Efficient Zero Knowledge for Regular Language

Michael Raymond[1][000−0002−6838−3453], Gillian Evers[1][0009−0008−9746−2258], Jan Ponti[1][0000−0002−2221−6828], Diya Krishnan[2][0000−0002−8879−1411], Xiang Fu[1][0000−0002−6608−1654]

[1] Hofstra University, `Xiang.Fu@hofstra.edu`
[2] Ramanujan Homeschool Academy

Abstract. A succinct zero knowledge proof for regular language membership, i.e., to prove a secret string behind an encryption (hash) belongs to a regular language is useful, e.g., for asserting that an encrypted email is free of malware. The great challenge in practice is that the regular language used is often huge. We present **`zkreg`**, a distributed commit-and-prove system that handles such complexity. In **`zkreg`**, cryptographic operations are encoded using arithmetic circuits, and input acceptance is modeled as a zero knowledge subset problem using Σ-protocols. We introduce a Feedback Commit-and-Prove (FB-CP) scheme, which connects Σ-protocols and the Groth16 system with $O(1)$ proof size and verifier cost. We present a close-to-optimal univariate instantiation of zk-VPD, a zero knowledge variation of the KZG polynomial commitment scheme, based on which an efficient zk-subset protocol is developed. We develop a 2-phase proof scheme to further exploit the locality of Aho-Corasick automata. To demonstrate the performance and scalability of **`zkreg`**, we prove that all ELF files (encrypted and hashed) in a Linux CentOS 7 are malware free. Applying inner pairing product argument, we obtain an aggregated proof of 1.96 MB which can be verified in 6.5 seconds.

Keywords: zero knowledge proof, zkSNARK, Aho-Corasick Automata, commit-and-prove, commitment schemes

1 Introduction

Consider how email exchange servers are secured for user safety. A common practice is to run anti-virus software on all incoming emails before they are delivered, however, at the cost of user privacy. Instead, a sender can encrypt her email and provide a succinct zero knowledge (zk)-proof certifying its freedom of malware. Given that most prevalent malware scanners use regular languages for signatures, in this paper, we call it the *zero knowledge regular language membership* (zk-Reg) problem. The prover knows a secret string s and a secret key k. The verifier knows a public finite state machine $\mathbf{A}$ which captures all allowed/benign strings, and is given $\mathsf{hash}(\mathsf{encrypt}(s, k))$. The prover hopes to convince the verifier with a succinct proof π that $s \in L(\mathbf{A})$. In the context of networking and secure communication, zk-Reg has many applications, e.g., to prove that an encrypted DNS request does not contain any forbidden site, and to show an encrypted packet has already passed firewall rules. The non-zk version of the problem can be applied in software distribution where s does not have to be a secret.

It is well known that all NP statements have a zero knowledge proof [40]. There is also a long line of zk-proof research on more expressive machine models such as von Neumann and RAM machines [14, 35]. Here, the great challenge in practice is that the size of $\mathbf{A}$ is out of reach of most (zk)-SNARK provers. Take ClamAV as one example, ignoring all other signature formats, its standard hexadecimal signature database alone generates an Aho-Corasick automaton (AC-DFA) with 19 million states and over 300 million transitions. To directly encode ClamAV's AC-DFA would reach the limit of distributed provers such as DIZK [66]. Thus to prove even a small email message would be prohibitively expensive.

1.1 Technical Overview

Given the problem size, our overall approach is the Commit-and-Prove scheme [28, 27, 30, 2, 6], which combines the benefits of zk-SNARK and Σ-protocols. There is a trusted set-up who generates prover/verifier keys, a public set-up that publishes the AC-DFA $\mathbf{A}$. The prover owns a secret key k and an input string s. The verifier is provided with $\mathsf{hash}(\mathsf{encrypt}(s, k))$. The goal of the proof is to convince the verifier that $s \in L(\mathbf{A})$.

We plan to exploit the locality of AC-DFA. Let *depth* of a state be its shortest distance from the initial state. We notice that for all of the 2479 ELF (object and executable) files in a Linux CentOs 7, after excluding a small frequently visited set (5.5% of all states), over 34.49% (79.95%) have depth no greater than 10 (20). To exploit the locality, we first arithmetize $\mathbf{A}$ by encoding its states and transitions as elements of a large prime field. Let T be the arithmetization of $\mathbf{A}$. Let $\{T_1, ..., T_n\}$ be the subsets of T with T_i containing the elements bounded by depth i. All of these subsets are made public. The prover runs s over $\mathbf{A}$ and let $\mathbf{S}$ be the set of states/transitions along the acceptance path, and m be its max depth. Then our proof scheme consists of essentially two subset proof: (1) $\mathbf{S} \subseteq T_m$ and (2) $T_m \subseteq T$.

