ZK Contingent Payments for Trained Neural Networks

Zhelei Zhou, Xinle Cao, Jian Liu, Bingsheng Zhang, and Kui Ren

Zhejiang University

Abstract. Nowadays, neural networks have been widely used in many machine learning tasks. In practice, one might not have enough expertise to fine-tune a neural network model; therefore, it becomes increasingly popular to outsource the model training process to a machine learning expert. This activity brings out the needs of fair model exchange: if the seller sends the model first, the buyer might refuse to pay; if the buyer pays first, the seller might refuse to send the model or send an inferior model. In this work, we aim to address this problem so that neither the buyer nor the seller can deceive the other. We start from Zero Knowledge Contingent Payment (ZKCP), which is used for fair exchange of digital goods and payment over blockchain, and extend it to *Zero Knowledge Contingent Model Payment* (ZKCMP). We then instantiate our ZKCMP with two state-of-the-art NIZK proofs: zk-SNARKs and Libra. We also propose a random sampling technique to improve the efficiency of zk-SNARKs. We extensively conduct experiments to demonstrate the practicality of our proposal.

1 Introduction

Deep neural networks have recently gained much popularity due to their record-breaking performance on a wide range of machine learning tasks such as pattern recognition [5], medical diagnosis [9] and credit-risk assessment [3]. It is well-known that the final performance of a neural network model highly depends on its training data. However, the data owners usually do not have enough expertise to fine-tune the model, thereby they would like to outsource the training process to some machine learning (ML) experts. This gives ML experts an opportunity to monetize their skills, but brings the challenge of *fairly exchanging the model*: if the seller (i.e., ML expert) returns the model first, the buyer might refuse to pay the honorarium; if the buyer pays first, the seller might refuse to provide the model or provide an inferior model. In this paper, we aim to address this problem so that neither the buyer nor the seller can cheat the other.

Our starting point is *zero knowledge contingent payment* (ZKCP), which allows fair exchange of digital goods and payments over Bitcoin. Most cryptocurrencies like Bitcoin and Ethereum allow a payer to make a payment by specifying a condition that needs to be met in order for the money to be redeemed by the payee. One example of such conditions is a *hash-locked transaction* [2], where a payment can be redeemed by presenting a SHA256 preimage of a hash value. In ZKCP, the seller first encrypts the digital goods s as a ciphertext c and sends it to the buyer together with the hash of the encryption key $y := \mathsf{SHA256}(k)$. Then, the buyer makes a hash-locked transaction requiring the seller to post k to the blockchain to redeem the payment. Meanwhile, the buyer can decrypt c and obtain the purchased information. Moreover, the seller is required to prove that c really encrypts the "desired information" and the preimage of y is the encryption key, via a zero-knowledge (ZK) proof [13], which guarantees that the proof does not leak anything about s and k.

In our case, the digital goods s is a trained neural network, and the "desired information" means that s reaches a certain level of accuracy. We introduce a new notion named *zero knowledge* *contingent model payment* (ZKCMP), the whole procedure of which is as follows: (i) the buyer sends the training dataset to the seller; (ii) the seller trains a model s and commits it to the buyer, denoted as $c = \mathsf{Enc}(k, s)$ and $y = \mathsf{hash}(k)$; (iii) the buyer sends the testing dataset together with desired accuracy acc to the seller; (iv) the seller evaluates the model on the testing dataset; (v) the seller sends a ZK proof, proving that "the preimage of y can decrypt c and gets the previously committed model s, which achieves an accuracy of acc when being evaluated on the testing dataset"; (vi) the buyer verifies the proof and posts a hash-locked transaction for y; (vii) the seller redeems the payment by posting k to the blockchain.

For the ZK proof, we investigate both zk-SNARKs [6] and Libra [17]. The challenge for zk-SNARKs is that the proof generation phase is time-consuming and memory-consuming since it involves billions of gates for running a neural network over the testing dataset. To this end, we propose a *random sampling* technique to reduce the computational overhead for generating a proof, while still keeping the exchange secure and fair. The challenge for Libra is that the layered arithmetic circuit being used can only have a single output, thereby cannot support the proof for both accuracy and encryption. To conquer this, we construct separate arithmetic circuits for different output, and use zero-knowledge polynomial commitment to connect them (i.e., commit the I/O of each circuit so that circuits can be connected in a zero-knowledge way). A common challenge for both zk-SNARKs and Libra is that proving the final accuracy of the neural network is non-trivial. We design a customized circuit by composing a series of matrix operations so that the final accuracy can be proved efficiently. We summarize our contribution as follows:

- We propose a new notion named *zero knowledge contingent model payment* (ZKCMP), which allows fair exchange of a trained machine learning model and a cryptocurrency payment (e.g., Bitcoin) (cf. Sec. 3).
- We instantiate ZKCMP with zk-SNARKs and Libra respectively, which involves a series of sophistic circuit designs (cf. Sec. 4).
- We propose a random sampling technique to improve the efficiency of zk-SNARKs (cf. Sec. 4.1).
- We provide a full-fledged implementation and conduct experiments extensively (cf. Sec. 6).

2 Preliminaries

2.1 Notations

Let λ be the security parameter. Let $\mathsf{negl}(\cdot)$ denote a negligible function. Let PRF be a pseudorandom function. Let $\mathbb{F}$ be a finite field of prime order. We denote $[x]$ as the set $\{1, 2, \ldots, x\}$. Let $\mathcal{M}$ be a trained model, we denote w as the model parameter. We denote $\mathcal{D} := \{\langle x_i, L_i \rangle\}_{i \in n}$ (where x_i is the data sample entry, and L_i is its corresponding label) of size n.

2.2 Commitment scheme

A commitment scheme consisits of the following algorithms:

- $\mathsf{Setup}(1^\lambda)$. It is the public parameter generation algorithm that takes input as the security parameter λ, and it outputs public parameter pp (to be used by the other algorithms implicitly).
- $\mathsf{Commit}(m; r)$. It is the commitment generation algorithm that takes input as: the message m, the random coin r. It outputs commitment-opening pair (E, d). When r is not important in the context, we use $\mathsf{Commit}(m)$ for simplicity.

- $\mathsf{Verify}(c, d, m)$. It is the verification algorithm that takes input as: the commitment c, the opening d, the message m. It outputs a bit $b \in \{0, 1\}$, indicating acceptance or rejection.

Definition 1 (Commitment scheme) *A triple of algorithms* $\mathsf{COM} := (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$ *is a commitment scheme if the following properties holds:*

- ***Computational Hiding.*** *We say that a commitment scheme is computational hiding if for all PPT adversary* $\mathcal{A}$*, the adversarial advantage* $\mathsf{Adv}^{\mathcal{A},Hide}_{\mathsf{COM}}(\lambda)$ *is:*

$$\left| Pr\left[pp \leftarrow \mathsf{Setup}(1^\lambda); (m_0, m_1) \leftarrow \mathcal{A}(pp); b \leftarrow \{0,1\}; (E, d) \leftarrow \mathsf{Commit}(m_b) : \mathcal{A}(E) = b\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda)$$

- ***Computational Binding.*** *We say that a commitment scheme is computational binding if for any PPT adversary* $\mathcal{A}$*, the adversarial advantage* $\mathsf{Adv}^{\mathcal{A},Bind}_{\mathsf{COM}}(\lambda)$ *is:*

$$Pr\left[\begin{matrix} pp \leftarrow \mathsf{Setup}(1^\lambda); \\ (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(pp) \end{matrix} : \begin{matrix} m_0 \neq m_1 \wedge \\ \mathsf{Commit}(pp, m_0, r_0) = \mathsf{Commit}(pp, m_1, r_1) \end{matrix}\right] = \mathsf{negl}(\lambda)$$

In this work, we instantiate COM with salted $\mathsf{SHA256}$. In particular, we pick a random $r \leftarrow \{0,1\}^\lambda$, and commit to m by $E \leftarrow \mathsf{SHA256}(m||r)$, the opening is set as $d := (m, r)$.

2.3 Non-interactive Zero-Knowledge (NIZK) proofs

Let $\mathcal{R}$ be an efficiently decidable binary relation, which defines the NP language $\mathcal{L} := \{\mathsf{st}| \exists \mathsf{wit} : (\mathsf{st}, \mathsf{wit}) \in \mathcal{R}\}$. A NIZK proof system $\mathsf{NIZK}_\mathcal{R}$ for $\mathcal{R}$ consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda)$. It is the common reference string (CRS) generation algorithm that takes input as the security parameter λ, and it outputs a CRS crs.
- $\mathsf{Prove}(crs, \mathsf{st}, \mathsf{wit})$. It is the proof generation algorithm that takes input as: the CRS crs, statement st , witness wit. It outputs a proof π.
- $\mathsf{Verify}(crs, \mathsf{st}, \pi)$. It is the verification algorithm that takes input as: the CRS crs, statement s, proof π. It outputs a bit $b \in \{0, 1\}$, indicating acceptance or rejection.

