present an optimized approach to CNN evaluation. Finally, we demonstrate the feasibility of our approach by measuring the performance of our scheme on a CNN for classifying the MNIST dataset.

C. Pöpper and L. Batina (Eds.): ACNS 2024, LNCS 14584, pp. 373–402, 2024.
https://doi.org/10.1007/978-3-031-54773-7_15

1 Introduction

Deep learning has shown itself to be a tremendously useful tool in many application areas, and convolution neural networks (CNNs) in particular have emerged as one of the most successful deep learning techniques for tasks such as image recognition and classification. However, to provide accurate results, CNNs often require a large amount of training data. While this is unproblematic in application areas where training data is readily available, obtaining and correctly labeling sufficient training data in other areas is a challenging task made more difficult by issues related to data ownership. To further complicate matters, the training data, as well as the input data to be classified, might, in some applications, be sensitive, and data holders might be unable to share their data with any other party due to privacy concerns.

Privacy-preserving machine learning aims at addressing these issues by making deep learning techniques applicable to sensitive data which cannot be publicly shared. More specifically, a particular active line of research within this research area focuses on secure *classification* (e.g. see [5,7,9,10,19,24,26,29,36–38] to name just a few works). This allows a server, holding a CNN model M and a client, holding an input x, to evaluate the CNN defined by M on x, without the server having to disclose M or the client having to reveal x to the server. A different, but closely related line of research focuses on secure *learning* (e.g. see [11,31,32,41]), which enables a set of servers, each holding different datasets, to train a CNN based on the combined dataset, without each server having to reveal his individual dataset. Note that, due to how training is performed in a CNN, secure classification can easily be derived from secure training. While these works allow the entities jointly computing classification or training a CNN model to keep their inputs private, they do not provide *verifiability* i.e. the ability to verify that a given classification result was indeed obtained for input x with respect to a given model M.

A recent set of works, specifically ZEN [17], vCNN [28] and zkCNN [30], address this by constructing zero-knowledge succinct non-interactive arguments (zk-SNARGs) for CNN classification. Specifically, these works allow a server holding a CNN model M to commit to this, obtaining the commitment com_M, and subsequently producing a proof π, that a given input x will lead to a given classification result with respect to the model committed in com_M. Furthermore, neither the input x nor the classification result need to be given in the clear to a potential verifier, but can themselves be contained in commitments, thereby hiding M, x and the classification result from the verifier. In other words, letting $y \leftarrow \mathsf{EvalCNN}(M, x)$ denote that the classification result y is obtained by evaluating the CNN model M on input x, the proof π is informally a zk-SNARG for the language

$$\{(\mathsf{com}_M, \mathsf{com}_x, \mathsf{com}_y) \mid \exists M, r_M, x, r_x, y, r_y \text{ s.t.} \\ \mathsf{com}_M = \mathsf{Commit}(M, r_M) \wedge \mathsf{com}_x = \mathsf{Commit}(x, r_x) \wedge \\ \mathsf{com}_y = \mathsf{Commit}(y, r_y) \wedge y = \mathsf{EvalCNN}(M, x)\}$$

where Commit denotes the commitment algorithm of a commitment scheme. A potential verifier will of course not gain much information from verifying π alone, but a client holding y and r_y can choose to either directly provide (y, r_y) or prove additional properties about y contained in com_y in a separate proof, thereby choosing what information about y is disclosed while maintaining verifiability of the correctness of the classification result y. Overall, this provides a privacy-preserving way for a client to convince a verifier about the correct classification of his input x as well as selective information about y and x. To illustrate the usefulness of this type of primitive, [28] highlights the example of using deep learning to diagnose diseases. In this case, a central hospital or medical company will hold the model M and publish com_M, and a patient will obtain input x via some form of examination. Obtaining a diagnosis, i.e. the classification result of x based on M, and proof of the above type will allow the patient to show to a third-party e.g. an insurance company, that his diagnosis satisfies the condition of a specific insurance policy, while keeping the exact examination results and corresponding diagnose private.

However, in contrast to secure classification and secure learning, a shared feature of ZEN, vCNN and zkCNN is that the proof generation requires the input x to be available to the entity holding the model. As a consequence, this entity learns both x and the corresponding classification result. In other words, in the example above, the patient will have to sacrifice his privacy with respect to the central hospital or medical company holding M, to be able to obtain a privacy-preserving proof verifiable to a third party.

Our Contribution. In this paper, we address the above highlighted privacy issue regarding the input x and the obtained classification result.

Specifically, our contribution consists of the following:

- Firstly, we propose a new privacy-preserving notion of a verifiable CNN that allows the model M, the input x and the classification result to be kept private.
- Secondly, as a stepping stone towards achieving this notion, we construct a new collaborative zk-SNARK based on Bulletproofs [6].
- Finally, based on our collaborative zk-SNARK, we present a new construction of a verifiable CNN that satisfies our stronger privacy-preserving notion and provide a performance evaluation of this.

In the following, we will provide further details on each of the above items.

New Privacy-Preserving Notion for Verifiable CNNs. Our new privacy-preserving notion for verifiable CNNs requires that the model M, the input x and the resulting classification are kept private while still ensuring that a publicly verifiable proof of correct classification is obtained. Additionally, our new notion enables the entity holding x to selectively reveal properties of the classification result. The definition of our verifiable CNN notion resembles the notion of a publicly auditable multi-party computation (PA-MPC) introduced by Baum *et al.* [2], and concretely extends the PA-MPC definition by Ozdemir-Boneh [34] to cover randomized functionalities as well as the specialized properties required

for our application purpose. More precisely, our definition requires a verifiable CNN to satisfy two main properties. Firstly, a verifiable CNN must implement an interactive proof generation protocol that satisfies the standard notion of maliciously secure MPC, thereby informally guaranteeing that the protocol leaks no information regarding the parties' input besides what can be computed from the protocol output. Here, the output for the party holding the model is defined to be a commitment com_y to the classification result y, whereas the output for the party holding the input is defined to consist of com_y, y and the opening of com_y. Secondly, the proof generation must satisfy the notion of a collaborative zk-SNARK [35]. Note that the witness of the proof, the model and the input, are essentially shared between the parties, and in this setting, a plain zk-SNARK is insufficient since a malicious entity might be able to influence the proof generation in such a way that he can derive information about the other entity's part of the witness from the proof. The above two notions combined ensure that no information leaks regarding the model, the input or the obtained classification result.[1]

A New zk-SNARK Based on Bulletproofs. The main tool we use to obtain our concrete construction of a verifiable CNN is a new collaborative zk-SNARK based on Bulletproofs [6] i.e. we construct a new maliciously-secure protocol for the joint generation of Bulletproofs from shared witnesses. As Bulletproofs involve various group operations and in particular commitments, a naive implementation of this would be highly inefficient. However, we observe that a careful setup of the groups over which the computation is performed combined with a corresponding efficient realization of a commitment functionality, will allow efficient joint computation of Bulletproofs. We break up our construction into three relatively simple steps: firstly, we define a new extended arithmetic black-box (ABB) functionality which provides a setup and commitment functionality tailored to the requirements of Bulletproofs (Sect. 3.1); we then provide an efficient protocol for the joint computation of Bulletproofs based on this (Sect. 3.2); and finally, we show how the extended ABB can be securely realized through an extension of SPDZ [15,40] (Sect. 3.3). We note that as discussed in [35], collaborative zk-SNARK has a number of practical applications, such as healthcare statistics, calculation of credit scores, and audits of financial systems, to name a few, and our Bulletproof-based zk-SNARK provides a new zk-SNARK with a different set of tradeoffs that can be used in these applications

A New Privacy-Preserving Verifiable CNN. Based on our new Bulletproof-based collaborative zk-SNARK, we build a new efficient verifiable CNN with succint proofs. The main advantage compared to the existing schemes such as ZEN, vCNN and zkCNN is that our scheme satisfies our new notion of a privacy-preserving verifiable CNN and thereby provides the stronger privacy-preserving properties guaranteed by this. (As highlighted above, the structure of ZEN, vCNN and zkCNN requires the input x to be revealed in the clear to the entity

[1] Here, the model denotes the *parameters* used in the CNN, and like ZEN, vCNN and zkCNN, the *structure* of the CNN (i.e. the number and different types of CNN layers used) is assumed to be public knowledge.

	Input privacy	Model privacy	Transparent setup	Malicious security	Setting	Optimized for CNNs
CNN-specific schemes						
zkCNN [30]	○[1]	●[1]	●	N/A[2]	Standalone	●
vCNN [28]	○[1]	●[1]	○	N/A[2]	Standalone	●
ZEN [17]	○[1]	●[1]	○	N/A[2]	Standalone	●
pvCNN [42]	◐[3]	◐[3]	○	N/A[3]	Third-party[3]	●
Ours	●	●	●	●	Collaborative	●
General-purpose schemes						
OB22 [35]	●	●	○[4]	●	Collaborative	○
DPPSV22 [16]	●	●	●	○	Collaborative	○

●: The property is satisfied. ○: The property is not satisfied. ◐: The property is not fully satisfied.

[1] These protocols are standalone algorithms for generating an NIZK proof of CNN classification, and requires plaintext access to model and input. When used as suggested in [30, 28, 17], this leads to model privacy but no input privacy.

[2] These protocols do not use an interactive protocol to generate a proof.

[3] This protocol requires a semi-trusted third-party, see the explanation in Section 1.1.

[4] Ozdemir-Boneh's construction can be instantiated with various zk-SNARKs and the authors highlight that an instantiation based on Fractal [13] enables a transparent setup. However, the authors do not implement an instantiation based on Fractal due to the complexity of this and we thus regard the Ozdemir-Boneh construction not to have transparent setup.

Fig. 1. Comparison among verifiable CNN schemes and collaborative zero-knowledge schemes.

holding the model M, and hence by design, these schemes cannot provide similar privacy guarantees.) Besides the strong privacy-preserving properties, our construction inherits the transparency property of Bulletproofs i.e. a common reference string (CRS) generated by a trusted party is not required. By maintaining this property of the original Bulletproofs, we ensure that the verifier need not trust any third party to provide an honestly generated CRS. In contrast, we note that both ZEN and vCNN rely on the zk-SNARK by Groth [21], which requires a trusted CRS. Finally, we provide an experimental evaluation of our verifiable CNN construction based on classifying the MNIST dataset.

1.1 Related Works

There are several recent works that consider verifiability (in a zero-knowledge manner) of CNN classification, such as vCNN [28], zkCNN [30], and ZEN [17]. Additionally, Kang et al. [22] proposed, among other things, an approach to efficiently generate a zk-SNARK for CNN classification and a protocol for verifying accuracy of a CNN model based on this. However, these works do not achieve the privacy guarantees we are considering in this paper i.e. that model M and the input x are kept private by the entities holding these, and the classification result is only learned by the entity holding x. Note that achieving this requires a different structure of the underlying proof generation algorithms which must allow a "joint generation" of a publicly verifiable proof.

The recently proposed pvCNN [42] for privacy-preserving CNN testing is defined in a different setting to the above related works and ours. Specifically, an additionally semi-trusted third party is introduced to perform a (latter) part of the classification computation in *plaintext*. Due to this, some information about the input and CNN model is leaked. In constrast, we aim to not rely on such an external party in this work.

Ozdemir and Boneh [35] introduced the notion of collaborative zk-SNARK which we use in this paper. This is a zk-SNARK system in which the prover's algorithm is distributed among multiple provers who each hold a "witness share"

which constitute a valid witness when combined. They also constructed collaborative zk-SNARK protocols from the plain zk-SNARK systems [12,13,18,21], using MPC. We point out that none of these zk-SNARKs, except for Fractal [13], support a transparent setup (i.e. the CRS must be generated by a trusted party). Furthermore, an instantiation using Fractal is not implemented by the authors, as this was deemed too computationally heavy. In contrast, our collaborative zk-SNARK is derived from Bulletproofs [6] and inherits the transparency property from these. Furthermore, [35] also gave an alternative definition of publicly-auditable multi-party computation (PA-MPC) [2] on which our definition of privacy-preserving verifiable CNN is based. See Sect. 4.1 for the details.

