ZKCPlus: Optimized Fair-exchange Protocol Supporting Practical and Flexible Data Exchange

Yun Li
Tsinghua University
liyun19@mails.tsinghua.edu.cn

Cun Ye
K-Orange/SECBIT Labs
cun.ye@secbit.io

Yuguang Hu
SECBIT Labs
yuguang.hu@outlook.com

Ivring Morpheus
SECBIT Labs
ivring.morpheus@gmail.com

Yu Guo
SECBIT Labs
yu.guo@secbit.io

Chao Zhang✉
Tsinghua University
chaoz@tsinghua.edu.cn

Yupeng Zhang
Texas A&M University
zhangyp@tamu.edu

Zhipeng Sun
SECBIT Labs
zp@secbit.io

Yiwen Lu
SECBIT Labs
even@secbit.io

Haodi Wang
Beijing Normal University
whd@mail.bnu.edu.cn

Abstract

Devising a fair-exchange protocol for digital goods has been an appealing line of research in the past decades. The Zero-Knowledge Contingent Payment (ZKCP) protocol first achieves fair exchange in a trustless manner with the aid of the Bitcoin network and zero-knowledge proofs. However, it incurs setup issues and substantial proving overhead, and has difficulties handling complicated validation of large-scale data.

In this paper, we propose an improved solution ZKCPlus for practical and flexible fair exchange. ZKCPlus incorporates a new commit-and-prove non-interactive zero-knowledge (CP-NIZK) argument of knowledge under standard discrete logarithmic assumption, which is prover-efficient for data-parallel computations. With this argument we avoid the setup issues of ZKCP and reduce seller's proving overhead, more importantly enable the protocol to handle complicated data validations.

We have implemented a prototype of ZKCPlus and built several applications atop it. We rework a ZKCP's classic application of trading sudoku solutions, and ZKCPlus achieves 21-67× improvement in seller efficiency than ZKCP, with only milliseconds of setup time and 1 MB public parameters. In particular, our CP-NIZK argument shows an order of magnitude higher proving efficiency than the zkSNARK adopted by ZKCP. We also built a realistic application of trading trained CNN models. For a 3-layer CNN containing 8,620 parameters, it takes less than 1 second to prove and verify an inference computation, and also about 1 second to deliver the parameters, which is very promising for practical use.

CCS Concepts

• **Security and privacy** → **Cryptography**.

Keywords

fair exchange; zero-knowledge argument; commit-and-prove

ACM Reference Format:
Yun Li, Cun Ye, Yuguang Hu, Ivring Morpheus, Yu Guo, Chao Zhang, Yupeng Zhang, Zhipeng Sun, Yiwen Lu, and Haodi Wang. 2021. ZKCPlus: Optimized Fair-exchange Protocol Supporting Practical and Flexible Data Exchange . In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3460120.3484558

1 Introduction

Building a fair protocol for exchanging digital goods has been studied for a long time. It has been proved that strong fairness is unattainable without the aid of trusted third parties [45]. In practice these trusted parties are often centralized organizations, which bring them all kinds of risks, including single point of failure [22], user information misuse [20], etc.

The advent of blockchain offers new possibilities. In 2011, Gregory Maxwell proposed the first blockchain-based fair exchange solution, Zero-Knowledge Contingent Payment (ZKCP) [43]. By incorporation of Bitcoin [44] network and zero-knowledge proofs, it is able to achieve fair exchange of digital goods and currencies in one atomic swap. In ZKCP, a buyer $\mathcal{B}$ wants to purchase from a seller $\mathcal{S}$ a piece of digital good x which satisfies a particular predicate $\phi(x) = 1$. To this end, $\mathcal{S}$ first chooses a secret key k to encrypt the data x using an encryption scheme Enc, i.e., $z \leftarrow \mathsf{Enc}_k(x)$. Next, $\mathcal{S}$ constructs a "hash lock" $h = \mathsf{H}(k)$ with a cryptographic hash

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484558

Table 1: Performance of ZKCP on a 16×16 sudoku solution

Size of proving key		68 MB
Proving time		10-20 s
Verification time		40 ms
Proof size		288 B
Circuit cost	Encryption	86.81%
	SHA256 hash	10.23%
	Solution validity	4.42%

function H, and a zero-knowledge proof π for the NP statement

$$\mathsf{Enc}_k(x) = z \ \wedge \ \phi(x) = 1 \ \wedge \ \mathsf{H}(k) = h.$$

Then $\mathcal{S}$ delivers (z, h, π) to $\mathcal{B}$. On verifying the proof, $\mathcal{B}$ submits a hash-locked payment to the blockchain, saying that only the one who discloses a correct key w.r.t. the hash h can redeem this payment. $\mathcal{S}$ then reveals k and thus gets the payment from the blockchain; $\mathcal{B}$ decrypts the ciphertext with k and recovers x. The blockchain acts as an impartial arbiter, and the exchange ultimately happens in $\mathcal{S}$'s atomic action of revealing k.

As a seminal work, ZKCP demonstrates the great potential of blockchains in replacing traditional trusted third parties, but it also has some notable limitations. In 2016, the protocol was first implemented for trading solutions to sudoku puzzles. The implementation adopts a simple stream cipher from SHA256, and uses the Pinocchio/BCTV14 zkSNARK scheme [7, 46] for the underlying zero-knowledge proofs. Table 1 presents the performance statistics on a 16×16 sudoku solution from the ZKCP developers [8]. We have three main observations from the statistics.

First, *the trusted setup of the zkSNARK incurs security issues and high overhead.* The Pinocchio/BCTV14 zkSNARK requires a trusted party to generate a common reference string (CRS), which contradicts ZKCP's very goal of eliminating trusted third parties. As a circumvent, ZKCP entrusts the buyer to produce the CRS, but this is proved to be insecure [13, 25] as it compromises the zero-knowledge property. Moreover, the CRS is cumbersome that even for a 16×16 sudoku solution which costs only 256 bytes, the size of the proving key in the CRS reaches 68 MB.

Second, *the system's performance is limited by the seller's proving procedure.* With Pinocchio/BCTV14 zkSNARK, for trading a 16×16 sudoku solution the verification time is only 40 ms, but the proving time gets to 10-20 seconds. The bottleneck of the whole system lies on the prover (seller) side. Besides, the proof succinctness is not very helpful in ZKCP context, given that the sub-linear-size proof π is transferred off chain along with the linear-size ciphertext z. To improve the overall performance of ZKCP, it's critical to cut down the seller's proving overhead.

Third, *it has difficulties validating complicated predicates on large-scale data.* In trading sudoku solutions, the predicate for solution validity is very simple (as its related constraints amount to only 4.42% of the whole circuit, while the rest are for encryption and hash lock). However in practice, predicates can be arbitrarily complicated, possibly involving large-scale data-parallel computations or interactive decision-making processes. A typical example is to trade a trained neural network model in a Machine-Learning-as-a-Service (MLaaS) scenario, where the predicate is that the model yields sufficiently high accuracy on a test set chosen by the buyer, which translates into multiple rounds of variable inference processes. As previous work [41] evaluated, with Groth16 zkSNARK [32] (an optimization of Pinocchio/BCTV14 zkSNARK), the prover takes 45 minutes to prove for a single inference process of a 5-layer convolutional neural network (CNN) LeNet-5 [39] with the setup time up to 1.5 hours and CRS size up to 11 GB. Clearly it is impractical for ZKCP to handle such predicates.

In this work we propose `ZKCPlus`, an extended version of ZKCP which supports practical and flexible fair exchange of large-scale digital goods. We make several major modifications to ZKCP to address the aforementioned problems. Specifically, we replace the Pinocchio/BCTV14 zkSNARK of ZKCP with prover-efficient *commit-and-prove non-interactive zero-knowledge* (CP-NIZK) arguments of knowledge, and design a CP-NIZK argument scheme optimized for data-parallel computations based on the standard discrete logarithmic assumption. It replaces the costly and insecure trusted setup of ZKCP with a lightweight public setup phase, and by incorporation of a circuit-friendly block cipher in a data-parallel encryption mode (i.e., counter mode), it greatly reduces seller's proving overhead. With the commit-and-prove feature, we can further reduce proving overhead by replacing the hash lock of ZKCP with a commitment lock on the blockchains supporting commitment opening like Ethereum [54] More importantly, utilizing CP-NIZK arguments' powerful capability of composition [12, 14, 16], `ZKCPlus` can handle excessively complicated predicates, especially those involving large-scale data-parallel computations.

Our CP-NIZK argument for data-parallel computations builds on the fact that all parallelized sub-circuits have exactly identical rank-1 constraint systems, and thus the satisfiability of the whole circuit can be reduced to a more concise relation which can be handled with the techniques in [11, 31]. Concerning prover efficiency, our construction is several times faster than Groth16 zkSNARK [32], and comparable to the non-succinct version of Spartan$_{\mathrm{DL}}$ [49].

The `ZKCPlus` protocol supports a broad class of applications. Besides the toy application of "pay to sudoku solution" demonstrated by ZKCP, it also applies to more realistic scenarios, including (but not limited to) tradings of convolutional neural networks, structured query results over databases, mathematical statistics and so on. We have implemented the `ZKCPlus` protocol, built the applications, and evaluated their performances. For the "pay to sudoku solution" application, on an ordinary desktop computer, the seller of ZKCP runs out of memory when the size of sudoku solution gets larger than 49×49. As a contrast, `ZKCPlus` successfully handles all solutions up to 256×256, and reduces the seller's proving overhead by 21-67×, with only milliseconds of setup time and about 1 MB public parameters. Even under the same encryption scheme, our CP-NIZK argument still shows an order of magnitude higher prover efficiency than the zkSNARK adopted by ZKCP. In the "pay to CNN model" application where `ZKCPlus` is applied to trading trained CNN models, the two parties can finish proving for an inference process of a 3-layer CNN with 8,620 parameters in about 1 s, and then finish delivering all these parameters within about 1 s as well. The application scales up to deep CNNs; on the well-known 16-layer VGG16 [50] model which contains about fifteen million parameters, `ZKCPlus` yields a practically acceptable overhead, which highlights a great advancement over ZKCP. We also build a "pay to SQL query" application for trading SQL query results on databases; the two

parities can finish trading all the results of a sub-string selection over 100,000 records in several seconds.

In summary, we make the following contributions:

- We propose ZKCPlus, an extended and optimized version of ZKCP, which supports efficient and versatile fair-exchange applications on large-scale data. With a new prover-efficient CP-NIZK argument of knowledge, ZKCPlus features light-weighted public setup phase and low seller's proving overhead, and is capable of supporting very complicated predicates.
- We build a prototype of ZKCPlus and develop several applications upon it, including a practical "pay to CNN model" application. Evaluations demonstrate the excellent performance of ZKCPlus, vastly surpassing the original ZKCP in many aspects.

2 Preliminaries

2.1 Notations

Let $\mathbb{G}$ be a cyclic group of a prime order p, and $\mathbb{F}_p$ be the corresponding scalar field. Vectors of length n over $\mathbb{G}$ and $\mathbb{F}_p$ are denoted by $\mathbb{G}^n$ and $\mathbb{F}_p^n$ respectively. By $x \xleftarrow{\$} \mathbb{F}_p$ we mean uniformly sampling an element x from $\mathbb{F}_p$. For a vector $\mathbf{a} \in \mathbb{F}_p^n$ and a scalar $c \in \mathbb{F}_p$ we write $\mathbf{b} = c \cdot \mathbf{a} \in \mathbb{F}_p^n$ where $\mathbf{b} = (c \cdot a_1, \ldots, c \cdot a_n)$. We write $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{j \in [n]} a_j \cdot b_j$ for inner product of $\mathbf{a}$ and $\mathbf{b}$, and $\mathbf{a} \circ \mathbf{b} = (a_1 \cdot b_1, \ldots, a_n \cdot b_n)$ for Hadamard (entry-wise) product. Group operations are written additively. For $x \in \mathbb{F}_p$ and $G \in \mathbb{G}$, scalar multiplication is denoted as $x \cdot G$, and multi-scalar multiplication is written in vector inner product form as $\langle \mathbf{x}, \mathbf{G} \rangle$.

Let R be a polynomial-time decidable relation on the pair (s, w), where s is an NP statement, and w is the witness. We write $\mathsf{R}(s, w) = 1$ if relation R holds on (s, w), and $\mathsf{R}(s, w) = 0$ otherwise. When necessary, we write out R explicitly as

$$\{\text{public inputs; witness} | \text{relation}\}.$$

2.2 Commitments

A non-interactive commitment scheme is a tuple of algorithms $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Check})$ which satisfies the notion of *correctness*, *binding* and *hiding* (formal definitions in Appendix A.2):

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ generates public parameter pp for the scheme on input of security parameter λ;
- $\mathsf{Commit}(\mathsf{pp}, u) \to (c, r)$ takes message u in message space M_{pp}, outputs its commitment c in commitment space C_{pp} and opening randomness (blinder) r in randomness space R_{pp};
- $\mathsf{Check}(\mathsf{pp}, c, u, r) \to b$ takes commitment c, message u, randomness r, and accepts ($b = 1$) or rejects ($b = 0$).

In this work, we mostly use the Pedersen commitment [47] scheme. Combining the scalar and vector forms, for message space $\mathsf{M}_{\mathsf{pp}} = \mathbb{F}_p^n$ with $n \geq 1$, randomness space $\mathsf{R}_{\mathsf{pp}} = \mathbb{F}_p$, and commitment space $\mathsf{C}_{\mathsf{pp}} = \mathbb{G}$, the scheme runs as

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ randomly samples $H \xleftarrow{\$} \mathbb{G}$, $\mathbf{G} \xleftarrow{\$} \mathbb{G}^n$ from a group $\mathbb{G}$ whose order is implicitly dependent on the security parameter, and outputs $\mathsf{pp} = (H, \mathbf{G})$;
- For vector input $\mathbf{v}$, $\mathsf{Commit}(\mathsf{pp}, \mathbf{v}) \to (V, v)$ samples $v \xleftarrow{\$} \mathbb{F}_p$, and computes $V = \langle \mathbf{v}, \mathbf{G} \rangle + v \cdot H$. When $n = 1$, the vector $\mathbf{v}$ reduces to a scalar v, and $V = v \cdot G + v \cdot H$;
- For vector input $\mathbf{v}$, $\mathsf{Check}(\mathsf{pp}, V, \mathbf{v}, v) \to b$ outputs binary decision $b = 1$ iff $V = \langle \mathbf{v}, \mathbf{G} \rangle + v \cdot H$. When $n = 1$, for scalar input v, $\mathsf{Check}(\mathsf{pp}, V, v, v) \to b$ outputs $b = 1$ iff $V = v \cdot U + v \cdot H$.

The scheme is *correct*, *perfectly hiding* and *computationally binding* under the discrete logarithm relation assumption defined in Appendix A.1. In addition, the scheme is *homomorphic* as

$$\mathsf{Commit}(\mathsf{pp}, \mathbf{v}_1) + \mathsf{Commit}(\mathsf{pp}, \mathbf{v}_2) = \mathsf{Commit}(\mathsf{pp}, \mathbf{v}_1 + \mathbf{v}_2),$$

where the add operation + is define in $\mathbb{G}$ on the LHS and in $\mathbb{F}_p$ on the RHS.

2.3 Zero-knowledge arguments of knowledge

For a polynomial-time decidable relation R and a statement s, a zero-knowledge argument of knowledge allows prover $\mathcal{P}$ to demonstrate to verifier $\mathcal{V}$ its knowledge of a witness w s.t. $\mathsf{R}(s, w) = 1$, and in the meanwhile reveal no information about w, except what can be inferred from the statement s. For interactive $\mathcal{P}$ and $\mathcal{V}$ both referring to a common reference string (CRS) σ generated by a preliminary algorithm KeyGen, we denote their transcript as $\mathsf{tr} \leftarrow \langle \mathcal{P}(\sigma, s, w), \mathcal{V}(\sigma, s; \rho) \rangle$, where ρ denotes the verifier's randomness. We write $\langle \mathcal{P}(\sigma, s, w), \mathcal{V}(\sigma, s; \rho) \rangle = b$ depending on whether $\mathcal{V}$ accepts ($b = 1$) or rejects ($b = 0$).

We say the tuple $\mathsf{Arg} = (\mathsf{KeyGen}, \mathcal{P}, \mathcal{V})$ is an argument of knowledge if it satisfies the notion of *perfect completeness* and *computational witness-extended emulation*. The argument Arg is said to be *public-coin* if all messages sent by $\mathcal{V}$ are sampled uniformly at random and independent of $\mathcal{P}$'s messages. A public-coin Arg is said to have *special honest-verifier zero-knowledge* if it is possible to simulate the transcript for any set of challenges without accessing the witness w. Formal definitions of these terms can be found in Appendix A.3.

By Fiat-Shamir heuristic [24], an interactive public-coin zero-knowledge argument Arg can be transformed into a non-interactive zero-knowledge (NIZK) argument of knowledge, where $\mathcal{V}$'s messages are replaced by outputs of a cryptographic hash function over the transcript up to the point. We model the resulting NIZK argument as a tuple of algorithms $\Pi = (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ s.t.

- $\mathsf{KeyGen}(1^\lambda) \to \sigma$ generates the CRS σ on input parameter λ;
- $\mathsf{Prove}(\sigma, s, w) \to \pi$ outputs proof π for $\mathsf{R}(s, w) = 1$;
- $\mathsf{Verify}(\sigma, s, \pi) \to b$ accepts ($b = 1$) or rejects ($b = 0$) the proof π.

In random oracle model, the resulting Π has *perfect completeness*, *computational knowledge soundness* and *zero-knowledge* (formal definitions in Appendix A.4), provided that the original interactive argument Arg has *perfect completeness*, *computational witness-extended emulation*, and *special honest-verifier zero-knowledge* [4].

In Appendix B, we describe an argument for inner products for vectors from [31], a basic building block that we use repeatedly on various occasions throughout this work.

