Embedded Proofs for Verifiable Neural Networks

Hervé Chabanne[1,3], Julien Keuffer[1,2], and Refik Molva[2]

[1] Idemia, Issy-les-Moulineaux, France
[2] Eurecom, Biot, France
[3] Telecom ParisTech, Paris, France
{herve.chabanne,julien.keuffer}@morpho.com
refik.molva@eurecom.fr

Abstract. The increasing use of machine learning algorithms to deal with large amount of data and the expertise required by these algorithms lead users to outsource machine learning services. This raises a trust issue about their result when executed in an untrusted environment. Verifiable computing (VC) tackles this issue and provides computational integrity for an outsourced computation, although the bottleneck of state of the art VC protocols is the prover time. In this paper, we design a VC protocol tailored to verify a sequence of operations for which existing VC schemes do not perform well on *all* the operations. We thus suggest a technique to compose several specialized and efficient VC schemes with Parno et al.'s general purpose VC protocol Pinocchio, by integrating the verification of the proofs generated by these specialized schemes as a function that is part of the sequence of operations verified using Pinocchio. The resulting scheme keeps Pinocchio's property while being more efficient for the prover. Our scheme relies on the underlying cryptographic assumptions of the composed protocols for correctness and soundness.

Keywords: verifiable computation, proof composition, neural networks

1 Introduction

1.1 Motivation

Neural networks (NN) are machine learning techniques that achieve state of the art performance in various classification tasks such as handwritten digit recognition, object or face recognition. Despite their excellent accuracy performances, these algorithms are computationally expensive and need to be trained with large amounts of data to perform well during the *classification phase*, where all the algorithm parameters are set. Therefore, cloud providers such as Amazon or Microsoft start offering Machine Learning as a Service (MLaaS) to perform complex classification tasks. However if a user does not trust the cloud provider, neither shall he trust the result of the computation returned by the server. Verifiable computing gives an answer to this problem and aims at providing computational integrity without any assumptions on hardware nor on failures that could happen. Existing verifiable computing (VC) systems can theoretically prove and verify all **NP** computations [14]. In practice nevertheless, trade-offs have to be taken between expressiveness and functionality [34].

Due to the sequential nature of NNs, a simple solution would be to compute proofs for each part of the NN sequence. However, this solution would degrade the verifier performance, increase the communication and force the prover to send all the intermediate results. As a consequence, the prover would reveal the parameters of his NN, which are sensitive data.

The goal of this paper is the following: we consider a Neural Network performing classification, which means that it has already been trained and that its parameters are set and we want to define a VC system able to efficiently verify the NN, even if it has a large number of parameters. Following Valiant [32], we propose to compose proofs to address the verifiability of NNs. We reach generality and efficiency by a careful choice of the VC systems to compose. We develop our proposition in the case study of NN but we stress that our methods are general and can fit other use cases.

1.2 Problem Statement

Majority of applications involve several sequences of function evaluations combined through control structures. Assuring the verifiability of these applications has to face the challenge that the functions evaluated as part of these applications may feature computational characteristics that are too variant to be efficiently addressed by a unique VC scheme. For instance, in the case of an application that involves a combination of computationally intensive linear operations with simple non-linear ones, none of the existing VC techniques would be suitable since there is no single VC approach that can efficiently handle both. This question is perfectly illustrated by the sample scenario described in the previous section, namely dealing with the verifiability of Neural Network Algorithms, which can be viewed as a repeated sequence of a matrix product and a non-linear activation function. For instance, a two layer neural network, denoted by g, on an input x can be written as:

$$g(x) = W_2 \cdot f(W_1 \cdot x) \qquad (1)$$

Here W_1 and W_2 are matrices and f is a non-linear function like the frequently chosen Rectified Linear Unit (ReLU) function $x \mapsto max(0, x)$. For efficiency, the inputs are often batched and the linear operations involved in the Neural Network are matrix products instead of products between a vector and a matrix. The batched version of (1) therefore is:

$$g(X) = W_2 \cdot f(W_1 \cdot X) \qquad (2)$$

where X is a batch of inputs to be classified.

In an attempt to assure the verifiability of this neural network, two alternative VC schemes seem potentially suited: the CMT protocol [8] based on interactive proofs and Pinocchio [28]. CMT can efficiently deal with the matrix products but when it comes to the non-linear part of the operations, problems arise since, using CMT, each function to be verified has to be represented as a layered arithmetic circuit (i.e. as an acyclic graph of computation over a finite field with an addition or a multiplication at each node, and where the circuit can be decomposed into layers, each gate of one layer being only connected to an adjacent layer). Nevertheless the second component of the neural network algorithm, that is, the ReLU activation function, does not lend itself to a simple representation as a layered circuit. [18] and [8] have proposed solutions to deal with non-layered circuits at the cost of very complex pre-processing resulting in a substantial increase in the prover's work and the overall circuit size. Conversely, Pinocchio eliminates the latter problem by allowing for efficient verification of the non-linear ReLU activation function while suffering from excessive complexity in the generation of proofs for the products of large matrices (benchmarks on matrix multiplication proofs can be found in [34]).

This sample scenario points to the basic limitation of existing VC schemes in efficiently addressing the requirements of common scenarios involving several components with divergent characteristics such as the mix of linear and non-linear operations as part of the same application. The objective of our work therefore is to come up with a new VC scheme that can efficiently handle these divergent characteristics in the sub-components as part of a single VC protocol.

1.3 Idea of the Solution: Embedded Proofs

Our solution is based on a method that enables the composition of a general purpose VC scheme suited to handle sequences of functions with one or several specialized VC schemes that can achieve efficiency in case of a component function with excessive requirements like very large linear operations. Without loss of generality, we apply the method to a pair of VC schemes, assuming that one is a general purpose VC scheme, called `GVC`, like Pinocchio [28], which can efficiently assure the verifiability of an application consisting of a sequence of functions, whereas the other VC scheme assures the verifiability of a single function in a very efficient way, like, for instance, a VC scheme that can handle large matrix products efficiently. We call this scheme `EVC`. The main idea underlying the VC composition method is that the verifiability of the complex operation (for which the `GVC` is not efficient) is *outsourced* to the `EVC` whereas the remaining non-complex functions are all handled by the `GVC`. In order to get the verifiability of the entire application by the `GVC`, instead of including the complex operation as part of the sequence of functions handled by the `GVC`, this operation is separately handled by the `EVC` that generates a standalone verifiability proof for that operation and the verification of that proof is viewed as an additional function embedded in the sequence of functions handled by the `GVC`. Even though the verifiability of the complex operation by the `GVC` is not feasible due to its complexity, the verifiability of the proof on this operation is feasible by the basic principle of VC, that is, because the proof is much less complex than the operation itself.

