Efficient Proof Composition for Verifiable Computation

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

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

Abstract. Outsourcing machine learning algorithms helps users to deal with large amounts of data without the need to develop the expertise required by these algorithms. Outsourcing however raises severe security issues due to potentially untrusted service providers. Verifiable computing (VC) tackles some of these issues by assuring computational integrity for an outsourced computation. In this paper, we design a VC protocol tailored to verify a sequence of operations for which no existing VC scheme is suitable to achieve realistic performance objective for the entire sequence. We thus suggest a technique to compose several specialized and efficient VC schemes with a general purpose VC protocol, like Parno et al.'s 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 the general purpose scheme. The resulting scheme achieves the objectives of the general purpose scheme with increased efficiency for the prover. The scheme relies on the underlying cryptographic assumptions of the composed protocols for correctness and soundness.

Keywords: Verifiable computation, Proof composition, Neural networks

1 Introduction

While achieving excellent results in diverse areas, machine learning algorithms require expertise and a large training material to be fine-tuned. Therefore, cloud providers such as Amazon or Microsoft have started offering Machine Learning as a Service (MLaaS) to perform complex machine learning tasks on behalf of users. Despite these advantages, outsourcing raises a new requirement: in the face of potentially malicious service providers the users need additional guarantees to gain confidence in the results of outsourced computations. As an answer to this problem, verifiable computing (VC) provides proofs of computational integrity without any assumptions on hardware or on potential failures. Existing VC systems can theoretically prove and verify all **NP** computations [8]. Nevertheless, despite the variety of existing solutions, existing VC schemes have to make trade-offs between expressiveness and functionality [20] and therefore cannot efficiently handle the verifiability of a sequence of operations with a high

variance in nature and complexity, like the ones involved in machine learning techniques. Even if expressive VC schemes such as Pinocchio [16] can ensure the verifiability of a machine learning algorithm, the cryptographic work required to produce the proof prevents from dealing with large but simple computations such as matrix multiplications. On the other hand, some schemes like Cormode et al.'s CMT [6] are very efficient and can deal with large computations, e.g. large matrix multiplications, but cannot handle the variety of even very simple operations such as number comparisons. Hence there is a need for a VC scheme that achieves both efficiency by handling complex operations and expressiveness through the variety of types of operations it can support. In this paper, we propose a scheme that combines a general purpose VC scheme like Pinocchio [16] or Groth's scheme [13] and various specialized VC schemes that achieve efficient verification of complex operations like large matrix multiplications.
Thanks to our proof composition scheme, the resulting VC scheme:

1. efficiently addresses the verifiability of a sequence of operations,
2. inherits the properties of the outer scheme, notably a short and single proof for a complex computation and privacy for inputs supplied by the prover.

In order to highlight the relevance of our proposal, we sketch the application of the resulting scheme on a neural network, which is a popular machine learning technique achieving state of the art performance in various classification tasks such as handwritten digit recognition, object or face recognition. Furthermore we propose a concrete instance of our scheme, using a Pinocchio-like scheme [13] and the Sum-Check protocol [15]. Thanks to our composition techniques, we are able to achieve unprecedented performance gains in the verifiability of computations involving large matrix multiplication and non-linear operations.

1.1 Problem Statement

Most 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:

$$g(x) = W_2 \cdot f(W_1 \cdot x) \tag{1}$$

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. Denoting X a batch of inputs to classify, the batched version of (1) therefore is:

$$g(X) = W_2 \cdot f(W_1 \cdot X) \tag{2}$$

In an attempt to assure the verifiability of this neural network, two alternative VC schemes seem potentially suited: the CMT protocol [6] based on interactive proofs and schemes deriving from Pinocchio [16]. CMT can efficiently deal with the matrix products but problems arise when it comes to the non-linear part of the operations 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. [11] and [6] 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-like schemes eliminate the latter problem by allowing for efficient verification of a 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 [20]).
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.

1.2 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. We apply this generic method to a pair of VC schemes, assuming that one is a general purpose VC scheme, called `GVC`, like Pinocchio [16], which can efficiently assure the verifiability of an application consisting of a sequence of functions, whereas the other VC scheme, which we call `EVC`, 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. 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

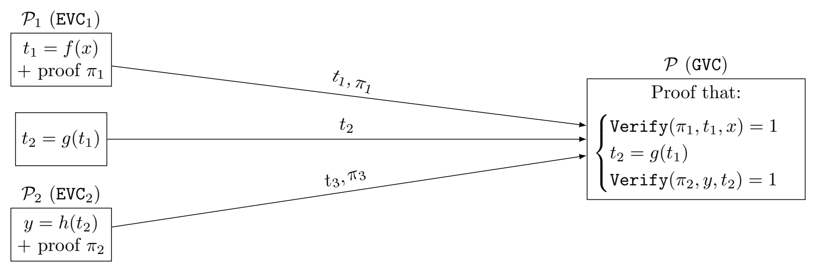
$\mathcal{P}_1$ ($\mathtt{EVC}_1$)
$t_1 = f(x)$
+ proof π_1
$t_2 = g(t_1)$
$\mathcal{P}_2$ ($\mathtt{EVC}_2$)
$y = h(t_2)$
+ proof π_2
t_1, π_1
t_2
t_3, π_3
$\mathcal{P}$ ($\mathtt{GVC}$)
Proof that:
$\mathtt{Verify}(\pi_1, t_1, x) = 1$
$t_2 = g(t_1)$
$\mathtt{Verify}(\pi_2, y, t_2) = 1$