2.4 Commit-and-prove arguments

We adopt the notion of *commit-and-prove* non-interactive arguments from LegoSNARK [12]. For a relation R and a commitment scheme Com with message space M_{pp}, a *commit-and-prove* NIZK (CP-NIZK) argument of knowledge proves the knowledge of a witness w for a statement s s.t. $\mathsf{R}(s, w) = 1$, where $w = (u, \mathsf{aux})$ splits into a message part $u \in \mathsf{M}_{\mathsf{pp}}$ which opens a specified commitment c, and an auxiliary part aux. More specifically, it demonstrates a relation

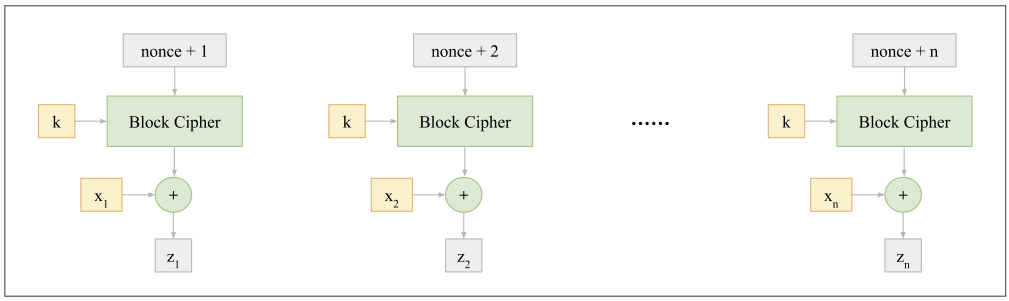

Figure 1: Block cipher in CTR mode

$\mathsf{R}^{\mathsf{Com}}$ defined over pairs $(\mathbf{s}, \mathbf{w})$ where $\mathbf{s} = (s, c)$ and $\mathbf{w} = (u, r, \mathsf{aux})$, s.t. $\mathsf{R}^{\mathsf{Com}}(\mathbf{s}, \mathbf{w}) = 1$ iff

$$\mathsf{Com.Check}(\mathsf{pp}, c, u, r) = 1 \ \wedge \ \mathsf{R}(s, (u, \mathsf{aux})) = 1. \tag{1}$$

We model the CP-NIZK argument of knowledge as a tuple of algorithms $\Pi^{\mathsf{Com}} = (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ where

- $\mathsf{KeyGen}(1^\lambda) \longrightarrow \sigma$ generates σ, which includes the public parameters of the commitment scheme $\mathsf{pp} \leftarrow \mathsf{Com.Setup}(1^\lambda)$;
- $\mathsf{Prove}(\sigma, (s, c), (u, r, \mathsf{aux})) \longrightarrow \pi$ outputs proof π for statement $\mathbf{s} = (s, c)$ and witness $\mathbf{w} = (u, r, \mathsf{aux})$;
- $\mathsf{Verify}(\sigma, (s, c), \pi) \longrightarrow b$ accepts ($b = 1$) or rejects ($b = 0$).

It should satisfy the notion of *completeness*, *knowledge soundness*, and *zero-knowledge* as a usual NIZK argument of knowledge. In practice, u may split into many segments. Informally, we write $u = \{u_i\}$, and they are committed as $c = \{c_i\}$ with blinders $r = \{r_i\}$. Correspondingly, $\mathsf{R}^{\mathsf{Com}}$ holds iff

$$\bigwedge_i \mathsf{Com.Check}(\mathsf{pp}, c_i, u_i, r_i) = 1 \ \wedge \ \mathsf{R}(s, (\{u_i\}, \mathsf{aux})) = 1. \tag{2}$$

The notion of *commit-and-prove* allows a "bottom-up" approach to build arguments for composed relations. The base case is the conjunction of relations with shared inputs. For two CP-NIZK arguments of knowledge Π_0^{Com} and Π_1^{Com} which respectively attesting relations $\mathsf{R}_0(s_0, (u, \mathsf{aux}_0)) = 1$ and $\mathsf{R}_1(s_1, (u, \mathsf{aux}_1)) = 1$, with shared input u. It is proved that for a *computationally binding* commitment scheme Com, there exists a CP-NIZK argument of knowledge $\Pi_\wedge^{\mathsf{Com}}$ for conjunction of R_0 and R_1. More specifically, $\Pi_\wedge^{\mathsf{Com}}$ works for relation $\mathsf{R}_\wedge$ s.t $\mathsf{R}_\wedge((s_0, s_1), (u, \mathsf{aux}_0, \mathsf{aux}_1)) = 1$ iff

$$\mathsf{R}_0(s_0, (u, \mathsf{aux}_0)) = 1 \ \wedge \ \mathsf{R}_1(s_1, (u, \mathsf{aux}_1)) = 1. \tag{3}$$

Extension to conjunctions of more than two relations is trivial. Disjunctions and sequential composition of functions can be reduced to this base case as well. Refer to Appendix C for more details.

Herein we single out a special type of CP-NIZK argument denoted as $\Pi_{\mathsf{link}}^{\mathsf{Com}}$. Let Π_0^{Com} and Π_1^{Com} be CP-NIZK arguments of knowledge for relation R_0 and R_1 s.t. $\mathsf{R}_0(s_0, (u_0, \mathsf{aux}_0)) = 1$, and $\mathsf{R}_1(s_1, (u_1, \mathsf{aux}_1)) = 1$. Argument $\Pi_{\mathsf{link}}^{\mathsf{Com}}$ shares input u_0 with Π_0^{Com}, input u_1 with Π_1^{Com}, and proves that u_0 and u_1 are related by an interlinking function f s.t. $f(u_0, u_1) = 0$. By composability of CP-NIZK arguments, Π_0^{Com}, Π_1^{Com} and $\Pi_{\mathsf{link}}^{\mathsf{Com}}$ jointly prove the relation

$$\mathsf{R}_0(s_0, (u_0, \mathsf{aux}_0)) = 1 \ \wedge \ \mathsf{R}_1(s_1, (u_1, \mathsf{aux}_1)) = 1 \ \wedge \ f(u_0, u_1) = 0. \tag{4}$$

Therefore, $\Pi_{\mathsf{link}}^{\mathsf{Com}}$ acts as an adaptor between Π_0^{Com} and Π_1^{Com}. With $\Pi_{\mathsf{link}}^{\mathsf{Com}}$, it is possible to assemble small CP-NIZK arguments together, and build an argument that works for very complicated relations.

2.5 Rank-1 constraint system

Rank-1 Constraint System (R1CS) extends the notion of Quadratic Arithmetic Program (QAP) [27], and offers a convenient description for arithmetic circuits. For a circuit of size m (containing m multiplication gates), its corresponding R1CS is written as $\{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathbf{s}; \mathbf{w}\}$, where $\mathsf{A}, \mathsf{B}, \mathsf{C} \in \mathbb{F}_p^{m \times l}$ are the coefficient matrices, $\mathbf{s}$ represents the public input, and $\mathbf{w}$ the witness for the circuit. An assignment $\mathbf{x} = (1, \mathbf{s}, \mathbf{w}) \in \mathbb{F}_p^l$ satisfies the circuit iff

$$\mathsf{A}\mathbf{x} \circ \mathsf{B}\mathbf{x} = \mathsf{C}\mathbf{x}. \tag{5}$$

For simplicity, in the following we shall assume $m = l$.

2.6 MiMC block cipher

MiMC [1] is a family of symmetric cryptographic primitives that works natively in finite fields. In particular, MiMC-p/p is a block cipher $C_k(\cdot)$ operating over a prime field $\mathbb{F}_p$, constructed by embedding a non-linear permutation $F(x) = x^d$ for $x \in \mathbb{F}_p$ in its round function, where the exponent d satisfies the condition $gcd(d, p - 1) = 1$. For each round, the round function is defined as $F_i(x) = F(x + k + c_i)$, where k denotes the encryption key, and c_i a round constant. The number of rounds should be sufficiently large to thwart the interpolation attacks [36]. The block cipher is arithmetic circuit friendly; for example, it promises 129-bit security with only 82 multiplications in $\mathbb{F}_p$.

We use the MiMC-p/p block cipher in the counter (CTR) mode. An arbitrary input message $\mathbf{x}$ is encoded as a vector over $\mathbb{F}_p$ of length n. With key k, $\mathbf{x} \in \mathbb{F}_p^n$ is encrypted into ciphertext $\mathbf{z} \in \mathbb{F}_p^n$, whose i-th element is computed by

$$z_i = x_i + C_k(\mathsf{ctr}_i), \tag{6}$$

where counter ctr_i is offset by a random nonce, $\mathsf{ctr}_i = \mathsf{nonce} + i$, and the operation $+$ is defined in $\mathbb{F}_p$. Collectively, we write $\mathbf{z} = \mathsf{Enc}_k(\mathbf{x})$.

2.7 Blockchain and smart contract

Bitcoin [44] is a decentralized ledger system jointly maintained by indefinite number of nodes via a peer-to-peer consensus protocol. If a majority of the nodes honour the protocol, the ledger is ensured correctness without any centralized coordination. On Bitcoin blockchain, users' currencies and transactions are managed by simple Bitcoin scripts with limited functionalities.

Going one step further, Ethereum [54] extends the ledger to a "world state" consisting of rich variables, which can be modified by code called "smart contracts". A smart contract consists of global variables and functions, and the functions can be invoked by Ethereum users or other contracts reactively. The miners execute the invoked functions to update the global state of the ledger. In contrast with the limited Bitcoin scripts, smart contracts on Ethereum support Turing complete language. In particular, it allows elliptic curve operations, so on-chain Pedersen commitment opening is made easy.

3 Building block: CP-NIZK argument for data-parallel computations

In this section, we focus on the construction of an efficient CP-NIZK argument for data-parallel circuits. We say an arithmetic circuit is *data-parallel* if it consists of multiple copies of identical sub-circuits running on separated inputs. This argument is a core building block for our ZKCPlus protocol.

3.1 Argument construction

Consider a data-parallel circuit consisting of n identical sub-circuits, and each sub-circuit is of size m. Instead of constructing a large R1CS for the whole circuit of size $n \times m$, we use R1CS for the sub-circuit of size m and apply it to n assignments. Specifically, we define matrix

$$\mathsf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{F}_p^{m \times n}, \tag{9}$$

where $\mathbf{x}_k$ is the assignment to the k-th sub-circuit. The satisfiability relation for the whole circuit can be stated concisely as

$$\mathsf{AX} \circ \mathsf{BX} = \mathsf{CX}. \tag{10}$$

Instead of interpreting X as a collection of n columns of $\mathbf{x}_k \in \mathbb{F}_p^m$, each of length m, henceforth we view it as a stack of m row vectors each of length n. We denote the row vectors as $\mathbf{v}_j \in \mathbb{F}_p^n$ for $j \in [m]$. They are divided into two categories, those corresponding to public inputs (statement) with $j \in \mathsf{pub}$, and those for $\mathcal{P}$'s private inputs (witness) with $j \in \mathsf{priv}$, where $\mathsf{pub} \cup \mathsf{priv} = [m]$. We reformulate the satisfiability relation of (10) as m identities about Hadamard products, namely

$$\left\{ \begin{array}{c|l} (\mathsf{A},\mathsf{B},\mathsf{C}) \in \mathbb{F}_p^{m\times m}, & \mathbf{a}_i = \sum_{j\in[m]} A_{ij}\cdot\mathbf{v}_j, \\ \{\mathbf{v}_j \in \mathbb{F}_p^n\}_{j\in\mathsf{pub}}; & \mathbf{b}_i = \sum_{j\in[m]} B_{ij}\cdot\mathbf{v}_j, \\ \{\mathbf{v}_j \in \mathbb{F}_p^n\}_{j\in\mathsf{priv}} & \mathbf{c}_i = \sum_{j\in[m]} C_{ij}\cdot\mathbf{v}_j, \\ & \bigwedge_{i\in[m]} \mathbf{a}_i \circ \mathbf{b}_i = \mathbf{c}_i \end{array} \right\}, \tag{11}$$

where M_{ij} denotes an element of matrix M with row and column indices i and j for $M \in \{A, B, C\}$. We build an interactive protocol for this relation using techniques in [11, 31]. The details are in Figure 2.

The protocol relies on the Pedersen commitment scheme Com as described in Section 2.2. In a nutshell, it involves a prover $\mathcal{P}$ committing $\mathbf{v}_j$ as V_j. With V_j the verifier $\mathcal{V}$ can compute commitments to $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i$ utilizing the homomorphic property of Com as

$$A_i = \sum_{j\in[m]} A_{ij}\cdot V_j, \quad B_i = \sum_{j\in[m]} B_{ij}\cdot V_j, \quad C_i = \sum_{j\in[m]} C_{ij}\cdot V_j. \tag{12}$$

Then $\mathcal{P}$ contrives to convince $\mathcal{V}$ that for $i \in [m]$ the vectors $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i$ satisfy the equations $\mathbf{a}_i \circ \mathbf{b}_i = \mathbf{c}_i$. It is in turn converted into an equation for inner product $\langle \mathbf{a}_i, \mathbf{b}_i \circ \mathbf{y}_n \rangle - \langle \mathbf{c}_i, \mathbf{y}_n \rangle = 0$, where $\mathbf{y}_n =$

Public input: R1CS instance $\mathsf{A}, \mathsf{B}, \mathsf{C} \in \mathbb{F}_p^{m\times m}$.

$\mathcal{P}$**'s private input**: $\{\mathbf{v}_j \in \mathbb{F}_p^n\}_{i\in[m]}$.

$\mathcal{P}$: for $i \in [m]$ $\quad (V_j, v_j) \leftarrow \mathsf{Commit}(\mathsf{pp}, \mathbf{v}_j)$.

$\mathcal{P} \to \mathcal{V}$: $\{V_j\}_{j\in[m]}$.

$\mathcal{V} \to \mathcal{P}$: $w, y \xleftarrow{\$} \mathbb{F}_p$.

$\mathcal{P}$: for $i \in [m]$

$$\mathbf{l}_i = w^i \cdot \Big(\sum_j A_{ij}\cdot\mathbf{v}_j\Big), \quad \mathbf{l}_{i+m} = w^i \cdot \Big(\sum_j C_{ij}\cdot\mathbf{v}_j\Big), \tag{7}$$

$$\mathbf{r}_i = \Big(\sum_j B_{ij}\cdot\mathbf{v}_j\Big) \circ \mathbf{y}_n, \quad \mathbf{r}_{i+m} = -\mathbf{y}_n, \tag{8}$$

$$\iota_i = w^i \cdot \Big(\sum_j A_{ij}\cdot v_j\Big), \quad \iota_{i+m} = w^i \cdot \Big(\sum_j C_{ij}\cdot v_j\Big),$$

$$\rho_i = \Big(\sum_j B_{ij}\cdot v_j\Big), \quad \rho_{i+m} = 0.$$

$\mathcal{P}$**'s private input**: $\{\mathbf{l}_i, \mathbf{r}_i \in \mathbb{F}_p^n\}_{i\in[2m]}, \{\iota_i, \rho_i \in \mathbb{F}_p\}_{i\in[2m]}, \tau = 0$.

Repeat for k **in** $[\log m \ldots 0]$:

$\mathcal{P}$: $d = 2^k, \quad t_k^+ = \sum_{i=1}^{d} \langle \mathbf{l}_i, \mathbf{r}_{i+d} \rangle, \quad t_k^- = \sum_{i=1}^{d} \langle \mathbf{l}_{i+d}, \mathbf{r}_i \rangle,$

$(T_k^+, \tau_k^+) \leftarrow \mathsf{Commit}(\mathsf{pp}, t_k^+), \quad (T_k^-, \tau_k^-) \leftarrow \mathsf{Commit}(\mathsf{pp}, t_k^-).$

$\mathcal{P} \longrightarrow \mathcal{V}$: T_k^+, T_k^-.

$\mathcal{V} \longrightarrow \mathcal{P}$: $u_k \xleftarrow{\$} \mathbb{F}_p$.

$\mathcal{P}$: for $i \in [d]$

$$\mathbf{l}_i \leftarrow u_k\cdot\mathbf{l}_i + u_k^{-1}\cdot\mathbf{l}_{i+d}, \quad \mathbf{r}_i \leftarrow u_k^{-1}\cdot\mathbf{l}_i + u_k\cdot\mathbf{r}_{i+d},$$

$$\iota_i \leftarrow u_k\cdot\iota_i + u_k^{-1}\cdot\iota_{i+d}, \quad \mathbf{r}_i \leftarrow u_k^{-1}\cdot\rho_i + u_k\cdot\rho_{i+d},$$

$$\tau \leftarrow \tau + u_k^2\cdot\tau_k^+ + u_k^{-2}\cdot\tau_k^-.$$

Reduced: $\mathbf{z} = (u_0u_1\cdots u_k,\ u_0u_1\cdots u_k^{-1},\ \ldots,\ u_0^{-1}u_1^{-1}\cdots u_k^{-1}) \in \mathbb{F}_p^{2m}$,

$$\mathbf{l} = \sum_{i\in[2m]} z_i\cdot\mathbf{l}_i, \quad \mathbf{r} = \sum_{i\in[2m]} z_i^{-1}\cdot\mathbf{r}_i,$$

$$\iota = \sum_{i\in[2m]} z_i\cdot\iota_i, \quad \rho = \sum_{i\in[2m]} z_i^{-1}\cdot\rho_i,$$

$$\tau = \sum_{k\in[0,\log m]} u_k^2\cdot\tau_k^+ + u_k^{-2}\cdot\tau_k^-.$$

$\mathcal{P}$: $\mathbf{l}_d, \mathbf{r}_d \xleftarrow{\$} \mathbb{F}_p^n$,

$$s_1 = \langle \mathbf{l}, \mathbf{r}_d \circ \mathbf{y}_n \rangle + \langle \mathbf{l}_d, \mathbf{r} \rangle, \quad s_2 = \langle \mathbf{l}_d, \mathbf{r}_d \circ \mathbf{y}_n \rangle,$$

$$(L_d, \iota_d) \leftarrow \mathsf{Commit}(\mathsf{pp}, \mathbf{l}_d), \quad (R_d, \rho_d) \leftarrow \mathsf{Commit}(\mathsf{pp}, \mathbf{r}_d),$$

$$(S_1, \sigma_1) \leftarrow \mathsf{Commit}(\mathsf{pp}, s_1), \quad (S_2, \sigma_2) \leftarrow \mathsf{Commit}(\mathsf{pp}, s_2).$$

$\mathcal{P} \to \mathcal{V}$: L_d, R_d, S_1, S_2.

$\mathcal{V} \to \mathcal{P}$: $e \xleftarrow{\$} \mathbb{F}_p$.

$\mathcal{P}$: $\mathbf{l}' = \mathbf{l} + e\cdot\mathbf{l}_d, \quad \mathbf{r}' = \mathbf{r} + e\cdot\mathbf{r}_d \circ \mathbf{y}_n,$

$$\iota' = \iota + e\cdot\iota_d, \quad \rho' = \rho + e\cdot\rho_d, \quad \tau' = \tau + e\cdot\sigma_1 + e^2\cdot\sigma_2.$$

$\mathcal{P} \to \mathcal{V}$: $\mathbf{l}', \mathbf{r}', \iota', \rho', \tau'$.