Apparently, $\mathbf{S}$ and T_m need to be represented succinctly and in zero knowledge. In [63], a zk-batch accumulator proof is provided by extending bilinear accumulator [58] using Σ-protocol of product proof to "cancel" the additional terms caused by blinding factors. We adapt it for zk-subset proof in our context, and provide a more efficient protocol in Section 3.2 by cutting its product proof.

The second component of `zkreg` is a modified Groth16 system (adapting the ccSnark scheme in [28]). It is needed because arithmetic circuit is more convenient than Σ-protocols in encoding encryption and hash operations. In addition, we need to encode AC-DFA, which has more sophisticated machinery than a standard FSA, for its "fail-edge" semantics. In particular, we need to handle a *support-set* (root-set) problem. Recall that the "allowed" transitions/states are published and encoded in bilinear accumulator based zk-set. We need to extract the standard set (i.e., no duplicates) from the multi-set of states and transitions appearing on the acceptance path, before engaging them in the zk-subset proof. This is solved by taking advantage of a well known result for large prime field that: for a multi-set A, the characteristic polynomial of its support-set is $p_A / \gcd(p_A, p'_A)$ where p'_A is the formal derivative of p_A.

The two proof systems have to be connected. We present a Feedback Commit-and-Prove (FB-CP) scheme. To prevent the a malicious prover fake a polynomial witness to the circuit, the system "enforces" her to commit *before* she sees the random challenge by verifier. It takes an arithmetic circuit, whose secret witness input wires are divided into k segments. The prover first computes the partial proof of the first segment. It is used as a *commitment* to the inputs, so that the Fiat-Shamir heuristics can be applied and some public input wires are re-computed as the hash of the commitment. Then the rest of witness inputs, intermediate and output wires are computed, and the complete Groth16 proof is generated. We show that FB-CP achieves $O(1)$ proof size and verifier cost.

The FB-CP scheme still lacks zero knowledge, as opening KZG commitment leaks one evaluation of the secrete polynomial. This problem is solved by introducing the zk-VPD scheme in zk-VSQL [71]. The basic idea is to hide the polynomial evaluation behind a Pedersen commitment, and use Σ-protocol to reason about the value. When instantiated to univariate polynomials, the verification cost of zk-VPD scheme needs 5 pairings. We show a close-to-optimal improvement to it so that only 2-pairings are needed.

We fully implement the `zkreg` system and demonstrate its performance by proving all ELF files in a Linux Centos 7 malware free. The source code and experimental data is available at: [3]

1.2 Related Work

Since its inception [41] zero knowledge proof has generated not only theoretical interests but also numerous applications, e.g., electronic payment systems [54, 11, 68, 57, 53]), anonymous machine learning [55, 52], verifiable cloud database [71], and authentication protocols [51]. zkMiddleBox [45] allows one to to reason about if an encrypted DNS request does not contain any blocked words. The problem size of our work is much larger and we resolve to distirbued commit-and-prove scheme to cut prover cost. See [65] for a complete survey of the recent progress of the field.

This work is related to the line of research on zk-proofs for state machines [12, 14, 35]. In [12], QAP based zkSNARK is used to prove TinyRAM programs over von Neumann architecture, with prover complexity $O(NT)$ where N is the program size and T the trace length. The complexity is improved to additive in [14] to $O((N+T)\mathsf{log}(N+T))$ by using a $T\mathsf{log}(T)$ non-deterministic switch network for validating consistency of memory operations. In [35], the prover complexity is further improved to $O(N+T)$ via VOLE based zero knowledge, accomplishing constant overhead memory access for RAM machines. We achieved similar prover complexity in the context of zkSNARK.