Fig. 1: High level view of the embedded proofs

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 a running example the Neural Network defined with formula (2) in Section 1.1. Here, the application consists of the sequential execution of three functions f, g and h (see Figure 1), where f and h are not suitable to be efficiently proved using 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 ultimate goal therefore is to verify $y = h(g(f(x)))$. In our example, $f : X \mapsto W_1 \cdot X$, $h : X \mapsto W_2 \cdot X$ 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 $\mathtt{EVC}_1$ and $\mathtt{EVC}_2$ that are specialized schemes yielding efficient proofs with such functions. $\pi_{\mathtt{EVC}_1}$ denotes the proof generated by $\mathtt{EVC}_1$ on f, $\pi_{\mathtt{EVC}_2}$ denotes the proof generated by $\mathtt{EVC}_2$ on h and $\Pi_{\mathtt{GVC}}$ denotes the proof generated by GVC. For the sequential execution of functions f, g and h, denoting $t_1 = f(x)$ and $t_2 = g(t_1)$, the final proof then is:

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

Here the GVC system verifies the computation of g and the verification algorithms of the $\mathtt{EVC}_1$ and $\mathtt{EVC}_2$ systems, which output 1 if the proof is accepted and 0 otherwise. We note that this method can easily be extended to applications involving more than three functions, Section 3 describes the embedded proof protocol for an arbitrary number of functions. Interestingly, various specialized VC techniques can be selected as EVC based on their suitability to the special functions requirements provided that:

1. The verification algorithm of each EVC proof is compatible with the GVC scheme.
2. The verification algorithm of 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 [8]. 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 any Pinocchio-like scheme as the GVC and an efficient interactive proof protocol, namely the Sum-Check protocol [15] as the EVC. We further develop this instance with a Neural Network verification example.

1.3 Related Work

Verifying computation made by an untrusted party has been studied for a long time, but protocols leading to practical implementations are recent, see [20] and the references therein for details. Most of these proof systems build on quadratic arithmetic programs [8] and we focus on zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) schemes [3]. Proof composition for SNARKs have been proposed by Bitansky et al. [5] and the implementation of SNARKs recursive composition has later been proposed by Ben-Sasson et al. in [4]. 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 for the base field to have the same size as the elliptic curve group order. Ben-Sasson et al. therefore propose a cycle of elliptic curves to enable proof composition. When two such elliptic curves form a cycle, the finite field defined by the prime divisor in the group order of the first curve is equal to the base field (or field of definition) of the second curve and vice versa. 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 a part of a computation by sub-circuit verifying the sub-computation that can then be executed outside the prover.

In SafetyNets [9], 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 using state of the art activation functions such as ReLU. Indeed, Ghodsi et al. replace ReLU functions by a quadratic activation function, namely $x \mapsto x^2$, which squares the input values element-wise. This solution unfortunately causes instability during the training phase of the network compared to ReLU functions. A second disadvantage is the impossibility for the prover to prove a non-deterministic computation, i.e. to prove the correctness of a computation while hiding some inputs. 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 proposed method the prover could keep the model private and nonetheless be able to produce a proof of correct execution.

1.4 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. Following our embedded proof protocol, we first describe a VC scheme involving composition in Section 3 and then present a specialized instance of the GVC and EVC schemes to fit the Neural Network use-case in section 4. We report experimental results on the implementation of the latter scheme in Section 5 and conclude in Section 6. A security proof of our scheme is given in Appendix A and prover's input privacy are considered in Appendix B.

2 Building blocks

2.1 GVC: Verifiable Computation based on QAPs

Quadratic Arithmetic Programs In [8], 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 [16]). *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 $k \in \{0, \ldots, m\}$ 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 $(c_1, \ldots, c_N) \in \mathbb{F}^N$ is a valid assignment for function F if and only if* *there exists coefficients* $(c_{N+1}, \ldots, c_m)$ *such that* $t(x)$ *divides* $p(x)$*, as follows:*

$$p(x) = \left(v_0(x) + \sum_{k=1}^{m} c_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} c_k \cdot w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} c_k \cdot y_k(x)\right). \tag{4}$$

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 [16]. 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. In the verification phase, the divisibility check along with checks to ensure 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 are performed with a pairing. 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 algorithm* `KeyGen` *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 algorithm* `Verify` *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, namely *correctness*, *soundness* and *efficiency* have been formally defined in [8].

Costs In QAP-based protocols, the proof consists of few elliptic curve elements, e.g. 8 group elements in Pinocchio [16] or 3 group elements in Groth's state of the art VC system [13]. It has constant size no matter the computation to be verified, thus the verification is fast. In 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 pre-processing verifiable computation. This setup phase has to be run once, the keys are reusable for

later inputs and the cost of the pre-processing is amortized over all further computations. The bottleneck of the scheme is the prover computations: 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 randomize the proof by adding multiples of the target polynomial $t(x)$ to hide inputs he supplied in the computation. The proof obtained using Parno et al.'s protocol [16] or Groth's scheme [13] is thus a zero-knowledge Succinct Non-Interactive Argument (zk-SNARK). In the zk-SNARKs setting, results are meaningful even if the efficiency requirement is not satisfied 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 [15] enables to prove the correctness of the sum of a multilinear polynomial over a subcube, the protocol is a public coin interactive proof with n rounds of interaction. Suppose that P is a polynomial with n variables defined over $\mathbb{F}^n$. Using the Sum-Check protocol, a prover $\mathcal{P}$ can convince a verifier $\mathcal{V}$ that he knows the evaluation of P over $\{0,1\}^n$, namely:

$$H = \sum_{t_1 \in \{0,1\}} \sum_{t_2 \in \{0,1\}} \cdots \sum_{t_n \in \{0,1\}} P(t_1, \ldots, t_n) \tag{5}$$