$\mathcal{V}$: $L = \sum_{i\in[m]}\sum_{j\in[m]} w^i\cdot\big(z_i\cdot A_{ij} + z_{i+m}\cdot C_{ij}\big)\cdot V_j,$

$$L' = L_0 + e\cdot L_d, \quad \mathsf{Check}(\mathsf{pp}, L', \mathbf{l}', \iota') \stackrel{?}{=} 1,$$

$$R = \sum_{i\in[m]}\sum_{j\in[m]} \big(z_i\cdot B_{ij}\big)\cdot V_j + \sum_{i\in[m]} z_{i+m}\cdot\langle \mathbf{1}, \mathbf{G}\rangle,$$

$$R' = R + e\cdot R_d, \quad \mathsf{Check}(\mathsf{pp}, R', \mathbf{r}'\circ\mathbf{y}_n^{-1}, \rho') \stackrel{?}{=} 1,$$

$$T' = \sum_{k=0}^{\log 2m} \Big(u_k^2\cdot T_k^+ + u_k^{-2}\cdot T_k^-\Big) + e\cdot S_1 + e^2\cdot S_2,$$

$$\mathsf{Check}(\mathsf{pp}, T', \langle \mathbf{l}, \mathbf{r}\rangle, \tau') \stackrel{?}{=} 1.$$

Figure 2: Protocol for data-parallel circuits

$(y, y^2, \cdots, y^n)$ is a vector generated by challenger $y \xleftarrow{\$} \mathbb{F}_p$. Further, the m equations are combined by another challenger $w \xleftarrow{\$} \mathbb{F}_p$,

$$\sum_{i\in[m]} w^i\Big(\langle \mathbf{a}_i, \mathbf{b}_i \circ \mathbf{y}_n\rangle - \langle \mathbf{c}_i, \mathbf{y}_n\rangle\Big) = 0. \tag{13}$$

Note that (13) is about a sum of $2m$ inner products. For simplicity, we define $\{\mathbf{l}_i\}_{i\in[2m]}$ and $\{\mathbf{r}_i\}_{i\in[2m]}$ in (7) and (8), and rewrite (13) as $\sum_{i\in[2m]}\langle \mathbf{l}_i, \mathbf{r}_i\rangle = 0$. Following a recursive method in [31], the protocol proceeds in $(\log m + 1)$ rounds of interactions (starting from $k = \log m$ and finishing by $k = 0$). In the k-th round $\mathcal{P}$ sends the commitments to

$$t_k^+ = \sum_{i=1}^{2^k}\langle \mathbf{l}_i, \mathbf{r}_{i+2^k}\rangle, \quad t_k^- = \sum_{i=1}^{2^k}\langle \mathbf{l}_{i+2^k}, \mathbf{r}_i\rangle, \tag{14}$$

and $\mathcal{V}$ responds with a random challenger u_k. $\mathcal{P}$ then computes

$$\mathbf{l}_i \leftarrow u_k \cdot \mathbf{l}_i + u_k^{-1} \cdot \mathbf{l}_{i+m}, \quad \mathbf{r}_i \leftarrow u_k^{-1} \cdot \mathbf{l}_i + u_k \cdot \mathbf{r}_{i+m} \tag{15}$$

for the next round. Note how the number of involving vectors is halved. After the final round, we arrive at two vectors

$$\mathbf{l} = \sum_{i=1}^{2m} z_i \cdot \mathbf{l}_i, \quad \mathbf{r} = \sum_{i=1}^{2m} z_i^{-1} \cdot \mathbf{r}_i, \tag{16}$$

where z_i is the i-th element vector $\mathbf{z}$ with a binary counting structure

$$\mathbf{z} = \begin{pmatrix} u_0 u_1 \dots u_{\log m} \\ u_0^{-1} u_1 \dots u_{\log m} \\ \vdots \\ u_0^{-1} u_1^{-1} \dots u_{\log m}^{-1} \end{pmatrix}, \tag{17}$$

whereas (13) holds iff the following equation

$$\langle \mathbf{l}, \mathbf{r}\rangle = \sum_{k=0}^{\log m} u_k^2 \cdot t_k^+ + u_k^{-2} \cdot t_k^- \tag{18}$$

holds. More details about the reduction is in Appendix B.4.

Theorem 3.1. *The protocol in Figure 2 for relation* (11) *has* perfect completeness, computational witness-extended emulation *and* perfect special honest-verifier zero-knowledge *under the discrete logarithm relation assumption.*

Proof for Theorem 3.1 is in Appendix D. By the Fiat-shamir heuristic, we can convert it into an NIZK argument of knowledge, which has *perfect completeness*, *computational knowledge soundness* and *perfect zero-knowledge* in the random oracle model.

3.2 Proof composition

Observe that the construction in Figure 2 starts with $\mathcal{P}$ committing private $\mathbf{v}_j$ as V_j for $j \in \mathsf{priv}$. We may include a subset of V_j for $j \in \mathsf{cms}$ where $\mathsf{cms} \subset \mathsf{priv}$ as part of the public input (statement), and convert the relation of (11) into the form

$$\left\{ \begin{array}{l|c} (\mathsf{A}, \mathsf{B}, \mathsf{C}) \in \mathbb{F}_p^{m\times m}, & \mathbf{a}_i = \sum_{j\in[m]} A_{ij}\mathbf{v}_j, \\ \{\mathbf{v}_j \in \mathbb{F}_p^n\}_{j\in\mathsf{pub}}, & \mathbf{b}_i = \sum_{j\in[m]} B_{ij}\mathbf{v}_j, \\ \{V_j \in \mathbb{G}\}_{j\in\mathsf{cms}}; & \mathbf{c}_i = \sum_{j\in[m]} C_{ij}\mathbf{v}_j, \\ \{\mathbf{v}_j \in \mathbb{F}_p^n\}_{j\in\mathsf{priv}}, & \bigwedge_{i\in[m]} \mathbf{a}_i \circ \mathbf{b}_i = \mathbf{c}_i \wedge \\ \{v_j \in \mathbb{F}_p\}_{j\in\mathsf{cms}} & \bigwedge_{j\in\mathsf{cms}} \mathsf{Check}(\mathsf{pp}, V_j, \mathbf{v}_j, v_j) = 1 \end{array} \right\}. \tag{19}$$

It is straightforward to see that relation (19) fits the notion of $\mathsf{R}^{\mathsf{Com}}$ for CP-NIZK arguments.

The composable nature of CP-NIZK arguments allows modularly building of arguments for very complex relations by linking arguments for relatively simpler subroutines. Moreover, in our construction vector $\mathbf{v}_j \in \mathbb{F}_p^n$ is arranged to include the assignments to the same j-th entry for each of the n sub-circuits, which makes reusing its commitment V_j across different subroutines almost free of cost. For example, consider the typical MapReduce model for large-scale data processing, where a map function is applied in parallel to each chunk of data, and the outputs are aggregated and fed into a reduce function. The map function itself may contain many identical subroutines running on shared inputs, or on outputs of the previous subroutines. Argument in Figure 2 allows modular proof constructions, where the shared inputs/outputs are committed once and for all, despite their multiple appearances across the full computation.

Another promising application is for CNN models in machine learning context. A CNN is typically aggregated into consecutive layers, whereas each layer is comprised of an array of identical neurons. Our argument applies naturally to the intra-layer computation, and the inter-layer coupling can be solved by simple adaptors $\Pi_{\mathsf{link}}^{\mathsf{Com}}$. Extended discussion on this point can be found in Section 5.

3.3 Performance

The protocol in Figure 2 features a proof size of roughly $(m \cdot \mathbb{G} + 2n \cdot \mathbb{F}_p)$, counting the commitments V_j and vectors $\mathbf{l}', \mathbf{r}'$. $\mathcal{P}$'s computation is dominated by mn scalar multiplications in $\mathbb{G}$ for committing $\mathbf{v}_j$, and $O(nm)$ multiplications in $\mathbb{F}_p$. On $\mathcal{V}$'s side, the most expensive operations are to compute T', L' and R' and to run Verify procedures for $\mathbf{l}', \mathbf{r}'$, which amounts to $(m+n)$ scalar multiplications in $\mathbb{G}$ and $O(nm)$ multiplications in $\mathbb{F}_p$. We can further squeeze the proof size down to $(m + 2\log n) \cdot \mathbb{G}$ using the reduction technique in Bulletproofs [11]. The main characteristics are summarised in Table 2, where for simplicity, only the leading terms are listed.

For better illustration of its performance, in Table 2 we also compare it to two other zero-knowledge argument systems for R1CS, Groth16 zkSNARK [32], and a non-succinct version of $\mathsf{Spartan}_{\mathrm{DL}}$ [49]. These two schemes are chosen as two extremes. Groth16 requires a trusted setup phase, and features an almost optimum proof size and an extremely efficient verifier. On the contrary, $\mathsf{Spartan}_{\mathrm{DL}}$ is discrete logarithmic assumption-based, and it offers arguably the fastest prover so far among elliptic-curve-based NIZK arguments for R1CS. For comparison, we assume a data-parallel circuit comprised of n sub-circuits, where each sub-circuit is of size m. For Groth16 and $\mathsf{Spartan}_{\mathrm{DL}}$, it is interpreted as a single large circuit of size mn.

In many aspects, our construction is comparable to $\mathsf{Spartan}_{\mathrm{DL}}$. They both avoid the trusted setup phase required by Groth16. They offer efficient provers, several times faster than that of Groth16 (considering that Groth16 needs an elliptic curve that is both pairing-friendly and fft-friendly while $\mathsf{Spartan}_{\mathrm{DL}}$ and our construction not, in practice the difference could be more than 10×). They have sub-linear verifiers and proof sizes, where those of Groth16 are constant.

For ZKCP, a trusted setup phase is undesirable, the proof size is usually not a bottleneck, and the system throughput (i.e., amount of data exchanged end-to-end in unit time span) is jointly determined

Table 2: Comparison with Groth16 and Spartan$_{\mathrm{DL}}$

Construction	Setup type		Sizes		Time complexity	
			$\lvert\sigma\rvert$	$\lvert\pi\rvert$	Prover	Verifier
Groth16	Trusted	$\mathbb{G}$	$3mn$	2	$4mn$	3 Pairings
		$\mathbb{G}_2$	mn	1	mn	
		$\mathbb{F}_p$	–	–	$O(mn * \log mn)$	–
Spartan$_{\mathrm{DL}}$	Public	$\mathbb{G}$	$\sqrt{mn}$	$\sqrt{mn}$	mn	$2\sqrt{mn}$
		$\mathbb{G}_2$	–	–	–	–
		$\mathbb{F}_p$	–	$O(\log mn)$	$O(mn)$	$O(\sqrt{mn})$
This work	Public	$\mathbb{G}$	n	m	mn	$m+n$
		$\mathbb{G}_2$	–	–	–	–
		$\mathbb{F}_p$	–	$O(\log mn)$	$O(mn)$	$O(m+n)$

by both the prover and the verifier. So Spartan$_{\mathrm{DL}}$ and our construction seem to be better choices than Groth16. In the context of data-parallel computations, we prefer our construction to Spartan$_{\mathrm{DL}}$ for flexible proof compositions and reuse of commitments. We note that Spartan$_{\mathrm{DL}}$ can also be modified into a data-parallel form, but the details are beyond the scope of the current work.

4 The ZKCPlus protocol

In this section we describe ZKCPlus, an extension of ZKCP, which focuses on optimizing and scaling up the ZKCP protocol, and adapting it to practically large-scale data exchange.

4.1 ZKCP revisited

Zero-Knowledge Contingent Payment (ZKCP) protocol tackles the problem of conducting fair exchange of digital goods and currencies without any centralized trust. It involves three parties, a seller $\mathcal{S}$, a buyer $\mathcal{B}$, and a *script/smart contract* implemented on a blockchain network acting as an arbiter $\mathcal{J}$. $\mathcal{S}$ runs on a private input of digital good x s.t. for a prescribed predicate ϕ, $\phi(x) = 1$ holds. $\mathcal{B}$ runs on a private input of digital currency p.

Let Enc be an encryption scheme, H a cryptographic hash function, and Π an NIZK argument of knowledge. With a polynomial-time computable predicate ϕ, a ZKCP protocol generally runs as follows:

1. **Validate & Deliver**:
   - $\mathcal{S}$ on input of digital good x, chooses an encryption key k, and computes ciphertext $z \leftarrow \mathsf{Enc}_k(x)$, hash image (hash-lock) $h \leftarrow \mathsf{H}(k)$, and proof $\pi \leftarrow \Pi.\mathsf{Prove}(\sigma, (z, h), (x, k))$ which attests the knowledge of (x, k) s.t.
   $$\phi(x) = 1 \wedge z = \mathsf{Enc}_k(x) \wedge h = \mathsf{H}(k), \tag{20}$$
   and sends the tuple (z, h, π) to $\mathcal{B}$;
   - $\mathcal{B}$ on receipt of (z, h, π) runs $b \leftarrow \Pi.\mathsf{Verify}(\sigma, (z, h), \pi)$, aborts if $b = 0$, otherwise builds a transaction $\mathsf{tx} = (p, h)_{\mathsf{sig}_{\mathcal{B}}}$ which pays funds p to one who opens the hash-lock h by revealing its preimage, and sends tx to $\mathcal{J}$;
2. **Reveal**: $\mathcal{S}$ checks if tx posted by $\mathcal{B}$ contains the designated hash image h, aborts if not, otherwise reveals k;
3. **Finalize**: On the revealed k, $\mathcal{J}$ executes tx if $\mathsf{H}(k) = h$, which will transfer $\mathcal{B}$'s funds p to $\mathcal{S}$, otherwise the funds are returned to $\mathcal{B}$. If k is correct, $\mathcal{B}$ can retrieve the digital good by running $x \leftarrow \mathsf{Enc}_k^{-1}(z)$.

A ZKCP protocol is said to be *secure* if it fulfills the following criteria:

- (*buyer fairness*) for any possibly malicious seller $\mathcal{S}^*$, if its balance increases with non-negligble probability, then the buyer $\mathcal{B}$ learns some x' s.t. $\phi(x') = 1$;
- (*seller fairness*) for a seller $\mathcal{S}$ interacting with a possibly malicious buyer $\mathcal{B}^*$, if $\mathcal{S}$'s balance does not increase, then except for negligible probability, $\mathcal{B}^*$ learns no information of x, apart from what can be inferred from ϕ.

As for its performance, the most important metric is end-to-end throughput, the amount of data transferred from $\mathcal{S}$ to $\mathcal{B}$ in unit time.

The original ZKCP design uses Pinocchio/BCTV14 zkSNARK for Π, and SHA256-based stream cipher for Enc [9]. In retrospect, we observe that these choices are not optimal. As for Π, although Pinocchio/BCTV14 [7, 46] (and its variate Groth16 [32]) have many strong points, including constant proof size and very efficient verifier, they also have some significant drawbacks. For one thing, they require a trusted CRS, but relying on a trusted third party to produce the CRS contradicts ZKCP's very premise of being trustless. So ZKCP instead entrusts the buyer $\mathcal{B}$ to produce the CRS. Unfortunately, this circumvention turns out to be problematic: the proof system has to ensure subversion zero-knowledge to prevent $\mathcal{B}$ from learning information about x [13, 25], but subversion zero-knowledge is usually too expensive to be of practical use. For another, the throughput of a ZKCP protocol is jointly determined by both parties. While these zkSNARKs offer extremely efficient verifier, it is the prover efficiency that sets the upper limit on the overall throughput. Another prominent merit of these schemes, namely the constant proof size, does not help too much either, since in ZKCP the proof is transferred off chain alongside with the linear-sized ciphertext.

Also concerning Enc and H, many zero-knowledge-proof-friendly ciphers and hash functions emerge in recent years. Operating natively in large prime fields (or binary fields), they promise comparable security strength as AES or SHA256 with orders of magnitude less constraints in the circuits. Incorporating these schemes into the existing ZKCP protocol can vastly boost the protocol's performance.

4.2 Construction of ZKCPlus

We propose a variate of the ZKCP protocol, which we call ZKCPlus, as listed in details in the following. By ZKCPlus we aim at adapting ZKCP to efficient and flexible fair exchange of data on large scale.Henceforth we write the digital good in vector form $\mathbf{x} \in \mathbb{F}_p^n$, and accordingly its ciphertext $\mathbf{z} \in \mathbb{F}_p^n$.

1. **Commit**: $\mathcal{S}$ runs $(c_{\mathbf{x}}, r_{\mathbf{x}}) \leftarrow \mathsf{Com.Commit}(\mathsf{pp}, \mathbf{x})$, and publishes the commitment $c_{\mathbf{x}}$;
2. **Validate**:
   - $\mathcal{B}$ posts predicate ϕ, which specifies a relation R, possibly with input s, s.t. $\phi(\mathbf{x}) = 1$ iff $\mathsf{R}(s, \mathbf{x}) = 1$;
   - $\mathcal{S}$ runs $\pi_\phi \leftarrow \Pi^{\mathsf{Com}}.\mathsf{Prove}(\sigma, (s, c_{\mathbf{x}}), (\mathbf{x}, r_{\mathbf{x}}, \mathsf{aux}))$ using a CP-NIZK argument Π^{Com}, with aux being auxiliary witness;
   - $\mathcal{B}$ blindly checks whether $\mathbf{x}$ is satisfying by running $b \leftarrow \Pi^{\mathsf{Com}}.\mathsf{Verify}(\sigma, (s, c_{\mathbf{x}}), \pi_\phi)$, and aborts if $b = 0$;
3. **Deliver**:
   - $\mathcal{S}$ chooses a key k, encrypts $\mathbf{x}$ by $\mathbf{z} \leftarrow \mathsf{Enc}_k(\mathbf{x})$, and produces a "proof of delivery" by
   $$\pi_z \leftarrow \Pi^{\mathsf{Com}}.\mathsf{Prove}(\sigma, (\mathbf{z}, c_{\mathbf{x}}, h), (x, k, r_{\mathbf{x}}, \mathsf{aux}))$$
   which attests the knowledge of $(\mathbf{x}, k)$ s.t. $\mathbf{z} = \mathsf{Enc}_k(\mathbf{x}) \wedge \mathsf{H}(k) = h$; $\mathcal{S}$ sends out tuple $(\mathbf{z}, h, \pi_z)$;
   - $\mathcal{B}$ runs $b \leftarrow \Pi^{\mathsf{Com}}.\mathsf{Verify}\big(\sigma, (\mathbf{z}, c_x, h), \pi_z\big)$, aborts if $b = 0$, otherwise builds a transaction $\mathsf{tx} = (p, h)_{\mathsf{sig}_{\mathcal{B}}}$ which pays funds p to who presents the preimage of h, and sends tx to $\mathcal{J}$;
4. **Reveal**: (same as ZKCP);
5. **Finalize**: (same as ZKCP).

4.2.1 Modifications as compared to ZKCP ZKCPlus's core modification is to replace Pinocchio/BCTV14 zkSNARKs by CP-NIZK argument schemes. The digital good $\mathbf{x}$ in exchange is committed in the first place, and all later proofs refer to its commitment $c_{\mathbf{x}}$. In this respect, we can say ZKCPlus is more "data-centric". Moreover, the modularity of CP-NIZK arguments allows finer-gained organization of the protocol. Concretely, we split "validate" and "deliver" into two phases. In a "validate" phase, $\mathcal{B}$ posts desired predicate ϕ, whereas $\mathcal{S}$ responds with a proof π_ϕ for

$$\phi(\mathbf{x}) = 1. \tag{21}$$

Then in a "deliver" phase, $\mathcal{S}$ sends out ciphertext $\mathbf{z}$, accompanied by a "proof of delivery" π_z attesting that $\mathbf{z}$ is faithfully computed from $\mathbf{x}$ using encryption key k, namely

$$\mathbf{z} = \mathsf{Enc}_k(\mathbf{x}) \wedge h = \mathsf{H}(k). \tag{22}$$

Note relations in (21) and (22) share input of $\mathbf{x}$. Given the composition properties of CP-NIZK arguments, π_ϕ and π_z jointly attests their conjunction of (20).

