Dubhe: Succinct Zero-Knowledge Proofs for Standard AES and related Applications

Changchang Ding and Yan Huang

Indiana University, Bloomington

Abstract

We explore a new approach to construct zero-knowledge proofs by combining ideas from the succinct proof system GKR, the Fully Linear PCP (FLPCP), and MPC-in-the-Head ZKPoK. Our discovery contributes to the state-of-the-art of ZKP in two aspects:

(1) **Methodology:** We demonstrate a way to build transparent ZK proofs from simplified variant of FLPCP and KKW. The resulting proofs are practically efficient ($O(|C|)$-time prover, $O(\log(|C|)$-time verifier, $O(\log(|C|))$-bandwidth where $|C|$ is the number of *polynomial* gates), and work readily for circuits defined with polynomial gates over any finite field.

(2) **Applications:** We present efficient (interactive) identification schemes, ring identification schemes, (non-interactive) digital signatures and ring signatures, all based on the standard AES ciphersuite. We also show the first practically efficient verifiable symmetric-key encryption scheme, based on counter-mode AES.

1 Introduction

Zero-Knowledge Proofs (ZKP) allow a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ a valid computational relation without revealing any extra information such as the secret witness. ZKPs enable a wide range of interesting applications, from building efficient post-quantum digital signatures, to secure electronic voting, to privacy-preserving decentralized cryptocurrency.

Many ZKP protocols followed the MPC-in-the-Head paradigm [2, 5, 15, 20, 23, 24]. In this paradigm, the prover divides the witness into its secret-shares, simulating the computation used to check the relation using some MPC protocol, then commits to the transcripts of the simulated MPC players, of which the verifier randomly select a subset to verify prover's behavior without learning anything about the witness. By design, this approach can easily support circuits over any finite field and is able to produce quantum-resistant proofs without trusted setup. In addition, these proofs (except Ligero [2]) are typically memory-friendly since they can be executed using as little space as what is required to compute the relation-verification circuit, rather than linear space to remember all intermediate values produced in the circuit evaluation. On the flip side, however, these protocols require linear-time verifiers and linear communication,[1] hence they are not succinct.

On the other hand, GKR [21] offered an efficient protocol that allows to verify results of delegated non-secret computation in sublinear time using purely public random-coins. Recently, researchers have leveraged GKR to build *succinct* ZKPs [31, 32, 34, 35]. However, these constructions relied on polynomial commitments (such as Hyrax [31] and Libra [32]) or Low Degree Tests (LDT) (such as Virgo [35] and Virgo++ [34]) to make the GKR procedure fully zero-knowledge, thus inheriting some constraints from those techniques: (1) They don't easily support computations over many fields that are incompatible with the underlying polynomial commitment or LDT schemes. E.g., Virgo/Virgo++'s competitive performance hinges on the extended Mersenne prime fields $\mathbb{F}_{p^2}$ (e.g., they used $p = 2^{61} - 1$). So even if AES can be more efficiently computed by a $\mathbb{F}_{2^8}$ circuit, Virgo/Virgo++ have to prove it with a (much larger) Boolean circuit while "lifting" every plaintext bit to an element in $\mathbb{F}_{p^2}$. (2) Adjusting the concrete security parameters of those protocols for better soundness guarantees often becomes an unwieldy task causing a significant performance dent. E.g., using $\mathbb{F}_{p^2}$ with $p = 2^{61} - 1$, Virgo/Virgo++ are conjectured to offer $\sim$100-bit computational security for proving a circuit of a single SHA256 hash, but only $\sim$93-bit for a circuit of 256 SHA256 hashes. Tuning the soundness even just a few bits up would require setting p to the next Mersenne prime $2^{89} - 1$, which can be 2–4x more expensive based on benchmarks on the field operations.

Hence, our work is motivated by the question: "*Is it possible to construct concretely-efficient succinct zero-knowledge*

[1] Except Ligero, which requires linear verification and square-root traffic.

proof system that can easily support computation on arbitrary finite fields?" In this paper, we present ZKPs built from integrating three seminal uses of random coins: one for sublinear proof of correctness (GKR), one for making any computation zero-knowledge (MPC-in-the-Head), and one for a little bit of both (Fully Linear PCP). We propose a new method to construct succinct[2] ZKPs and work out a number of details in selecting concretely efficient component protocols.

ZKP Applications. In addition to the several (somewhat) hypothetical benchmark computations (e.g. SHA256 and matrix multiplication) commonly used to evaluate ZKP systems, we also improve the state-of-art ZKPs for several applications more relevant to addressing real-world security needs. These include digital identification and signature schemes, ring identification and ring signature schemes, and, to the best of our knowledge, the first succinct proof of symmetric-key encryption.

The first distinction of our work across these applications lies in a highly efficient, succinct-verifier ZKP system for the *standard* AES cipher suite. Early signatures built from ZKP of one-way functions chose *MPC-friendly* ciphers [1] for better-looking performance, at some cost of security. Those ciphers are relatively new and lack the same level of scrutiny that standardized ciphers like AES have undergone. Moreover, AES is already widely used in many applications, often accelerated with dedicated hardware, hence in scenarios such as proving a plaintext-relation among ciphertexts, it is technically infeasible to replace AES with a MPC-friendly cipher just for the sake of ZKP. Instead, more desirable would be an efficient ZK protocol for proving standard AES.

Several recent works have considered ZKP of AES [4, 5, 18, 28]. However, they only work for a modified version of AES where no input byte to any of the SubByte modules is allowed to be 0-byte. Since this constitutes a non-standard use of the standard cipher suite, it can cause security and utility concerns similar to those coming from nonstandard ciphers: (1) Security of the non-standard use of AES is not well-understood — it is unclear if the modified usage would make some (otherwise difficult on standard usage) cryptanalysis easier. (2) It won't work for situations where inputs to AES are beyond control (e.g., when calling AES repetitively in some modes-of-operation or for circuit garbling) since 0-byte input to SubByte is inevitable. Finally, their technique would make the key generation more than $2\times$ slower due to repeated key trials.

We explore novel ways to prove a single block of AES, many blocks of AES, private membership, as well as their combinations. Our circuits work particularly well with our methodology of building ZKPs.

[2]Here we consider on a weaker notion of succinctness as was defined in [10,11], which only require sublinear proofs in terms of circuit size. However, we note that some protocols like Virgo++/Ligero++ support a stronger notion of succinctness that requires proofs grow sublinearly in both circuit size and secret input length.

Table 1: Asymptotic costs of transparent ZKP systems

	KKW [24]	Virgo [35]	Ligero++ [9]	Virgo++ [34]	Limbo [18]	Dubhe
$\mathcal{P}$	$C+w$	$C+w\log w$	$(C+w)\log(C+w)$	$C+w\log w$	$C+w$	$C+w$
$\mathcal{V}$	$C+w$	$d\log C+\log^2 w$	$C+w$	$d\log C+d^2+\log^2 w$	$C+w$	$d\log C+w$
$\lvert\pi\rvert$	$C+w$	$d\log C+\log^2 w$	$\mathsf{polylog}(C+w)$	$d\log C+d^2+\log^2 w$	$\log C+w$	$d\log C+w$

C denotes circuit size measured in the number of gates. w denotes the length of secret witness. The big-O's are omitted in all cells. With Virgo and Virgo++, $O(C)$ extra secret witnesses are often deliberately added in practice to reduce the number of circuit layers.

Contributions. We explore a new approach, implemented as Dubhe, to construct efficient ZKP protocols by combining MPC-in-the-Head (Section 2.1), GKR (Section 2.2), and Fully linear PCP (Section 2.3). Like all MPC-in-the-Head ZKP protocols, Dubhe easily works for computations over any fields. Comparing to existing MPC-in-the-Head ZKPs, Dubhe protocols are both asymptotically and concretely more efficient. We compare asymptotic costs of our approach to some state-of-the-art transparent ZKP systems in Table 1.

We applied our approach to develop a number of applications, including efficient identification and digital signature schemes (Section 5.1), ring identification and ring signatures (Section 5.2), as well as verifiable symmetric-key encryption schemes (Section 5.3), all of which were based on the standard *unmodified* AES. We experimentally evaluated Dubhe protocols and find them highly competitive compared to best prior works in both interactive and non-interactive settings.[3]

2 Preliminaries

Notation. We denote the prover and verifier by $\mathcal{P}$ and $\mathcal{V}$. We use π to denote the proof (or communication) resulted from running a proof system. We use d to denote the depth of a circuit, but d for the degree of polynomial gates in the circuit. For all positive integer n, $[n] \stackrel{\text{def}}{=} \{0,1,\ldots,n-1\}$. If S is a set, "$a \leftarrow S$" means uniformly sample an element from S and name it a. We always use "$a = b$" to denote equality while ":=" to denote assignment. We model $H(\cdot)$ as a random oracle. Appendix A has relevant definitions such as ZK Proof/Argument of Knowledge, Σ-protocols, and Interactive Oracle Proofs (IOP).

2.1 "MPC-in-the-Head" ZKP

The MPC-in-the-Head paradigm for constructing efficient ZKP was originally proposed by Ishai et al. [23], first efficiently implemented in ZKBoo [20] and improved in several

[3]Dubhe source code: https://github.com/zkPrfs/dubhe.

subsequent works [2, 9, 15, 24]. The high-level idea of this paradigm works as follows:

(1) $\mathcal{P}$ divides its witness w into n shares, run an n-party secure computation protocol $\Pi_{R(\cdot,x)}$ "in its head" to emulate the computation of $R(w,x)$, and commits n views of the parties;

(2) $\mathcal{V}$ picks a random subset of the n views to check;

(3) $\mathcal{P}$ de-commits the chosen subset of views. $\mathcal{V}$ accepts if and only if every disclosed view is consistent with all other disclosed views, the public input x, the definition and output of R, and the specification of protocol $\Pi_{R(\cdot,x)}$.

Different instantiations of the MPC-in-the-Head idea differ in their underlying MPC protocols, which affect the exact definition of a party's view, optimizations for committing and de-committing the views, as well as the security and cost of these protocols.

Our work uses a substantially simplified version of KKW [24] as a building block that only needs to handle linear gates. In the terminology of MPC-in-the-Head protocols, our underlying protocol $\Pi_{R(\cdot,x)}$ is $(n-1)$-*private* (i.e., $n-1$ parties' views can be revealed without leaking the secret), 0-*robust* (i.e., a malicious prover needs to corrupt at least 1 party per iteration to succeed). The analysis of our MPC-in-the-Head building block is consequently much simpler: the soundness error of each iteration is at most $1/n$; and KKW's 5-round interactive HVZK can be naturally reduced to 3-round since we never need to handle multiplication gates.

2.2 GKR

Originally proposed by Goldwasser et al. [21], GKR has been significantly improved in a series of research papers [16, 17, 30, 32]. Here we just recall enough of its basics to understand our new ZK argument system.

Multilinear Extension. Let $V : \{0,1\}^\ell \mapsto \mathbb{F}$ be a function. The *multilinear extension* of V is the unique polynomial $\widetilde{V} : \mathbb{F}^\ell \mapsto \mathbb{F}$ defined as below

$$\widetilde{V}(x_0,\ldots,x_{\ell-1}) \stackrel{\text{def}}{=} \sum_{b\in\{0,1\}^\ell} \left(V(b) \prod_{i\in\ell} \left((1-x_i)(1-b_i)+x_i b_i\right)\right).$$

It is easy to see that $\forall x \in \{0,1\}^\ell, \widetilde{V}(x) = V(x)$.

The Sumcheck Protocol. The goal of sumcheck protocol [25] is to efficiently verify the summation of a polynomial $f : \mathbb{F}^\ell \mapsto \mathbb{F}$ on a binary hypercube, i.e., $\sum_{b_i\in\{0,1\}} f(b_1,\ldots,b_\ell)$. The sumcheck protocol reduces the correctness of summation $\sum_{b_i\in\{0,1\}} f(b_1,\ldots,b_\ell)$ to f's value on a random point in $\mathbb{F}^\ell$, i.e., $f(r_1,\ldots,r_\ell)$ with uniform $r_i \in \mathbb{F}$ selected by $\mathcal{V}$. The protocol runs in ℓ rounds:

(1) In the first round, $\mathcal{P}$ sends the univariate polynomial $f_1(x_1) \stackrel{\text{def}}{=} \sum_{b_2,\ldots,b_\ell\in\{0,1\}} f(x_1,b_2,\ldots,b_\ell)$ by sending $\deg(f_1)+1$ coefficients of f_1 where $\deg(f_1)$ is f_1's degree. $\mathcal{V}$ verifies

$$S = f_1(0) + f_1(1), \tag{1}$$

then sends $r_1 \leftarrow \mathbb{F}$ to $\mathcal{P}$.

