Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning

Chenkai Weng* Kang Yang† Xiang Xie‡ Jonathan Katz§ Xiao Wang*

Abstract

Recent progress in interactive zero-knowledge (ZK) proofs has improved the efficiency of proving large-scale computations significantly. Nevertheless, real-life applications (e.g., in the context of private inference using deep neural networks) often involve highly complex computations, and existing ZK protocols lack the expressiveness and scalability to prove results about such computations efficiently.

In this paper, we design, develop, and evaluate a ZK system (Mystique) that allows for efficient conversions between arithmetic and Boolean values, between publicly committed and privately authenticated values, and between fixed-point and floating-point numbers. Targeting large-scale neural-network inference, we also present an improved ZK protocol for matrix multiplication that yields a 7× improvement compared to the state-of-the-art. Finally, we incorporate Mystique in Rosetta, a TensorFlow-based privacy-preserving framework.

Mystique is able to prove correctness of an inference on a private image using a committed (private) ResNet-101 model in 28 minutes, and can do the same task when the model is public in 5 minutes, with only a 0.02% decrease in accuracy compared to a non-ZK execution when testing on the CIFAR-10 dataset. Our system is the first to support ZK proofs about neural-network models with over 100 layers with virtually no loss of accuracy.

1 Introduction

Zero-knowledge (ZK) proofs allow one party with a secret witness to prove some statement about that witness without revealing any additional information. In recent years we have seen massive progress in the efficiency and scalability of ZK proofs based on many different ideas [BCCT12, IKOS07, GKR08, JKO13, BCGI18]. With such improvements, we envision a huge potential in applying ZK proofs to *machine learning* (ML) applications, particularly neural-network inference. As examples:

- **Zero-knowledge proofs of evasion attacks.** A pre-trained model $\mathcal{M}$ might be publicly released to be used by the general public. Using a ZK protocol, a white-hat hacker who discovers a bug in the model (e.g., an evasion attack) could prove existence of that bug in zero knowledge, e.g., they could prove knowledge of two "close" inputs x_1 and x_2 for which $\mathcal{M}(x_1) \neq \mathcal{M}(x_2)$.

*Northwestern University. **Email:** `ckweng@u.northwestern.edu`, `wangxiao@cs.northwestern.edu`

†State Key Laboratory of Cryptology. **Email:** `yangk@sklc.org`

‡Shanghai Key Laboratory of Privacy-Preserving Computation and MatrixElements Technologies. **Email:** `xiexiang@matrixelements.com`

§University of Maryland. **Email:** `jkatz2@gmail.com`

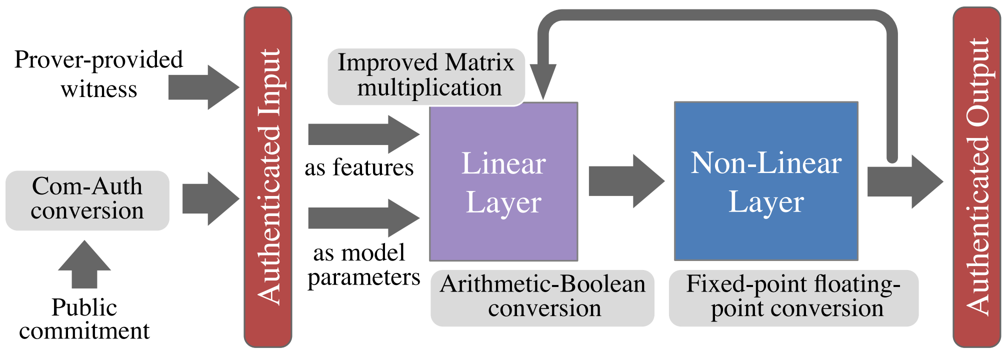

Figure 1: **Overview of our system for ZK neural-network inference.**

- **Zero-knowledge proofs of correct inference.** An ML model may require huge effort to train and thus may only be accessible as a paid service (e.g., GPT-3 that contains 175 billion parameters [BMR+20]). In this case, the model parameters are kept private, and users need to send their inputs to the owner of an ML model to be classified. Currently, such users have no guarantee that the model owner applies the correct model. Using a ZK protocol, the model owner could publicly commit to a model, and then proves in zero knowledge that the committed model was applied to the user's submitted input, yielding the claimed result.

- **Zero-knowledge proofs of private benchmarks.** An ML model may be evaluated on private testing data. Here, the owner of the testing data can publicly commit to its data; a model trainer can then send its model (developed using independent training data) to the data owner, who locally evaluates the accuracy of the model. The data owner can use a ZK protocol to prove that the submitted model was executed on the committed data.

Unfortunately, after examining existing ZK proof systems, we found that no existing solutions were sufficiently scalable or practical for any of the above applications once reasonably complicated neural-network models were involved. For example, zk-SNARKs [BCTV14b, BCC+16, BBB+18, WTs+18, BCR+19, BBHR19, Set20, KPPS20] provide excellent proof size and verification time, and support verifiable computation on committed data [FFG+16]. However, state-of-the-art zk-SNARKs require the memory of the prover to be proportional to the statement size; proving a statement with a billion constraints would require about 640 GB of memory. For ML applications, they can only handle simple models like decision trees [ZFZS20]. Recent ZK protocols based on *subfield vector oblivious linear evaluation* (sVOLE) [WYKW21, DIO20, BMRS20, YSWW21], privacy-free garbled circuits [JKO13, FNO15, HK20b], or the "MPC-in-the-head" paradigm [GMO16, CDG+17, KKW18, AHIV17, BN20] are efficient in terms of execution time and memory overhead, but do not work efficiently with publicly committed data and the overall communication is still fairly large. While in principle one could use zk-SNARKs with recursive composition [COS20, BDFG20], their concrete performance is still quite poor.

1.1 Our Contributions

We propose a system (Mystique[1]) based on recent sVOLE-based interactive ZK protocols that includes a set of building blocks for efficient ZK proofs of large-scale neural-network inference.

[1]Mystique is a shape-shifter; our system supports efficient conversions ("shape shifting") in zero knowledge.

Crucially, our system includes efficient techniques for three types of conversions:

1. **Arithmetic/Boolean values.** Inspired by a similar ideas in the setting of secure computation [EGK+20], we design optimized protocols to efficiently convert between arithmetic and Boolean values (to support mixed-mode circuits) in the context of sVOLE-based zero knowledge.

2. **Committed/authenticated values.** To allow publicly committed values to be used in ZK proofs, we design an efficient protocol that converts such values to values that are privately authenticated by the prover and verifier, and can thus be used directly in sVOLE-based ZK protocols.

3. **Fixed-point/floating-point values.** We designed circuits for IEEE-754-compliant floating-point operations, and designed efficient protocols to convert between fixed-point values (encoded in a field) and floating-point numbers.

In addition to the above, we also design an efficient ZK proof for matrix multiplication, such that the number of private multiplications required is *sublinear* in the matrix size. Compared to the previously best-known ZK proof for matrix multiplication [YSWW21], our ZK protocol improves the execution time by a factor of $7\times$.

We integrated the above in Rosetta [CHS+20], a privacy-preserving framework based on TensorFlow [AAB+15], and use the resulting system for ZK proofs regarding neural-network inference. As shown in Figure 1, linear layers of the neural network are accelerated by using our matrix-multiplication optimization, while the non-linear layers rely on our fixed-point/floating-point conversions. (We implemented ReLU, Sigmoid, Max Pooling, SoftMax, and Batch Normalization in the non-linear layers; other operations can be added easily.) All computations can be done using either arithmetic or Boolean values, depending on which is more efficient at any given step. Due to our improved cryptographic protocols and integrated implementation, we can implement ZK proofs on large neural networks (e.g., ResNet-50 and ResNet-101) with millions of model parameters; see Section 7.

2 Preliminaries

2.1 Notation

We use λ and ρ to denote the computational and statistical security parameters, respectively. For a finite set S, we use $x \leftarrow S$ to denote that x is chosen uniformly from S. For $a, b \in \mathbb{N}$, we denote by $[a, b]$ the set $\{a, \dots, b\}$, and by $[a, b)$ the set $\{a, \dots, b-1\}$. We use bold lower-case letters like $\boldsymbol{x}$ for column vectors, and denote by x_i the i-th component of $\boldsymbol{x}$ where x_1 is the first entry. We use $\boldsymbol{x}^\top$ to denote the transposition of $\boldsymbol{x}$. We use $\mathsf{negl}(\cdot)$ to denote a negligible function.

For an extension field $\mathbb{F}_{q^k}$ of a field $\mathbb{F}_q$, we fix some monic, irreducible polynomial $f(X)$ of degree k so that $\mathbb{F}_{q^k} \cong \mathbb{F}_q[X]/f(X)$. Thus, every element $a \in \mathbb{F}_{q^k}$ can be uniquely written as $a = \sum_{h \in [1,k]} a_h \cdot X^{h-1}$ with $a_h \in \mathbb{F}_q$ for all $h \in [1, k]$. When we write arithmetic expressions involving both elements of $\mathbb{F}_q$ and $\mathbb{F}_{q^k}$, operations are performed in $\mathbb{F}_{q^k}$ with elements of $\mathbb{F}_q$ viewed as elements of $\mathbb{F}_{q^k}$ in the natural way. In general, we work in an extension field such that $q^k \geq 2^\rho$.

2.2 Universal Composability

We say that protocol Π *UC-realizes* ideal functionality $\mathcal{F}$ if for any *probabilistic polynomial time* (PPT) adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$ with arbitrary auxiliary input, the output distribution of $\mathcal{Z}$ in the *real-world* execution where $\mathcal{Z}$ interacts with $\mathcal{A}$ and the parties running Π is computationally indistinguishable from the output distribution of $\mathcal{Z}$ in the *ideal-world* execution where $\mathcal{Z}$ interacts with $\mathcal{S}$ and $\mathcal{F}$. In the $\mathcal{G}$-hybrid model the parties execute a protocol given access to ideal functionality $\mathcal{G}$. We say that protocol Π UC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model with statistical error ε if the statistical difference between the output distributions of $\mathcal{Z}$ in the real-world execution and hybrid-world execution is bounded by ε.

2.3 Information-Theoretic MACs

We use *information-theoretic message authentication codes* (IT-MACs), which were originally proposed for maliciously secure two-party computation [BDOZ11, NNOB12]. We authenticate values in $\mathbb{F}_q$, but the authentication itself is done over an extension field $\mathbb{F}_{q^k}$. Specifically, let $\Delta \in \mathbb{F}_{q^k}$ be a uniform *global key* known only to the verifier $\mathcal{V}$. A value $x \in \mathbb{F}_q$ known by the prover $\mathcal{P}$ is authenticated by giving $\mathcal{V}$ a uniform *local key* $\mathsf{K} \in \mathbb{F}_{q^k}$ and giving $\mathcal{P}$ the corresponding tag

$$\mathsf{M} = \mathsf{K} + \Delta \cdot x \in \mathbb{F}_{q^k}.$$

We denote such an authenticated value by $[x]$, which means that $\mathcal{P}$ holds (x, M) and $\mathcal{V}$ holds K. When we want to make the field explicit, we write $[x]_q$. We extend the above notation to vectors or matrices of authenticated values as well. In this case, $[\boldsymbol{x}]$ means that $\mathcal{P}$ holds $\boldsymbol{x} \in \mathbb{F}_q^n$ and $\mathbf{M} \in (\mathbb{F}_{q^k})^n$, while $\mathcal{V}$ holds $\mathbf{K} \in (\mathbb{F}_{q^k})^n$ with $\mathbf{M} = \mathbf{K} + \Delta \cdot \boldsymbol{x}$.

Authenticated values are additively homomorphic. That is, given authenticated values $[x_1], \ldots, [x_\ell]$ and public coefficients $c_1, \ldots, c_\ell, c \in \mathbb{F}_q$, the parties can compute $[y] = \sum_{i=1}^{\ell} c_i \cdot [x_i] + c$ using only local computation.

Batch opening. An authenticated value $[x]$ can be opened by having $\mathcal{P}$ send (x, M) to $\mathcal{V}$, who verifies $M = K + \Delta \cdot x$. When opening ℓ values, it is possible to do better than ℓ parallel repetitions of this procedure; specifically, all ℓ values can be opened using only $\ell \log q + \lambda$ bits of communication. We use $\mathsf{BatchCheck}$ for the following batch-opening procedure:

- Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function modeled as a random oracle. Suppose that $[x_1], \ldots, [x_\ell]$ are ℓ authenticated values to be opened.
- $\mathcal{P}$ sends $x_1, \ldots, x_\ell \in \mathbb{F}_q$ to $\mathcal{V}$.
- Additionally:
  - If $q = 2$ and $k = \lambda$, the two parties compute $\chi := \mathsf{H}(x_1, \ldots, x_\ell) \in \mathbb{F}_{2^\lambda}$. $\mathcal{P}$ computes $\sigma := \sum_{i \in [1,\ell]} \mathsf{M}_i \cdot \chi^i \in \mathbb{F}_{2^\lambda}$ and sends it to $\mathcal{V}$, who checks whether $\sigma = \sum_{i \in [1,\ell]} (\mathsf{K}_i + \Delta \cdot x_i) \cdot \chi^i$. As in prior work [HSS17, WYKW21], the soundness error is bounded by $(q_{\mathsf{H}} + \ell + 1)/2^\lambda$, where q_{H} is the number of queries to H.
  - Otherwise, $\mathcal{P}$ computes $\sigma := \mathsf{H}(\mathsf{M}_1, \ldots, \mathsf{M}_\ell) \in \{0,1\}^\rho$ and sends it to $\mathcal{V}$, who can then check whether $\sigma := \mathsf{H}(\mathsf{K}_1 + \Delta \cdot x_1, \ldots, \mathsf{K}_\ell + \Delta \cdot x_\ell)$. The soundness error is at most $1/p^k + q_{\mathsf{H}}/2^\lambda$ [DNNR17].

When the opened values are all zero and so need not be sent, we use CheckZero to denote the batch-opening procedure.

Authenticated values from sVOLE. We can view authenticated values as *subfield vector oblivious linear evaluation* (sVOLE) correlations. Thus, random authenticated values can be efficiently generated using the recent LPN-based sVOLE protocols [SGRR19, BCG+19, YWL+20, WYKW21], which have communication complexity *sublinear* in the number of authenticated values.

2.4 Constant-Round Zero-Knowledge Proofs based on sVOLE

Several recent works [WYKW21, DIO20, BMRS20, YSWW21] explored the efficiency of sVOLE-based interactive ZK proofs. One can consider authenticated values as a form of commitments on values held by a prover $\mathcal{P}$, which can be verified by the verifier $\mathcal{V}$. Therefore, one can construct a ZK protocol following the "commit-and-prove" paradigm as follows:

1. In a preprocessing phase, $\mathcal{P}$ and $\mathcal{V}$ execute the sVOLE protocol to generate $n + N$ uniform authenticated values, where n is the witness size and N is the number of multiplication gates in the circuit. The parties also compute a uniform authenticated value $A_1^* \in \mathbb{F}_{q^k}$ by generating k uniform authenticated values in $\mathbb{F}_q$ and then using packing (see [YSWW21] for details). Thus, $\mathcal{P}$ obtains $A_0^*, A_1^* \in \mathbb{F}_{q^k}$ and $\mathcal{V}$ gets $B^* \in \mathbb{F}_{q^k}$ such that $B^* = A_0^* + \Delta \cdot A_1^*$.

2. Using the uniform authenticated values, $\mathcal{P}$ commits to all the wire values in an evaluation of the circuit on its witness. In particular, for each input wire of the circuit or output wire of a multiplication gate with associated value $x \in \mathbb{F}_q$, $\mathcal{P}$ sends $d = x - r \in \mathbb{F}_q$ to $\mathcal{V}$ and then both parties compute $[x] := [r] + d$, where $[r]$ is a random authenticated value computed in the previous step. Due to the additively homomorphic property of the underlying authenticated values, the parties can process addition gates for free and so this allows $\mathcal{P}$ and $\mathcal{V}$ to compute authenticated values on every wire in the circuit.