Dayama et al. [16] introduced the notion of distributed-prover zero-knowledge protocols, which is a special kind of interactive zero-knowledge protocols in which multiple provers each holding a witness share try to convince the verifier of the validity of an NP statement. Thus, when restricted to the case in which the provers alone can generate a proof verifiable in public non-interactive manner, it is essentially the same notion as collaborative zk-SNARK. They then constructed distributed-prover versions of recent interactive oracle proofs [1,3,6,39] (which can be transformed to zk-SNARKs via the Fiat-Shamir paradigm). In particular, they instantiated a distributed-prover version of Bulletproof [6]. We remark that their constructions assume that there is a special entity among the provers called an aggregator that is assumed to behave semi-honestly, which does not have a counterpart in the collaborative zk-SNARK of [35]. This prevents the constructions from meaningfully achieving malicious security, which we consider in this paper.

Figure 1 shows a comparison between the above mentioned works and ours.

2 Preliminaries

Basic Notation. For a natural number n, we define $[n] := \{1, \ldots, n\}$. For a discrete finite set S, $x \leftarrow S$ denotes sampling an element x uniformly at random from S.

For a vector $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ and $1 \leq \ell \leq n$, we use the notations $\boldsymbol{a}_{\leq \ell} = (a_1, \ldots, a_\ell)$ and $\boldsymbol{a}_{\ell+1\leq} = (a_{\ell+1}, \ldots, a_n)$. For vectors $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ and $\boldsymbol{b} = (b_1, \ldots, b_n) \in \mathbb{Z}_p^n$, we denote by $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ the inner product $a_1 b_1 + \cdots + a_n b_n$. We also use the notation of a "vector polynomial" $p(X) = \boldsymbol{p}_0 + \boldsymbol{p}_1 X + \cdots + \boldsymbol{p}_d X^d$ and its inner product $\langle l(X), r(X) \rangle = \sum_{i=0}^{d} \sum_{j=0}^{d'} \langle \boldsymbol{l}_i, \boldsymbol{r}_j \rangle X^{i+j}$ where $l(X) = \boldsymbol{l}_0 + \boldsymbol{l}_1 X + \cdots + \boldsymbol{l}_d X^d$ and $r(X) = \boldsymbol{r}_0 + \boldsymbol{r}_1 X + \cdots + \boldsymbol{r}_{d'} X^{d'}$.

In this paper, N will always denote the number of parties participating in a multi-party computation protocol, and λ (given in unary) will always denote the security parameter. PPT stands for *probabilistic polynomial-time*. An efficient algorithm is PPT.

Convolutional Neural Networks. In this paper, we consider feedforward convolutional neural networks (CNNs). These networks consist of several *layers*, which each processes the output of the previous layer and forwards the result as

input to the next layer. The first layer is the *input* layer, whereas the last layer is the *output* layer, the latter typically assigning a confidence score to each class in a set of classes into which the input is supposed to be classified. While most CNNs can be described using a small set of layer types, the number of layers, the ordering of these, and their exact configuration depend on the specific CNN. In the following, we will outline the abstraction of CNNs we rely on in our definitions related to privacy-preserving verifiable CNNs, and the types of layers we consider the CNNs to consist of.

CNN Abstraction. To capture different CNN structures, we will use a generic CNN evaluation algorithm, denoted $\mathsf{EvalCNN}$, with the following syntax:

$$y \leftarrow \mathsf{EvalCNN}(\mathcal{S}, M, x).$$

Here, $\mathcal{S}$ is a representation of the *structure* of the CNN i.e. the used layer types, their ordering and interconnections. M denotes the *model* and consists of the parameters determined by the training of the CNN, and x denotes the input which is to be classified. Finally, y denotes the obtained classification of x. The algorithm $\mathsf{EvalCNN}$ is assumed to iteratively evaluate each layer specified in $\mathcal{S}$, applying the corresponding parameters from M, and using the obtained result as input for the next layer, until the output layer is reached. The output layer typically defines several classes, and we let the final classification y denote the class with the highest score. In the following description of the layer types we consider, we will highlight what parameters are considered to be part of the model M. Note that we will consider the structure $\mathcal{S}$ of the CNN being evaluated to be publicly available.

Layers. In this paper, we will consider the following layers.

Convolution Layer. A convolution layer divides input variables into mutually overlapped small local regions and computes the inner product of each square and weights (a *filter* or a *kernel*). These inner products consist of the output of the layer. Here, weights are a part of the model M.

Pooling Layer. Similarly to a convolution layer, a pooling layer divides input variables into mutually overlapped small local regions. For each region, the values are replaced with another value computed from the values in the region. Different subtypes of pooling layers use different replacements. A *max pooling* layer is a typical pooling layer, in which a region is replaced with the maximum of the values in the region. There is no model parameter in pooling layers.

Activation Function. An activation function is a non-linear function which is directly applied to each value in the output of a previous layer. A typical activation function is the rectified linear unit (ReLU) function, which maps a negative value to zero and a zero or positive value to the same value.

Fully Connected Layer. A fully connected layer, given a set of input values, outputs a set of different values, each of which is the inner product of all the input values and fixed weights. Different outputs use different sets of weights. These sets of weights are a part of M.

SoftMax Function. The SoftMax function is the typical final layer. This layer is given as input a set of input values which constitutes the confidences of

the classifications in that i-th value is the confidence for the classification to the i-th class. Then this layer normalizes by mapping each confidence to the value between 0 and 1 so that the sum of all values is 1. There is no model parameter in the SoftMax function.

We also note that we can extend our construction to support other types of layers. See further details for Sect. 4.2.

Secure Multi-party Computation. We will make use of the standard notion of secure multi-party computation (MPC). Particularly, our main constructions will consider security with abort in the dishonest majority and static corruption setting.

Let $\mathcal{F}$ be a (possibly probabilistic) N-input, N-output function. An N-party computation for f is a protocol among N parties $P_1, \ldots, P_N$ such that each party P_i, which takes x_i as input, receives y_i as the result of an execution of the protocol, where $(y_1, \ldots, y_N) \leftarrow \mathcal{F}(x_1, \ldots, x_N)$. (A function computed by a multi-party protocol is often called a *functionality*.)

For security definitions of a multi-party computation protocol, we use the standard definition of security based on the real/ideal paradigm [8,20]. We will consider *security with abort* as a default notion, where a malicious party may obtain the final result while making the protocol abort and preventing honest parties from obtaining the final results. We note that this security notion is sufficient for our purpose. Also, we will consider the *dishonest majority* and *static corruption* setting. The former means that the number t of corrupted parties can be up to $N-1$, and the latter means that the adversary decides the set of corrupted parties before the execution of the protocol. We will also consider semi-honest security, where corrupted parties do not deviate from the protocol specification.

Definition 1. *Let $\mathcal{F} : \prod_{i\in[N]} \mathcal{X}_i \to \prod_{i\in[N]} \mathcal{Y}_i$ be a (possibly probabilistic) efficiently computable function. We say that an N-party protocol Π for f is* secure with abort *(in the dishonest majority, static corruption setting) if for any PPT adversary $\mathcal{A}$, there exists a PPT ideal-world adversary (also called a simulator) $\mathcal{S}$ such that for any input $\boldsymbol{x} = (x_1, \ldots, x_N) \in \prod_{i\in[N]} \mathcal{X}_i$ and auxiliary-input string $z \in \{0,1\}^*$, the two random variables $\mathsf{real}^{\Pi}_{\mathcal{A}}(\boldsymbol{x}, z)$ and $\mathsf{ideal}^{f}_{\mathcal{S}}(\boldsymbol{x}, z)$ are computationally indistinguishable, where these random variables are defined as follows:*

- *Real execution $\mathsf{real}^{\Pi}_{\mathcal{A}}(\boldsymbol{x}, z)$, generated from an interaction among the set of parties $P_1, \ldots, P_N$ and the adversary $\mathcal{A}$: Given z as input, an adversary $\mathcal{A}$ specifies the set of indices $\mathcal{C} \subset [N]$ of corrupted parties such that $|\mathcal{C}| \leq N-1$, and receives the inputs $\{x_i\}_{i\in\mathcal{C}}$. Then, the protocol Π is executed, where during the protocol execution, the behavior of the corrupted parties P_i with $i \in \mathcal{C}$ is determined by $\mathcal{A}$. After the execution, $\mathcal{A}$ outputs an arbitrary string as its final output. $\mathsf{real}^{\Pi}_{\mathcal{A}}(\boldsymbol{x})$ consists of the outputs of all the honest (i.e. uncorrupted) parties concatenated with $\mathcal{A}$'s final output.*
- *Ideal execution $\mathsf{ideal}^{\mathcal{F}}_{\mathcal{S}}(\boldsymbol{x}, z)$, generated from an interaction between the trusted party (for computing $\mathcal{F}$) and the ideal-world adversary $\mathcal{S}$: Given z as*

input, an ideal-world adversary $\mathcal{S}$ specifies the set of indices $\mathcal{C} \subset [N]$ of corrupted parties such that $|\mathcal{C}| \leq N-1$. At this point, $\mathcal{S}$ may ask the trusted party to abort, in which case, the honest parties' output is forced to be the abort *symbol. Then, $\mathcal{S}$ give an arbitrary value x'_i (not necessarily x_i) for all corrupted indices $i \in \mathcal{C}$. Also, $x'_i = x_i$ is passed to the trusted party for all non-corrupted parties. Then, the trusted party computes $(y_1, \ldots, y_N) \leftarrow \mathcal{F}(x'_1, \ldots, x'_N)$, and gives $(y_i)_{i \in \mathcal{C}}$ to $\mathcal{S}$. For each uncorrupted party index $i \in [N] \setminus \mathcal{C}$, $\mathcal{S}$ decides whether the party i aborts or not. In the former case, y_i is replaced with the* abort *symbol, while y_i is untouched in the latter case. Finally, $\mathcal{S}$ outputs an arbitrary string as its final output.* $\mathsf{ideal}^{\mathcal{F}}_{\mathcal{S}}(\boldsymbol{x}, z)$ *consists of $(y_i)_{i \in [N] \setminus \mathcal{C}}$ concatenated with $\mathcal{S}$'s final output.*

Furthermore, we say that Π is secure against semi-honest parties *if the above indistinguishability is guaranteed only when the corrupted parties controlled by an adversary $\mathcal{A}$ always follow the protocol specification.*

Hybrid Model. We will show the security of our protocols in a hybrid model, where the parties execute a protocol with real messages and also have access to a trusted party computing a subfunctionality for them. The modular sequential composition theorem of [8] states that one can replace the trusted party computing the subfunctionality with a real secure protocol computing the subfunctionality. (This works both security with abort and security against semi-honest parties.) When the subfunctionality is $\mathcal{G}$, we say that the protocol works in the $\mathcal{G}$-hybrid model.

Privacy-Preserving CNNs. To be able to define privacy-preserving verifiable CNNs in Sect. 4, we need to first specify what it means for a 2-party protocol to compute a CNN classification in a privacy-preserving manner (without considering verifiability). We define it as a secure 2-party protocol realizing the functionality described in Fig. 2. Note that the functionality is associated with some commitment scheme whose definition is given below.

Commitments. A commitment scheme consists of two algorithms $\mathsf{Setup}_{\mathsf{Com}}$ and Commit: $\mathsf{Setup}_{\mathsf{Com}}$ is the setup algorithm that takes a security parameter 1^λ as input, and outputs a public parameter pp; Commit is the commitment generation algorithm that takes pp, a message m, and a randomness r as input, and outputs a commitmemt com. As usual, we require *hiding* and *binding* for a commitment scheme. The hiding property states that $\mathsf{Commit}(\mathsf{pp}, m_0, r)$ and $\mathsf{Commit}(\mathsf{pp}, m_1, r)$ are indistinguishable for any two messages m_0 and m_1, where pp is generated by $\mathsf{Setup}_{\mathsf{Com}}$ and r is chosen uniformly at random; The binding property states that given pp generated by $\mathsf{Setup}_{\mathsf{Com}}$, it is hard to find a pair (m_0, r_0) and (m_1, r_1) such that $\mathsf{Commit}(\mathsf{pp}, m_0, r_0) = \mathsf{Commit}(\mathsf{pp}, m_1, r_1)$ and $m_0 \neq m_1$.