While a direct computation performed by the verifier would require at least 2^n evaluations, the Sum-Check protocol only requires $O(n)$ evaluations for the verifier. $\mathcal{P}$ first computes $P_1(x) = \sum_{t_2 \in \{0,1\}} \cdots \sum_{t_n \in \{0,1\}} P(x, t_2, \ldots, t_n)$ and sends it to $\mathcal{V}$, who checks if $H = P_1(0) + P_1(1)$. If so, $\mathcal{P}$'s claim on P_1 holds, otherwise $\mathcal{V}$ rejects and the protocol stops. $\mathcal{V}$ picks a random value $r_1 \in \mathbb{F}$ and sends it to $\mathcal{P}$, who computes $P_2 = \sum_{t_3 \in \{0,1\}} \cdots \sum_{t_n \in \{0,1\}} P(r_1, x, t_3, \ldots, t_n)$. Upon receiving P_2, $\mathcal{V}$ checks if: $P_1(r_1) = P_2(0) + P_2(1)$. The protocol goes on until the nth round where $\mathcal{V}$ receives the value $P_n(x) = P(r_1, r_2, \ldots, r_{n-1}, x)$. $\mathcal{V}$ can now pick a last random field value r_n and check that: $P_n(r_n) = P(r_1, \ldots, r_n)$. If so, $\mathcal{V}$ is convinced that H has been evaluated as in (5), otherwise $\mathcal{V}$ rejects H. The Sum-Check protocol has the following properties:

1. The protocol is *correct*: if $\mathcal{P}$'s claim about H is true, then $\mathcal{V}$ accepts with probability 1.
2. The protocol is *sound*: 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, QAP-based VC schemes require the complete specification of each computation as input to the 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 [7] to obtain a non-interactive version of the Sum-Check protocol that can be used as an input to `GVC`. 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 [17].

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 [18] 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, a *multilinear extension* (MLE) of a function $f : \{0,1\}^d \to \mathbb{F}$ is a polynomial that agrees with f on $\{0,1\}^d$ and has degree at most 1 in each variable. Any function $f : \{0,1\}^d \to \mathbb{F}$ has a unique multilinear extension over $\mathbb{F}$, which we will denote hereafter by $\tilde{f}$. Using Lagrange interpolation, an explicit expression of MLE can be obtained:

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

$$\forall (x_1, \ldots, x_d) \in \mathbb{F}^d,\ \tilde{f}(x_1, \ldots, x_d) = \sum_{w \in \{0,1\}^d} f(w) \chi_w(x_1, \ldots, x_d) \qquad (6)$$

where: $w = (w_1, \ldots, w_d)$ *and* $\chi_w(x_1, \ldots, x_d) = \prod_{i=1}^{d} \big(x_i w_i + (1-x_i)(1-w_i)\big)$ (7)

2.4 Ajtai Hash Function

As mentioned in Section 1.1, 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 [7]. This transformation needs 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 would be too high: [2] reports about 27,000 multiplicative gates to implement the compression function of SHA256. Instead, we choose a function better suited for arithmetic circuits, namely the Ajtai hash function [1] that is based on the subset sum problem as defined below:

Definition 3. *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:*

$$\forall x \in \{0,1\}^m, \quad H_{n,m,q} = A \times x \mod q \tag{8}$$

As proved by Goldreich et al. [10], the collision resistance of the hash function relies on the hardness of the Short Integer Solution (SIS) problem. The function is also *regular*: it maps an uniform input to an uniform output. Ben-Sasson et al. [4] 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 [14]. 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 since it involves multiplications by constants (the matrix A is public): to hash m bits, m multiplicative gates are needed to ensure that the input vector is binary and 3 more gates are needed to ensure that the output is the linear combination of the input and the matrix. With the parameters selected in [14], this means that 1527 gates are needed to hash 1524 bits.

3 Embedded Proofs

3.1 High level description of the generic protocol

Let us consider two sets of functions $(f_i)_{1\leq i\leq n}$ and $(g_i)_{1\leq i\leq n}$ such that the f_i do not lend themselves to an efficient verification with the GVC system whereas the g_i can be handled by the GVC system efficiently. For an input x, we denote by y the evaluation of x by the function $g_n \circ f_n \circ \ldots g_1 \circ f_1$. In our embedded proof protocol, each function f_i is handled by a sub-prover $\mathcal{P}_i$ while the g_i functions are handled by the prover $\mathcal{P}$. The sub-prover $\mathcal{P}_i$ is in charge of the efficient VC algorithm $\mathtt{EVC}_i$ and the prover $\mathcal{P}$ runs the GVC algorithm. The steps of the proof generation are depicted in Figure 2. Basically, each sub-prover $\mathcal{P}_i$ will evaluate the function f_i on a given input, produce a proof of correct evaluation using the $\mathtt{EVC}_i$ system and pass the output of f_i and the related proof π_i to $\mathcal{P}$, who will compute the next g_i evaluation and pass the result to the next sub-prover $\mathcal{P}_{i+1}$. In the **Setup** phase, the verifier and the prover agree on an arithmetic circuit which describes the computation of the functions g_i along with the verification algorithms of the proof that the functions f_i were correctly computed. The preprocessing phase of the GVC system takes the resulting circuit and outputs the corresponding evaluation and verification keys.