We illustrate the VC composition method using as running example the Neural Network defined with formula (2) in Section 1.2. Here, the application consists of the sequential execution of three functions f, g and h (see Figure 1),

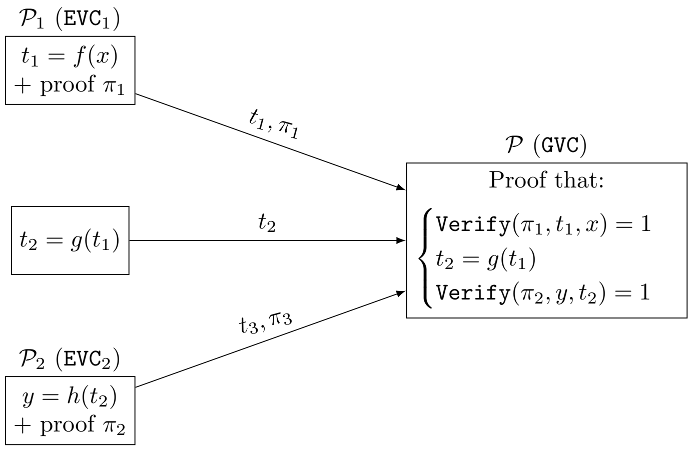

$\mathcal{P}_1$ ($\texttt{EVC}_1$)
$t_1 = f(x)$ + proof π_1
t_1, π_1
$\mathcal{P}$ ($\texttt{GVC}$)
Proof that:
$\texttt{Verify}(\pi_1, t_1, x) = 1$
$t_2 = g(t_1)$
$\texttt{Verify}(\pi_2, y, t_2) = 1$
$t_2 = g(t_1)$
t_2
t_3, π_3
$\mathcal{P}_2$ ($\texttt{EVC}_2$)
$y = h(t_2)$ + proof π_2

Fig. 1. High level view of the embedded proofs

where f and h are not suitable to be efficiently proved correct by GVC while g is. Note that we consider that g cannot be proved correct by any EVC systems or at least not as efficiently as with the GVC system. The computation to verify is therefore $y = h(g(f(x)))$. In our example, the functions f, g and h are $f : X \mapsto W_1 \cdot X$, $h : X \mapsto W_2 \cdot X$ and $g : X \mapsto \max(0, X)$, where X, W_1 and W_2 are matrices and g applies the max function element-wise to the input matrix X.

In order to cope with the increased complexity of f and h, we have recourse to $\texttt{EVC}_1$ and $\texttt{EVC}_2$ that are specialized schemes yielding efficient proofs with such functions. $\pi_{\texttt{EVC}_1}$ denotes the proof generated by $\texttt{EVC}_1$ on f, $\pi_{\texttt{EVC}_2}$ denotes the proof generated by $\texttt{EVC}_2$ on h and $\mathit{\Pi}_{\texttt{GVC}}$ denotes the proof generated by GVC. For the sequential execution of functions f, g and h, the final proof then is:

$$\mathit{\Pi}_{\texttt{GVC}}\Big(\big(\texttt{Verif}_{\texttt{EVC}_1}(\pi_{\texttt{EVC}_1}, x, t_1) \overset{?}{=} 1\big) \wedge \big(g(t_1) \overset{?}{=} t_2\big) \wedge \big(\texttt{Verif}_{\texttt{EVC}_2}(\pi_{\texttt{EVC}_2}, t_2, y) \overset{?}{=} 1\big)\Big) \tag{3}$$

where the inputs of the GVC system are respectively the computation of g and the verification algorithms of $\texttt{EVC}_1$ and $\texttt{EVC}_2$ systems, which output 1 if the proof is accepted and 0 otherwise.

The method can easily be extended to applications involving more than three functions and various specialized VC techniques like EVC that would be selected based on their suitability to the requirements of special functions provided that:

1. The verification algorithm for each EVC proof is compatible with the GVC scheme.
2. The verification algorithm for each EVC proof should have much lower complexity than the outsourced computations (by the basic VC advantage).
3. The EVC schemes should not be VC's with a designated verifier but instead publicly verifiable ones [13]. Indeed, since the prover of the whole computation is the verifier of the EVC, no secret value should be shared between the

prover of the EVC and the prover of the GVC. Otherwise, a malicious prover can easily forge a proof for EVC and break the security of the scheme.

In the sequel of this paper we present a concrete instance of our VC composition method using Pinocchio [28] as the GVC and an efficient interactive proof protocol, namely the Sum-Check protocol [24] as the EVC. We develop further this instance with a Neural Network verification example. We assume that the matrix involved in the products do not have the same sizes so there will be two instances of the Sum-Check protocol. Indeed, the Pinocchio system requires that the verification algorithms are expressed as arithmetic circuits in order to generate evaluation and verification keys for the system. As the parameters of the verification algorithms are different, the two Sum-Check verification protocols are distinct as arithmetic circuits.

1.4 Related Works

Verifying computation made by an untrusted party has been studied for a long time. Theoretical solutions have been proposed: interactive proofs [19], probabilistically checkable proofs (PCP) [2], computationally sound proofs [25] or arguments [22]. Despite solving theoretically the problem, the implementation of these protocols would have been impossible or dramatically inefficient, either in time or space. A decade ago, Goldwasser et al. [18] proposed an efficient interactive proof that, after some optimizations, led to a practical implementation by Cormode et al. [8], later refined by Thaler [31]. Quite at the same moment, Ishai et al. [21] proposed a particular PCP theorem that also led to an implementation by Setty et al. [30]. The next steps toward practical implementations of verifiable computation were taken by Gennaro et al. with the definition of Quadratic Arithmetic Programs [14], an efficient way to encode circuit satisfiability. Building on QAPs, Parno et al. [28] proposed Pinocchio, achieving generality (the protocol can handle any type of computation), succinctness of the proof and possibility for the prover to supply private inputs. The latter property enables to produce proofs that are zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs). Most of the subsequent works also built on QAPs either optimizing the cryptographic protocol embedding the QAPs [10, 4] or improving the expressiveness of the systems [7, 5, 33]. Leveraging the zk-SNARKs property of the proof, several applications of verifiable computation (VC) systems have been proposed: a new cryptocurrency [3], an image authentication protocol [27] or a proposal to improve efficiency, privacy, and integrity of the X.509 public key infrastructure [11].

In SafetyNets [15], Ghodsi et al. build an interactive proof protocol to verify the execution of a deep neural network on an untrusted cloud. This approach, albeit efficient has several disadvantages over ours. The first is that expressivity of the interactive proof protocol used in SafetyNets prevents from using state of the art activation functions such as ReLU. Following CryptoNets [16], Ghodsi et al. replace ReLU functions by a quadratic activation function, namely $x \mapsto x^2$, which squares the input values element-wise. The fact that square activation functions have unbounded derivative causes instability on the neural network training, as mentioned in [16]. A second disadvantage is the impossibility for the prover to hide some of his inputs, i.e. to prove a non-deterministic computation. As a consequence, the verifier and the prover of SafetyNets have to share the model of the neural network, namely the values of the matrices (e.g. W_1 and W_2 in formula (1)). This situation is quite unusual in machine learning: since the training of neural networks is expensive and requires a large amount of data, powerful hardware and technical skills to obtain a classifier with good accuracy, it is unlikely that cloud providers share their models with users. In contrast, with our proposition the prover could keep the model private and nonetheless be able to produce a proof of correct execution.