Definition 2 (NIZK) *A triple of algorithms* $\mathsf{NIZK}_\mathcal{R} := (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is a NIZK proof for the relation* $\mathcal{R}$ *if the following properties holds:*

- ***Perfect Completeness.*** *We say that a NIZK system for* $\mathcal{R}$ *is perfectly complete if for any adversary* $\mathcal{A}$*, we have:*

$$Pr\left[\begin{matrix} crs \leftarrow \mathsf{Setup}(1^\lambda); (\mathsf{st}, \mathsf{wit}) \leftarrow \mathcal{A}(crs); \\ \pi \leftarrow \mathsf{Prove}(crs, \mathsf{st}, \mathsf{wit}) \end{matrix} : (\mathsf{st}, \mathsf{wit}) \notin \mathcal{R} \vee \mathsf{Verify}(crs, \mathsf{st}, \pi) = 1\right] = 1$$

- ***Computational Soundness.****We say that a NIZK system for* $\mathcal{R}$ *is computationally sound if for any PPT adversary* $\mathcal{A}$*, the adversarial advantage* $\mathsf{Adv}^{\mathcal{A},Sound}_{\mathsf{NIZK}}(\lambda)$ *is:*

$$Pr\left[\begin{matrix} crs \leftarrow \mathsf{Setup}(1^\lambda); \\ (\mathsf{st}, \pi) \leftarrow \mathcal{A}(crs) \end{matrix} : \mathsf{Verify}(crs, \mathsf{st}, \pi) = 1 \wedge \mathsf{st} \notin \mathcal{L}\right] = \mathsf{negl}(\lambda)$$

- ***Computational Zero-Knowledge.*** *We say that a NIZK system for $\mathcal{R}$ is computationally zero-knowledge if there exists a pair of PPT simulators* $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that for any PPT adversary* $\mathcal{A}$*, , the adversarial advantage* $\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},ZK}(\lambda)$ *is:*

$$\left| Pr\begin{bmatrix} crs \leftarrow \mathsf{Setup}(1^\lambda): \\ \mathcal{A}^{\mathsf{Prove}(crs,\cdot,\cdot)}(crs) = 1 \end{bmatrix} - Pr\begin{bmatrix} (crs^*, \mathsf{td}) \leftarrow \mathsf{Sim}_1(1^\lambda): \\ \mathcal{A}^{\mathsf{Sim}^*(crs^*,\mathsf{td},\cdot,\cdot)}(crs^*) = 1 \end{bmatrix} \right| = \mathsf{negl}(\lambda)$$

Where the oracle $\mathsf{Sim}^*(crs^*, \mathsf{td}, \mathsf{st}, \mathsf{wit}) := \mathsf{Sim}_2(crs^*, \mathsf{st}, \mathsf{td})$ *for* $(\mathsf{st}, \mathsf{wit}) \in \mathcal{R}$ *and the oracle outputs* $\perp$ *if* $(\mathsf{st}, \mathsf{wit}) \notin \mathcal{R}$.

In practice, the relation decision algorithm is instantiated by a circuit $\mathcal{C}_{\mathcal{R}}(\mathsf{st}, \mathsf{wit})$, which outputs 1 if $(\mathsf{st}, \mathsf{wit}) \in \mathcal{R}$; otherwise, it outputs 0.

zk-SNARKs. Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) [6] is a type of widely used NIZK proof systems. It achieves succinct proof size and verification time, whereas the proving cost is heavy. zk-SNARKs can be used to prove the satisfaction problem of a system of rank-1 quadratic equations over a finite field $\mathbb{F}$. See more details in [4].

Definition 3 *Denote N_g the number of rank-1 quadratic equations, N_v the number of variables, and ℓ the statement size. A system of rank-1 quadratic equations over $\mathbb{F}$ is a tuple $\mathcal{S} = ((a_j, b_j, c_j)_{j=1}^{N_g}, \ell, N_v)$ where $a_j, b_j, c_j \in \mathbb{F}^{1+N_v}$ and $\ell \leq N_v$. Such a system S is satisfiable with an input $x \in \mathbb{F}^\ell$ if there is a witness $w \in \mathbb{F}^{N_v - \ell}$ such that: $\forall j \in [N_g], \langle a_j, (1, x, w)\rangle \cdot \langle b_j, (1, x, w)\rangle = \langle c_j, (1, x, w)\rangle$. In such a case, we write $\mathcal{S}(x, w) = 1$.*

In brief, zk-SNARKs aims to prove the relation $\mathcal{R}_S = \{(x, w) \in \mathbb{F}^\ell \times \mathbb{F}^{N_v - \ell} : \mathcal{S}(x, w) = 1\}$ holds. In this article, we call the system of rank-1 quadrtic equations as *constraint system*, the rank-1 quadratic equation as *constraint equation* to make our expression clear.

Libra. Libra [17] is a zero-knowledge proof system that is designed for layered arithmetic circuits. The performance of Libra is competitive, it has very fast prover time and succinct proof size/verification time. To be specific, if C is the size of log-space uniform circuit with depth D, the prover time is $O(C)$ while the proof size and verification time is $O(D \cdot \log C)$. Libra uses zero-knowledge GKR protocol [12] and zero-knowledge verifiable polynomial delegation scheme (zkVPD) [18] as two main building blocks. The construction for non-interactive version of Libra is presented in Fig. 1. In Appendix A, we provide brief description of its main building blocks.

2.4 Neural networks

A neural network consists of a pipeline of layers, typically including a *fully connected* or a *convolutional* layer, an *activation* layer, a *pooling* layer and an *output* layer. Each layer receives input and processes it to produce an output that serves as input to the next layer.

In each fully connected layer, the input is multiplied by a *weight matrix* and added with a *bias vector*. Convolution can be converted into matrix multiplication and addition as well. An activation function is applied to build the nonlinear relationships between input and output. The output layer typically applies a Softmax function to compute a probability distribution in categorical classification. In the inference phase, it calculates the maximal value over the output of the last layer (typically a fully connected layer), which is called *logits*, to predict the most likely outcome.

```
Non-interactive Libra for $\mathcal{C}(\mathsf{st}, \mathsf{wit}) = 1$

$\mathsf{Libra.Setup}(1^\lambda)$:

  – Generate $(pp, vp) \leftarrow \mathsf{zkVPD.KeyGen}(1^\lambda)$;
  – Output $crs := (pp, vp)$;

$\mathsf{Libra.Prove}(\mathsf{st}, \mathsf{wit}, \mathcal{C})$:

  – Select a random bivariate polynomial $R$;
  – Compute $\overline{\mathsf{wit}}$ which is defined as Eq. 2;(cf. Eq. 2 in Appendix A)
  – Pick random $r \leftarrow \mathbb{F}$;
  – Commit $Com \leftarrow \mathsf{zkVPD.Commit}(\overline{\mathsf{wit}}, r, pp)$;
  – Generate GKR proof [12] $\pi_0$ showing $\mathcal{C}(\mathsf{st}, \mathsf{wit}) = 1$;
  – Compute $(e_1, e_2) \leftarrow \mathsf{hash}(\mathsf{st}, \pi_0, Com)$;
  – For $i \in \{1, 2\}$: generate proof $(y_i, \pi_i) \leftarrow \mathsf{zkVPD.Open}(\overline{\mathsf{wit}}, e_i, r, pp)$;
  – Output $\pi := (Com, \pi_0, y_1, \pi_1, y_2, \pi_2)$;

$\mathsf{Libra.Verify}(\pi)$:

  – Compute $(e_1, e_2) \leftarrow \mathsf{hash}(\mathsf{st}, \pi_0, Com)$;
  – Return 1 if and only if:
      • The GKR proof $\pi_0$ is verified;
      • For $i \in \{1, 2\}$:
          ∗ $\mathsf{zkVPD.Verify}(Com, e_i, y_i, \pi_i, vp) = 1$;
          ∗ $y_i \leftarrow \overline{\mathsf{wit}}(e_i)$;
```

Fig. 1: Non-interactive Libra for $\mathcal{C}(\mathsf{st}, \mathsf{wit}) = 1$

3 Design Overview

A brief introduction to ZKCP. ZKCP [1] [7] is a blockchain based fair exchange protocol. For the most common scenarios, the seller $\mathcal{S}$ has some digital goods s which the buyer $\mathcal{B}$ wants to purchase on condition that $f(s) = 1$ for a verification function $f : \{0,1\}^\lambda \mapsto \{0,1\}$. In the off-chain phase, $\mathcal{S}$ uses a symmetric encryption Enc to encrypt s with a random key k and publishes the ciphertext $c \leftarrow \mathsf{Enc}(s, k)$ and the committed key $E \leftarrow \mathsf{SHA256}(k)$ together with a ZK proof showing

$$\mathsf{SHA256}(k) = E \ \wedge \ f(\mathsf{Enc}^{-1}(c, k)) = 1 \ .$$

If the proof is correct, $\mathcal{B}$ and $\mathcal{S}$ then enter the on-chain phase using a hash-locked transaction [2].