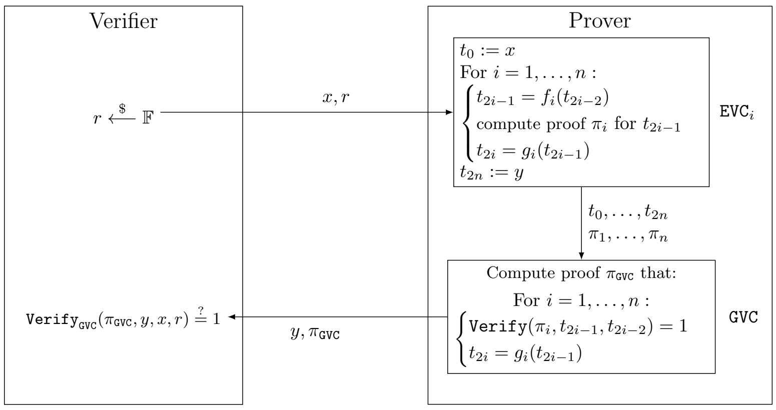
Verifier
Prover
$r \xleftarrow{\$} \mathbb{F}$
x, r
$t_0 := x$
For $i = 1, \ldots, n$:
$t_{2i-1} = f_i(t_{2i-2})$
compute proof π_i for t_{2i-1}
$t_{2i} = g_i(t_{2i-1})$
$t_{2n} := y$
$\mathtt{EVC}_i$
$t_0, \ldots, t_{2n}$
$\pi_1, \ldots, \pi_n$
Compute proof $\pi_{\mathtt{GVC}}$ that:
For $i = 1, \ldots, n$:
$\mathtt{Verify}(\pi_i, t_{2i-1}, t_{2i-2}) = 1$
$t_{2i} = g_i(t_{2i-1})$
GVC
$\mathtt{Verify}_{\mathtt{GVC}}(\pi_{\mathtt{GVC}}, y, x, r) \overset{?}{=} 1$
$y, \pi_{\mathtt{GVC}}$

Fig. 2: Embedded proof protocol

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 $\mathcal{P}_i$. In the **proving** phase, $\mathcal{P}_1$ first computes $t_1 = f(x)$ and produces a proof π_1 of the correctness of the computation, using the efficient proving algorithm $\mathtt{EVC}_1$. The prover $\mathcal{P}$ then computes the value $t_2 = g_1(t_1)$ and passes the value t_2 to $\mathcal{P}_2$, who computes $t_3 = f_2(t_2)$ along with the proof of correctness π_2, using the $\mathtt{EVC}_2$ proving system. The protocol proceeds until $y = t_{2n}$ is computed. Finally, $\mathcal{P}$ provides the inputs/outputs of the computations and the intermediate proofs π_i to the GVC system and, using the evaluation key computed in the setup phase, builds a proof $\pi_{\mathtt{GVC}}$ that for $i = 1, \ldots, n$:

1. the proof π_i computed with the $\mathtt{EVC}_i$ system is correct,
2. the computation $t_{2i} = g_i(t_{2i-1})$ is correct.

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 lower than the one of 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 into our scheme.

3.2 A Protocol instance

In this section, we specify the embedded proofs protocol in the case where f_i are matrix products $f_i : X \mapsto W_i \times X$ and where the functions g_i cannot be efficiently verified by a VC system except by $\mathtt{GVC}$. We use the Sum-Check protocol to prove correctness of the matrix multiplications, as in [18] and any QAP-based VC scheme as the global proof mechanism. We assume that the matrices involved in the f_i functions do not have the same sizes so there will be several instances of the Sum-Check protocol. It thus makes sense to define different efficient proving algorithms $\texttt{EVC}_i$ since the GVC scheme 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 Sum-Check verification protocols are distinct as arithmetic circuits. For the sake of simplicity, the W_i matrices are assumed to be square matrices of size n_i. We denote $d_i = \log n_i$ and assume that $n_i \geq n_{i+1}$. We denote by H the Ajtai hash function (see Section 2). The protocol between the verifier $\mathcal{V}$ and the prover $\mathcal{P}$, which has n sub-provers $\mathcal{P}_i$ is the following:

Setup:
- $\mathcal{V}$ and $\mathcal{P}$ agree on an arithmetic circuit $\mathcal{C}$ description for the computation. $\mathcal{C}$ implements both the evaluations of the functions g_i and the verification algorithms of the Sum-Check protocols for the n matrix multiplications.
- $(EK_{\mathcal{C}}, VK_{\mathcal{C}}) \leftarrow \texttt{KeyGen}(1^\lambda, \mathcal{C})$

Query
- $\mathcal{V}$ generates a random challenge (r_L, r_R) such that: $(r_L, r_R) \in \mathbb{F}^{d_1} \times \mathbb{F}^{d_1}$
- $\mathcal{V}$ sends $\mathcal{P}$ the tuple (X, r_L, r_R), where X is the input matrix.