Pedersen Commitment. Our proposed protocol will make use of the Pedersen commitment: Its public parameter consists of two group elements $g, h \in \mathbb{G}$ of prime order p. Given a message $m \in \mathbb{Z}_p$ to be committed, choose a randomness $r \in \mathbb{Z}_p$ uniformly at random, and the commitment com is $\mathsf{com} = g^m \cdot h^r$. It is

$\mathcal{F}_{\mathsf{Com}}^{pCNN}$

This is a 2-party functionality (invoked by P_1 and P_2) associated with a commitment scheme $\mathsf{Com} = (\mathsf{Setup}_{\mathsf{Com}}, \mathsf{Commit})$. The functionality expects three types of inputs:

- P_1's private inputs, comprising a CNN model M and a randomness r_M.
- P_2's private inputs, comprising a CNN input x and a randomness r_x.
- public common inputs, consisting of the CNN structure $\mathcal{S}$, a public parameter pp of the underlying commitment scheme Com, and commitments com_M and com_x (which are supposedly generated as $\mathsf{com}_M = \mathsf{Commit}(\mathsf{pp}, M, r_M)$ and $\mathsf{com}_x = \mathsf{Commit}(\mathsf{pp}, x, r_x)$, respectively).

Given the inputs, the functionality performs the following procedure:

1. Compute $y \leftarrow \mathsf{EvalCNN}(\mathcal{S}, M, x)$.
2. Pick r_y randomly, and compute $\mathsf{com}_y \leftarrow \mathsf{Commit}(\mathsf{pp}, y, r_y)$.
3. Output com_y for P_1, and (y, r_y, com_y) for P_2.

Fig. 2. Privacy-preserving CNN functionality.

well-known that the Pedersen commitment scheme is perfectly hiding, and computationally binding under the assumption that the discrete logarithm problem is hard in $\mathbb{G}$.

Collaborative zk-SNARKs. Here, we recall the definition of a collaborative zk-SNARK formalized by Ozdemir and Boneh [35]. (A large part of this paragraph is taken verbatim from [35].) Let $\mathcal{R} \subseteq \{0,1\} \times \{0,1\}^*$ be a binary relation. A collaborative zk-SNARK for $\mathcal{R}$ consists of $(\mathsf{Setup}, \Pi, \mathsf{Verify})$ each of whose syntax is defined as follows:

- Setup is the setup algorithm that takes a security parameter 1^λ as input, and outputs a public parameter pp.
- Π is the proof generation protocol, executed among N parties (provers) $P_1, \ldots, P_N$, where the parties have a public parameter pp and a statement x as public input, and each party P_i has a witness share[2] w_i as private (local) input; As the result of the protocol, the parties output a proof π. For notational convenience, we denote an execution of the protocol (by honest parties) by $\pi \leftarrow \Pi(\mathsf{pp}, x, \boldsymbol{w})$, where $\boldsymbol{w} = (w_1, \ldots, w_N)$.
- Verify is the verification algorithm that takes pp, a statement x, and a proof π as input, and outputs either $\top$ (accept) or $\bot$ (reject).

Note that the verification is non-interactive, and anyone given a statement and a proof can verify the validity of the statement.

A collaborative zk-SNARK in the random oracle model, where each of the algorithms has access to a random oracle $H : X_\lambda \to Y_\lambda$, is denoted by $(\mathsf{Setup}^H, \Pi^H, \mathsf{Verify}^H)$.

[2] Here, a witness share need not be a share of a secret sharing of a witness.

Definition 2. *We require a collaborative zk-SNARK for $\mathcal{R}$ in the random oracle model (where H is modeled as a random oracle), $(\mathsf{Setup}^H, \Pi^H, \mathsf{Verify}^H)$, to satisfy the following properties. Below, let $\mathcal{U}(\lambda)$ be the set of all functions from X_λ to Y_λ.*

- Completeness: *For all $(x, \boldsymbol{w}) \in \mathcal{R}$, the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{l} H \leftarrow \mathcal{U}(\lambda); \\ \mathsf{pp} \leftarrow \mathsf{Setup}^H(1^\lambda); \\ \pi \leftarrow \Pi^H(\mathsf{pp}, x, \boldsymbol{w}) \end{array} : \mathsf{Verify}^H(\mathsf{pp}, x, \pi) = \bot\right].$$

- Knowledge soundness: *For all x, for all sets of PPT algorithms $\boldsymbol{P} = (P_1^*, \ldots, P_N^*)$, there exists a PPT extractor Ext and a negligible function ϵ such that*

$$\Pr\left[\begin{array}{l} H \leftarrow \mathcal{U}(\lambda); \\ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ \boldsymbol{w} \leftarrow \mathsf{Ext}^{H, \boldsymbol{P}^H}(\mathsf{pp}, x) \end{array} : (x, \boldsymbol{w}) \in \mathcal{R}\right] \geq$$
$$\Pr\left[\begin{array}{l} H \leftarrow \mathcal{U}(\lambda); \\ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ \pi \leftarrow \boldsymbol{P}^H(\mathsf{pp}, x) \end{array} : \mathsf{Verify}^H(\mathsf{pp}, x, \pi) = \top\right] - \epsilon(\lambda).$$

  Here, $\mathsf{Ext}^{H, \boldsymbol{P}^H}$ denotes that Ext has oracle access to H and may re-run the collection of provers $\boldsymbol{P}(\mathsf{pp}, x)$, reprogramming the random oracle H each time, and receiving only the final output produced by $\boldsymbol{P}$.
- Succinctness: *Proof size and verification time are $o(|\mathcal{R}|)$, where $|\mathcal{R}|$ denotes the size of the description.*
- t-zero-knowledge: *For any PPT adversary $\mathcal{A}$ controlling $k \leq t$ provers: $P_{i_1}, \ldots, P_{i_k}$, there exists a PPT simulator $\mathcal{S}$ such that for all x, $\boldsymbol{w}$, and for all PPT distinguishers $\mathcal{D}$,*

$$\left|\Pr\left[\begin{array}{l} H \leftarrow \mathcal{U}(\lambda); \\ \mathsf{pp} \leftarrow \mathsf{Setup}^H(1^\lambda); \\ b \leftarrow \mathcal{R}(x, \boldsymbol{w}); \\ (\mathsf{tr}, \mu) \leftarrow \mathcal{S}^H(\mathsf{pp}, x, w_{i_1}, \ldots, w_{i_k}, b) \end{array} : \mathcal{D}^{H[\mu]}(\mathsf{tr}) = 1\right]\right.$$
$$\left. - \Pr\left[\begin{array}{l} H \leftarrow \mathcal{U}(\lambda); \\ \mathsf{pp} \leftarrow \mathsf{Setup}^H(1^\lambda); \\ \mathsf{tr} \leftarrow \mathsf{View}_{\mathcal{A}}^H(x, \boldsymbol{w}) \end{array} : \mathcal{D}^H(\mathsf{tr}) = 1\right]\right|$$

  is negligible in λ, where tr denotes a transcript, $\mathsf{View}_{\mathcal{A}}^H(x, \boldsymbol{w})$ denotes the view of $\mathcal{A}$ when provers $P_1, \ldots, P_N$ interact with input x and witness $\boldsymbol{w}$ (the honest provers follow Π, but dishonest ones may not), μ denotes a partial function from the domain of H, and $H[\mu]$ denotes a re-programmed random oracle (by $\mathcal{S}$) that maps x to $\mu(x)$ if x is defined in μ and $H(x)$ otherwise.

If the above indistinguishability is guaranteed to hold only when the corrupted provers follow the protocol specification, we say that a collaborative zk-SNARK is t-zero-knowledge in the presence of semi-honest provers.[3]

3 Collaborative Bulletproofs

In this section, we will present our new collaborative zk-SNARK based on Bulletproofs. This will be a crucial tool in our construction of a privacy-preserving verifiable CNN presented in Sect. 4.

Bulletproofs support zero-knowledge arguments for arbitrary arithmetic circuits, which is achieved via a proof for a Hadamard-product relation. More specifically, all 'left' and 'right' inputs to multiplication gates are represented as vectors $\boldsymbol{a}_{\mathrm{L}}$ and $\boldsymbol{a}_{\mathrm{R}}$, respectively, and the output as $\boldsymbol{a}_{\mathrm{O}} = \boldsymbol{a}_{\mathrm{L}} \circ \boldsymbol{a}_{\mathrm{R}}$, where $\circ$ denotes the Hadamard product. By adding additional $Q \leq 2 \cdot n$ constraints (expressed via matrices $\boldsymbol{W}_{\mathrm{L}}, \boldsymbol{W}_{\mathrm{R}}, \boldsymbol{W}_{\mathrm{O}}$), where n is the number of multiplication gates, any arithmetic circuits can be captured (see [4]). Bulletproofs additionally include commitments V_j (and commitment weights $\boldsymbol{W}_{\mathrm{V}}$) as part of the statement. Concretely, for the Bulletproof relation $\mathcal{R}$, a statement x is of the form:

$$\boldsymbol{V} \in \mathbb{G}^m, \boldsymbol{W}_{\mathrm{L}}, \boldsymbol{W}_{\mathrm{R}}, \boldsymbol{W}_{\mathrm{O}} \in \mathbb{Z}_p^{Q \times n}, \boldsymbol{W}_{\mathrm{V}} \in \mathbb{Z}_p^{Q \times m}, \boldsymbol{c} \in \mathbb{Z}_p^Q, \tag{1}$$

and a witness w is of the form:

$$\boldsymbol{a}_{\mathrm{L}}, \boldsymbol{a}_{\mathrm{R}}, \boldsymbol{a}_{\mathrm{O}} \in \mathbb{Z}_p^n, v_1, \ldots, v_m, \gamma_1, \ldots, \gamma_m \in \mathbb{Z}_p^m. \tag{2}$$

Then, $(x, w) \in \mathcal{R}$ if and only if

$$\begin{cases} V_j = g^{v_j} h^{\gamma_j} & (j \in [m]) \\ \boldsymbol{a}_{\mathrm{L}} \circ \boldsymbol{a}_{\mathrm{R}} = \boldsymbol{a}_{\mathrm{O}} & \\ \boldsymbol{W}_{\mathrm{L}} \cdot \boldsymbol{a}_{\mathrm{L}} + \boldsymbol{W}_{\mathrm{R}} \cdot \boldsymbol{a}_{\mathrm{R}} + \boldsymbol{W}_{\mathrm{O}} \cdot \boldsymbol{a}_{\mathrm{O}} = \boldsymbol{W}_{\mathrm{V}} \cdot \boldsymbol{v} + \boldsymbol{c} & \end{cases}, \tag{3}$$

where $\boldsymbol{v} = (v_1, \ldots, v_m)$. Like Bulletproofs, our collaborative zk-SNARK will be for this relation, and we will refer to x (resp. w) of the above form as a Bulletproof statement (resp. witness).

We approach our construction gradually, firstly introducing the extended arithmetic black-box abstraction we build our MPC protocol upon in Sect. 3.1, and then the actual protocol construction in Sect. 3.2. Finally, and in Sect. 3.3 we show how the constructed protocol can be realized efficiently for both semi-honest and malicious security, thereby obtaining our collaborative zk-SNARK.

[3] Note that t-zero-knowledge in the presence of semi-honest provers still provides the ordinary zero-knowledge property of a (single-prover) zk-SNARK against a malicious verifier (that does not participate in the proof generation protocol).

3.1 Extended Arithmetic Black-Box

The arithmetic black-box abstraction (ABB) [14] is a commonly used approach for constructing MPC protocols. It abstracts away the details of tools (e.g. secret-sharing, homomorphic commitments and encryption) and corresponding protocols, and allows us to perform field arithmetic in an ideal "black-box" without explicitly knowing the values of the operands. In this paper, we will only treat an ABB functionality whose underlying field is a prime field, and denote its characteristic by p, and for an element $a \in \mathbb{Z}_p$, we use the notation "$[a]$" to mean that a is stored in the black box maintained in the functionality.