Proof composition has been proposed by Valiant [32] and refined in the case of SNARKs by Bitansky et al. [6]. The implementation of SNARKs recursive composition has later been proposed by Ben-Sasson et al. in [5]. The high level idea of the latter proof system is to prove or verify the satisfiability of an arithmetic circuit that checks the validity of the previous proofs. Thus, the verifier should be implemented as an arithmetic circuit and used as a sub-circuit of the next prover. However, SNARKs verifiers perform the verification checks using an elliptic curve pairing and it is mathematically impossible that the base field has the same size as the elliptic curve group order. Ben-Sasson et al. therefore propose a cycle of elliptic curves i.e. at least two elliptic curves where the verifier's finite field of the first curve is the base field of the second curve, enabling proof composition. Although proofs can theoretically be composed as many times as desired, this method has severe overhead. Our method has a more limited spectrum than Ben-Sasson et al.'s but our resulting system is still general purpose and enjoys the property of the GVC system, such as succinctness or efficiency for the prover. Furthermore, our proposal improves the prover time, replacing some part of a computation by sub-circuit verifying the sub-computation that can then be executed outside the prover.

1.5 Paper organization

The rest of the paper is organized as follows: we first introduce the building blocks required to instantiate our method in Section 2, then we state the details of a VC composition based on our method and using the selected `GVC` and `EVC` schemes in Section 3, evaluate the efficiency of our scheme in Section 4, its security in Section 5 and conclude in Section 6.

2 Building blocks

2.1 GVC: Verifiable Computation based on QAPs

Quadratic Arithmetic Programs. In [14], Gennaro et al. defined Quadratic Arithmetic Programs (QAP) as an efficient object for circuit satisfiability. The computation to verify has first to be represented as an arithmetic circuit, from which a QAP is computed. Using the representation based on QAPs, the correctness of the computation can be tested by a divisibility check between polynomials. A cryptographic protocol enables to check the divisibility in only one point of the polynomial and to prevent a cheating prover to build a proof of a false statement that will be accepted.

Definition 1 (from [28]). *A QAP $\mathcal{Q}$ over field $\mathbb{F}$ contains three sets of $m+1$ polynomials $\mathcal{V} = \{(v_k(x))\}$, $\mathcal{W} = \{(w_k(x))\}$, $\mathcal{Y} = \{(y_k(x))\}$ for $k \in \{0, \dots, m\}$ and a target polynomial $t(x)$. Let F be a function that takes as input n elements of $\mathbb{F}$ and outputs n' elements and let us define N as the sum of n and n'. A N-tuple $(c_1, \dots, c_N) \in \mathbb{F}^N$ is a valid assignment for function F if and only if there exists coefficients $(c_{N+1}, \dots, c_m)$ such that $t(x)$ divides $p(x)$, as follows:*

$$p(x) = \left(v_0(x) + \sum_{k=1}^{m} c_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} c_k \cdot w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} c_k \cdot y_k(x)\right).$$

A QAP $\mathcal{Q}$ that satisfies this definition computes F. It has size m and its degree is the degree of $t(x)$.

In the above definition, $t(x) = \prod_{g \in G}(x - r_g)$, where G is the set of multiplicative gates of the arithmetic circuit and each r_g is an arbitrary value labeling a multiplicative gate of the circuit. The polynomials in $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{Y}$ encode the left inputs, the right inputs and the outputs for each gate respectively. By definition, if the polynomial $p(x)$ vanishes at a value r_g, $p(r_g)$ expresses the relation between the inputs and outputs of the corresponding multiplicative gate g. An example of a QAP construction from an arithmetic circuit is given in [28]. It is important to note that the size of the QAP is the number of multiplicative gates in the arithmetic circuit to verify, which also is the metric used to evaluate the efficiency of the VC protocol.

VC protocol. Once a QAP has been built from an arithmetic circuit, a cryptographic protocol embeds it in an elliptic curve. The verification phase makes use of a pairing to perform the divisibility check and to ensure that the inputs of the computation were correctly incorporated and that the QAP has been computed with the same coefficients c_k for the $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{Y}$ polynomials during p's computation. This results in a publicly verifiable computation scheme, as defined below.

Definition 2. *Let F be a function, expressed as an arithmetic circuit over a finite field $\mathbb{F}$ and λ be a security parameter.*

- $(EK_F, VK_F) \leftarrow \texttt{KeyGen}(1^\lambda, F)$*: the randomized* `KeyGen` *algorithm takes as input a security parameter and an arithmetic circuit and produces two public keys, an evaluation key EK_F and a verification key VK_F.*
- $(y, \pi) \leftarrow \texttt{Prove}(EK_F, x)$*: the deterministic* `Prove` *algorithm, takes as inputs x and the evaluation key EK_F and computes $y = F(x)$ and a proof π that y has been correctly computed.*

– $\{0,1\} \leftarrow \texttt{Verify}(VK_F, x, y, \pi)$*: the deterministic* `Verify` *algorithm takes the input/output* (x, y) *of the computation* F*, the proof* π *and the verification key* VK_F *and outputs* 1 *if* $y = F(x)$ *and* 0 *otherwise.*

Security. The desired security properties for a publicly verifiable VC scheme have been defined in [14]:

– *Correctness*: for any function F and any input x:

$$Pr\left[\begin{array}{r}(EK_F, VK_F) \leftarrow \texttt{KeyGen}(F, 1^\lambda);\\ (y,\pi) \leftarrow \texttt{Compute}(EK_F, x);\\ \texttt{Verify}(VK_F, x, y, \pi) = 1\end{array}\right] = 1$$

– *Soundness*: for any function F and any probabilistic polynomial-time adversary $\mathcal{A}$,

$$Pr\left[\begin{array}{l}(u, y, \pi) \leftarrow \mathcal{A}(EK_F, VK_F) :\\ F(u) \neq y \text{ and } \texttt{Verify}(VK_F, u, y, \pi) = 1\end{array}\right] \leqslant \texttt{negl}(\lambda)$$

– *Efficiency*: the cost of `Verify` is lower than the cost of F.

Parno et al. [28] prove that their VC protocol is correct and sound if some knowledge of exponent assumptions [20] hold.

Costs. In QAP-based protocols, the proof consists of few elliptic curve elements (e.g. 8 group elements in Pinocchio [28]) and has constant size no matter the computation to be verified: the verification is fast. Regarding the set-up phase, the `KeyGen` algorithm outputs evaluation and verification keys that depend on the function F, but not on its inputs. The resulting model is often called verifiable computation with pre-processing. The setup phase has to take place only once, the keys are reusable for later inputs and the cost of the pre-processing is amortized over all further computations. The bottleneck is due to the computations performed by the prover: for an arithmetic circuit of N multiplication gates, the prover has to compute $O(N)$ cryptographic operations and $O(N \log^2 N)$ non-cryptographic operations.