Proof : for $i = 1, \ldots, n$, on input (T_{2i-2}, r_L, r_R),

Sub-prover $\mathcal{P}_i$:
- computes the product $T_{2i-1} = W_i \times T_{2i-2}$, (denoting $T_0 := X$)
- computes r_{L_i} and r_{R_i} (the d_i first component of r_L and r_R),
- computes the multilinear extension evaluation $\widetilde{T}_{2i-1}(r_{L_i}, r_{R_i})$
- computes with serialized Sum-Check, the proof π_i of P_i evaluation:

$$P_i(x) = \widetilde{W}_i(r_{L_i}, x) \cdot \widetilde{T}_{2i-2}(x, r_{R_i}) \text{ where } x = (x_1, \ldots, x_{d_i}) \in \mathbb{F}^{d_i}. \quad (9)$$

- sends the tuple $(T_{2i-2}, T_{2i-1}, W_i, \pi_i, r_{L_i}, r_{R_i})$ to prover $\mathcal{P}$.

Prover $\mathcal{P}$:
- computes $T_{2i} = g_i(T_{2i-1})$ and sends (T_{2i}, r_L, r_R) to sub-prover $\mathcal{P}_{i+1}$
- receiving the inputs $\{(T_{2i-2}, T_{2i-1}, W_i, \pi_i, r_{L_i}, r_{R_i})\}_{i=1,\ldots,n}$ from sub-provers:
  - Computes $\widetilde{T}_{2i-1}(r_{L_i}, r_{R_i})$.
  - Parses π_i as $(P_{i,1}, r_{i,1}, P_{i,2}, r_{i,2}, ..., P_{i,d_1}, r_{i,d_i})$, where the proof contains the coefficient of the degree two polynomials $P_{i,j}$ that we denote by $(a_{i,j}, b_{i,j}, c_{i,j})$ if: $P_{i,j}(x) = a_{i,j}x^2 + b_{i,j}x + c_{i,j}$
  - Verifies π_i:
    - Checks: $P_{i,1}(0) + P_{i,1}(1) \stackrel{?}{=} \widetilde{T}_{2i-1}(r_{L_i}, r_{R_i})$
    - Computes: $r_{i,1} = \left(\sum_j r_{L_i}[j]\right) \cdot \left(\sum_j r_{R_i}[j]\right)$
    - For $j = 2, \ldots, d_i$:
      - Check: $P_{i,j}(0) + P_{i,j}(1) \stackrel{?}{=} P_{i,j-1}(r_{i,j-1})$
      - Computes: $r_{i,j}$ as the product of components of the Ajtai hash function output, i.e. $r_{i,j} = \prod_{k=1}^{3} H(a_{i,j}, b_{i,j}, c_{i,j}, r_{i,j})[k]$

  - ∗ From T_{2i-2} and W_i, computes the evaluated multilinear extensions $\widetilde{W}_i(r_{L_i}, r_{i,1}, \ldots, r_{i,d_1})$ and $\widetilde{T}_{2i-2}(r_{i,1}, \ldots, r_{i,d_1}, r_{R_i})$
  - ∗ Checks that $P_{d_i}(r_{i,d_i})$ is the product of the multilinear extensions $\widetilde{W}_i(r_{L_i}, r_{i,1},..., r_{i,d_i})$ and $\widetilde{T}_{2i-2}(r_{i,1},..., r_{i,d_i}, r_{R_i})$.
- Aborts if one of the previous checks fails. Otherwise, accepts T_{2i-1} as the product of W_i and T_{2i-2}.
- Repeat the above instructions until the proof π_n has been verified.
- Using the GVC scheme, computes the final proof $\pi_{\texttt{GVC}}$ that all the $\texttt{EVC}_i$ proofs π_i have been verified and all the T_{2i} values have been correctly computed from T_{2i-1}.
- Sends $(Y, \pi_{\texttt{GVC}})$ to the Verifier.

Verification

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

– If Verify fails, verifier rejects the value Y. Otherwise the value Y is accepted as the result of: $Y = g_n(...(g_2(W_2(g_1(W_1 \cdot X))))...)$

4 Embedded proofs for Neural Networks

4.1 Motivation

In order to show the relevance of the proposed embedded proof scheme, we apply the resulting scheme to Neural Networks (NN), which are machine learning techniques achieving state of the art performance in various classification tasks such as handwritten digit recognition, object or face recognition. As stated in Section 1.1, a NN can be viewed as a sequence of operations, the main ones being linear operations followed by so-called activation functions. The linear operations are modeled as matrix multiplications while the activation functions are non-linear functions. A common activation function choice is the ReLU function defined by: $x \mapsto \max(0, x)$. Due to the sequential nature of NNs, a simple solution to obtain a verifiable NN would consist of computing proofs for each part of the NN sequence. However, this solution would degrade the verifier's performance, increase the communication costs and force the prover to send all the intermediate results, revealing sensitive data such as the parameters of the prover's NN. On the other hand, even if it is feasible in principle to implement the NN inside a GVC system like Pinocchio, the size of the matrices involved in the linear operations would be an obstacle. The upper bound for the total number of multiplications QAP-based VC schemes can support as part of one application is estimated at 10^7 [19]. This threshold would be reached with a single multiplication between two 220×220 matrices. In contrast, our embedded proof protocol enables to reach much larger matrix sizes or, for a given matrix size, to perform faster verifications of matrix multiplications.