3. $\mathcal{P}$ proves that the committed values at all multiplication gates are correct by running a consistency-check procedure with $\mathcal{V}$. The known sVOLE-based ZK proofs [WYKW21, DIO20, BMRS20, YSWW21] use different consistency-check procedures. The state-of-the-art consistency check [DIO20, YSWW21] works as follows:

   (a) Consider the i-th multiplication gate with authenticated values $([x], [y], [z])$. If it is computed correctly (i.e., $z = x \cdot y$), then:

$$\begin{aligned}\overbrace{B_i = \mathsf{K}_x \cdot \mathsf{K}_y + \mathsf{K}_z \cdot \Delta}^{\text{known to } \mathcal{V}} &= (\mathsf{M}_x - x \cdot \Delta) \cdot (\mathsf{M}_y - y \cdot \Delta) + (\mathsf{M}_z - z \cdot \Delta) \cdot \Delta \\ &= \mathsf{M}_x \cdot \mathsf{M}_y + (\mathsf{M}_z - y \cdot \mathsf{M}_x - x \cdot \mathsf{M}_y) \cdot \Delta + (x \cdot y - z) \cdot \Delta^2 \\ &= \underbrace{\mathsf{M}_x \cdot \mathsf{M}_y}_{\substack{\text{known to } \mathcal{P} \\ \text{denoted as } A_{0,i}}} + \underbrace{(\mathsf{M}_z - y \cdot \mathsf{M}_x - x \cdot \mathsf{M}_y)}_{\substack{\text{known to } \mathcal{P} \\ \text{denoted as } A_{1,i}}} \cdot \underbrace{\Delta}_{\substack{\text{known to } \mathcal{V} \\ \text{global key}}}\end{aligned}$$

   If $z \neq x \cdot y$, then the above holds with probability at most $2/q^k$ over choice of Δ.

   (b) The parties can check all N of the above equations at once by taking a random linear combination. In particular, $\mathcal{V}$ samples and sends a uniform $\chi \in \mathbb{F}_{q^k}$ to $\mathcal{P}$. Then $\mathcal{P}$ sends

Functionality $\mathcal{F}_{\mathsf{authZK}}$

This functionality is parameterized by a prime $p > 2$ and an integer k such that $p^k \geq 2^\rho$, and can invoke a macro $\mathsf{Auth}()$ defined in Figure 4. Let $m = \lceil \log p \rceil$.

Initialize: On input (init) from a prover $\mathcal{P}$ and verifier $\mathcal{V}$, sample $\Delta \leftarrow \mathbb{F}_{2^\lambda}$ and $\Gamma \leftarrow \mathbb{F}_{p^k}$ if $\mathcal{V}$ is honest, and receive $\Delta \in \mathbb{F}_{2^\lambda}$ and $\Gamma \in \mathbb{F}_{p^k}$ from the adversary otherwise. Store (Δ, Γ) and send them to $\mathcal{V}$, and ignore all subsequent (init) commands.

Input: On input (input, w, q) from $\mathcal{P}$ and (input, q) from $\mathcal{V}$, where $w \in \mathbb{F}_q$ and $q \in \{2, p\}$, execute $\mathsf{Auth}(w, q)$ so that the parties obtain $[w]$.

Output: On input $(\mathsf{output}, [z], q)$ from the parties with $q \in \{2, p\}$, check if $[z]$ is valid and abort if the check fails; otherwise send z to $\mathcal{V}$.

Circuit-based commands

Random: On input (random, q) from the parties with $q \in \{2, p\}$, sample $w \leftarrow \mathbb{F}_q$ if $\mathcal{P}$ is honest; otherwise receive $w \in \mathbb{F}_q$ from the adversary. Execute $\mathsf{Auth}(w, q)$ so the parties obtain $[w]$.

Multiply: On input $(\mathsf{mult}, [x], [y], q)$ from both parties with $q \in \{2, p\}$, check that $[x], [y]$ are valid and abort if not. Compute $z := x \cdot y \in \mathbb{F}_q$ and run $\mathsf{Auth}(z, q)$ so the parties obtain $[z]$.

Figure 2: **Zero-knowledge functionality with authenticated values.** When we say the parties input $[x]$, we mean that $\mathcal{P}$ inputs (x, M_x) and $\mathcal{V}$ inputs K_x.

$U_0 := \sum_{i \in [1,N]} A_{0,i} \cdot \chi^i + A_0^*$ and $U_1 := \sum_{i \in [1,N]} A_{1,i} \cdot \chi^i + A_1^*$ to $\mathcal{V}$, who checks that $\sum_{i \in [1,N]} B_i \cdot \chi^i + B^* = U_0 + U_1 \cdot \Delta$. This can be made non-interactive using the Fiat-Shamir transform (i.e., computing χ by hashing the protocol transcript), and can be further optimized when q is large [WYKW21, YSWW21].

The ideal functionality for ZK proofs in this setting is summarized in Figure 2, where both arithmetic and Boolean circuits are considered. (Prior work [WYKW21, DIO20, BMRS20, YSWW21] efficiently realizes either arithmetic or Boolean circuits, but not mixed-mode computations.) For convenience, we also include in the ideal functionality the other conversions we support in our work.

3 Technical Overview

As mentioned in Section 1.1, we propose new protocols for arithmetic-Boolean and commitment-authentication conversions that are highly useful in real-world applications. We summarize in Figure 3 for the functionality definition of the two types of conversions. At a high level, our arithmetic-Boolean conversion allows authenticated values to be converted between arithmetic and Boolean circuits, while at the same time ensuring that the consistent values are converted. The commitment-authentication conversion allows us to convert from publicly committed values to privately authenticated values: the former provides a unified view of data across multiple verifiers, while the later can be efficiently processed by the sVOLE-based ZK protocols.

For ML applications, we also present the conversion between fixed-point and floating-point numbers, and an improved ZK proof for matrix multiplication. Below we provide an overview of our techniques and leave the detailed protocol description as well as proofs of security in later sections.

Functionality $\mathcal{F}_{\mathsf{authZK}}$, continued

Conversion between arithmetic and Boolean values

From arithmetic to Boolean: On input ($\mathsf{convertA2B}$, $[x]_p$) from both parties, check that $[x]_p$ is valid and abort if not. Express $x \in \mathbb{F}_p$ as $(x_0, \ldots, x_{m-1}) \in \{0,1\}^m$ such that $x = \sum_{i=0}^{m-1} x_i \cdot 2^i \bmod p$. Then, for $i \in [0, m)$, execute $\mathsf{Auth}(x_i, 2)$ so that the parties obtain $[x_i]_2$.

From Boolean to arithmetic: On input ($\mathsf{convertB2A}$, $[x_0]_2, \ldots, [x_{m-1}]_2$) from both parties, check that $[x_i]_2$ is valid for all $i \in [0, m)$ and abort if not. Compute $x := \sum_{i=0}^{m-1} x_i \cdot 2^i \bmod p$ and execute $\mathsf{Auth}(x, p)$ so that the parties obtain $[x]_p$.

Conversion from publicly committed values to privately authenticated values

Commit: On input (commit, x, q) from $\mathcal{P}$ with $q \in \{2, p\}$ and $x \in \mathbb{F}_q$, store (cid, x, q) with a fresh identifier cid, and then send cid to any potential verifiers.

From committed to authenticated values: On input $(\mathsf{convertC2A}, cid)$ from $\mathcal{P}$ and $\mathcal{V}$, check that (cid, w, q) is stored and abort if not. Execute $\mathsf{Auth}(w, q)$ so that $\mathcal{P}$ and $\mathcal{V}$ obtain $[w]$.

Figure 3: **Zero-knowledge functionality with authenticated values, continued.**

3.1 Arithmetic-Boolean Conversion

Enabling ZK proofs to support both arithmetic and Boolean circuits have been an important topic and studied in prior work. Particularly, in zk-SNARKs, it is often referred to as bit-decomposition [CFH+15, WSR+15, BCTV14a, PHGR13, BCTV14b]. Suppose that a prover has a witness x and the statement needs to compute on the bit representation of x. The prover can provide a bit decomposition of x, namely $\{x_i\}_{i \in [0,m)}$, with m as the bit-length of x. The prover can then prove in zero-knowledge that $x_i \cdot (x_i - 1) = 0$ for all $i \in [0, m)$ and $\sum_{i \in [0,m)} x_i \cdot 2^i = x$. Essentially, this is a way to simulate bit computation on an arithmetic circuit, which does not improve the underlying ZK proof.

Another solution [HK20a] is to combine garbled-circuit zero-knowledge proofs [JKO13, FNO15, HK20b] (GCZK) with arithmetic garbling [BMR16]. However, it only supports multiplication by public constants, and thus proving multiplication of two values over field $\mathbb{F}_p$ still needs to take communication of $\lambda \log p$ bits.

Our approach. Some recent works on sVOLE-based ZK protocols achieve high concrete efficiency [WYKW21, DIO20, BMRS20, YSWW21]. They support either arithmetic or Boolean circuits, and compute a circuit with authenticated wire values. The conversion between two types of circuits boils down to converting between authenticated arithmetic values and authenticated Boolean values. These cases are similar to some *secure multi-party computation* (MPC) protocols, which only support operations over either arithmetic or Boolean circuits, and use IT-MACs to authenticate secretly-sharing values.

In the MPC setting, converting authenticated shares between arithmetic and Boolean circuits can be accomplished by so-called *doubly authenticated bits* (daBits) [RW19, AOR+19, DEF+19]. The key idea of daBits is to prepare for secretly-shared random bits that are authenticated in both

Macro $\mathsf{Auth}(x, q)$

On input $x \in \mathbb{F}_q$ and $q \in \{2, p\}$, this subroutine interacts with two parties $\mathcal{P}$ and $\mathcal{V}$, and generates an authenticated value $[x]$ for the the parties. Let $k = \lambda$ and $\Phi = \Delta$ if $q = 2$. Let $k \in \mathbb{N}$ such that $q^k \geq 2^\rho$ and $\Phi = \Gamma$ if $q = p$.

1. If $\mathcal{V}$ is honest, sample $\mathsf{K} \leftarrow \mathbb{F}_{q^k}$. Otherwise, receive $\mathsf{K} \in \mathbb{F}_{q^k}$ from the adversary.
2. If $\mathcal{P}$ is honest, compute $\mathsf{M} := \mathsf{K} + x \cdot \Phi \in \mathbb{F}_{q^k}$. Otherwise, receive $\mathsf{M} \in \mathbb{F}_{q^k}$ from the adversary and recompute $\mathsf{K} := \mathsf{M} - x \cdot \Phi \in \mathbb{F}_{q^k}$.
3. Output $[x]$ to the parties, i.e., send (x, M) to $\mathcal{P}$ and K to $\mathcal{V}$.

Figure 4: **Macro used by functionalities $\mathcal{F}_{\mathsf{authZK}}$ and $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ to generate authenticated values.**

fields $\mathbb{F}_2$ and $\mathbb{F}_p$ with a large prime p (meaning that $p \geq 2^\rho$), so that one set of MAC tags support Boolean operations (i.e., AND and XOR), while the other set of MAC tags are arithmetic-operation (i.e., MULT and ADD) homomorphic. To perform a conversion, we need $m = \lceil \log p \rceil$ such daBits, which are used to convert the shares of $x_0, \ldots, x_{m-1} \in \mathbb{F}_2$ to that of $x = \sum_{h=0}^{m-1} x_h \cdot 2^h \in \mathbb{F}_p$, where the related MAC tags are also converted accordingly. In the ZK setting, we can use a similar method. Unfortunately, although we can authenticate a field element over $\mathbb{F}_p$ efficiently in communication of $O(\log p)$ bits, authenticating a bit with the MAC tag in $\mathbb{F}_p$ still takes $O(\log p)$ bits (instead of one bit) for communication. As a result, the conversion requires a total communication of $O(\log^2 p)$ bits for generating m daBits, not even counting the cost to perform conversion using daBits.

To overcome this, we instead follow the more recent *extended daBits* (edaBits) [EGK+20], which can be viewed as a more compact representation of daBits. An edaBit consists of a set of m random secretly-shared bits $([r_0]_2, \ldots, [r_{m-1}]_2)$ with the MAC tags in $\mathbb{F}_{2^\lambda}$ and a secretly-shared field element $[r]_p$ with the MAC tag in $\mathbb{F}_p$, such that $(r_0, \ldots, r_{m-1}) \in \mathbb{F}_2^m$ is equal to the bit-decomposition of $r \in \mathbb{F}_p$. Such an edaBit is sufficient to perform conversion between arithmetic and Boolean circuits. Now, generating an edaBit requires only $O(\log p)$ bits of communication. Inspired by the edaBits approach for MPC, we first construct *ZK-friendly edaBits* (zk-edaBits) to keep compatible with the recent sVOLE-based ZK proofs [WYKW21, DIO20, BMRS20, YSWW21], and then construct two conversion protocols using our random zk-edaBits, where one can convert authenticated values from arithmetic to Boolean circuits and the other converts in another direction.

Constructing zk-edaBits. Similar to edaBit in the MPC setting, a zk-edaBit consists of a random authenticated value $[r]_p$ and m authenticated bits $[r_0]_2, \ldots, [r_{m-1}]_2$, such that $r = \sum_{h=0}^{m-1} r_h \cdot 2^h \bmod p$. In the ZK setting, the prover is allowed to know $(r_0, \ldots, r_{m-1}, r)$ as it knows all wire values in the circuit, and thus the secret sharing of these values is *unnecessary*. Here, we do not assume that p is a large prime, and instead allow any prime $p > 2$, as we consider authentication is done in an extension field $\mathbb{F}_{p^k}$ with $p^k \geq 2^\rho$. In the MPC setting, r is also considered as a ring element over $\mathbb{Z}_{2^k}$ (e.g., $k = 64$) [DEF+19, EGK+20], which may allow to obtain better efficiency for ML applications. In the ZK setting, the state-of-the-art consistency check [DIO20, YSWW21] for sVOLE-based ZK proofs is *not* friendly for such rings, which makes the ZK proofs over $\mathbb{Z}_{2^k}$ be less efficient.

Using two sVOLE-based ZK proofs where one for $\mathbb{F}_2$ and the other for $\mathbb{F}_p$, we can construct authenticated values $([r_0]_2, \dots, [r_{m-1}]_2)$ and $[r]_p$ with communication of $O(\log p)$ bits in total. Nevertheless, if the prover is malicious, then there may be an inconsistency (i.e, $r \neq \sum_{h=0}^{m-1} r_h \cdot 2^h \mod p$). Now, our task is to check the consistency of N faulty zk-edaBits computed as above, where N is the number of zk-edaBits needed. This could be done using the "cut-and-bucketing" technique similar to prior work [FLNW17, ABF+17, WYKW21]. Specifically, we let a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ generate extra $N(B-1)+c$ faulty zk-edaBits, where B, c are two parameters. Then two parties use a random permutation to shuffle $NB+c$ faulty zk-edaBits. The last c faulty zk-edaBits are opened, and their correctness is checked by $\mathcal{V}$. Next, the remaining NB faulty zk-edaBits are partitioned into N buckets with each of size B. Finally, for each bucket, the parties perform the "combine-and-open" check $B-1$ times of between the first zk-edaBit in the bucket and the other $B-1$ zk-edaBits. See Section 4.1 for more details and optimization.

Arithmetic-Boolean conversions using zk-edaBits. In Figure 3, we define the functionality for converting authenticated wire values between arithmetic and Boolean circuits. We will use random zk-edaBits to realize the conversion of authenticated values between arithmetic and Boolean wires.

Given a random zk-edaBit $([r_0]_2, \dots, [r_{m-1}]_2, [r]_p)$ and an authenticated value $[x]_p$ to be converted, $\mathcal{P}$ and $\mathcal{V}$ can open a *masked* value $[z]_p = [x]_p - [r]_p$, and call functionality $\mathcal{F}_{\mathsf{authZK}}$ with only circuit-based commands to compute a modular-addition circuit on a public input $(z_0, \dots, z_{m-1})$ and a secret input $([r_0]_2, \dots, [r_{m-1}]_2)$ so that they obtain $([x_0]_2, \dots, [x_{m-1}]_2)$, where $(z_0, \dots, z_{m-1})$ is the bit-decomposition of z. From $z = x - r \mod p$ and $\sum_{h=0}^{m-1} z_h \cdot 2^h = z \mod p$, one can easily verify that $(x_0, \dots, x_{m-1})$ is the bit-decomposition of x. The other direction can be constructed in a similar way.

