SNNzkSNARK: An Efficient Design and Implementation of a Secure Neural Network Verification System Using zkSNARKs

Zachary DeStefano A-4
Villanova University
(Mentors: Michael Dixon and Juston Moore)

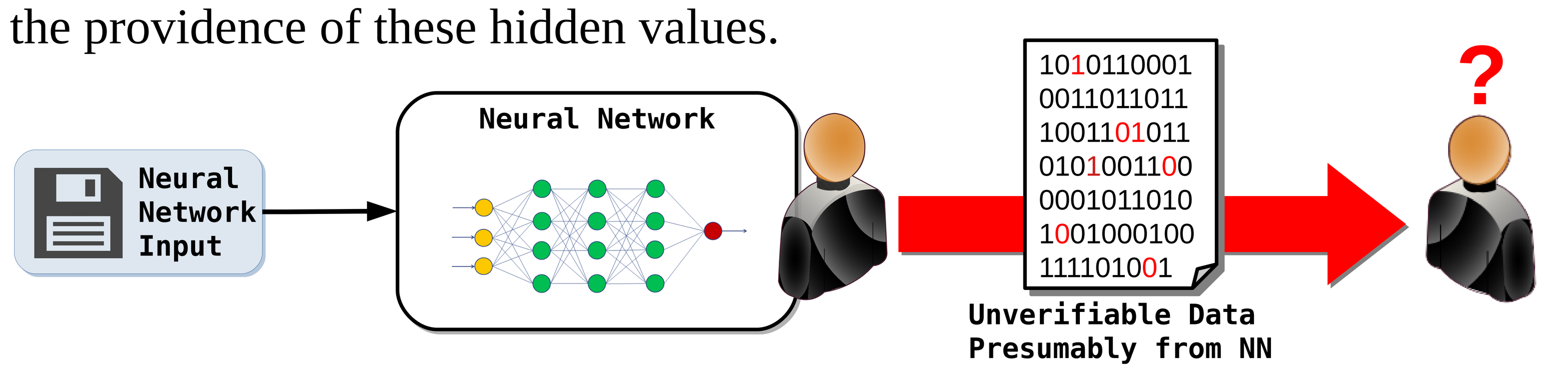
Abstract

We present an efficient and succinct zero-knowledge proof application using zkSNARKs for remotely verifying the forward-pass execution of an arbitrarily-sized neural network with hidden inputs and model parameters. Our zero-knowledge guarantee allows the prover to hide information about the input and model parameters from the verifier while being able to attest to the integrity of these parameters and the model's execution.

Our approach is transformative for various applications such as nuclear treaty verification without the need to disclose sensitive data, security camera auditing without the need to leak footage, and secure patient diagnosis without the need to disclose individually identifiable health information. We demonstrate an end-to-end implementation of this proof system using custom gadgets in libsnark on a neural network for the classification of MNIST handwritten digits.

Problem Statement

Neural Networks are increasingly being used for decision making and analysis; however, there does not exist an efficient method for remotely verifying the execution of a neural network or for hiding information about a neural network such as input or weights while maintaining verification of the providence of these hidden values.



In response to this problem, in this work, we present an efficient system for the remote verification of the execution of a neural network and the verification of the input to this program. We demonstrate the functionality of this program on the toy problem of digit classification using the MNIST database of handwritten digits.

Background

A **Zero-Knowledge Proof (π)** is a way to prove a claim without leaking details about why the claim is true.

π says "$\exists \vec{s}, \vec{in} : P(\vec{in}, \vec{out}, \vec{s}) = \top$" for any computable property P