Even through ZKCP has been extensively explored by both researchers and practitioners, to the best of our knowledge, there is still no formalization so far. For the first time, we formalize the syntax of ZKCP as follows:

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$: It is the public parameter generation algorithm that takes input as: the security parameter 1^λ, and it outputs the public parameter pp.
  - Invoke $crs \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$;
  - Output $pp := crs$.
- $(c, E, d, \pi) \leftarrow \mathsf{Prove}(pp, s)$: It is the prove algorithm that takes input as: the public parameter pp, the digital file s. It then outputs the ciphertext c, a commitment-opening pair (E, d) and a proof π.
  - Sample a random key $k \leftarrow \{0,1\}^\lambda$;
  - Generate the ciphertext $c \leftarrow \mathsf{Enc}(s, k)$, where Enc is a symmetry encryption;

  - Commit to the key $(E, d) \leftarrow \mathsf{COM.Commit}(k)$;
  - Generate $\pi \leftarrow \mathsf{NIZK.Prove}(pp, (c, E), (k, s))$;
  - Output (c, E, d, π).
- $b \leftarrow \mathsf{Verify}(pp, c, E, \pi)$. It is the verification algorithm that takes input as: the public parameter pp, the ciphertext c, the commitment E, and the proof π. It then outputs a bit $b \in \{0, 1\}$, indicating acceptance or rejection.
  - Verify $b \leftarrow \mathsf{NIZK.Verify}(pp, (c, E), \pi)$;
  - Output b.
- $\mathcal{F}_{ex}[\mathsf{COM}]$. It is the the trusted exchange functionality to guarantee the fairness and security of the payment. As depicted in Fig. 2, when $\mathcal{B}$ sends instruction (Commit, v, E) to $\mathcal{F}_{ex}[\mathsf{COM}]$, where v is the payment amount, and E is the committed key, $\mathcal{F}_{ex}[\mathsf{COM}]$ checks $\mathcal{B}$'s account balance $Acc_{\mathcal{B}} \geq v$ and then stores E. When $\mathcal{S}$ sends instruction (Redeem, k, d) to $\mathcal{F}_{ex}[\mathsf{COM}]$, where k is the key and d is the opening, $\mathcal{F}_{ex}$ calls COM to check the validity. If $\mathsf{COM.Verify}(E, d, k) = 1$, $\mathcal{F}_{ex}[\mathsf{COM}]$ transfers v from $\mathcal{B}$'s account balance $Acc_{\mathcal{B}}$ to $\mathcal{S}$'s account balance $Acc_{\mathcal{S}}$. $\mathcal{F}_{ex}[\mathsf{COM}]$ can be instantiated by hash-locked transaction.

The exchange functionality $\mathcal{F}_{ex}[\mathsf{COM}]$

It is parameterized with a commitment scheme COM.
The buyer $\mathcal{B}$'s account balance: $Acc_{\mathcal{B}}$;
The seller $\mathcal{S}$'s account balance: $Acc_{\mathcal{S}}$;

Description:

- Upon receiving (Commit, v, E) from $\mathcal{B}$:
  - Assert $Acc_{\mathcal{B}} \geq v$;
  - Store E;
- Upon receiving (Redeem, k, d) from $\mathcal{S}$:
  - Assert $\mathsf{COM.Verify}(E, d, k) = 1$;
  - Set $Acc_{\mathcal{S}} := Acc_{\mathcal{S}} + v$;
  - Set $Acc_{\mathcal{B}} := Acc_{\mathcal{B}} - v$;
  - Send k to $\mathcal{B}$.

Fig. 2: The exchange functionality $\mathcal{F}_{ex}[\mathsf{COM}]$

From ZKCP to ZKCMP. We extend the idea of ZKCP to a specific class of problems: paying for qualified trained models. Consider such a scenario, $\mathcal{S}$ has a trained model $\mathcal{M}$ which $\mathcal{B}$ wants to purchase. Since the model structure is a common knowledge, they only need to exchange the model parameters, denoted as w. However, the original ZKCP protocol is not sufficient for our purpose. This is because in practice $\mathcal{B}$ may want to test the accuracy of $\mathcal{M}$ w.r.t his own testing dataset $\mathcal{D}$ before paying for it; however, according to the ZKCP protocol, $\mathcal{S}$ can provide the encrypted model after seeing the testing dataset, which nullifies the testing soundness.

To address this issue, we propose a new notion named *zero knowledge contingent model payments* (ZKCMP). Intuitively, let us focus on the Prove. Prove serves two purposes: (i) send the encryption of the digital goods, so $\mathcal{S}$ cannot change it latter; (ii) prove the digital goods satisfies the requirement. We introduce a new algorithm Seal to seal the the model parameter w to separate these two purposes. The separation is important, since $\mathcal{S}$ must perform $\mathsf{Seal(w)}$ before $\mathcal{B}$ sending his testing dataset $\mathcal{D}$. After receiving $\mathcal{D}$, $\mathcal{S}$ generates a proof to prove the accuracy of $\mathcal{M}$.

The workflow of ZKCMP is presented in Fig. 3. Before the protocol starts, $\mathsf{Setup}(1^\lambda)$ is invoked by a trusted entity to generate pp. It is then sent to both $\mathcal{B}$ and $\mathcal{S}$. To prevent $\mathcal{S}$ from cheating, $\mathcal{S}$ is required to seal the model parameters w by invoking $(c, E, d) \leftarrow \mathsf{Seal}(w)$ and send the sealed model c and the committed key E to $\mathcal{B}$. In this way, $\mathcal{S}$ cannot modify the model later. Then $\mathcal{B}$ randomly picks a testing dataset $\mathcal{D} := \{\langle x_i, L_i \rangle\}_{i \in n}$ (where x_i is the data sample entry, and L_i is its corresponding label) of size n by invoking $\mathcal{D} \leftarrow \mathsf{Sample}(n)$ and sends $\mathcal{D}$ to $\mathcal{S}$ together with a threshold τ. After receiving $\mathcal{D}$, $\mathcal{S}$ evaluates the model on $\mathcal{D}$ by $F(w, x_i) = y_i \ \wedge \ \mathsf{argmax}(y_i) = L_i$, where F is the model prediction function corresponding to $\mathcal{M}$, and y_i is the final model output. argmax is used for the transformation from final model output to its corresponding label. Then $\mathcal{S}$ is required to prove that the accuracy of the trained model $\mathcal{M}$ on $\mathcal{D}$ exceeds the threshold τ as well as E is the commitment of k and c is the encryption of w with k. $\mathcal{S}$ obtains the proof π by invoking $\pi \leftarrow \mathsf{Prove}(pp, c, E, w, k, \mathcal{D}, \tau)$ and sends π back to $\mathcal{B}$. After that, $\mathcal{B}$ checks the validity of π by invoking $b \leftarrow \mathsf{Verify}(pp, c, E, \mathcal{D}, \tau, \pi)$. If $b = 1$, $\mathcal{B}$ sends (Commit, v, E) to the trusted exchange functionality $\mathcal{F}_{ex}[\mathsf{COM}]$. $\mathcal{S}$ can receive the payment by submitting k to $\mathcal{F}_{ex}[\mathsf{COM}]$; $\mathcal{F}_{ex}[\mathsf{COM}]$ will then send (k, d) to $\mathcal{B}$. Subsequently, $\mathcal{B}$ can obtain the model parameters $w \leftarrow \mathsf{Ext}(c, k)$. To prevent redundancy, we present the syntax of ZKCMP that differs from ZKCP here.

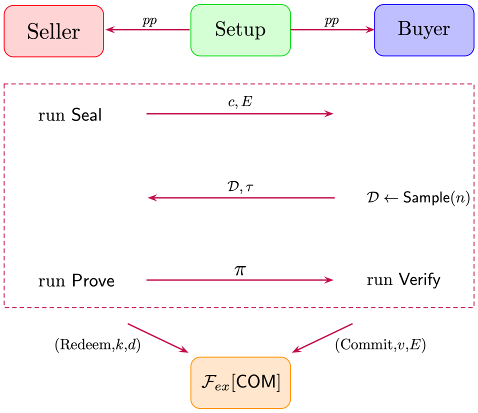


Fig. 3: Our designed protocol

- $(c, E, d) \leftarrow \mathsf{Seal}(w)$. It is the model sealing algorithm that takes input as: the model parameters w, and it outputs the sealed model c, the commitment-opening pair (E, d).
  - Sample a random key $k \leftarrow \{0, 1\}^\lambda$;
  - Compute $(w_1, \ldots, w_\ell) \leftarrow \mathsf{Convert}(w)$, where $\mathsf{Convert}$ is a function mapping model parameter w into strings (c.f. Sec.4.2);
  - For $i \in [\ell]$:
    - Generate $ks_i \leftarrow \mathsf{PRF}(k, i)$;
    - Set $c_i := w_i \oplus ks_i$;
  - Commit to the key by $(E, d) \leftarrow \mathsf{COM.Commit}(k)$;
  - Output $(c := (c_1, \ldots, c_\ell), E, d)$;

- $\pi \leftarrow \mathsf{Prove}(pp, c, E, w, k, \mathcal{D}, \tau)$. It is the prove algorithm that takes input as: the public parameter pp, the sealed model c, the committed key E, the model parameter w, the key k, the testing dataset $\mathcal{D}$, and the threshold τ. It then outputs a proof π.
    - Compute $I = \{i \mid F(w, x_i) = y_i \ \wedge\ \mathsf{argmax}(y_i) = L_i\}$;
    - Output $\pi \leftarrow \mathsf{NIZK.Prove}(pp, (c, E, \mathcal{D}, \tau), (w, k))$;
- $b \leftarrow \mathsf{Verify}(pp, c, E, \mathcal{D}, \tau, \pi)$. It is the verification algorithm that takes input as: the public parameter pp, the sealed model c, the committed key E, the testing dataset $\mathcal{D}$, the threshold τ and the proof π. It then outputs a bit $b \in \{0, 1\}$, indicating acceptance or rejection.
    - Output $b \leftarrow \mathsf{NIZK.Verify}(pp, (c, E, \mathcal{D}, \tau), \pi)$;

A ZKCMP protocol $\mathit{\Pi}$ is a three-party communication protocol: the buyer $\mathcal{B}$, the seller $\mathcal{S}$, and a trusted functionality $\mathcal{F}_{ex}$. We denote with $[a, b] \leftarrow \langle \mathcal{B}(F, \tau), \mathcal{S}(w, F, \tau), \mathcal{F}_{ex} \rangle$ the event that at the end of $\mathit{\Pi}$, $\mathcal{B}$ gets a and $\mathcal{S}$ gets b, where a, b can be $\perp$ meaning that the parties reject the execution, and neither party will learn any information. We denote with $\mathcal{E}_v$ the event that the seller $\mathcal{S}$'s account balance $Acc_{\mathcal{S}}$ increases v. We define the view of $\mathcal{B}$ as his money v and all the messages exchanged during the protocol: $View_{\mathcal{B}} := [v || Message\langle \mathcal{B}(F, \tau), \mathcal{S}(w, F, \tau), \mathcal{F}_{ex} \rangle || Out\langle \mathcal{B}(F, \tau), \mathcal{S}(w, F, \tau), \mathcal{F}_{ex} \rangle]$.