(2) In the i^{th} $(2 \le i < \ell-1)$ round, $\mathcal{P}$ sends a univariate polynomial

$$f_i(x_i) \stackrel{\text{def}}{=} \sum_{b_{i+1},\ldots,b_\ell\in\{0,1\}} f(r_1,\ldots,r_{i-1},x_i,b_{i+1},\ldots,b_\ell)$$

by sending the $\deg(f_i)+1$ coefficients of f_i. $\mathcal{V}$ verifies

$$f_{i-1}(r_{i-1}) = f_i(0) + f_i(1) \tag{2}$$

then sends $r_i \leftarrow \mathbb{F}$ to $\mathcal{P}$.

(3) In the ℓ^{th} round, $\mathcal{P}$ sends a univariate polynomial $f_\ell(x_\ell) \stackrel{\text{def}}{=} f(r_1,\ldots,r_{\ell-1},x_\ell)$, by sending $\deg(f_\ell)+1$ coefficients of f_ℓ. $\mathcal{V}$ verifies

$$f_{\ell-1}(r_{\ell-1}) = f_\ell(0) + f_\ell(1) \tag{3}$$

and accepts if and only if $f_\ell(r_\ell) = f(r_1,\ldots,r_\ell)$.

Despite the $O(2^\ell)$ time to compute the sum, the sumcheck protocol only takes $\mathcal{V}$ $O(\ell)$ time to verify it.

The GKR Protocol. At a high level, the GKR protocol processes a circuit of bounded-fan-in (say, fan-in 2) gates layer by layer. It encodes the computation on each layer of circuit as the sum of a bounded-degree (degree-2 for fan-in-2 circuits) multivariate polynomial over a binary hypercube. Assume circuit C has d layers of gates, hence $d+1$ layers of wires, and layer-i has 2^{s_i} wires. Let $V_i : \{0,1\}^{s_i} \mapsto \mathbb{F}$ be a function mapping layer i wire indices to wire values, and $\widetilde{V}_i : \mathbb{F}^{s_i} \mapsto \mathbb{F}$ be the multilinear extension of V_i. Let add_i (resp. $mult_i$) be a predicate function $\{0,1\}^{s_{i+1}} \times \{0,1\}^{s_{i+1}} \times \{0,1\}^{s_i} \mapsto \{0,1\}$ such that $add_i(x,y,z) = 1$ (resp. $mult_i(x,y,z) = 1$) if and only if wire indices x,y,z are respectively the left input, right input, and output wires of an addition (resp. multiplication) gate. Thus, for any wire index z on layer-i, $V_i(z)$ can be formulated as

$$V_i(z) = \sum_{x,y\in\{0,1\}^{s_{i+1}}} \Big(add_i(x,y,z)\big(V_{i+1}(x)+V_{i+1}(y)\big) + mult_i(x,y,z)V_{i+1}(x)\cdot V_{i+1}(y)\Big).$$

The GKR protocol begins with $\mathcal{V}$ picking a uniform $r \in \mathbb{F}^{s_0}$ and computing $\widetilde{V}_0(r)$ (recall that $\widetilde{V}_0$, the multilinear extension of V_0, can be computed from the final outputs of the circuit). Then, the parties run the sumcheck protocol on $\widetilde{V}_i(z)$ for all $i \in [d]$ to reduce the validity of $\widetilde{V}_i(r)$ to that of

$f_{i+1}(u_{i+1}, v_{i+1})$ where $u_{i+1}, v_{i+1} \in \mathbb{F}^{s_{i+1}}$ are picked in sumcheck rounds and

$$f_i(x,y) \stackrel{\text{def}}{=} \widetilde{add}_{i-1}(x,y,r)\left(\widetilde{V}_i(x) + \widetilde{V}_i(y)\right) + \widetilde{mult}_{i-1}(x,y,z)\widetilde{V}_i(x) \cdot \widetilde{V}_i(y) \tag{4}$$

$\mathcal{P}$ will send $\widetilde{V}_{i+1}(u_{i+1})$ and $\widetilde{V}_{i+1}(v_{i+1})$ to prove the equality.

To verify the validity of $\widetilde{V}_{i+1}(u_{i+1}), \widetilde{V}_{i+1}(v_{i+1})$, the two points will be combined, using uniform random coefficients $\alpha_{i+1}, \beta_{i+1}$, into a single random point [16]:

$$\begin{aligned}
&\alpha_{i+1}\widetilde{V}_{i+1}(u_{i+1}) + \beta_{i+1}\widetilde{V}_{i+1}(v_{i+1}) \\
&= \sum_{x,y \in \{0,1\}^{s_2}} \Big[\left(\alpha_{i+1}\widetilde{add}_{i+2}(x,y,u_{i+1}) + \right. \\
&\qquad \left. \beta_{i+1}\widetilde{add}_{i+2}(x,y,v_{i+1})\right) \cdot \left(\widetilde{V}_{i+2}(x) + \widetilde{V}_{i+2}(y)\right) + \\
&\qquad \left(\alpha_{i+1}\widetilde{mult}_{i+2}(x,y,u_{i+1}) + \beta_{i+1}\widetilde{mult}_{i+2}(x,y,v_{i+1})\right) \cdot \\
&\qquad \left(\widetilde{V}_{i+2}(x) \cdot \widetilde{V}_{i+2}(y)\right) \Big]
\end{aligned} \tag{5}$$

Thus, the validity of $\widetilde{V}_{i+1}(u_{i+1}), \widetilde{V}_{i+1}(v_{i+1})$ can be further reduced to a claim on $\widetilde{V}_{i+2}(u_{i+2}), \widetilde{V}_{i+2}(v_{i+2})$ for some uniform random u_{i+2}, v_{i+2} in $\mathbb{F}$, by sumchecking Equation (5). This procedure proceeds recursively towards the initial input layer, where $\widetilde{V}_d(u_d), \widetilde{V}_d(v_d)$ are checked against the multilinear extension polynomial defined by the inputs.

For circuits of fan-in-2 degree-d polynomial gates, the soundness error of GKR, $\mathcal{E}_{\text{GKR}} = 1 - \Pi_{i=0}^{d}(1 - d/|\mathbb{F}|)^{2s_i}$. In Libra, Xie et al. presented an linear algorithm for the prover [32]. Their result is later strengthened by a linear prover for general circuits [34]. If the predicates $add_i, mult_i$ can be computed in $O(s_i)$ time, then complexity of the GKR verifier is $O(\sum s_i)$.

2.3 Fully Linear PCP (FLPCP)

The idea of FLPCP that we use in our protocol was proposed by Boneh et al. [12]. Below we first explain a non-ZK FLPCP protocol, then shows how it can be efficiently transformed to offer ZK without using any secret multiplications.

Single-polynomial FLPCP without ZK. Let $G : \mathbb{F}^\ell \to \mathbb{F}$ be an ℓ-variable degree-d polynomial over $\mathbb{F}$, w_i for $i \in [\ell]$ be (potentially secret) input values, and $G(w_0, \ldots, w_{\ell-1}) = 0$ be the equation to prove. $\forall i \in [\ell]$, let p_i be a degree-1 polynomial defined by $p_i(0) = w_i$ and $p_i(1) \leftarrow \mathbb{F}$. Define

$$p_\ell(x) \stackrel{\text{def}}{=} G\left(p_0(x), \ldots, p_{\ell-1}(x)\right).$$

Then $p_\ell(x)$ is a polynomial of degree at most d. The protocol works as follows:

(1) $\forall i \in [\ell]$, $\mathcal{P}$ sends w_i, $p_i(1)$ to $\mathcal{V}$.
$\forall j \in [d+1]\backslash\{0\}$, $\mathcal{P}$ computes $q_j := p_\ell(j)$ and sends q_j to $\mathcal{V}$. $\mathcal{V}$ sets $q_0 := 0$.

(2) $\mathcal{V}$ picks and sends $r \leftarrow \mathbb{F}\backslash\{0\}$ to $\mathcal{P}$.

(3) $\mathcal{P}$ computes and sends $p_i(r)$ for all $i \in [\ell+1]$.
For each $i \in [\ell]$, $\mathcal{V}$ interpolates a value, which we call $p'_i(r)$, from points $(0, w_i), (1, p_i(1))$. Finally, $\mathcal{V}$ interpolates another value, which we call $p'_\ell(r)$, from points $(0, q_0), (1, q_1), \ldots, (d, q_d)$ and verifies that both of the following hold:

$$p'_i(r) = p_i(r),\ \forall i \in [\ell];$$
$$p'_\ell(r) = G(p_0(r), \ldots, p_{\ell-1}(r)).$$

It is straightforward to verify the completeness of the above protocol. For soundness, since G's degree is at most d, the protocol has soundness error $d/(|\mathbb{F}| - 1)$.

Single-polynomial FLPCP with ZK. If $w_0, \ldots, w_{\ell-1}$ are secret, to prove $G(w_0, \ldots, w_{\ell-1}) = 0$ in ZK, $\mathcal{P}$ and $\mathcal{V}$ simply execute the red-highlighted actions above using a linear ZK proof system. In our case, it suffices to use KKW-LO, a simplified KKW that efficiently proves linear operations because

- Sending secret values (e.g., "send $w_i, p_i(1)$" and "send q_j") now means treating the secrets as inputs of the circuit which will be proved in KKW-LO;
- Interpolating secret polynomials (e.g., $p'_i(\cdot)$) on a public point only involves linear operations on secret values;
- The final equality checks on $p'_i(r)$ for all $i \in [\ell+1]$ can be easily realized as checking public values of circuit output-wires using KKW-LO.

3 Our Approach

As we show in Table 2, the GKR, KKW, and FLPCP represent three distinct proof systems each featuring its own pros and cons. GKR allows sublinear verification and communication on uniform circuits but does not offer zero-knowledge; KKW offers ZK and easily handles (non-uniform) linear computations but is not *succinct* and is unwieldy to prove non-linear operations; FLPCP seems a solution lying somewhere in between: it offers a unique way to prove multiplications with square-root proof size, but requires linear-time verification and needs some external mechanism to provide ZK. In this work, we propose to combine all three protocols for the best of three worlds. The combination is not black-box, but it turns out that we can significantly simplify the building blocks before combining them, making it easier to analyze and implement.

3.1 Intuition and Key Ideas

Approach Overview. We illustrate the high-level idea of our approach in Figure 1. We assume the original computation to prove in ZK is represented as a uniform circuit of polynomial

Table 2: Compare proposed protocol and its building blocks.

	ZK	**Fast $\mathcal{V}$**	**Short $\lvert\pi\rvert$**	$\times$	$+$
GKR	☹	☺	☺	☹[1]	
KKW [24]	☺	☹[2]	☹	☹[5]	☺
FLPCP [12]	☺[3]	☹[2]	☺[4]	☺[6]	—
Proposed	☺	☺	☺	☺	☺

[1] Proving unstructured $\times$ and $+$ is slow; [2] Linear time;
[3] Need help from other ZKP systems to prove linear relations;
[4] Square-root communication; [5] Extra rounds to generate and verify multip. triples; making NIZK more expensive;
[6] No extra rounds, easy to compile to NIZK with Fiat-Shamir.

gates (Figure 1 (a)). We further assume the circuit has d layers, each containing (up to) m polynomial gates, and all final outputs are revealed.

Then we view the GKR and FLPCP protocols as two probabilistic transformations that guarantee the equivalence of computational correctness before and after the transformation, with perfect completeness and bounded soundness errors. More specifically, we first apply GKR to transform the original circuit (Figure 1 (a)) into one depicted in Figure 1 (b). As a result of this transform, the $d \cdot m$ polynomial gates are turned into $O(d \log m)$ linear gates, d polynomial gates, and $O(d \log m)$ extra inputs. These extra inputs are due to the univariate polynomials during sumcheck and the d points (one per layer) sent by the GKR prover to the verifier. The $O(d \log m)$ linear gates are due to the GKR verifier's computation to check the prover's sumcheck responses. The d polynomial gates come from the GKR verifier's work to verify $\left(\widetilde{V}_i(u_i), \widetilde{V}_i(v_i)\right)$ and $\left(\widetilde{V}_{i+1}(u_{i+1}), \widetilde{V}_{i+1}(v_{i+1})\right)$ are consistent as specified in Equation (4) and Equation (5). The soundness and cost of this GKR transform will depend on d, m, and the degree of the polynomial gates, which are application-specific.