Zero-knowledge. QAPs also achieve the zero-knowledge property with little overhead: the prover can supply private inputs in the computation and randomize the proof by adding multiples of the target polynomial $t(x)$ to hide his inputs. The proof obtained using the protocol by Parno et al. thus is called a zero-knowledge Succinct Non-Interactive Argument (zk-SNARK). zk-SNARKs have found many applications [11, 3, 33, 27] due to their (relative) efficient production and their non-interactivity. Moreover, in the zk-SNARKs setting, the efficiency requirement is no longer meaningful since the computation could not have been performed by the verifier. Indeed, some of the inputs are supplied by the prover and remain private, making the computation impossible to perform by the sole verifier.

2.2 EVC: Sum-Check Protocol

The Sum-Check protocol [24] enables to prove the correct evaluation of a multivariate polynomial P defined over a finite field. Suppose that P is a polynomial with n variables, defined over $\mathbb{F}^n$ and it has to be evaluated on $\{0,1\}^n$. Using the Sum-Check protocol, a prover $\mathcal{P}$ can convince a verifier $\mathcal{V}$ that he knows the value:

$$H = \sum_{t_1 \in \{0,1\}} \sum_{t_2 \in \{0,1\}} \dots \sum_{t_n \in \{0,1\}} P(t_1, \dots, t_n) \tag{4}$$

A direct computation performed by the verifier requires at least 2^n evaluations while the Sum-Check protocol only requires $O(n)$ evaluations for the verifier.

The protocol is a public coin interactive proof with n rounds of interaction during which the prover computes n univariate polynomials $P_i, i = 1 \dots, n$ derived from P. During each round, $\mathcal{V}$ generates a random field value that $\mathcal{P}$ has to integrate in the computation of the next P_i. The consistency of $\mathcal{P}$'s answer is then checked with this value. $\mathcal{V}$ is supposed to be able to compute P in one point, namely in the point $(r_1, r_2, \dots, r_n)$, corresponding to the random challenges r_i he has sent to $\mathcal{P}$ during the rounds of the protocol.
The n rounds of the protocol are summarized below:

- $\mathcal{P}$ computes the univariate polynomial P_1 claimed to be computed as follow:

$$P_1(x) = \sum_{t_2 \in \{0,1\}, \dots, t_n \in \{0,1\}} g(x, t_2, \dots, t_n) \tag{5}$$

- $\mathcal{P}$ sends H and P_1 to $\mathcal{V}$.
- $\mathcal{V}$ checks if $\mathcal{P}$'s claim on P_1 holds using: $H = P_1(0) + P_1(1)$. If it does not, $\mathcal{V}$ rejects and the protocol stops. Otherwise, $\mathcal{V}$ chooses uniformly at random $r_1 \in \mathbb{F}$ and sends it to $\mathcal{P}$.
- $\mathcal{P}$ sends $\mathcal{V}$ a univariate polynomial P_2 and claims that:

$$P_2(x) = \sum_{t_3 \in \{0,1\}, \dots, t_n \in \{0,1\}} P(r_1, x, t_3, \dots, t_n) \tag{6}$$

- $\mathcal{V}$ checks if $\mathcal{P}$'s claim on P_2 holds using: $P_1(r1) = P_2(0) + P_2(1)$. If the claim does not hold, $\mathcal{V}$ rejects the response ad the protocol stops. Otherwise, $\mathcal{V}$ picks another value $r_2 \in \mathbb{F}$ uniformly at random and sends it to $\mathcal{P}$
  The protocol goes on, until the n-th round, where:
- $\mathcal{P}$ sends the univariate polynomial P_n and claims that:

$$P_n(x) = P(r_1, r_2, r_3, \dots, r_{n-1}, x) \tag{7}$$

- $\mathcal{V}$ picks a random value $r_n \in \mathbb{F}$ and checks that $P_n(r_n) = P(r_1, r_2, \dots, r_n)$. If the equality holds, then $\mathcal{V}$ is convinced that $H = P_1(0) + P_1(1)$ and that H has been evaluated as in (4).

The Sum-Check protocol has the following properties:

1. The protocol is *correct*, i.e. if $\mathcal{P}$'s claim about H is true,then $\mathcal{V}$ accepts with probability 1
2. The protocol is *sound*, i.e. if the claim on H is false, the probability that $\mathcal{P}$ can make $\mathcal{V}$ accept H is bounded by $nd/|\mathbb{F}|$, where n is the number of variables and d the degree of the polynomial P.

Note that the soundness is here information theoretic: no assumption is made on the prover power. To be able to implement the Sum-Check protocol verification algorithm into an arithmetic circuit we need a non-interactive version of the protocol. Indeed, Pinocchio requires the complete specification of each computation as input to its QAP generation process (see Section 2.1). Due to the interactive nature of the Sum-Check protocol, the proof cannot be generated before the actual execution of the protocol. We therefore use the Fiat-Shamir transformation [12] to obtain a non-interactive version of the Sum-Check protocol that can be used as an input to Pinocchio. In the Fiat-Shamir transformation, the prover replaces the uniformly random challenges sent by the verifier by challenges he computes applying a public hash function to the transcript of the protocol so far. The prover then sends the whole protocol transcript, which can be verified recomputing the challenges with the same hash function. This method has been proved secure in the random oracle model [29].

2.3 Multilinear extensions

Multilinear extensions allow to apply the Sum-Check protocol to polynomials defined over some finite set included in the finite field where all the operations of the protocol are performed. Thaler [31] showed how multilinear extensions and the Sum-Check protocol can be combined to give a time-optimal proof for matrix multiplication.
Let $\mathbb{F}$ be a finite field. For any d-variate polynomial P, we denote by $\deg_i(P)$ the degree of P in variable i. A n-variate polynomial P is *multilinear* if: $\deg_i(P) \leqslant 1, \forall i \in \{1, \ldots, d\}$.

Definition 3. *Let $f : \{0,1\}^d \to \mathbb{F}$ be any function mapping the d-dimensional boolean hypercube to $\mathbb{F}$. A d-variate polynomial g over $\mathbb{F}$ is said to be an* extension *of f if: $g(x) = f(x) \forall x \in \{0,1\}^d$.*

A multilinear extension (MLE) of a function f is a polynomial that is an extension of f and has degree at most 1 in each variable.