We use the CP-NIZK argument described in the previous section in ZKCPlus whenever a computation is data-parallel. In particular, with a block cipher running in CTR mode, the encryption procedure in the "deliver" phase is in fact data-parallel. We develop a "proof of delivery" based on this observation.

4.2.2 Proof of delivery We use MiMC-p/p block cipher in CTR mode for the encryption scheme Enc. More specifically, the i-th element of $\mathbf{z}$ (denoted z_i) is computed from the i-th element of $\mathbf{x}$ (denote x_i) by

$$z_i = x_i + C_k(\mathsf{nonce} + i). \tag{23}$$

As illustrated in Figure 1, this is in essence a data-parallel computation, where i-th block cipher runs on inputs of x_i, k and (nonce + i), and produces output z_i. Therefore, the CP-NIZK argument for data-parallel circuits in Section 3 applies. In aggregation, we write $\mathbf{k} = k \cdot \mathbf{1} = [k, k, \ldots, k]$, $\mathbf{ctr} = [\mathsf{nonce} + 1, \mathsf{nonce} + 2, \ldots, \mathsf{nonce} + n]$; $\mathbf{ctr}$ and $\mathbf{z}$ are public, whereas $\mathbf{x}$ and $\mathbf{k}$ are private and committed as

$$(c_{\mathbf{x}}, r_{\mathbf{x}}) \leftarrow \mathsf{Commit}(\mathsf{pp}, \mathbf{x}), \quad (c_{\mathbf{k}}, r_{\mathbf{k}}) \leftarrow \mathsf{Commit}(\mathsf{pp}, \mathbf{k}). \tag{24}$$

Note that $c_{\mathbf{x}}$ is the very commitment of $\mathbf{x}$ sent in the "commit" phase. We employ the argument described in Section 3 to prove the relation

$$\left\{ \begin{array}{l|r} \mathsf{A}, \mathsf{B}, \mathsf{C} \in \mathbb{F}_p^{m \times m}, & \mathbf{a}_i = \sum_{j \in [m]} A_{ij} \cdot \mathbf{v}_j, \\ \mathsf{ctr}, \mathbf{z} \in \mathbb{F}_p^n, & \mathbf{b}_i = \sum_{j \in [m]} B_{ij} \cdot \mathbf{v}_j, \\ c_{\mathbf{x}}, c_{\mathbf{k}} \in \mathbb{G}; & \mathbf{c}_i = \sum_{j \in [m]} C_{ij} \cdot \mathbf{v}_j, \\ \{\mathbf{v}_j \in \mathbb{F}_p^n\}_{j \in \mathsf{priv}}, & \bigwedge_{i \in [m]} \mathbf{a}_i \circ \mathbf{b}_i = \mathbf{c}_i \\ r_{\mathbf{x}}, r_{\mathbf{k}} \in \mathbb{F}_p & \wedge \mathsf{Check}(\mathsf{pp}, c_{\mathbf{x}}, \mathbf{x}, r_{\mathbf{x}}) = 1 \\ & \wedge \mathsf{Check}(\mathsf{pp}, c_{\mathbf{k}}, \mathbf{k}, r_{\mathbf{k}}) = 1 \end{array} \right\}, \tag{25}$$

where $\mathsf{A}, \mathsf{B}, \mathsf{C} \in \mathbb{F}_p^{m \times m}$ are the R1CS coefficient matrices for C, with public assignments $\{\mathbf{v}_j\}_{j \in \mathsf{pub}} = \{\mathbf{ctr}, \mathbf{z}\}$, and committed assignments $\{\mathbf{v}_j\}_{j \in \mathsf{cms}} = \{\mathbf{x}, \mathbf{k}\}$.

Taking advantage of the composition property of Π^{Com}, we can even omit the hash-lock h, and instead use the commitment $c_{\mathbf{k}}$ for $\mathbf{k}$ as the lock. Specifically, in the "deliver" phase, $\mathcal{S}$ sends $c_{\mathbf{k}}$ in place of the hash image h, and correspondingly, in the "reveal" phase it reveals a pair of $(k, r_{\mathbf{k}})$. The arbiter $\mathcal{J}$, checks whether

$$\mathsf{Check}(\mathsf{pp}, c_{\mathbf{k}}, (k \cdot \mathbf{1}), r_{\mathbf{k}}) = 1. \tag{26}$$

In ZKCPlus the arbiter $\mathcal{J}$ is instantiated by a smart contract on blockchain whose computational power is limited. To prevent the linear overhead of (26), it may store a pre-computed $G_0 = \langle \mathbf{1}, \mathbf{G} \rangle$, and check the revealed $(k, r_{\mathbf{k}})$ pair by

$$k \cdot G_0 + r_{\mathbf{k}} \cdot H \stackrel{?}{=} c_{\mathbf{k}}. \tag{27}$$

While security of the hash lock is built on the collision- and preimage-resistance of the hash function H, here we rely on the the binding and hiding properties of the Pedersen commitment scheme to ensure that neither $\mathcal{S}$ nor $\mathcal{B}$ can cheat with $c_{\mathbf{k}}$. Refer to Appendix E for more details.

Theorem 4.1. ZKCPlus *is a* secure *ZKCP construction, given that the embedded CP-NIZK argument* Π^{Com} *satisfies* completeness, computational knowledge soundness *and* zero-knowledge.

We include a sketch of the proof for Theorem 4.1 in Appendix E.

4.3 Functionalities and extensions

By incorporating the composition property of CP-NIZK arguments, ZKCPlus can support some functionalities that could be difficult or inefficient for the original ZKCP design.

4.3.1 Compositions of predicates Due to composability of Π^{Com}, it is possible to further split the "validate" phase into multiple rounds. In each round, $\mathcal{B}$ challenges with a predicate $\phi^{(i)}$, possibly dependent on previous predicates, and $\mathcal{S}$ responds with a proof $\pi^{(i)}$ attesting $\phi^{(i)}(\mathbf{x}) = 1$. $\{\pi^{(i)}\}$ jointly validates the relation

$$\bigwedge_i \phi^{(i)}(\mathbf{x}) = 1. \tag{28}$$

Of these predicates, some may be data-parallel while others not. We can use the argument in Section 3 for those are, and other suitable CP-NIZK arguments for those are not, as long as they refer to the same initial commitments.

By this modular approach, ZKCPlus is capable of building up very complicated predicates, or validating a predicate that holds up to a probability threshold. See 5.1 for an example.

4.3.2 Selective transfer With ZKCPlus, it is easy to implement a "selective transfer" mode, where after the "validate" phase, $\mathcal{B}$ decides that it is interested in purchasing only a subset of the bulk $\mathbf{x}$, so it uses a binary mask $\mathbf{b} \in \{0,1\}^n$, marking the wanted part of $\mathbf{x}$ by 1 and the rest by 0. Accordingly, $\mathcal{S}$ runs the "deliver" phase on input of $\mathbf{x}' = \mathbf{x} \circ \mathbf{b}$, combined with an adaptor $\Pi_{\mathsf{link}}^{\mathsf{Com}}$ attesting that $\mathbf{x}'$ is consistent with the original $\mathbf{x}$ s.t. the linking function

$$f(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{b} \circ \mathbf{y} \rangle - \langle \mathbf{x}', \mathbf{y} \rangle = 0 \tag{29}$$

holds for randomly sampled $\mathbf{y} \xleftarrow{\$} \mathbb{F}_p^n$. By Schwartz-Zippel lemma, except for a negligible soundness error $\frac{n}{|\mathbb{F}_p|}$, $f(\mathbf{x}, \mathbf{x}') = 0$ implies that $\mathbf{x}' = \mathbf{x} \circ \mathbf{b}$. Note that (29) is about a sum of two inner vector products, so the argument in Appendix B.2 applies. We include one example for this mode in Section 5.2.

4.3.3 ZKCSP A zero-knowledge contingent *service* payment (ZKCSP) [13] protocol allows payment for digital services rather than digital goods. Take file storage service as an example. $\mathcal{B}$ will pay if $\mathcal{S}$ can prove the integrity of a stored file, without actually retrieving the file. ZKCSP utilizes a pair of *claw-free* hash functions $(\mathsf{H}_1, \mathsf{H}_2)$, for which finding a pair (x_1, x_2) s.t. $\mathsf{H}_1(x_1) = \mathsf{H}_2(x_2)$ is computationally infeasible. For a designated file $\mathbf{x}$ and predicate ϕ s.t. $\phi(\mathbf{x}) = 1$ iff $\mathbf{x}$ is intact, $\mathcal{S}$ selects $k \xleftarrow{\$} \mathbb{F}_p$, computes a hash lock $h \leftarrow \mathsf{H}_1(k)$, and a proof π for the relation

$$\big(\mathsf{H}_1(k) = h \ \wedge \ \phi(\mathbf{x}) = 1\big) \ \vee \ \big(\mathsf{H}_2(k) = h \ \wedge \ \phi(\mathbf{x}) = 0\big). \tag{30}$$

$\mathcal{S}$ sends out tuple (π, h), and $\mathcal{B}$ on verification of π sends out a conditional transaction $\mathsf{tx} = (p, h)_{\mathsf{sig}_{\mathcal{B}}}$ locked on h. If $\mathcal{S}$ reveals a k s.t. $\mathsf{H}_1(k) = h$, $\mathcal{J}$ executes the transaction and finalizes the payment.

Then by skipping the "deliver" phase, ZKCPlus naturally adapts to the ZKCSP scenario. Moreover, in ZKCPlus the initial commitment $c_{\mathbf{x}}$ for $\mathbf{x}$ is reusable for multiple runs of the protocol. This comes in handy for a long-term file storage service, where the integrity of $\mathbf{x}$ has to be checked from time to time.

5 Applications

As an extension of ZKCP, ZKCPlus supports all possible applications of ZKCP. In particular, we describe how it adapts to trading sudoku solutions in Appendix F. But ZKCPlus is not limited to this. With its modular nature, it applies to a broader scope of applications, and some of them are of practical interest. In this section, we develop a comprehensive example of trading a trained CNN model in the MLaaS context. We also explore ZKCPlus's potential application in cloud database systems.

5.1 Pay to CNN model

MLaaS allows a computationally limited client to delegate the heavy tasks of training machine learning models to an untrusted server who has ample computational resources. With prescribed model architecture and hyperparameters, the server does the training, and sells the trained parameters (e.g., weights and biases) as a digital good. The client is willing to pay for the trained model only if it shows sufficiently high accuracy. To achieve fairness, the question arises as how the client can test the model's accuracy without knowing any of the trained parameters.

This becomes easy with ZKCPlus. Take CNN for image classification as an example. First, in the "commit" phase the server publishes the commitment c to the trained parameters of the model. Then in the "validate" phase, the client challenges with some test image. The server runs the trained model on the test image, acquires a classification result, and generates a proof π using a CP-NIZK argument with reference to the commitment c. The client exams the classification result and its accompanying proof, and aborts if either is wrong. Otherwise, the client should be confident that the model applied to the test image yields correct classification. If the client decides that one testing result is not convincing enough, ZKCPlus allows multiple rounds of challenges. The more rounds of successful challenges are performed, the more confident the client will be, until eventually it decides to accept or to abort.

Moreover, we note that many machine learning related computations are data-parallel. Typically, a CNN model is organized as a series of consecutive layers, including convolutional layers, activation layers, pooling layers, fully connected layers and others, whereas each of these layers consists of an array of identical units (i.e., neurons) running in parallel. The CP-NIZK argument for data-parallel computations of Section 3 adapts smoothly to most of these layers.

Convolutional layer Convolutional layer convolves an input feature map with a series of filters, and outputs a new feature map. For simplicity, we write the input feature map as vector $\mathbf{x} \in \mathbb{F}_p^n$, and the filter as vector $\mathbf{t} \in \mathbb{F}_p^k$ of size $k < n$. It works by sliding $\mathbf{t}$ through $\mathbf{x}$, and at each step computing a weighted sum of

$$x_i' = \langle \mathbf{x}_i, \mathbf{t} \rangle, \tag{31}$$

where $\mathbf{x}_i$ marks the subset of $\mathbf{x}$ that overlaps with $\mathbf{t}$ at the i-th step. All x_i' aggregate into the output feature map $\mathbf{x}'$. From $\mathbf{x}$ to $\mathbf{x}'$, the computation is data-parallel, so the argument of Section 3 applies.

Activation layer Take the widely used activation function, *Rectified Linear Unit* (ReLU) as an example. On input of vector $\mathbf{x} \in \mathbb{F}_p^n$, the ReLU layer outputs $\mathbf{x}' \in \mathbb{F}_p^n$ of the same length, whose i-th element is determined by

$$x_i' = \begin{cases} x_i, & \text{if } x_i > 0, \\ 0, & \text{if } x_i \le 0. \end{cases} \tag{32}$$

It is fully data-parallel, so the argument of Section 3 applies.

Pooling layer We consider the widely used pooling layer, *max pooling*, which reduces the size of a feature map by dividing it into small patches and extracting the local maximum of each patch. We assume the usual case of 2×2-pixel patches. For an input feature map $\mathbf{x} \in \mathbb{F}_p^n$, we divide it into 4 segments $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)} \in \mathbb{F}_p^{n/4}$ such that their respective i-th elements $\{x_i^{(0)}, x_i^{(1)}, x_i^{(2)}, x_i^{(3)}\}$ constitute a 2×2-pixel patch. Local maximums of these 2×2-pixel patches constitute a quarter-sized output feature map $\mathbf{x}' \in \mathbb{F}_p^{n/4}$, whose i-th element is

$$x_i' = \max(x_i^{(0)}, x_i^{(1)}, x_i^{(2)}, x_i^{(3)}). \tag{33}$$

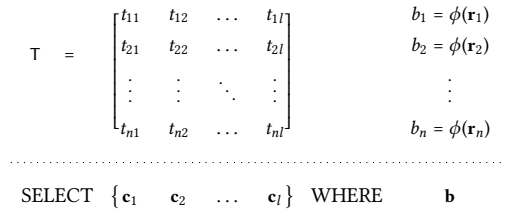

Figure 3: Database table modeled as a matrix

From $\{\mathbf{x}^{(i)}\}_{i\in[0,3]}$ to $\mathbf{x}'$, the computation is data-parallel, so the argument of Section 3 applies. To justify the consistency between $\mathbf{x}$ and $\{\mathbf{x}^{(i)}\}_{i\in[0,3]}$, we employ an adaptor $\Pi^{\mathsf{Com}}_{\mathsf{link}}$ for interlinking function

$$f(\mathbf{x}, \mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}) = \sum_{i\in[0,3]} \langle \mathbf{x}^{(i)}, \mathbf{y}^{(i)} \rangle - \langle \mathbf{x}, \mathbf{y} \rangle = 0, \tag{34}$$

where $\mathbf{y} \xleftarrow{\$} \mathbb{F}_p^n$ is a randomly sampled challenge vector, and it breaks into segments $\{\mathbf{y}^{(i)}\}_{i\in[0,3]}$ in the same way as $\mathbf{x}$ breaks into $\{\mathbf{x}^{(i)}\}_{i\in[0,3]}$. Except for negligible error $\frac{n}{|\mathbb{F}_p|}$, $f(\mathbf{x}, \mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}) = 0$ holds iff $\{\mathbf{x}^{(i)}\}_{i\in[0,3]}$ are consistent with $\mathbf{x}$. Note (34) is a sum of inner products, the argument in Appendix B.2 applies.

Fully connected layer In essence, fully connected layer performs matrix-vector product of a weight matrix T and a flattened feature map $\mathbf{x}$, outputting $\mathbf{x}' = \mathsf{T}\mathbf{x}$. We can reduce the matrix-vector product to a group of inner products, and employ the CP-NIZK argument in Appendix B.2 to prove it. Though not data-parallel, this argument is composable with the data-parallel argument for the previous layer via a shared commitment to $\mathbf{x}$.

Based on these observations, in Section 6.4.1 we develop and evaluate an end-to-end demonstration for exchanging CNN models [40, 50]. Alternatively, one may imagine an application of exchanging images using a pre-trained model acting as the predicate ϕ, where $\mathcal{B}$ only pays for images classified as a particular type by the pre-trained model. Similar applications extend to other valuable digital assets, such as documents, videos, and software.

5.2 Pay to SQL query

ZKCPlus and the data-parallel CP-NIZK argument also find their potential applications in could databases systems. A typical scenario is that a client outsources a database to an untrusted server, and then the server responds to the client's queries [58, 59]. For fairness, the client only pays for correct query results.

As an explanatory example, we consider a database table modeled as a matrix of finite field elements $\mathsf{T} \in \mathbb{F}_p^{n\times l}$. For ZKCPlus, in the "commit" phase the table is committed by each column $\mathbf{c}_j \in \mathbb{F}_p^n$ for $j \in [l]$. In the "validate" phase, the clients sends a SQL query

SELECT [columns] WHERE [predicate],

which specifies the desired columns $\mathbf{c}_j$, with a "WHERE" clause defining a predicate ϕ that restricts the rows used for computing the query result. More specifically, ϕ runs on each row $\mathbf{r}_i$ and outputs a binary decision

$$b_i = \phi(\mathbf{r}_i) = \phi(t_{i1}, \ldots, t_{il}). \tag{35}$$

As illustrated in Figure 3, applying the "WHERE" clause to each row of the table is data-parallel, so we use the argument described in Section 3 with reference to the initial commitments to columns $\mathbf{c}_j$. In practice the "WEHER" clause usually consists of a sequence of sub-clauses connected by logical operators such as "AND" and "OR", in which case the argument's composability comes in handy. The results b_i aggregate into a binary mask $\mathbf{b} \in \{0,1\}^n$. Then in the "deliver" phase, the server runs "proof of delivery" on selected set $\mathbf{c}'_j = \mathbf{c}_j \circ \mathbf{b}$.

In Section 6.4.2 we evaluate a minimal demo of sub-string query on a table containing one single column of string type. Apart from the basic "SELECT" query, it is also possible for ZKCPlus to support other SQL queries such as "COUNT", "SUM", etc.

6 Evaluation

In this section we give a brief description of our implementation, and present comprehensive evaluations on its performance.

6.1 Implementation

We have built a fair-exchange platform on the testnet of Ethereum with ZKCPlus. It consists of the following three main components.

- ZKCPlus-lib, a core library of our CP-NIZK argument scheme shipping with Golang bindings, 23k LoC in C++.
- ZKCPlus-node, a node application dealing with communication, smart contract invoking, etc., 35k LoC in Golang.
- ZKCPlus-contract, a smart contract for payment management and commitment opening, 900 LoC in Solidity.