In general, we need to convert multiple authenticated values between arithmetic and Boolean wires, and thus can open multiple authenticated values *in a batch* to reduce the communication cost. In Section 4.2, we will provide the full description of our protocols and formal proofs of security.

3.2 Conversion from Publicly Committed Values to Privately Authenticated Values

Our second task is to convert from non-interactive commitments (publicly available to all parties) to authenticated values (privately available between two parties). The former is suitable for committing values in a public repository, while the latter is better for efficient sVOLE-based ZK proofs and also compatible with the above conversion between arithmetic and Boolean circuits.

Our conversion uses the commitment scheme in a non-black-box way: a prover $\mathcal{P}$ first authenticates to a verifier $\mathcal{V}$ the committed values as well as the decommitment, and then proves in zero-knowledge that the authenticated values satisfy the opening of the public commitment. This establishes a connection between a public commitment and privately authenticated values. The efficiency of the protocol crucially relies on the size of the circuit to represent the opening of a commitment scheme. If we use cryptographic hash functions like SHA-256, it would require more than 22,000 AND gates to commit a 512-bit message, averaging to 42 gates per bit. One can also use LowMC [ARS+15] as a block cipher with the 256-bit key and 256-bit block length, where LowMC allows much less AND gates than standard block ciphers such as AES. When being modeled as an ideal cipher, LowMC can be converted to a suitable hash function using the Merkle–Damgård structure. However, the computation complexity in this case could be very high as we need to calculate a lot of matrix multiplications with random bits.

To minimize the circuit size, we use a "hybrid commitment" scheme: to commit a set of messages $\{\boldsymbol{x}_i\}_{i\in[1,\ell]}$ with $\boldsymbol{x}_i \in \{0,1\}^m$, we first pick a random key $sk \leftarrow \{0,1\}^\lambda$, commit to this key by using a slow commitment scheme (e.g., $\mathsf{H}(sk, r)$ for a random oracle H and a randomness r), and then commit the messages as $\boldsymbol{c}_i := \mathsf{PRF}(sk, i) \oplus \boldsymbol{x}_i$ for $i \in [1, \ell]$, where PRF is a standard pseudorandom function. The security of this hybrid commitment scheme can be reduced to the security of the slow commitment scheme as well as the pseudo-randomness of PRF. What's more, if the slow commitment scheme is extractable, then the overall commitment scheme is also extractable. Note that we cannot equivocate $\boldsymbol{x}_i$ to any vector if we use the natural "open" algorithm with sending sk and $\boldsymbol{x}_i$, as the function PRF is fixed. However, we can make this commitment scheme equivocal in an interactive way: we prove knowledge of sk and $\boldsymbol{x}_i$ such that all relationships hold, and convert the committed value to an authenticated value in zero knowledge. Besides, we easily extend the above hybrid commitment scheme to support committed values over any field $\mathbb{F}_q$ by extending the output range of PRF to $\mathbb{F}_q^m$.

To reduce the number of AND gates for PRF, we choose to use LowMC to instantiate PRF. To obtain faster computation, we adopt a smaller block size (i.e., 64 bits). As a result, our protocol can convert 18,000 64-bit committed values (144 KB in total) to authenticated values in a second.

3.3 Optimizations for ML Applications

To make ZK proofs of ML applications practical, we also propose several optimizations specifically to reduce the overhead of some key ML components and to integrate with TensorFlow [AAB+15]. Detailed descriptions can be found in Section 6.

Matrix multiplication. Directly proving matrix multiplication in zero knowledge would require $O(n^3)$ number of multiplications, which could be improved to $O(n^{2.8})$ (or even lower) based on a better algorithm [Str69]. Although the prover time has to be linear to this complexity, we could reduce the circuit size for ZK proofs significantly. Suppose in a certain stage of the ZK protocol, a prover $\mathcal{P}$ wants to prove the relation that $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$ with $\mathbf{A} \in \mathbb{F}_q^{n\times m}, \mathbf{B} \in \mathbb{F}_q^{m\times \ell}, \mathbf{C} \in \mathbb{F}_q^{n\times \ell}$, where $\mathbf{A}, \mathbf{B}, \mathbf{C}$ have been committed using authenticated values resulting in $[\mathbf{A}], [\mathbf{B}], [\mathbf{C}]$. By generalizing the Freivalds algorithm [Fre77], we use a random-linear-combination approach to prove that $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$ holds. Specifically, we can let a verifier $\mathcal{V}$ sample two uniformly random vectors $\boldsymbol{u} \in (\mathbb{F}_{q^k})^n, \boldsymbol{v} \in (\mathbb{F}_{q^k})^\ell$. Instead of directly proving $[\mathbf{A}] \cdot [\mathbf{B}] = [\mathbf{C}]$, we can prove:

$$\boldsymbol{u}^\top \cdot [\mathbf{A}] \cdot [\mathbf{B}] \cdot \boldsymbol{v} = \boldsymbol{u}^\top \cdot [\mathbf{C}] \cdot \boldsymbol{v}.$$

Now, the parties can locally compute vectors of authenticated values $[\boldsymbol{x}]^\top = [\boldsymbol{u}^\top \cdot \mathbf{A}] \in (\mathbb{F}_{q^k})^m$, $[\boldsymbol{y}] = [\mathbf{B} \cdot \boldsymbol{v}] \in (\mathbb{F}_{q^k})^m$ and $[z] = [\boldsymbol{u}^\top \cdot \mathbf{C} \cdot \boldsymbol{v}] \in \mathbb{F}_{q^k}$. Thus, they only need to prove in zero-knowledge that $[\boldsymbol{x}]^\top \cdot [\boldsymbol{y}] = [z]$, which takes only the communication of $O(k \log q)$ bits using the latest ZK proof [YSWW21], where k is an integer satisfying $q^k > 2^\rho$. Note that this ZK protocol [YSWW21] allows to prove the statements over $\mathbb{F}_{q^k}$.

Fixed-point and floating-point conversions. The above matrix-multiplication protocol only works over a field. However, in the neural-network inference, all operations are for real numbers. To address this discrepancy, we use both fixed-point and floating-point encodings of real numbers at different stages of our protocol. Firstly, we encode a signed, fixed-point number x with

$-\frac{p-1}{2^{s+1}} \leq x \leq \frac{p-1}{2^{s+1}}$ into a field element in $[-\frac{p-1}{2}, \frac{p-1}{2}]$ by computing $\lfloor 2^s \cdot x \rceil$, where $p > 2$ is a prime and $s \in \mathbb{N}$ is a precision parameter. Then we easily encode field elements in $[-\frac{p-1}{2}, \frac{p-1}{2}]$ into field elements in $[0, p-1]$. In this way, the addition and multiplication of fixed-point numbers are the same as addition and multiplication over field $\mathbb{F}_p$, as long as there is no overflow. One caveat is that overflow can happen quickly if the multiplication depth is high. Fortunately, for matrix multiplication, the multiplication depth is 1. After linear layers, we usually need to perform many non-linear operations like Batch Normalization (which needs square root and inverse), SoftMax (which additionally needs exponentiation), ReLU (which additionally needs comparison), etc. To support these operations efficiently and accurately, we convert between fixed-point numbers and IEEE-754 compliant floating-point numbers using functionality $\mathcal{F}_{\mathsf{authZK}}$ with only circuit-based commands, such that non-linear operations can be performed in zero-knowledge.

Integration with TensorFlow. To easily implement complicated neural networks, we integrated our backend protocol with TensorFlow [AAB+15], so that existing TensorFlow neural network implementations can be directly executed in our protocol, while keeping the TensorFlow interfaces unchanged. In particular, we implemented a set of common operators that are needed and hook them with TensorFlow using a dynamic pass. Due to our use of floating-point numbers in the non-linear layers, adding more operators is fairly straightforward. See Section 6.3 for more details.

4 Arithmetic-Boolean Conversion for Zero-Knowledge Proofs

In this section, we provide full details on how to construct *ZK-friendly extended doubly authenticated bits* (zk-edaBits) efficiently, and then show how to use them to securely realize conversions between arithmetic and Boolean circuits.

4.1 Extended Doubly Authenticated Bits for Zero-Knowledge Proofs

As described in Section 3.1, zk-edaBit is a key tool in this work to efficiently perform conversions between arithmetic and Boolean circuits. A zk-edaBit consists of a set of m authenticated bits $([r_0]_2, \ldots, [r_{m-1}]_2)$ along with a *random* authenticated value $[r]_p$ such that $r = \sum_{h=0}^{m-1} r_h \cdot 2^h \in \mathbb{F}_p$. We provide the ideal functionality for zk-edaBits in Figure 5.

A prover $\mathcal{P}$ and a verifier $\mathcal{V}$ can generate faulty zk-edaBits by calling functionality $\mathcal{F}_{\mathsf{authZK}}$, and then use a "cut-and-bucketing" technique to check the consistency of resulting zk-edaBits. Recall that the overview of our technique has been described in Section 3.1. Thus, we directly provide the details of our zk-edaBits protocol in Figure 6. In this protocol, the prover and verifier use $\mathcal{F}_{\mathsf{authZK}}$ with only circuit-based commands to compute a Boolean circuit AdderModp, which efficiently realizes the module-addition computation that adds two m-bit integers and then modules a prime p.

Theorem 1. *Protocol $\Pi_{\mathsf{zk\text{-}edaBits}}$ shown in Figure 6 UC-realizes functionality $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ in the presence of a static, malicious adversary with statistical error at most* $\binom{N(B-1)+c}{B-1}^{-1} + \frac{1}{p^k}$ *in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model.*

The proof of this theorem can be found in Appendix A. Given the number N of zk-edaBits, we can choose suitable parameters B and c such that $\binom{N(B-1)+c}{B-1}^{-1} \leq 2^{-\rho}$. For example, when

Functionality $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$

This functionality is parameterized by a prime $p > 2$ and an integer $k \geq 1$ with $p^k \geq 2^\rho$. Let $m = \lceil \log p \rceil$.

Initialize: On input (init) from $\mathcal{P}$ and $\mathcal{V}$, sample $\Delta \leftarrow \mathbb{F}_{2^\lambda}$ and $\Gamma \leftarrow \mathbb{F}_{p^k}$ if $\mathcal{V}$ is honest, and receive $\Delta \in \mathbb{F}_{2^\lambda}$ and $\Gamma \in \mathbb{F}_{p^k}$ from the adversary otherwise. Store two global keys (Δ, Γ) and send them to $\mathcal{V}$, and ignore all subsequent (init) commands.

Generate ZK-friendly edaBits: On input (random) from two parties $\mathcal{P}$ and $\mathcal{V}$, generate a random zk-edaBit $([r_0]_2, \ldots, [r_{m-1}]_2, [r]_p)$ with $r_i \in \mathbb{F}_2$ for $i \in [0, m)$ and $r = \sum_{i=0}^{m-1} r_i \cdot 2^i \in \mathbb{F}_p$ as follows:

1. If $\mathcal{P}$ is honest, sample $r \leftarrow \mathbb{F}_p$. Otherwise, receive $r \in \mathbb{F}_p$ from the adversary. Decompose r to $(r_0, \ldots, r_{m-1})$ such that $r = \sum_{i=0}^{m-1} r_i \cdot 2^i \mod p$.
2. Execute $[r_i]_2 \leftarrow \mathsf{Auth}(r_i, 2)$ for $i \in [0, m)$ and $[r]_p \leftarrow \mathsf{Auth}(r, p)$, where the macro $\mathsf{Auth}(\cdot)$ is described in Figure 4. Thus, the two parties obtain $([r_0]_2, \ldots, [r_{m-1}]_2, [r]_p)$.

Figure 5: **Functionality for ZK-friendly extended doubly authenticated bits.**

$N = 10^6$, we can choose $B = 3$ and $c = 2$, and achieve at least 40-bit statistical security.

4.2 Arithmetic-Boolean Conversion Protocols

Using functionality $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ efficiently realized in the previous sub-section, we propose two efficient protocols to convert authenticated wire values from an arithmetic circuit to a Boolean circuit and to convert in another direction. In the two protocols, the prover and verifier would also use functionality $\mathcal{F}_{\mathsf{authZK}}$ with only circuit-based commands to compute a Boolean circuit AdderModp. In both of two protocols, we assume that $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ shares the same initialization procedure with $\mathcal{F}_{\mathsf{authZK}}$, and thus the same global keys are used in the two functionalities. This is the case, when we use the protocol $\Pi_{\mathsf{zk\text{-}edaBits}}$ shown in Figure 6 to UC-realize $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model.

We provide the full details about the conversion from arithmetic to Boolean circuits in Figure 7. In Figure 8, we describe in details how to perform an efficient conversion from Boolean to arithmetic circuits.

Below, we prove the security of the two protocols in the following theorems.

Theorem 2. *Protocol $\Pi^{\mathsf{A2B}}_{\mathsf{Convert}}$ UC-realizes the* convertA2B *command of functionality $\mathcal{F}_{\mathsf{authZK}}$ in the presence of a static, malicious adversary with statistical error $1/p^k$ in the $(\mathcal{F}_{\mathsf{zk\text{-}edaBits}}, \mathcal{F}_{\mathsf{authZK}})$-hybrid model.*

Theorem 3. *Protocol $\Pi^{\mathsf{B2A}}_{\mathsf{Convert}}$ UC-realizes the* convertB2A *command of functionality $\mathcal{F}_{\mathsf{authZK}}$ in the presence of a static, malicious adversary in the $(\mathcal{F}_{\mathsf{zk\text{-}edaBits}}, \mathcal{F}_{\mathsf{authZK}})$-hybrid model.*

The proofs of these theorems can be found in Appendix B.

Optimization using circuit-based zk-edaBits. In the conversion protocols described as above, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ generate random zk-edaBits using functionality $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ in the pre-

Protocol $\Pi_{\textsf{zk-edaBits}}$

Parameters: Let $p > 2$ be a prime, $m = \lceil \log p \rceil$ and $k \in \mathbb{N}$ with $p^k \geq 2^\rho$. Two parties want to generate N zk-edaBits. Let B, c be some parameters to be specified later and $\ell = NB + c$.

Initialize: $\mathcal{P}$ and $\mathcal{V}$ send (init) to $\mathcal{F}_{\textsf{authZK}}$, which returns two uniform global keys to $\mathcal{V}$.

Generating random zk-edaBits:

1. The parties generate random authenticated values $[r^i]_p$ for $i \in [1, \ell]$. Then, for $i \in [1, \ell]$, $\mathcal{P}$ decomposes r^i to $(r^i_0, \ldots, r^i_{m-1})$ such that $r^i = \sum_{h=0}^{m-1} r^i_h \cdot 2^h \mod p$.

2. For $i \in [1, \ell]$, $\mathcal{P}$ inputs $(r^i_0, \ldots, r^i_{m-1}) \in \mathbb{F}_2^m$ to $\mathcal{F}_{\textsf{authZK}}$, which returns $([r^i_0]_2, \ldots, [r^i_{m-1}]_2)$ to the parties.

3. Place the first N zk-edaBits into N buckets in order, where each bucket has exactly one zk-edaBit. Then, $\mathcal{V}$ samples a random permutation $\pi : [N+1, \ell] \to [N+1, \ell]$ and sends it to $\mathcal{P}$. Use π to permute the remaining $\ell - N$ zk-edaBits.

4. The parties check that the last c zk-edaBits are correctly computed and abort if not. Divide the remaining $N(B-1)$ (unopened) zk-edaBits into N buckets accordingly, such that each bucket has B zk-edaBits.

5. For each bucket, both parties choose the first zk-edaBit $([r_0]_2, \ldots, [r_{m-1}]_2, [r]_p)$ (that is placed into the bucket in the step 3), and for every other zk-edaBit $([s_0]_2, \ldots, [s_{m-1}]_2, [s]_p)$ in the same bucket, execute the following check:

   (a) Compute $[t]_p := [r]_p + [s]_p$, and then execute $([t_0]_2, \ldots, [t_{m-1}]_2) := \textsf{AdderModp}([r_0]_2, \ldots, [r_{m-1}]_2, [s_0]_2, \ldots, [s_{m-1}]_2)$, where AdderModp is the modular-addition circuit, and $\sum_{h=0}^{m-1} t_h \cdot 2^h = \sum_{h=0}^{m-1} r_h \cdot 2^h + \sum_{h=0}^{m-1} s_h \cdot 2^h \mod p$.