Definition 4 *A ZKCMP protocol $\mathit{\Pi}$ satisfies the following properties.*

- ***Completeness.*** *We say a ZKCMP protocol $\mathit{\Pi}$ is complete if for any $(w, \mathcal{D})$ such that $|I| \geq n \cdot \tau$, where $I := \{i \mid i \in [n] \ \wedge\ F(w, x_i) = y_i \ \wedge\ argmax(y_i) = L_i\}$, the following holds:*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda); \\ (c, E) \leftarrow \mathsf{Seal}(w); \\ \pi \leftarrow \mathsf{Prove}(pp, c, E, w, k, \mathcal{D}, \tau) \end{array} : \begin{array}{c} \mathsf{Verify}(pp, c, E, \mathcal{D}, \tau, \pi) = 1 \wedge \\ [k, \mathcal{E}_v] \leftarrow \langle \mathcal{B}(F, \tau), \mathcal{S}(w, F, \tau), \mathcal{F}_{ex} \rangle \end{array}\right] = 1$$

- *ε-**soundness.*** *We say a ZKCMP protocol $\mathit{\Pi}$ is ε-sound , $\varepsilon \geq 0$, if for any possibly malicious PPT $\hat{\mathcal{S}}$, if at the end of the protocol $\hat{\mathcal{S}}$'s account balance $Acc_{\hat{\mathcal{S}}}$ increases with non-negligible probability, then there exists an PPT extractor $\mathsf{Ext}_{\hat{\mathcal{S}}}$ which outputs $\hat{w}$ s.t.*

$$\tau - \frac{1}{n} |\{i | F(\hat{w}, x_i) = y'_i \wedge \mathsf{argmax}(y'_i) = L_i\}| \leq \varepsilon$$

- ***Zero-knowledge.*** *We say a ZKCMP protocol $\mathit{\Pi}$ is zero-knowledge, if for any possibly malicious PPT $\hat{\mathcal{B}}$, there exists a PPT simulator $\mathsf{Sim}_{\hat{\mathcal{B}}}$ s.t.*

$$\mathsf{Sim}_{\hat{\mathcal{B}}}(1^\lambda) \overset{c}{\approx} View_{\hat{\mathcal{B}}}(1^\lambda)$$

4 Instantiation

In this section, we give two efficient solutions to instantiate ZKCMP for trained neural networks respectively: (i) zk-SNARKs and (ii) Libra. We use the neural network in CryptoNets [8] as an example; its mathematical expression can be expressed as

$$w^{(3)}(w^{(2)}(w^{(1)}x + b^{(1)})^2 + b^{(2)})^2 + b^{(3)} = y \ \wedge\ \mathsf{argmax}(y) = L \qquad (1)$$

where $w^{(i)}, b^{(i)}\ \forall i \in [3]$ are the model parameters. Note that both zk-SNARKs and Libra works in $\mathbb{F}$, while neural networks require floating-point arithmetic. A simple solution is to scale the floating-point numbers up to integers by multiplying the same constant to all values and drop the fractional parts. A similar technique is used to reduce memory requirements in neural network predictions [14], at negligible loss of accuracy.

In Prove, $\mathcal{S}$ aims to give a NIZK proof for the statement:

$$\exists w, k, \text{s.t. } \forall i \in [\ell],\ c_i = w_i \oplus \mathsf{PRF}(k, i)\ \wedge \mathsf{COM.Verify}(E, d, k) = 1 \wedge$$
$$\frac{1}{n}|\{i \mid w^{(3)}(w^{(2)}(w^{(1)}x_i + b^{(1)})^2 + b^{(2)})^2 + b^{(3)} = y_i\ \wedge\ \mathsf{argmax}(y_i) = L_i\}| \geq \tau$$

To generate a NIZK proof, $\mathcal{S}$ constructs a circuit for the statement. The circuit consists of two components, which we describe in their specific context:

1. **Proof of accuracy**: Prove $I = \{i \mid F(w, x_i) = y_i\ \wedge\ \mathsf{argmax}(y_i) = L_i\}$ and $|I| > n \cdot \tau$;
2. **Proof of encryption**: Prove $\forall i \in [\ell]$: $ks_i \leftarrow \mathsf{PRF}(k, i)\ \wedge\ c_i := w_i \oplus ks_i$ and $(E, d) \leftarrow \mathsf{COM.Commit}(k)$;

Note that COM is used only once while PRF is frequently invoked, so we instantiate PRF with a lightweight hash function $\mathsf{MiMC7}$ and instantiate COM with $\mathsf{SHA256}$ (with the random nonce) due to Bitcoin restrictions.

In this section, we are going to mainly discuss how we construct the circuit to make Prove practical and efficient with neural networks in CryptoNets using zk-SNARKs and Libra respectively.

4.1 zk-SNARKs-based Solution

Our first approach is based on zk-SNARKs and we give a high level description of our approach here. $\mathcal{S}$ transforms the statement into the format of constraint system described in Sec. 2.3. $\mathcal{S}$ then constructs a circuit to show the correctness of every constraint equation in the constraint system. The proof size of zk-SNARKs is a small constant, which means low communication between $\mathcal{S}$ and $\mathcal{B}$. However, naively using zk-SNARKs to generate NIZK proofs for neural networks can be quite time-consuming and memory-consuming. The performance bottleneck lies in the huge computation of $w^{(3)}(w^{(2)}(w^{(1)}x_i + b^{(1)})^2 + b^{(2)})^2 + b^{(3)} = y_i$ which means enormous constraint equations need to be added into the constraint system. In addition, the computation of I is not trivial.

We conclude the main challenge for this approach in the following:

- How to reduce the computation of $w^{(3)}(w^{(2)}(w^{(1)}x_i + b^{(1)})^2 + b^{(2)})^2 + b^{(3)} = y_i$ while maintain the soundness;
- How to show the correct computation of $I = |\{i \mid w^{(3)}(w^{(2)}(w^{(1)}x_i + b^{(1)})^2 + b^{(2)})^2 + b^{(3)} = y_i\ \wedge\ \mathsf{argmax}(y_i) = L_i\}|$ and prove $|I| \geq n \cdot \tau$.

Reducing computational complexity. Generally speaking, in zk-SNARKs, all the constraint equations need to be added to a constraint system to maintain the soundness. Intuitively, we would like to check a subset of the constraint equations whereas the soundness error are still tolerated. We introduce a $\mathsf{Sampling}$ algorithm to reduce the computation.

As described in Sec. 2.3, the constraint system over $\mathbb{F}$ is denoted as $Q = ((a_j, b_j, c_j)_{j=1}^{N_g}, N)$. $\mathsf{Sampling}$ takes input as the original constraint system Q, a selecting parameter u, and it outputs a reduced constraint system Q' (cf. Fig. 4).

Now, we construct a constraint system for our example neural network and apply $\mathsf{Sampling}$ to reduce the computation. According to Eq. 1, we divide the model into four layers as follows:

```
Sampling algorithm: $Q' \leftarrow \mathsf{Sampling}(Q, u)$

Input: Original constraint system $Q = ((a_j, b_j, c_j)_{j=1}^{N_g}, l, N_v)$, selecting parameter $u$
Outputs: Reduced constraint system $Q' = ((a_j, b_j, c_j)_{j=1}^{u \cdot N_g}, l, N_v)$

Description:

  – $\mathsf{Sampling}(Q, u)$:
      • Sample a random set $T \subset [N_g]$, where $|T| = u \cdot N_g$;
      • For $k \in [u \cdot N_g]$:
          ∗ Select a random constrain equation $(a_{T_k}, b_{T_k}, c_{T_k})$ from $Q$;
          ∗ Add $(a_{T_k}, b_{T_k}, c_{T_k})$ into $Q'$
      • Output $Q' := ((a_{T_k}, b_{T_k}, c_{T_k})_{k=1}^{u \cdot N_g}, l, N_v)$.
```

Fig. 4: Sampling algorithm: $Q' \leftarrow \mathsf{Sampling}(Q, u)$

- $x^{(1)} = (w^{(1)} x + b^{(1)})^2$
- $x^{(2)} = (w^{(2)} x^{(1)} + b^{(2)})^2$
- $y = w^{(3)} x^{(2)} + b^{(3)}$
- $L = \mathsf{argmax}(y)$

We take the third layer $y = w^{(3)} x^{(2)} + b^{(3)}$, where $w^{(3)} \in \mathbb{F}^{10\times 100}, x^{(2)} \in \mathbb{F}^{100}, b^{(3)} \in \mathbb{F}^{10}$ and $y \in \mathbb{F}^{10}$ (we use specific numbers here for better explaination), as the example to show how these constraints are produced.