We envision our work as an instance of the Commit-and-Prove (CP) proof systems [30, 2, 28, 27, 6]. In particular, the k-ccGro16 component in our proof

[3] `https://github.com/xfu2006/zkregex.git`

system is a direct generalization of the ccGro16 system in LegoSnark [29]. In most applications of CP-Snark, a commitment is used to connect heterogeneous proof systems. In our work, the commitment is also used for fixing witness input wires so that Fiat-Shamir is applied. One witness input wire is fed back to the circuit, which results in a Feedback Commit-and-Prove scheme. The FB-CP scheme can be regarded as an addition to the $\mathtt{CP_{link}}$ constructions in [28, 27, 6, 18]. It essentially asserts the equivalence of two committed polynomials in two proof systems. Compared in particular with the Pedersen like $\mathtt{CP_{link}}$ component of LegoSnark [28], FB-CP also accomplishes $O(1)$ proof size and verifier cost. Using FB-CP there is no need to create extra prover/verifier keys to apply the linear subspace scheme [48]. In addition, there is no restriction on how the second polynomial is committed, i.e., it can be constructed via sophisticated polynomial operations in arithmetic circuit.

In the seminal KZG paper [46], an almost (but not complete) zk-set solution is provided. Our solution based on pairing based accumulator [63] offers both complete zk and great concrete performance. We note that the performance of lookup arguments has been improved rapidly [36, 69, 33]. In particular, the most recent construction [33] offers quasi-linear complexity independent of the size of super-set. However, at this moment, it still lacks zero-knowledge. Integrating lookup arguments in our framework remains as a future direction. We refer readers to [25, 58, 72, 17, 50, 38] for other related constructions.

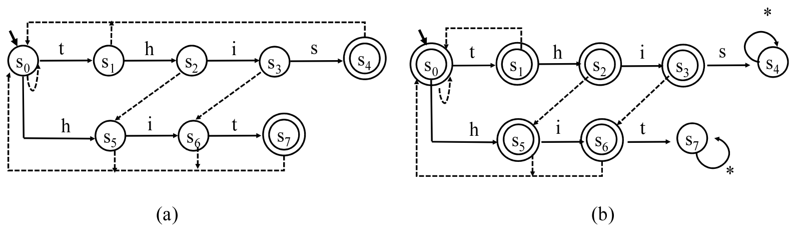
1.3 Contributions

The following is a summary of our contributions. (1) We present a *2-phase proof* solution to the zk-Reg problem, for exploiting the locality of AC-DFA. The scheme brings improvement of prover performance by an order of magnitude. (2) We design a *Feedback Commit-and-Prove (FB-CP)* scheme to connect Σ-protocols and zk-SNARK systems. The scheme provides an alternative to the $\mathtt{CP_{link}}$ constructions in [28, 6, 27] with $O(1)$ proof size and verification time. (3) We develop several cryptographic constructions that improve the state of art: (a) a close-to-optimal univariate instantiation of the zk-VPD commitment scheme [71], which only requires 2-pairings for verification; and (b) a zk-subset proof which further improves the efficiency of the zk-batch bilinear membership proof given in [63]. (4) We provide a full implementation and evaluation of the proposed 2-phase proof scheme. We show 107× speed-up of Groth16 proof generation compared with prior work. We prove that all ELFs in a Linux CentOs 7 malware free, obtaining a $1.96MB$ proof that can be verified in 6.5 seconds.

2 Preliminaries

2.1 AC-DFA

An Aho-Corasick automaton (AC-DFA) [3] is a deterministic finite state machine built from a set of strings as the virus signature database. Let $V = \{v_0, ..., v_k\}$



(a)

(b)

Fig. 1. AC-DFA

and let $m = \sum_{i=0}^{k} |v_i|$. The AC-DFA for V can be constructed in $O(m)$ time. Running a string s over the AC-DFA discovers all contained virus patterns with no more than $2|s|$ transitions. This is achieved by dividing the AC-DFA's transition function into two parts: a regular forward edge relation and a failure edge function. Each state has up to one failure edge, which points to a state that holds the longest suffix of the current input string. Figure 1 shows an example of the AC-DFA for $V = \{\texttt{this}, \texttt{hit}\}$. One can see that the acceptance run of an input string "`thit`" travels through state sequence $s_0, s_1, s_2, s_3, s_6, s_7$ with the transition $s_3 \to s_6$ as a failure link. Negation of an AC-DFA for a virus signature set, e.g., as shown in Figure 1(b), can capture the set of "benign" strings.