   (b) Execute the BatchCheck procedure on $([t_0]_2, \ldots, [t_{m-1}]_2)$ to obtain $(t_0, \ldots, t_{m-1})$, and then compute $t' := \sum_{h=0}^{m-1} t_h \cdot 2^h \mod p$.

   (c) Execute the CheckZero procedure on $[t]_p - t'$ to verify that $t = t'$.

6. If any check fails, $\mathcal{V}$ aborts. Otherwise, the parties output the first zk-edaBit from each of the N buckets.

Figure 6: **Protocol for generating ZK-friendly edaBits in the $\mathcal{F}_{\textsf{authZK}}$-hybrid model.**

processing phase, and then convert authenticated values between arithmetic and Boolean circuits using these random zk-edaBits in the online phase.

We can use an alternative approach to convert authenticated values between arithmetic and Boolean circuits, and obtain better whole efficiency but larger online cost. Specifically, for authenticated bits $[x_0]_2, \ldots, [x_{m-1}]_2$ on m output wires of a Boolean circuit, $\mathcal{P}$ can compute $x := \sum_{h=0}^{m-1} x_h \cdot 2^h \mod p$ locally. Then, $\mathcal{P}$ sends $(\textsf{input}, x, p)$ to $\mathcal{F}_{\textsf{authZK}}$ and $\mathcal{V}$ sends $(\textsf{input}, p)$ to $\mathcal{F}_{\textsf{authZK}}$, which returns $[x]_p$ to the parties. Similarly, the parties can also convert an authenticated value $[x]_p$ on an output wire of an arithmetic circuit into m authenticated bits $[x_0]_2, \ldots, [x_{m-1}]_2$, by calling the (input) command of $\mathcal{F}_{\textsf{authZK}}$. In this way, two parties can create N circuit-based zk-edaBits for

Protocol $\Pi^{\mathsf{A2B}}_{\mathsf{Convert}}$

Let $p > 2$ be a prime and $m = \lceil \log p \rceil$.

Initialize: $\mathcal{P}$ and $\mathcal{V}$ send (init) to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$, which returns two uniform global keys to $\mathcal{V}$.

Input: The parties have an authenticated value $[x]_p$.

Convert: $\mathcal{P}$ and $\mathcal{V}$ convert an authenticated value over field $\mathbb{F}_p$ into m authenticated bits as follows:

1. Send (random) to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$, which returns $([r_0]_2, \ldots, [r_{m-1}]_2, [r]_p)$ to the parties.
2. Compute $[z]_p := [x]_p - [r]_p$, and then execute the $\mathsf{BatchCheck}$ procedure on $[z]_p$ to obtain z.
3. Decompose $z \in \mathbb{F}_p$ as $(z_0, \ldots, z_{m-1}) \in \mathbb{F}_2^m$ such that $z = \sum_{h=0}^{m-1} z_h \cdot 2^h \mod p$, and then compute $([x_0]_2, \ldots, [x_{m-1}]_2) := \mathsf{AdderModp}(z_0, \ldots, z_{m-1}, [r_0]_2, \ldots, [r_{m-1}]_2)$ by calling $\mathcal{F}_{\mathsf{authZK}}$ where $z_0, \ldots, z_{m-1}$ are public constants.
4. Output $([x_0]_2, \ldots, [x_{m-1}]_2)$.

Figure 7: **Protocol for conversion from arithmetic to Boolean in the $(\mathcal{F}_{\mathsf{zk\text{-}edaBits}}, \mathcal{F}_{\mathsf{authZK}})$-hybrid model.**

Protocol $\Pi^{\mathsf{B2A}}_{\mathsf{Convert}}$

Let $p > 2$ be a prime and $m = \lceil \log p \rceil$.

Initialize: $\mathcal{P}$ and $\mathcal{V}$ send (init) to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$, which returns two uniform global keys to $\mathcal{V}$.

Input: Two parties $\mathcal{P}$ and $\mathcal{V}$ hold m authenticated bits $[x_0]_2, \ldots, [x_{m-1}]_2$.

Convert: $\mathcal{P}$ and $\mathcal{V}$ convert m authenticated bits into one authenticated value over field $\mathbb{F}_p$ as follows:

1. Send (random) to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$, which returns $([r_0]_2, \ldots, [r_{m-1}]_2, [r]_p)$ to the parties.
2. Compute $([z_0]_2, \ldots, [z_{m-1}]_2) := \mathsf{AdderModp}([x_0]_2, \ldots, [x_{m-1}]_2, [r_0]_2, \ldots, [r_{m-1}]_2)$ by calling functionality $\mathcal{F}_{\mathsf{authZK}}$, such that $\sum_{h=0}^{m-1} z_h \cdot 2^h = \sum_{h=0}^{m-1} x_h \cdot 2^h + \sum_{h=0}^{m-1} r_h \cdot 2^h \mod p$.
3. Execute the $\mathsf{BatchCheck}$ procedure on $([z_0]_2, \ldots, [z_{m-1}]_2)$ to obtain $(z_0, \ldots, z_{m-1})$, and then compute $z := \sum_{h=0}^{m-1} z_h \cdot 2^h \mod p$.
4. Compute and output $[x]_p := z - [r]_p$.

Figure 8: **Protocol for conversion from Boolean to arithmetic in the $(\mathcal{F}_{\mathsf{zk\text{-}edaBits}}, \mathcal{F}_{\mathsf{authZK}})$-hybrid model.**

some integer N. However, in the circuit-based zk-edaBits, a malicious prover may cause the field elements over $\mathbb{F}_p$ are inconsistent with corresponding bits. Verifier $\mathcal{V}$ can check the consistency of these circuit-based zk-edaBits using the cut-and-bucketing technique. Specifically, in the online phase, two parties can execute the checking procedure shown in Figure 6 to check the consistency of these circuit-based $\mathsf{zk\text{-}edaBits}$ by sacrificing $(B-1)N+c$ random $\mathsf{zk\text{-}edaBits}$ generated in the preprocessing phase. Using this optimization, for computing N circuit-based $\mathsf{zk\text{-}edaBits}$, we can save N random $\mathsf{zk\text{-}edaBits}$ and N evaluations of circuit $\mathsf{AdderModp}$ in terms of the whole efficiency, but increase the online cost by a factor of $B-1$.

5 Converting Publicly Committed Values to Privately Authenticated Values

The second type of conversions that we would like to study is the conversion from publicly committed data to privately authenticated data. Here, publicly committed data referred to those committed with a short digest, which can be published on something that can be modeled as a bulletin board (e.g., well-established websites or some blockchain). Privately authenticated data refers to the values only known by a prover that are authenticated by a designated verifier based on IT-MACs, and thus can be efficiently used to prove any mixed arithmetic-Boolean circuit using the recent ZK protocols [WYKW21, DIO20, BMRS20, YSWW21] and our arithmetic-Boolean conversion protocols. The conversion from publicly committed data to privately authenticated data will allow us to efficiently prove statements on consistent committed data to multiple different verifiers for multiple times.

Our commitment-authentication conversion protocol. We present our efficient conversion protocol in Figure 9. This protocol consists of two phases: 1) generating a non-interactive commitment and 2) converting publicly committed values to privately authenticated values in an interactive manner. To commit a large volume of data or different types of data, we divide them into pieces, where the i-th piece is denoted by $\boldsymbol{x}_i \in \mathbb{F}_q^m$ with a prime $q \geq 2$ and a parameter m. Then, we let the prover sample a random key sk and a uniform randomness r both in $\{0,1\}^\lambda$. Our commitment consists of $\mathsf{com}_0 = \mathsf{H}(sk, r)$ and $\boldsymbol{c}_i = \mathsf{PRF}(sk, i) + \boldsymbol{x}_i \in \mathbb{F}_q^m$ for all $i \in [1, \ell]$ with some $\ell \in \mathbb{N}$, where H is a random oracle and PRF is a pseudorandom function. To perform conversion, the prover $\mathcal{P}$ proves knowledge of sk and $\boldsymbol{x}_i$, such that com_0 and $\boldsymbol{c}_i$ are computed with the key and data piece. Since $\boldsymbol{c}_i$ can be put in the public domain, one can further reduce the size of the overall commitment by computing a Merkle tree on top of all $\boldsymbol{c}_i$'s. In this way, the commitment only has a size of 4λ bits, including com_0 and the root of the Merkle tree (i.e., com_1).

Since key $sk \in \{0,1\}^\lambda$ has a high entropy, we can actually remove the randomness r. That is, the prover can just set $\mathsf{H}(sk)$ as com_0 in the commitment phase and prove $\mathsf{com}_0 = \mathsf{H}([\boldsymbol{sk}]_2)$ in the conversion phase. This will slightly improve the efficiency of this protocol.

Theorem 4. *Let* H *be a random oracle and* PRF *be a pseudorandom function. Then protocol* $\Pi_{\mathsf{NICom}\to[\cdot]}$ *shown in Figure 9 UC-realizes the* $\mathsf{convertC2A}$ *command of functionality* $\mathcal{F}_{\mathsf{authZK}}$ *in the presence of a static, malicious adversary in the* $\mathcal{F}_{\mathsf{authZK}}$*-hybrid model.*

Below, we discuss the intuition of the above theorem and leave the full formal proof to Appendix C. We commit to sk using a standard UC commitment in the random-oracle model, and so sk is computationally hiding, meaning that $\mathsf{PRF}(sk, i)$ for all $i \in [1, \ell]$ are indistinguishable from uniformly random values in $\mathbb{F}_q^m$. In the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model, the ZK proof does not reveal any information of committed values. Overall, the committed data is hidden. In the proof of security, the simulator can extract the key sk from com_0 in the random-oracle model. Once sk was extracted,

Protocol $\Pi_{\mathsf{NICom}\to[\cdot]}$

Let $q \geq 2$ be a prime. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ be a cryptographic hash function modeled as a random oracle, and $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{F}_q^m$ be a pseudorandom function.

Compute a public commitment: A prover $\mathcal{P}$ computes and publishes a *non-interactive* commitment on values:

1. Sample $sk, r \leftarrow \{0,1\}^\lambda$; compute $\mathsf{com}_0 := \mathsf{H}(sk, r)$.
2. On input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell \in \mathbb{F}_q^m$ with $\ell, m \in \mathbb{N}$, compute $\boldsymbol{c}_i := \mathsf{PRF}(sk, i) + \boldsymbol{x}_i \in \mathbb{F}_q^m$ for $i \in [1, \ell]$.
3. Compute $d_i := \mathsf{H}(\boldsymbol{c}_i)$ for all $i \in [1, \ell]$; build a Merkle tree on these values using H with com_1 as the root.
4. Publish the commitment $(\mathsf{com}_0, \mathsf{com}_1)$.

Initialize: Prover $\mathcal{P}$ and a verifier $\mathcal{V}$ send (init) to $\mathcal{F}_{\mathsf{authZK}}$, which returns two uniform global keys to $\mathcal{V}$.

Convert committed values into authenticated values: This procedure can be executed multiple times. For some $i \in [1, \ell]$, $\mathcal{P}$ and $\mathcal{V}$ convert a committed value $\boldsymbol{x}_i \in \mathbb{F}_q^m$ only known by $\mathcal{P}$ to m authenticated values $[x_{i,1}], \ldots, [x_{i,m}]$:

1. Let path_i be the set containing all siblings of the nodes in the path from the i-th leaf to the root com_1. Prover $\mathcal{P}$ sends $(\boldsymbol{c}_i, \mathsf{path}_i)$ to $\mathcal{V}$, who verifies that $\mathsf{H}(\boldsymbol{c}_i)$ is a leaf node rooted in com_1.
2. By calling functionality $\mathcal{F}_{\mathsf{authZK}}$, the parties obtain authenticated bit-vectors $[\boldsymbol{sk}]_2, [\boldsymbol{r}]_2$ on key sk and randomness r, and then $\mathcal{P}$ proves in zero-knowledge that $\mathsf{com}_0 = \mathsf{H}([\boldsymbol{sk}]_2, [\boldsymbol{r}]_2)$.
3. The parties call functionality $\mathcal{F}_{\mathsf{authZK}}$ to compute $([x_{i,1}], \ldots, [x_{i,m}]) \leftarrow \boldsymbol{c}_i - \mathsf{PRF}([\boldsymbol{sk}]_2, i) \in \mathbb{F}_q^m$, and then output $\{[x_{i,j}]\}_{j \in [1,m]}$.

Figure 9: **Protocol for converting committed values into authenticated values in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model.**

the simulator can easily recover $\boldsymbol{x}_i$ by decrypting $\boldsymbol{c}_i$ for $i \in [1, \ell]$. This also implies the binding property. Together with the soundness of the ZK protocol realizing $\mathcal{F}_{\mathsf{authZK}}$, we can ensure the consistency between authenticated values and committed values.

Note that this commitment $(\mathsf{com}_0, \mathsf{com}_1)$ itself is not equivocal if we use the natural "open" algorithm that sends (sk, r): although it is possible to equivocate the key sk to any value by programming the random oracle, the function PRF is fixed. Equivocating from $\boldsymbol{x}_i$ to $\boldsymbol{x}_i'$ would require finding a key sk' such that $\mathsf{PRF}(sk', i) - \mathsf{PRF}(sk, i) = \boldsymbol{x}_i - \boldsymbol{x}_i'$ over $\mathbb{F}_q^m$. However, we can make it equivocal by an interactive opening: instead of directly sending (sk, r), we can send $\boldsymbol{c}_i$ and the corresponding path that can be verified with com_1, and prove knowledge of a key sk and a randomness r such that $\mathsf{com}_0 = \mathsf{H}(sk, r)$ and the other relationship on $\boldsymbol{c}_i$ hold. In this way, we can use the zero-knowledge property to equivocate the commitment.

Instantiation of PRF. We use LowMC [ARS+15] to instantiate PRF for reducing circuit complexity. One issue with LowMC is that it contains a lot of XOR gates. Although they are free cryptographically, the computation complexity can be fairly high. We adopt the following optimizations for competitive performance:

- Similar to the signature scheme Picnic [ZCD+19], we need to run PRF on a single key for many times, and thus can precompute the matrix multiplication about the key *only once* and use it for all PRF evaluations.
- We pick the block size as 64 bits to further reduce the number of XOR operations. The resulting protocol is highly efficient, and can convert 18,000 publicly committed data blocks (totally 144KB) to authenticated values per second.
- To reduce the number of rounds in LowMC, we choose the data complexity to be 2^{30} blocks, which is sufficient to commit 8 GB data. If the data is larger than that, we can just pick a new PRF key and commit this key.

Comparing with other candidates. We briefly discuss the concrete efficiency of our protocol for one commitment-authentication conversion, and compares it with other alternatives shown in Table 1. Here, we ignore the efficiency comparison for the commitment-generation phase, as it needs to be executed only once.

Scheme	This work	SHA-256	LowMC-256
Time (μs)	55	395	$\geq$ 1000
Comm. (bits)	62	705	49

Table 1: **Efficiency comparison between our protocol and alternative protocol with natural commitments.** Running time in microsecond (μs) is based on two Amazon EC2 machines of type `m5.2xlarge`.

For SHA-256 and LowMC-256, they refer to building a hash function modeled as a random oracle, and further construct a commitment on message x via $\mathsf{H}(x, r)$ with a randomness r. For SHA-256, one invocation takes 22573 AND gates and can commit 256 bits of messages. For LowMC-256, we first pick a LowMC block cipher with 256-bit key and block sizes, and then use Davies–Meyer to build a hash function. The SHA-256 method requires a lot of communication due to a large circuit size. The LowMC-256 approach is significantly slower compared to ours because: 1) our 64-bit block cipher only computes 64-bit matrix multiplication, but LowMC-256 needs 256-bit matrix multiplication meaning 16 times more operations; 2) we only need 11 rounds but LowMC-256 needs 53 rounds; 3) we can use a fixed key for all messages but LowMC-256 needs to rekey for every block of the message.