Next, we apply FLPCP to transform the Figure 1 (b) circuit to the one in Figure 1 (b), which contains linear gates only. During the transformation, the d polynomial gates in Figure 1 (b) will be turned into $O(d \cdot \mathsf{deg}(G))$ extra inputs (due to the variables $\{q_j\}_{j \in [\mathsf{deg}(G)]}$ as defined in Step (1) of FLPCP) and $O(d)$ linear gates (due to the polynomial interpolation work in Step (5) of FLPCP). The soundness of this transformation is that of FLPCP. This final circuit after the FLPCP transform only contains linear operations, thus can be efficiently proved using a simplified KKW protocol.

Regarding the linear gates in the original circuit, we note that sometimes it pays to transform them through GKR and FLPCP too. If this is done, the white blocks of "Linear Gates" in all three sub-figures will completely disappear. Common benefits of transforming linear gates by GKR include: (1) achieving sublinear verification, even in the number of linear gates; and (2) avoiding introducing new witnesses that will eventually burden the MPC-in-the-Head protocol. Our verification circuit for multiple AES blocks (Section 4.1) serves an example where linear gates are also fully transformed for better performance.

Zooming into the actual steps we perform when combining the three components, it would be helpful to understand the concepts of "sending secret values in ZK" and how it allows us to bring ZK to GKR.

Sending Secrets in ZK. In our protocol, both GKR and FLPCP need some help from KKW to prove linear operations on secrets in ZK. In particular, KKW is used whenever some sensitive values needs to flow from $\mathcal{P}$ to $\mathcal{V}$. We treat "sending secrets" as introducing secret inputs to a (public) circuit whose logic is already fully specified by the computation $\mathcal{V}$ should run over the secrets in the respective protocols (i.e., GKR and FLPCP). Therefore, instead of sending the secrets to $\mathcal{V}$, $\mathcal{P}$ just sends the *input correction records* (see Figure 2) corresponding to these secrets, so that $\mathcal{P}$ can no longer change them (without being caught with certain probability) and $\mathcal{V}$ can run their checks in ZK using the simplified KKW protocol. We note the difference between this treatment of secrets and a commitment of secrets: unlike commitments, here the secrets will never be revealed while commitments are supposed to be de-committed at some point. However, this treatment still allows computation to happen on those secrets and finally an MPC-in-the-Head style of cut-and-choose is used to establish the soundness guarantee.

Bring ZK to GKR.[4] To turn GKR zero-knowledge, $\mathcal{P}$ simply applies the "send secrets in ZK" idea to all confidential values flowing from $\mathcal{P}$ to $\mathcal{V}$. These may include the secret inputs to the original computation, all of the polynomial coefficients sent during the sumcheck, and the $\widetilde{V}_i(u), \widetilde{V}_i(v)$ values sent at the end of each layer's sumcheck. Then $\mathcal{V}$'s (public) GKR verify algorithm will be executed in KKW, allowing $\mathcal{V}$ to accomplish verification with a slightly larger (but still bounded) soundness error but without leaking the secrets. The circuit realizing $\mathcal{V}$'s verify algorithm consists mostly of linear operations. The only non-linear operation on secret values is due to checking Equation (4), which occurs only once per non-linear layer of the circuit. As an optimization, we leveraged every linear equation between the secret values to save transmitting one secret value in ZK, by representing each saved secret as a linear combination of other secrets.

Use Simplified FLPCP. Thanks to GKR, only one polynomial per layer needs to be proved. Thus, it suffices to invoke *single-polynomial ZK-FLPCP* (see Section 2.3) for circuit-depth number of times.

Use Simplified KKW. Since multiplications can be entirely eliminated by GKR and FLPCP, we can specialize KKW to just handle linear computations, i.e., secret addition/subtraction and constant multiplication. This can signif-

[4] The GKR paper [22, Section 5] was later updated with a theoretical construction of ZK-GKR based on generic ZKP and commitment schemes. In contrast, our construction is very different and more efficient.

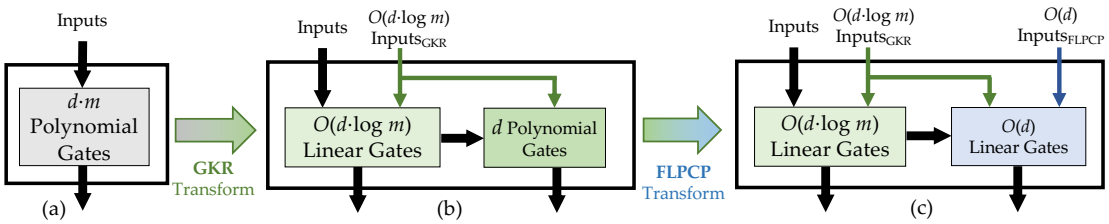

Figure 1: Approach Overview

(All inputs are secret and all outputs are public.)

icantly reduce the complexity of the original KKW: because there is no need to generate and verify multiplication triples via cut-and-choose, KKW's 5-round interactive HVZK naturally reduces to 3-round. Therefore, it is much cheaper to convert the simplified KKW to an NIZK via Fiat-Shamir transform. (In comparison, the original KKW obtained its 3-round version by paying a heavy price on the prover-side: requiring the prover to evaluate the circuit M times and commits all M transcripts even if only τ transcripts are eventually checked and $M \gg \tau$. For 128-bit computational security, $(M = 916, \tau = 20)$ is needed.)

Non-blackbox Combination of Three. Originally, GKR, KKW, and FLPCP each would work with a different finite field and has their own parameters such as the number of iterations to achieve their respective soundness guarantees. When combining them, we use the same finite field in all three protocols. We then analyze the soundness of one iteration of the combined protocol, and finally boost its soundness through repetition.

3.2 The Main Protocol

KKW-LO. KKW-LO (Figure 2) is an honest-verifier ZK (HVZK) derived by specializing KKW for only proving linear operations. In the first round, $\mathcal{P}$ emulates the linear operations with n parties in its head, then "commits" its emulation by sending ICR (the correction bits for all secret inputs to C) and a random string h (obtained from calling the random oracle with all the secret-shares of the output of C). Then, $\mathcal{V}$ randomly chooses $n-1$ parties to check. Finally, $\mathcal{P}$ reveals the seeds of $n-1$ circuits so that $\mathcal{V}$ can verify that $\mathcal{P}$ executed honestly in the emulation.

Completeness of KKW-LO can be easily verified from its construction. KKW-LO has "2-special" soundness since it is easy for $\mathcal{V}$ to recover $\mathcal{P}$'s witness if $\mathcal{P}$ had opened a single emulation in two different but convincing ways. The soundness error of each iteration of KKW-LO is $1/n$. KKW-LO provides honest-verify zero-knowledge because given an honest $\mathcal{V}$'s challenge r, a simulator without the witness can easily generate accepting transcripts indistinguishable from those produced in real interactions between a $\mathcal{P}$ holding the witness and $\mathcal{V}$. A full security proof of KKW-LO can also be specialized from the proof of original KKW.

FLPCP-KKW. FLPCP-KKW (Figure 3) efficiently proves circuits with nonlinear computation. It combines a simplified FLPCP for proving individual polynomial gates (Section 2.3) with KKW-LO, assuming the simplified FLPCP and KKW-LO use the same field $\mathbb{F}$. The first three rounds of FLPCP-KKW apply single-polynomial FLPCP to transform circuit C into an equivalent, augmented verification circuit that contains linear operations only. Then KKW-LO is invoked to prove the augmented circuit in zero-knowledge. Unlike most existing MPC-in-the-Head protocols such as [15, 20, 24], FLPCP-KKW can directly work for circuits with polynomial gates.

The verifier in FLPCP-KKW only uses public-coins. Each iteration of FLPCP-KKW has soundness error at most $1 - (1 - \frac{\deg(G)}{|\mathbb{F}|-1}) \cdot (1 - \frac{1}{n})$, because a malicious $\mathcal{P}$ needs to subvert either FLPCP or KKW-LO to corrupt an iteration.

GKR-FLPCP-KKW. (Figure 4) The basic idea resembles that of FLPCP-KKW, except that now we will apply GKR to compile a log-space uniform circuit C into a circuit that is statistically equivalent for the purpose of verifying the NP relation, but only needs d (circuit-depth) polynomial gates. All secret values of the GKR prover are augmented as extra inputs whose correction records are immediately sent to $\mathcal{V}$, and the constraints involving secrets that the GKR verifier needs to check become linear and polynomial gates in the augmented circuit. Finally, FLPCP-KKW is invoked to prove the augmented circuit in ZK. We stress that all three sub-protocols will operate on the same field $\mathbb{F}$, and the verifier only needs public coins to derive its challenges.

Costs. GKR-FLPCP-KKW requires $O(d \log C)$ rounds for GKR transform, 1 round for FLPCP transform to handle all d polynomials, and 1 final round for executing KKW-LO. For log-space uniform application circuits, the time and communication costs of GKR-FLPCP-KKW only depends on the size of the linear circuit eventually fed into KKW-LO, which is $O(d \log C)$.

Limitation. Since verification and communication costs of MPC-in-the-Head protocols grow linearly with the number of secret inputs, this is in contrast to Virgo/Virgo++ which allow logarithmic verification and proof size thanks to the efficient LDT. Therefore, our approach may not perform well for computations involving large amount of secrets, or those

Security parameters: n MPC parties; H is the random oracle.
Public Input: A circuit C with just linear operations on $\mathbb{F}$.
Secret Input: Only $\mathcal{P}$ knows the secret inputs to C.

Round 1:

(1) $\mathcal{P}$ samples uniform $seed_i$ and its commitment $\mathsf{comSeed}_i \leftarrow \mathrm{Com}(seed_i)$ for $i \in [n]$;

(2) $\mathcal{P}$ traverses C in a topological order:

(a) For each secret input value w on a wire with id wid,

i. $\forall i \in [n], w_i := \mathsf{PRG}(seed_i, wid)$;

ii. $ICR[wid] := w - \sum_{j\in[n-1]} w_j$;
//$ICR[wid]$ is called the input correction record of wire wid.

iii. $w_0 := w_0 - ICR[wid]$;

(b) For each addition gate with inputs u, v and output w, set $w_i := u_i + v_i, \forall i \in [n]$.

(c) For each constant multiplication gate with input v, public constant a, and output w, set $w_i := a \cdot v_i, \forall i \in [n]$.

(d) For each constant addition gate with input v, public constant a, and output w, set $w_i := \begin{cases} a + v_i, & i = 0 \\ v_i, & \text{otherwise} \end{cases}$.

(3) For final output w of C, $\mathcal{P}$ computes $h := H\left(\{w_i\}_{\forall i\in[n]}\right)$.

(4) $\mathcal{P}$ sends ICR, h and $\{\mathsf{comSeed}_i\}_{i\in[n]}$ to $\mathcal{V}$.

Round 2: $\mathcal{V}$ calls H to sample $r \leftarrow [n]$ and sends it to $\mathcal{P}$.
Round 3:

(1) $\forall i \in [n]\backslash\{r\}$, $\mathcal{P}$ decommits $seed_i$. For final output w of C, $\mathcal{V}$ computes w_i from $(seed_i, ICR)$ as $\mathcal{P}$ did in Round 1.

(2) $\mathcal{V}$ computes $w_r := w - \sum_{i\in[n]\backslash\{r\}} w_i$;

(3) $\mathcal{V}$ verifies the recovered w_is are consistent with h.

Figure 2: KKW-LO: A 3-round HVZK for Linear Operations

using lots of secret witness to flatten the verification circuits.

Security. GKR-FLPCP-KKW, in essence, is a $O(d \log C)$-round, public-coin interactive oracle proof (IOP) [7, 18] where the circuit to be proved by a classic Σ-protocol (KKW-LO) is dynamically generated in its first $O(d \log C)$ rounds due to GKR and FLPCP. The security of GKR-FLPCP-KKW can be proved based on the security of KKW and the fundamental theorem of algebra (as was used in the security proof of GKR and FLPCP), see Appendix C.

Theorem 3.1. Given an NP relation defined by R, τ iterations of GKR-FLPCP-KKW is a public-coin, honest-verifier zero-knowledge argument of knowledge for R, with soundness error $[1 - (1 - \mathcal{E}_{\mathrm{GKR}}) \cdot (1 - \mathcal{E}_{\mathrm{FLPCP}}) \cdot (1 - \mathcal{E}_{\mathrm{KKW}})]^\tau$ where $\mathcal{E}_{\mathrm{GKR}}, \mathcal{E}_{\mathrm{FLPCP}}, \mathcal{E}_{\mathrm{KKW}}$ are the soundness errors due to the GKR transform, the FLPCP transform, and the KKW-LO proof resp. in one iteration of GKR-FLPCP-KKW.