Our ZKCPlus implementation is built with the mcl library [33], using a BN_SNARK1 curve which has the same parameters as the BN128 curve of the libSNARK library [38] used by ZKCP. We utilize Pippenger's algorithm [48] for efficient multi-scalar multiplication, which has $O(n/\log n)$ computational complexity w.r.t. vector length n. According to our benchmark, our mcl library [33] based multi-scalar multiplication implementation shows comparable efficiency as the libSNARK library [38] used by ZKCP, with the relative ratio ranging from 0.8 to 1.4 as n varies from 2^5 to 2^{14}. As for the block cipher, since the BN_SNARK1 curve has a 254-bit prime order p, we use 110 rounds of MiMC-p/p with the exponent of the non-linear permutation function $d = 5$ to thwart interpolation attacks [36].

We have realized the three applications described in Section 5, i.e., pay to sudoku solutions, pay to CNN models, and pay to SQL query results.

6.2 Experimental setup

For a direct comparison between ZKCPlus and ZKCP, we measure their performances in the application of "pay to sudoku solution". We compare their respective setup cost, running time and communication overhead. Then we make some modifications to the original implementations of the two protocols to compare their respective end-to-end throughput for varied-size data and prover efficiency of their underlying zero-knowledge schemes. The experiments are carried out with both of the seller and the buyer running on Ubuntu 18.04 with 3.2 GHz CPU and 32 GB memory. Considering the ZKCP implementation runs single-threaded, we limit ZKCPlus to one thread as well.

Besides, we also evaluate the applications of ZKCPlus in "pay to CNN model" and "pay to SQL query" scenarios. Here we shift to another experimental setup, where the seller (server) runs with 24 threads on a machine with 2.9 GHz CPU and 256 GB memory, and

Table 3: Setup cost of ZKCP and ZKCPlus

	Setup time		CRS size	
Solution size	ZKCP	ZKCPlus	ZKCP	ZKCPlus
9×9	17.88 s	0.03 s	30.88 MB	1.02 MB
16×16	39.18 s		69.75 MB	
25×25	87.21 s		165.67 MB	
36×25	180.12 s		336.61 MB	
49×49	322.54 s		635.93 MB	

Table 4: Seller running time of ZKCP and ZKCPlus in "validate" and "deliver" phases

	Validate phase		Deliver phase	
Solution size	ZKCP	ZKCPlus	ZKCP	ZKCPlus
9×9	0.12 s	0.08 s	5.31 s	0.19 s
16×16	0.53 s	0.14 s	11.43 s	0.26 s
25×25	1.92 s	0.33 s	28.46 s	0.41 s
36×25	5.26 s	0.54 s	56.09 s	0.52 s
49×49	13.61 s	1.07 s	110.33 s	0.78 s

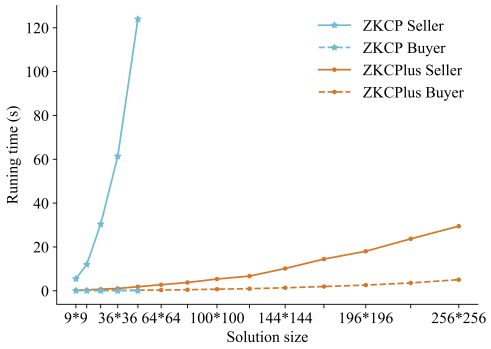

Figure 4: Running time of ZKCP and ZKCPlus in "validate" and "deliver" phases

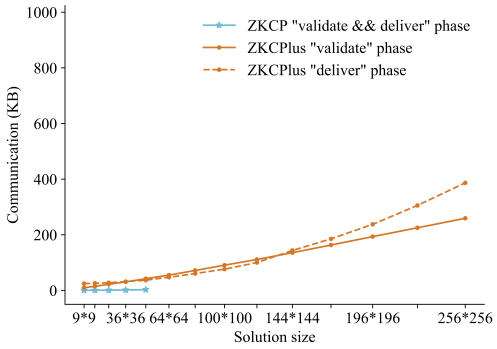

Figure 5: Communication cost of ZKCP and ZKCPlus in "validate" and "deliver" phases

the buyer (client) with 6 threads on the aforementioned machine with 3.2 GHz CPU and 32 GB memory. We evaluate the two applications in the running time of the seller and the buyer respectively, as well as the communication cost between them. We also show other metrics (i.e., gas cost of our smart contract) in Appendix G.

6.3 Comparison with ZKCP

6.3.1 Performance on "pay to sudoku solution" We first compare the two protocols in the application of "pay to sudoku solution", considering their respective setup cost, runtime overhead and communication cost, with the size of sudoku solutions varying from 9×9 to 256×256.

Setup cost In setup phase, ZKCPlus samples a vector of group elements (with length $n = 2^{14}$ by default) as public and circuit-independent parameters, while ZKCP yields a circuit-dependent CRS involving some secret randomnesses. In Table 3 we list the running time of setup phase and the size of generated CRS of ZKCP and ZKCPlus for trading varied-size sudoku solutions respectively. Note that ZKCP runs out of memory when the sudoku gets larger than 49×49, so we only show the statistics we collected on the smaller ones.

As Table 3 shows, ZKCPlus outperforms ZKCP in both setup time and CRS size. The setup phase of ZKCPlus is extremely lightweight with only tens of milliseconds running time and about 1 MB public parameters, while for ZKCP, the setup time and CRS size rise up greatly as circuit size grows. For example, the generated CRS reaches up to 635.93 MB for trading a 49×49 sudoku solution.

Runtime overhead We focus on the running time of the "validate" and "deliver" phases, given that they are the dominating factors for both protocols. In Figure 4, we show the total running time of these two phases respectively for the seller and the buyer.

As shown in Figure 4, ZKCP incurs extremely asymmetrical overheads for the two trading parties. The buyer is quite efficient but the seller's running time skyrockets. For a 49×49 sudoku solution, it takes about 124 s for the seller to generate a proof, but only a few milliseconds for the buyer to verify it. Also, ZKCP is memory-consuming, that the seller runs out of memory when solution size gets larger than 49×49.

By contrast, ZKCPlus reduces seller running time by 21-67×. Specifically, for the 49×49 sudoku solution, the two phases only take the seller about 1.8 s to run. Meanwhile, the buyer of ZKCPlus shows competitive efficiency as that of ZKCP. ZKCPlus also demonstrates higher scalability; For the largest 256×256 (64 KB) sudoku solution, the two phases finish within about 35 s.

More detailed statistics on seller running time of the two phases are shown in Table 4. For ZKCP, we calculate the running time of the two phases according to their corresponding number of constraints in the circuit. ZKCPlus seller is 1.6-12.7× more efficient than ZKCP in "validate" phase, and 28.7-142.6× more efficient in "deliver" phase.

Communication cost We show communication cost between the two parties in "validate" and "deliver" phases in Figure 5. The proof size in ZKCP is succinct due to the underlying zkSNARK scheme. However, the overall communication cost still grows linearly due to its linear-size ciphertext. (Due to memory limitations, the experiment cuts at 49× 49 solution size). Although the proof size of our CP-NIZK argument is larger and not constant, it won't be a bottleneck since the cost is acceptable for off-chain communications.

6.3.2 Throughput of large-scale delivery To further evaluate scalability of the two protocols, we measure their throughput for

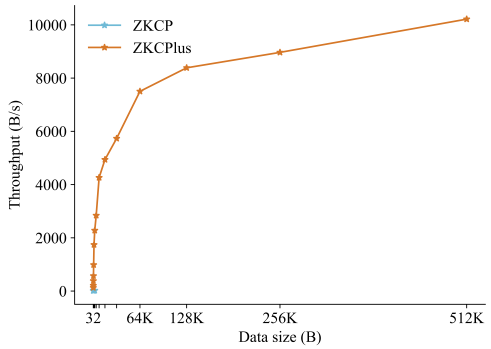

Figure 6: Throughput of ZKCP and ZKCPlus in "deliver" phase

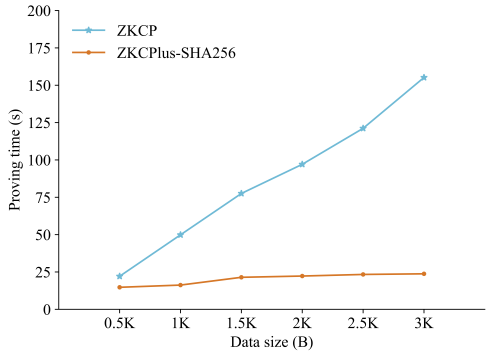

Figure 7: Proving time of ZKCP and ZKCPlus-SHA256 in "deliver" phase (under the same encryption scheme)

Table 5: Performance of "pay to CNN model" application on two CNN models

		Seller time	Buyer time	Communication cost
3-layer CNN	Validate phase	0.79 s	0.14 s	51.35 KB
	Deliver phase	0.95 s	0.11 s	211.47 KB
VGG16 model	Validate phase	6.15 min	56.97 s	80.71 KB
	Deliver phase	6.13 min	18.50 s	298.10 MB

Table 6: Performance of "pay to SQL query" application on a sub-string query over 100,000 records

	Seller time	Buyer time	Communication cost
Validate phase	5.70 s	0.77 s	988.18 KB
Deliver phase	0.31 s	0.08 s	96.61 KB

large-sacle data delivery. For simplicity, we remove solution validation from ZKCP, and compare its throughput to that of ZKCPlus in "deliver" phase only, with the data size varying from 32 B to 512 KB. We define throughput by $\frac{D}{T}$, where D and T denote the size of the delivered data and the total running time of the two parties in "deliver" phase. We show our statistics in Figure 6.

Again, ZKCP runs out of memory when data size exceeds 4 KB; for 2 KB data, its throughput gets to about 22 B/s. In contrast, ZKCPlus shows a throughput of 2278 B/s for 2 KB data, 104× higher than that of ZKCP. Note that the throughput of ZKCPlus scales logarithmically with data size, since the most costly operation multi-scalar multiplication has $O(n/\log n)$ computational complexity.

6.3.3 Prover efficiency of CP-NIZK argument For a better illustration of how the prover efficiency of our CP-NIZK argument compares to the zkSNARK used by ZKCP, we replace the MiMC block cipher in ZKCPlus with the same encryption scheme adopted by ZKCP, i.e., a stream cipher constructed by running a SHA256-based pseudo-random function in CTR mode. We denote this implementation of ZKCPlus by "ZKCPlus-SHA256". Then we encrypt varied-size data and measure proving time for encryption subroutine in ZKCP and ZKCPlus-SHA256 respectively. Again, for fair comparison, we run ZKCPlus-SHA256 with one single thread.

Limited by ZKCP, we can only collect statistics on data smaller than 4 KB. As shown in Figure 7, under the same encryption scheme, the prover of ZKCPlus-SHA256 is still more efficient than that of ZKCP. The tendency of proving time varying with data size of the two protocols both roughly conform with the computational complexity of multi-scalar multiplication, i.e., $O(n/\log n)$, but the constant of ZKCPlus-SHA256 is much smaller. By a linear fit, we can make a rough estimation that ZKCPlus-SHA256 prover has a 13× smaller constant than ZKCP, indicating that our CP-NIZK argument is an order of magnitude faster in proving time than the zkSNARK scheme adopted by ZKCP. (As mentioned earlier, the differences from the underlying multi-scalar multiplication implementations account for only 0.8-1.4×, which is negligible.)

6.4 Performance of ZKCPlus applications

6.4.1 Pay to CNN model We evaluate the "pay to CNN model" application on two neural network models, a simple 3-layer CNN for hand-written digit recognition, and a well-known deep CNN VGG16 [50] for image classification. The 3-layer CNN is composed of one convolutional layer with Relu activation and 2×2 max pooling by stride size 2, and two fully-connected layers. We train the model on the MNIST dataset [40], with an input size of 28×28, and a total number of 8,620 parameters. The VGG16 model is far more complicated, covering all the representative components described in Section 5, mainly consisting of thirteen convolutional layers and three fully-connected layers. We train the VGG16 model on a 10-class colored image dataset CIFAR-10 [37], with a 32×32 input size and up to 14,991,946 parameters. We use fixed-point arithmetics for the both models.

After training the two models on the server machine, we measure the running time and communication cost of the server and the client in "validate" and "deliver" phases for trading the models. Without loss of generality, here we only show our statistics on one test image case. In practice the buyer may challenge the seller with several test images in "validate" phase.

As shown in Table 5, for the 3-layer CNN, the two phases finish for either the server or the client within 1 s, and incur only 51.35 KB and 211.47 KB communication cost respectively. For the VGG16 network which involves about fifteen million parameters, it takes the seller 6.15 min and the buyer 56.97 s to prove/verify an inference process on one test image. In the later "deliver" phase it takes the seller 6.13 min and the buyer 18.50 s to generate/verify the proof

of delivery for all the fifteen million parameters. The proof size in "validate" phase is about 80.71 KB, and the ciphertext and proof of delivery in "deliver" phase is less than 300 MB.

In fact our CP-NIZK argument shows a great advancement in verifiable inference computation of CNN. Previous study [41] has evaluated Groth16 zkSNARK scheme on the LeNet-5 [39] CNN which has sixty thousand parameters. On a machine with 3.4 GHz CPU and 32 GB memory, it takes about 1.5 h to generate a 11 GB CRS, and 45 min to prove an inference process. In contrast, for a much larger model that has 250× more parameters, ZKCPlus can prove an inference process within 7.5× less time and negligible setup time and CRS size.

6.4.2 Pay to SQL query We construct a database storing a table of 100,000 31-byte strings, and measure the performance of the "pay to SQL query" application on trading subset of the strings that contain a specific 5-byte sub-string. We list the statistics in Table 6.

In "validate" phase the seller executes the "SELECT" query and constructs an argument for the selection process. It only takes the seller about 8.67 s to build the proof for selection query over all the 100,000 records, and the buyer 0.77 s to verify it, with less than 1 MB communication cost. There are 402 entries selected out, and in "deliver" phase it takes both the two parties less than 1 s to generate/verify the proof of delivery, with less than 100 KB communication cost.

7 Related Work

7.1 Blockchain-based fair-exchange protocols

Although it is well recognized that without further assumptions fair exchange is unachievable without the aid of a trusted third party [45], the emergence of blockchain networks such as Bitcoin and Ethereum offers a possible workaround, where instead of centralized third parties the participants can place their trusts in the decentralized networks [21, 23, 51, 52].

The ZKCP [43] protocol is a seminal work in achieving fairness between two trustless parties, which uses blockchain-based scripts as the arbiter for the exchange. However it faces problems of low practicality and scalability. Another interesting fair-exchange protocol based on blockchain, FairSwap [23], aims at reducing the parties' computational cost by avoiding the use of heavy zero-knowledge proofs. It instead adopts a lightweight proof of misbehavior and allows a cheated buyer to issue a dispute to the arbiter (smart contract) with the misbehavior proof in order to restore fairness. Clearly, it offers "weak" fairness [3, 45] compared to ZKCP and ZKCPlus. Besides, the on-chain gas cost of verifying a misbehavior proof scales up with the proof size, severely limiting its practicality and scalability. As a contrast, ZKCP and ZKCPlus offer $O(1)$ on-chain gas cost.

7.2 Zero-knowledge proofs

The field of zero knowledge proofs [30] are undergoing fast developments. Many efficient protocols and systems emerge in the recent years. Categorized by their underlying techniques, there are pairing-based succinct NIZK argument of knowledge (zkSNARK) schemes [17, 26, 27, 32, 42, 46, 55], discrete logarithm-based proofs [10, 11, 31, 49, 53], interactive oracle proofs [5, 6, 18, 57], MPC-in-the-head-based proofs [2, 15, 28, 34, 35] and so on. They depend on different security assumptions, and offer different trade-offs between prover time, verifier time and proof size. The original ZKCP design uses pairing-based arguments [7, 46]. To avoid the problematic trusted setup phase, we switch to discrete logarithm-based schemes that only require public setup. In particular, our construction largely draws on the techniques proposed by [31].

We build our argument for data-parallel computations on top of R1CS, an extension of quadratic arithmetic programs [27]. Another choice is the GKR protocol [19, 29], which offers asymptotically optimal prover time for data-parallel computations. There have been several GKR protocol-based argument schemes [53, 55, 57]. Traditionally this approach assumes layered arithmetic circuits, which significantly limits its scope of application and is not fit for our protocol as well. Recently, [56] demonstrates how this restriction can be removed, making GKR-based arguments a promising direction to explore in the future. By incorporating their prover efficiency, we may further boost ZKCPlus's throughput.

Another interesting direction to explore involves interactive oracle proofs. There have been quite a few ready-made protocols for both R1CS [5, 6, 18] and GKR protocol [57]. They use lightweight cryptographic primitives (mainly collision-resistant hash functions), and feature high prover efficiency, full transparency, and plausible post-quantum security. Their main disadvantage is relatively large proof size, which is not a big problem for ZKCPlus.

8 Conclusion

In this paper we propose a practical and flexible fair-exchange protocol ZKCPlus, as an optimization and extension of ZKCP. We observed that the ZKCP protocol incurs a problematic trusted setup phase and heavy proving overhead, and is not practical for complicated data validations. Thereby we design a prover-efficient CP-NIZK argument for data-parallel computations, which requires only public setup. With this argument we can relieve the protocol of the costly setup phase, and in combination with circuit-friendly cryptographic primitives, we are able to significantly reduce the seller's proving overhead; its composable nature furthermore allows the protocol to validate complicated predicates in a flexible modular manner. Our evaluations show that ZKCPlus significantly outperforms the ZKCP protocol, with both much lighter weighted setup phase and higher overall throughput. We also demonstrate that with ZKCPlus, we are able to build applications that are of practical interests.

Acknowledgement

This work was supported by: donations from Nervos Foundation and HashKey, National Natural Science Foundation of China under Grant 61772308, 61972224 and U1736209, and BNRist Network and Software Security Research Program under Grant BNR2019TD01004 and BNR2019RC01009. Yupeng Zhang is supported by DARPA under Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

References

[1] Martin R Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In *Advances in Cryptology-ASIACRYPT 2016-22nd International Conference on the Theory and Application of Cryptology and*

Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. 191–219.

[2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2087–2104.

[3] N Asokan. 1998. Fairness in electronic commerce. (1998).

[4] Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: a paradigm for designing efficient protocols. *1st ACM Conference on Computer and Communications Security* November 1993 (1993), 62–73.

[5] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive* 2018 (2018), 46.

[6] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. 2019. Aurora: Transparent succinct arguments for R1CS. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 103–128.

[7] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct non-interactive zero knowledge for a von Neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 781–796.

[8] Sean Bowe. 2016. pay-to-sudoku. http://diyhpl.us/~bryan/papers2/bitcoin/zkcp3.pdf.

[9] Sean Bowe. 2016. pay-to-sudoku. https://github.com/zcash-hackworks/pay-to-sudoku.

[10] Sean Bowe, Jack Grigg, and Daira Hopwood. 2019. Halo: Recursive Proof Composition without a Trusted Setup. *IACR Cryptol. ePrint Arch.* 2019 (2019), 1021.