Theorem 1. *Any function $f : \{0,1\}^d \to \{0,1\}$ has a unique multilinear extension over $\mathbb{F}$.*

We will hereafter denote by $\tilde{f}$ the MLE of f. Using Lagrange interpolation, an explicit expression of MLE can be obtained:

Lemma 1. *Let $f : \{0,1\}^d \to \{0,1\}$. Then $\tilde{f} : \mathbb{F}^d \to \{0,1\}$ has the following expression:*

$$\tilde{f}(x_1,\dots,x_d) = \sum_{w\in\{0,1\}^d} f(w)\chi_w(x_1,\dots,x_d) \qquad (8)$$

where χ_w is defined for all $w = (w_1,\dots,w_d)$ as :

$$\chi_w(x_1,\dots,x_d) = \prod_{i=1}^{d}(x_i w_i + (1-x_i)(1-w_i)) \qquad (9)$$

2.4 Ajtai Hash Function

As mentioned in Section 1.2, our goal is to compute a proof of an expensive sub-computation with the Sum-Check protocol and to verify that proof using the Pinocchio protocol. The non-interactive nature of Pinocchio prevents from proving the sub-computation with an interactive protocol. As explained in Section 2.2, we turn the Sum-Check protocol into a non-interactive argument using the Fiat-Shamir transform [12]. This transformation need a hash function to simulate the challenges that would have been provided by the verifier. The choice of the hash function to compute challenges in the Fiat-Shamir transformation here is crucial because we want to verify the proof transcript inside the `GVC` system, which will be instantiated with the Pinocchio protocol. This means that the computations of the hash function have to be verified by the `GVC` system and that the verification should not be more complex than the execution of the original algorithm inside the `GVC` system. For instance the costs using a standard hash function such as SHA256 [26] would be too high: [3] reports about 27,000 multiplicative gates to implement the compression function of SHA256. Instead, we choose a hash function which is better suited for arithmetic circuits, namely the Ajtai hash function [1] that is based on the subset sum problem as defined below:

Definition 4. *Let m, n be positive integers and q a prime number. For a randomly picked matrix $A \in \mathbb{Z}_q^{n\times m}$, the Ajtai hash $H_{n,m,q} : \{0,1\}^m \to \mathbb{Z}_q^n$ is defined as:*

$$\forall x \in \{0,1\}^m, \quad H_{n,m,q} = A \times x \mod q \qquad (10)$$

As proved by Goldreich et al. [17], the collision resistance of the hash function relies on the hardness of the Short Integer Solution (SIS) problem. Ajtai hash functions have first been used in verifiable computation by Braun et al. in [7]. Ben-Sasson et al. [5] then noticed that the translation in arithmetic circuit is better if the parameters are chosen to fit with the underlying field of the computations. A concrete hardness evaluation is studied by Kosba et al. in [23]. Choosing $\mathbb{F}_p$, with $p \approx 2^{254}$ to be the field where the computations of the arithmetic circuit take place leads to the following parameters for approximately 100 bit of security:

$$n = 3, m = 1524, q = p \approx 2^{254}.$$

Few gates are needed to implement an arithmetic circuit for this hash function: to hash m bits, $n \times m$ multiplicative gates are needed. With the parameters selected in [23], this means that 4572 gates are needed to hash 1524 bits.

3 Verifiable Neural Networks via Embedded Proofs

3.1 High level description of our protocol

The generation of a proof for $y = h(g(f(x)))$ involves several steps as depicted in Figure 2. Since the prover has to execute three different proving algorithms, the prover is viewed as the union of three sub-provers, the first and the second sub-provers $\mathcal{P}_1$ and $\mathcal{P}_2$ being in charge of the efficient VC algorithm (EVC) and the third sub-prover $\mathcal{P}_3$ using the general verifiable computation (GVC) algorithm. In the **Setup** phase, the verifier and the prover agree on an arithmetic circuit which describes the computation of the function g along with the verification algorithms of the proof that f and h were correctly computed. The pre-processing phase of the GVC system takes the resulting circuit and outputs the corresponding evaluation and verification keys. In the **query** phase, the verifier sends the prover an input x for the computation along with a random value that will be an input for the efficient sub-provers $\mathcal{P}_1$ and $\mathcal{P}_2$. In the **proving** phase, $\mathcal{P}_1$ first computes $t_1 = f(x)$ and produces a proof $\pi_{\mathtt{EVC}_1}$ of the correctness of the computation, using the efficient proving algorithm. The prover $\mathcal{P}$ then computes the value $t_2 = g(t_1)$ and sub-prover $\mathcal{P}_2$ computes $y = h(t_2)$. Sub-prover $\mathcal{P}_3$ then verifies the proofs $\pi_{\mathtt{EVC}_1}$ and $\pi_{\mathtt{EVC}_2}$: if the verification fails it aborts. Otherwise, $\mathcal{P}_3$ provides the inputs/outputs of the previous computations to the GVC system and, using the evaluation key computed in the setup phase, builds a proof $\pi_{\mathtt{GVC}}$ that:

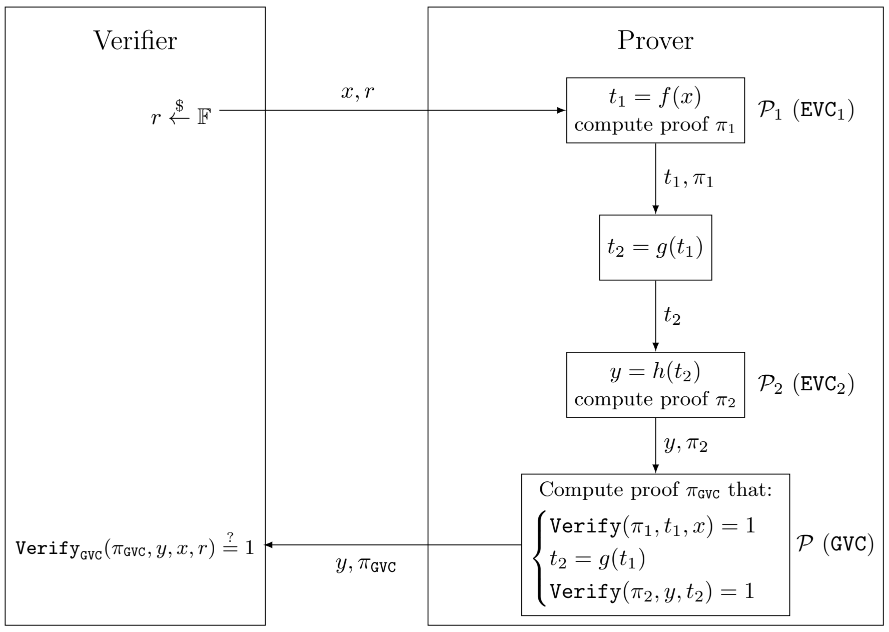
1. the proof $\pi_{\mathtt{EVC}_1}$ has been correctly verified,
2. the computation $t_2 = g(t_1)$ is correct,
3. the proof $\pi_{\mathtt{EVC}_2}$ has been correctly verified

In the **verification** phase, the verifier checks that y was correctly computed using the GVC's verification algorithm, the couple $(y, \pi_{\mathtt{GVC}})$ received from the prover, and (x, r).

Recall that our goal is to gain efficiency compared with the proof generation of the whole computation inside the GVC system. Therefore, we need proof algorithms with a verification algorithm that can be implemented efficiently as an arithmetic circuit and for which the running time of the verification algorithm is more efficient than running the computation. Since the Sum-Check protocol involves algebraic computations over a finite field, it can easily be implemented as an arithmetic circuit and fits well to our model.

3.2 Protocol description

In this section, we give a description of embedded proofs in the case where the function f takes as input a matrix X and returns the product $W_1 \times X$, the function h takes as input a matrix X and returns the product $W_2 \times X$ and the function g is the ReLU function $x \mapsto \max(0, x)$. We use the Sum-Check protocol to prove correctness of the matrix multiplication, as in [31] and Pinocchio as the global proof mechanism.