4.2 A Verifiable Neural Network Architecture

We here describe how our proposal can provide benefits in the verification of a neural network (NN) [12]: in the sequel, we compare the execution of a GVC

protocol on a two-layer NN with the execution of the embedded proof protocol on the same NN. Since NN involve several matrix multiplications, embedded proofs enable substantial gains, see Section 5.2 for implementation report. We stress that we consider neural networks in the *classification* phase, which means we consider that all the values have been set during the *training* phase, using an appropriate set of labeled inputs.
The NN we verify starts with a fully connected layer combined with a ReLU activation layer. We then apply a max pooling layer to decrease the dimensions and finally apply another fully connected layer. The execution of the NN can be described as: $\boxed{\text{INPUT}} \rightarrow \boxed{\text{FC}} \rightarrow \boxed{\text{RELU}} \rightarrow \boxed{\text{MAX POOLING}} \rightarrow \boxed{\text{FC}}$
The *fully connected layer* takes as input a value and performs a dot product between this value and a parameter that can be learned. Gathering all the fully connected layer parameters in a matrix, the operation performed on the whole inputs is a matrix multiplication. The *ReLU* layer takes as input a matrix and performs the operation $x \mapsto \max(0, x)$ element-wise. The *max pooling* layer takes as input a matrix and return a matrix with smaller dimensions. This layer applies a max function on sub-matrices of the input matrix, which can be considered as sliding a window over the input matrix and taking the max of all the values belonging to the window. The size of the window and the number of inputs skipped between two mapping of the max function are parameters of the layer but do not change during the training phase nor on the classification phase. Usually a 2×2 window slides over the input matrix, with no overlapping over the inputs. Therefore, the MaxPool function takes as input a matrix and outputs a matrix which row and column size have been divided by 2. Denoting by W_1 and W_2 the matrices holding the parameters of the fully connected layers, X the input matrix, and Y the output of the NN computation, the whole computation can be described as a sequence of operations:

$$X \rightarrow T_1 = W_1 \cdot X \rightarrow T_2 = \text{ReLU}(T_1) \rightarrow T_3 = \text{MaxPool}(T_2) \rightarrow Y = W_2 \cdot T_3 \quad (10)$$

5 Implementation and Performance Evaluation

We ran two sets of experiments to compare the cost of execution between our embedded proof scheme and a baseline scheme using the `GVC` scheme. The first set focuses only on the cost of a matrix multiplication since these are a relevant representative of complex operations whereby the embedded proof scheme is likely to achieve major performance gains. The second set takes into account an entire application involving several operations including matrix multiplications, namely a small neural network architecture.

5.1 Matrix multiplication benchmark

We implemented our embedded proof protocol on a 8-core machine running at 2.9 GHz with 16 GB of RAM. The GVC system is Groth's state of the art zk-

Table 1: Matrix multiplication benchmark

(a) Matrix multiplication proving time

n	16	32	64	128	256	512
Baseline (GVC only)	0.23 s	1.34 s	9.15 s	71.10 s	697.72	–
Embedded proofs	0.281 s	0.924 s	3.138 s	11.718 s	43.014 s	168.347 s
Time division	0.28|0.001	0.92|0.004	3.12|0.018	11.65|0.068	42.71|0.304	166.88|1.467

(b) Matrix multiplication key generation time

n	16	32	64	128	256	512
Baseline (GVC only)	0.28 s	1.56 s	10.50 s	76.62 s	585.21 s	–
Embedded proofs	0.37 s	1.03 s	3.54 s	12.95 s	47.52 s	176.41 s

(c) Matrix multiplication key generation size

n	16	32	64	128	256	512
Baseline (GVC only) PK	508 KB	5.60 MB	26.9 MB	208 MB	1.63 GB	–
Embedded proofs PK	757 kB	2.24 MB	7.87 MB	30.1 MB	118.7 MB	472 MB
Baseline (GVC only) VK	31 kB	123 kB	490 kB	1.96 MB	7.84 MB	–
Embedded proofs VK	32 KB	123 KB	491 KB	1.96 MB	7.84 MB	31.36 MB

SNARK [13] and is implemented using the `libsnark` library [4] while the EVC system is our own implemention of Thaler's special purpose matrix multiplication verification protocol [18] using the NTL library [5]

The proving time reported in table 1a measures the time to produce the proof using the EVC system and to verify the result inside the GVC system. The last row of the table breaks down the proving time into the sumcheck proof time and the embedded proof time. We note that the sumcheck proving time brings a very small contribution to the final proving time. For the value $n = 512$, the proof using the `GVC` is not feasible whereas the embedded proof approach still achieves realistic performance. Table 1b compares the key generation time using the embedded proof system with the one using the `GVC`. Table 1c states the sizes of the proving key (PK) and the verification key (VK) used in the previous scenarios. The embedded proof protocol provides substantial benefits: the protocol improves the proving time as soon as the matrix has a size greater than 32×32, giving a proving time 7 times better for 128×128 matrix multiplication and 16 times better for a 256×256 matrix multiplication. Embedded proofs also enable to reach higher value for matrix multiplications: we were able to perform a multiplication proof for 512×512 matrices whereas the computation was not able to terminate due to lack of RAM for the baseline system.