[11] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 315–334.

[12] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2075–2092.

[13] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. 2017. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 229–243.

[14] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. 2002. Universally composable two-party and multi-party secure computation. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 494–503.

[15] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. 2017. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1825–1842.

[16] Melissa Chase, Chaya Ganesh, and Payman Mohassel. 2016. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In *Annual International Cryptology Conference*. Springer, 499–530.

[17] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 738–768.

[18] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. 2020. Fractal: Post-quantum and transparent recursive proofs from holography. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 769–793.

[19] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. 90–112.

[20] John D'Arcy, Anat Hovav, and Dennis Galletta. 2009. User awareness of security countermeasures and its impact on information systems misuse: A deterrence approach. *Information systems research* 20, 1 (2009), 79–98.

[21] Sergi Delgado-Segura, Cristina Pérez-Solà, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí. 2017. A fair protocol for data trading based on Bitcoin transactions. *Future Generation Computer Systems* (2017).

[22] Kevin Dooley. 2001. *Designing Large Scale Lans: Help for Network Designers*. " O'Reilly Media, Inc.".

[23] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. 2018. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 967–984.

[24] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 186–194.

[25] Georg Fuchsbauer. 2019. WI is not enough: Zero-knowledge contingent (service) payments revisited. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 49–62.

[26] Ariel Gabizon and Zachary J Williamson. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. (2019).

[27] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic span programs and succinct NIZKs without PCPs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 626–645.

[28] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. 2016. Zkboo: Faster zero-knowledge for boolean circuits. In *25th USENIX Security Symposium (USENIX Security 16)*. 1069–1083.

[29] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. 2015. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)* 62, 4 (2015), 27.

[30] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989), 186–208.

[31] Jens Groth. 2009. Linear algebra with sub-linear zero-knowledge arguments. In *Annual International Cryptology Conference*. Springer, 192–208.

[32] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 305–326.

[33] herumi. 2015. mcl library. https://github.com/herumi/mcl.

[34] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2007. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM, 21–30.

[35] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. 2008. Founding cryptography on oblivious transfer–efficiently. In *Annual International Cryptology Conference*. Springer, 572–591.

[36] Thomas Jakobsen and Lars R. Knudsen. 1997. The Interpolation Attack on Block Ciphers. In *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1267)*, Eli Biham (Ed.). Springer, 28–40. https://doi.org/10.1007/BFb0052332

[37] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[38] Scipr Lab. 2012. libsnark. https://github.com/scipr-lab/libsnark.

[39] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[40] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[41] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2020. vCNN: Verifiable Convolutional Neural Network. *IACR Cryptol. ePrint Arch.* 2020 (2020), 584.

[42] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 2019. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2111–2128.

[43] Gregory Maxwell. 2011. Zero Knowledge Contingent Payment. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment.

[44] Satoshi Nakamoto et al. 2008. Bitcoin: A peer-to-peer electronic cash system.

[45] Henning Pagnia and Felix C Gärtner. 1999. *On the impossibility of fair exchange without a trusted third party*. Technical Report. Technical Report TUD-BS-1999-02, Darmstadt University of Technology.

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

[47] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*. Springer, 129–140.

[48] Nicholas Pippenger. 1980. On the evaluation of powers and monomials. *SIAM J. Comput.* 9, 2 (1980), 230–250.

[49] Srinath Setty. 2020. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*. Springer, 704–737.

[50] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[51] Peter Todd and Amir Taaki. 2014. Paypub: Trustless payments for information publishing on bitcoin. Github Project.

[52] Florian Tramer, Fan Zhang, Huang Lin, Jean-Pierre Hubaux, Ari Juels, and Elaine Shi. 2017. Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 19–34.

[53] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. 2018. Doubly-Efficient zkSNARKs Without Trusted Setup. In *2018 IEEE Symposium on Security and Privacy (SP)*. 926–943. https://doi.org/10.1109/SP.2018.00060

[54] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[55] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover

Computation. In *Advances in Cryptology – CRYPTO 2019*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer International Publishing, Cham, 733–764.
[56] Jiaheng Zhang, Weijie Wang, Yinuo Zhang, and Yupeng Zhang. 2020. *Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time.* Technical Report. Cryptology ePrint Archive, Report 2020/1247, 2020. https://eprint. iacr. org ….
[57] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2020. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 859–876.
[58] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 863–880.
[59] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. A Zero-Knowledge Version of vSQL. (2017).

A Formal definitions

A.1 Discrete logarithm relation assumption

Given a security parameter λ, for every polynomial-time adversary $\mathcal{A}$ and for all $n \geq 1$, the discrete logarithm relation assumption [11] assumes that

$$\Pr\left[\begin{array}{l|l} \exists\, a_i \neq 0 \wedge & \mathbb{G} \leftarrow \mathsf{Setup}(1^\lambda), \\ \sum a_i \cdot G_i = 0 & G_1, \ldots, G_n \xleftarrow{\$} \mathbb{G}, \\ & a_1, \ldots, a_n \leftarrow \mathcal{A}(G_1, \ldots, G_n) \end{array}\right] \leq \mathsf{negl}(\lambda). \tag{36}$$

where $\mathsf{negl}(\lambda)$ is a negligible function in the security parameter λ. We say $\sum_{i\in[n]} a_i \cdot G_i = 0$ is a non-trivial discrete logarithm relation between $G_1, \ldots, G_n$. The discrete logarithm relation assumption states that an adversary cannot find a non-trivial relation between randomly chosen group elements. It degenerates to the discrete logarithm assumption when $n = 1$.

A.2 Properties of Commitment schemes

A secure commitment scheme $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Check})$ must satisfy the notion of *correctness*, *binding* and *hiding*.

Correctness: for any $u \in \mathsf{M}_{\mathsf{pp}}$, there is

$$\Pr\left[\begin{array}{l|l} \mathsf{Check}(\mathsf{pp}, c, u, r) = 1 & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ & (c, r) \leftarrow \mathsf{Commit}(\mathsf{pp}, u) \end{array}\right] = 1. \tag{37}$$

Binding: Com is *computationally binding* if for every polynomial-time adversary $\mathcal{A}$

$$\Pr\left[\begin{array}{l|l} \mathsf{Check}(\mathsf{pp}, c, u_0, r_0) = 1 \wedge & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathsf{Check}(\mathsf{pp}, c, u_1, r_1) = 1 \wedge & (c, u_0, r_0, u_1, r_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ u_0 \neq u_1 & \end{array}\right] \leq \mathsf{negl}(\lambda). \tag{38}$$

Hiding: Com is *perfectly hiding* if for every polynomial-time adversary $\mathcal{A}$

$$\Pr\left[\begin{array}{l|l} \mathcal{A}(\mathsf{pp}, c) = b & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ & (u_0, u_1) \leftarrow \mathcal{A}(\mathsf{pp}), \\ & b \xleftarrow{\$} \{0, 1\}, \\ & (c, r) \leftarrow \mathsf{Commit}(\mathsf{pp}, u_b) \end{array}\right] = \frac{1}{2}. \tag{39}$$

A.3 Properties of interactive arguments of knowledge

The key security properties for an interactive argument of knowledge $\mathsf{Arg} = (\mathsf{KeyGen}, \mathcal{P}, \mathcal{V})$ are

Completeness: Arg has *perfect completeness* if for every polynomial time adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l|l} (s, w) \notin \mathsf{R} \vee & \sigma \leftarrow \mathsf{KeyGen}(1^\lambda), \\ \langle \mathcal{P}(\sigma, s, w), \mathcal{V}(\sigma, s; \rho)\rangle = 1 & (s, w) \leftarrow \mathcal{A}(\sigma) \end{array}\right] = 1. \tag{40}$$

Witness-extended emulation: Arg has *computational witness-extended emulation* if for all deterministic polynomial-time $\mathcal{P}^*$, there exists a polynomial-time emulator $\mathcal{E}$ s.t. for every pair of polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\left|\begin{array}{l} \Pr\left[\begin{array}{l|l} \mathcal{A}_1(\mathsf{tr}) = 1 & \sigma \leftarrow \mathsf{KeyGen}(1^\lambda), \\ & (s, w) \leftarrow \mathcal{A}_2(\sigma), \\ & \mathsf{tr} \leftarrow \langle \mathcal{P}^*(\sigma, s, w), \mathcal{V}(\sigma, s)\rangle \end{array}\right] \\ - \Pr\left[\begin{array}{l|l} \mathcal{A}_1(\mathsf{tr}) = 1 \wedge & \sigma \leftarrow \mathsf{KeyGen}(1^\lambda), \\ \mathsf{tr}\ \text{accepting} \Rightarrow & (s, w) \leftarrow \mathcal{A}_2(\sigma), \\ (s, w) \in \mathsf{R} & (\mathsf{tr}, w) \leftarrow \mathcal{E}^{\mathcal{O}}(\sigma, s) \end{array}\right] \end{array}\right| \leq \mathsf{negl}(\lambda), \tag{41}$$

where $\mathcal{E}$ is given oracle access to $\mathcal{O} = \langle \mathcal{P}^*(\sigma, s, w), \mathcal{V}(\sigma, s)\rangle$, which permits rewinding to a specific point and resuming with fresh verifier randomness from this point onwards.

Public-coin: Arg is said to be *public-coin* is all messages sent by $\mathcal{V}$ are sampled uniformly at random and independent of $\mathcal{P}$'s messages.

Special honest-verifier zero knowledge: Arg has *perfect special honest-verifier zero knowledge* (PSHVZK) if for every pair of polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and $(s, w) \in \mathsf{R}$, there exists a probablistic polynomial-time simulator Sim s.t.

$$\begin{aligned} &\Pr\left[\begin{array}{l|l} \mathcal{A}_1(\mathsf{tr}, \sigma, s) = 1 & \sigma \leftarrow \mathsf{KeyGen}(1^\lambda), \\ & (s, w, \rho) \leftarrow \mathcal{A}_2(\sigma), \\ & \mathsf{tr} \leftarrow \langle \mathcal{P}(\sigma, s, w), \mathcal{V}(\sigma, s; \rho)\rangle \end{array}\right] \\ &= \Pr\left[\begin{array}{l|l} \mathcal{A}_1(\mathsf{tr}, \sigma, s) = 1 & \sigma \leftarrow \mathsf{KeyGen}(1^\lambda), \\ & (s, w, \rho) \leftarrow \mathcal{A}_2(\sigma), \\ & \mathsf{tr} \leftarrow \mathsf{Sim}(\sigma, s, \rho) \end{array}\right]. \end{aligned} \tag{42}$$

A.4 Properties of non-interactive zero-knowledge arguments of knowledge

The key security properties for a non-interactive zero-knowledge argument of knowledge Π are

Completeness: Π has *perfect completeness* if for every $(s, w) \in \mathsf{R}$,

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\sigma, s, \pi) = 1 & \sigma \leftarrow \mathsf{KeyGen}(1^\lambda), \\ & \pi \leftarrow \mathsf{Prove}(\sigma, s, w) \end{array}\right] = 1 \tag{43}$$

Knowledge soundness: Π has *computational knowledge soundness* if for every polynomial-time adversary $\mathcal{A}$ there exists a polynomial-time extractor Ext s.t.

$$\Pr\left[\begin{array}{l|l} (s, w) \notin \mathsf{R} \wedge & \sigma \leftarrow \mathsf{KeyGen}(1^\lambda), \\ \mathsf{Verify}(\sigma, s, \pi) = 1 & (s, \pi; w) \leftarrow (\mathcal{A}\|\mathsf{Ext})(\sigma) \end{array}\right] \leq \mathsf{negl}(\lambda), \tag{44}$$

Zero-knowledge: Π has *perfect zero-knowledge* if for every pair of $(s, w) \in \mathsf{R}$ and every adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a simulator Sim that

$$\begin{aligned} &\Pr\left[\begin{array}{l|l} (s, w) \in \mathsf{R} \wedge & \sigma \leftarrow \mathsf{KeyGen}(1^\lambda), \\ \mathcal{A}_1(\sigma, s, \pi) = 1 & (s, w) \leftarrow \mathcal{A}_2(\sigma), \\ & \pi \leftarrow \mathsf{Prove}(\sigma, s, w) \end{array}\right] \\ &= \Pr\left[\begin{array}{l|l} (s, w) \in \mathsf{R} \wedge & \sigma \leftarrow \mathsf{KeyGen}(1^\lambda), \\ \mathcal{A}_1(\sigma, s, \pi) = 1 & (s, w) \leftarrow \mathcal{A}_2(\sigma), \\ & \pi \leftarrow \mathsf{Sim}(\sigma, s) \end{array}\right]. \end{aligned} \tag{45}$$

In random oracle model, Sim is allowed to run a partial simulation of the random oracle [4].

B Arguments for inner products

The arguments in this section all refer to the Pedersen commitment scheme in Section 2.2.

B.1 Inner product

We first consider a simple case that with commitments $L,\ R \in \mathbb{G}$ to vectors $\mathbf{l}, \mathbf{r} \in \mathbb{F}_p^n$ and commitment $T \in \mathbb{G}$ to scalar $t \in \mathbb{F}_p$, a prover $\mathcal{P}$ wants to demonstrate to a verifier $\mathcal{V}$ that $\langle \mathbf{l}, \mathbf{r} \rangle = t$. In Figure 8 we show a simple protocol for the relation

$$\left\{ \begin{array}{c} L, R, T \in \mathbb{G}; \\ \mathbf{l}, \mathbf{r} \in \mathbb{F}_p^n, t \in \mathbb{F}_p, \\ \iota, \rho, \tau \in \mathbb{F}_p \end{array} \middle| \begin{array}{c} \mathsf{Check}(\mathsf{pp}, L, \mathbf{l}, \iota) = 1 \ \wedge \\ \mathsf{Check}(\mathsf{pp}, R, \mathbf{r}, \rho) = 1 \ \wedge \\ \mathsf{Check}(\mathsf{pp}, T, t, \tau) = 1 \ \wedge \\ \langle \mathbf{l}, \mathbf{r} \rangle = t \end{array} \right\}. \tag{46}$$

It has *completeness, witness-extended emulation* and *zero knowledge*. In particular, it achieves *zero knowledge* by introducing randomly sampled mask vectors $\mathbf{l}_d$ and $\mathbf{r}_d$.

Public input: commitments $L, R, T \in \mathbb{G}$

$\mathcal{P}$'s private input: $\mathbf{l}, \mathbf{r} \in \mathbb{F}_p^n$, $t \in \mathbb{F}_p$, blinders $\iota, \rho, \tau \in \mathbb{F}_p$

$$\begin{aligned}
&\mathcal{P}: \mathbf{l}_d, \mathbf{r}_d \xleftarrow{\$} \mathbb{F}_p^n, \\
&\quad (L_d, \iota_d) \leftarrow \mathsf{Commit}(\mathsf{pp}, \mathbf{l}_d), \quad (R_d, \rho_d) \leftarrow \mathsf{Commit}(\mathsf{pp}, \mathbf{r}_d), \\
&\quad s_1 = \langle \mathbf{l}, \mathbf{r}_d \rangle + \langle \mathbf{l}_d, \mathbf{r} \rangle, \quad s_2 = \langle \mathbf{l}_d, \mathbf{r}_d \rangle, \\
&\quad (S_1, \sigma_1) \leftarrow \mathsf{Commit}(\mathsf{pp}, s_1), \quad (S_2, \sigma_2) \leftarrow \mathsf{Commit}(\mathsf{pp}, s_2). \\
&\mathcal{P} \to \mathcal{V}: L_d, R_d, S_1, S_2. \\
&\mathcal{V} \to \mathcal{P}: e \xleftarrow{\$} \mathbb{F}_p. \\
&\mathcal{P}: \mathbf{l}' = \mathbf{l} + e \cdot \mathbf{l}_d, \quad \mathbf{r} = \mathbf{r} + e \cdot \mathbf{r}_d, \\
&\quad \iota' = \iota + e \cdot \iota, \quad \rho' = \rho + e \cdot \rho_d, \quad \tau' = \tau + e \cdot \sigma_1 + e^2 \cdot \sigma_2. \\
&\mathcal{P} \to \mathcal{V}: \mathbf{l}', \mathbf{r}', \iota', \rho', \tau'. \\
&\mathcal{V}: L' = L + e \cdot L_d, \quad \mathsf{Check}(\mathsf{pp}, L, \mathbf{l}', \iota') \stackrel{?}{=} 1, \\
&\quad R' = R + e \cdot R_d, \quad \mathsf{Check}(\mathsf{pp}, R, \mathbf{r}', \rho') \stackrel{?}{=} 1, \\
&\quad T' = T + e \cdot S_1 + e^2 \cdot S_2, \\
&\quad \mathsf{Check}(\mathsf{pp}, T', \langle \mathbf{l}', \mathbf{r}' \rangle, \gamma') \stackrel{?}{=} 1.
\end{aligned}$$

Figure 8: Zero-knowledge argument for inner product

B.2 Sum of inner products

Another common scenario is that for a group of vectors $\{\mathbf{l}_i, \mathbf{r}_i \in \mathbb{F}_p^n\}_{i\in[m]}$ committed as $\{L_i, R_i \in \mathbb{G}\}_{i\in[m]}$, and scalar value $t \in \mathbb{F}_p$ committed as $T \in \mathbb{G}$, $\mathcal{P}$ wants to demonstrate that t is the sum of the m inner products $\sum_{i\in m}\langle \mathbf{l}_i, \mathbf{r}_i \rangle = t$, or more formally

$$\left\{ \begin{array}{c} \{L_i, R_i \in \mathbb{G}\}_{i\in[m]}, \\ T \in \mathbb{G}; \\ \{\mathbf{l}_i, \mathbf{r}_i \in \mathbb{F}_p^n\}_{i\in[m]}, \\ \{\iota_i, \rho_i \in \mathbb{F}_p\}_{i\in[m]}, \\ t \in \mathbb{F}_p, \tau \in \mathbb{F}_p \end{array} \middle| \begin{array}{c} \bigwedge_{i\in[m]} \mathsf{Check}(\mathsf{pp}, L_i, \mathbf{l}_i, \iota_i) = 1 \ \wedge \\ \bigwedge_{i\in[m]} \mathsf{Check}(\mathsf{pp}, R_i, \mathbf{r}_i, \rho_i) = 1 \ \wedge \\ \mathsf{Check}(\mathsf{pp}, T, t, \tau) = 1 \ \wedge \\ \sum_{i\in[m]} \langle \mathbf{l}_i, \mathbf{r}_i \rangle = t \end{array} \right\}. \tag{47}$$

In Figure 9, we show how it can be reduced to the single inner product case. With a random challenger $z \xleftarrow{\$} \mathbb{F}_p$, one may compute