Verifier
Prover
$r \xleftarrow{\$} \mathbb{F}$
x, r
$t_1 = f(x)$
compute proof π_1
$\mathcal{P}_1$ (EVC$_1$)
t_1, π_1
$t_2 = g(t_1)$
t_2
$y = h(t_2)$
compute proof π_2
$\mathcal{P}_2$ (EVC$_2$)
y, π_2
Compute proof $\pi_{\texttt{GVC}}$ that:
$\texttt{Verify}(\pi_1, t_1, x) = 1$
$t_2 = g(t_1)$
$\texttt{Verify}(\pi_2, y, t_2) = 1$
$\mathcal{P}$ (GVC)
$\texttt{Verify}_{\texttt{GVC}}(\pi_{\texttt{GVC}}, y, x, r) \overset{?}{=} 1$
$y, \pi_{\texttt{GVC}}$

Fig. 2. Embedded proof protocol

For the sake of simplicity, W_1 and W_2 are assumed to be square matrices: $W_1 \in \mathcal{M}_{n,n}(\mathbb{F})$, $W_2 \in \mathcal{M}_{m,m}(\mathbb{F})$. We denote $d_1 = \log n$, $d_2 = \log m$ and we denote by H the Ajtai hash function (see Section 2 for details). The protocol is the following:

Setup:
- Verifier and Prover agree on an arithmetic circuit $\mathcal{C}$ description for the computation. $\mathcal{C}$ implements both the evaluation of the function g and the verification algorithms of the Sum-Check protocols for the two matrix multiplications.
- $(EK_{\mathcal{C}}, VK_{\mathcal{C}}) \leftarrow \texttt{KeyGen}(1^\lambda, \mathcal{C})$

Query Verifier:
- generates a random challenge $(r_L, r_R) \in \mathbb{F}^{\log n} \times \mathbb{F}^{\log n}$
- sends the prover a matrix input X and the challenge (r_L, r_R)

Proof Sub-prover $\mathcal{P}_1$, on input X:
- computes the product $T_1 = W_1 \times X$
- computes the multilinear extension $\widetilde{T_1}(r_L, r_R)$
- computes, using serialized Sum-Check protocol, the proof $\pi_{\texttt{EVC}_1}$ of the evaluation of the polynomial:

$$P(x) = \widetilde{W_1}(r_L, x) \cdot \widetilde{X}(x, r_R) \tag{11}$$

  where $x = (x_1, \ldots, x_d) \in \mathbb{F}^d$.
- Sends prover $\mathcal{P}$ the value $(X, W_1, T_1, \pi_{\texttt{EVC}_1}, r_L, r_R)$.

– Prover $\mathcal{P}$ computes the value $T_2 = g(T_1)$ and sends it to sub-prover $\mathcal{P}_2$

Sub-prover $\mathcal{P}_2$, on input T_2:

– computes the product $Y = W_2 \times T_2$

– computes the multilinear extension $\widetilde{Y}(r_L, r_R)$

– computes, using serialized Sum-Check protocol, the proof $\pi_{\texttt{EVC}_2}$ of the evaluation of the polynomial:

$$P(x) = \widetilde{W}_2(r_L, x) \cdot \widetilde{T}_2(x, r_R) \qquad (12)$$

where $x = (x_1, \ldots, x_d) \in \mathbb{F}^d$.

– Sends prover $\mathcal{P}$ the tuple $(T_2, W_2, Y, \pi_{\texttt{EVC}_2}, r_L, r_R)$.

Prover $\mathcal{P}$: on input $(X, W_1, T_1, \pi_{\texttt{EVC}_1}, r_L, r_R)$

– Computes $\widetilde{T}_1(r_L, r_R)$.

– Parses $\pi_{\texttt{EVC}_1}$ as: $(P_1, r_1, P_2, r_2, \ldots, P_d, r_d)$

– Verifies π:
  - Checks that: $P_1(0) + P_1(1) = \widetilde{T}_1(r_L, r_R)$
  - Computes: $r_{i+1} = H(a_i, b_i, c_i, r_i)$, for $i = 1, \ldots, d$, with $r_1 = r_L \parallel r_R$
  - Checks that: $P_i(0) + P_i(1) = P_{i-1}(r_{i-1})$ for $i = 2, \ldots, d$.
  - From X and W_1, computes the multilinear extensions: $\widetilde{W}_1(r_L, r_1, \ldots, r_d)$ and $\widetilde{X}(r_1, \ldots, r_d, r_R)$
  - Checks that:

$$P_d(r_d) = \widetilde{W}_1(r_L, r_1, \ldots, r_d) \cdot \widetilde{X}(r_1, \ldots, r_d, r_R) \qquad (13)$$

– Aborts if one of the previous checks fails. Otherwise, accepts T_1 as the product of W_1 and X.

– On input $(T_2, W_2, Y, \pi_{\texttt{EVC}_2}, r_L, r_R)$, apply the verification algorithm as above. If all the checks pass, accept Y as the product of W_2 and T_2. Otherwise, abort.

– Computes $T_2 = ReLU(T_1)$.

– Using Pinocchio, computes the final proof $\pi_{\texttt{GVC}}$ that $\pi_{\texttt{EVC}_1}$ and $\pi_{\texttt{EVC}_2}$ have been verified and T_2 has been correctly computed from T_1.

– Sends $(Y, \pi_{\texttt{GVC}})$ to the Verifier.

Verification Verifier:

– computes `Verify`$(X, r_R, r_L, Y, \pi_{\texttt{GVC}})$

– If `Verify` fails, rejects the value Y. Otherwise accepts the value Y as the result of $Y = W_2 \cdot ReLU(W_1 \cdot X)$

4 Efficiency Evaluation

Regarding the embedded proofs protocol, we seek efficiency gains over the execution of the complete function evaluation inside the GVC system. Since the complexity of the Pinocchio scheme is related to the number of multiplicative gates of the arithmetic circuit to verify, it is natural to study the number of multiplications performed by the `GVC` prover $\mathcal{P}$. More precisely, we need to compare the cost of implementing the verification algorithm of the `EVC` inside the `GVC` system with the cost of executing the function directly in the `GVC` system. For the example we developed in Section 3.2, we need to compare the cost of executing a matrix multiplication in the Pinocchio system with the cost of implementing the verification of the Sum-Check protocol inside Pinocchio.

Matrix multiplication cost. QAP encodes the constraints of all multiplication gates of the circuit to verify, hence the last operation performed in the circuit cannot be an addition. In a matrix multiplication $C = A \cdot B$, each component $c_{i,j}$ of the product is the result of an operation $\sum_k a_{i,k} \cdot b_{k,j}$. Multiplications between components are first performed and the product component $c_{i,j}$ is the addition of the latter multiplications. The representation as a circuit has thus to add an extra multiplication gate, which is a multiplication by 1, to enable the verification of the last addition in the circuit constraints. Consequently, in the multiplication between $n \times n$ matrices, the computation of each of the n^2 component requires $n + 1$ multiplicative gates in the corresponding arithmetic circuit.

To sum up, a circuit implementing matrix multiplication between two $n \times n$ matrices requires $(n + 1) \times n^2 = n^3 + n^2$ multiplication gates.