Conversion from authenticated values to publicly committed values. In some applications, two parties $\mathcal{P}$ and $\mathcal{V}$ may want to convert authenticated values (say, output by some MPC protocol) into a public commitment on the same values. Based on the protocol $\Pi_{\mathsf{NICom}\to[\cdot]}$ shown in Figure 9, this is easy to be realized by the following execution:

1. To convert authenticated values $\{[x_{i,j}]\}_{i\in[1,\ell],j\in[1,m]}$ into publicly committed values, $\mathcal{P}$ commits these vectors $(x_{i,1}, \ldots, x_{i,m})$ for $i \in [1, \ell]$ by executing the commitment-generation phase of protoocol $\Pi_{\mathsf{NICom}\to[\cdot]}$. Then $\mathcal{P}$ publishes the resulting commitment $(\mathsf{com}_0, \mathsf{com}_1)$.

Protocol Π_{MatMul}

Inputs: A prover $\mathcal{P}$ and a verifier $\mathcal{V}$ have three authenticated matrices $[\mathbf{A}], [\mathbf{B}]$ and $[\mathbf{C}]$, where $\mathbf{A} \in \mathbb{F}_q^{n\times m}, \mathbf{B} \in \mathbb{F}_q^{m\times \ell}$ and $\mathbf{C} \in \mathbb{F}_q^{n\times \ell}$.

Protocol execution: $\mathcal{P}$ proves in zero-knowledge that $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$ holds by interacting with $\mathcal{V}$ as follows:

1. $\mathcal{V}$ samples $\boldsymbol{u} \leftarrow (\mathbb{F}_{q^k})^n, \boldsymbol{v} \leftarrow (\mathbb{F}_{q^k})^\ell$, and then sends them to $\mathcal{P}$.
2. $\mathcal{P}$ and $\mathcal{V}$ compute $[\boldsymbol{x}]^\top := \boldsymbol{u}^\top \cdot [\mathbf{A}]$ and $[\boldsymbol{y}] := [\mathbf{B}] \cdot \boldsymbol{v}$ locally. Both parties also compute $[z] := \boldsymbol{u}^\top \cdot [\mathbf{C}] \cdot \boldsymbol{v}$.
3. The parties compute $[z'] := [\boldsymbol{x}]^\top \cdot [\boldsymbol{y}]$ by calling the (mult) command of $\mathcal{F}_{\mathsf{authZK}}$, where $z' = \boldsymbol{x}^\top \cdot \boldsymbol{y}$.
4. Both parties execute the $\mathsf{CheckZero}$ procedure on $[z] - [z']$ to verify that $z = z'$. If the check fails, $\mathcal{V}$ outputs false and aborts; otherwise, it outputs true.

Figure 10: **Zero-knowledge protocol for proving matrix multiplication in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model.**

2. Then, $\mathcal{P}$ and $\mathcal{V}$ execute protocol $\Pi_{\mathsf{NICom}\rightarrow[\cdot]}$ to convert commitment $(\mathsf{com}_0, \mathsf{com}_1)$ into authenticated values $\{[x'_{i,j}]\}_{i\in[1,\ell],j\in[1,m]}$.
3. The parties call the $\mathsf{CheckZero}$ procedure on $[x'_{i,j}] - [x_{i,j}]$ for all $i \in [1,\ell], j \in [1,m]$, and abort if the check fails.

6 More Optimizations for ML Applications

In this section, we will discuss several optimizations for key components in the *machine learning* (ML) applications and how they are connected. Then, we describe how to support various types of ML algorithms by extending TensorFlow [AAB+15].

6.1 Optimizing Matrix Multiplication

By generalizing the Freivalds algorithm [Fre77], we propose a ZK protocol to prove matrix multiplication with dimension $n \times n$ over a field $\mathbb{F}_q$ (for any prime $q \geq 2$), which only needs to prove n private multiplications rather than n^3 using a naive algorithm. Since the intuition of the protocol has been discussed in Section 3.3, we directly present the ZK protocol in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model in Figure 10, where suppose that two parties have computed the authenticated values on all entries in the matrices to be proven by calling the (input) command of $\mathcal{F}_{\mathsf{authZK}}$ before running this protocol.

In the following theorem, we prove the security of this protocol, where we refer the reader to [WYKW21] for the standard ZK functionality. The detailed proof of this theorem can be found in Appendix D.

Theorem 5. *Protocol Π_{MatMul} shown in Figure 10 UC-realizes the standard ZK functionality $\mathcal{F}_{\mathsf{ZK}}$ in the presence of a static, malicious adversary with soundness error $3/q^k$ in the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model.*

Further optimizations. We can further optimize the protocol shown in Figure 10 by letting the verifier send a random seed to the prover and then the two parties compute $\boldsymbol{u}$ and $\boldsymbol{v}$ by applying a random oracle to the seed.

In protocol Π_{MatMul}, the parties compute $[z'] = [\boldsymbol{x}]^\top \cdot [\boldsymbol{y}]$ by calling the (mult) command of $\mathcal{F}_{\mathsf{authZK}}$. This require communication of $O(m \log q)$ bits. To optimize the communication cost, we can define a multivariate polynomial $f(\boldsymbol{x}, \boldsymbol{y}, z) = \boldsymbol{x}^\top \cdot \boldsymbol{y} - z$, and then prove knowledge of $\boldsymbol{x}, \boldsymbol{y}, z$ such that $f(\boldsymbol{x}, \boldsymbol{y}, z) = 0$ using the latest ZK protocol [YSWW21]. This optimization can reduce the communication cost to only $O(k \log q)$ bits, independent of m.

6.2 Support Fixed-Point and Floating-Point

There are many non-linear operations in typical ML algorithms, including ReLU, Max Pooling, Sigmoid, SoftMax, etc. These operations are complicated to compute, and may often cause some accuracy loss when values are represented as fixed-point numbers. In our implementation, we support native IEEE-754 single-precision number in ZK proofs so that we can obtain maximum accuracy.

Encoding signed, fixed-point numbers. Linear layers and non-linear layers appear alternately. It is crucial to encode data in $\mathbb{F}_p$ for linear layers so that we can enjoy our highly efficient matrix-multiplication protocol described as above. For non-linear layers, data is encoded as floating-point numbers. To eventually convert between floating-point numbers and elements over $\mathbb{F}_p$, we need to find a way to encode signed, fixed-point numbers into $\mathbb{F}_p$, and execute an encoding procedure in another direction.

Given a prime $p > 2$, we can define an encoding procedure from $\mathbb{Z}$ to $\mathbb{F}_p$ as $\mathsf{Encode}(x \in \mathbb{Z}) = (x \bmod p)$, where an integer lies in $[-(p-1)/2, (p-1)/2]$. Note that field elements over $\mathbb{F}_p$ are represented in $[0, p-1]$. The corresponding decoding procedure is described as follows:

$$\mathsf{Decode}(x \in \mathbb{F}_p) = \begin{cases} x, & x \le (p-1)/2 \\ x - p, & x > (p-1)/2 \end{cases}$$

Now given a fixed-point number x and a precision parameter $s \in \mathbb{N}$, we can encode x into $\mathbb{F}_p$ by $\mathsf{Encode}(\lfloor 2^s \cdot x \rceil)$. If $\lfloor 2^s \cdot x \rceil \in [-(p-1)/2, (p-1)/2]$, there is almost no accuracy loss. Encoding in another direction from elements over $\mathbb{F}_p$ to fixed-point numbers can be executed in a straight-forward inverse process. There is one caveat: since we will perform matrix multiplication after non-linear layers, it is important to leave enough slack so that the precision does not overflow. In our implementation, we use a Mersenne prime $p = 2^{61} - 1$ and encode fixed-point numbers into a 30-bit range (where $s = 16$). Since in our application, the numbers never reach close to the 30-bit range, this ensures that the matrix multiplication would not overflow.

Converting between floating-point and fixed-point numbers. With values encoded as fixed-point numbers, we could convert between these fixed-point numbers and their binary representation via the arithmetic-Boolean conversion as shown in Section 4 and the encoding procedure described as above. Thus, the remaining task is to design efficient Boolean circuits for conversions between fixed-point and floating-point numbers. We use the single-precision circuits in EMP [WMK16], where the operations conform with the IEEE-754 standard. To perform the conversion from a floating-point number to a fixed-point number, we follow the definition of IEEE-754. The key components are private logical left shift and right shift, each of which is implemented using a

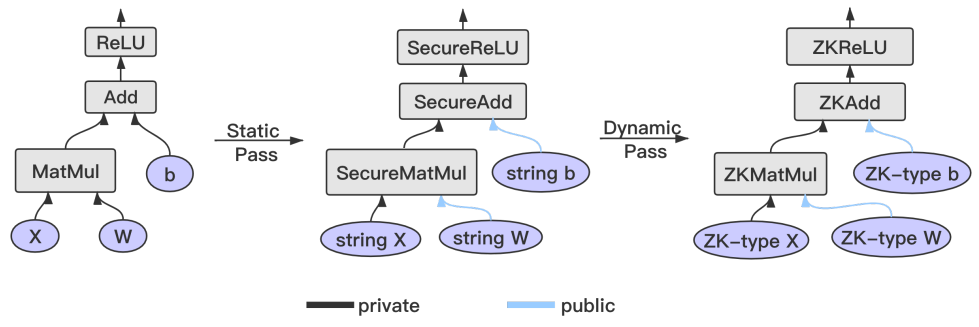

Figure 11: **Integration with TensorFlow.** Static and dynamic passes used in Rosetta to connect TensorFlow and our ZK protocol.

$(n \log n)$-sized circuit when shifting n bits. This procedure takes about 580 AND gates for $n = 61$. However, we found that converting from fixed-point to floating-point numbers is about $3\times$ slower, since private logical right shift is done with $n = 61$, but private logical left shift is handled with $n = 24$ (defined by IEEE-754). To close the efficiency gap between two directions, we can let the prover provide a converted floating-point number as an extended witness on-demand, and then only prove in zero-knowledge the conversion from a floating-point number to a fixed-point number.

6.3 Integrating with TensorFlow

We integrate the algorithms into Rosetta [CHS+20], which is an efficient and easy-to-use framework based on TensorFlow [AAB+15]. Specifically, we implemented our ZK backend protocol in C++ to maintain high efficiency and integrated to the backend of TensorFlow. In this way, developers could use simple interfaces in the frontend (in Python) to build complicated machine learning models without knowing details of the underlying cryptographic protocols. As a result, one can reuse the original code and interfaces of TensorFlow, and import an additional package to enable our ZK protocol. Below we discuss details of our integration. The main components of Rosetta are *static* and *dynamic passes* described as follows.

Static pass. In the frontend of TensorFlow, developers could write a model with Python language. The underlying compiler will convert the model into a graph, which consists of nodes and edges. Nodes are actually different operators, and edges are inputs/outputs of operators with specific data types (e.g., `int` and `float`). Static pass, described in Figure 11, is implemented in our framework, which additionally turns this graph into an abstract secure graph. Secure graph differs from the original graph in edges and nodes. Particularly, all the edges in secure graph are `string` type, which will contain the input and output information of each operator implemented with the underlying protocol (e.g., authenticated values in our ZK protocol). This is designed to be applicable to various cryptographic algorithms or protocols. Secure operators additionally specify the edges to be either public or private according to applications. The nodes in secure graph represent secure operators as shown in Figure 11.

Dynamic pass. The graph will be executed by TensorFlow in the backend when data is fed, and the string-type data will flow across the graph. Dynamic pass shown in Figure 11 is designed to integrate the graph execution with our ZK protocol. When handling a specific operator (e.g., matrix

	50 Mbps	200 Mbps	500 Mbps	1 Gbps
	Conversions			
A2B	107 μs	45 μs	34 μs	29 μs
B2A	109 μs	49 μs	38 μs	33 μs
C2A	56 μs	55 μs	55 μs	55 μs
Fix2Float	50 μs	46 μs	46 μs	46 μs
Float2Fix	49 μs	46 μs	46 μs	46 μs
	Machine Learning (ML) Functions			
Sigmoid	2.1 ms	1.6 ms	1.6 ms	1.6 ms
Max Pooling	1.6 ms	0.5 ms	0.4 ms	0.4 ms
ReLU	908 μs	262 μs	185 μs	188 μs
SoftMax-10	209 ms	157 ms	161 ms	171 ms
Batch Norm	415 ms	261 ms	257 ms	269 ms
	Matrix Multiplications			
MatMult-512	361 ms	186 ms	185 ms	185 ms
MatMult-1024	2.42 s	1.48 s	1.39 s	1.37 s
MatMult-2048	15.19 s	11.30 s	10.63 s	10.39 s

Table 2: **Performance of the basic building blocks.** The dimension of Max Pooling is 2×2. The dimension of Batch Normalization is $[1, 16, 16, 4]$, which stands for the batch size, height, weight and channels. For ML functions, the inputs and outputs are authenticated values in $\mathbb{F}_p$ with $p = 2^{61} - 1$. The performance result assumes that the inputs and outputs are all private to the verifier.

multiplication), dynamic pass will first convert the string-type data into ZK-friendly authenticated values (i.e., ZK type in Figure 11), and then call the underlying ZK protocol for this operator and get the authenticated output. Finally, dynamic pass will convert the resulting authenticated values back to string-type data, such that the data can be handled by TensorFlow and passed to the next operator. The universal composability of our protocol ensures that our approach is secure. To make sure all operators can be composed together directly as well as reduce the memory overhead, we encode the inputs and outputs of all operators into authenticated values over $\mathbb{F}_p$. When needing to prove using a Boolean circuit, we convert authenticated values over $\mathbb{F}_p$ into authenticated bits, and then convert the resulting authenticated bits back to authenticated values over $\mathbb{F}_p$, using our arithmetic-Boolean conversion protocols described in Section 4.

Extendibility. In addition to our ZK protocols, Rosetta [CHS+20] is also capable of integrating with other cryptographic protocols and algorithms, such as MPC and homomorphic encryption. It is feasible to support mixed protocols between ZK proofs and MPC, where we will leave as a future work.

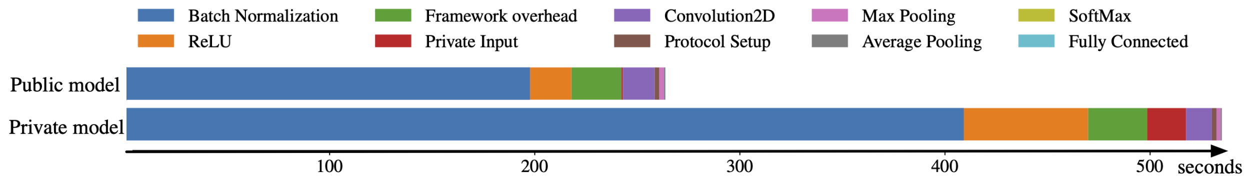

Figure 12: **Execution-time decomposition for ResNet-101 Inference.** The top bar is for public-model private-feature inference; the bottom bar is for private-model private-feature inference. The network bandwidth is throttled to 200 Mbps.

7 Performance Evaluation

In this section, we benchmark the speed of Mystique and how it performs on large-scale ML-inspired applications. We used three neural network models: LeNet-5[2] (5 layers, 62000 model parameters), ResNet-50 (50 layers, 23.5 million model parameters), and ResNet-101 (101 layers, 42.5 million model parameters). All experimental results are obtained by running the protocol over two Amazon EC2 machines of type `m5.2xlarge`, each with 32 GB memory. We use all CPU resources but only a fraction of the memory. The largest example is for ResNet-101 that uses 12 GB of memory. Our implementations use the latest sVOLE-based protocol [YSWW21] as the underlying ZK proof. All our implementations achieve the computational security parameter $\lambda = 128$ and statistical security parameter $\rho \geq 40$.

7.1 Benchmarking Our Building Blocks

We test the performance of our key building blocks discussed in this paper and summarized the results in Table 2. From this table, we can see that our protocol is highly scalable and all basic operations are highly efficient. The arithmetic-Boolean conversion (i.e., A2B and B2A) consists of two phases. In the preprocessing phase, two parties generate random zk-edaBits, and the execution time per zk-edaBit decreases from 95 μs to 19 μs when the bandwidth increases from 50 Mbps to 1 Gbps. In the online phase, two parties can convert authenticated wire values between arithmetic and Boolean circuits cheaply. The efficiency of the conversion from a public commitment to privately authenticated values (i.e., C2A) is mainly dominated by the computation of PRF in a Boolean circuit. It only takes around 56 μs to apply the PRF to a 64-bit data block, when the network bandwidth is at least 50 Mbps, due to the high communication efficiency of our protocol. The terms Fix2Float and Float2Fix represent the conversions between fixed-point and floating-point numbers, where the execution time for both conversions is around 46 μs per conversion when the network bandwidth is larger than 50 Mbps.