Public input: commitments $\{L_i, R_i\}_{i\in[m]}$, T

$\mathcal{P}$'s private input: $\{\mathbf{l}_i, \mathbf{r}_i \in \mathbb{F}_p^n\}_{i\in[m]}$, $t \in \mathbb{F}_p$, blinders $\{\iota_i, \rho_i\}_{i\in[m]}, \tau \in \mathbb{F}_p$

$$\begin{aligned}
&\mathcal{P}: \mathbf{l}(Z) = \sum_{i\in[m]} \mathbf{l}_i \cdot Z^i, \quad \mathbf{r}(Z) = \sum_{i\in[m]} \mathbf{r}_i \cdot Z^{-i}, \\
&\quad \langle \mathbf{l}(Z), \mathbf{r}(Z) \rangle = t + \sum_{k\in[1-m,m-1]\backslash 0} t_k \cdot Z^k, \\
&\quad \forall\, k \in [1-m, m-1]\backslash 0, \quad (T_k, \tau_k) \leftarrow \mathsf{Commit}(\mathsf{pp}, t_k). \\
&\mathcal{P} \to \mathcal{V}: \{T_k\}. \\
&\mathcal{V} \to \mathcal{P}: z \xleftarrow{\$} \mathbb{F}_p. \\
&\mathcal{P}: \mathbf{l}_z = \mathbf{l}(z) = \sum_{i\in[m]} z^i \cdot \mathbf{l}_i, \quad \mathbf{r}_z = \mathbf{r}(z) = \sum_{i\in[m]} z^{-i} \cdot \mathbf{r}_i, \\
&\quad t_z = t + \sum_{k\in[1-m,m-1]\backslash 0} z^k \cdot t_k, \\
&\quad \iota_z = \sum_{i\in[m]} z^i \cdot \iota_i, \quad \rho_z = \sum_{i\in[m]} z^{-i} \cdot \rho_i, \\
&\quad \tau_z = \tau + \sum_{k\in[1-m,m-1]\backslash 0} z^k \cdot \tau_k. \\
&\mathcal{P}\ \&\ \mathcal{V}: L_z = \sum_{i\in[m]} z^i \cdot L_i, \quad R_z = \sum_{i\in[m]} z^{-i} \cdot R_i, \\
&\quad T_z = \sum_{k\in[1-m,m-1]\backslash 0} z^k \cdot T_k + T.
\end{aligned}$$

Public input: commitments $L_z, R_z, T_z \in \mathbb{G}$,

$\mathcal{P}$'s private input: $\mathbf{l}_z, \mathbf{r}_z \in \mathbb{F}_p^n$, $t_z \in \mathbb{F}_p$, blinders $\iota_z, \rho_z, \tau_z \in \mathbb{F}_p$

Figure 9: Argument for sum of inner products

linear combinations of $\{\mathbf{l}_i\}_{i\in[m]}$ and $\{\mathbf{r}_i\}_{i\in[m]}$ as

$$\mathbf{l}_z = \sum_{i\in[m]} z^i \cdot \mathbf{l}_i, \quad \mathbf{r}_z = \sum_{i\in[m]} z^{-i} \cdot \mathbf{r}_i, \tag{48}$$

and

$$\langle \mathbf{l}_z, \mathbf{r}_z \rangle = \sum_{i\in[m]} \langle \mathbf{l}_i, \mathbf{r}_i \rangle + \sum_{k\in[1-m,m-1]\backslash 0} t_k \cdot z^k. \tag{49}$$

According to the Schwartz-Zippel lemma, apart from negligible error $\frac{2m}{|\mathbb{F}_p|}$, the equation

$$\langle \mathbf{l}_z, \mathbf{r}_z \rangle = t \quad + \sum_{k\in[1-m,m-1]\backslash 0} t_k \cdot z^k \tag{50}$$

holds only if $\sum_{i\in[m]}\langle \mathbf{l}_i, \mathbf{r}_i \rangle = t$. So a sum of inner products reduces to a single inner product; then the protocol in Figure 8 applies.

B.3 When some vectors are public

A slightly different case is that of some of the vectors, e.g. $\{\mathbf{r}_i\}$ are public.

$$\left\{ \begin{array}{c} \{L_i \in \mathbb{G}\}_{i\in[m]}, \\ t \in \mathbb{F}_p, \{\mathbf{r}_i \in \mathbb{F}_p^n\}_{i\in[m]}; \\ \{\mathbf{l}_i \in \mathbb{F}_p^n\}_{i\in[m]}, \\ \{\iota_i \in \mathbb{F}_p\}_{i\in[m]} \end{array} \middle| \begin{array}{c} \bigwedge_{i\in[m]} \mathsf{Check}(\mathsf{pp}, L_i, \mathbf{l}_i, \iota_i) = 1 \\ \wedge \sum_{i\in[m]} \langle \mathbf{l}_i, \mathbf{r}_i \rangle = t \end{array} \right\}. \tag{51}$$

Omitting the operations related to commitments R and R_i in Figure 8, Figure 9, it is not difficult to construct an argument for relation (51).

B.4 Recursive reduction

The reduction in Section B.2 involves computing and committing the cross terms t_k for $k \in [1-m, m-1]\backslash 0$, which corresponds to $O(m^2 n)$ multiplications in $\mathbb{F}_p$ and $2m$ scalar multiplications in $\mathbb{G}$. The cost becomes substantial when m is large. In this section we

describe a recursive protocol, which accomplishes the same task of reducing a sum of inner products into one single inner product, but with way less computational cost.

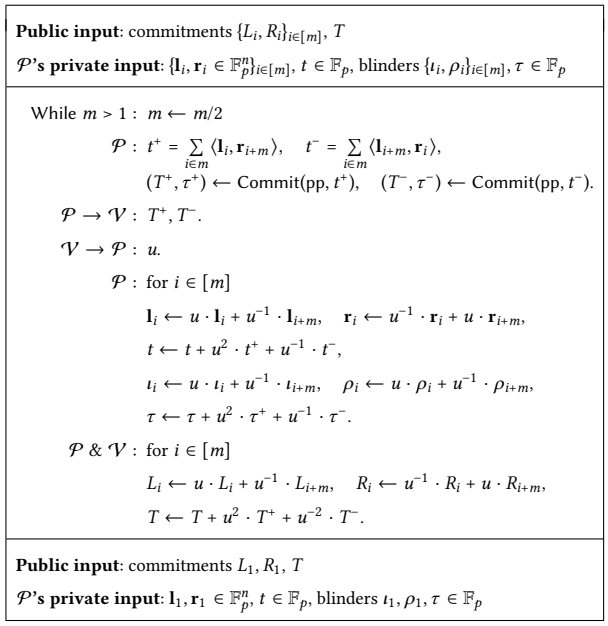

Figure 10: Recursive argument for sum of inner products

The protocol takes $\log m$ rounds, where in each round on $\mathcal{V}$'s challenger u, $\mathcal{P}$ halves the number of vectors from m into $m' = m/2$ by

$$\mathbf{l}'_i = u \cdot \mathbf{l}_i + u^{-1} \cdot \mathbf{l}_{i+m'}, \quad \mathbf{r}'_i = u^{-1} \cdot \mathbf{r}_i + u_k \cdot \mathbf{r}_{i+m'}, \tag{52}$$

that

$$\begin{aligned}\sum_{i\in[m']} \langle \mathbf{l}'_i, \mathbf{r}'_i\rangle &= \sum_{i\in[m]} \langle \mathbf{l}_i, \mathbf{r}_i\rangle + u^2 \cdot \sum_{i\in[m']} \langle \mathbf{l}_i, \mathbf{r}_{i+d'}\rangle + u^{-2} \cdot \sum_{i\in[m']} \langle \mathbf{l}_{i+d'}, \mathbf{r}_i\rangle \\ &= \sum_{i\in[m]} \langle \mathbf{l}_i, \mathbf{r}_i\rangle + u^2 \cdot t^+ + u^{-2} \cdot t^-.\end{aligned} \tag{53}$$

Summarizing all $\log m$ rounds, we have

$$\mathbf{l} = \sum_{i=1}^{m} z_i \cdot \mathbf{l}_i, \quad \mathbf{r} = \sum_{i=1}^{m} z_i^{-1} \cdot \mathbf{r}_i, \tag{54}$$

where z_i is the i-th element vector of $\mathbf{z}$ with a binary counting structure

$$\mathbf{z} = \begin{pmatrix} u_1 \dots u_{\log m} \\ u_1^{-1} \dots u_{\log m} \\ \vdots \\ u_1^{-1} \dots u_{\log m}^{-1} \end{pmatrix}, \tag{55}$$

and their inner product

$$\langle \mathbf{l}, \mathbf{r}\rangle = \sum_{i\in[m]} \langle \mathbf{l}_i, \mathbf{r}_i\rangle + \sum_{k\in[\log m]} u_k^2 \cdot t_k^+ + \sum_{k\in[\log m]} u^{-2} \cdot t_k^-. \tag{56}$$

According to the Schwartz-Zippel lemma, except for negligible error $\frac{4\cdot\log m}{|\mathbb{F}_p|}$, the equation

$$\langle \mathbf{l}, \mathbf{r}\rangle = t + \sum_{k\in[\log m]} u_k^2 \cdot t_k^+ + \sum_{k\in[\log m]} u^{-2} \cdot t_k^- \tag{57}$$

holds only if $t = \sum_{i\in[m]} \langle \mathbf{l}_i, \mathbf{r}_i\rangle$.

This recursive reduction involves computing and committing t_k^+ and t_k^- for $k \in [\log m]$, costing $O(mn)$ multiplications in $\mathbb{F}_p$ and $2\log m$ scalar multiplications in $\mathbb{G}$, a significant boost as compared to that of Section B.2.

C Composition properties of CP-NIZK arguments

C.1 Conjunction of relations with shared inputs

With a *computationally binding* commitment scheme Com and CP-NIZK arguments Π_0^{Com} and Π_1^{Com} which respectively attesting relations $\mathsf{R}_0(s_0, u, \mathsf{aux}_0) = 1$ and $\mathsf{R}_1(s_1, u, \mathsf{aux}_1) = 1$, with the common input u committed as c, one can build a CP-NIZK argument $\Pi_\wedge^{\mathsf{Com}}$ for conjunction $\mathsf{R}_\wedge$ s.t. $\mathsf{R}_\wedge(s_0, s_1, u, \mathsf{aux}_0, \mathsf{aux}_1) = 1$ iff $\mathsf{R}_0(s_0, u, \mathsf{aux}_0) = 1 \ \wedge \mathsf{R}_1(s_1, u, \mathsf{aux}_1) = 1$:

- $\mathsf{KeyGen}(1^\lambda) \to \sigma$: outputs $\sigma = (\sigma_0, \sigma_1)$, where
$$\sigma_0 \leftarrow \Pi_0^{\mathsf{Com}}.\mathsf{KeyGen}(1^\lambda),$$
$$\sigma_1 \leftarrow \Pi_1^{\mathsf{Com}}.\mathsf{KeyGen}(1^\lambda);$$
- $\mathsf{Prove}(\sigma, s_0, s_1, c, u, \mathsf{aux}_0, \mathsf{aux}_1) \to \pi$: outputs $\pi = (\pi_0, \pi_1)$ where
$$\pi_0 \leftarrow \Pi_0^{\mathsf{Com}}.\mathsf{Prove}(\sigma, (s_0, c), (u, \mathsf{aux}_0)),$$
$$\pi_1 \leftarrow \Pi_1^{\mathsf{Com}}.\mathsf{Prove}(\sigma, (s_1, c), (u, \mathsf{aux}_1));$$
- $\mathsf{Verify}(\sigma, s_0, s_1, c, \pi) \to b$: outputs $b = b_0 \wedge b_1$ where
$$b_0 \leftarrow \Pi_0^{\mathsf{Com}}.\mathsf{Verify}(\sigma, (s_0, c), \pi_0),$$
$$b_1 \leftarrow \Pi_1^{\mathsf{Com}}.\mathsf{Verify}(\sigma, (s_1, c), \pi_1);$$

The *completeness* of $\Pi_\wedge^{\mathsf{Com}}$ follows directly from the *completenesse* of Π_0^{Com} and Π_1^{Com}. For *knowledge soundness*, if the proof π verifies, one can construct an extractor Ext for $\Pi_\wedge^{\mathsf{Com}}$, which invokes the extractor Ext_0 and Ext_1 for Π_0^{Com} and Π_1^{Com} and outputs witnesses (u_0, aux_0) and (u_1, aux_1) s.t. except for negligible error,

$$\begin{aligned}&\mathsf{Com.Check}(\mathsf{pp}, c, u_0, r_0) = 1 \quad \wedge \quad \mathsf{R}_0(s_0, u_0, \mathsf{aux}_0) = 1 \\ \wedge \quad &\mathsf{Com.Check}(\mathsf{pp}, c, u_1, r_1) = 1 \quad \wedge \quad \mathsf{R}_1(s_1, u_1, \mathsf{aux}_1) = 1.\end{aligned}$$

Moreover, except for negligible possibility the identity $u_0 = u_1$ should hold, otherwise it breaks the binding property of Com. As for *zero-knowledge*, one can simulate the proof π by combining the simulated proofs π_0 and π_1 from the respective simulators for Π_0^{Com} and Π_1^{Com}.

C.2 Disjunction of relations with shared inputs

For relation $\mathsf{R}(s, u)$ define $\hat{\mathsf{R}}(s, u, t)$ s.t.

$$\hat{\mathsf{R}}(s, u, t) = 1 \text{ iff } \begin{cases} \mathsf{R}(s, u) = 1 \ \wedge \ t = 0, \\ \mathsf{R}(s, u) = 0 \ \wedge \ t \neq 0. \end{cases} \tag{58}$$

With this definition, the disjunction of relations is converted into conjunctions. Specifically, for $\mathsf{R}_0(s_0, u)$ and $\mathsf{R}_1(s_1, u)$ with shared inputs u, relation $\mathsf{R}_\vee$ is defined as $\mathsf{R}_\vee(s_0, s_1, u) = 1$ iff $\mathsf{R}_0(s_0, u) = 1 \vee \mathsf{R}_1(s_1, u) = 1$, which is equivalent to

$$\hat{\mathsf{R}}_0(s_0, u, t_0) = 1 \ \wedge \ \hat{\mathsf{R}}_1(s_1, u, t_1) = 1 \ \wedge \ t_0 \cdot t_1 = 0. \tag{59}$$

C.3 Sequential composition of functions

Define relations R_g and R_h associated with functions g and h as

$$\begin{aligned} \mathsf{R}_g(z, x, w) = 1 \text{ iff } \exists\ (x, w) \text{ s.t. } g(x, w) = z, \\ \mathsf{R}_h(s, z, y) = 1 \text{ iff } \exists\ (z, y) \text{ s.t. } h(z, y) = s. \end{aligned} \tag{60}$$

Let f be the composition of g and h s.t. $f(x, y, w) = h(g(x, w), y)$. Then the relation associated with f, namely $\exists\,(x, y, w)$ s.t $f(x, y, w) = s$, can be reduced to conjunction of R_g and R_h

$$\mathsf{R}_g(z, x, w) = 1\ \wedge \mathsf{R}_h(s, z, y) = 1. \tag{61}$$

D Proof for Theorem 3.1

Theorem 3.1 states that the argument in Figure 2 has *perfect completeness*, *computational witness-extended emulation* and PHVZK under the discrete logarithm relation assumption.

Perfect completeness is straightforward.

To prove ***computational witness-extended emulation***, we construct an efficient extractor Ext, which runs $\mathcal{P}$ with n different y, m different w, 2 different u_k for $k \in [0, \log m]$ and 3 different e, in total $n \times m \times 2 \times \log 2m \times 3$ valid proof transcripts. For simplicity, we denote $K = [1 - 2m, 2m - 1] \backslash 0$.

Choose 2 transcripts for $e \in \{e^{(1)}, e^{(2)}\}$ and fixed $w, y, \{u_k\}$. From the transmitted $\mathbf{l}'$, Ext can extract $\mathbf{l}_d, \iota_d$ s.t. $L_d = \langle \mathbf{l}_d, \mathbf{G} \rangle + \iota_d \cdot H$. If for any other set of challenges Ext extracts different $\mathbf{l}_d, \iota_d$ then it yields a non-trivial discrete logarithm relation among $(H, \mathbf{G})$ w.r.t. commitment L_d. With $\mathbf{l}_d$ fixed, Ext can extract $\mathbf{l}, \iota$ from transmitted $\mathbf{l}', \iota'$ by $\mathbf{l} = \mathbf{l}' - e \cdot \mathbf{l}_d$, $\iota = \iota' - e \cdot \iota_d$. In case of $\mathsf{Check}(\mathsf{pp}, L', \mathbf{l}', \iota') = 1$, it must hold that $\langle \mathbf{l}, \mathbf{G} \rangle + \iota \cdot H = L' - e \cdot L_d = L$, otherwise it yields a non-trivial discrete relation among $(H, \mathbf{G})$ with respect to L. Similarly, Ext can extract $\mathbf{r}, \rho$ from transmitted $\mathbf{r}', \rho'$ s.t. $\langle \mathbf{r}, \mathbf{G} \rangle + \rho \cdot H = R' - e \cdot R_d = R$. For inner product, in case of $\mathsf{Check}(\mathsf{pp}, T', \langle \mathbf{l}', \mathbf{r}' \rangle, \tau') = 1$, by 3 transcripts with different $e \in \{e^{(1)}, e^{(2)}, e^{(3)}\}$, Ext can extract τ from transmitted σ s.t.

$$\langle \mathbf{l}, \mathbf{r} \rangle \cdot U + \tau \cdot H = \sum_{k=0}^{\log m} \left(u_k^2 \cdot T_k^+ + u_k^{-2} \cdot T_k^- \right). \tag{62}$$

Choose 2 transcripts with different $u_0 \in \{u_0^{(1)}, u_0^{(2)}\}$ but same u_k for $k \in [1, \log m]$. Correspondingly there are $\mathbf{l} \in \{\mathbf{l}^{(1)}, \mathbf{l}^{(2)}\}$, $\mathbf{r} \in \{\mathbf{r}^{(1)}, \mathbf{r}^{(2)}\}$, and $\tau \in \{\tau^{(1)}, \tau^{(2)}\}$. Ext is able to compute from $\mathbf{l} \in \{\mathbf{l}^{(1)}, \mathbf{l}^{(2)}\}$ two vectors $\mathbf{l}_1, \mathbf{l}_2$ s.t. $\mathbf{l} = u_0 \cdot \mathbf{l}_1 + u_0^{-1} \cdot \mathbf{l}_2$. Similarly, Ext recovers $\mathbf{r}_1, \mathbf{r}_2$ s.t. $\mathbf{r} = u_0 \cdot \mathbf{r}_1 + u_0^{-1} \cdot \mathbf{r}_2$. Moreover, their inner products must satisfy the equation

$$\langle \mathbf{l}, \mathbf{r} \rangle = \sum_{i=1}^{2} \langle \mathbf{l}_i, \mathbf{r}_i \rangle + u_0^2 \cdot \langle \mathbf{l}_1, \mathbf{r}_2 \rangle + \langle \mathbf{l}_2, \mathbf{r}_1 \rangle. \tag{63}$$

Run the process recursively, in the k-th round picking two different u_k, after $\log m$ more rounds, Ext will be able to construct $\{\mathbf{l}_i\}_{i \in [2m]}$ and $\{\mathbf{r}_i\}_{i \in [2m]}$ s.t.

$$\langle \mathbf{l}, \mathbf{r} \rangle = \sum_{i=1}^{2m} \langle \mathbf{l}_i, \mathbf{r}_i \rangle + \sum_{k=0}^{\log m} \left(u_k^2 \cdot \sum_{i=1}^{2^k} \langle \mathbf{l}_i, \mathbf{r}_{i+2^k} \rangle + u_k^{-2} \cdot \sum_{i=1}^{2^k} \langle \mathbf{l}_{i+2^k}, \mathbf{r}_i \rangle \right). \tag{64}$$

Equations (62) and (64) imply that

$$\sum_{i=1}^{2m} \langle \mathbf{l}_i, \mathbf{r}_i \rangle = 0. \tag{65}$$

Otherwise they should yield a non-trivial discrete logarithm relation between (U, H).