Soundness. In the proof of the theorem (See Appendix C), we derive the overall soundness of our protocol, expressed as the formula of $\mathcal{E}_{\mathrm{GKR}}, \mathcal{E}_{\mathrm{FLPCP}}, \mathcal{E}_{\mathrm{KKW}}, \tau$. In our protocols, since $\mathcal{P}$ emulates n parties in KKW-LO, we thus have $\mathcal{E}_{\mathrm{KKW}} = 1/n$. $\mathcal{E}_{\mathrm{FLPCP}}$ and $\mathcal{E}_{\mathrm{GKR}}$, however, are application-dependent: As we've shown in Section 2.3, a call to FLPCP will introduce $\deg(G)/(|\mathbb{F}| - 1)$ soundness error where G is the polynomial gate that can vary with application; in addition to G, $\mathcal{E}_{\mathrm{GKR}}$ is also affected by the shape of the circuit, i.e., the number of gates in a circuit layer, the number of layers, etc. We will detail the calculation of $\mathcal{E}_{\mathrm{FLPCP}}$ and $\mathcal{E}_{\mathrm{GKR}}$ with respect to each circuit and application in Section 4 and Section 5.

Security parameter: A finite field $\mathbb{F}$.
Public Input: A circuit C of linear and polynomial gates on $\mathbb{F}$.
Secret Input: Only $\mathcal{P}$ knows the secret inputs to C.

Round 1:

(1) For every polynomial gate G in C, $\mathcal{P}$ and $\mathcal{V}$ run Step (1) of the single-polynomial FLPCP protocol (Section 2.3), augmenting C with additional secret input wires corresponding to the variables $\{q_j\}_{j\in[d]}$ in the FLPCP protocol.

(2) $\mathcal{P}$ computes and sends ICR for the augmented new inputs as specified by Step (4) in Round 1 of KKW-LO (Figure 2).

Round 2: $\mathcal{P}$ and $\mathcal{V}$ run Step (4) of single-polynomial FLPCP.
Round 3:

(1) $\mathcal{P}$ and $\mathcal{V}$ run Step (5) of the single-polynomial FLPCP protocol, augmenting C with linear gates (for interpolating the secret polynomials) and outputs (to reveal $p'_i(r)$ in the FLPCP protocol).

(2) $\mathcal{P}$ runs Round 1 of KKW-LO for the augmented circuit C.

Round 4: $\mathcal{P}$ and $\mathcal{V}$ run Round 2 of KKW-LO for augmented C.
Round 5: $\mathcal{P}$ and $\mathcal{V}$ run Round 3 of KKW-LO for augmented C.

Figure 3: FLPCP-KKW: An HVZK from FLPCP and KKW-LO

Removing Interaction. Like many existing ZKP systems [9, 18, 32, 34, 35], we can apply the Fiat-Shamir heuristic [19] to obtain NIZK arguments of knowledge. Since our protocol is a random-coin IOP, we used the same idea and algorithm used by Banquet [5] and Limbo [18] to calculate the number of iterations needed to achieve certain computational security. The calculation is done through plugging the soundness errors of the rounds in our protocol into the search algorithm as described in [5, Section 6.1] that works for any public-coin interactive oracle proofs. Computational security is measured by the expected number of random oracle queries needed to break the IOP with high probability after the Fiat-Shamir transform. Correctness of their algorithm is proved in [5, Section 5, Lemma 2].

4 Improved Verification Circuits

In this section, we describe the design of our AES, membership and SHA256 circuits.

Security parameter: A finite field $\mathbb{F}$.
Public Input: A log-space uniform circuit C consisting of linear and polynomial gates over $\mathbb{F}$.
Secret Input: Only $\mathcal{P}$ knows the secret inputs to C.

(1) $\mathcal{P}$ and $\mathcal{V}$ run the GKR protocol (see Section 2.2):

(a) In every sumcheck (called by GKR) round when $\mathcal{P}$ needs to send the univariate polynomial $f_i(\cdot)$ (in any of the three steps of sumcheck (Section 2.2)), both parties treat the coefficients of f_i as secrets and augment C with secret input wires representing these coefficients, and augment C with linear operations to check one of Equation (1), Equation (2), Equation (3) based on the exact round of the sumcheck. $\mathcal{P}$ sends the *ICR* (defined in Figure 2) for the augmented input wires.

(b) Every time after sumchecking a layer of C, to prove Equation (4) in ZK, $\mathcal{P}$ and $\mathcal{V}$ augment C with new secret input wires representing $\widetilde{V}_i(u_i), \widetilde{V}_i(v_i)$, and augment C with new linear and polynomial gates for verifying the equation in ZK. $\mathcal{P}$ sends the *ICR* for the augmented input wires.

(2) $\mathcal{P}$ and $\mathcal{V}$ run FLPCP-KKW (of Figure 3) to prove C.

Figure 4: GKR-FLPCP-KKW: An efficient interactive HVZK via combining GKR and FLPCP-KKW

4.1 AES

AES is a NIST standard symmetric-key cipher. It allows three configurable cipher key lengths—128, 192, or 256 bits, hence are referred to as AES-128, AES-192, AES-256, resp. All variants use 128-bit input blocks (or 16 bytes). Take AES-128 as an example, it has 10 rounds and uses a Key Expansion module to expand a 128-bit key into 11 128-bit round-keys. Within each round, there are a sequence of SubByte, ShiftRows, MixColumns, and AddRoundKey steps:

SubByte Each of the 16 input bytes is substituted by another byte based on its value. SubByte is commonly implemented by indexing a public table of 256 entries.

ShiftRows Viewing the 16 input bytes as a 4×4 matrix, ShiftRows cyclically shifts the last three rows by an offset of 1, 2, 3, resp.

MixColumns Viewing the 16 input bytes as a 4×4 matrix, MixColumns treats each column of 4 bytes as a polynomial over $\mathbb{F}_{2^8}$ in the polynomial ring modulo x^4+1, and transforms each column of the matrix by ring-multiplying it with a constant polynomial in the ring.

AddRoundKey Treating the input as 16 bytes and XOR them with a 16-byte round-key.

The Key Expansion consists mostly of bit-XORs but also a number of SubBytes (e.g., 40 SubByte for AES-128). The structure of AES-192 and AES-256 are very similar to AES-128, but run with slightly different constants in the algorithms. We refer readers to the FIPS-197 [26] for detailed description of the standard.

Using Extra-Witness. Like many useful algorithms, it is easier to *verify* than to *compute* an AES. Note the only non-linear part of AES computation is SubByte. Given an input byte $b \in \mathbb{F}_{2^8}$, SubByte consists of two steps:

(1) A special inversion: $b' := \begin{cases} b^{-1} & b \neq 0; \\ 0 & b = 0 \end{cases}$

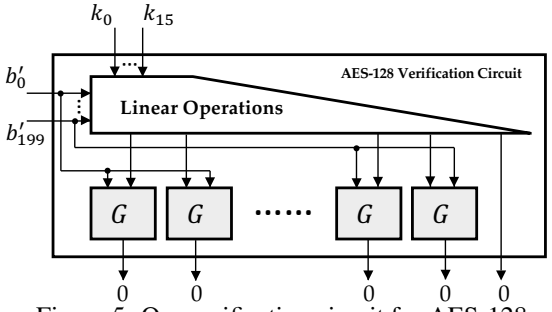

Figure 5: Our verification circuit for AES-128

(2) Treat b' as a (column) vector of 8 bits and compute $b'' = A \cdot b' + C$ with constant matrix $A \in \{0,1\}^{8\times 8}$ and constant vector $C \in \{0,1\}^{8\times 1}$.

$\mathcal{P}$ can supply b' as an additional witness, so correctness of step (1) is equivalent to $(b' \cdot b = 1) \vee (b = 0 \wedge b' = 0)$. I.e.,

$$\big((b' \cdot b - 1) \cdot b' = 0\big) \wedge \big((b' \cdot b - 1) \cdot b = 0\big). \tag{6}$$

As an optimization, we define $G(x,y) \stackrel{\text{def}}{=} x \cdot (xy-1)$, then each special inversion can be verified using two polynomial gates, ensuring $(G(b',b) = 0) \wedge (G(b,b') = 0)$.

For step (2), note that given any secret byte represented as additive shares in $\mathbb{F}_{2^8}$ (which is exactly the case using KKW-LO), it is free to convert between a secret byte and its 8 secret bits (in both directions) because addition in $\mathbb{F}_{2^8}$ happens to be bitwise-XOR hence concatenation of 8 $\mathbb{F}_2$-shares is exactly an $\mathbb{F}_{2^8}$-share. Therefore, that affine transformation only involves linear operation, as are ShiftRows, MixColumns, AddRoundKey steps.

Figure 5 depicts the verification circuit for AES128 before the GKR and FLPCP transformations take place. Here we only draw the secret inputs explicitly: $k_0, \ldots, k_{15}$ are the 16 bytes of secret key while $b'_0, \ldots, b'_{199}$ are the 200 bytes of witness for special inversion. Plaintext values like the 16-byte

input message are much cheaper to handle thus are omitted. It is also assumed that all outputs of the circuit are 0. Note that each b'_i is sent to two G gates, once as the left input and once as the right input.

Therefore, each block of AES-128 verification circuit needs 200 bytes of secret witness, which are fed into 400 bivariate polynomial gates. By GKR, these 400 polynomial gates can be transformed into checking the polynomial $V_0(z) : \mathbb{F}_2^9 \mapsto \mathbb{F}_{2^8}$ where

$$V_0(z) = \sum_{x,y \in \{0,1\}^9} g(x,y,z) \cdot G(V_1(x), V_1(y))$$

where g is the predicate function for gate G. This will require 18 sumcheck rounds. As an optimization, we exploit the pattern in predicate g and rearrange the indices x, y such that $g(x,y,z) = 1$ *iff* x, y differ only in their last bit. So we can rewrite $V_0(z)$ as

$$V_0(z) = \sum_{x \in \{0,1\}^8} g'(x, z_0 \dots z_7) \cdot \big((1-z_8)G(L(x)) + z_8 G(R(x))\big)$$

where z_i denotes the i^{th} bit of z and

$$g'(x,z) \stackrel{\text{def}}{=} \prod_{i=0}^{7} (x_i z_i + (1-x_i)(1-z_i)),$$
$$L(x) \stackrel{\text{def}}{=} (V_1(x\|0), V_1(x\|1))$$
$$R(x) \stackrel{\text{def}}{=} (V_1(x\|1), V_1(x\|0)).$$

Now the sumcheck can be done in 8 rounds, each of which involves a degree-4 polynomial.

Thanks to GKR, the layer of 400 polynomial gates in AES-128 can be transformed into 8 linear equations and 1 degree-4 polynomial equation over $4 \cdot 8 = 32$ new secret inputs. The single polynomial gate will then be transformed by FLPCP into 2 additional linear equations (for interpolating 2 secret polynomials at plaintext points) over 4 new secret inputs. This final linear circuit will be handled by KKW-LO.

Since GKR encodes the 400 zero outputs of the G gates with a 9-variable multi-linear polynomial and checks them using a random point on its multilinear extension, this introduces a soundness error of $9/|\mathbb{F}|$. Next, the soundness error due to sumchecking $V'_0(z)$ is at most $(1-4/|\mathbb{F}|)^8$. Thus, $\mathcal{E}_{\text{GKR}}(\text{AES-128}) \coloneqq 1 - (1-9/|\mathbb{F}|)(1-4/|\mathbb{F}|)^8$. Because FLPCP is invoked only once for checking a degree-4 polynomial, $\mathcal{E}_{\text{FLPCP}} = \frac{4}{|\mathbb{F}|-1}$.

For AES-based signatures, we opt to not use GKR but directly verify the 400 polynomial gates with the batched version of FLPCP (Appendix B.2) and KKW-LO, which is concretely more efficient since those GKR rounds can be avoided.

Without Extra-Witness. A drawback of the witness-based AES verification circuit is that its computation and communication costs will grow linearly with the number of AES blocks, In scenarios when many AES blocks need to be proved, e.g., running AES in the counter-mode, its performance doesn't scale up well. Hence, we propose a witness-free circuit design aiming to prove m blocks of AES using only $O(\log m)$ verification time and bandwidth.

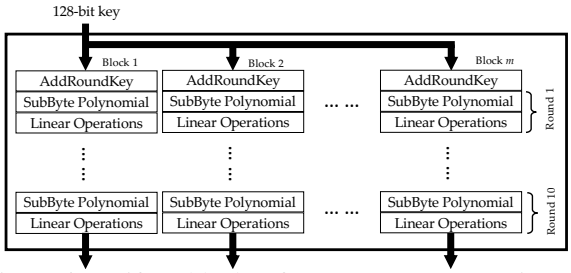

Figure 6: Verify m blocks of AES-128 (no extra witness)
(Plaintext inputs and the key-expansion module are omitted.)