Sum-check verification cost. The verification algorithm of the Sum-Check protocol has three parts. The first one is the consistency checks for the received univariate polynomials, the second one is the computation of the multilinear extension to perform the last check of the protocol while the last one is the hash computation of the challenge in the serialized proof. In the sequel of the paper, we assume that the sub-prover $\mathcal{P}$ has received a proof $\pi_{\mathtt{EVC}_1} = (P_1, r_1, P_2, r_2, \ldots, P_d, r_d)$ from sub-prover $\mathcal{P}_1$. The same reasoning would apply for $\mathcal{P}_2$.

Consistency checks. Recall that the verifier has to check if:$P_1(0) + P_1(1)$ is equal to the claimed value and if $P_i(0) + P_i(1) = P_{i-1}(r_{i-1})$ for $i = 2, \ldots, d$. If sub-prover $\mathcal{P}_1$ sends the coefficient of the degree 2 polynomials P_i, which we will denote by a_i, b_i, c_i, then the check becomes:

$$\begin{aligned} a_i + b_i + 2c_i &= a_{i-1}r_{i-1}^2 + b_{i-1}r_{i-1} + c_{i-1} \\ &= (a_{i-1}r_{i-1} + b_{i-1})r_{i-1} + c_{i-1}. \end{aligned} \tag{14}$$

The cost of this check is 3 multiplicative gates for the equality testing (see [28]) and, using Horner algorithm, 2 multiplication gates for the $P_{i-1}(r_{i-1})$ evaluation, the computation of the left hand side being free. Adding the first check, which is only an equality check, we obtain a cost of $5 \cdot \log n + 3$ multiplicative gates for (14).

Multilinear extension computation. For the final check, sub-prover $\mathcal{P}$ has to compute the multilinear extension of the input matrices that we will denote by A, B, C for convenience. Suppose that A is a (n, n) matrix and denote $d = \log n$. A can be interpreted as a function $A : \{0, 1\}^{d \times d} \to \mathbb{F}$, associating to each index (i, j) value written in binary form the value $A(i, j)$. In the last step of the Sum-Check protocol, the verifier has to compute $\tilde{A}(r_L, r_1, \ldots, r_d)$, where $r_L = (r_{L_1}, \ldots, r_{L_d}) \in \mathbb{F}^d$ is the randomness sent to the prover. Cormode et al. [9] describe an algorithm to compute this multilinear extension, using $O(n \times d)$ time and $O(d)$ space. For convenience, we rewrite the randomness $(r_1, \ldots, r_{2d})$.

The algorithm is the following:

$$\tilde{A}(r_1,\dots,r_{2d}) \leftarrow 0$$
$$\tilde{A}(r_1,\dots,r_{2d}) \leftarrow \tilde{A}(r_1,\dots,r_{2d}) + A(i_1,\dots,i_{2d}) \cdot \chi_{(i_1,\dots,i_{2d})}(r_1,\dots,r_{2d}) \tag{15}$$

Where $(i_1,\dots,i_{2d})$ spans the hypercube $\{0,1\}^{2d}$ and χ is defined in Section 2.3 in equation (9). On each loop the actualization of $\tilde{A}(r_1,\dots,r_{2d})$ requires $6d$ multiplications. The total number of multiplications is thus $6d2^{2d} = 6n^2 \log n$. Since the verifier has 3 multilinear extensions to compute from matrices of the same size, the total number of multiplications for the matrix multilinear extensions is $18n^2 \log n$.
Challenge computations in the proof serialization. The Sum-Check protocol, as performed for the matrix multiplication (Section 3.2), involves d rounds. The verifier need to compute a hash value to check the challenge of the next round is correct. The total cost is thus $d \cdot \text{cost}(H)$, where $\text{cost}(H)$ is the cost required to compute the hash value to get the challenge. Using Ajtai hash function with the parameters given in Section 2.4, the cost of hashing the 4 field elements to get the new challenge would be: $\text{cost}(H) = 4572$ gates.

Gathering all the sub-costs, the overall cost to verify the Sum-Check protocol for matrix multiplication is $18n^2 \log n + 4577 \log n + 3$.

Table 1 compares the cost required to prove a matrix multiplication using our embedded proof system and computing the whole proof inside the GVC system. The cost is expressed as the number of multiplication gates inside the arithmetic circuit performing the computation.

Table 1. Matrix multiplication cost comparison (in multiplicative gates)

n	64	128	256	512	1024
Whole GVC	266240	2113536	16842752	134479872	1074790400
Embedded proofs	469833	2096426	9473803	42508524	188789453
Gains	−76%	0.8%	44%	68%	82%

5 Security Evaluation

Our embedded proof system has to satisfy the correctness and soundness requirements. Suppose that we have a `GVC` and two `EVC` systems to prove the computation of $y = h(g(f(x)))$. These systems already satisfy the correctness and soundness requirements. Let denote by $\epsilon_{\texttt{GVC}}$, $\epsilon_{\texttt{EVC}_1}$ and $\epsilon_{\texttt{EVC}_2}$ the respective soundness errors of these systems. Note that while the `EVC` systems prove that $t_1 = f(x)$ and $y = h(t_2)$ have been correctly computed, the `GVC` system proves the correctness of three computations, namely that the verification of the `EVC` proofs has passed and that the computation $t_2 = g(t_1)$ is correct. Furthermore, the GVC system proves the correct execution of the function F as defined below:

$$F : (x, t_1, t_2, \pi, g, y) \mapsto \begin{cases} 1 & \text{if } \mathtt{Verify}_{\mathtt{EVC}_1}(x, t_1, \pi_{\mathtt{EVC}_1}) = 1 \text{ and } t_2 = g(t_1) \\ & \text{and } \mathtt{Verify}_{\mathtt{EVC}_2}(t_2, y, \pi_{\mathtt{EVC}_2}) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

5.1 Correctness

Theorem 2. *If the* EVC *and the* GVC *systems are correct then our embedded proof system is correct.*

Proof. Assume that the value $y = h(g(f(x)))$ has been correctly computed. It means that $t_1 = f(x)$ and $t_2 = g(t_1)$ have also been correctly computed. Since the GVC system is correct, it ensures that the function F will pass the GVC verification with probability 1, provided that its result is correct. Now, since the $\mathtt{EVC}_1$ system is correct, the probability that $\mathtt{Verify}_{\mathtt{EVC}_1}(x, t_1, \pi_{\mathtt{EVC}_1}) = 1$ is 1. The same argument applies for the $\mathtt{EVC}_2$ system: since it is correct, the probability that $\mathtt{Verify}_{\mathtt{EVC}_2}(t_2, y, \pi_{\mathtt{EVC}_2}) = 1$ is 1.

Therefore, if $y = h(g(f(x)))$ has been correctly computed, then the function F will also be correctly computed and the verification of the embedded proof system will pass with probability 1.