As opposed to relying on a standard ABB implementing the most common arithmetic operations, we will define an extended ABB providing additional functionality tailored to the specific computation required in the construction of Bulletproofs. This will in turn simplify and make efficient the ABB-based construction of the protocol for the joint computation of Bulletproofs presented in Sect. 3.2. Specifically, we consider an ABB functionality which is parameterized by a base cyclic group $\mathbb{G}$ of (prime) order p.[4] Besides the standard arithmetic operations on stored values, we will allow the computation of multi-exponentiations with respect to a (public) vector of group elements $\boldsymbol{g} = (g_1, \ldots, g_\ell)$ for some ℓ i.e. for values $[a_1], \ldots, [a_\ell]$ stored in the ABB, the entities interacting with the ABB will be able to obtain the group element $g_1^{a_1} \cdots g_\ell^{a_\ell} \in \mathbb{G}$. In other words, the extended ABB implements a restricted form of computation over the group elements $\boldsymbol{g}$. This restricted functionality allows the computation of Pedersen-style commitments, which play a crucial role in Bulletproofs. Note that while it would be possible to use generic MPC protocols for exponentiation on top of a standard ABB to achieve a similar functionality, the crucial insight here is that the restricted functionality discussed above can be instantiated very efficiently; see Sect. 3.3 for how we achieve this. Additionally, we require several basic non-linear operations such as equality, max, argmax, and bit-decomposition to be provided by the ABB. These functionalities will be used in our verifiable CNN construction presented in Sect. 4.

The full extended ABB functionality is defined in Fig. 3. To ease the notation, we will for values $[x]$ and $[y]$ stored by the ABB and $a \in \mathbb{Z}_p$ use the notation $[x] + [y]$, $a \cdot [x]$, and $[x] \cdot [y]$ to denote the operations $\mathsf{Add}([x], [y])$, $\mathsf{SMult}(a, [x])$, and $\mathsf{Mult}([x], [y])$, respectively. Furthermore, we will omit the operation "$\cdot$" if it is clear from the context.

3.2 Our Construction

We will now present our construction of a collaborative zk-SNARK ($\mathsf{Setup}, \Pi, \mathsf{Verify}$) based on Bulletproofs.

Note that in the prover algorithm of Bulletproofs, multi-exponentiation is central and is used for computing the Pedersen-style commitments Bulletproofs

[4] Note that the order p of $\mathbb{G}$ is identical to the characteristic of the field $\mathbb{Z}_p$ which the values in the ABB are elements of. We require p to be of 2λ bits so that the discrete logarithm problem is hard in $\mathbb{G}$.

$\mathcal{F}_{\mathbb{G}}^{ABB}$

Let $[x]$ denote the identifier for the value $x \in \mathbb{Z}_p$ stored by the functionality. Each of the following operations returns the identifier of the result to all parties unless the output is explicitly defined.

Input(x): Receive $x \in \mathbb{Z}_p$ from a party and store this.
Rand(): Pick random $z \leftarrow \mathbb{Z}_p$ and store z.
Add($[x], [y]$): Compute $z = a + b$ and store $z \in \mathbb{Z}_p$.
SMult($x, [y]$): Compute $z = xy$ and store $z \in \mathbb{Z}_p$.
Mult($[x], [y]$): Compute $z = xy$ and store $z \in \mathbb{Z}_p$.
Com($[x_1], \ldots, [x_\ell], \boldsymbol{g} = (g_1, \ldots, g_\ell) \in \mathbb{G}^\ell$) (for some ℓ): Compute $Z = g_1^{x_1} \cdots g_\ell^{x_\ell}$ and output $Z \in \mathbb{G}$ to all parties.
EQ($[x], [y]$): Let $z = 1$ if $x = y$ and 0 otherwise. Store $z \in \mathbb{Z}_p$.
Max($[x_1], \ldots, [x_\ell]$) (for some ℓ): Compute $z = \max_{1 \le i \le \ell} x_i$ and store $z \in \mathbb{Z}_p$.
Argmax($[x_1], \ldots, [x_\ell]$) (for some ℓ): Compute $z = \arg\max_{1 \le j \le \ell} x_i$ and store z.
BitDecomp($[x]$): Compute $(z_0, \ldots, z_{2\lambda-1})$ such that $x = \sum_{i=0}^{2\lambda-1} 2^i z_i$, and store $z_0, \ldots, z_{2\lambda-1} \in \mathbb{Z}_p$.
Output($[x]$): Output x to all parties.
DOutput($[x], i \in [N]$): Output x (only) to the party P_i.

Fig. 3. Extended arithmetic black-box functionality.

are based on. Computing these is one of the most computationally heavy steps of proof generation, and could be a potential bottleneck when constructing a Bulletproof-based zk-SNARK since the exponents will correspond to witnesses which will be shared among the collaborating parties. However, note that this computation is straightforward to realize when relying on the extended ABB described above, as computing a Pedersen-style commitment can be done via a single call to Com. As a consequence, constructing an efficient prover protocol with respect to the extended ABB becomes a much simpler task (to obtain an efficient realization of the protocol, it will of course be required that the extended ABB itself can be realized efficiently; how this can be done is shown in Sect. 3.3).

In the description of our protocol, we will assume the statement and the corresponding witness are of the form described in Eq. (1) and Eq. (2), respectively, and that the witness is stored (component-wise) in $\mathcal{F}_{\mathbb{G}}^{ABB}$. This will be the case for the application of our protocol in our privacy-preserving verifiable CNN described in Sect. 4. Note, however, that for any arithmetic circuit over $\mathbb{Z}_p$ and a corresponding (witness) input assignment $\boldsymbol{w} = (w_1, \ldots, w_N)$, a representation corresponding to Eq. (1) and Eq. (2) can be computed in a straightforward manner using the functionality of $\mathcal{F}_{\mathbb{G}}^{ABB}$. In the following, we will let $\mathbb{G}$ be a group with 2λ-bit prime order p, and $H_1 : \{0,1\}^* \to \mathbb{Z}_p$, $H_2 : \{0,1\}^* \to \mathbb{Z}_p^2$, $H_3 : \{0,1\}^* \to \mathbb{Z}_p \times \mathbb{G}$, and $G : \mathbb{N} \to \mathbb{G}$ be hash functions (modeled as random oracles).

Setup. This algorithm generates group elements $g, h \in \mathbb{G}, \boldsymbol{g}, \boldsymbol{h} \in \mathbb{G}^n$ using the hash function G, and outputs these as pp.

Proof Generation Protocol Π. Our protocol follows the structure of the original Bulletproofs, and consists of an 'outer' protocol, JointBulletproof shown in Fig. 4, for jointly computing a proof for an arithmetic circuit of the form described in Eq. (1) and Eq. (2), as well as an 'inner' sub-protocol, JointProveIP shown in Fig. 5, for jointly computing a proof for an inner product. The latter is invoked as part of JointBulletproof. Note that compared to the original Bulletproofs, recursion has been eliminated from the inner product computation to avoid complications arising from this in a protocol setting.

Given the Bulletproof statement and the ABB-stored values of a Bulletproof witness, our protocol(s) proceeds by iteratively computing the witness-dependent values required for the next prover message using $\mathcal{F}_{\mathbb{G}}^{ABB}$ (e.g. line (5) and (6) in Fig. 4 or line (3a) in Fig. 5). Then the protocol uses the Com functionality of $\mathcal{F}_{\mathbb{G}}^{ABB}$ to reveal the prover message (e.g. line (8) in Fig. 4 or line (3b) in Fig. 5). Both parties will then hash the revealed prover message (potentially with addition of a public input) to obtain hash values which will be treated as a challenge from the verifier in ordinary Bulletproofs (e.g. line (9) in Fig. 4 or line (3c) in Fig. 5). (Note that this corresponds to the Fiat-Shamir conversion of Bulletproofs to make these non-interactive.) Finally, the challenge will be used in the computation of subsequent prover messages. The protocol continues this until a full Bulletproof is obtained.

A key property here is that all hash values are computed over messages available to both parties in the clear (revealed in Com calls). Hence, the protocol can avoid computing the hash of ABB-stored values, which would have made the protocol prohibitively expensive to evaluate in practice. The only computations that need to be carried out on the ABB-stored values are modular arithmetic over $\mathbb{Z}_p$ and exponentiation over $\mathbb{G}$. The structure of the protocols furthermore highlights the usefulness of the Com functionality of $\mathcal{F}_{\mathbb{G}}^{ABB}$ which plays a crucial role in efficiently instantiating the protocols (see also Sect. 3.3).

Since the operations in the protocols consist of only calls of the functionality in $\mathcal{F}_{\mathbb{G}}^{ABB}$ or local computations by each party, the following theorem can easily be seen to hold.

Theorem 1. *The protocol* JointBulletproof *combined with a compiler protocol is a secure-with-abort protocol realizing the proof generation of the Bulletproof zk-SNARK for arithmetic circuits, in the $\mathcal{F}_{\mathbb{G}}^{ABB}$-hybrid model.*

Verification. The verification algorithm, VerifyAC, is identical to that of the ordinary Bulletproof zk-SNARK. Due to space limitations, the description is deferred to the full version.

3.3 Secure Realization

To realize our Bulletproof-based collaborative zk-SNARK presented in Sect. 3.2, it remains to securely realize the ABB functionality $\mathcal{F}_{\mathbb{G}}^{ABB}$ from Sect. 3.1. Crucially, to maintain the efficiency of the protocol presented in Sect. 3, the $\mathcal{F}_{\mathbb{G}}^{ABB}$

Public Input: $\mathsf{pp} = (g, h, \boldsymbol{g}, \boldsymbol{h})$, $\mathsf{stmt} = (\boldsymbol{V}, \boldsymbol{W}_{\mathrm{L}}, \boldsymbol{W}_{\mathrm{R}}, \boldsymbol{W}_{\mathrm{O}}, \boldsymbol{W}_{\mathrm{V}}, \boldsymbol{c})$, where $\boldsymbol{V} \in \mathbb{G}^m$, $\boldsymbol{W}_{\mathrm{L}}$, $\boldsymbol{W}_{\mathrm{R}}$, $\boldsymbol{W}_{\mathrm{O}} \in \mathbb{Z}_p^{Q \times n}$, $\boldsymbol{W}_{\mathrm{V}} \in \mathbb{Z}_p^{Q \times m}$, and $\boldsymbol{c} \in \mathbb{Z}_p^Q$.

Private Input (stored by $\mathcal{F}_{\mathbb{G}}^{ABB}$): $[\boldsymbol{a}_{\mathrm{L}}]$, $[\boldsymbol{a}_{\mathrm{R}}]$, $[\boldsymbol{a}_{\mathrm{O}}]$, $[v_1]$, …, $[v_m]$, $[\gamma_1]$, …, and $[\gamma_m]$, where $\boldsymbol{a}_{\mathrm{L}}, \boldsymbol{a}_{\mathrm{R}}, \boldsymbol{a}_{\mathrm{O}} \in \mathbb{Z}_p^n$ and $v_i, \gamma_i \in \mathbb{Z}_p$.