First, we treat each SubByte as 8 table-lookup operations each on a constant table of 256 entries of a single bit. We encode each of the 8 tables as an 8-variable polynomial. That is, let $\{t_i\}_{i \in [256]}$ be bits in a constant table, the multilinear polynomial $f : \mathbb{F}_2^8 \mapsto \mathbb{F}_2$ encoding the table is

$$f(x_0, \dots, x_7) \stackrel{\text{def}}{=} \sum_{p \in [256]} \left(t_p \cdot \prod_{i \in [8]} \chi_{p_i}(x_i) \right)$$

where p_i denotes the i^{th} bit of p and χ_0 (resp. χ_1) is the univariate polynomial $\chi_0(x) = 1 - x$ (resp. $\chi_1(x) = x$). Thus, a table lookup using an 8-bit index is essentially a polynomial gate that can be handled by our protocol. This polynomial can be realized by 3 layers of fan-in-2 multiplications (to compute $\prod_{i \in [8]} \chi_{p_i}(x_i)$) and 1 layer of additions (to compute the sum in f). In addition, we apply symbolic execution to merge all consecutive layers of linear operations (including step (2) of SubByte, and subsequent MixColumns, AddRoundKey, ShiftRows steps) into a single layer. The merged layer of linear gates has 128 output wires per block while the entire layer across all m blocks can be transformed into logarithmic number of linear operations using GKR. Namely, let $U, V : \{0,1\}^{7+\log m} \mapsto \{0,1\}$ be the input and output wire value functions of this layer, and $pred(x,z)$ the predicate that returns 1 if and only if $U(x)$ is a linear component of $V(z)$, then it suffices to verify $V(z) \stackrel{\text{def}}{=} \sum_{x \in \{0,1\}^{7+\log m}} pred(x,z)U(x)$ in GKR for this layer. Thus the entire m blocks of AES can be efficiently verified using $O(\log m)$ work (Figure 6).

This circuit has 11 linear layers and 3×10 multiplication layers. Each linear layer has $128m$ linear gates. The 3 multiplication layers in each round have $4 \times 256 \times 16m$, $2 \times 256 \times 16m$, $256 \times 16m$ multiplication gates, respectively. For the $128m$ output bits, checking a random point of the multilinear extension encoding the outputs introduces a soundness error

of $(7+\log m)/|\mathbb{F}|$. Hence $\mathcal{E}_{\text{GKR}}(m\text{AES})$ is

$$1-[1-(7+\log m)/|\mathbb{F}|]\,[1-2\cdot 2\cdot(14+\log m)/|\mathbb{F}|]^{10}$$
$$[1-2\cdot 2\cdot(13+\log m)/|\mathbb{F}|]^{10}[1-2\cdot 2\cdot(12+\log m)/|\mathbb{F}|]^{10}$$
$$[1-2\cdot(7+\log m)/|\mathbb{F}|]^{11}.$$

Each multiplication layer requires checking a degree-2 polynomial using FLPCP. For a total of 30 multiplications, we have $\mathcal{E}_{\text{FLPCP}}(m\text{AES}) := 1-(1-2/(|\mathbb{F}|-1))^{30}$.

4.2 Membership

We define the *private membership proof* problem as follows: Given a public set $\{ID_i\}_{i\in[m]}$ where ID_i's are unique identifiers (e.g., 16-byte long in the context of ring signatures), the goal is for $\mathcal{P}$ who holds a secret certificate of its identifier ID to convince $\mathcal{V}$ that its ID belongs to a public set without leaking ID nor its certificate. In the context of designing ring signatures, a user's secret key can serve as the certificate.

Katz et al. [24] proposed a solution to the private membership problem using Merkle trees. They assumed the public set is publicized as the root of the Merkle tree with leafs being member IDs; then the prover presents a ZKP of a path of hashes from the leaf of its own ID to the root. For a size-m set, their approach requires proving $\log m$ hashes. Since standard hashing algorithms (like SHA3 and SHA256) can be slow to prove in ZK, they used LowMC, a non-standard MPC-friendly cipher.

We chose a simpler polynomial-based approach to private membership proof that exhibits significantly better performance for small to medium size sets (e.g., set size ≤ 8192). Let $f(x)$ be a degree-m polynomial defined as

$$f(x) \stackrel{\text{def}}{=} \prod_{i\in[m]}(x-ID_i). \tag{7}$$

The membership of a private $ID \in \{ID_i\}_{i\in[m]}$ is essentially a proof of $f(ID)=0$. Thanks to GKR, the $O(m)$ multiplications can be verified this way using $O(\log m)$ linear operations, which are much cheaper than hashes. In practice, it also makes sense to avoid operations on large field elements (e.g., $\mathbb{F}_{2^{128}}$) by breaking the 128-bit IDs into smaller pieces, e.g., 8 $\mathbb{F}_{2^{16}}$ elements or 4 $\mathbb{F}_{2^{32}}$ elements, and projecting each 128-bit ID onto a $\mathbb{F}_{2^{16}}$ or $\mathbb{F}_{2^{32}}$ element using random linear combination, so Equation (7) can be proved on smaller fields. Although this can introduce $m/|\mathbb{F}|$ soundness error due to collision, we note that when $m \ll |\mathbb{F}|$, the degradation in soundness can be easily compensated by slightly increasing the number of iterations.

Our circuit has $\log m-1$ layers where the i-th layer has 2^i multiplication gates for all $i \in [\log m-1]$. The circuit has only one output. Thus, the soundness error all comes from sumcheck: $\mathcal{E}_{\text{GKR}}(\text{MBR}_m) := 1-\prod_{i=0}^{\log m-1}(1-2\cdot 2\cdot i/|\mathbb{F}|)$. Since $\log m-1$ multiplications are checked by FLPCP, hence $\mathcal{E}_{\text{FLPCP}}(\text{MBR}_m) := 1-(1-2/(|\mathbb{F}|-1))^{\log m-1}$. Overall, we have $\mathcal{E}(\text{MBR}_m) = 1-(1-\mathcal{E}_{\text{GKR}}(\text{MBR}_m))(1-\mathcal{E}_{\text{FLPCP}}(\text{MBR}_m))(1-m/|\mathbb{F}|)$.

4.3 SHA256

We used a binary circuit from [29] for SHA256, which was due to the improvement by [14]. The circuit contains 22573 AND gates, 110644 XORs and 1856 NOTs. To leverage the efficiency of GKR, we augmented the circuit with 22573 secret input bits, each corresponding to the output of an AND gate, so that all AND gate can be placed in a single layer to be batch-processed by GKR. Since every AND has 2 inputs and it suffices to send two secret elements per sumcheck round, GKR is able to reduce the 22573 ANDs to $\lceil\log 22573\rceil\times 2\times 2+2=62$ new secret inputs and a number of linear operations over them, plus a single secret multiplication which can be further translated via FLPCP to 2 additional secret inputs and several linear operations over them.

This approach to obtain a circuit for SHA256 is probably less than ideal, leveraging (almost) no domain-specific features. However, the strategy of introducing a secret bit witness per AND gate to allow efficient batched proof of all ANDs represents a simple, generic technique that is applicable to proving any binary circuit computation using our protocol. So the performance of our protocol on SHA256, although not ideal, could provide some data-points about the efficiency of our approach when little domain knowledge is used in the circuit design. It turns out our ZKP for SHA256 is also very competitive, especially in proof size and prover's time (Table 8).

The GKR transform of a single layer of 22573 multiplications introduces $2\cdot 2\cdot\lceil\log 22573\rceil/|\mathbb{F}|$ soundness error in sumcheck and $\lceil\log 22573\rceil/|F|$ in checking the outputs. Thus, $\mathcal{E}_{\text{GKR}}(\text{SHA256}) := 1-\left(1-\frac{2\cdot 2\cdot 15}{|\mathbb{F}|}\right)\left(1-\frac{15}{|\mathbb{F}|}\right)$. FLPCP only needs to handle one multiplication, hence $\mathcal{E}_{\text{FLPCP}}(\text{SHA256}) := 2/(|\mathbb{F}|-1)$.

5 Applications and Experiments

We implemented Dubhe protocols in C++ with field operations optimized using SIMD instructions: Techniques described in [27] allow us to perform 16 parallel $\mathbb{F}_{2^{16}}$ multiplications stored in an `_mm256` data using AVX2 instructions. However, we did not use multi-threading in our implementation, which can be an interesting future work to further reduce time costs. We aim at achieving 128-bit computational security in all experiments, unless explicitly specified otherwise. For interactive protocols, the statistical security parameter s varies for easier comparison, and is specified per experiment. All protocols (both ours and various baselines) were run on the same machine with an AMD Ryzen 5800X CPU and 32GB RAM unless noted otherwise.

Table 3: AES-based Identification ($s = 100$)

Protocol	$T_{\mathcal{P}}$ (ms)	$T_{\mathcal{V}}$ (ms)	Comm. (KB)
QuickSilver	334	334	1644
Virgo	2265	21.4	174
Virgo++	751	36	132
Limbo ($n = 16, \tau = 10$)	2.7	2.5	10
Dubhe ($n = 16, \tau = 11$)	2.8	2.0	9.2
Limbo ($n = 128, \tau = 6$)	12	11	5.8
Dubhe ($n = 128, \tau = 6$)	6.6	6.0	6.1

Limbo made a non-standard use of AES; others used standard AES.

Table 4: AES-based Signature Schemes

k	Protocol	T_{Sign} (ms)	T_{Verify} (ms)	$\lvert\sigma\rvert$ (KB)
101	Virgo (18 layers, 2^{14} inputs)	2265	21	174
103	Virgo++ (243 layers, 2^{8} inputs)	49	55	775
104	Virgo++ (62 layers, 2^{11} inputs)	78	26	194
101	Virgo++ (9 layers, 2^{14} inputs)	409	32	129
100	Virgo++ (5 layers, 2^{15} inputs)	751	36	132
99	Virgo++ (2 layers, 2^{16} inputs)	1554	54	140
127	Limbo ($n = 16, \tau = 40$)	3.6	2.5	21
128	Dubhe ($n = 16, \tau = 58$)	4.8	4.0	30
133	SPHINCS$^+$-128 (smaller $\lvert\sigma\rvert$)	98	0.3	7.7
128	SPHINCS$^+$-128 (faster sign)	4.9	0.4	17
256	ZKB++	100	70	469
256	KKW	110	110	182
256	Ligero++	256	56	224
253	Limbo ($n = 16, \tau = 82$)	15	12	82
256	Dubhe ($n = 16, \tau = 120$)	15	11	113
255	SPHINCS$^+$-256 (smaller $\lvert\sigma\rvert$)	164	0.4	29
255	SPHINCS$^+$-256 (faster sign)	17	0.7	49

Data for ZKB++, KKW, and Ligero++ are taken from [9] which used a machine with 512GB memory, 16 times more than ours.

5.1 Identification and Signature Schemes

Identification Schemes. An identification scheme is an interactive protocol that allows a prover to prove its identity to a verifier who knows the public key of the prover. It can be efficiently built from any ZK argument of knowledge protocol and the standard AES ciphersuite: (1) To generate a key pair, a user with identifier $ID_u \in \{0,1\}^{128}$ picks a uniform private key $sk \leftarrow \{0,1\}^{128}$, computes $pk := AES_{sk}(ID_u)$; (2) To prove the identity, the user proves in ZK that he/she knows the sk such that $pk = AES_{sk}(ID_u)$.

The circuit for our AES-based identification was described in Section 4.1, where $\mathcal{F}_{\text{GKR}}, \mathcal{F}_{\text{FLPCP}}$ were also given. We used the field $\mathbb{F}_{2^{16}}$ in our protocol. Table 3 compares our signature scheme with several prior implementations including ZKB++, KKW, Ligero++, Limbo, and Virgo/Virgo++, where all protocols are aligned to achieving 100-bit statistical security. We used the AES circuit from [29], which contains 6400 ANDs, 28176 XORs and 2087 NOTs, to run experiments for Virgo, Virgo++ and QuickSilver [33]. The verification circuit for Limbo reflects a non-standard use of AES. Our protocols are 1–2 orders-of-magnitude more efficient than Virgo, Virgo++, and QuickSilver. Comparing to Limbo, our identification schemes are still highly competitive even if our AES circuit has 3× more multiplications than theirs.

Signature Schemes. A signature scheme can be realized using NIZK and AES:

KeyGen This is the same as that for identification.

Sign To sign a message m, a user with (ID_u, sk) generates a NIZK proof σ of the fact $pk = AES_{sk}(ID_u)$, using randomness from $H(m)$ where H denotes a random oracle.