We first define the following variables: $(1, S, w^{(3)}, x^{(2)}, b^{(3)}, y)$, where $S = (S_{(1)}, ..., S_{(1000)}), w^{(3)} = (w^{(3)}_{(1,1)}, w^{(3)}_{(1,2)}, ..., w^{(3)}_{(10,99)}, w^{(3)}_{(10,100)}), x^{(2)} = (x^{(2)}_{(1)}, ..., x^{(2)}_{(100)}), b^{(3)} = (b^{(3)}_{(1)}, ..., b^{(3)}_{(10)}), y = (y_{(1)}, ..., y_{(100)})$. Then we rewrite $y = w^{(3)} x^{(2)} + b^{(3)}$ in the terms of two operations as follows:

- **Inner Product** (i.e. $w^{(3)} x^{(2)} = S$): It produces the constraint equations as follows:
  - For $i \in [10]$:
    - ∗ $S_{((i-1)\times 100+1)} = w^{(3)}_{(i,1)} \cdot x^{(2)}_{(1)}$;
    - ∗ $S_{((i-1)\times 100+j)} = w^{(3)}_{(i,j)} \cdot x^{(2)}_{(i)} + S_{((i-1)\times 100+j-1)},\ 1 < j \leq 100$
- **Addition** (i.e. $S + b^{(3)} = y$): It produces the constraint equations as follows:
  - For $i \in [10]$: $y_{(i)} = S_{(100\times i)} + b^{(3)}_{(i)}$;

Note that, the constraints equations we described above are produced by a single test case. Our goal is to evaluate the model on a large testing dataset, so we need $\mathsf{Sampling}$ to reduce the computation. We denote the set of constraints equations produced by test case x_i as s_i. We apply $\mathsf{Sampling}$ to each s_i to get the reduced constraint system $Q' \leftarrow \mathsf{Sampling}(Q, u)$, where $\mathsf{Sampling}(Q, u) = \mathsf{Sampling}((s_i)_{i=1}^{10000}, u)$.

Remark. In our construction, u is generally set to a small value, such as 3%. With $\mathsf{Sampling}$, the computation can be largely reduced, and we will show the soundness error is acceptable in Sec. 5.

Computing $|I|$. After computing the model output $\forall i \in [n] : y_i \in \mathbb{F}^m$, we use it to compute $|I|$. Here we present algorithms $\mathsf{VCompute}$ to compute $|I|$ and show whether $|I| \geq n \cdot \tau$ without revealing I:

- $\mathsf{VComputing}(y, L, \tau)$:

- For $i \in [n]$:
    - Find the maximum value of y_i: $y_{i,max} \leftarrow \mathsf{max}(y_i)$;
    - Construct $d_i \in \mathbb{Z}_2^m$ as follows:
        - For $k \in [m]$: set $d_{i,k} = 1$ iff $y_{i,k} \geq y_{i,max}$; otherwise, set $d_{i,k} = 0$;
    - Add the constraint equation $\sum_{k=1}^{m} d_{i,k} = 1$ to Q';
- Compute $|I| = \sum_{i=1}^{n} d_{i,L_i}$, and add it to Q';
- Output 1 iff $|I| \geq n \cdot \tau$; otherwise, output 0.

Remark. With overwhelming probability, there is a unique index j such that $y_{i,j} = y_{i,max}$. If there are more than one entry reaching the same maximum value, it means the model cannot determine the which entry is the correct output. In case of such a rare occurrence, we may choose a rule in advance (e.g. random selection) to ensure that there is only one index j such that $d_{i,j} = 1$. A similar situation may occur in the our Libra-based solution, and we take the same approach.

4.2 Libra-based Solution

We first give a high level description. $\mathcal{S}$ commits to the witness w, k at the beginning. Then $\mathcal{S}$ constructs multiple circuits which consist of two main components: (i) evaluate accuracy of the model; (ii) encrypt the model parameter $w := (w^{(1)}, w^{(2)}, w^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$. Note that, in the latter part, the model parameter $w^{(i)} \in \mathbb{F}^{n_i \times m_i}, b^{(i)} \in \mathbb{F}^{n_{i+3} \times m_{i+3}}, \forall i \in [3]$, needs to be converted into $(w_1, \ldots, w_\ell) \in \mathbb{F}^\ell$ for efficient encryption. This leads to a subtle issue: how to show the consistency between model parameter $w^{(i)}, b^{(i)}, \forall i \in [3]$, and the converted model parameter $(w_1, \ldots, w_\ell) \in \mathbb{F}^\ell$.

For this approach, our main effort is to show:

- How to show the correct computation of $|I|$ and prove $|I| \geq n \cdot \tau$;
- How to combine multiple circuits with the purpose of maintaining zero-knowledge property.

Computing $|I|$. $\mathcal{S}$ constructs three circuits as follows.

- $\mathcal{C}_1 : F(w, x) = y$, where $F(w, x) = w^{(3)}(w^{(2)}(w^{(1)}x + b^{(1)})^2 + b^{(2)})^2 + b^{(3)}, y \in \mathbb{F}^{n \times m}$.
- $\mathcal{C}_2 : \mathsf{argmax}(y) = L'$, where $L' \in \mathbb{F}^{n \times m}$.
- $\mathcal{C}_3 : \mathsf{Judge}(L', L, \tau) = r$, where $r \in \mathbb{F}$.

Unlike Sec. 4.1, we present the ground-truth label L in the form of one-hot vectors.

After computing $y \leftarrow w_3(w_2(w_1 x + b_1)^2 + b_2)^2 + b_3$ using $\mathcal{C}_1$, $\mathcal{S}$ computes the predicted label L' in the following way:

- $\mathsf{argmax}(y)$:
    - For $i \in [n]$:
        - Find the maximum value of y_i: $y_{i,max} \leftarrow \mathsf{max}(y_i)$;
        - Compute $y_{i,mid} \leftarrow y_{i,max} - 1$;
        - Compute $L'_i \leftarrow relu(y_i - y_{i,mid} \cdot \vec{1}_m)$, where $relu(x) = \mathsf{max}(0, x), \vec{1}_m \in \mathbb{F}^m$;
    - Output the predicted label $L' := (L'_1, \ldots, L'_n)$, where $L' \in \mathbb{F}^{n \times m}$.

With overwhelming probability, there is only one entry is non-zero (i.e. 1) in L'_i. If not, approach desribed in Sec. 4.1 will be taken. $\mathcal{S}$ can prove $L' \leftarrow \mathsf{argmax}(y)$ by $\pi_2 \leftarrow \mathsf{Libra.Prove}(L', y, \mathcal{C}_2)$, and $\mathcal{B}$ can validate it by $b \leftarrow \mathsf{Libra.Verify}(\pi_2)$. The construction of Libra is presented in Fig. 1. Then $\mathcal{S}$ can judge whether the model accuracy exceeds the threshold τ:

- $\mathsf{Judge}(L', L, \tau)$:
  - Compute $\delta \leftarrow relu(L' - L)$;
  - Compute $|\delta| \leftarrow \vec{1}_n^T \cdot \delta \cdot \vec{1}_m$, where $\vec{1}_n \in \mathbb{F}^n$ and $\vec{1}_m \in \mathbb{F}^m$;
  - Compute $|I| \leftarrow n - |\delta|$;
  - Output $r \leftarrow relu(n \cdot \tau - |I|)$.

If $r = 0$, it means that $|I| \geq n \cdot \tau$, that is, this model meets $\mathcal{B}$'s requirement. $r \leftarrow \mathsf{Judge}(L', L, \tau)$ can be proved by $\pi_3 \leftarrow \mathsf{Libra.Prove}((L, \tau), L', \mathcal{C}_3)$.

Converting w. For efficient encryption, the model parameter $w^{(i)} \in \mathbb{F}^{n_i \times m_i}, b^{(i)} \in \mathbb{F}^{n_{i+3} \times m_{i+3}}, \forall i \in [3]$, needs to be converted into $(w_1, \dots, w_\ell) \in \mathbb{F}^\ell$. Without loss of generality, we resize $w^{(i)}$ and $b^{(i)}$ into matrix with the same number of columns denoted as m'. Then $\mathcal{S}$ can convert w as follows:

- $\mathsf{Convert}(w)$:
  - Resize the model parameter into the same column m': $\forall i \in [3], w^{(i)} \in \mathbb{F}^{\ell_i \times m'}, b^{(i)} \in \mathbb{F}^{\ell_{i+3} \times m'}$, and set $\ell := \sum_{i=1}^{6} \ell_i, \ell_0 := 0$;
  - Set $B := (\beta, \beta^2, \dots, \beta^{m'})^T$, where β is a public variable to control the precision of w;
  - Compute $\forall i \in [3]$, $w^{(i)} \cdot B = (w_{l_{i-1}+1}, \cdots, w_{l_i})^T, b^{(i)} \cdot B = (w_{l_{i+2}+1}, \cdots, w_{l_{i+3}})^T$;
  - Output $(w_1, \dots, w_\ell) \in \mathbb{F}^\ell$.