[4] Libsnark, a C++ library for zkSNARK proofs, available at `https://github.com/scipr-lab/libsnark`

[5] V. Shoup, NTL – A Library for Doing Number Theory, available at `http://www.shoup.net`

Table 2: Experiments on 2-layer networks

(a) NN-64-32

	`KeyGen`	PK size	VK size	Prove	Verify
Baseline (GVC only)	59 s	148 MB	490 kB	25.48 s	0.011 s
Embedded proofs	44 s	123 MB	778 kB	16.80 s	0.016 s

(b) NN-128-64

	`KeyGen`	PK size	VK size	Prove	Verify
Baseline (GVC only)	261.9 s	701.5 MB	1.96 MB	149.5 s	0.046 s
Embedded proofs	162.7 s	490 MB	3.1 MB	66.96 s	0.067 s

5.2 Two-Layer Verifiable Neural Network Experimentations

We implemented the verification of an example of 2-layer neural network, which can be seen as one matrix multiplication followed by the application of two non-linear functions, namely a ReLU and a max pooling function as described in Section 4. For our experiments, the max pooling layers have filters of size 2×2 and no data overlap. Thus, setting for instance the first weight matrix to 64×64, the second weight matrix size is 32×32; we denote by NN-64-32 such a neural network. Table 2a reports experiments on a 2-layer neural network with a first 64×64 matrix product, followed by a ReLU and a max-pooling function, and ending with a second 32×32 matrix product. Experimental times for a NN-128-64 network (with the same architecture as above) are reported in table 2b.

Experiments show a proving time twice better than using the baseline proving system. The overall gain is lower than for the matrix product benchmark because the other operations (ReLU and max pooling) are implemented the same way for the two systems. It should be noted that the goal of the implementation was to achieve a proof of concept for our scheme on a complete composition scenario involving several functions rather than putting in evidence the performance advantages of the scheme over the baseline, hence the particularly low size of the matrices used in the 2-layer NN and an advantage as low as the one in table 2a and table 2b. The gap between the embedded proof scheme and the baseline using a realistic scenario with larger NN would definitely be much more significant due to the impact of larger matrices as shown in the matrix product benchmark.

6 Conclusion

We designed an efficient verifiable computing scheme that builds on the notion of proof composition and leverages an efficient VC scheme, namely the Sum-Check protocol to improve the performance of a general purpose QAP-based VC protocol, in proving matrix multiplications. As an application, our scheme can prove the correctness of a neural network algorithm. We implement our scheme and provide an evaluation of its efficiency. The security is evaluated in Appendix A. We stress that the composition technique described in the article can be extended using other efficient VC schemes and an arbitrary number of sequential function evaluations, provided that they respect the requirements defined in Section 1.2. Our proposal could be integrated as a sub-module in existing verifiable computing systems in order to improve their performance when verifying computations including high complexity operations such as matrix multiplications.

Acknowledgment

The authors would like to thank Gaïd Revaud for her precious programming assistance. This work was partly supported by the TREDISEC project (G.A. no 644412), funded by the European Union (EU) under the Information and Communication Technologies (ICT) theme of the Horizon 2020 (H2020) research and innovation programme.

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. 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)
3. 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)
4. 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)
5. 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
6. 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)
7. 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)
8. 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)

9. 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`
10. 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`
11. 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)
12. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), `http://www.deeplearningbook.org`
13. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II. pp. 305–326 (2016)
14. 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`
15. 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)
16. 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)
17. 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`
18. 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)
19. 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)
20. 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`

A Appendix: Embedded Proofs Security

Our embedded proof system has to satisfy the correctness and soundness requirements. Suppose that we have a GVC and n EVC systems to prove the correct computation of $y = g_n \circ f_n \circ \ldots \circ g_1 \circ f_1(x)$. We will denote by $\mathtt{EVC}_i$, $i = 1, \ldots, n$ the EVC systems. We also keep notations defined in Section 3: the value t_i, $i = 0, \ldots, 2n$ represents intermediate computation results, t_{2i-1} being the output of the f_i function, t_{2i} being the output of the g_i function, $t_0 := x$ and $t_{2n} = y$. The $\mathtt{EVC}_i$ and GVC systems already satisfy the correctness and soundness requirements. Let denote by $\epsilon_{\mathtt{GVC}}$ the soundness error of the GVC system and $\epsilon_{\mathtt{EVC}_i}$ the soundness error of the $\texttt{EVC}_i$ system. Note that while the $\texttt{EVC}_i$ systems prove that $t_{2i-1} = f_i(t_{2i-2})$ have been correctly computed, the $\texttt{GVC}$ system proves the correctness of $2n$ computations, namely that the *verification* of the $\texttt{EVC}_i$ proofs has passed and that the computations $t_{2i} = g_i(t_{2i-1})$ are correct. Furthermore, the $\texttt{GVC}$ system proves the correct execution of the function F that takes as input the tuple $(x, y, r, (t_i)_{i=1,\ldots,2n}, (\pi_i)_{i=1,\ldots,n})$ and outputs 1 if for all $i = 1, \ldots, n$, $\texttt{Verify}_{\texttt{EVC}_i}(\pi_i, t_{2i-1}, t_{2i-2}) = 1$ and $t_{2i} = g_i(t_{2i-1})$. F outputs 0 otherwise. For convenience, we denote by comp_n the function $g_n \circ f_n \circ \ldots \circ g_1 \circ f_1$.

A.1 Correctness

Theorem 1. *If the* $\texttt{EVC}_i$ *and the* $\texttt{GVC}$ *systems are correct then our embedded proof system is correct.*

Proof. Assume that the value $y = \mathrm{comp}_n(x)$ has been correctly computed. This means that for $i = 1, \ldots, n$, the values $t_{2i-1} = f_i(t_{2i-2})$ and $t_{2i} = g_i(t_{2i-1})$ have been correctly computed. Since the $\texttt{GVC}$ system is correct, it ensures that the function F will pass the $\texttt{GVC}$ verification with probability 1, provided that its result is correct. Now, since the $\texttt{EVC}_i$ systems are correct, with probability 1 we have that: $\texttt{Verify}_{\texttt{EVC}_i}(t_{2i-1}, t_{2i-2}, \pi_i) = 1$.