Verify To verify σ is a valid signature of m signed by a user ID_u with public key pk, call the NIZK verify algorithm and accepts if and only if it accepts the proof.

Signature schemes built this way are quantum-resistant by far, and feature smaller keys, faster key generation, and faster signing than best cryptographic hash-based constructions (e.g., XMSS [13] and SPHINCS [8]). Unlike in AES-based identification, our proof for signatures skipped the GKR transformation for better concrete efficiency.

We compare our signature schemes to baselines constructed from several state-of-the-art ZKPs (Table 4). Comparing to Virgo/Virgo++, our protocol significantly outperforms Virgo/Virgo++ based signatures in all aspects meanwhile offering higher security guarantee. The security degradation in Virgo/Virgo++ is because the soundness error introduced by the LDT subroutine (see [6, Theorem 3.3, Formula (8)]) depends on the structure of the flattened AES circuits, causing their computational security to fluctuate from 99-bit to 104-bit. Different ways to flatten the AES circuit require different number of witnesses to be added so the computation can be verified in less number of layers.

ZKB++, KKW, and Ligero++ are three representative implementations of MPC-in-the-Head ZKP protocols. To compare with these protocols on AES-based signatures, we took the data points from Bhadauria et al. [9, Table 2], which was obtained on a more powerful machine (16-core 32-thread 3.2GHz CPU, 512GB 1.6GHz DDR3 RAM), since their source code is not available. Compared with their data, our construction with $n = 16$ signs 6.6 – 17× faster, verifies 5 – 10× faster, and produces signatures 2 – 4.15× smaller. Our protocols also perform very competitively to Limbo's signatures, even though the standard AES verification circuit we used has 3x multiplication gates than theirs. Limbo's schemes offers 1-bit less computational security because their key space has to shrink to avoid 0-byte input to

Table 5: Security Parameters for Ring Identification

$\log m$	4	7	10	13	16
$\log \lvert\mathbb{F}\rvert$	16	16	16	32	32
τ ($n = 16$)	11	11	11	11	11

Table 6: Security Parameters for Ring Signature ($\mathbb{F}_{2^{16}}, n = 16$)

$\log m$	4	6	7	8	10	12	13
τ	58	58	59	59	61	63	64

SubByte.

5.2 Ring Identification and Ring Signatures

Ring identification is an interactive protocol that allows one to prove to another that its identity belongs to a predefined set of identities but without leaking its identity beyond the membership. Similarly, ring signatures, which are enhanced signatures, allow any member of a group to independently sign messages on behalf of the group without revealing its identity. The notion of *unforgeability* and *anonymity* can be formalized for these schemes [24].

Katz et al. [24] first considered combining ZKP and one-way functions to build ring identification and ring signatures. For performance reasons, however, their construction relied on the non-standard cipher LowMC [1]: both as the one-way function for signatures and in the Davies-Meyer construction of collision-resistant compression function needed in the Merkle tree of public keys.

Ring Identification. For ring identification over a set of m users, we combined the ZKPs of membership (Section 4.2) and individual AES (Section 4.1) to efficiently prove

$$AES_{sk}(0) = ID \quad \wedge \quad ID \in \{ID_i\}_{i\in[m]}.$$

where sk and ID are the secret witness whereas $\{ID_i\}_{i\in[m]}$ are plaintext. Note that the combined circuit will guarantee that the same ID is used in both the proofs of AES and membership. Note that an attack succeeds if either the ZKP of AES or that of membership is subverted, thus,

$$\mathcal{E}(\text{Ident}_{ring}) = 1 - [1 - \mathcal{E}(\text{AES})] \cdot [1 - \mathcal{E}(\text{MBR}_m)]$$

Then, $\mathcal{E}_{\text{Main}}$for both schemes can be derived using Equation (8). Concrete parameters for our identification scheme are given in Table 5. Note we switched $\mathbb{F}$ from $\mathbb{F}_{2^{16}}$ (for sets whose $m \leq 1024$) to $\mathbb{F}_{2^{32}}$ (for sets whose $m > 1024$) to keep τ reasonably small, because when m gets close to $|\mathbb{F}|$, $m/|\mathbb{F}|$, the soundness error due to element projection, will be too close to 1. Figure 7 shows the performance of our ring identification protocol.

Ring Signatures. Our ring signature scheme used the same circuit design as that of our ring identification above, except that we did not use the GKR transform but relied on FLPCP to process all non-linear polynomials in the private membership and AES circuits. This change allows the identification scheme to finish in just 4 rounds of challenges, thus resulting concretely better performance when transformed into non-interactive signatures. Further, we used degree-4 polynomials to more efficiently verify the membership circuit.

Table 7: τ for Counter-Mode AES (interactive, $\mathbb{F}_{2^{16}}, n = 16, s = 40$)

Number of Blocks	1	4	16	64	256	1024
τ (no-extra-witness)	12	12	12	13	13	13
τ (extra-witness)	11	11	11	11	11	11

To analyze the soundness of our signature scheme, we observe that the first round involves break ID into 8 16-bit elements (we used $\mathbb{F}_{2^{16}}$) and verify Equation (7) through a random linear combination of the 8 elements. The soundness error introduced by this random linear combination round is $m/|\mathbb{F}|$ since Equation (7) has degree m. The next two rounds of challenges are due to FLPCP of all polynomial gates in the circuit. The membership (Equation (7)) can be verified by a quadtree of $1+4+16+\cdots+m/4 = (m-1)/3$ 4-variate degree-4 polynomials; while the single AES block has 400 degree-3 polynomials. Treating all polynomials as degree-4 ones, the two rounds of the batched variant of FLPCP (Appendix B) have soundness errors $1/|\mathbb{F}|$ and $4\sqrt{M}/(|\mathbb{F}| - \sqrt{M})$, respectively, where $M = \frac{m-1}{3} + 400$. The final round of challenge is due to checking all-but-one of the n MPC-in-the-Head parties, hence soundness error $1/n$. We followed the idea of Banquet [5] to calculate the number of iterations needed to achieve $k = 128$-bit computational security. The concrete parameters are shown in Table 6.

Figure 8 compares our ring signature scheme with that of Katz et al [24]. On ring sizes ranging from 128 to 8192, our ring signature sizes are 1.18–7.3× smaller. Regarding the sign/verify times, Dubhe is more than 80× faster for 128 or smaller rings and can outperform theirs when the ring has ≤ 8192 users.

5.3 ZKP for Many Blocks of AES

Efficient ZK Argument of Knowledge for many AES blocks can be useful in building two cryptographic primitives: (1) Given a symmetric-key encryption of a multi-block message $\{M_i\}_{i\in[\ell]}$, it allows to prove that $\{M_i\}_{i\in[\ell]}$ satisfy a property $p(\cdot)$ without leaking any extra information about the message; (2) To construct a pseudorandom generator whose honest execution can be efficiently verified without leaking the PRG seed.

We have two protocols for proving multiple blocks of AES. The first one consists of 41 circuit layers but does not use extra witness, as was described in Section 4.1. The second one uses 200 witness bytes per block but the GKR transform is applied to each circuit layer of $400m$ polyno-

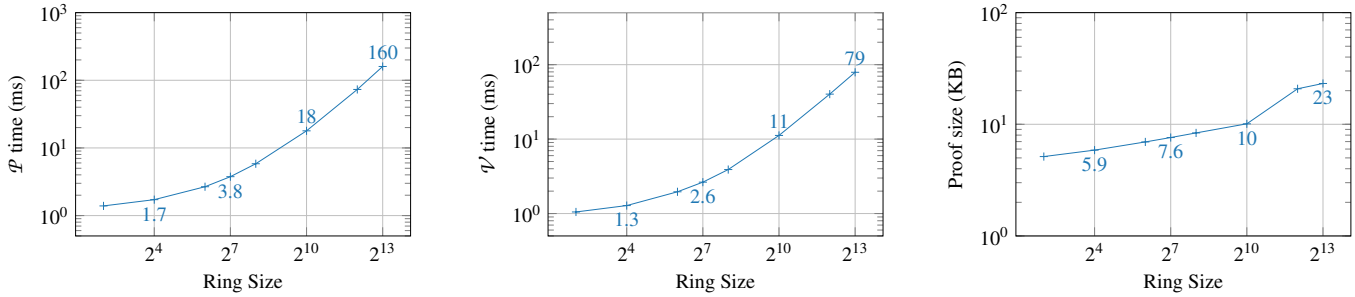

Figure 7: Ring Identification ($s = 40$)

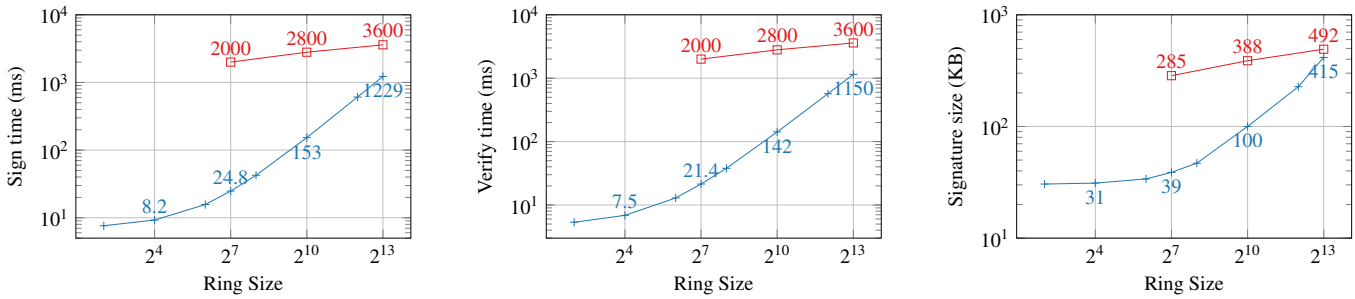

Figure 8: Ring Signature ($k = 128$) —+— Dubhe (AES), —□— KKW (LowMC)

mial gates *across* the block boundaries.

Security parameters for our no-extra-witnesses protocol was analyzed in Section 4.1. For the witness-based mAES protocol, checking $400m$ G gates introduces soundness error $\mathcal{E}_{\text{GKR}} = 1 - \left(1 - \frac{9+\log m}{|\mathbb{F}|}\right)\left(1 - \frac{4}{|\mathbb{F}|}\right)^{8+\log m}$. A degree-4 polynomial will be checked by FLPCP, thus $\mathcal{E}_{\text{FLPCP}} = \frac{4}{|\mathbb{F}|-1}$. We used $n = 16$ and $\mathbb{F}_{2^{16}}$, with τ varying with the number of blocks to achieve $s = 40$ (Table 7).

Figure 9 shows the performance of the two approaches on varying number of AES-128 blocks. Both approaches have a linear prover but the extra-witness-based prover runs 110–160× faster. The protocol with extra-witness also has faster verification if $m \leq 64$, and use less communication when $m < 16$. But for larger m, the no-extra-witness protocol offers faster verification and less communication. E.g., to prove 1024 blocks of AES, the extra-witness based protocol will use 44× more communication than the other.

We also benchmarked Virgo/Virgo++, Limbo, and QuickSilver on counter-mode AES. We set the Mersenne prime $p = 2^{31} - 1$ in Virgo/Virgo++ for conjectured 40-bit statistical security. Virgo++'s verifier used linear time in circuit size because it spent linear time to compute GKR's predicate function. Virgo was not able to handle more than 2^8 AES blocks before memory ran out. We compared QuickSilver and our protocol in Figure 10. QuickSilver exhibited almost constant costs when the number of AES blocks is ≤ 1024, because the linear part of its cost is shadowed by a relatively big constant part spent on generating the authenticated triples and values. The security of LPN assumption, which QuickSilver relies on, requires those triples and values be generated in sufficiently large batches even if the actual circuit is small. However, when there are more than 1024 AES blocks, the linear part of their cost regains dominance in the overall cost and we observe the same growth rate as our extra-witness based protocol.