$(w_1, \dots, w_\ell) \leftarrow \mathsf{Convert}(w)$ can be proved by $\pi_4 \leftarrow \mathsf{Libra.Prove}((w_1, \dots, w_\ell), w, \mathcal{C}_4)$. Note that, in the above algorithm, we assume there exists m' such that $m' | n_i m_i, \forall i \in [6]$. If the original model parameter does not have enough entries to fill $\ell_i m'$ entries of the resized version, we just pad 0.

Putting the Pieces Together. In order to prove the whole process, $\mathcal{S}$ constructs multiple circuits as follows:

- Component 1: proof of accuracy
  - $\mathcal{C}_1 : F(w, x) = y$.
  - $\mathcal{C}_2 : \mathsf{argmax}(y) = L'$.
  - $\mathcal{C}_3 : \mathsf{Judge}(L', L, \tau) = r$.
- Component 2: proof of encryption
  - $\mathcal{C}_4 : \mathsf{Convert}(w) = ws$, where $ws := (w_1, \dots, w_\ell)$.
  - $\mathcal{C}_5 : \forall i \in [\ell], \mathsf{PRF}(k, i) = ks_i$.
  - $\mathcal{C}_6 : \mathsf{COM.Commit}(k) = (E, d)$.
  - $\mathcal{C}_7 : \forall i \in [\ell], w_i \oplus ks_i = c_i$.

These circuits are connected in the way shown in Fig. 5.

However, the output of a circuit may be the input of another which should be kept private. Our solution is to use zkVPD scheme (cf. Appendix A) to connect the circuits.

Take the component 2 as an example. At the beginning, the prover $\mathcal{P}$ computes $\overline{w}, \overline{k}, \overline{ws}$ and $\overline{ks}$ defined as Eq. 2 and commits to them using $\mathsf{zkVPD.Commit}$. After the verifier $\mathcal{V}$ receiving a claim about the output of $\mathcal{C}_7$, that is c, she computes $\overline{c}$ and evaluates it on u where u is randomly selected. Then $\mathcal{P}$ and $\mathcal{V}$ will reduce $\overline{c}(u)$ layer by layer recursively until it reaches the input layer. At the end, $\mathcal{V}$ queries the evaluations of $\overline{ws}(p), \overline{ks}(q)$ using $\mathsf{zkVPD.Open}$ where p and q are randomly selected by $\mathcal{V}$, and validates them by $\mathsf{zkVPD.Verify}$. If $\mathsf{zkVPD.Verify}$ outputs 1, $\mathcal{P}$ and $\mathcal{V}$ continue to deal with $\mathcal{C}_4$ and $\mathcal{C}_5$. For $\mathcal{C}_4$, $\mathcal{V}$ uses $\overline{ws}(p)$ as the start point instead of evaluating on a random point. Then they will reduce $\overline{ws}(p)$ layer by layer to complete the protocol. For $\mathcal{C}_5$, the similar approaches will be applied for $\overline{ks}(q)$.

The construction above can be made non-interactive in the random oracle model by applying Fiat-Shamir heuristic [10]. The non-interactive version of Libra based on a 254-bit prime field can provide a security level of 100+ bits [17].

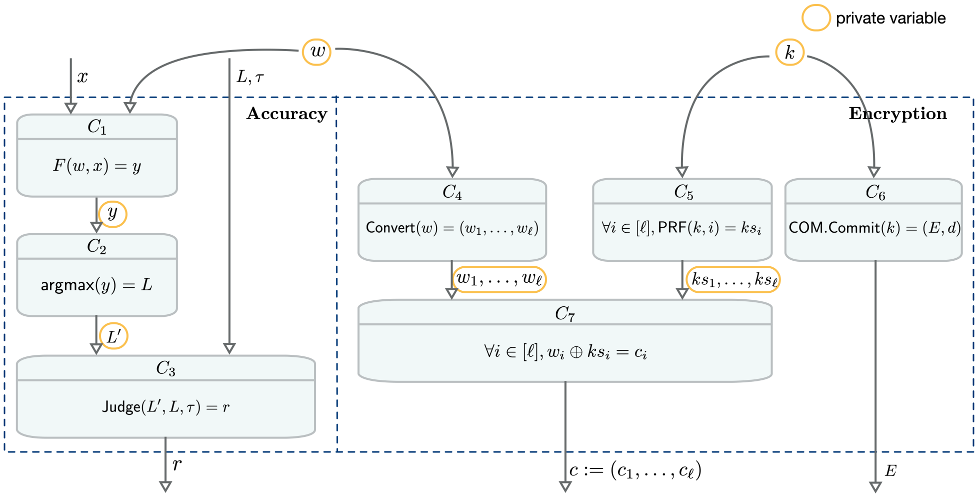

private variable
w
k
x
L, τ
Accuracy
Encryption
C_1
$F(w, x) = y$
y
C_2
$\mathsf{argmax}(y) = L$
L'
C_3
$\mathsf{Judge}(L', L, \tau) = r$
r
C_4
$\mathsf{Convert}(w) = (w_1, \ldots, w_\ell)$
$w_1, \ldots, w_\ell$
C_5
$\forall i \in [\ell], \mathsf{PRF}(k, i) = ks_i$
$ks_1, \ldots, ks_\ell$
C_6
$\mathsf{COM.Commit}(k) = (E, d)$
C_7
$\forall i \in [\ell], w_i \oplus ks_i = c_i$
$c := (c_1, \ldots, c_\ell)$
E

Fig. 5: The circuits designed for Libra

5 Security Analysis

Security analysis of $\mathsf{Sampling}$**.** We now examine the soundness of our reduced constraint system Q'. As defined in Sec. 3, if the system is ε-sound with accuracy τ, then the real model accuracy is bounded by $(\tau - \epsilon, 1]$. Recall the proof has two components: (i) show the prover knows a set of model parameters w such that $(c, E, d) \leftarrow \mathsf{Seal}(w)$ and (ii) show the prover knows a set of model parameters w' such that

$$|\{i \,|\, F(w', x_i) = y_i \;\wedge\; argmax(y_i) = L_i\}| \geq n \cdot \tau \quad .$$

In general, an adversary may perform the following two types of attacks:

- **Inconsistent model parameters attack.** In this attack, the adversary try to use inconsistent $w \neq w'$ between the encrypted model and the testing model to produce a valid proof, whereas $F(w', x_i)$ is correctly computed.
- **Model execution tampering attack.** In this attack, the adversary try to tamper the model execution $F(w', x_i) = y_i$ by modifying the intermediate variable values during the computation.

Lemma 1 *Let u be the selecting parameter, n be the testing dataset size. Denote $\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},Sound}(\lambda)$ as the soundness advantage of the underlying NIZK proof system. The probability that any PPT adversary can success with the inconsistent model parameters attack is*

$$\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},Sound}(\lambda) + (1 - \mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},Sound}(\lambda)) \cdot (1 - u)^n.$$

Proof. First, the adversary could break the soundness of underlying NIZK system to produce a valid proof, and the probability is bounded by $\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},Sound}(\lambda)$. Assume the adversary cannot break the soundness of the underlying NIZK system, we give the following analysis.

Every parameter in w will be checked at least once in constraint equations set s_i. Since we select $N'_g \cdot u$ constraint equations from s_i to check, the probability for adversary to escape from capturing in a single test case is $(1 - u)$. With n test cases, the probability is $(1 - u)^n$. □

Remark. For CryptoNets, when $n = 10000$, $u = 3\%$, the cheating prob. is bounded by 0.01%.

Lemma 2 *Let u be the selecting parameter, n be the testing dataset size. Denote $\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},Sound}(\lambda)$ as the soundness advantage of the underlying NIZK proof system. The probability that any PPT adversary can success model execution tampering attack is*

$$\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},Sound}(\lambda) + (1 - \mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},Sound}(\lambda)) \cdot (1-u)^{n\cdot\varepsilon}.$$

Proof. Here we adapt the weakest assumption: adversary only needs to change one intermediate variable value to influence accuracy. Denote there are n test cases in $\mathcal{D}$ and adversary changes M test cases to influence accuracy, and we have $\varepsilon = \frac{M}{n}$.

Denote the probability that the adversary escapes from capturing is p. In our assumption, there is one constraint can not be satisfied in each constraints set produced by M test cases. Similar to the proof of Lemma 1, the probability p is $(1-u)^M$

Replace M with $n \cdot \varepsilon$, the equation $p = (1-u)^{n\cdot\varepsilon}$ holds, which is equivalent to $\varepsilon = \frac{log(p)}{n \cdot log(1-u)}$. □

Remark. As shown in Fig. 6, when $n = 10000$, if the prover deviates the model accuracy from τ for 1% (a.k.a. 1%-soundness), she will be caught with at least 95% probability.

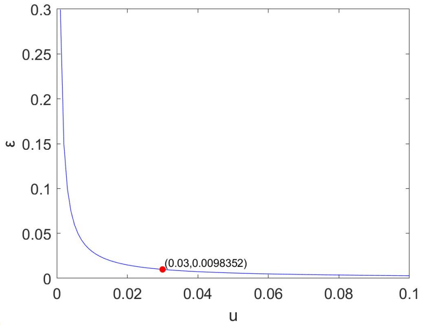


Fig. 6: Fix $p = 0.95$ and $n = 10000$, the relation between ε and u.