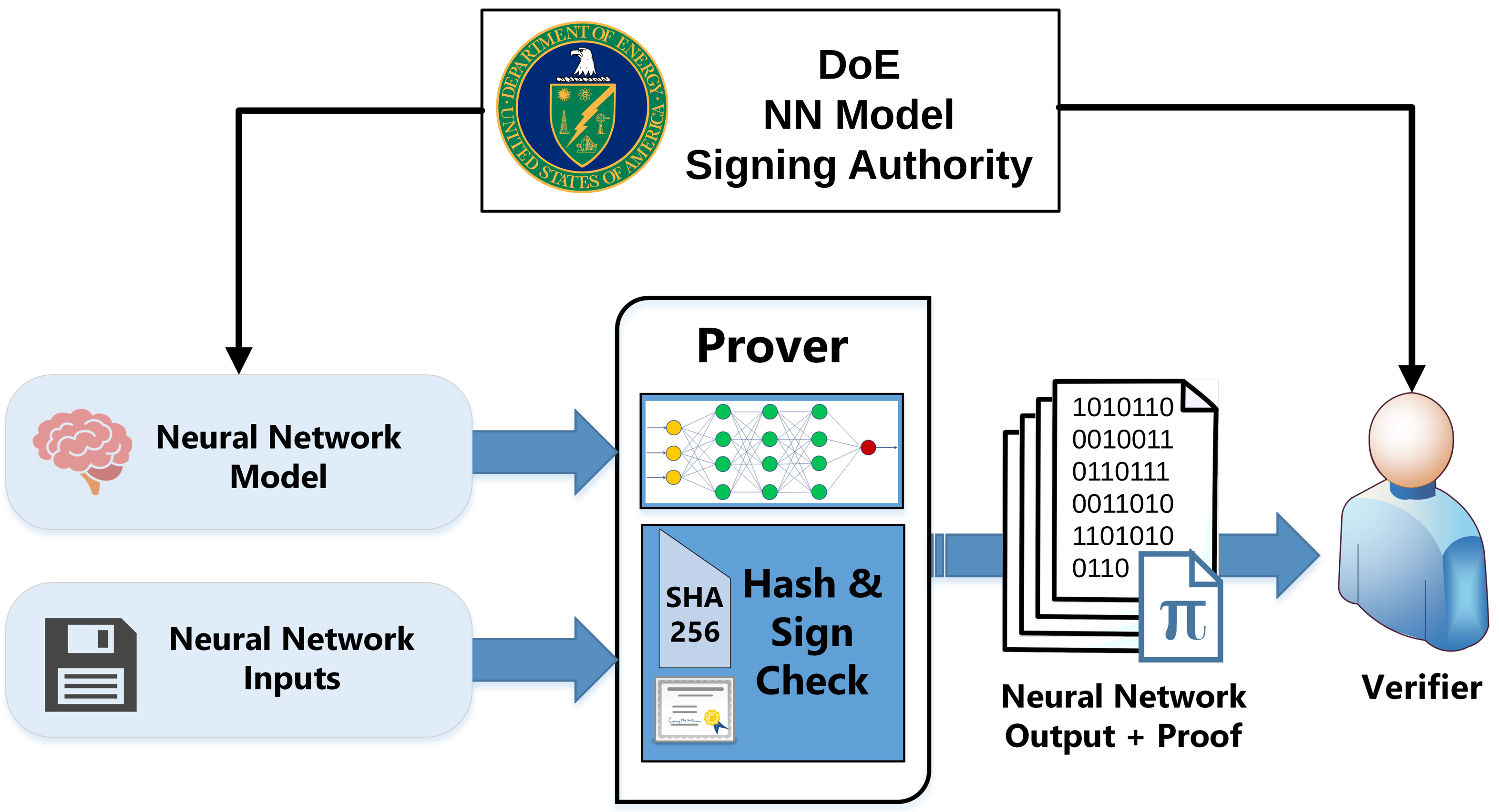
With **Soundness**: $\varphi(\vec{x}) = \bot \Rightarrow Pr[Ver(\pi) = \top] \leq \text{negl}(\vec{x})$
And **Completeness**: $\varphi(\vec{x}) = \top \Rightarrow Pr[Ver(\pi) = \top] = 1$

A **zkSNARK** is a
- **z**ero-**k**nowledge: No secret information is revealed by the proof
- **S**uccinct: The size of the proof that is generated is small : $O(1)$
- **N**on-interactive: no challenge-response protocol
- **AR**gument of **K**nowledge: It is computationally intractable for the prover to produce a fake proof

A **zkSNARK** can be compiled from a **R1CS (Rank 1 Constraint System)**
- **R1CS**: an expression or system of expressions in the form:
(S • A) * (S • B) - (S • C) = 0
Where S is a vector containing the values of all of the variables $(\vec{in}, \vec{out}, \vec{s})$