5.2 Soundness

Theorem 3. *If the* EVC *and the* GVC *systems are sound with soundness error respectively equal to $\epsilon_{\mathtt{EVC}}$ and $\epsilon_{\mathtt{GVC}}$, then our embedded proof system is sound with soundness error at most $\epsilon := \epsilon_{\mathtt{EVC}_1} + \epsilon_{\mathtt{EVC}_2} + \epsilon_{\mathtt{GVC}}$.*

Proof. Assume that a p.p.t. adversary $\mathcal{A}_{emb}$ returns a cheating proof π for a result y' on input x, i.e. $y' \neq h(g(f(x)))$ and π is accepted by the verifier with probability higer than ϵ. We then construct an adversary $\mathcal{B}$ that breaks the soundness property of either the GVC or of one of the EVC systems. We build $\mathcal{B}$ as follow: $\mathcal{A}_{emb}$ interacts with the verifier of the embedded system until a cheating proof is accepted. $\mathcal{A}_{emb}$ then forwards the cheating tuple $(x, t_1, t_2, \pi_{\mathtt{EVC}_1}, \pi_{\mathtt{EVC}_2}, y, \pi)$ for which the proof π has been accepted. $\mathcal{B}$ can then submit a cheating proof to the GVC or the EVC systems, depending on the result of the tests $t_1 \stackrel{?}{=} f(x)$, $t_2 \stackrel{?}{=} g(t_1)$ or $y \stackrel{?}{=} h(t_2)$:

– *$t_1 = f(x)$ is not correct:*
  By definition of the proof π, this means that the $\mathtt{EVC}_1$ proof has been accepted by the verification algorithm implemented inside the GVC system. $\mathcal{A}_{emb}$ can then forward to the adversary $\mathcal{B}$ the tuple $(x, t_1, \pi_{\mathtt{EVC}_1})$. Now if $\mathcal{B}$ presents the tuple $(x, t, \pi_{\mathtt{EVC}_1})$ to the $\mathtt{EVC}_1$ system, it succeeds with probability 1.

Therefore, the probability that the verifier of the embedded proof system accepts is:

$$\begin{aligned} Pr[\mathcal{V}_{\mathtt{EVC}_1} \text{ accepts } \pi] &= Pr[\mathcal{V}_{\mathtt{EVC}_1} \text{ accepts } /\mathcal{V}_{emb} \text{ accepts } \pi] \times Pr[\mathcal{V}_{emb} \text{ accepts } \pi] \\ &= 1 \times \epsilon \\ &\geqslant \epsilon_{\mathtt{EVC}_1} \end{aligned}$$

Thus $\mathcal{B}$ breaks the soundness property of $\mathtt{EVC}_1$.

– $y = h(t_2)$ *is not correct:*
  Based on the same reasoning, $\mathcal{B}$ will break the soundness property of $\mathtt{EVC}_2$ by forwarding the tuple $(t_2, y, \pi_{\mathtt{EVC}_2})$ to the $\mathtt{EVC}_2$ system.
– $t_2 = g(t_1)$ *is not correct:*
  This means that the proof π computed by the $\mathtt{GVC}$ system is accepted by $\mathcal{V}_{emb}$ even if t_2 has not been correctly computed. If $\mathcal{B}$ forwards π to a GVC system, it breaks its soundness.

6 Conclusion

We design an efficient verifiable computing scheme aiming at proving the correctness of a neural network algorithm. Our scheme builds on the notion of proof composition and leverages an efficient VC scheme, namely the Sum-Check protocol to improve the performance of Pinocchio in proving matrix multiplications. We prove that our scheme is sound and give an efficiency evaluation. As noted in Section 1.3, the composition technique described in the article can be extended to other VC schemes and to an arbitrary number of sequential function evaluations, provided that they respect the requirements defined therein.

References

1. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996. pp. 99–108 (1996)
2. Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of NP. J. ACM 45(1), 70–122 (1998)
3. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014. pp. 459–474 (2014)
4. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for C: verifying program executions succinctly and in zero knowledge. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. pp. 90–108 (2013)
5. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II. pp. 276–294 (2014)

6. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013. pp. 111–120 (2013), `http://doi.acm.org/10.1145/2488608.2488623`
7. Braun, B., Feldman, A.J., Ren, Z., Setty, S.T.V., Blumberg, A.J., Walfish, M.: Verifying computations with state. In: ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP'13, Farmington, PA, USA, November 3-6, 2013. pp. 341–357 (2013)
8. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 90–112 (2012)
9. Cormode, G., Thaler, J., Yi, K.: Verifying computations with streaming interactive proofs. PVLDB 5(1), 25–36 (2011), `http://www.vldb.org/pvldb/vol5/p025_grahamcormode_vldb2012.pdf`
10. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015. pp. 253–270 (2015)
11. Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Parno, B.: Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016. pp. 235–254 (2016)
12. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. pp. 186–194 (1986)
13. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. pp. 465–482 (2010)
14. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. pp. 626–645 (2013)
15. Ghodsi, Z., Gu, T., Garg, S.: Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. CoRR abs/1706.10268 (2017), `http://arxiv.org/abs/1706.10268`
16. 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: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. pp. 201–210 (2016), `http://jmlr.org/proceedings/papers/v48/gilad-bachrach16.html`
17. Goldreich, O., Goldwasser, S., Halevi, S.: Collision-free hashing from lattice problems. Electronic Colloquium on Computational Complexity (ECCC) 3(42) (1996), `http://eccc.hpi-web.de/eccc-reports/1996/TR96-042/index.html`
18. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008. pp. 113–122 (2008)
19. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. 18(1), 186–208 (1989)

20. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings. pp. 321–340 (2010)
21. Ishai, Y., Kushilevitz, E., Ostrovsky, R.: Efficient arguments without short pcps. In: 22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA. pp. 278–291 (2007)
22. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada. pp. 723–732 (1992)
23. Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., abhi shelat, Shi, E.: C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093 (2015), `http://eprint.iacr.org/2015/1093`
24. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. In: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I. pp. 2–10 (1990)
25. Micali, S.: Computationally sound proofs. SIAM J. Comput. 30(4), 1253–1298 (2000)
26. Secure Hash Standard (SHS). Federal Information Processing Standard 180-4 (2015), National Institute of Standards and Technology.
27. Naveh, A., Tromer, E.: Photoproof: Cryptographic image authentication for any set of permissible transformations. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016. pp. 255–271 (2016)
28. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013. pp. 238–252 (2013)
29. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. J. Cryptology 13(3), 361–396 (2000), `https://doi.org/10.1007/s001450010003`
30. Setty, S.T.V., McPherson, R., Blumberg, A.J., Walfish, M.: Making argument systems for outsourced computation practical (sometimes). In: 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012 (2012)
31. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. pp. 71–89 (2013)
32. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. pp. 1–18 (2008), `https://doi.org/10.1007/978-3-540-78524-8_1`
33. Wahby, R.S., Setty, S.T.V., Ren, Z., Blumberg, A.J., Walfish, M.: Efficient RAM and control flow in verifiable outsourced computation. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015 (2015)
34. Walfish, M., Blumberg, A.J.: Verifying computations without reexecuting them. Commun. ACM 58(2), 74–84 (Jan 2015), `http://doi.acm.org/10.1145/2641562`