The Protocol:

1. Call Rand to obtain $[\alpha]$, $[\beta]$, $[\rho]$, $[\boldsymbol{s}_{\mathrm{L}}]$, and $[\boldsymbol{s}_{\mathrm{R}}]$, where α, β, $\rho \in \mathbb{Z}_p$ and $\boldsymbol{s}_{\mathrm{L}}$, $\boldsymbol{s}_{\mathrm{R}} \in \mathbb{Z}_p^n$.
2. Call Com to obtain $A_{\mathrm{I}} = h^{\alpha} \boldsymbol{g}^{\boldsymbol{a}_{\mathrm{L}}} \boldsymbol{h}^{\boldsymbol{a}_{\mathrm{R}}}$, $A_{\mathrm{O}} = h^{\beta} \boldsymbol{g}^{\boldsymbol{a}_{\mathrm{O}}}$, and $S = h^{\rho} \boldsymbol{g}^{\boldsymbol{s}_{\mathrm{L}}} \boldsymbol{h}^{\boldsymbol{s}_{\mathrm{R}}}$.
3. Locally compute $(y, z) \leftarrow H_2(\mathsf{pp}, \mathsf{stmt}, A_{\mathrm{I}}, A_{\mathrm{O}}, S)$.
4. Locally compute $\boldsymbol{y}^n \leftarrow (1, y, y^2, \ldots, y^{n-1})$ and $\boldsymbol{z}_{1\leq}^{Q+1} \leftarrow (z, z^2, z^3, \ldots, z^Q)$.
5. Call SMult and Add to obtain $[l](X) \leftarrow [\boldsymbol{a}_{\mathrm{L}}] \cdot X + [\boldsymbol{a}_{\mathrm{O}}] \cdot X^2 + \boldsymbol{y}^{-n} \circ (\boldsymbol{z}_{1\leq}^{Q+1} \cdot \boldsymbol{W}_{\mathrm{R}}) \cdot X + [\boldsymbol{s}_{\mathrm{L}}] \cdot X^3$ and $[r](X) \leftarrow \boldsymbol{y}^n \circ [\boldsymbol{a}_{\mathrm{R}}] \cdot X - \boldsymbol{y}^n + \boldsymbol{z}_{1\leq}^{Q+1} \cdot (\boldsymbol{W}_{\mathrm{L}} \cdot X + \boldsymbol{W}_{\mathrm{O}}) + \boldsymbol{y}^n \circ [\boldsymbol{s}_{\mathrm{R}}] \cdot X^3$.
6. Call Mult and Add to receive $[t](X) = \sum_{i=1}^{6} [t_i] \cdot X^i \leftarrow \langle [l](X), [r](X) \rangle$.
7. Call Rand to obtain $[\tau_i]$ $(i \in \{1, 3, 4, 5, 6\})$, where $\tau_i \in \mathbb{Z}_p$.
8. Call Com to obtain $T_i = g^{t_i} h^{\tau_i}$ $(i \in \{1, 3, 4, 5, 6\})$.
9. Locally compute $x \leftarrow H_1(\mathsf{pp}, \mathsf{stmt}, A_{\mathrm{I}}, A_{\mathrm{O}}, S, T_1, T_3, T_4, T_5, T_6)$.
10. Call SMult and Add to compute $[\boldsymbol{l}] \leftarrow [l](x)$ and $[\boldsymbol{r}] \leftarrow [r](x)$.
11. Call Mult and Add to receive $[\hat{t}] \leftarrow \langle [\boldsymbol{l}], [\boldsymbol{r}] \rangle$.
12. Call SMult and Add to compute $[\tau_{\mathrm{x}}] \leftarrow \sum_{i \in \{1,3,4,5,6\}} [\tau_i] \cdot x^i + x^2 \cdot \langle \boldsymbol{z}_{1\leq}^{Q+1}, \boldsymbol{W}_{\mathrm{V}} \cdot [\boldsymbol{\gamma}] \rangle$ and $[\mu] \leftarrow [\alpha] \cdot x + [\beta] \cdot x^2 + [\rho] \cdot x^3$.
13. Call Output to receive τ_{x}, μ, $\hat{t}$, $\boldsymbol{l}$, and $\boldsymbol{r}$.
14. Locally compute $h_i' \leftarrow (h_i)^{y^{-i+1}}$ $(i \in \{1, \ldots, n\})$, $W_{\mathrm{L}} \leftarrow (\boldsymbol{h}')^{\boldsymbol{z}_{1\leq}^{Q+1} \cdot \boldsymbol{W}_{\mathrm{L}}}$, $W_{\mathrm{R}} \leftarrow \boldsymbol{g}^{\boldsymbol{y}^{-n} \circ (\boldsymbol{z}_{1\leq}^{Q+1} \cdot \boldsymbol{W}_{\mathrm{R}})}$, $W_{\mathrm{O}} \leftarrow (\boldsymbol{h}')^{\boldsymbol{z}_{1\leq}^{Q+1} \cdot \boldsymbol{W}_{\mathrm{O}}}$, and $P \leftarrow (A_{\mathrm{I}})^x \cdot (A_{\mathrm{O}})^{x^2} \cdot (\boldsymbol{h}')^{-\boldsymbol{y}^n} \cdot (W_{\mathrm{L}})^x \cdot (W_{\mathrm{R}})^x \cdot W_{\mathrm{O}} \cdot S^{x^3}$.
15. Execute the sub-protocol $\mathsf{JointProveIP}$ on public input $\boldsymbol{g}$, $\boldsymbol{h}$, $P \cdot h^{-\mu}$, $\hat{t}$, and $\mathsf{aux} = (\mathsf{pp}, \mathsf{stmt}, A_{\mathrm{I}}, A_{\mathrm{O}}, S, T_1, T_3, T_4, T_5, T_6)$, and private input $[\boldsymbol{l}]$ and $[\boldsymbol{r}]$, and then receive π_{IP} as the result.

Output: $\pi = (A_{\mathrm{I}}, A_{\mathrm{O}}, S, T_1, T_3, T_4, T_5, T_6, \tau_{\mathrm{x}}, \mu, \hat{t}, \pi_{\mathsf{IP}})$.

Fig. 4. Protocol $\mathsf{JointBulletproof}$ for jointly generating Bulletproof for arithmetic circuits.

realization must itself be efficient. Note that our goal is to obtain a collaborative zk-SNARK satisfying (malicious) security with abort which is achieved by a secure-with-abort realization of the $\mathcal{F}_{\mathbb{G}}^{ABB}$.

Our starting point is the SPDZ protocol [15] which is a secure-with-abort protocol realizing the standard arithmetic functionalities (as SPDZ is based on additive secret sharing, we will in the following use the notation $[\cdot]$ to denote additive sharing as opposed to a value stored in an abstract ABB). However, SPDZ by itself does not provide an efficient way to instantiate the Com functionality of $\mathcal{F}_{\mathbb{G}}^{ABB}$ which is central to our protocol in Sect. 3. To efficiently realize Com, we make use of an insight by Smart and Alaoui [40] who showed that the SPDZ protocol that can be extended to deal with operations for cyclic groups

Public Input: $\boldsymbol{g}, \boldsymbol{h} \in \mathbb{G}^n$, $P \in \mathbb{G}$, $c \in \mathbb{Z}_p$, and $\mathsf{aux} \in \{0,1\}^*$.

Private Input (stored by $\mathcal{F}_{\mathbb{G}}^{ABB}$): $[\boldsymbol{a}], [\boldsymbol{b}] \in \mathbb{Z}_p^n$.

The Protocol:

1. Locally compute $(x, u_0) \leftarrow H_3(\boldsymbol{g}, \boldsymbol{h}, P, c, \mathsf{aux})$.
2. Locally set $\boldsymbol{g}^{(n)} \leftarrow \boldsymbol{g}$, $\boldsymbol{h}^{(n)} \leftarrow \boldsymbol{h}$, $u \leftarrow u_0^x$, and $P^{(n)} \leftarrow P \cdot u_0^{x \cdot c}$, and (implicitly) let $\boldsymbol{a}^{(n)} \leftarrow \boldsymbol{a}$ and $\boldsymbol{b}^{(n)} \leftarrow \boldsymbol{b}$.
3. Repeat the followings for $m = n, n/2, n/2^2, \ldots, 2$.
   (a) Call Mult and Add to receive $[c_{\mathrm{L}}^{(m)}] \leftarrow [\langle \boldsymbol{a}_{\leq m/2}^{(m)}, \boldsymbol{b}_{m/2+1\leq}^{(m)} \rangle]$ and $[c_{\mathrm{R}}^{(m)}] \leftarrow [\langle \boldsymbol{a}_{m/2+1\leq}^{(m)}, \boldsymbol{b}_{\leq m/2}^{(m)} \rangle]$.
   (b) Call Com to receive $L^{(m)} = (\boldsymbol{g}_{m/2+1\leq}^{(m)})^{\boldsymbol{a}_{\leq m/2}^{(m)}} (\boldsymbol{h}_{\leq m/2}^{(m)})^{\boldsymbol{b}_{m/2+1\leq}^{(m)}} u^{c_{\mathrm{L}}^{(m)}}$ and $R^{(m)} = (\boldsymbol{g}_{\leq m/2}^{(m)})^{\boldsymbol{a}_{m/2+1\leq}^{(m)}} (\boldsymbol{h}_{m/2+1\leq}^{(m)})^{\boldsymbol{b}_{\leq m/2}^{(m)}} u^{c_{\mathrm{R}}^{(m)}}$.
   (c) Locally compute $x^{(m)} \leftarrow H_1(L^{(n)}, R^{(n)}, L^{(n/2)}, R^{(n/2)}, \ldots, L^{(m)}, R^{(m)}, \boldsymbol{g}, \boldsymbol{h}, P, c, \mathsf{aux})$.
   (d) Locally compute $\boldsymbol{g}^{(m/2)} \leftarrow (\boldsymbol{g}_{\leq m/2}^{(m)})^{(x^{(m)})^{-1}} (\boldsymbol{g}_{m/2+1\leq}^{(m)})^{x^{(m)}}$, $\boldsymbol{h}^{(m/2)} \leftarrow (\boldsymbol{h}_{\leq m/2}^{(m)})^{x^{(m)}} (\boldsymbol{h}_{m/2+1\leq}^{(m)})^{(x^{(m)})^{-1}}$, and $P^{(m/2)} \leftarrow (L^{(m)})^{(x^{(m)})^2} P^{(m)} (R^{(m)})^{(x^{(m)})^{-2}}$.
   (e) Call SMult and Add to receive $[\boldsymbol{a}^{(m/2)}] \leftarrow [\boldsymbol{a}_{\leq m/2}^{(m)}] \cdot x^{(m)} + [\boldsymbol{a}_{m/2+1\leq}^{(m)}] \cdot (x^{(m)})^{-1}$ and $[\boldsymbol{b}^{(m/2)}] \leftarrow [\boldsymbol{b}_{\leq m/2}^{(m)}] \cdot (x^{(m)})^{-1} + [\boldsymbol{b}_{m/2+1\leq}^{(m)}] \cdot x^{(m)}$.
4. Call Output to receive $\boldsymbol{a}^{(1)}$ and $\boldsymbol{b}^{(1)}$.

Output: $\pi_{\mathsf{IP}} = (L^{(n)}, R^{(n)}, L^{(n/2)}, R^{(n/2)}, \ldots, L^{(2)}, R^{(2)}, \boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)})$.

Fig. 5. Protocol $\mathsf{JointProveIP}$ for jointly generating Bulletproof for inner products.

over an elliptic curve whose order coincides with that of the underlying field of the SPDZ protocol. While Smart and Alaoui are concerned with implementing full elliptic curve circuit evaluation, the restricted Com functionality required in $\mathcal{F}_{\mathbb{G}}^{ABB}$ is comparably simple and can be implemented very efficiently. Specifically, recall that the main idea of SPDZ is to let each party hold a share of a global MAC key $k \in \mathbb{Z}_p$ i.e. party i holds $[k]_i$ such that $\sum [k]_i = k$. Then each value x (stored in the ABB) is shared among all parties where each share is of the form $([x]_i, [m]_i)$ and $\sum [x]_i = x$ and $\sum [m]_i = k \cdot x$. The parties will then perform any (arithmetic) computation over the shares (consuming multiplication tuples in the process) while maintaining the above format of shares. Finally, when the computation is done, the parities will firstly check correctness of any value opened during the computation and then the computation result by checking $x \cdot k - m = 0$ (for each value x). This approach readily extends to our restricted Com functionality. Specifically, given values $a_1, \ldots, a_\ell$ shared among the parties as $([a_i], [m_i])$ where $m_i = ka_i$, each party can *locally* compute

$$X_i = g_1^{[a_1]_i} \cdots g_\ell^{[a_\ell]_i} \quad \text{and} \quad M_i = g_1^{[m_1]_i} \cdots g_\ell^{[m_\ell]_i}.$$

Note that since the characteristic of the field $\mathbb{Z}_p$ the values a_i are (additively) shared over is the same as the order p of the group elements g_i, we have that $X = \prod_i X_i = g_1^{a_1} \cdots g_\ell^{a_\ell} = \mathsf{Com}([a]_1, \ldots, [a]_\ell, g_1, \ldots, g_\ell)$. Hence, to open X, each party simply broadcasts X_i. Finally, to check an opened commitment X, the parties check that $X^k - M = 0$ where $M = \prod_i M_i$. It is relatively easy to see that this approach inherits the security properties of SPDZ.