Therefore, if $y = \mathrm{comp}_n(x)$ 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.

A.2 Soundness

Theorem 2. *If the* $\texttt{EVC}_i$ *and the* $\texttt{GVC}$ *systems are sound with soundness error respectively equal to* $\epsilon_{\texttt{EVC}_i}$ *and* $\epsilon_{\texttt{GVC}}$*, then our embedded proof system is sound with soundness error at most* $\epsilon := \sum \epsilon_{\texttt{EVC}_i} + \epsilon_{\texttt{GVC}}$.

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. $y' \neq \mathrm{comp}(x)$ and π is accepted by the verifier $\mathcal{V}_{emb}$ with probability higher than ϵ. We then construct an adversary $\mathcal{B}$ that breaks the soundness property of either the $\texttt{GVC}$ or of one of the $\texttt{EVC}$ systems. We build $\mathcal{B}$ as follows: $\mathcal{A}_{emb}$ interacts with the verifier $\mathcal{V}_{emb}$ of the embedded system until a cheating proof is accepted. $\mathcal{A}_{emb}$ then forwards the cheating tuple $(x, y, r, (t_i)_{i=1,\ldots,2n}, (\pi_i)_{i=1,\ldots,n})$ for which the proof π has been accepted. Since $y' \neq \mathrm{comp}(x)$, there exists an index $i \in \{1, \ldots, n\}$ such that either $t_{2i-1} \neq f_i(t_{2i-2})$ or $t_{2i} \neq g_i(t_{2i-1})$. $\mathcal{B}$ can thus submit a cheating proof to the $\texttt{GVC}$ system or to one of the $\texttt{EVC}_i$ system, depending on the value of i.

Case $\mathbf{t_{2i-1} \neq f_i(t_{2i-2})}$:

By definition of the proof π, this means that the proof π_i has been accepted by the verification algorithm of $\texttt{EVC}_i$ implemented inside the $\texttt{GVC}$ system. $\mathcal{A}_{emb}$ can then forward to the adversary $\mathcal{B}$ the tuple $(t_{2i-1}, t_{2i-2}, \pi_i)$. Now if $\mathcal{B}$ presents the tuple $(t_{2i-1}, t_{2i-2}, \pi_i)$ to the $\texttt{EVC}_i$ system, it succeeds with probability 1.

Therefore, the probability that the verifier $\mathcal{V}_{emb}$ of the embedded proof system accepts is superior to $\epsilon_{\mathtt{EVC}_i}$, which breaks the soundness property of $\mathtt{EVC}_i$.

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

Case $\mathbf{t_{2i} \neq g_i(t_{2i-1})}$:
This means that the proof π computed by the GVC system is accepted by $\mathcal{V}_{emb}$ even if $t_{2i} \neq g_i(t_{2i-1})$ has not been correctly computed. We proceed as in the previous case: $\mathcal{A}_{emb}$ forwards $\mathcal{B}$ the cheating tuple and the cheating proof π. The tuple and the proof thus break the soundness of the GVC scheme because:

$$Pr[\mathcal{V}_{emb} \text{ accepts } \pi] = \epsilon \geq \epsilon_{\mathtt{GVC}}$$

B Appendix: Prover's input privacy

B.1 Prover's input privacy

The combination of the proof of knowledge and zero knowledge properties in zk-SNARK proofs enables the prover to provide some inputs for the computation to be proved for which no information will leak. Gennaro et al. proved in [8] that their QAP-based protocol (see Section 2.1) is zero-knowledge for input provided by the prover: there exists a simulator that could have generated a proof without knowledge of the witness (here the input privately provided by the prover). Subsequent works on QAP-based VC schemes achieve the same goal, with differences on the cryptographic assumptions and on the flavor of zero-knowledge achieved: for instance Groth's scheme [13] achieves perfect zero-knowledge at the expense of a proof in the generic group model while Gennaro et al.'s scheme achieves statistical zero-knowledge with a knowledge of exponent assumption.

We now sketch how the zero-knowledge property is achieved for our embedded proof protocol. We first have to assume that the QAP-based VC scheme we consider for GVC can support auxiliary inputs (as in NP statements), which is achieved for Pinocchio [16] or Groth's scheme [13]. Leveraging the zero-knowledge property, the GVC prover can hide from the verifier the intermediate results of the computation provided by the sub-provers $\mathtt{EVC}_i$ while still allowing the verifier to check the correctness of the final result. Therefore, the overall VC system obtained by composing the EVC systems inside the GVC achieves zero-knowledge: the simulator defined for the GVC system is still a valid one. Note that even if the simulator gains knowledge of the intermediate computations performed by sub-provers $\mathtt{EVC}_i$, the goal is to protect the leakage of information from *outside*, namely from the verifier. In detail, keeping the notations of figure 2, the verifier only knows t_0, i.e. x and t_{2n}, i.e. y, which are the inputs and outputs of the global computation. Intermediate inputs, such as $t_i, i = 1, \dots, 2n-1$, are hidden from the verifier even though they are taken into account during the verification of the intermediate proofs by the GVC prover. Therefore, thanks to the zk-SNARKs, the intermediate results are verified but not disclosed to the verifier.