Construct vector $\mathbf{z}$ from $\{u_k\}_{k=0}^{\log m}$ with a binary counting structure

$$\mathbf{z} = \begin{pmatrix} u_0 u_1 \dots u_{\log m} \\ u_0^{-1} u_1 \dots u_{\log m} \\ \vdots \\ u_0^{-1} u_1^{-1} \dots u_{\log m}^{-1} \end{pmatrix}, \tag{66}$$

and note that $\mathbf{l}, \mathbf{r}$ and $\{\mathbf{l}_i, \mathbf{r}_i\}_{i \in [2m]}$ are related by the relation that

$$\mathbf{l} = \sum_{i \in [2m]} z_i \cdot \mathbf{l}_i, \quad \mathbf{r} = \sum_{i \in [2m]} z_i^{-1} \cdot \mathbf{r}_i, \tag{67}$$

where z_i is the i-th element of $\mathbf{z}$. Similarly, Ext can construct $\{\iota_i, \rho_i\}_{i \in [2m]}$ from ι, ρ s.t.

$$\iota = \sum_{i \in [2m]} z_i \cdot \iota_i, \quad \rho = \sum_{i \in [2m]} z_i^{-1} \cdot \rho_i. \tag{68}$$

When a transcript is accepting, it must hold that

$$L = \langle \mathbf{l}, \mathbf{G} \rangle + \iota \cdot H = \sum_{i \in [2m]} z_i \cdot \langle \mathbf{l}_i, \mathbf{G} \rangle + \sum_{i \in [2m]} \iota_i \cdot H. \tag{69}$$

Recall the definition of L

$$L = \sum_{i \in [m]} \sum_{j \in [m]} w^i \cdot \left(z_i \cdot A_{ij} + z_{i+m} \cdot C_{ij} \right) \cdot V_j. \tag{70}$$

Equations (69) and (70) imply that for $i \in [m]$

$$\begin{aligned} \langle \mathbf{l}_i, \mathbf{G} \rangle + \iota_i \cdot H = w^i \cdot \sum_{j \in [m]} A_{ij} \cdot V_j. \\ \langle \mathbf{l}_{i+m}, \mathbf{G} \rangle + \iota_{i+m} \cdot H = w^i \cdot \sum_{j \in [m]} C_{ij} \cdot V_j. \end{aligned} \tag{71}$$

Assume $V_j = \langle \mathbf{v}_j, \mathbf{G} \rangle + v_j \cdot H$, then it should hold that

$$\mathbf{l}_i = w^i \cdot \sum_{j \in [m]} A_{ij} \cdot \mathbf{v}_j, \quad \mathbf{l}_{i+m} = w^i \cdot \sum_{j \in [m]} C_{ij} \cdot \mathbf{v}_j. \tag{72}$$

Otherwise it yields a non-trivial discrete logarithm relation among $(H, \mathbf{G})$. Similarly, there are

$$\mathbf{r}_i = \sum_{j \in [m]} B_{ij} \cdot \mathbf{v}_j \circ \mathbf{y}_n, \quad \mathbf{r}_{i+m} = \mathbf{y}_n. \tag{73}$$

According to (65), it must hold that

$$\sum_{i \in [m]} w^i \cdot \left(\left\langle \sum_{j \in [m]} A_{ij} \cdot \mathbf{v}_j, \sum_{j \in [m]} B_{ij} \cdot \mathbf{v}_j \circ \mathbf{y}_n \right\rangle - \left\langle \sum_{j \in [m]} C_{ij} \cdot \mathbf{v}_j, \mathbf{y}_n \right\rangle \right) = 0. \tag{74}$$

Using a fixed y and m different w challenges, we can infer that

$$\left\langle \sum_{j \in [m]} A_{ij} \cdot \mathbf{v}_j, \sum_{j \in [m]} B_{ij} \cdot \mathbf{v}_j \circ \mathbf{y}_n \right\rangle - \left\langle \sum_{j \in [m]} C_{ij} \cdot \mathbf{v}_j, \mathbf{y}_n \right\rangle = 0. \tag{75}$$

Then using n different y challenges, we can infer that

$$\left(\sum_{j \in [m]} A_{ij} \cdot \mathbf{v}_j \right) \circ \left(\sum_{j \in [m]} B_{ij} \cdot \mathbf{v}_j \right) - \sum_{j \in [m]} C_{ij} \cdot \mathbf{v}_j = 0. \tag{76}$$

To prove **PSHVZK**, we construct an efficient simulator Sim that produces a distribution of transcripts

$$\begin{aligned} \mathsf{tr} = (\{V_j\}_{j \in [m]}, w, y, \{T_k^+, T_k^-, u_k\}_{k \in [0, \log m]}, \\ L_d, R_d, S_1, S_2, e, \mathbf{l}', \mathbf{r}', \iota', \rho', \sigma'), \end{aligned} \tag{77}$$

and the distribution is indistinguishable from that produced by an honest $\mathcal{P}$ interacting with an honest $\mathcal{V}$. Sim uses $\mathcal{V}$'s randomness, and runs as follows.

$$
\begin{aligned}
&V_j \xleftarrow{\$} \mathbb{G}, \quad \mathbf{l}', \mathbf{r}' \xleftarrow{\$} \mathbb{F}_p^n, \quad \iota', \rho', \tau' \xleftarrow{\$} \mathbb{F}_p, \quad S_1 \xleftarrow{\$} \mathbb{G} \\
&T_k^+, T_k^- \xleftarrow{\$} \mathbb{G}, \quad \forall\, k \in [0, \log m], \\
&L \leftarrow \sum_{j\in[m]} \sum_{i\in[m]} w^i \cdot \left(s_i \cdot A_{ij} + s_{i+m} \cdot C_{ij}\right) \cdot V_j, \\
&R \leftarrow \sum_{i\in[m]} \sum_{j\in[m]} \left(s_i \cdot B_{ij}\right) \cdot V_j + \sum_{i\in[m]} s_{i+m} \cdot \langle \mathbf{1}, \mathbf{G}\rangle, \\
&L_d \leftarrow e^{-1} \cdot \left(\langle \mathbf{l}', \mathbf{G}\rangle + \iota \cdot H - L\right), \\
&R_d \leftarrow e^{-1} \cdot \left(\langle \mathbf{r}' \circ \mathbf{y}_n^{-1}, \mathbf{G}\rangle + \rho \cdot H - R\right), \\
&S_2 \leftarrow e^{-2} \cdot \left(\langle \mathbf{l}', \mathbf{r}'\rangle \cdot U + \tau' \cdot H\right) + e^{-1} \cdot S_1 \\
&\qquad - e^{-2} \cdot \sum_{k=0}^{\log 2m} \left(u_k^2 \cdot T_k^+ + u_k^{-2} \cdot T_k^-\right).
\end{aligned}
\tag{78}
$$

It is straightforward to see that such a proof produced by Sim will be accepted by an honest $\mathcal{V}$. $\mathbf{l}', \mathbf{r}', \iota', \rho', \sigma'$ are randomly distributed over $\mathbb{F}_p$. Due to the *prefect hiding* property of Pedersen commitment, $L_d, R_d, \{T_k^+, T_k^-\}, S_1, S_2$ are also random elements of $\mathbb{G}$, apart from the relation defined by the verification procedure. Therefore, the transcript given by (78) is indistinguishable from an honestly computed proof with uniformly sampled challengers.

E Proof for Theorem 4.1

Theorem 4.1 states that ZKCPlus is *secure* if the embedded CP-NIZK argument Π^{Com} satisfies *completeness*, *computational knowledge soundness* and *zero-knowledge*, where by *secure* we mean:

- (*buyer fairness*) for any possibly malicious seller $\mathcal{S}^*$, if its balance increases with non-negligble probability, then the buyer $\mathcal{B}$ learns some $\mathbf{x}'$ s.t. $\phi(\mathbf{x}') = 1$;
- (*seller fairness*) for a seller $\mathcal{S}$ interacting with a possibly malicious buyer $\mathcal{B}^*$, if $\mathcal{S}$'s balance does not increase, then except for negligible probability, $\mathcal{B}^*$ learns no information of $\mathbf{x}$, apart from what can be inferred from ϕ.

For *buyer fairness*, If $\mathcal{B}$ interacts with a possibly malicious $\mathcal{S}^*$, and $\mathcal{S}^*$'s balance increases, then it must be the case that all three of the following conditions are met

1. in the "validate" phase, $\mathcal{B}$ receives π s.t.
$$\Pi^{\mathsf{Com}}.\mathsf{Verify}(\sigma, (\phi, c_{\mathbf{x}}), \pi) = 1;$$
2. by the end of the "deliver" phase, $\mathbf{z}$, h and (π_z) s.t.
$$\Pi^{\mathsf{Com}}.\mathsf{Verify}(\sigma, (\mathbf{z}, h, c_{\mathbf{x}}), \pi_z) = 1;$$
3. in the "finalize" phase, $\mathcal{J}$ gets k s.t. $H(k) = h$.

Due to the computational knowledge soundness of Π^{Com}, for condition 2. to hold, there exists an extractor Ext which except for probability $\mathsf{negl}(\lambda)$, outputs $(\mathbf{x}', k')$

$$\mathsf{Enc}_k(\mathbf{x}') = \mathbf{z} \ \wedge h = \mathsf{H}(k').$$

Similarly for condition 1. to hold, there must exist an extractor Ext which except for probability $\mathsf{negl}(\lambda)$, outputs $\mathbf{x}''$ s.t.

$$\phi(\mathbf{x}'') = 1.$$

Due to the binding property of Com, except for negligible probability, there is $\mathbf{x}'' = \mathbf{x}'$, and hence $\phi(x') = 1$. Assume that at the end of protocol execution, $\mathcal{S}^*$'s balance increases, but $\mathcal{B}$ does not learn $\mathbf{x}'$, then it must be the case that in the "finalize" phase $\mathcal{J}$ receives some k different from k' but $\mathsf{H}(k) = h$. This breaks the collision resistance of H.

In Section 4.2.2, we propose using commitment $c_{\mathbf{k}}$ instead of h. Then $\mathcal{B}$ loses its funds but learns a wrong $\mathbf{x}$ iff in the "finalize" phase $\mathcal{J}$ learns a pair $(k, r_{\mathbf{k}})$ s.t. $k \neq k'$ but $\mathsf{Check}(\mathsf{pp}, c_{\mathbf{k}}, (k \cdot \mathbf{1}), r_{\mathbf{k}}) = 1$. This breaks the binding property of the commitment scheme Com.

For *seller fairness*, if $\mathcal{S}$ interacts with a possibly malicious $\mathcal{B}^*$, and $\mathcal{S}$'s funds do not increase, we can construct a simulator $\mathsf{Sim}_{\mathcal{B}^*}$ which runs on input $(c_{\mathbf{x}}, \mathbf{z}, h)$, and for $\mathcal{B}^*$ it is indistinguishable from an honest seller $\mathcal{S}$. The simulator $\mathsf{Sim}_{\mathcal{B}^*}$ works as follows:

1. in "validate" phase, $\mathsf{Sim}_{\mathcal{B}^*}$ runs Π^{Com}'s simulator Sim, sends $\pi' \leftarrow \mathsf{Sim}(\sigma, (s, c_{\mathbf{x}}))$;
2. in "deliver" phase, $\mathsf{Sim}_{\mathcal{B}^*}$ runs Π^{Com}'s simulator Sim, gets $\pi_z' \leftarrow \mathsf{Sim}(\sigma, (\mathbf{z}, c_{\mathbf{x}}, h))$, sends $(\mathbf{z}, h, \pi_z')$;
3. in "reveal" phase, $\mathsf{Sim}_{\mathcal{B}^*}$ aborts.

Due to the zero-knowledge property of Π^{Com}, π' and π_z' are indistinguishable from that sent by an honest $\mathcal{S}$. Moreover, on the premise that $\mathcal{S}$'s funds does not increase, either $\mathcal{B}^*$ aborts, or $\mathcal{S}$ sees a transaction tx posted by $\mathcal{B}^*$ which contains some h s.t. $\mathsf{H}(k) \neq h$, in which case $\mathcal{S}$ must abort in "reveal" phase.

Moreover, due to preimage resistance of H, it is infeasible for $\mathcal{B}^*$ to infer k from h, so it cannot decrypt $\mathbf{z}$ to $\mathbf{x}$. Similar conclusion holds when we use $c_{\mathbf{k}}$ instead of h, due to the hiding property of Com.

F Pay to sudoku solution by ZKCPlus

Herein we describe how ZKCPlus adapts to the scenario of exchanging sudoku solutions. Assume a sudoku puzzle of dimension $D \times D$. The protocol starts with the "commit" phase where the seller $\mathcal{S}$ makes commitment to the solution $\mathbf{x} \in \mathbb{F}_p^{D^2}$. We've described how the data-parallel CP-NIZK argument in Section 3 applies to the "delivier" phase of ZKCPlus; what is not so obvious is that it also applies to the "validate" phase for a sudoku solution as well. To convince the buyer that $\mathbf{x}$ is correct, $\mathcal{S}$ has to prove that of the $D \times D$ solution, each row $\mathbf{r}_i$, column $\mathbf{c}_i$, and sub-grid $\mathbf{s}_i$ for $i \in [D]$ is a permutation of digits $\{1, \ldots, D\}$, namely

$$\mathsf{isPerm}_D(\mathbf{r}_i) = 1, \quad \mathsf{isPerm}_D(\mathbf{c}_i) = 1, \quad \mathsf{isPerm}_D(\mathbf{s}_i) = 1 \tag{79}$$

hold for all $i \in [D]$. This constitutes a data-parallel computation with $3D$ copies of function isPerm running in parallel. Besides, $\mathcal{S}$ has to prove that $\{\mathbf{r}_i\}$ are consistent the committed $\mathbf{x}$. It can be accomplished by a CP-NIZK argument for inner products

$$\langle \mathbf{x}, \mathbf{y}\rangle - \sum_{i\in[D]} \langle \mathbf{r}_i, \mathbf{y}_i\rangle = 0, \tag{80}$$

where $\mathbf{y} \xleftarrow{\$} \mathbb{F}_p^{D^2}$, and $\mathbf{y} = (\mathbf{y}_1 || \ldots || \mathbf{y}_D)$. The Schwartz-Zippel lemma states that except for error $\frac{D^2}{|\mathbb{F}_p|}$, (80) holds iff $\mathbf{x}$ and $\{\mathbf{r}_i\}$ are consistent. Similar arguments apply to $\{\mathbf{c}_i\}$ and $\{\mathbf{s}_i\}$. In addition, $\mathcal{S}$ has to prove that $\mathbf{x}$ is consistent with the puzzle setting $\mathbf{p}$, that at position i where the digit is open, there is $x_i = p_i$. With binary mask $\mathbf{b} \in \mathbb{F}_p^{D^2}$

where

$$\mathbf{b}_i = \begin{cases} 1 & \text{position } i \text{ is open,} \\ 0 & \text{otherwise,} \end{cases} \tag{81}$$

the consistency between $\mathbf{x}$ and $\mathbf{p}$ translates into an equation $\mathbf{x} \circ \mathbf{b} = \mathbf{p}$. Again $\mathcal{S}$ can use an inner product argument for

$$\langle \mathbf{x} \circ \mathbf{b},\ \mathbf{y} \rangle = \langle \mathbf{p},\ \mathbf{y} \rangle \tag{82}$$

to prove it.

With the "validate" phase and "deliver" phase separated, one may optionally incorporate a data compression procedure. Since each element of $\mathbf{x}$ is a digit in the range of $[1, D]$, encoding it as an element of field $\mathbb{F}_p$ is somewhat wasteful. Alternatively, with l marking the bit length of $\mathbb{F}_p$, and $k = \lceil \log(D+1) \rceil$ marking the bit length of each digit in $[1, D]$, $\mathcal{S}$ may group $t = \lfloor l/k \rfloor$ digits into one element of $\mathbb{F}_p$, and reform a solution $\mathbf{x}$ of length D^2 into $\mathbf{x}'$ of length $\lceil D^2/t \rceil$. To justify the consistency between $\mathbf{x}$ and $\mathbf{x}'$, $\mathcal{S}$ may randomly sample $\mathbf{y}' \xleftarrow{\$} \mathbb{F}_p^{\lceil D^2/t \rceil}$, and by a length t vector $\mathbf{2}_t^k = [1, 2^k, \cdots, 2^{(t-1)k}]$ construct $\mathbf{y}$ as

$$\mathbf{y} = [y'_1 \cdot \mathbf{2}_t^k, \cdots, y'_{\lceil D^2/t \rceil} \cdot \mathbf{2}_t^k] \in \mathbb{F}_p^{\lceil D^2/t \rceil \cdot t}, \tag{83}$$

where y'_i is the i-th element vector of $\mathbf{y}'$, and then employ an inner product argument for

$$\langle \mathbf{x}, \mathbf{y} \rangle - \langle \mathbf{x}', \mathbf{y}' \rangle = 0, \tag{84}$$

with $\mathbf{x}$ padded with zeros into the same length as $\mathbf{y}$. With the compression on, the "deliver" phase runs on $\mathbf{x}'$ instead of $\mathbf{x}$, which is t times more efficient.

G Other evaluation results

We also measure computational cost of the on-chain arbiter in ZKCPlus's "finalize" phase, i.e., the gas cost of the smart contract. In our protocol the contract takes charge of commitment opening and concurrent transferring, and by storing a prepared group element, the computational complexity can be reduced to $O(1)$ (see Section 4.2.2 for details).

Our evaluation shows that the total on-chain gas cost of the smart contract in "finalize" phase is quite economical, around 156,770 gas independent of data size. Among all these consumed gas, transferring payment and updating the ledger's state (i.e., writing to storage) account for the majority of the computational cost by 102,082 gas, and commitment opening costs only about 15,900 gas, 10.14% of the total gas. Considering ZKCP, computing hash on chain is cheaper than commitment opening, but actually it does not help much to overall on-chain gas cost, since either hash computation or commitment opening accounts for only a minor portion of overall gas cost.