Lastly, the additional non-linear functionalities in $\mathcal{F}_{\mathbb{G}}^{ABB}$ can be realized via standard generic techniques. Specifically, equality and comparison can be efficiently computed by an appropriate combination of addition, multiplication, and output operations supported by a standard ABB functionality, as shown by Nishide and Ohta [33]; Max and argmax can be easily realized using comparison [25]; Bit-decomposition can be also computed using the protocol of [33].

Based on the above, we obtain the following result.

Theorem 2. *There exists a secure realization of $\mathcal{F}_{\mathbb{G}}^{ABB}$ based on the above described extension of the SPDZ protocol [15].*

Combining the efficient realization of $\mathcal{F}_{\mathbb{G}}^{ABB}$ with the protocol from Sect. 3.2 provides us with a secure-with-abort protocol for the joint computation of Bulletproofs. As shown by Ozdemir and Boneh [34,35], it is fairly straightforward to show that if a (single-prover) zk-SNARK system is zero-knowledge and has knowledge soundness, and the prover algorithm is computed by a secure-with-abort MPC protocol against t corrupted parties so that each party's private input is a witness share (where the concatenation of all parties' witness shares constitutes a witness), then the resulting protocol is a secure-with-abort collaborative zk-SNARK satisfying t-zero-knowledge. Hence, combined with Theorem 1, we obtain the following theorem.

Theorem 3. *Our collaborative Bulletproof protocol instantiated with the extended SPDZ protocol [15] is a secure-with-abort collaborative zk-SNARK.*[5]

4 Privacy-Preserving Verifiable CNNs

In this section, we will first introduce our formal definition of a privacy-preserving verifiable CNN in Sect. 4.1. Then, we present our proposed privacy-preserving verifiable CNN in Sect. 4.2.

4.1 Formal Definition

Our definition of a privacy-preserving verifiable CNN will provide strong privacy guarantees and in particular ensure that no information regarding the CNN model M, the input x, or the obtained classification will leak to any other party. To achieve this, we require a privacy-preserving verifiable CNN to have similar security properties to publicly-auditable 2-party computation [2]. Informally, a

[5] If a semi-honest MPC protocol for $\mathcal{F}_{\mathbb{G}}^{ABB}$ is used instead of SPDZ, our protocol is still guaranteed to achieve t-zero-knowledge in the presence of semi-honest parties.

publicly-auditable multi-party computation is an extension of a secure multi-party computation protocol that, in addition to computing a functionality, can generate a publicly verifiable proof that the output of the protocol is correct with respect to commitments to each party's input. Ozdemir and Boneh [34, 35] gave a definition of publicly auditable multi-party computation based on collaborative ZK, and our definition of privacy-preserving verifiable CNN follows their definitional approach, but with modifications to deal with a probabilistic functionality[6] and to capture the CNN setting we consider here.

Formally, a privacy-preserving verifiable CNN is associated with some commitment scheme $\mathsf{Com} = (\mathsf{Setup}_{\mathsf{Com}}, \mathsf{Commit})$, and consists of $(\mathsf{Setup}, \Pi, \mathsf{Verify})$ each of whose syntax is defined as follows:

- Setup is the setup algorithm that takes a security parameter 1^λ as input, and outputs a public parameter pp.
- Π is an interactive protocol between two parties P_1 (holding a CNN model) and P_2 (holding a CNN input). The protocol is executed using three types of inputs:
  - P_1's private inputs: a CNN model M and randomness r_M.
  - P_2's private inputs: a CNN input x and randomness r_x.
  - public common inputs: the CNN structure $\mathcal{S}$, a public parameter $\mathsf{pp}_{\mathsf{Com}}$ of the underlying commitmemnt scheme Com, and commitments com_M and com_x (which are supposedly generated as $\mathsf{com}_M = \mathsf{Commit}(\mathsf{pp}_{\mathsf{Com}}, M, r_M)$ and $\mathsf{com}_x = \mathsf{Commit}(\mathsf{pp}_{\mathsf{Com}}, x, r_x)$, respectively).

  As the results of an execution of the protocol, P_1 outputs a commitment com_y (for the CNN result y) and a proof π, and P_2 outputs a CNN output y, a randomness r_y, a commitment com_y, and a proof π.
- Verify is the verification algorithm that takes pp, commitments $(\mathsf{com}_M, \mathsf{com}_x, \mathsf{com}_y)$, and a proof π as input, and outputs either $\top$ (accept) or $\bot$ (reject).

Definition 3. *A privacy-preserving verfiable CNN* $(\mathsf{Setup}, \Pi, \mathsf{Verify})$ *associated with* $\mathsf{Com} = (\mathsf{Setup}_{\mathsf{Com}}, \mathit{Commit})$ *is* secure *if it satisfies the following two properties:*

- *Let* Π_y *denote* Π*, in which the proof* π *is excluded from each party's output. Then,* Π_y *is a secure-with-abort 2-party protocol for the functionality* $\mathcal{F}_{\mathsf{Com}}^{pCNN}$.
- *Let* Π_π *denote* Π*, in which only the proof* π *is treated as the output (of both parties). Then,* $(\mathsf{Setup}, \Pi_\pi, \mathsf{Verify})$ *associated with* Com *satisfies the requirements of a collaborative ZK protocol (Definition 2) for the following relation* $\mathcal{R}$*:*

[6] The definition of publicly-auditable computation in [34, Appendix D] is for a deterministic functionality.

$$\Big\{ \underbrace{\Big(\mathcal{S}, \mathsf{pp}_{\mathsf{Com}}, \mathsf{com}_M, \mathsf{com}_x, \mathsf{com}_y\Big)}_{common\ input/output}, \Big(\underbrace{(M, r_M)}_{P_1\text{'s input/output}}, \underbrace{(x, r_x, y, r_y)}_{P_2\text{'s input/output}} \Big) \Big\} \in \mathcal{R}$$

$$\iff$$

$$\begin{aligned} \mathsf{com}_M = \mathit{Commit}(\mathsf{pp}_{\mathsf{Com}}, M, r_M) \wedge \mathsf{com}_x = \mathit{Commit}(\mathsf{pp}_{\mathsf{Com}}, x, r_x) \\ \wedge\, \mathsf{com}_y = \mathit{Commit}(\mathsf{pp}_{\mathsf{Com}}, y, r_y) \wedge y = \mathit{EvalCNN}(\mathcal{S}, M, x). \end{aligned} \quad (4)$$

Furthermore, we say that a privacy-preserving verifiable CNN is semi-honest secure *if Π_y is semi-honest secure and the* 1*-zero-knowledge property of* (Setup, Π_π, Verify) *is replaced with* 1*-zero-knowledge property in the presence of semi-honest parties.*

4.2 Construction

Our construction of a privacy-preserving verifiable CNN $(\mathsf{Setup}, \Pi, \mathsf{Verify})$ is based on the collaborative zk-SNARK presented in Sect. 3. In fact, we directly use the Setup and Verify algorithms from Sect. 3 as the corresponding Setup and Verify algorithms for the privacy-preserving verifiable CNN, respectively.

The interactive proof generation protocol itself will be based on an "augmented" CNN computation (which we will describe with respect to the ABB functionality presented in Sect. 3.1). The augmented CNN computation not only computes the classification of an input, but also intermediate variables, which will provide the parties with a witness for proving the correctness of the classification via our collaborative zk-SNARK. We will denote this process as

$$([y], [w]) \leftarrow \mathsf{JointEvalCNN}(\mathcal{S}, [M], [x])$$

using the notation in Sect. 2 where M is the CNN model, x is the CNN input, y is the result of the classification and w is the witness generated in this process (note that the process is deterministic). This process can be divided further into steps corresponding to each layer of the CNN:

$$\begin{aligned} &([y_1], [w_1]) \leftarrow \mathsf{JointEvalLayer1}([M], [x]) \\ &([y_2], [w_2]) \leftarrow \mathsf{JointEvalLayer2}([M], [y_1], [w_1]) \\ &\dots \\ &([y_N], [w_N]) \leftarrow \mathsf{JointEvalLayerN}([M], [y_{N-1}], [w_{N-1}]) \end{aligned}$$

where each algorithm only computes a single layer in the CNN structure $\mathcal{S}$, and y_i and w_i are the output and corresponding witness of layer i, respectively. Here we assume that layer i appends its new witness variables to $[w_{i-1}]$ and outputs this concatenation as $[w_i]$ such that the final witness $[w_N]$ contains all witness for the entire classification. In the following, we explain protocols that perform the

computations that constitute each layer, e.g., an affine relation between weights, the ReLU relation, or a max pooling relation. Parties execute multiple instances of these protocols to complete the computation of each layer.

This process is finalized by computing a commitment com_y to the final classification result y_N (the corresponding randomness r_{com} is given to the party holding x). Once this step is completed, the parties will be able to use the final witness $[w_N]$ (which contains the witnesses for all prior layers) to jointly run the JointBulletproof from Sect. 3.2 to obtain a proof for the computed classification.

In the following, we present protocol instantiations for the initialization and each CNN layer described in Sect. 2 which will allow the parties to complete the above outlined steps. Note that we will use fixed-point computation to represent all arithmetic computations done as part of CNN classification. Specifically, a rational number $x_0 + 2^{-d}x_1 \in \mathbb{Q}$ where $x_0 \in \{-2^\ell, \ldots, 2^\ell - 1\}$ and $x_1 \in \{0, \ldots, 2^d - 1\}$ is represented by the integer $2^d x_0 + x_1 \in \mathbb{Z}_p$. Note also that when multiplication is done between two numbers in fixed-point representation, truncation of the last d bits is required to maintain the correct representation of the result.

Initialization. The parties will have to generate witness vectors $[\boldsymbol{a}_\mathrm{L}]$, $[\boldsymbol{a}_\mathrm{R}]$,$[\boldsymbol{a}_\mathrm{O}]$, $[v_1], \ldots, [v_m]$, and $[\gamma_1], \ldots, [\gamma_m]$, which satisfy Bulletproof's statement Eq. (3). Here, $\boldsymbol{a}_\mathrm{L}$, $\boldsymbol{a}_\mathrm{R}$, and $\boldsymbol{a}_O$ should satisfy the Hadamard product relation $\boldsymbol{a}_\mathrm{L} \circ \boldsymbol{a}_\mathrm{R} = \boldsymbol{a}_\mathrm{O}$ and the v_j's and γ_j's are the input values and the randomness used to commit to these in the commitments $V_j = g^{v_j} h^{\gamma_j}$. The parties receive (M, r_M) and (x, r_x) as private input, respectively, as well as public input $(\mathsf{com}_M, \mathsf{com}_x)$, where $\mathsf{com}_M = \mathsf{Commit}(\mathsf{pp}_{\mathsf{Com}}, M, r_M)$ and $\mathsf{com}_x = \mathsf{Commit}(\mathsf{pp}_{\mathsf{Com}}, x, r_x)$. The values $[v_1], \ldots, [v_m]$, and $[\gamma_1], \ldots, [\gamma_m]$ are initialized by the parties calling Input of $\mathcal{F}_{\mathbb{G}}^{ABB}$ on the model M, r_M, the input x and r_x. As described above, each of the following layer protocols will append appropriate witnesses to $[\boldsymbol{a}_\mathrm{L}]$, $[\boldsymbol{a}_\mathrm{R}]$, and $[\boldsymbol{a}_\mathrm{O}]$ and add linear relations to be proven among $\boldsymbol{a}_\mathrm{L}$, $\boldsymbol{a}_\mathrm{R}$, $\boldsymbol{a}_\mathrm{O}$, $\boldsymbol{v}$, and $\boldsymbol{c}$. The linear relations will be added by appending extra rows to the matrices $\boldsymbol{W}_\mathrm{L}$, $\boldsymbol{W}_\mathrm{R}$, $\boldsymbol{W}_\mathrm{O}$, $\boldsymbol{W}_\mathrm{V}$, and the vector $\boldsymbol{c}$ (see Eq. (3)). Note that to compute the layers in the model, the parties execute a set of protocols presented below sequentially. In this sequential execution, the output of a protocol is appended to the vector $\boldsymbol{a}_\mathrm{L}$ (as described above, each protocol appends elements to $\boldsymbol{a}_\mathrm{L}$, $\boldsymbol{a}_\mathrm{R}$, and $\boldsymbol{a}_\mathrm{O}$, some of which are intermediate variables and some of which are the output of the protocol). This appended output of the protocol is later used by subsequent protocols. The final classification result y and the randomness used for the commitment com_y computed in the finalization will be stored in the appropriate $[v_1], \ldots, [v_m]$, and $[\gamma_1], \ldots, [\gamma_m]$ positions. Note that some protocols will additionally require an index to know which part of the witness from the previous layer is used in the computation e.g. the ReLU function takes indices $(i, u) \in \{1, \ldots, n\} \times \{\mathrm{L}, \mathrm{R}, \mathrm{O}\}$ and assumes that the input to the ReLU function is stored at $\boldsymbol{a}_u[i]$.