For the ZK proof of matrix multiplication (i.e., MatMul), our protocol can obtain around 185 ms of execution time for dimension 512×512, when the network bandwidth is at least 200 Mbps. The execution time is increased to about 1.5 s and 11 s for dimensions 1024×1024 and 2048×2048, respectively. The main efficiency bottleneck is the local computation of matrix multiplication by the prover. Compared to the state-of-the-art ZK proof for matrix multiplication [YSWW21], which

[2]We use ReLU as activation function instead of tanh for better accuracy.

Model	Image	LeNet-5	ResNet-50	ResNet-101
Communication				
Private	Private	16.5 MB	1.27 GB	1.98 GB
Private	Public	16.5 MB	1.27 GB	1.98 GB
Public	Private	16.4 MB	0.53 GB	0.99 GB
Execution time (seconds) in a 50 Mbps network				
Private	Private	7.3	465	736
Private	Public	7.5	463	735
Public	Private	6.5	210	369
Execution time (seconds) in a 200 Mbps network				
Private	Private	5.9	333	535
Private	Public	5.5	336	541
Public	Private	4.9	158	262

Table 3: **Performance of zero-knowledge neural-network inference.** All models are trained using the CIFAR-10 dataset.

takes 10 seconds to prove a 1024×1024 matrix multiplication over a network bandwidth of 500 Mbps, our ZK protocol achieves a $7\times$ improvement.

7.2 Benchmarking Private Inference

With these building blocks, we connect them together to build a ZK system to prove the inference of large neural networks as we described in Section 6.3. We consider three canonical settings, where the model parameters and model feature input can either be private to the prover or public to both parties. We focus on three neural networks: LeNet-5, ResNet-50, and ResNet-101. While the first example is relatively simple, the last two examples represent the state-of-the-art neural networks in terms of accuracy and complexity.

In Table 3, we summarize the performance for all neural networks, where the commitment on a model or data is not involved. After all optimizations, the slowest component in our protocol is Batch Normalization, which only exists in ResNet-50 and ResNet-101. For all models, we observe that when the model is private, the overall execution time is higher than the case in which the model parameters are public. This is because more operations have to be done in ZK proofs for private models. Regardless of this setting, LeNet-5 inference takes several seconds to finish. For all settings, ResNet-50 (resp., ResNet-101) takes about 2.6–5.6 (resp., 4.4–9) minutes to accomplish under a 200 Mbps network.

Microbenchmark. Figure 12 reports the microbenchmark of our ResNet-101 inference. We collect the time usage of different components including the protocol setup, private input (i.e., computing corresponding authenticated values), different operators and framework overhead. Significant amount of costs are used in Batch Normalization, ReLU, convolution2D and framework overhead. When the model is private, an additional proportion of time will also be used for private input. Note that the Batch Normalization takes around 70% of time in both cases because it involves complicated arithmetic operations and conversions between floating-point and fixed-point numbers,

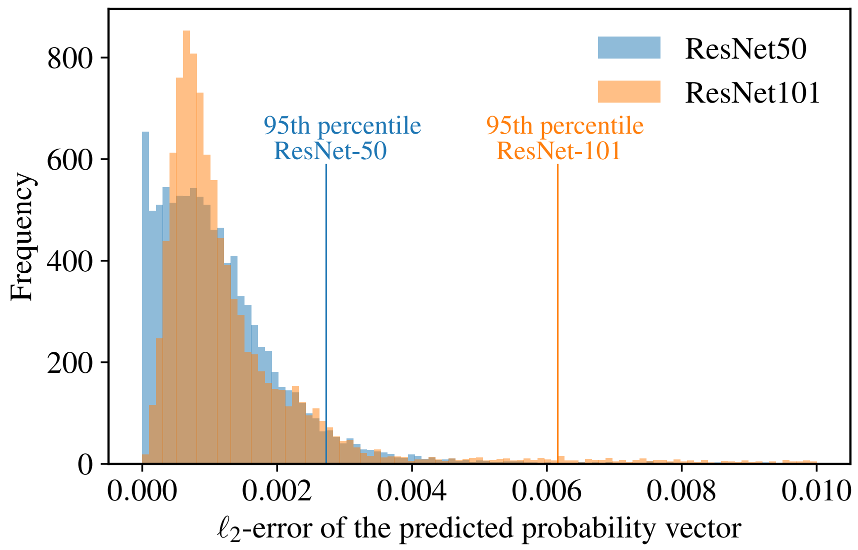

Figure 13: **ℓ_2-norm distance between the plaintext-inference probability vector and the ZK-inference probability vector.** The mean difference is 0.0011 for ResNet-50 and 0.0019 for ResNet-101.

which are costly to maintain accuracy. It will be an interesting future work to further improve the efficiency of Batch Normalization and ReLU without losing accuracy.

Benchmarking the accuracy. Our approach is highly accurate, but could still cause some accuracy loss. This could particularly be a concern for deep neural networks with hundred of layers where the error could propagate and get amplified. To benchmark the accuracy of our protocol, we ran the whole CIFAR-10 testing dataset [KNH] containing 10000 imagines. CIFAR is one of the standard ML dataset to benchmark the performance of algorithms. Imagines in CIFAR-10 are all labeled within 10 different classes, each imagine is a 32×32 color picture. The accuracy difference between the plaintext model and our ZK model is only 0.02% for both ResNet-50 and ResNet-101. To further understand the accuracy difference, we also compare the underlying probability vector predicted for each testing imagine. The dataset CIFAR-10 has 10 classes, and thus each inference produces a probability vector of length 10, denoted as $\boldsymbol{p}_i$ for all $i \in [1, 10000]$. The final prediction of the i-th testing imagine is $\mathsf{ArgMax}_i(\boldsymbol{p}_i)$. We are interested in the distribution of $\|\boldsymbol{p}_i - \boldsymbol{p}'_i\|_2$, where $\boldsymbol{p}_i$ is from plaintext inference and $\boldsymbol{p}'_i$ is from ZK inference. In Figure 13, we show the ℓ_2-norm differences of all 10000 inferences, and we can see that even for ResNet-101, the ℓ_2-norm difference is smaller than 0.006 for 95% of the case. For LeNet-5, 99.9% of the ℓ_2-norm difference are below 0.006. Therefore, for top-k accuracy such as $k = 5$ (commonly used for ImageNet), our ZK inference will be highly accurate.

7.3 End-to-End Applications

By connecting the private models/features to publicly committed models/features, Mystique can be used to build the three end-to-end applications mentioned in the Introduction. Since we use CIFAR-10 dataset, each image is of size 32×32 pixels and each pixel uses 3 bytes to represent the color. This means that one image is of size 3072 bytes and takes about 2.6 milliseconds to convert from

ML applications	LeNet-5	ResNet-50	ResNet-101
ZK for evasion attacks	9.8 s	316 s	524 s
ZK for genuine inference	7.2 s	16.4 m	28 m
ZK for private benchmark	8.2 m	4.4 h	7.3 h

Table 4: **Efficiency of our ZK system in different applications.** All execution time is reported based on a 200 Mbps network and two `m5.2xlarge` machines.

publicly committed values to privately authenticated values. The sizes of three models considered in this paper are 0.25 MB, 94 MB, and 170 MB. They take 1.7 seconds, 646 seconds, and 1169 seconds to convert from a public commitment to authenticated values that can be used in our protocols directly. The cost to "pull" a publicly committed model to be used in ZK proofs is high, but it could always be amortized over multiple private inferences.

- **ZK proofs for evasion attacks.** In this case, we need to prove knowledge of two almost identical inputs that get classified to different results under a public model. Therefore, the main cost is to prove the classification result in zero-knowledge under a public model twice.
- **ZK proofs for genuine inference.** In this application, the model parameters are private but publicly committed, while the input data is public. The main overhead is from: 1) proving the consistency between committed values and authenticated values for all model parameters; and 2) proving correct classification with private model and public input.
- **ZK proofs for private benchmark.** In this application, the testing data set is publicly committed and the model is public. Therefore, the main overhead comes from: 1) proving the consistency between committed testing data and authenticated data; and 2) proving correct classification with private input data and public model. In our example, we assume a testing data set of 100 images, and thus the second step is executed for 100 times, once for each image.

The execution time for every end-to-end application is reported in Table 4. Note that in the "ZK for private benchmark" application, 100 testing images were publicly committed, and then are converted to privately authenticated values using our conversion protocol shown in Section 5. Thus, the execution time for this application is significantly higher.

8 Conclusion

This paper presents various conversion protocols and builds zero-knowledge machine-learning inference on top of it. Although we have made a huge progress in proving ML algorithms in zero-knowledge, there are still limitations to our ZK system that deserves further exploration in future works. In particular, our ZK protocol can only prove to one verifier at a time, and the communication cost is fairly high compared to succinct ZK proofs like zk-SNARKs. We also observed a very high overhead for Batch Normalization, which may potentially be further optimized.

Acknowledgements

We thank Yuanfeng Chen, Gaofeng Huang, Junjie Shi, and Yilin Yan from MatrixElements Technologies for helping to integrate our protocol with Rosetta. Work of Kang Yang is supported by the National Key Research and Development Program of China (Grant No. 2020YFA0309703), and the National Natural Science Foundation of China (Grant Nos. 62022018, 61932019). Work of Jonathan Katz is supported by DARPA under Contract No. HR00112020025 and by NSF award #1563722. Work of Xiao Wang is supported in part by DARPA under Contract No. HR001120C0087. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as reflecting the position or policy of the the Department of Defense or the U.S. Government, and no official endorsement should be inferred.

References

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

[ABF+17] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *IEEE Symp. Security and Privacy 2017*, pages 843–862. IEEE, 2017.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 2087–2104. ACM Press, 2017.

[AOR+19] Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P. Smart, and Tim Wood. Zaphod: Efficiently combining lsss and garbled circuits in scale. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'19, page 33–44, 2019.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *Advances in Cryptology—Eurocrypt 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symp. Security and Privacy 2018*, pages 315–334. IEEE, 2018.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Advances in Cryptology—Crypto 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, 2019.

[BCC$^+$16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology—Eurocrypt 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, 2016.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS 2012*, pages 326–349, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery.

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM Conf. on Computer and Communications Security (CCS) 2019*, pages 291–308. ACM Press, 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *ACM Conf. on Computer and Communications Security (CCS) 2018*, pages 896–912. ACM Press, 2018.

[BCR$^+$19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology—Eurocrypt 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, 2019.

[BCTV14a] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *Advances in Cryptology—Crypto 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, 2014.

[BCTV14b] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium 2014*, pages 781–796. USENIX Association, 2014.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Recursive zk-snarks from any additive polynomial commitment scheme. Cryptology ePrint Archive, Report 2020/1536, 2020. https://eprint.iacr.org/2020/1536.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology—Eurocrypt 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, 2011.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In *ACM Conf. on Computer and Communications Security (CCS) 2016*, pages 565–577. ACM Press, 2016.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020.

[BMRS20] Carsten Baum, Alex J. Malozemoff, Marc Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for arithmetic circuits with nested disjunctions. Cryptology ePrint Archive, Report 2020/1410, 2020. https://eprint.iacr.org/2020/1410.

[BN20] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *Intl. Conference on Theory and Practice of Public Key Cryptography 2020, Part I*, LNCS, pages 495–526. Springer, 2020.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 1825–1842. ACM Press, 2017.

[CFH+15] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *IEEE Symp. Security and Privacy 2015*, pages 253–270. IEEE, 2015.

[CHS+20] Yuanfeng Chen, Gaofeng Huang, Junjie Shi, Xiang Xie, and Yilin Yan. Rosetta: A Privacy-Preserving Framework Based on TensorFlow. https://github.com/LatticeX-Foundation/Rosetta, 2020.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology—Eurocrypt 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, 2020.

[DEF+19] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *IEEE Symp. Security and Privacy 2019*, pages 1102–1120. IEEE, 2019.

[DIO20] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. Cryptology ePrint Archive, Report 2020/1446, 2020. https://eprint.iacr.org/2020/1446.

[DNNR17] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *Advances in Cryptology—Crypto 2017, Part I*, volume 10401 of *LNCS*, pages 167–187. Springer, 2017.

[EGK$^+$20] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *Advances in Cryptology—Crypto 2020, Part II*, LNCS, pages 823–852. Springer, 2020.

[FFG$^+$16] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *ACM Conf. on Computer and Communications Security (CCS) 2016*, pages 1304–1316. ACM Press, 2016.

[FLNW17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Advances in Cryptology—Eurocrypt 2017, Part II*, volume 10211 of *LNCS*, pages 225–255. Springer, 2017.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology—Eurocrypt 2015, Part II*, volume 9057 of *LNCS*, pages 191–219. Springer, 2015.

[Fre77] R. Freivalds. Probabilistic machines can use less running time. In *IFIP Congress*, 1977.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 113–122. ACM Press, 2008.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In *USENIX Security Symposium 2016*, pages 1069–1083. USENIX Association, 2016.

[HK20a] David Heath and Vladimir Kolesnikov. A 2.1 KHz zero-knowledge processor with BubbleRAM. In *ACM Conf. on Computer and Communications Security (CCS) 2020*, pages 2055–2074. ACM Press, 2020.

[HK20b] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In *Advances in Cryptology—Eurocrypt 2020, Part III*, volume 12107 of *LNCS*, pages 569–598. Springer, 2020.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *Advances in Cryptology—Asiacrypt 2017, Part I*, LNCS, pages 598–628. Springer, 2017.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 21–30. ACM Press, 2007.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *ACM Conf. on Computer and Communications Security (CCS) 2013*, pages 955–966. ACM Press, 2013.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM Conf. on Computer and Communications Security (CCS) 2018*, pages 525–537. ACM Press, 2018.

[KNH] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[KPPS20] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. MIRAGE: Succinct arguments for randomized algorithms with applications to universal zk-SNARKs. In *USENIX Security Symposium 2020*, pages 2129–2146. USENIX Association, 2020.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symp. Security and Privacy 2013*, pages 238–252. IEEE, 2013.

[RW19] Dragos Rotaru and Tim Wood. MArBled circuits: Mixing arithmetic and Boolean circuits with active security. In *Progress in Cryptology—Indocrypt 2019*, LNCS, pages 227–249. Springer, 2019.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Advances in Cryptology—Crypto 2020, Part III*, LNCS, pages 704–737. Springer, 2020.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In *ACM Conf. on Computer and Communications Security (CCS) 2019*, pages 1055–1072. ACM Press, 2019.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13(4):354–356, August 1969.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient Multi-Party computation toolkit. https://github.com/emp-toolkit, 2016.

[WSR$^+$15] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2015.

[WTs$^+$18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *IEEE Symp. Security and Privacy 2018*, pages 926–943. IEEE, 2018.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *IEEE Symp. Security and Privacy 2021*. IEEE, 2021.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *ACM Conf. on Computer and Communications Security (CCS) 2021*. ACM Press, 2021.

[YWL$^+$20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM Conf. on Computer and Communications Security (CCS) 2020*, pages 1607–1626. ACM Press, 2020.

[ZCD$^+$19] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, and Vladmir Kolesnikov. Picnic. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[ZFZS20] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. Zero knowledge proofs for decision tree predictions and accuracy. In *ACM Conf. on Computer and Communications Security (CCS) 2020*, pages 2039–2053. ACM Press, 2020.

A Proof of Theorem 1

The security of protocol $\Pi_{\mathsf{zk\text{-}edaBits}}$ shown in Figure 6 in the presence of a malicious prover crucially depends on the "cut-and-bucketing" procedure. As in previous work [FLNW17, ABF$^+$17, WYKW21], we model this procedure as the "balls-and-buckets" game:

1. Adversary $\mathcal{A}$ prepares $\ell = NB + c$ balls denoted by $\mathcal{B}_1, \ldots, \mathcal{B}_\ell$. Every ball corresponds to a zk-edaBit, and can be either good or bad depending on whether it is honestly generated by $\mathcal{A}$.
2. The first N balls $\{\mathcal{B}_1, \ldots, \mathcal{B}_N\}$ are placed into N buckets, where $\mathcal{B}_i$ is assigned into the i-th bucket.
3. In the set $\{\mathcal{B}_{N+1}, \ldots, \mathcal{B}_\ell\}$, c balls are randomly chosen and opened. If one of the chosen balls is bad, then $\mathcal{A}$ loses the game. Otherwise, the game proceeds to the next step.
4. The remaining $N(B-1)$ balls are randomly divided into the N buckets, such that each bucket has an equal size B.
5. We define that a bucket is fully good (resp., fully bad) if all the balls inside it are good (resp., bad). $\mathcal{A}$ wins if and only if there exists at least one bucket that is fully bad, and all other buckets are either fully good or fully bad.