2.2 Notations and Security Assumptions

Let λ be the security parameter. We denote negligible in λ as $\epsilon(\lambda)$. We write $f = \epsilon(\lambda)$ as $f \approx 0$, and $|f - g| = \epsilon(\lambda)$ as $f \approx g$. Let $\mathcal{G}$ be a generator of bilinear groups, i.e., $(p, \mathbf{g}_1, \mathbf{g}_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$. Here $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ all have prime order p, with $\mathbf{g}_1$ ($\mathbf{g}_2$) as the generator of $\mathbb{G}_1$ ($\mathbb{G}_2$). $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is the bilinear map s.t.. for any $a, b \in \mathbb{Z}_p$: $e({\mathbf{g}_1}^a, {\mathbf{g}_2}^b) = e(\mathbf{g}_1, \mathbf{g}_2)^{ab}$ and $e(\mathbf{g}_1, \mathbf{g}_2)$ is the generator of $\mathbb{G}_T$. Field elements are denoted as English letters or Greek symbols with regular font (e.g., a, b, α, β), and group elements using bold fonts (e.g., $\mathbf{g}$, $\mathbf{h}$). Given a field $\mathbb{F}$, $a \xleftarrow{\$} \mathbb{F}$ means to sample a from $\mathbb{F}$ uniformly.

Following Groth16 [44], we write $\mathbb{G}_1$ and $\mathbb{G}_2$ as additive groups. Given $a \in \mathbb{Z}_p$, we denote ${\mathbf{g}_1}^a$ as $[a]_1$, and similar are $\mathbb{G}_2$ and $\mathbb{G}_T$. For instance, ${\mathbf{g}_1}^a {\mathbf{g}_1}^b$ is written as $[a]_1 + [b]_1$ or $[a+b]_1$, $({\mathbf{g}_2}^a)^b$ as $[ab]_2$, and $e({\mathbf{g}_1}^a, {\mathbf{g}_2}^b)$ is denoted as $[a]_1 \cdot [b]_2$ or $[ab]_T$. We use $\boldsymbol{a} \in \mathbb{Z}_p^n$ to define a vector of n field elements: $(\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1})$, where $\boldsymbol{a}_0$ is the first element of $\boldsymbol{a}$. Similarly, $[\boldsymbol{a}]_1$ represents a vector of $\mathbb{G}_1$ elements $([\boldsymbol{a}_0]_1, \ldots, [\boldsymbol{a}_{n-1}]_1)$. Given $\boldsymbol{b} \in \mathbb{Z}_p^n$, $\boldsymbol{b}[\boldsymbol{a}]_1$ is a vector $([\boldsymbol{a}_0 \boldsymbol{b}_0]_1, \ldots, [\boldsymbol{a}_{n-1} \boldsymbol{b}_{n-1}]_1)$, and dot product $[\boldsymbol{a}]_1 \cdot [\boldsymbol{b}]_2$ is defined as $([\boldsymbol{a}_0 \boldsymbol{b}_0]_T, \ldots, [\boldsymbol{a}_{n-1} \boldsymbol{b}_{n-1}]_T)$. We may also use $\boldsymbol{a}[i]$ to indicate its i'th element (index from 0). For a two dimensional array $\boldsymbol{b}$, $\boldsymbol{b}[i]_j$ denote the element at row i column j. Given a multi-set S, its *support set* is $\hat{S} = \{x \mid x \in S\}$. For instance, given $T = \{1, 2, 3, 2, 3\}$, $\hat{T} = \{1, 2, 3\}$.

Our system is based on a number of security assumptions: discrete logarithm assumption (DL) [46], q-Strong Diffie-Hellman (q-SDH) [46], q-Power Knowledge of Exponent (q-PKE) [43], and q-computational power Diffie-Hellman (q-CPDH) [43]. These are frequently used assumptions in commitment schemes and zk-

SNARK. We refer readers to [43, 46, 44] for their design idea and computational hardness. Their formal definitions are given in Appendix A.

2.3 Σ-Protocols and zk-SNARK

Given an NP relation R, we say that $(\mathcal{P}, \mathcal{V})$ is an interactive proof system if prover $\mathcal{P}$ demonstrates the knowledge of $(x, w) \in R$ to verifier $\mathcal{V}$, disclosing zero knowledge about the witness w. A 3-round interactive protocol is called a Σ-protocol. The Σ-protocols we present in this paper are perfect complete, knowledge sound, and honest verifier zero knowledge (HVZK). Its formal definition is given in Appendix B. A Σ-protocol can be converted to non-interactive using the Fiat-Shamir heuristic under the random oracle model [34].