Security analysis of main construction. In this section, we examine the security of our construction. Intuitively, the security of our ZKCMP protocol largely depends on the soundness and zero-knowledge properties of underlying NIZK proofs. We also assume COM and PRF are a secure commitment scheme and secure pseudorandom function, respectively. More formally, we prove the security of our construction by the following theorem, and its proof is provided in Appendix B.

Theorem 1 *Let $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \mapsto \{0,1\}^{\mu(\lambda)}$ be a secure pseudorandom function, and $\mathsf{COM} : \{0,1\}^* \mapsto \{0,1\}^\lambda$ be a commitment scheme. The protocol described in Sec. 3 with $\mathcal{F}_{ex}[\mathsf{COM}]$ as depicted in Fig. 2 is perfect complete, 0-sound, and computational zero-knowledge if the underlying NIZK protocol is perfect complete, computational sound, and computational zero-knowledge.*

6 Implementation and Experiments

We fully implement the proposed instantiations and evaluate the neural network described in CryptoNets [8] on MNIST dataset. All experiments are conducted on the same machine that has 80 Intel Xeon E5-2680 v4 vCPUs@2.5 GHz with 700GB RAM and is running on Ubuntu.

Efficiency of Sampling. First of all, we conduct experiments to demonstrate the efficiency of our proposed Sampling technique. Fixing selecting parameter $u = 3\%$, the results are shown in Table 1 and Fig. 7. The results show that this technique has largely reduced the cost of proof generation.

Performance of our solutions. Since Prove and Verify account for main cost, we focus on describing our experiment results on them. We compare our instantiations in terms of Prover Time, Verifier Time and Proof Size. Fixing selecting parameter $u = 3\%$, we compare the zk-SNARKs based solution and Libra based solution, the experiment results are shown in Table 2 and Fig. 8. Evaluating 10000 testing cases, zk-SNARKs based solution has much less communication volume which is a constant (1019 bits) while Prover Time (5781.95s) and Verifier Time (18.5781s) is acceptable. For Libra based solution, although more communication volume is needed (104.4257KB), it costs much less in both Prover Time (1034.1239s) and Verifier Time (0.4413s).

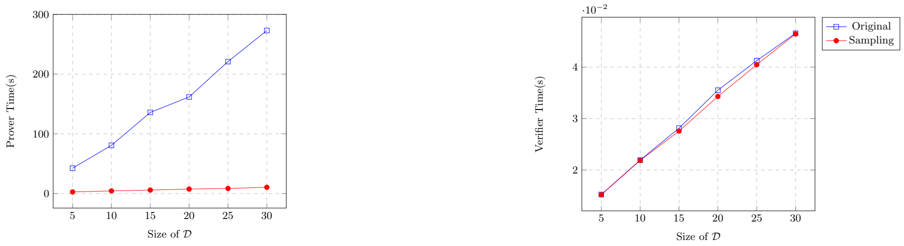
Remark. Our solutions can be applied to different application scenarios. In a scenario where communication volume matters, zk-SNARKs based solution is more suitable. If more emphasis are put on time consumption, Libra based solution may be adopted.

Table 1: Performance of Sampling

Image number		5	10	15	20	25	30
Prover Time(s)	Original	42.4675	80.6847	135.829	161.769	220.973	272.984
	Sampling	2.72504	4.43177	5.82092	7.50026	8.54351	10.4192
Verifier Time(s)	Original	0.01525	0.021936	0.028138	0.035503	0.041257	0.054159
	Sampling	0.0152	0.021861	0.027569	0.034297	0.040474	0.049441

Table 2: Performance of solutions

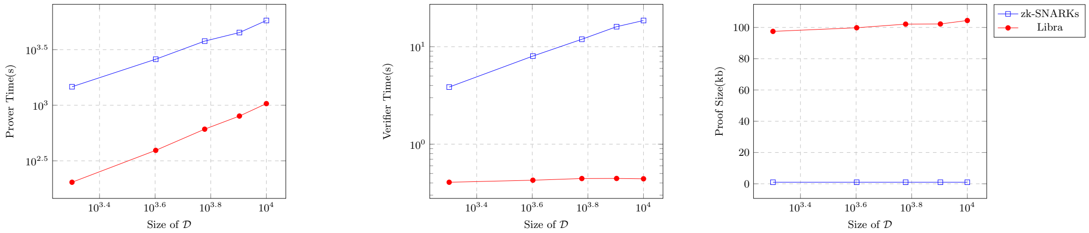
Image number		0	2000	4000	6000	8000	10000
Prover Time(s)	zk-SNARKs	464.36	1466.78	2593.56	3773.49	4503.66	5781.95
	Libra	11.8639	202.9769	392.7659	609.2089	799.3409	1034.1239
Verifier Time(s)	zk-SNARKs	0.0126	3.8566	8.0160	11.9275	16.0533	18.5781
	Libra	0.1546	0.4067	0.4268	0.4440	0.4449	0.4413
Proof Size(KB)	zk-SNARKs	0.9951	0.9951	0.9951	0.9951	0.9951	0.9951
	Libra	50.1967	97.4724	99.7902	102.1080	102.2080	104.4257



Prover Time

Verifier Time

Fig. 7: Performance of Sampling



Prover Time

Verifier Time

Proof Size

Fig. 8: Performance of our instantiations

7 Related Work

SafetyNets [11] is a framework that enables a computationally weak client to outsource neural network inferences to an untrusted server (cloud), and allows the server to prove the correctness of the inference results using interactive proof. In this scenario, clients knows both the model and the input data, whereas in our case both the model and the input data has to be committed to the verifiers. Slalom [16] is a framework that allows neural network evaluations inside trusted execution environments (TEEs), and the matrix multiplication layers are outsourced to an untrusted GPU without compromising integrity or privacy. They use a lightweight way for verification but it is limited to linear layers and requires TEEs. In [19], Zhao et al. proposed to use zk-SNARKs naively to valid neural network prediction. The prover commits to the value of all intermediate layers, and the verifier validates one random layer with a zk-SNARKs proof. This scheme neither provide negligible soundness nor support validation for large testing dataset.

8 Conclusion

In this paper, we address the problem of fair model exchange by proposing a new concept called Zero Knowledge Contingent Model Payment (ZKCMP). We investigate two state-of-the-art NIZK proofs: zk-SNARKs and Libra, and use them as the main building block to instantiate our ZKCMP protocol respectively. In particular, we propose a random sampling technique to improve the efficiency of zk-SNARKs. Therefore, our proposal is able to support validation for large testing dataset. To demonstrate the practicality of our proposal, we have conducted extensive experiments.

References

1. The fisrt successful zero-knowledge contingent payment. https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/ (2016)
2. Hashlock. https://en.bitcoin.it/wiki/Hashlock (2016)
3. Angelini, E., di Tollo, G., Roli, A.: A neural network approach for credit risk evaluation. Quarterly Review of Economics and Finance **48**(4), 733–755 (2008). https://doi.org/10.1016/j.qref.2007.04.001
4. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for c: Verifying program executions succinctly and in zero knowledge. In: CRYPTO 2013. pp. 90–108. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
5. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg (2006)
6. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS 2012. p. 326–349. Association for Computing Machinery, New York, NY, USA (2012)
7. Campanelli, M., Gennaro, R., Goldfeder, S., Nizzardo, L.: Zero-knowledge contingent payments revisited: Attacks and payments for services. In: CCS 2017. pp. 229–243 (2017)
8. Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. Tech. rep. (February 2016)
9. Fakoor, R., Ladhak, F., Nazi, A., Huber, M.: Using deep learning to enhance cancer diagnosis and classification. In: Proceedings of the international conference on machine learning. vol. 28. ACM New York, USA (2013)
10. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
11. Ghodsi, Z., Gu, T., Garg, S.: Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In: Advances in Neural Information Processing Systems 30, pp. 4672–4681. Curran Associates, Inc. (2017)
12. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. Journal of the Acm **62**(4), 1–64 (2015)
13. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on computing **18**(1), 186–208 (1989)
14. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: CCS 2017. p. 619–631. Association for Computing Machinery, New York, NY, USA (2017)
15. Lund, C., Fortnow, L., Karloff, H.: Algebraic methods for interactive proof systems. Journal of the ACM **39**(4) (1999)
16. Tramer, F., Boneh, D.: Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. arXiv preprint arXiv:1806.03287 (2018)
17. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: CRYPTO 2019. pp. 733–764. Springer (2019)
18. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 863–880. IEEE (2017)

19. Zhao, L., Wang, Q., Wang, C., Li, Q., Shen, C., Lin, X., Hu, S., Du, M.: Veriml: Enabling integrity assurances and fair payments for machine learning as a service. arXiv preprint arXiv:1909.06961 (2019)

A The Main Building Blocks of Libra

zkVPD scheme. Zero-knowledge verifiable polynomial delegation scheme (zkVPD) [18] allows a verifier to delegate the computation of polynomial evaluations to a powerful prover without leaking any sensitive information, and validates the result in time that is constant or logarithmic to the size of the polynomial. Let $\mathcal{F}$ be a family of l-variate polynomial over $\mathbb{F}$. A zkVPD for $f \in \mathcal{F}$ and $\mathbf{t} \in \mathbb{F}^l$ consists of the following algorithms:

- $(pp, vp) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $com \leftarrow \mathsf{Commit}(f, r_f, pp)$
- $\{0,1\} \leftarrow \mathsf{Check}(com, vp)$
- $(y, \pi) \leftarrow \mathsf{Open}(f, t, r_f, pp)$
- $\{0,1\} \leftarrow \mathsf{Verify}(com, t, y, \pi, vp)$

GKR protocol. Using sumcheck protocol [15] as a main building block, Goldwasser *et al.* [12] constructed an interactive protocol for layered arithmetic circuits with size C and depth d. Formally speaking, we denote the number of gates in the i-th layer as C_i and let $c_i = \lceil \log_2 S_i \rceil$. We then define a function $V_i : \{0,1\}^{c_i} \to \mathbb{F}$ that takes a binary string $b \in \{0,1\}^{c_i}$ as input and returns the output of gate b in layer i, where b is called the gate label. Therefore, V_0 corresponds to the output of the circuit and V_D corresponds to the input. Then we extend V_i to its *multilinear extension.*

Definition 5 (Multi-linear Extension) *Let $V : \{0,1\}^l \to \mathbb{F}$ be a function. The multilinear extension of V is the unique polynomial $\widetilde{V} : \mathbb{F}^l \to \mathbb{F}$ such that $\widetilde{V}(x_1, x_2, \ldots, x_l) = V(x_1, x_2, \ldots, x_l)$ for all $(x_1, x_2, \ldots, x_l) \in \{0,1\}^l$. $\widetilde{V}$ can be expressed as:*

$$\widetilde{V}(x_1, x_2, \ldots, x_l) = \sum_{b \in \{0,1\}}^{l} \prod_{i=1}^{l} ((1 - x_i)(1 - b_i) + x_i b_i) \cdot V(b),$$

where b_i is i-th bit of b.

To ensure zero knowledge, $\mathcal{P}$ masks the polynomial $\widetilde{V}_i$ and the sumcheck protocol by adding random polynomials. In particular, for layer i, $\mathcal{P}$ selects a random bivariate polynomial $R_i(x_1, z)$ and defines

$$\overline{V}_i(x_1, \ldots, x_{c_i}) = \widetilde{V}_i(x_1, \ldots, x_{c_i}) + Z_i(x_1, \ldots, x_{c_i}) \cdot \sum_{z \in \{0,1\}} R_i(x_1, z), \tag{2}$$

where $Z_i(x) = \prod_{i=1}^{c_i} x_i(1 - x_i)$, so $Z_i(x) = 0, \forall x \in \{0,1\}^{c_i}$. Since R_i is randomly selected, revealing evaluations of $\overline{V}_i$ does not leak information about $\widetilde{V}_i$. A random polynomial $\delta_i(x, y, z)$ is also selected to mask the sumcheck protocol. In this way, the sumcheck protocol will not leak information and thus be zero knowledge. See more details in [17].

B Proof of Theorem 1

Proof. For perfect completeness, since the underlying NIZK is perfect complete, it is straightforward that the verification Verify would return 1, and $\mathcal{F}_{ex}$ guarantees that the buyer $\mathcal{B}$ will recive k when the event $\mathcal{E}_m$ occurs.

For 0-soundness, the event $\mathcal{E}_v$ occurs when the potentially malicious seller $\hat{\mathcal{S}}$ produces an accepting proof π and submits (Redeem, k, d) to $\mathcal{F}_{ex}[\mathsf{COM}]$; By the soundness of the underlying NIZK protocol, with overwhelming probability, the model parameter $w := (w_1, \ldots, w_\ell)$ can satisfy $|\{i \mid F(w, x_i) = y_i \ \wedge \ \mathsf{argmax}(y_i) = L_i\}| \geq n \cdot \tau$, where $\forall i \in [\ell] : w_i = c_i \oplus \mathsf{PRF}(k, i) \ \wedge$ $\mathsf{COM.Verify}(E, d, k) = 1$. Moreover, due to the binding property of the commitment scheme COM, k cannot be changed afterwards. Therefore, we can construct an extractor $\mathsf{Ext}_{\hat{\mathcal{S}}}$ that takes input as $\{c_i\}_{i\in[\ell]}$ and k from the out-going messages of $\hat{\mathcal{S}}$, and outputs the model as $w_i = c_i \oplus \mathsf{PRF}(k, i)$.

For computational zero-knowledge, we first construct a simulator Sim works as follows.

- During Setup:
  - Invoke $(\mathsf{crs}^*, \mathsf{td}) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$;
  - Output $pp := \mathsf{crs}^*$;
- During Seal:
  - Pick a random key $k^* \leftarrow \{0,1\}^\lambda$;
  - Compute $(E^*, d^*) \leftarrow \mathsf{COM.Commit}(k^*)$;
  - For $i \in [\ell]$, compute $c_i^* \leftarrow \{0,1\}^{\mu(\lambda)}$;
  - Output $(c^* := (c_1^*, \ldots, c_\ell^*), E^*)$;
- During Prove:
  - Invoke $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(pp, (c^*, E^*, \mathcal{D}, \tau), \mathsf{td})$;
  - Output π^*;

Lemma 3 *The adversary's view output by the simulator* Sim *as described above is indistinguishable from the real view with advantage*

$$\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},ZK}(1^\lambda) + \mathsf{Adv}_{\mathsf{COM}}^{\mathcal{A},Hide}(1^\lambda) + \ell \cdot \mathsf{Adv}_{\mathsf{PRF}}^{\mathcal{A}}(1^\lambda).$$

Proof. We prove Lemma 3 by the sequence of hybrids $\mathcal{H}_0, \ldots, \mathcal{H}_3$ as follows.

Hybrid $\mathcal{H}_0$: it is the real view.

Hybrid $\mathcal{H}_1$: it is the same as Hybrid $\mathcal{H}_0$, except during Setup, $\mathsf{NIZK.Sim}_1(1^\lambda)$ is used to generate the simulated CRS crs^*; during Prove, π^* is generated by $\mathsf{NIZK.Sim}_2(pp, (c, E, \mathcal{D}, \tau), \mathsf{td})$ instead of the real proof.

Claim 1 *If the underlying NIZK proof system is computationally zero-knowledge with advantage* $\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},ZK}(1^\lambda)$*, then the view of Hybrid* $\mathcal{H}_1$ *is indistinguishable from the view of Hybrid* $\mathcal{H}_0$ *with distinguishing advantage* $\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},ZK}(1^\lambda)$.

Proof. By definition 2, it is straightforward that if an adversary $\mathcal{A}$ can distinguish $\mathcal{H}_1$ from $\mathcal{H}_0$ with advantage $\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},ZK}(1^\lambda)$, then $\mathcal{A}$ can break the zero-knowledge property of the underlying NIZK proof system with the same advantage. □

Hybrid $\mathcal{H}_2$: it is the same as Hybrid $\mathcal{H}_1$, except during Seal, replace E^* as $\mathsf{COM.Commit}(k^*)$ instead of $\mathsf{COM.Commit}(k)$.

Claim 2 *If the distinguishing advantage of the* COM *hiding property is* $\mathsf{Adv}_{\mathsf{COM}}^{\mathcal{A},Hide}(1^\lambda)$*, then the view of Hybrid* $\mathcal{H}_2$ *is indistinguishable from the view of Hybrid* $\mathcal{H}_1$ *with distinguishing advantage* $\mathsf{Adv}_{\mathsf{COM}}^{\mathcal{A},Hide}(1^\lambda)$.

Proof. It is straightforward by direct reduction. □

Hybrid $\mathcal{H}_3$: it is the same as Hybrid $\mathcal{H}_2$, except during Seal, for $i \in [\ell]$, replace c_i^* as $\{0,1\}^{\mu(\lambda)}$ instead of $w_i \oplus \mathsf{PRF}(k,i)$.

Claim 3 *If the distinguishing advantage of* PRF *is* $\mathsf{Adv}_{\mathsf{PRF}}^{\mathcal{A}}(1^\lambda)$*, then the view of Hybrid* $\mathcal{H}_3$ *is indistinguishable from the view of Hybrid* $\mathcal{H}_2$ *with distinguishing advantage* $\ell \cdot \mathsf{Adv}_{\mathsf{PRF}}^{\mathcal{A}}(1^\lambda)$.

Proof. First of all, the distribution of $D_i := c_i^* \oplus w_i$ is the uniformly random. Since the distinguishing advantage of D_i and $\mathsf{PRF}(k,i)$ is bounded by the advantage of PRF $\mathsf{Adv}_{\mathsf{PRF}}^{\mathcal{A}}(1^\lambda)$, by hybrid argument, the overall distinguishing advantage of $\mathcal{H}_3$ and $\mathcal{H}_2$ is bounded by $\ell \cdot \mathsf{Adv}_{\mathsf{PRF}}^{\mathcal{A}}(1^\lambda)$. □

Hybrid $\mathcal{H}_3$ is the simulated view; therefore, the overall distinguishing advantage is $\mathsf{Adv}_{\mathsf{NIZK}}^{\mathcal{A},ZK}(1^\lambda) + \mathsf{Adv}_{\mathsf{COM}}^{\mathcal{A},Hide}(1^\lambda) + \ell \cdot \mathsf{Adv}_{\mathsf{PRF}}^{\mathcal{A}}(1^\lambda)$. □

This concludes the proof. □