To prevent forged proofs, a computation must be **fully constrained**
- A problem is **fully constrained** *iff* for any **input** the system of constraints does not allow any false **output** to satisfy all constraints

System Architecture and Design

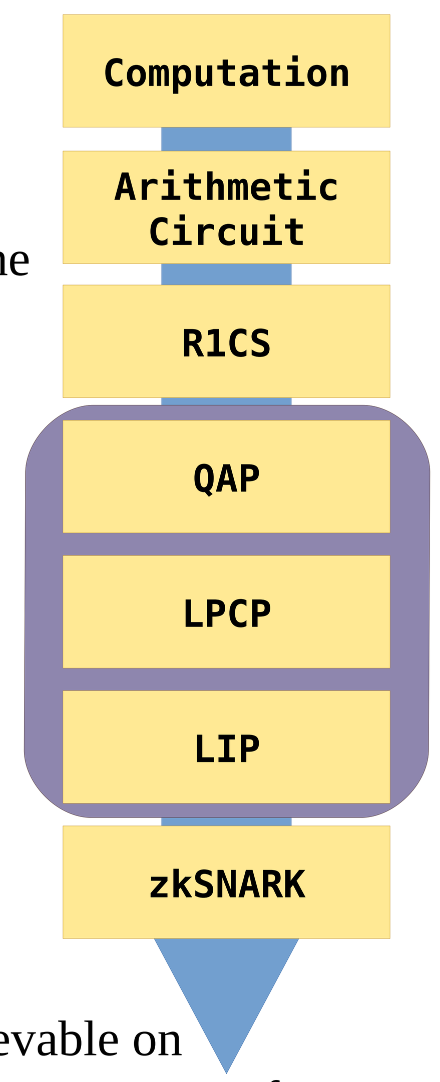


Above is a process view of our program's architecture. Arrows indicate the flow of data through the architecture, and also indicate which component (prover, verifier, or both) has access to the data.

In building the neural network verification system, we programmed a series of gadgets (program building blocks which execute and verify a specific computation given certain inputs and outputs) which provide the functionality required to efficiently execute and verify the execution of a neural network. These gadgets were built specifically for any size feed-forward neural network.

R1CS Optimization Strategies

The standard pipeline for the creation of zkSNARK generators and verifiers from general computation involves the pipeline (right). Due to the size of the pipeline, there is a large computational overhead that gets introduced in terms of both memory and total running time In this project, we wrote the program directly as a R1CS and perform optimizations firstly over the number of constraints and second over the size of the constraints to allow for larger and more complicated computations to be executed and verified with a smaller memory footprint.



For example, a traditional addition circuit with N full-adders to complete an N-bit addition requires ~5N constraints (5 per full-adder), but by taking advantage of the structure of R1CS, we reduce this addition to N+2 constraints with N+1 bitness checks (output bits and carry) and the following large constraint:

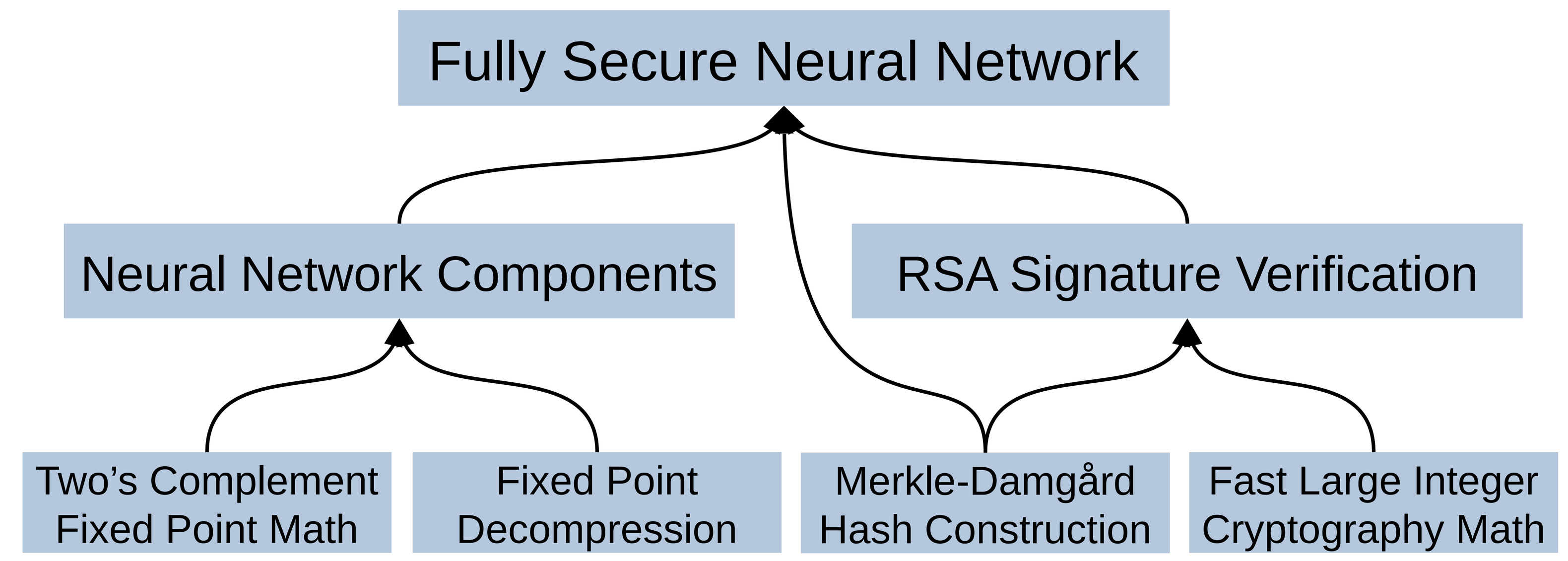
$$\sum_{i=0}^{N-1} 2^i (a_i + b_i - c_i) = 2^N t$$

This is an example of the level of simplification achievable on a single simple operation. It is possible to make use of the structure of combinations to reduce some operations by a quadratic factor with this technique.

One technique that we have developed for finding the least number of constraints required to fully constrain a problem is to reduce the problem to the least number of required multiplications between variables. Using this, we can easily calculate a soft lower bound for the number of constraints required to achieve to fully constrain an operation or system of operations.

Results

We constructed and tested a large number of reusable gadgets for a variety of operations from logic gate verification to full neural network execution and RSA signature checking. Below is a hierarchical taxonomy of our gadgets.



The following example output is a succinct version of what the verifier receives. Note that the signature and proof are truncated, and the input image is provided here as a reference despite being hidden completely from the verifier. This is example output for a fully connected feed-forward neural network with three layers of 784, 32, and 10 nodes respectively trained on MNIST.

MNIST Demo Stats	Setup Time (128 Threads)	Setup Ram	Prover Time	Proving Key Size
With RSA	5-10 min	80 GB	>1 min	30 GB
Without RSA	3-5 min	12 GB	<10 seconds	2 GB

```
Input Hash:
0x0D5394498EC5602F7D29DEC1A114CB39

Model Signature (256 Bytes):
A6 2E 94 84 6B 41 8F 69 89 34 E4 92 ... 9F 45

Output Weights:
0:(0.00000) 1:(0.03225) 2:(0.23016) 3:(0.00000)
4:(0.00011) 5:(0.00078) 6:(0.00000) 7:(0.99761)
8:(0.03148) 9:(0.00000)

ZK Proof:
11011100 11000101 00100101 01100111...00001010
```

Secret Input:

Conclusion and Future Work

We present and demonstrate an optimized architecture and implementation for the efficient execution and remote verification of feed-forward neural networks and their inputs. Our implementation is heavily optimized with the largest number of constraints coming from the hashing algorithm and digital signature verification scheme. This program currently uses fewer constraints than would be generated from the compilation from a general circuit. We suggest the following avenues for future research which either expand on this project or would aid with increasing the efficiency of zkSNARK construction.

1) Implement additional neural network features which effectively leverage the structure of R1CS constraints, such as convolutions.
2) Implement a neural network specific compiler and optimizer for the reduction from Computation to R1CS
3) Develop a method that is more efficient than QAPs or SSPs for the expansion of R1CS to a Linear PCP
4) Develop and Implement a post-quantum secure zkSNARK construction scheme and implement post-quantum cryptographic gadgets in R1CS