We use an adapted notation from [26] to specify zk-protocols. Consider Schnorr's DLOG protocol as an example: $\Sigma_{\mathtt{DLOG}}(\mathbf{h})\{(x) : \mathbf{h} = [x]_1\}$. Here DLOG in $\Sigma_{\mathtt{DLOG}}$ is a mnemonic. $(\mathbf{h})$ is the public information. The tuple before ":" is the secret known by prover only, i.e., (x). Then the statement inside curly braces states the relation: the prover knows the secret discrete logarithm of $\mathbf{h}$. Applying Fiat-Shamir heuristics to $\Sigma_{\mathtt{DLOG}}(\mathbf{h})$, we get an non-interactive proof and denote it as $\pi_{\mathtt{DLOG}}(\mathbf{h})$. Schnorr proof can be generalized to multiple bases. We use $1/0 \leftarrow \mathtt{CheckDLOG}(\mathbf{C}_x, \pi_x, (\mathbf{g}_1, \ldots, \mathbf{g}_k))$ to denote the verifier function for $\pi_{\mathtt{DLOG}}$: the prover knows $x, r_2, \ldots, r_k$ so that $\mathbf{C}_x = x\mathbf{g}_1 + r_2\mathbf{g}_2 + ... + r_k\mathbf{g}_k$.

zkSNARK (Zero Knowledge Succinct Non-interactive ARgument of Knowledge) systems (e.g., [37, 44, 59, 67, 62, 10, 13, 9, 39, 5, 22, 23]) provide generic specification for arbitrary relation, and succinct proof size and verification time. Recently, many bilinear group friendly encryption and hash algorithms are developed, e.g., Poseidon [42], and MiMC [4]. In our system, we encode the AC-DFA as an arithmetic circuit. This circuit is then converted to a Rank-1 Constraint System (R1CS), and then to a Quadratic Arithmetic Program (QAP). The QAP is then fed to a modified Groth16 system [44] for proof generation.

2.4 Commitment Schemes

A commitment is a cryptographic primitive that allows one to commit to a secret message and later to open it.

Pedersen Vector Commitment Let $\mathbf{g}, \mathbf{h} \in \mathbb{G}$ and $\mathsf{log}_{\mathbf{g}}(\mathbf{h})$ is unknown to the prover. Given $s \in \mathbb{Z}_p$ and r sampled from $\mathbb{Z}_p^*$, a Pedersen commitment [60] to s is defined as $\mathtt{CommitPed}(s, r) = s\mathbf{g} + r\mathbf{h}$. Given $\mathbf{g} \in \mathbb{G}^{n+1}$ where no linear relation is known for $\mathbf{g}$. a Pedersen vector commitment [23, 63] to $\boldsymbol{a} = (a_0, \ldots, a_{n-1})$, using opening r, is $\mathtt{CommitPed}(\boldsymbol{a}, r) = \sum_{i=0}^{n-1} \boldsymbol{a}_i\mathbf{g}_i + r\mathbf{g}_n$. Pedersen commitment is perfect hiding and computational binding.

Polynomial Commitment Given key $\left(\left([s^i]_1\right)_{i=0}^q, \left([\alpha s^i]_1\right)_{i=0}^q\right)$, the KZG commitment [46] $\mathbf{C}_{p,1} \in \mathbb{G}_1$ to a polynomial $p(X) \in \mathbb{Z}_p[X]$ is defined as $[p(s)]_1$. Let $p(X) = \sum_{i=0}^d a_i X^i$. $\mathbf{C}_{p,1}$ is then $\sum_{i=0}^d a_i [s^i]_1$, where each $[s^i]_1$ is from the prover key. One can provide $\mathbf{C}_{p,2} = [\alpha p(s)]_1$ as the *proof of knowledge* for $\mathbf{C}_{p,1}$. By the q-PKE assumption (Definition 6), the following bilinear pairing check convinces the verifier that the prover knows all coefficients of the hiding $p(X)$: $\mathbf{C}_{p,1} \cdot [\alpha]_2 = \mathbf{C}_{p,2} \cdot [1]_2$. Given a point t, and let $y = p(t)$. There exist a polynomial $w(X) = (p(X) - y)/(X - t)$. Then based on the q-SDH assumption (Definition 5), the following check proves that the $p(X)$ behind $\mathbf{C}_{p,1}$ evaluates to y at point t: $(\mathbf{C}_{p,1} - [y]_1) \cdot [1]_2 = [w(s)]_1 \cdot [s - t]_2$.