Table 8: SHA256 (interactive, $s = 40$)

Protocol	$T_{\mathcal{P}}$ (ms)	$T_{\mathcal{V}}$(ms)	$\lvert\pi\rvert$ (KB)
QuickSilver	358	355	1644
Virgo ($\mathbb{F}_{(2^{61}-1)^2}$)	1158	12	154
Virgo++ ($\mathbb{F}_{(2^{61}-1)^2}$)	255	37	123
Limbo ($n = 16, \tau = 11$)	65	48	41
Dubhe ($n = 16, \tau = 11$)	62	47	35
Limbo ($n = 64, \tau = 7$)	106	98	25
Dubhe ($n = 64, \tau = 7$)	92	89	22

5.4 Classic Benchmarks

We also tested our approach on two frequently-used benchmark applications, SHA256 (Table 8) and matrix multiplication (Table 9). For SHA256, Dubhe consistently outperforms Limbo when configured with the same n. Comparing to Virgo/Virgo++, Dubhe generally uses less communication and prover time whereas Virgo/Virgo++ allow much faster verification. For matrix multiplication, our protocol can outperform all tested prior protocols in prover time and verifier time. Note that our protocol can be made non-interactive and

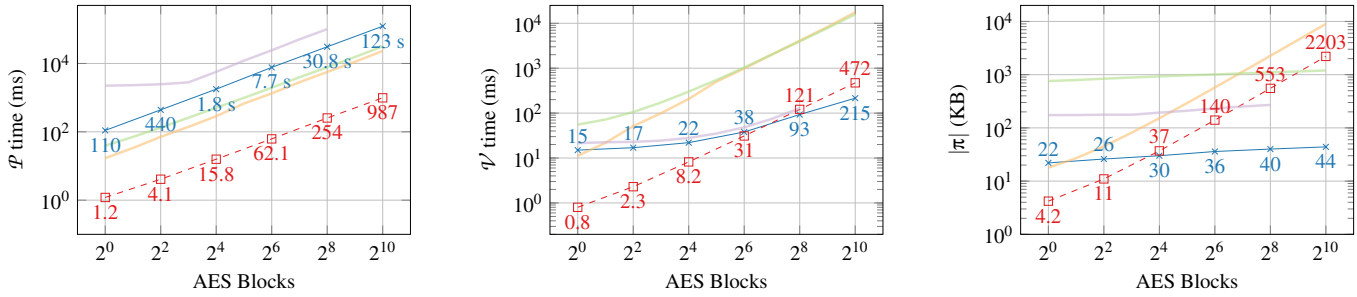

Figure 9: Counter-Mode AES (interactive, $s = 40$)

Dubhe (extra-witness, $n = 16$), Dubhe (no-extra-witness, $n = 16$), Limbo, Virgo++, Virgo

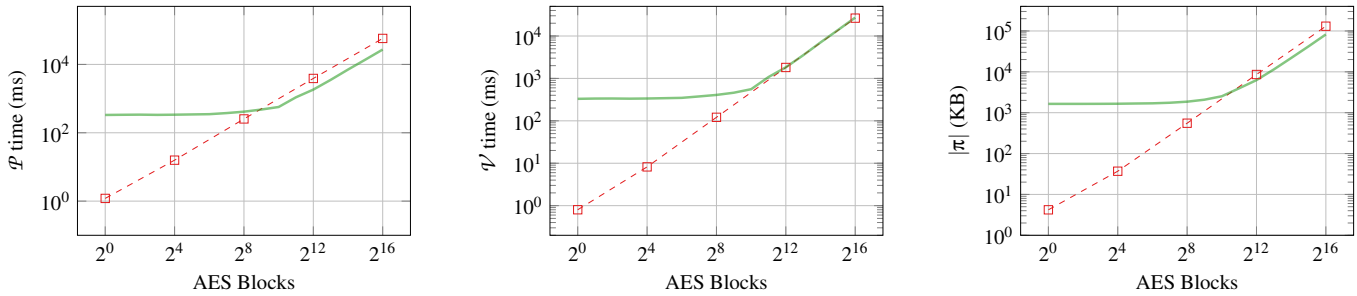

Figure 10: Compare QuickSilver with Dubhe on Counter-Mode AES (interactive, $s = 40$)

Dubhe (extra-witness, $n = 16$), QuickSilver

Table 9: Matrix Multiplication (interactive, $s = 40$)

Matrix Size	Appl. Field	Protocol	$T_{\mathcal{P}}$ (s)	$T_{\mathcal{V}}$ (s)	$\lvert\pi\rvert$ (MB)
256×256	$\mathbb{Z}_{2^{61}-1}$	Virgo++ ($\mathbb{F}_{(2^{61}-1)^2}$)	26	9.6	0.17
		Limbo ($n = 16, \tau = 11$)	37	29	22
		Dubhe ($n = 16, \tau = 10$)	0.29	0.22	10
	$\mathbb{F}_2$	Virgo++ ($\mathbb{F}_{(2^{61}-1)^2}$)	27	10	0.17
		Limbo ($n = 16, \tau = 11$)	44	33	0.28
		Dubhe ($n = 16, \tau = 10$)	0.33	0.24	0.41
2048×2048	$\mathbb{Z}_{2^{61}-1}$	QuickSilver (Circuit)	506	496	67821
		QuickSilver (Polynomial)	26	11	71
		Dubhe ($n = 16, \tau = 10$)	18	13	640
	$\mathbb{F}_2$	QuickSilver (Polynomial)	30	26	2.6
		Dubhe ($n = 16, \tau = 10$)	21	15	26

Communication costs exclude the transmission of output matrices.

publicly verifiable whereas QuickSilver couldn't. Virgo++ couldn't handle 2048×2048 matrix multiplication on our machine due to their extremely high memory demand, but exhibits clear advantage in communication efficiency at the scales that it can handle.

6 Conclusion

We combined specialized variants of GKR, FLPCP, and MPC-in-the-Head to build new succinct zero-knowledge argument of knowledge protocols that enable linear prover, logarithmic verifier, using logarithmic communication. This asymptotically attractive approach actually yields concretely efficient proofs that can outperform some best existing protocols in various applications.

Acknowledgements

We thank Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang for answering questions on Virgo and Virgo++. We thank Titouan Tanguy for answering questions on Limbo. We thank Xiao Wang for answering questions on QuickSilver.

References

[1] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. In *EUROCRYPT*, 2015.

[2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM CCS*, 2017.

[3] Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed Σ-protocol theory for lattices. In *CRYPTO*, 2021.

[4] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and

their application to lattice-based cryptography. In *PKC*, 2020.

[5] Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from aes. In *PKC*, 2021.

[6] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *ICALP*, 2018.

[7] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *TCC*, 2016.

[8] Daniel Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In *EUROCRYPT*, 2015.

[9] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear iop. In *ACM CCS*, 2020.

[10] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC*, 2013.

[11] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *CRYPTO*, 2012.

[12] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *CRYPTO*, 2019.

[13] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, 2011.

[14] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *ACM CCS*, 2017.

[15] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS*, 2017.

[16] Alessandro Chiesa, Michael A Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. *arXiv preprint arXiv:1704.02086*, 2017.

[17] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of Innovations in Theoretical Computer Science Conference*, 2012.

[18] Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge mpcith-based arguments. In *ACM CCS*, 2021.

[19] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[20] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster Zero-Knowledge for boolean circuits. In *USENIX Security*, 2016.

[21] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, 2008.

[22] Shafi Goldwasser, Guy N. Rothblum, and Yael Tauman Kalai. Delegating computation: Interactive proofs for muggles. *Electronic Colloquium on Computational Complexity*, 2017.

[23] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, 2007.

[24] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS*, 2018.

[25] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

[26] NIST. Federal information processing standards publication FIPS-197, advanced encryption standard, 1999.

[27] James Plank, Kevin Greenan, and Ethan Miller. Screaming fast Galois field arithmetic using Intel SIMD instructions. In *USENIX FAST*, 2013.

[28] Cyprien Delpech de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P Smart. Bbq: Using aes in picnic signatures. In *International Conference on Selected Areas in Cryptography*, 2019.

[29] Nigel Smart. 'Bristol Fashion' Circuits. https://homes.esat.kuleuven.be/~nsmart/MPC/, Accessed in April, 2022.

[30] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO*, 2013.

[31] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *IEEE Symposium on Security and Privacy*, 2018.

[32] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO*, 2019.

[33] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *ACM CCS*, 2021.

[34] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In *ACM CCS*, 2021.

[35] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *IEEE Symposium on Security and Privacy*, 2020.

A Definitions

We model $\mathcal{P}$ and $\mathcal{V}$ as two interactive Turing machines. We use $\langle \mathcal{P}, \mathcal{V} \rangle(\lambda, x)$ to denote the *transcript*, i.e., concatenation of all communications, of running $\mathcal{P}$ and $\mathcal{V}$ over public input x using public security parameter λ; while $(\mathcal{P}, \mathcal{V})(\lambda, x)$ denotes $\mathcal{V}$'s *acceptance bit* after interacting with $\mathcal{P}$ on input x and parameter λ. We say a predicate R defines an NP relation L if $x \in L \Leftrightarrow \exists w, R(w,x) = 1$.

A.1 Zero-Knowledge Proof of Knowledge

$(\mathcal{P}, \mathcal{V})$ is a zero-knowledge proof of knowledge for R if the following hold:

- ***Completeness.*** $\forall w, x$, $R(w,x) = 1 \Rightarrow (\mathcal{P}(w), \mathcal{V})(\lambda, x) = 1$.
- ***Soundness.*** For any $\mathcal{P}$', there exists an efficient extractor ε with black-box access to $\mathcal{P}'$ such that for all x, $\Pr[(\mathcal{P}'(w), \mathcal{V})(\lambda, x) = 1 \wedge R(w,x) \neq 1 \mid w \leftarrow \varepsilon^{\mathcal{P}'}(\lambda, x)]$ is negligible in λ.
- ***Zero-knowledge.*** There exists an efficient simulator $\mathcal{S}$ such that for all efficient $\mathcal{V}''$,

$$R(w,x) = 1 \Rightarrow \langle \mathcal{P}(w), \mathcal{V}'' \rangle(\lambda, x) \approx \mathcal{S}^{\mathcal{V}''}(x).$$

We call such a system zero-knowledge *argument* if the above soundness holds only for computational efficient $\mathcal{P}'$.

A.2 Sigma Protocols

$(\mathcal{P}, \mathcal{V})$ is a Σ-protocol for an NP relation defined by R if the following hold:

- ***3-move transcripts.*** For all x, the transcript of $\langle \mathcal{P}, \mathcal{V} \rangle(x)$ has a 3-move pattern (a, e, z) where a, z are from $\mathcal{P}$ to $\mathcal{V}$ and e is from $\mathcal{V}$ to $\mathcal{P}$. The messages a, e, z are commonly called *commit*, *challenge*, and *prove* messages, respectively.
- ***Completeness.*** $\forall w, x$, if $R(w,x) = 1$, then $(\mathcal{P}(w), \mathcal{V})(x) = 1$.
- ***n-Special Soundness.*** For any x, a witness w for $x \in L$ can be efficiently computed from any n accepting transcripts (a, e_i, z_i) (where $i \in [n]$) with distinct challenge e_i. (This implies that if a challenge e is uniformly picked from a set of size s, then the soundness error is at most $(n-1)/s$.)
- ***Special honest-verifier ZK.*** There exists an efficient simulator S such that $\langle \mathcal{P}, \mathcal{V} \rangle(x)$ is indistinguishable from $S(x, e)$ where e is the message from the honest-$\mathcal{V}$ to $\mathcal{P}$.

Σ-protocols can be efficiently transformed to full ZK and ZK Proof of Knowledge (ZKPoK) protocols. It is also easy to to boost the security of Σ-protocols through parallel composition. The Fiat-Shamir transform can be used to turn Σ-protocols into efficient *Non-Interactive* ZKP (NIZKP) in the random oracle model. Thanks to these properties, Σ-protocols is commonly the first step in building many cryptographic primitives such as identification schemes and digital signatures.

A.3 Interactive Oracle Proofs

$(\mathcal{P}, \mathcal{V})$ is a k-round public-coin IOP for an NP relation defined by R if the following hold:

- ***μ-round interactive protocol.*** After $\mathcal{P}$ creates an initial a_0, each time $\mathcal{V}$ sends a uniform random message e_i independent of $\mathcal{P}$'s messages, $\mathcal{P}$ replies with a_i, $\forall i \in \{1, 2, \ldots, \mu\}$. $\mathcal{V}$ has oracle access to $\{a_i : i \in \{1, 2, \ldots, \mu\}\}$ and outputs 1 bit.
- ***Completeness.*** $\forall w, x$, if $R(w,x) = 1$, then $(\mathcal{P}(w), \mathcal{V})(\lambda, x) = 1$.
- ***Soundness.*** $\forall x$, if $R(w,x) \neq 1$, $\Pr[(\mathcal{P}'(w), \mathcal{V})(\lambda, x) = 1]$ is negligible in λ for any $\mathcal{P}'$.

The definition can also be extended with the following properties:

- ***Proof of Knowledge.*** There exists an efficient extractor ε such that for all efficient $\mathcal{P}'$, $\Pr[(\mathcal{P}', \mathcal{V})(\lambda, x) = 1 \wedge R(w,x) \neq 1 \mid w \leftarrow \varepsilon^{\mathcal{P}'}(\lambda, x)]$ is negligible in λ.