Lemma 1. *Assume $c \geq B - 1$, then adversary $\mathcal{A}$ wins the above game with probability at most* $\binom{N(B-1)+c}{B-1}^{-1}$.

Proof. Assume that $\mathcal{A}$ makes m buckets bad for $1 \leq m \leq N$. $\mathcal{A}$ wins if exactly mB balls are bad, and they are placed into the m buckets. The probability of this event is computed below. In the step 2, N balls of ℓ balls are first placed in the N buckets, and thus m bad balls of N balls defines the m bad buckets. Let $B^* = B - 1$ and $\ell^* = NB^* + c$. At the step 3 of the game, the probability that none of mB^* bad balls is chosen is

$$p_1 = \frac{\binom{\ell^* - mB^*}{c}}{\binom{\ell^*}{c}} = \frac{(NB^* + c - mB^*)!(NB^*)!}{(NB^* + c)!(NB^* - mB^*)!}.$$

Assuming that this occurs. We are left with $\ell^* - c = NB^*$ balls that have not been placed in the buckets, of which mB^* balls are bad. In the step 4, the probability that mB^* bad balls are exactly put in the m buckets is

$$p_2 = \frac{(NB^* - mB^*)!(mB^*)!}{(NB^*)!}.$$

Overall, the probability that adversary $\mathcal{A}$ wins the game is

$$p = p_1 \cdot p_2 = \frac{(NB^* + c - mB^*)!(mB^*)!}{(NB^* + c)!} = \binom{NB^* + c}{mB^*}^{-1}.$$

When $c \geq B - 1 = B^*$ and $1 \leq m \leq N$, the probability p is maximized in the case of $m = 1$. So the probability that $\mathcal{A}$ wins the game is bounded by $\binom{N(B-1)+c}{B-1}^{-1}$. □

Below, we give the formal proof of Theorem 1.

Proof. We first consider the case of a malicious prover, and then consider the case of a malicious verifier. In each case, we construct a simulator $\mathcal{S}$ given access to functionality $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$, which runs an adversary $\mathcal{A}$ as a subroutine when emulating $\mathcal{F}_{\mathsf{authZK}}$. In both cases, we show that no environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution.

Malicious prover. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. In the process of generating faulty zk-edaBits, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{authZK}}$ for $\mathcal{A}$ by sampling uniform "dummy" global keys, and recording all the values $\{r^i\}_{i\in[1,\ell]}$ and $\{(r_0^i, \ldots, r_{m-1}^i)\}_{i\in[1,\ell]}$ and their corresponding MAC tags received from adversary $\mathcal{A}$. Note that these values and MAC tags naturally define corresponding local keys.

2. Simulator $\mathcal{S}$ plays the role of an honest verifier to perform the consistency-check procedure with $\mathcal{A}$, using the "dummy" global keys and the local keys. In the process of computing circuit AdderModp, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{authZK}}$ by sending the output bits to $\mathcal{A}$, and recording the MAC tags on these output bits, sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$. $\mathcal{S}$ uses the bits and MAC tags to define corresponding local keys.

3. If the honest verifier (simulated by $\mathcal{S}$) aborts in the consistency-check procedure, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ and aborts. Otherwise, $\mathcal{S}$ sends $r^1, \ldots, r^N$ to functionality $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$, and also sends their corresponding MAC tags and the MAC tags of $\{(r_0^i, \ldots, r_{m-1}^i)\}_{i\in[1,N]}$ to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$.

It is clear that the view of adversary $\mathcal{A}$ is perfectly simulated by $\mathcal{S}$, since the "dummy" global keys sampled by $\mathcal{S}$ have the same distribution as the real global keys, and then the local keys are totally determined by the values and MAC tags chosen by $\mathcal{A}$. If the honest verifier aborts in the real protocol execution, then the verifier also aborts in the ideal-world execution (as $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$). Therefore, it remains to bound the probability of the event $\mathsf{BadEvent}$ that the honest verifier accepts in the real protocol execution, but there exits one outputting $\mathsf{zk\text{-}edaBit}$ $(r_0^i, \dots, r_{m-1}^i, r^i)$ for some $i \in [1, N]$ with $r^i \neq \sum_{h=0}^{m-1} r_h^i \cdot 2^h \mod p$. In the following, we show that $\mathsf{BadEvent}$ occurs with probability at most $\binom{N(B-1)+c}{B-1}^{-1} + \frac{1}{p^k} + \mathsf{negl}(\lambda)$.

In the $\mathsf{BatchCheck}$ procedure, the probability which the honest verifier does not abort but there exists some value that is opened incorrectly is bounded by $\mathsf{negl}(\lambda)$. Below, we assume that this does not happen. In the consistency-check procedure, if there are a correct $\mathsf{zk\text{-}edaBit}$ and an incorrect $\mathsf{zk\text{-}edaBit}$ in the same bucket, then the honest verifier would abort unless $\mathcal{A}$ breaks the security of $\mathsf{CheckZero}$ with probability at most $1/p^k + \mathsf{negl}(\lambda)$. Now, we also assume that there is no mixed $\mathsf{zk\text{-}edaBits}$ in the same bucket. Therefore, according to Lemma 1, we have the probability that $\mathsf{BadEvent}$ occurs is at most $\binom{N(B-1)+c}{B-1}^{-1}$.

If $\mathsf{BadEvent}$ doest not occur except with probability at most $\binom{N(B-1)+c}{B-1}^{-1} + \frac{1}{p^k} + \mathsf{negl}(\lambda)$, the output distributions of the honest verifier in the real-world and ideal-world executions are identical, as global keys are uniform, and the local keys are uniquely determined by the values and MAC tags known by $\mathcal{A}$ and the independent global keys.

Malicious verifier. Simulator $\mathcal{S}$ has access to functionality $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$, and interacts with $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{authZK}}$ by recording global keys Δ, Γ and the local keys for all authenticated values, which are sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$. Then $\mathcal{S}$ sends Δ and Γ to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$.

2. $\mathcal{S}$ receives a permutation π from $\mathcal{A}$, and places the $\mathsf{zk\text{-}edaBits}$ into N buckets following the protocol specification. In the opening procedure of c $\mathsf{zk\text{-}edaBits}$ (step 4), for $i \in [1, c]$, $\mathcal{S}$ samples $r^i \leftarrow \mathbb{F}_p$ and decomposes it as $(r_0^i, \dots, r_{m-1}^i)$, and defines corresponding MAC tags using these values as well as global keys Δ, Γ and the related local keys. Then, $\mathcal{S}$ executes the $\mathsf{BatchCheck}$ procedure with $\mathcal{A}$ using $\{(r_0^i, \dots, r_{m-1}^i, r^i)\}_{i \in [1,c]}$ and these MAC tags.

3. For each bucket, in the checking procedure between the first $\mathsf{zk\text{-}edaBit}$ and every other $\mathsf{zk\text{-}edaBit}$, $\mathcal{S}$ simulates as follows:

   (a) $\mathcal{S}$ samples $t \leftarrow \mathbb{F}_p$ and computes its corresponding MAC tag using Γ and the associated local key defined as above, and also defines $(t_0, \dots, t_{m-1})$ such that $t = \sum_{i=0}^{m-1} t_i \cdot 2^i \mod p$.

   (b) In the process of executing circuit $\mathsf{AdderModp}$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{authZK}}$ by recording the local keys on t_i for all $i \in [0, m)$, which are computed using the keys sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$. Then $\mathcal{S}$ defines the MAC tags on $t_0, \dots, t_{m-1}$ using Δ and the corresponding local keys.

   (c) Simulator $\mathcal{S}$ uses $t_0, \dots, t_{m-1}$ along with their corresponding MAC tags to execute the $\mathsf{BatchCheck}$ procedure with adversary $\mathcal{A}$.

   (d) $\mathcal{S}$ uses the MAC tag on $[t]_p$ to run the $\mathsf{CheckZero}$ procedure with $\mathcal{A}$.

In the $\mathcal{F}_{\mathsf{authZK}}$-hybrid model, the arithmetic values over $\mathbb{F}_p$ on all $\mathsf{zk\text{-}edaBits}$ are uniformly random from the view of adversary $\mathcal{A}$. Therefore, in the real protocol execution, for each check in the step 5, the value t is uniform in $\mathbb{F}_p$, where r is always masked by a uniform value s. Besides, $\mathcal{S}$

will pass the checks in the $\mathsf{BatchCheck}$ and $\mathsf{CheckZero}$ procedures, as it always uses the correct MAC tags and the opened bits $t_0, \ldots, t_{m-1}$ satisfy the relation $t = \sum_{i=0}^{m-1} t_i \cdot 2^i \mod p$ for every pairwise check. Overall, the view of adversary $\mathcal{A}$ is perfectly simulated by $\mathcal{S}$. It is clear that the output distribution of the honest prover in the real-world execution is identical to that in the ideal-world execution, since the output values over $\mathbb{F}_2$ or $\mathbb{F}_p$ by the honest prover are uniform under the $\mathsf{zk\text{-}edaBit}$ condition in both worlds, and the MAC tags output by the honest prover are totally determined by the keys chosen by $\mathcal{A}$ and these output values. This completes the proof. □

B Proofs of Security for Our Arithmetic-Boolean Conversion Protocols

B.1 Proof of Theorem 2

Proof. We consider two cases of a malicious prover or a malicious verifier. In each case, we construct a simulator $\mathcal{S}$, which runs an adversary $\mathcal{A}$ as a subroutine, when emulating $\mathcal{F}_{\mathsf{authZK}}$ with only circuit-based commands. In both cases, we show that no environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution.

Malicious prover. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ by recording the bits $\{r_0, \ldots, r_{m-1}\}$ and field element r, and their corresponding MAC tags, sent to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ by $\mathcal{A}$.

2. $\mathcal{S}$ executes the $\mathsf{BatchCheck}$ procedure with $\mathcal{A}$. If the values sent by $\mathcal{A}$ are not consistent with $z = x - r \mod p$ and the value computed by $\mathcal{S}$ using the corresponding MAC tag, then $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{authZK}}$ and aborts. Note that x has been extracted by $\mathcal{S}$ in the procedure of generating authenticated value $[x]_p$.

3. $\mathcal{S}$ decomposes z as $(z_0, \ldots, z_{m-1}) \in \{0,1\}^m$. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{authZK}}$ with circuit-based commands to execute the evaluation of circuit $\mathsf{AdderModp}$ with $\mathcal{A}$. If the input bits and MAC tags sent by $\mathcal{A}$ are not consistent with $r_0, \ldots, r_{m-1}$ and their corresponding MAC tags recorded by $\mathcal{S}$, then $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{authZK}}$ and aborts. Otherwise, $\mathcal{S}$ sends $x_0, \ldots, x_{m-1}$ to $\mathcal{A}$ and records their corresponding MAC tags computed with the MAC tags received from $\mathcal{A}$, where $\sum_{h=0}^{m-1} x_h \cdot 2^h \mod p = x$.

4. $\mathcal{S}$ sends the MAC tags on bits $x_0, \ldots, x_{m-1}$ to $\mathcal{F}_{\mathsf{authZK}}$.

In the real protocol execution, if the value opened by $\mathcal{A}$ is not equal to $z = x - r \mod p$ in the $\mathsf{BatchCheck}$ procedure, then the honest verifier would abort except with probability at most $1/p^k + \mathsf{negl}(\lambda)$. If the opened value is identical to z, then using the MAC tag to check the correctness of the opened value, is equivalent to using the global and local keys to check, according to the IT-MAC relationship. Thus, the $\mathsf{BatchCheck}$ procedure simulated by $\mathcal{S}$ is indistinguishable from the real checking procedure. In the evaluation of circuit $\mathsf{AdderModp}$ of the real protocol execution, the random bits input by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{authZK}}$ must be identical to $r_0, \ldots, r_{m-1}$ recorded by $\mathcal{S}$, unless $\mathcal{A}$ forges an MAC tag with probability $1/2^\lambda$. Therefore, in the real-world execution, the bit decomposition of x (i.e., $x_0, \ldots, x_{m-1}$) are sent to $\mathcal{A}$. Overall, the view of adversary $\mathcal{A}$ that is simulated by $\mathcal{S}$ is indistinguishable from the view of $\mathcal{A}$ in the real protocol execution.

Malicious verifier. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. In the initialization phase, $\mathcal{S}$ emulates $\mathcal{F}_{\textsf{zk-edaBits}}$ and receives two global keys from $\mathcal{A}$, and then sends them to functionality $\mathcal{F}_{\textsf{authZK}}$.
2. $\mathcal{S}$ emulates $\mathcal{F}_{\textsf{zk-edaBits}}$ by recording the local keys on $(r_0, \ldots, r_{m-1}, r)$, sent to $\mathcal{F}_{\textsf{zk-edaBits}}$ by $\mathcal{A}$.
3. $\mathcal{S}$ runs the $\textsf{BatchCheck}$ procedure with $\mathcal{A}$ by sampling $z \leftarrow \mathbb{F}_p$, and sending it and the corresponding MAC tag to $\mathcal{A}$, where the local keys on $[x]_p$ and $[r]_p$ have been extracted by $\mathcal{S}$.
4. $\mathcal{S}$ decomposes z as $(z_0, \ldots, z_{m-1}) \in \{0,1\}^m$, and executes the evaluation of circuit $\textsf{AdderModp}$ with $\mathcal{A}$ when emulating $\mathcal{F}_{\textsf{authZK}}$ with circuit-based commands. During the procedure, $\mathcal{S}$ records the local keys on the output bits of circuit $\textsf{AdderModp}$, which are computed with the global and local keys received from $\mathcal{A}$. Then, $\mathcal{S}$ sends these local keys to functionality $\mathcal{F}_{\textsf{authZK}}$.

In the $(\mathcal{F}_{\textsf{zk-edaBits}}, \mathcal{F}_{\textsf{authZK}})$-hybrid model, r is uniform in the view of adversary $\mathcal{A}$. Therefore, field element z in the real protocol execution is uniform, as it is masked by r. In conclusion, the view of adversary $\mathcal{A}$ simulated by $\mathcal{S}$ is perfectly indistinguishable from its view in the real-world execution, which completes the proof. □

B.2 Proof of Theorem 3

Proof. As such, we consider two cases of a malicious prover or a malicious verifier. In each case, we construct a simulator $\mathcal{S}$, which runs an adversary $\mathcal{A}$ as a subroutine when emulating functionality $\mathcal{F}_{\textsf{authZK}}$ with circuit-based commands. In both cases, we show that any environment $\mathcal{Z}$ cannot distinguish the real-world execution from the ideal-world execution.

Malicious prover. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. $\mathcal{S}$ emulates functionality $\mathcal{F}_{\textsf{zk-edaBits}}$ by recording the values $r_0, \ldots, r_{m-1}, r$ and their corresponding MAC tags, sent to $\mathcal{F}_{\textsf{zk-edaBits}}$ by $\mathcal{A}$.
2. $\mathcal{S}$ emulates $\mathcal{F}_{\textsf{authZK}}$ with only circuit-based commands, by sending the output bits to $\mathcal{A}$, and also receiving the corresponding MAC tags from $\mathcal{A}$. Then $\mathcal{S}$ defines $z_0, \ldots, z_{m-1}$ as the output bits of circuit $\textsf{AdderModp}$ and records their corresponding MAC tags.
3. $\mathcal{S}$ executes the $\textsf{BatchCheck}$ procedure with $\mathcal{A}$. If these opened values are not equal to $z_0, \ldots, z_{m-1}$, or the value associated with MAC tags sent by $\mathcal{A}$ does not match with that computed by $\mathcal{S}$ with the recorded MAC tags, then $\mathcal{S}$ sends $\textsf{abort}$ to $\mathcal{F}_{\textsf{authZK}}$ and aborts.
4. $\mathcal{S}$ computes the MAC tag on $[x]_p = z - [r]_p$ locally where $z = \sum_{h=0}^{m-1} z_h \cdot 2^h \mod p$, and sends it to $\mathcal{F}_{\textsf{authZK}}$.