zk-VPD scheme The KZG commitment scheme is not hiding and it also leaks information of a point evaluation. In [71], a zero knowledge ℓ-variate polynomial delegation scheme (zk-VPD) is presented to address the problem. The idea is to hide the polynomial evaluation behind a Pedersen commitment, thus retaining zero knowledge. Concretely, a zk-VPD commitment to a polynomial $p(X)$ is a pair $(\mathbf{C}_{p,1}, \mathbf{C}_{p,2})$ where $\mathbf{C}_{p,2}$ is the proof of knowledge for $\mathbf{C}_{p,1}$, and $\mathbf{C}_{p,1}$ is an extended KZG commitment to $p(X)$ blinded by random factor. Its opening operation: $(\mathbf{C}_y, \pi) \leftarrow \texttt{Open}(p, r_p, t, \sigma_{\Sigma})$, produces a proof which asserts that the secret $p(X)$ behind $\mathbf{C}_{p,1}$ evaluates to a secret value y at t and $\mathbf{C}_y$ is a Pedersen commitment to y. The zk-VPD scheme is complete, binding and zero knowledge. We provide its formal definition for univariate polynomials in Appendix C.

3 Σ-Protocols

We provide two Σ-protocols in this section. The first protocol shows that a committed polynomial evaluates at a certain point in zero knowledge, and the second protocol proves a subset relation between two zero knowledge sets. We then apply the inner product arguments [24] to both protocols for proof aggregation.

3.1 ZK-Proof for Polynomial Evaluation

The zk-VPD scheme [71] provides a zero-knowledge solution for showing that an ℓ-variate polynomial behind a commitment evaluates to a secret value at a public point. [4] In this section, we provide a drop-in replacement for the construction provided in [71] for univariate polynomials. It provides lower concrete cost (2 pairings vs 5 pairings for verifier work and saving half of the prover cost). In Appendix C, we provide the original univariate zk-VPD construction as a baseline for comparison. The verifier cost of our construction is close to optimal, considering that the standard (non-zk and non-hiding) KZG commitment scheme's evaluation proof costs 2 pairings at verifier.

Figure 2 presents the $\Sigma_{\texttt{univar\_zk\_vpd}}$ construction. For simplicity, we do not distinguish between prover and verifier keys. Its `CommitPoly` and `Check` algorithms are the same as zk-VPD [71]. We thus focus on the zero knowledge polynomial evaluation proof (`Open` and `Verify`). We briefly describe its design idea and compare it with the construction presented in [71].

Recall that in the non-hiding KZG commitment scheme, the evaluation proof is built upon the following observation. If $y = p(t)$ then there exists a polynomial

[4] Many hiding variations of KZG, e.g., [46, Section 3.3] and Marlin [31, Appendix B.2], still leak evaluation values, which needs bounded-zk technique [31, 27]. In our case we need the evaluation to be zk, and hence adapting zk-VPD is the best fit.

1 **Trusted Set-up:** $\sigma_\Sigma \leftarrow \texttt{Setup}(1^\lambda, G, q)$
Parse G as $(p, \mathbf{g}_1, \mathbf{g}_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$. Sample s_1, s_2, α from $\mathbb{Z}_p^*$. Compute $\gamma \leftarrow \left(\left((s_1)^i\right)_{i=0}^{q}, \left(\alpha (s_1)^i\right)_{i=0}^{q}, s_2, \alpha s_2, s_1 s_2\right)$.
Let prover-verifier key $\sigma_\Sigma = ([\gamma]_1, [\gamma]_2)$. Return σ_Σ.

2 **Commit to Polynomial:** $(\mathbf{C}_p, \pi_p) \leftarrow \texttt{CommitPoly}(p(X), r_p, \sigma_\Sigma)$: Return $([p(s_1) + r_p s_2]_1, [\alpha(p(s_1) + r_p s_2)]_1)$.

3 **Validate Polynomial Commitment:** $1/0 \leftarrow \texttt{Check}(\mathbf{C}_p, \pi_p, \sigma_\Sigma)$: Return 1 if and only if $\mathbf{C}_p \cdot [\alpha]_2 = \pi_p \cdot [1]_2$.