Affine and Convolution Layers. Affine and convolution layers correspond to the computation of an inner product relation (note that average pooling corresponds to an affine layer). The computation is implemented via the JointEvalIP

protocol which is given shares of vectors $([\boldsymbol{a}_{u_1}[i_1]], \ldots, [\boldsymbol{a}_{u_m}[i_m]])$, and $([\boldsymbol{a}_{v_1}[j_1]], \ldots, [\boldsymbol{a}_{v_m}[j_m]])$ and a scalar $[\boldsymbol{a}_w[k]]$, and computes the inner product:

$$\boldsymbol{a} = \boldsymbol{a}_w[k] + \sum_{t=1}^{m} \boldsymbol{a}_{u_t}[i_t]\boldsymbol{a}_{v_t}[j_t]$$

where $\boldsymbol{a}_w[k]$ is a constant term that may be utilized by an affine layer and the result $\boldsymbol{a}$ will be appended to the vector $\boldsymbol{a}_{\mathrm{L}}$ during the protocol. The description of JointEvalIP is deferred to the full version, due to the page limitation.

Note that this protocol does not perform the truncation which would normally be required by the fixed-point multiplication. Instead, this truncation will be performed by the following ReLU protocol. Deferring this truncation improves efficiency. Specifically, the computation of an inner product requires multiple multiplications and thus multiple truncations, but if we defer the truncation to the ReLU proof, just a single truncation is sufficient for each inner product.

ReLU. The protocol JointEvalReLU computes the ReLU function. Namely, given as input a share $[\boldsymbol{a}_u[i]]$, the protocol computes

$$\boldsymbol{a} = \mathsf{ReLU}(\boldsymbol{a}_u[i])$$

where $\boldsymbol{a}$ will be appended to the vector $\boldsymbol{a}_{\mathrm{L}}$ during the protocol. Recall that ReLU computes the function

$$\mathsf{ReLU}(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x \geq 0 \end{cases}.$$

This is equivalently represented by bit decomposition:

$$\mathsf{ReLU}(x) = x_m \cdot \sum_{i=0}^{m-1} 2^i x_i$$

where $x + 2^m = \sum_{i=0}^{m} 2^i x_i$ with $x_0, \ldots, x_m \in \{0, 1\}$.

To implement this function, the protocol JointEvalReLU utilizes the subprotocol JointEvalRange which computes the bit decomposition of a shared integer in $\mathbb{Z}_p$. The name of the protocol JointEvalRange stems from the witness computed in the protocol is not only a bit decomposition, but can also be viewed as a witness for a range proof i.e. that the input falls into a certain range. The description of the protocols JointEvalReLU and JointEvalRange are deferred to the full version, due to the page limitation.

Note that this protocol truncates the input value by d bits. This truncation is realized by the summation in $\boldsymbol{a}_{\mathrm{R}}[n+1+2\ell+2d+1]$ (Here, the index $n+1+2\ell+2d+1$ comes from the following calculation: The variable n indicates the current number of the multiplication relations, $1+2\ell+2d$ comes from the range proof with $1+2\ell+2d$ bits, and the last 1 comes from an extra multiplication relation for proving the ReLU relation) in the protocol description which runs from $i = d$ to $2\ell + 2d - 1$, by which it truncates the least significant d bits.

Finalization. To define the output of a CNN, it is standard to use SoftMax to normalize the output of the last layer. However, since we are only interested in proving the obtained classification, it is sufficient to prove that some given public value is maximum in a given set of values.

The protocol $\mathsf{JointEvalFinalize}$ does exactly this. The protocol utilizes the following representation of the maximum relation $y = \max\{x_1, \ldots, x_m\}$: There exists a vector $(z_1, \ldots, z_m)$ satisfying that

$$\begin{aligned} &y = x_1 z_1 + \cdots + x_m z_m, \\ &z_1, \ldots, z_m \in \{0, 1\}, \\ &z_1 + \cdots + z_m = 1, \\ &y - x_1 \geq 0, \ldots, y - x_m \geq 0. \end{aligned}$$

The description of the protocol is deferred to the full version, due to the page limitation.

Proof Generation. Upon completion of the finalization described above, the parties simply invoke $\mathsf{JointBulletproof}$ (Fig. 4) from Sect. 3.2 using the generated witness $[\boldsymbol{a}_{\mathrm{L}}]$, $[\boldsymbol{a}_{\mathrm{R}}]$, $[\boldsymbol{a}_{\mathrm{O}}]$, $[v_1]$, …, $[v_m]$, and $[\gamma_1]$, …, $[\gamma_m]$ as input to collaboratively generate a Bulletproof π of correct classification.

Obtained Proof and Disclosure of Classification Information. Upon completion of the proof generation protocol, the party holding the input x, will obtain a proof π with respect to a commitment to the model com_M, a commitment to x and a commitment to the corresponding classification com_y. While this party can present π to a third-party verifier, the latter will not gain any information on the classification y by verifying π, as com_y hides y. This is insufficient in many applications. However, as the party holding x will receive the opening r_y to com_y, he will be able to disclose additional information regarding y. One option is simply to reveal both y and r_y, which would allow the verifier to check that y is indeed the correct classification result via the commitment scheme. However, a more fine-grained disclosure is possible. Note that com_y obtained in our verifiable CNN is simply a Pedersen commitment which allows the party holding x to produce an additional Bulletproof π_y showing any statement regarding y e.g. that y belong to a set $\mathcal{Y}$ of classification results. This proof can be generated independently and will be logarithmic in the size of the statement. By verifying both π and π_y, a third-party verifier will learn that y has been correctly computed with respect to M and x, and that y satisfies the additional statement shown by π_y, without learning any additional information on y.

4.3 Security

In the previous subsection, we have presented the procedures for our privacy-preserving verifiable CNN. It is not hard to see that during the protocol, the parties either call commands of $\mathcal{F}_{\mathbb{G}}^{ABB}$, local operations (including hashing on public values), or execute the proof generation protocol of our collaborative

Bulletproof protocol. Note that as the final result of an execution of our protocol, the party P_1 (holding a CNN model M) finally receives only public values (the commitments to the witnesses and the commitment to the evaluation result y of the CNN, and a proof of the collaborative Bulletproof); and the party P_2 (holding a CNN input x) will receive the same public values, as well as the CNN evaluation result y and its opening in the clear. Furthermore, the proof generation part of our protocol just invokes the proof generation of our collaborative Bulletproof protocol. Hence, we have the following theorems.

Theorem 4. *Let* $(\mathsf{Setup}, \Pi, \mathsf{Verify})$ *be our privacy-preserving verifiable CNN. Let* Π_y *denote the interactive protocol* Π *of our privacy-preserving verifiable CNN, such that the proof* π *is excluded from the output, and let* Π_π *denote* Π *such that the output is restricted to the proof* π*. Then,* $(\mathsf{Setup}, \Pi_y, \mathsf{Verify})$ *is a secure-with-abort protocol realizing* $\mathcal{F}_{\mathsf{Com}}^{pCNN}$*, in the* $\mathcal{F}_{\mathbb{G}}^{ABB}$*-hybrid model. Furthermore,* Π_π *is a collaborative zk-SNARK for proving the relation in Eq.* (4)*.*

Theorem 5. *Our privacy-preserving verifiable CNN instantiated with the SPDZ protocol [15] with the extension described in Sect. 3 is secure according to Definition 3.*

5 Implementation and Comparison

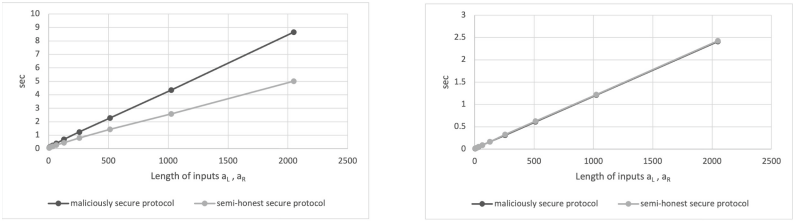
To measure the performance of our approach, we implemented our collaborative zk-SNARK protocol and estimated the performance of our verifiable CNN construction applied to the LeNet CNN [27] and the MNIST dataset [43]. In the following sections we provide the details of this as well as a comparison to related approaches.

5.1 Implementation of Collaborative Zk-SNARK

To evaluate the performance of our Bulletproof-based collaborative zk-SNARK, we made an implementation in Rust.[7] Specifically, we implemented the protocol using the elliptic curve library "curve25519-dalek."[8] This library provides group operations on the Edwards and Montgomery forms of Curve25519 and on the prime-order Ristretto group.

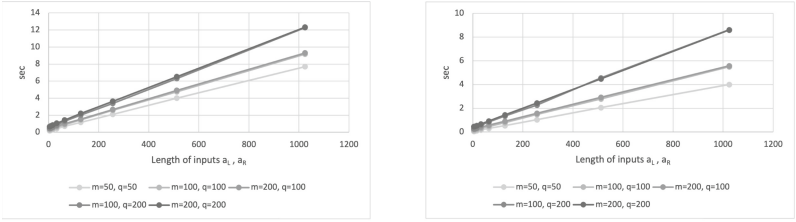
We implemented the protocol in the following setting: Firstly, in the protocol, we need to perform two-party multiplications, which require correlated randomness (i.e. Beaver triples). This correlated randomness is assumed to be generated in advance and made available to each party in a preprocessing phase. The cost of this phase can be estimated from [23] and is not included in the timing results presented below. Secondly, each party is implemented as a separate thread on a single server i.e. the implementation of each party is not parallelized. Finally, the communication between the two parties is simulated via the

[7] https://www.rust-lang.org/.
[8] https://doc.dalek.rs/curve25519_dalek/index.html.



(a) Proving time. (b) Verification time.

Fig. 6. Experimental timing results for proofs for inner products.



(a) Proving time. (b) Verification time.

Fig. 7. Experimental timing results for proofs for arithmetic circuits (malicious security).

"constrained-connection" library[9] set up to simulate a 1 Gbps connection with a round-trip-time of 0.5 ms. All experiments were performed on an Intel i5-7500 CPU @ 3.40 GHz and 16 GB of RAM.

Figure 6 shows the performance of both semi-honest and maliciously secure versions of the sub-protocol JointProveIP (Fig. 5) and its corresponding verification algorithm. (The verification algorithm is exactly that of the ordinary Bulletproofs [6] for inner product relations, and will be given in the full version.) These are for proving and verifying an inner product relation and the horizontal axis "Length of inputs a_L, a_R" corresponds to the dimension of the vectors in the inner product relation. As the figures show, the processing times increase linearly with the length of the input vectors. Furthermore, note that malicious security is obtained at roughly twice the cost of semi-honest security.

Figure 7 shows the performance of the maliciously-secure proof generation protocol JointBulletproof (Fig. 4) and its verification algorithm of our collaborative zk-SNARK for arithmetic circuits. (As above, the verification algorithm is exactly that of the Bulletproofs [6], and will be given in the full version.) Here, the horizontal axis "Length of inputs a_L, a_R" corresponds to the number of the multiplications in the proven arithmetic circuit. The parameter q is the number of the additive relations in the proven arithmetic circuit. The parameter m is the size of the committed message, which is, in the CNN application, the sum of

[9] https://docs.rs/constrained-connection/.