- ***Honest-verifier ZK.*** There exists an efficient simulator $\mathcal{S}$ such that for all efficient honest $\mathcal{V}'$,

$$R(w,x) = 1 \Rightarrow \langle \mathcal{P}(w), \mathcal{V}' \rangle (\lambda, x) \approx \mathcal{S}^{\mathcal{V}'}(x).$$

A.4 Tree of Transcripts

A $(k_1, \ldots, k_\mu)$-tree of transcripts for a μ-round public-coin interactive oracle protocol, is a set of $\Pi_{i=1}^{\mu} k_i$ transcripts organized in a tree structure:

- Each edge in the tree corresponds to a challenge message from $\mathcal{V}$.
- Each node at depth i in the tree corresponds to a response message from $\mathcal{P}$ to answer $\mathcal{V}$'s i^{th} challenge. This node has exactly k_i children, corresponding to k_i distinct challenges from $\mathcal{V}$.
- Each transcript consists of one path of messages from the root to a leaf node.

A.5 $(k_1, \ldots, k_\mu)$-Special Soundness

A μ-round public-coin protocol is $(k_1, \ldots, k_\mu)$-special sound if there exists an efficient algorithm that on input a $(k_1, \ldots, k_\mu)$-tree of accepting transcripts outputs a witness w for $R(w,x) = 1$.

B FLPCP for $m \cdot n$ Polynomial Equations

Let $G : \mathbb{F}^\ell \to \mathbb{F}$ be an ℓ-variable degree-d polynomial over $\mathbb{F}$. Let $w_{i,j}$ where $i \in [\ell], j \in [mn]$ be ℓmn (possibly secret) values, and

$$G(w_{0,j}, \ldots, w_{\ell-1,j}) = 0, \forall j \in [mn]$$

be $m \cdot n$ equations to be proved.

$\forall i \in [\ell], j \in [m]$, let $p_{i,j}$ be degree-n polynomials such that

$$p_{i,j}(k) = w_{i,j\cdot n+k}, \forall k \in [n]$$
$$p_{i,j}(n) \leftarrow \mathbb{F}$$

Define $p_{\ell,j}(x) \stackrel{\text{def}}{=} G\left(p_{0,j}(x), \ldots, p_{\ell-1,j}(x)\right), \forall j \in [m]$. Then $p_{\ell,j}(x)$ is a polynomial of degree at most $n \cdot d$.

B.1 FLPCP with Linear Bandwidth

The linear bandwidth FLPCP protocol works as follows:

(1) $\forall i \in [\ell], j \in [m], k \in [mn]$, $\mathcal{P}$ sends $w_{i,k}$, $p_{i,j}(n)$ to $\mathcal{V}$.
$\forall h \in [n \cdot d + 1] \backslash [n], j \in [m]$, $\mathcal{P}$ computes $q_{h,j} := p_{\ell,j}(h)$ and sends $q_{h,j}$ to $\mathcal{V}$. $\mathcal{V}$ sets $q_{h,j} := 0$ for all $j \in [m], h \in [n]$.

(2) $\mathcal{V}$ picks and sends $r \leftarrow \mathbb{F} \backslash \{0, 1, \ldots, n-1\}$ to $\mathcal{P}$.

(3) $\mathcal{P}$ computes and sends $p_{i,j}(r)$ for all $i \in [\ell+1], j \in [m]$. Then, $\forall i \in [\ell], j \in [m]$, $\mathcal{V}$ interpolates a value, which we call $p'_{i,j}(r)$, from points

$$(0, w_{i,jn}), (1, w_{i,jn+1}), \ldots, (n-1, w_{i,jn+(n-1)}), (n, p_{i,j}(n)).$$

Finally, for every $j \in [m]$, $\mathcal{V}$ interpolates another value that we call $p'_{\ell,j}(r)$, from points $(0, q_{0,j}), (1, q_{1,j}), \ldots, (n \cdot d, q_{n\cdot d,j})$ and verifies that both of the following hold for all $j \in [m]$:

$$p'_{i,j}(r) = p_{i,j}(r), \ \forall i \in [\ell];$$
$$p'_{\ell,j}(r) = G\left(p_{0,j}(r), \ldots, p_{\ell-1,j}(r)\right).$$

This protocol only has 3 rounds and the bandwidth complexity is $O(mnd + \ell m)$. Its soundness error is $d \cdot n / (|\mathbb{F}| - n)$.

By checking a random linear combination of $p_{\ell,j}(r)$, one can reduce the bandwidth cost to $O(nd + \ell m)$, as we will describe next.

B.2 FLPCP with Sub-linear Bandwidth

The sub-linear bandwidth FLPCP protocol works as follows:

(1) $\forall i \in [\ell], j \in [m], k \in [mn]$, $\mathcal{P}$ sends $w_{i,k}$, $p_{i,j}(n)$ to $\mathcal{V}$.

(2) $\mathcal{V}$ sends random coefficients $c_j \in \mathbb{F}$ where $j \in [m]$.

(3) $\forall h \in [nd+1] \backslash [n]$, $\mathcal{P}$ computes $q_h := \sum_{j \in [m]} c_j \cdot p_{\ell,j}(h)$ and sends q_h to $\mathcal{V}$. $\mathcal{V}$ sets $q_h := 0$ for all $h \in [n]$.

(4) $\mathcal{V}$ picks and sends $r \leftarrow \mathbb{F} \backslash \{0, 1, \ldots, n-1\}$ to $\mathcal{P}$.

(5) $\mathcal{P}$ computes and sends $p_{i,j}(r)$ for all $i \in [\ell+1], j \in [m]$. Then, $\forall i \in [\ell], j \in [m]$, $\mathcal{V}$ interpolates a value, which we call $p'_{i,j}(r)$, from points

$$(0, w_{i,jn}), (1, w_{i,jn+1}), \ldots, (n, p_{i,jn+n}(n)).$$

For every $j \in [m]$, $\mathcal{V}$ interpolates another value that we call $p'_\ell(r)$, from points $(0, q_0), (1, q_1), \ldots, (nd, q_{nd})$ and verifies that both of the following hold for all $j \in [m]$:

$$p'_{i,j}(r) = p_{i,j}(r), \ \forall i \in [\ell];$$
$$p'_\ell(r) = \sum_{j \in [m]} c_j \cdot G\left(p_{0,j}(r), \ldots, p_{\ell-1,j}(r)\right).$$

Therefore, the verifier will send two challenges throughout the protocol. The soundness error due to the first challenge is $1/|\mathbb{F}|$, and that of the second challenge is $\frac{nd}{|\mathbb{F}|-n}$. The bandwidth cost, excluding the ZK transmission of $w_{i,j}$, is $O(nd + \ell m))$.

C Proof of Theorem 3.1

Proof. Recall that our protocol is a public-coin, interactive oracle proof system. It starts with a random first message from $\mathcal{P}$ to $\mathcal{V}$, which is followed by μ_{GKR} challenge-response rounds according to the GKR protocol, then one challenge-response round of FLPCP, and finally one challenge-response round of KKW-LO. Each challenge-response round consists of one uniform random challenge message from $\mathcal{V}$ and one response message from $\mathcal{P}$.

Completeness. Thanks to the completeness of GKR, FLPCP, and KKW-LO, it is easy to verify that an honest $\mathcal{V}$ will always output 1 when interacting with an honest $\mathcal{P}$ who knows the witness w.

Soundness. We will start with one iteration of our protocol and show that its soundness error is $1-(1-\mathcal{E}_{\text{GKR}})\cdot(1-\mathcal{E}_{\text{FLPCP}})\cdot(1-\mathcal{E}_{\text{KKW}})$. In each of the $(\mu_{GKR}+2)$ rounds, we note that a malicious prover who doesn't know the witness has a fixed probability to guess the round challenge, and knows immediately whether the guess is correct or not when the challenge is received. The malicious prover only needs to guess correctly a single challenge in order to corrupt the whole $(\mu_{GKR}+2)$ rounds of interaction.

Let d be the degree of the polynomial gates in the circuit. In each of the first μ_{GKR} rounds, the soundness of GKR guarantees that the malicious prover can correctly guess the challenge with probability at most $d/|\mathbb{F}|$. In the FLPCP round, the soundness of FLPCP guarantees that the malicious prover can only pass the challenge with probability $d/(|\mathbb{F}|-1)$. In the final round of KKW-LO, soundness of KKW-LO guarantees that the malicious prover can only pass the uniform random challenge with probability $1/n$ where n is the number of parties used in KKW-LO. Thus, with probability $(1-d/|\mathbb{F}|)^{\mu_{GKR}}\cdot(1-d/(|\mathbb{F}|-1))\cdot(1-1/n)$, a malicious prover has to fail in answering all $(\mu_{GKR}+2)$ rounds of challenges. Therefore, its soundness error is $1-(1-d/|\mathbb{F}|)^{\mu_{GKR}}\cdot(1-d/(|\mathbb{F}|-1))\cdot(1-1/n)$. Combining with the facts that $\mu_{GKR}=\sum_{i=1}^{d}2s_i$, $\mathcal{E}_{\text{GKR}}=1-\Pi_{i=1}^{d}(1-d/|\mathbb{F}|)^{s_i}$, $\mathcal{E}_{\text{FLPCP}}=1-d/(|\mathbb{F}|-1)$, and $\mathcal{E}_{\text{KKW}}=1-1/n$, it is easy to see that soundness error of one iteration of the combined protocol can also be written as

$$\mathcal{E}_{\text{Main}}\stackrel{\text{def}}{=}1-(1-\mathcal{E}_{\text{GKR}})\cdot(1-\mathcal{E}_{\text{FLPCP}})\cdot(1-\mathcal{E}_{\text{KKW}}). \quad (8)$$

Our protocol offers knowledge soundness, because it is $(\underbrace{d+1,\ldots,d+1}_{\mu_{GKR}},d+1,2)$-special sound:

- For every node corresponding to the state of a sum-check round in the GKR phase, $d+1$ distinct valid challenge-response pairs allow to efficiently recover all secret coefficients of the degree-d polynomial in the sum-check round;
- For every node corresponding to the state of the FLPCP round, $d+1$ distinct valid challenge-response pairs allow to recover all secret inputs to the FLPCP protocol, which are encoded by a degree-d polynomial;
- For every node corresponding to the state of the KKW-LO round, 2 distinct valid challenge-response pairs allow to recover all secret inputs to the KKW-LO circuit.

Following a proved property of $(k_1,\ldots,k_\mu)$-special sound IOP [3, Theorem 1], let N_i be the size of the sampling space of the i^{th} round, the knowledge error of our protocol is

$$\begin{aligned}&\frac{\Pi_{i=1}^{\mu_{GKR}+2}N_i-\Pi_{i=1}^{\mu_{GKR}+1}(N_i-d)(N_{\mu_{GKR}+2}-1)}{\Pi_{i=1}^{\mu_{GKR}+2}N_i}\\ =&1-\Pi_{i=1}^{\mu_{GKR}+1}(1-d/N_i)(1-1/N_{\mu_{GKR}+2})\end{aligned}$$

where $N_i=|\mathbb{F}|$ for $i\in\{1,\ldots,\mu_{GKR}\}$, $N_{\mu_{GKR}+1}=|\mathbb{F}|-1$, and $N_{\mu_{GKR}+2}=n$. This knowledge error is precisely $\mathcal{E}_{\text{Main}}$ that was shown above.

Finally, τ iterations of the protocol allow to bound the soundness error to $\mathcal{E}_{\text{Main}}^{\tau}$.

Honest-Verifier Zero-Knowledge. The HVZK property of our protocol stems from that of KKW-LO. An honest-verifier $\mathcal{V}'$ follows the protocol and always derives its challenge message from the random oracle H. So we can construct the simulator $\mathcal{S}$ as below:

(1) $\mathcal{S}$ runs our protocol with the verifier invoked as a subroutine. Since $\mathcal{S}$ does not know the secret witness, it randomly selects one of the n parties and corrupts that party's internal state so that the interaction convinces a verifier who never sees the internal state of the corrupted party.

(2) In the final KKW-LO round, if the verifier indeed chooses not to check the corrupted party, $\mathcal{S}$ saves the transcript; otherwise, $\mathcal{S}$ discard the transcript and restart from Step (1).

Because $\mathcal{S}$ used the same steps as specified in the protocol to generate the transcript except the internal state of the corrupted party (which is not included in the transcript), the generated transcript is identically distributed as that between an honest prover and an honest verifier. In addition, it takes $\mathcal{S}$ expected time $O(n)$ to terminate with a transcript. Hence by repeating it τ times, an identically distributed full transcript of $\langle\mathcal{P}(w),\mathcal{V}'\rangle(\lambda,x)$ can be efficiently generated.

□