4 **Zk-Prove Polynomial Evaluation:** $(\mathbf{C}_y, \pi) \leftarrow \texttt{Open}(p(X), r_p, t, \sigma_\Sigma)$:
Compute: $y \leftarrow p(t)$. $q_1(X) \leftarrow (p(X) - y)/(X - t)$.
Sample r_y, r_q from $\mathbb{Z}_p^*$. Compute: $\mathbf{C}_y \leftarrow [y + r_y s_2]_1$, $\mathbf{C}_1 \leftarrow [q_1(s_1) + r_q s_2]_1$, and $\mathbf{C}_2 \leftarrow [s_2((s_1 - t) r_q + (r_y - r_p))]_1$.
Compute Schnorr proofs:
$\pi_{\mathsf{DLOG}y} \leftarrow \pi_{\mathsf{DLOG}}(\mathbf{C}_y, ([1]_1, [s_2]_1))\{(y, r_y) : \mathbf{C}_y = y[1]_1 + r_y[s_2]_1\}$, and
$\pi_{\mathsf{DLOG}2} \leftarrow \pi_{\mathsf{DLOG}}(\mathbf{C}_2, ([s_2(s_1 - t)]_1, [s_2]_1))\{(r_1, r_2) : \mathbf{C}_2 = r_1[s_2(s_1 - t)]_1 + r_2[s_2]_1\}$.
Let $\pi = (\mathbf{C}_1, \mathbf{C}_2, \pi_{\mathsf{DLOG}y}, \pi_{\mathsf{DLOG}2})$. Return $(\mathbf{C}_y, \pi)$.

5 **Check Polynomial Evaluation:** $1/0 \leftarrow \texttt{Verify}(\mathbf{C}_p, \mathbf{C}_y, \pi, \sigma_\Sigma)$:
Assumption: $\mathbf{C}_p$ has a knowledge extraction proof.
Parse π as $(\mathbf{C}_1, \mathbf{C}_2, \pi_{\mathsf{DLOG}y}, \pi_{\mathsf{DLOG}2})$. Return 1 if and only if:
$\texttt{CheckDLOG}(\mathbf{C}_y, \pi_{\mathsf{DLOG}y}, ([1]_1, [s_2]_1)) = 1 \ \wedge\ \texttt{CheckDLOG}(\mathbf{C}_2, \pi_{\mathsf{DLOG}2}, ([s_2(s_1 - t)]_1, [s_2]_1)) = 1 \ \wedge\ (\mathbf{C}_p - \mathbf{C}_y + \mathbf{C}_2) \cdot [1]_2 = \mathbf{C}_1 \cdot [s_1 - t]_2$.

Fig. 2. $\Sigma_{\texttt{univar\_zk\_vpd}}$: Univariate zk-VPD Scheme

$q_1(X)$ s.t. for any $u \in \mathbb{Z}_p$: $p(u) - y = q_1(u)(u - t)$. This is tested using an equation of two pairings: $([p(s_1)]_1 - [y]_1) \cdot [1]_2 = [q_1(s_1)]_1 \cdot [s_1 - t]_2$. Following the zk-VPD construction [71] all KZG commitments of polynomials are blinded in the $\texttt{Open}()$ operation in Figure 2: $[p(s_1)]_1$ is mapped to $\mathbf{C}_p$, $[q_1(s_1)]_1$ mapped to $\mathbf{C}_1$. Then another commitment $\mathbf{C}_2$ is then introduced to "balance off" the cross-terms introduced by the blinding factors. This is verified by the third equality check in the $\texttt{Verify}()$ operation. The first two equations in $\texttt{Verify}()$ perform the standard Schnorr DLOG verification, which demonstrates the prover's knowledge of multi-base discrete logarithm of $\mathbf{C}_2$ and $\mathbf{C}_y$.

In [71], all zk-VPD commitments come with a proof of knowledge. We observed that: $\mathbf{C}_1$ (for $q_1(X)$) can be used without a proof of knowledge. Then with a slight tweak of the formula of $\mathbf{C}_2$, we can cut from 5 pairings needed at the verifier side to only 2 pairings. In addition, because no proof of knowledge is needed for $\mathbf{C}_1$, the prover cost can be cut in half, because there is no need to generate $[\alpha q_1(s_1)]_1$. In Appendix **??**, we present the proof of our improved construction. The intuition of the proof is that even though an adversary can try to submit $\mathbf{C}_1$ with a "relaxed" requirement of no proof of knowledge needed, creating a fake zk-evaluation proof still breaks the DL and q-SDH assumptions.