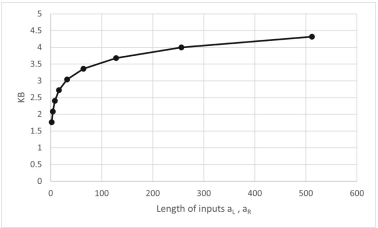


Fig. 8. Proof size of arithmetic circuit proofs.

the sizes of the CNN model and the CNN input. The figures show the processing time when the parameters n, m, and q in the protocol are varied, respectively.

Figure 8 shows the proof size of the protocol. (Note that the proof size is irrespective of whether it is computed by the malicious secure protocol or the semi-honest secure one.) According to the description of the protocol, the proof size depends only on n. This measurement confirms this.

5.2 Performance Estimate for Proof System for CNN

We will now discuss performance estimates when our protocol is applied to a CNN. For this performance estimation, we use the MNIST dataset [43]. MNIST is a dataset of hand-written digits, and the images are of size $28 \times 28 \times 1$. We use the LeNet network [27] which consists of two convolution layers, two pooling layers, and three fully connected layers. Note that for the purpose of comparison to vCNN and zkCNN, we use average pooling. The parameters of each layer are as follows:

- *Convolution:* Filter Size = 5×5, Stride = 1, Channels = 6
- *Average Pooling:* Filter Size = 2×2, Stride = 2, Channels = 6
- *Convolution:* Filter Size = 5×5, Stride = 1, Channels = 16
- *Average Pooling:* Filter Size = 2×2, Stride = 2, Channels = 16
- *Fully-Connected 1:* Input Size =400, Output Size = 120
- *Fully-Connected 2:* Input Size =120, Output Size = 84
- *Fully-Connected 3:* Input Size =84, Output Size = 10

When applying our protocol for an arithmetic circuit to the classification task of the above model, the parameters in the scheme are $n = m = 2^{19} = 524288$ and $Q = 611878$. We estimate the processing time of the protocol with these parameters based on the measurements in the previous section. As a result, the classification task of the above model takes about 2.9 h, the total communication cost is 236 MB, and the proof size is 7.68 KiB. Finally, based on [23] we estimate that 10 millions multiplications are needed in the on-line protocol and thus 1.7 h are needed for the off-line preprocessing.

	Algorithm	Model Privacy	Input Privacy	Comm.	Proof Size	Prover Time
Tensorflow	standalone	○	○	-	-	0.042ms
Groth	standalone	●	○	-	0.12 KiB	1.5h
vCNN	standalone	●	○	-	0.34 KiB	5.5s
zkCNN	standalone	●	○	-	63.6 KiB	0.5s
Ours	interactive	●	●	236 MB	7.68 KiB	2.9h

Fig. 9. Comparison of our verifiable CNN to related approaches. ● means the given property is achieved whereas ○ means this is not the case. The prover time for vCNN and zkCNN are from [30], and the time for Groth from [28] (see Sects. 5.1 and 5.3 for more details on all execution environments).

5.3 Comparison

Figure 9 shows a comparison between our verifiable CNN and related approaches. We stress that our verifiable CNN is fundamentally different from the related approaches shown in Fig. 9 in that it is an *interactive* protocol, which is required to obtain the strong notion of privacy considered in this paper, whereas the related approachs are all standalone algorithms executed locally by a single party. The results for Tensorflow[10] were obtained by classifying the full MNIST test set of 10000 samples in our local execution environment[11] and computing the average time for a single sample. The results for vCNN and zkCNN are from [30] and obtained on an AMD EPYC 7R32 64-Core CPU, whereas the results for the naive application of the Groth zk-SNARK [21] are from [28] and obtained on an quad-core Intel i5 CPU@3.4GHz (similar to our execution environment). While the measurement of each protocol uses a different computation environment and setup, the hardware differences are not significant for this comparison and the results remain useful for obtaining an overview of the performance of the protocols.

We note that compared to the plain Tensorflow computation, the fastest scheme providing model privacy, zkCNN, is orders of magnitude slower. Compared to zkCNN, our protocol is likewise orders of magnitude slower, but simultaneously provides model and input privacy which zkCNN cannot provide as it is a non-interactive standalone algorithm. This illustrates that protecting the privacy of the CNN input is a challenging task. However, we also note that our scheme is within a factor of two of the prover time for the Groth zk-SNARK when this is straightforwardly applied to ch1LeNet, despite the Groth zk-SNARK being a standalone algorithm. Finally, we note that the proof size of our approach is roughly an order of magnitude smaller than zkCNN, and an order of magnitude larger than vCNN.

These results highlight that it is feasible to provide the stronger notion of privacy we have introduced in this paper, which simultaneously protects the privacy of both CNN model and input. We stress that our implementation is a proof of concept only, and we believe that there is a lot of room for improvement in terms of prover running time by optimizing the implementation.

[10] https://www.tensorflow.org/.
[11] MacBook Pro M2 Pro.

Acknowledgement. The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. This work was partially supported by JST CREST Grant Number JPMJCR22M1 and JSPS KAKENHI Grant Number JP18K18055, Japan.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 2087–2104. ACM Press (2017). https://doi.org/10.1145/3133956.3134104
2. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. In: Abdalla, M., Prisco, R.D. (eds.) SCN 14. LNCS, vol. 8642, pp. 175–196. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10879-7_11
3. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4
4. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 327–357. Springer, Cham (2016). https://doi.org/10.1007/978-3-662-49896-5_12
5. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 483–512. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_17
6. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press (2018). https://doi.org/10.1109/SP.2018.00020
7. Byali, M., Chaudhari, H., Patra, A., Suresh, A.: FLASH: fast and robust framework for privacy-preserving machine learning. Proc. Privacy Enhanc. Technol. **2020**(2), 459–480 (2020). https://doi.org/10.2478/popets-2020-0036
8. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000). https://doi.org/10.1007/s001459910006
9. Chandran, N., Gupta, D., Rastogi, A., Sharma, R., Tripathi, S.: EzPC: programmable, efficient, and scalable secure two-party computation for machine learning. Cryptology ePrint Archive, Report 2017/1109 (2017). https://eprint.iacr.org/2017/1109
10. Chaudhari, H., Choudhury, A., Patra, A., Suresh, A.: ASTRA: high throughput 3pc over rings with application to secure prediction. In: Sion, R., Papamanthou, C. (eds.) Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW@CCS 2019, London, 11 November 2019, pp. 81–92. ACM (2019). https://doi.org/10.1145/3338466.3358922
11. Chaudhari, H., Rachuri, R., Suresh, A.: Trident: efficient 4PC framework for privacy preserving machine learning. In: NDSS 2020. The Internet Society (2020)
12. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_26

13. Chiesa, A., Ojha, D., Spooner, N.: Fractal: post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 769–793. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_27
14. Damgård, I., Nielsen, J.B.: Universally composable efficient multiparty computation from threshold homomorphic encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 247–264. Springer, Cham (2003). https://doi.org/10.1007/978-3-540-45146-4_15
15. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Cham (2012). https://doi.org/10.1007/978-3-642-32009-5_38
16. Dayama, P., Patra, A., Paul, P., Singh, N., Vinayagamurthy, D.: How to prove any NP statement jointly? Efficient distributed-prover zero-knowledge protocols. PoPETs **2022**(2), 517–556 (2022). https://doi.org/10.2478/popets-2022-0055
17. Feng, B., Qin, L., Zhang, Z., Ding, Y., Chu, S.: ZEN: An optimizing compiler for verifiable, zero-knowledge neural network inferences. Cryptology ePrint Archive, Report 2021/087 (2021). https://eprint.iacr.org/2021/087
18. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019). https://eprint.iacr.org/2019/953
19. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016)
20. Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Cambridge University Press, Cambridge (2004)
21. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
22. Kang, D., Hashimoto, T., Stoica, I., Sun, Y.: Scaling up trustless DNN inference with zero-knowledge proofs. arXiv preprint arXiv:2210.08674 (2022)
23. Keller, M., Orsini, E., Scholl, P.: MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 830–842. ACM Press (2016). https://doi.org/10.1145/2976749.2978357
24. Kitai, H., et al.: MOBIUS: model-oblivious binarized neural networks. IEEE Access **7**, 139021–139034 (2019). https://doi.org/10.1109/ACCESS.2019.2939410
25. Knott, B., Venkataraman, S., Hannun, A.Y., Sengupta, S., Ibrahim, M., van der Maaten, L.: Crypten: secure multi-party computation meets machine learning. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, pp. 4961–4973 (2021)
26. Koti, N., Pancholi, M., Patra, A., Suresh, A.: SWIFT: super-fast and robust privacy-preserving machine learning. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021, pp. 2651–2668. USENIX Association (2021)
27. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791

28. Lee, S., Ko, H., Kim, J., Oh, H.: vCNN: verifiable convolutional neural network. Cryptology ePrint Archive, Report 2020/584 (2020). https://eprint.iacr.org/2020/584
29. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via MiniONN transformations. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 619–631. ACM Press (2017). https://doi.org/10.1145/3133956.3134056
30. Liu, T., Xie, X., Zhang, Y.: zkCNN: zero knowledge proofs for convolutional neural network predictions and accuracy. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021, pp. 2968–2985. ACM Press (2021). https://doi.org/10.1145/3460120.3485379
31. Mohassel, P., Rindal, P.: ABY^3: a mixed protocol framework for machine learning. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018, pp. 35–52. ACM Press (2018). https://doi.org/10.1145/3243734.3243760
32. Mohassel, P., Zhang, Y.: SecureML: a system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, pp. 19–38. IEEE Computer Society Press (2017). https://doi.org/10.1109/SP.2017.12
33. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 343–360. Springer, Cham (2007). https://doi.org/10.1007/978-3-540-71677-8_23
34. Ozdemir, A., Boneh, D.: Experimenting with collaborative zk-SNARKs: zero-knowledge proofs for distributed secrets. Cryptology ePrint Archive, Report 2021/1530 (2021). https://eprint.iacr.org/2021/1530
35. Ozdemir, A., Boneh, D.: Experimenting with collaborative zk-SNARKs: zero-knowledge proofs for distributed secrets. In: Butler, K.R.B., Thomas, K. (eds.) USENIX Security 2022, pp. 4291–4308. USENIX Association (2022)
36. Patra, A., Suresh, A.: BLAZE: Blazing fast privacy-preserving machine learning. In: NDSS 2020. The Internet Society (2020)
37. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: a hybrid secure computation framework for machine learning applications. In: Kim, J., Ahn, G.J., Kim, S., Kim, Y., López, J., Kim, T. (eds.) ASIACCS 18, pp. 707–721. ACM Press (2018)
38. Rouhani, B.D., Riazi, M.S., Koushanfar, F.: Deepsecure: scalable provably-secure deep learning. In: Proceedings of the 55th Annual Design Automation Conference (DAC 2018), pp. 2:1–2:6. ACM (2018). https://doi.org/10.1145/3195970.3196023
39. Setty, S.: Spartan: efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Report 2019/550 (2019). https://eprint.iacr.org/2019/550
40. Smart, N.P., Talibi Alaoui, Y.: Distributing any elliptic curve based protocol. In: Albrecht, M. (ed.) 17th IMA International Conference on Cryptography and Coding. LNCS, vol. 11929, pp. 342–366. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35199-1_17
41. Wagh, S., Gupta, D., Chandran, N.: SecureNN: 3-party secure computation for neural network training. PoPETs **2019**(3), 26–49 (2019). https://doi.org/10.2478/popets-2019-0035
42. Weng, J., Weng, J., Tang, G., Yang, A., Li, M., Liu, J.N.: pvcnn: privacy-preserving and verifiable convolutional neural network testing (2022). https://arxiv.org/abs/2201.09186
43. LeCun, Y., Corinna Cortes, C.J.B.: The ch1MNIST database of handwritten digits (2010). http://yann.lecun.com/exdb/mnist/