Clearly, the view of adversary $\mathcal{A}$ simulated by $\mathcal{S}$ is perfect, except for the $\textsf{BatchCheck}$ procedure. In the real protocol execution, the honest verifier checks the correctness of opened values using its keys. In the ideal-world execution, $\mathcal{S}$ executes this procedure using the desired bits $z_0, \ldots, z_{m-1}$ and corresponding MAC tags. For $\textsf{BatchCheck}$, we know that the opened values are correct (and thus are equal to $z_0, \ldots, z_{m-1}$) if the honest verifier does not abort, except with probability $\textsf{negl}(\lambda)$. In this case, the checking procedure using the keys is the same as that using the MAC tags, according to the IT-MAC relationship. Therefore, the view of adversary $\mathcal{A}$ simulated by $\mathcal{S}$ is indistinguishable from the real view of $\mathcal{A}$. From the definitions of functionality $\mathcal{F}_{\textsf{authZK}}$ and circuit $\textsf{AdderModp}$, we have that $\sum_{h=0}^{m-1} z_h \cdot 2^h = \sum_{h=0}^{m-1} x_h \cdot 2^h + \sum_{h=0}^{m-1} r_h \cdot 2^h \mod p$ (and thus $z = x + r \mod p$) except with probability $1/2^\lambda$ that is negligible in λ. Thus, the output of the honest verifier in the real-world execution is indistinguishable from that in the ideal-world execution, where the verifier's output (i.e., the local key on x) is determined by its global key and the value and MAC tag known by $\mathcal{A}$.

Malicious verifier. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. In the initialization phase, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ and receives two global keys from $\mathcal{A}$, and then sends them to functionality $\mathcal{F}_{\mathsf{authZK}}$.

2. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ by recording the local keys on $(r_0, \ldots, r_{m-1}, r)$, sent to $\mathcal{F}_{\mathsf{zk\text{-}edaBits}}$ by $\mathcal{A}$.

3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{authZK}}$ with only circuit-based commands, and receives the local keys on the output bits of all AND gates from adversary $\mathcal{A}$. Then, from these local keys, $\mathcal{S}$ defines and records the local keys on the output bits of circuit $\mathsf{AdderModp}$.

4. $\mathcal{S}$ runs the $\mathsf{BatchCheck}$ procedure with $\mathcal{A}$ by sampling $z \leftarrow \mathbb{F}_p$, and sending $(z_0, \ldots, z_{m-1}) \in \{0,1\}^m$ and their corresponding MAC tags to $\mathcal{A}$, where $\sum_{h=0}^{m-1} z_h \cdot 2^h = z \mod p$ and the MAC tags are computed with the global keys and local keys recorded by $\mathcal{S}$.

5. $\mathcal{S}$ computes the local key on $[x]_p = z - [r]_p$ locally and sends it to functionality $\mathcal{F}_{\mathsf{authZK}}$.

In the $(\mathcal{F}_{\mathsf{zk\text{-}edaBits}}, \mathcal{F}_{\mathsf{authZK}})$-hybrid model, r is uniform in $\mathbb{F}_p$ and kept secret against adversary $\mathcal{A}$. In the real protocol execution, we have that $(z_0, \ldots, z_{m-1})$ is the bit-decomposition of a random element z, based on the uniformity of r. Therefore, the view of adversary $\mathcal{A}$ simulated by $\mathcal{S}$ is perfectly indistinguishable from its view in the real-world execution, which completes the proof. □

C Proof of Theorem 4

Proof. We prove the security against a malicious prover and a malicious verifier separately. In each case, we construct a PPT simulator $\mathcal{S}$, who runs a PPT adversary $\mathcal{A}$ as a subroutine and emulates $\mathcal{F}_{\mathsf{authZK}}$ with only circuit-based commands. In both cases, we show that no PPT environment $\mathcal{Z}$ can distinguish between the real-world execution and ideal-world execution.

Malicious prover. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. $\mathcal{S}$ simulates random oracle H by recording the queries made by $\mathcal{A}$, and sending the random responses to $\mathcal{A}$ when keeping the consistency of responses.

2. In the commitment phase, after receiving $(\mathsf{com}_0, \mathsf{com}_1)$ from $\mathcal{A}$, simulator $\mathcal{S}$ extracts sk by retrieving the H query whose output is com_0. If no such query or there exists two different queries whose outputs are com_0, $\mathcal{S}$ sends abort to functionality $\mathcal{F}_{\mathsf{authZK}}$ and aborts.

3. $\mathcal{S}$ extracts $\boldsymbol{c}_i$ for all $i \in [1, \ell]$ by retrieving the random-oracle queries whose responses are identical to the values from the root com_1 of the Merkle tree to the leaves. If no such queries is found or there is a collision for H, $\mathcal{S}$ sends abort to functionality $\mathcal{F}_{\mathsf{authZK}}$ and aborts. Otherwise, for $i \in [1, \ell]$, $\mathcal{S}$ computes $\boldsymbol{x}_i := \boldsymbol{c}_i - \mathsf{PRF}(sk, i) \in \mathbb{F}_q^m$, and then sends $(\mathsf{commit}, x_{i,j}, q)$ to $\mathcal{F}_{\mathsf{authZK}}$ for $j \in [1, m]$.

4. For a conversion execution, after receiving $(\boldsymbol{c}'_i, \mathsf{path}_i)$ from $\mathcal{A}$ for some $i \in [1, \ell]$, $\mathcal{S}$ checks that $\boldsymbol{c}'_i = \boldsymbol{c}_i$ where $\boldsymbol{c}_i$ is extracted by it above. If the check fails, $\mathcal{S}$ aborts.

5. In the procedure of proving $\mathsf{com}_0 = \mathsf{H}([\boldsymbol{sk}]_2, [\boldsymbol{r}]_2)$, $\mathcal{S}$ emulates functionality $\mathcal{F}_{\mathsf{authZK}}$ with only circuit-based commands by recording sk' that consists of the bits sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$. If $sk' \neq sk$ then $\mathcal{S}$ aborts.

6. During the process of evaluating authenticated bits from $\boldsymbol{c}_i - \mathsf{PRF}([\boldsymbol{sk}]_2, i)$, $\mathcal{S}$ continues to emulate $\mathcal{F}_{\mathsf{authZK}}$ by checking that the input sk' by $\mathcal{A}$ is consistent with sk extracted by it, and recording the output values $x_{i,1}, \ldots, x_{i,m} \in \mathbb{F}_q$ and their corresponding MAC tags, sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$. If the check fails, $\mathcal{S}$ aborts. Otherwise, for $j \in [1, m]$, $\mathcal{S}$ sends the MAC tag on $x_{i,j}$ to functionality $\mathcal{F}_{\mathsf{authZK}}$ for the $(\mathsf{convertC2A})$ command.

In the random-oracle model, if no related query is made by $\mathcal{A}$, then $\mathcal{A}$ guesses correctly the hash output either com_0 or com_1 with probability at most $1/2^{2\lambda} = \mathsf{negl}(\lambda)$. As the output of H is random, the probability that there exists a collision is bounded by $q_{\mathsf{H}}^2/2^{2\lambda} = \mathsf{negl}(\lambda)$, where q_{H} is the number of queries made by adversary $\mathcal{A}$ to random oracle H. In the following, we assume that the bad events do not happen. Then $\mathcal{S}$ can successfully extract key sk from com_0. Based on the security of Merkle trees, $\mathcal{S}$ can also extract the $\boldsymbol{c}_i$ for all $i \in [1, \ell]$ by observing the random-oracle queries and responses. Therefore, all the committed values can be extracted successfully by $\mathcal{S}$ via computing $\boldsymbol{x}_i := \boldsymbol{c}_i - \mathsf{PRF}(sk, i)$ for all $i \in [1, \ell]$.

In the ideal-world execution, for the emulation of $\mathcal{F}_{\mathsf{authZK}}$, $\mathcal{S}$ directly checks whether the key sk' input by $\mathcal{A}$ is consistent with sk extracted by it. In the real-world execution, the validity of the input key sk' is checked by verifying the input MAC tag. The difference of the checking manner between two worlds is bounded by $1/2^{\lambda} = \mathsf{negl}(\lambda)$, as the successful probability that $\mathcal{A}$ forges the MAC tag on a flipped bit is at most $1/2^{\lambda}$. In the following, we assume that the key input by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{authZK}}$ is consistent with that committed by itself. According to the definition of functionality $\mathcal{F}_{\mathsf{authZK}}$, if the honest verifier does not abort in the real protocol execution, then the bits $x_{i,1}, \ldots, x_{i,m}$ input by $\mathcal{A}$ in the conversion phase are consistent with the bits that have been committed by $\mathcal{A}$ in the commitment phase.

Overall, the PPT environment $\mathcal{Z}$ cannot distinguish the real-world execution from the ideal-world execution, except with probability $\mathsf{negl}(\lambda)$.

Malicious verifier. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. $\mathcal{S}$ simulates H by recording the queries made by $\mathcal{A}$, and responds with random strings when keeping the consistency.

2. In the commitment phase, $\mathcal{S}$ samples $\mathsf{com}_0 \leftarrow \{0,1\}^{2\lambda}$ and $\boldsymbol{c}_i \leftarrow \mathbb{F}_q^m$ for $i \in [1, \ell]$. Then, it computes com_1 following the protocol description, and then sends $(\mathsf{com}_0, \mathsf{com}_1)$ to $\mathcal{A}$.

3. In the initialization procedure, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{authZK}}$ with circuit-based commands by recording two global keys, sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$. Then $\mathcal{S}$ sends the global keys to functionality $\mathcal{F}_{\mathsf{authZK}}$ in the ideal-world execution.

4. In the conversion phase, for some $i \in [1, \ell]$, $\mathcal{S}$ sends $(\boldsymbol{c}_i, \mathsf{path}_i)$ to $\mathcal{A}$ following the protocol specification. Then, for the procedures that proving knowledge of (sk, r) such that $\mathsf{com}_0 = \mathsf{H}(sk, r)$ and computing authenticated bits, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{authZK}}$ with only circuit-based commands by recording the local keys, sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$.

5. Finally, $\mathcal{S}$ computes and sends the local keys on $[x_{i,j}]$ for $j \in [1, m]$ to functionality $\mathcal{F}_{\mathsf{authZK}}$.

For the emulation of $\mathcal{F}_{\mathsf{authZK}}$, the simulation of $\mathcal{S}$ is perfect. Thus, we focus on proving the indistinguishability of commitment $(\mathsf{com}_0, \mathsf{com}_1)$. The probability that adversary $\mathcal{A}$ makes the query (sk, r) to random oracle H is at most $q_{\mathsf{H}}/2^{2\lambda}$. Therefore, com_0 simulated by $\mathcal{S}$ is computationally indistinguishable from the real value, except with probability $\mathsf{negl}(\lambda)$. By the pseudo-randomness of PRF, we have that the $\{\boldsymbol{c}_i\}_{i\in[1,\ell]}$ computed with PRF and key sk are computationally indistinguishable from uniformly random vectors in $\mathbb{F}_q^m$. Given the $\{\boldsymbol{c}_i\}_{i\in[1,\ell]}$, $\mathcal{S}$ computes com_1 in the same way as the real protocol execution. Thus, the simulation for com_1 is also computationally indistinguishable from the real value. Overall, no PPT environment $\mathcal{Z}$ can distinguish between the real-world execution and ideal-world execution, except with probability $\mathsf{negl}(\lambda)$. This completes the proof. □

D Proof of Theorem 5

Proof. We consider two cases of a malicious prover (i.e., soundness) and a malicious verifier (i.e., zero knowledge), respectively. In each case, we construct a simulator $\mathcal{S}$ given access to functionality $\mathcal{F}_{\mathsf{ZK}}$, which runs an adversary $\mathcal{A}$ as a subroutine when emulating $\mathcal{F}_{\mathsf{authZK}}$.

In this proof, we assume the parties have computed the matrices authenticated by verifier $\mathcal{V}$ (i.e., $[\mathbf{A}], [\mathbf{B}]$ and $[\mathbf{C}]$) by calling the (input) command of $\mathcal{F}_{\mathsf{authZK}}$. During the procedure, $\mathcal{S}$ can record the entries in these matrices and their corresponding MAC tags if the prover is corrupted, or record the global key and local keys on $[\mathbf{A}], [\mathbf{B}]$ and $[\mathbf{C}]$ if the verifier is corrupted. In the case of a malicious prover, $\mathcal{S}$ can send $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ extracted by itself to functionality $\mathcal{F}_{\mathsf{ZK}}$.

Malicious prover. The simulation of $\mathcal{S}$ is fairly easy and described as follows:

1. $\mathcal{S}$ samples $\boldsymbol{u}$ and $\boldsymbol{v}$ uniformly at random, and sends them to $\mathcal{A}$. Then $\mathcal{S}$ can locally compute the MAC tag on $[z]$.

2. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{authZK}}$, and can compute the MAC tag on $[z']$ from the MAC tags, sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$.

3. $\mathcal{S}$ computes the MAC tag on $[z] - [z']$ locally, and use it to check the correctness of the MAC tag sent by $\mathcal{A}$ in the $\mathsf{CheckZero}$ procedure.

Clearly, the view of adversary $\mathcal{A}$ is simulated perfectly by $\mathcal{S}$, except for the $\mathsf{CheckZero}$ procedure. We note that the checking with the MAC tag in the ideal-world execution is equivalent to that with the keys in the real-world execution, unless $\mathcal{A}$ can break the security of $\mathsf{CheckZero}$ with probability at most $1/q^k + \mathsf{negl}(\lambda)$. If the check fails for $\mathsf{CheckZero}$, then $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{ZK}}$ which sends false to the honest verifier and aborts. In the following, we assume that $z = z'$.

We remain to prove the probability that the honest verifier outputs true in the real protocol execution but $\mathbf{A} \cdot \mathbf{B} \neq \mathbf{C}$. We define a multivariable polynomial $f(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^\top \cdot (\mathbf{AB} - \mathbf{C}) \cdot \boldsymbol{v}$ over extension field $\mathbb{F}_{q^k}$ and $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are constants over field $\mathbb{F}_q$. If $\mathbf{A} \cdot \mathbf{B} - \mathbf{C} \neq \mathbf{0}$, then f is a non-zero polynomial. It is clear that the degree of f is at most 2. According to Lemma 2, we have the probability that $f(\boldsymbol{u}, \boldsymbol{v}) = 0$ is at most $2/q^k$ if $\mathbf{AB} \neq \mathbf{C}$, as $\boldsymbol{u}, \boldsymbol{v}$ are sampled uniformly at random after $\mathbf{A}, \mathbf{B}, \mathbf{C}$ have been defined.

Overall, the environment $\mathcal{Z}$ cannot distinguish between the real-world execution and ideal-world execution, except with probability $3/q^k + \mathsf{negl}(\lambda)$.

Malicious verifier. $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. After receiving $\boldsymbol{u}$ and $\boldsymbol{v}$ from $\mathcal{A}$, simulator $\mathcal{S}$ computes the local key on $[z]$ locally.
2. $\mathcal{S}$ emulates functionality $\mathcal{F}_{\mathsf{authZK}}$ by recording the local keys, sent to $\mathcal{F}_{\mathsf{authZK}}$ by $\mathcal{A}$. Then $\mathcal{S}$ uses these local keys to compute the local key on $[z']$.
3. During the CheckZero procedure, $\mathcal{S}$ computes the local key on $[z] - [z']$ locally, and sends it to $\mathcal{A}$.

It is easy to see that the view of $\mathcal{A}$ is simulated perfectly by $\mathcal{S}$. Therefore, no environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution, which implies the perfect zero-knowledge property. This completes the proof. □

Lemma 2 (Schwartz–Zippel Lemma). *Let $f(x_1, ..., x_n)$ be a non-zero multi-variant polynomial over a finite field $\mathbb{F}$ with degree $d \geq 0$. Let $r_1, ..., r_n$ be independent and uniformly random from $\mathbb{F}$, then* $\Pr[f(r_1, ..., r_n) = 0] \leq \frac{d}{|\mathbb{F}|}$.