Lemma 1. *Under the DL, q-PKE, and q-SDH assumptions, the $\Sigma_{univar\_zk\_vpd}$ construction in Figure 2 is a univariate zk-VPD scheme defined in Definition 9.*

```
1 Prove Product Relation: $(\mathbf{C}'_q, \pi) =$
  $\texttt{PrvSubset}(\mathbf{C}_p, \mathbf{C}_q, \pi_q, [w(s_1)]_1, [\alpha w(s_1)]_1, [w(s_1)]_2, r_p, r_q, \sigma_\Sigma, \texttt{bMutate})$:
  # $p(X) = q(X)w(X)$. $r_p$ is the opening of $\mathbf{C}_p$, and $r_q$ is opening of $\mathbf{C}_q$
  Sample $r_w$ from $\mathbb{Z}_p^*$. If bMutate sample $r'_q$ from $\mathbb{Z}_p^*$ otherwise $r'_q \leftarrow 0$. Let
  $\mathbf{C}'_q \leftarrow \mathbf{C}_q + r'_q[s_2]_1$ and $\pi'_q \leftarrow \pi_q + r'_q[\alpha s_2]_1$. $r''_q \leftarrow r_q + r'_q$
  Compute: $\mathbf{C}_w \leftarrow [w(s_1)]_1 + r_w[s_2]_1$ and $\pi_w \leftarrow [\alpha w(s_1)]_1 + r_w[\alpha s_2]_1$.
  $\mathbf{C}_{w,2} \leftarrow [w(s_1)]_2 + r_w[s_2]_2$. Note $\mathbf{C}_w \in \mathbb{G}_1$ and $\mathbf{C}_{w,2} \in \mathbb{G}_2$.
  Compute: $\mathbf{C}_1 \leftarrow r_w\mathbf{C}'_q + r''_q\mathbf{C}_w - r''_q r_w[s_2]_1 - [r_p]_1$.
  $\pi_1 \leftarrow \pi_{\texttt{DLOG}}(\mathbf{C}_1, (\mathbf{C}'_q, \mathbf{C}_w, [s_2]_1, [1]_1))\{(r_1, r_2, r_3, r_4) : \mathbf{C}_1 =$
  $r_1\mathbf{C}_{q'} + r_2\mathbf{C}_w + r_3[s_2]_1 + r_4[1]_1\}$.
  Let $\pi = (\pi'_q, \mathbf{C}_1, \pi_1, \mathbf{C}_w, \pi_w, \mathbf{C}_{w,2})$. Return $(\mathbf{C}_{q'}, \pi)$.
2 Check Subset Relation: $1/0 \leftarrow \texttt{VerSubset}(\mathbf{C}_p, \mathbf{C}_q, \pi, \sigma_\Sigma)$:
  Assumption: $\mathbf{C}_p$ is validated with knowledge extraction proof.
  Parse $\pi$ as $(\pi_q, \mathbf{C}_1, \pi_1, \mathbf{C}_w, \pi_w, \mathbf{C}_{w,2})$. Return 1 if and only if
  $\texttt{CheckDLOG}(\mathbf{C}_1, \pi_1, (\mathbf{C}_q, \mathbf{C}_w, [s_2]_1, [1]_1)) = 1 \ \wedge\ \mathbf{C}_w \cdot [\alpha]_2 = \pi_w \cdot [1]_2 \ \wedge\ \mathbf{C}_q \cdot [\alpha]_2 =$
  $\pi_q \cdot [1]_2 \ \wedge\ \mathbf{C}_w \cdot [1]_2 = [1]_1 \cdot \mathbf{C}_{w,2} \ \wedge\ \mathbf{C}_p \cdot [1]_2 + \mathbf{C}_1 \cdot [s_2]_2 = \mathbf{C}_q \cdot \mathbf{C}_{w,2}$.
```

Fig. 3. $\Sigma_{\texttt{subset}}$: Zero Knowledge Subset Protocol

Efficiency: As each extended Schnorr proof has two bases, each sends 1 $\mathbb{G}_1$ and 2 $\mathbb{Z}_p$ elements. In total, the zk-polynomial evaluation proof costs 4 $\mathbb{G}_1$ and 4 $\mathbb{Z}_p$, and verifier spends 2 pairings plus 6 group operations over $\mathbb{G}_1$.

3.2 ZK Subset Proof

Accumulators such as RSA [16, 8] and bilinear pairing based [58] can compress a large set of elements into one succinct representation and provide membership or subset proofs. In this paper, we extend the bilinear accumulator [58, 63] for representing (multi)-set of transitions and states of an AC-DFA. We extend and present a more efficient construction than the recent work of zero knowledge batch membership for bilinear accumulator [63]. We are able to cut a Σ-product proof from the protocol of [63], resulting in a reduction of 2 group elements and 2 field elements from 5 group and 5 field elements in proof, had our technique applied in the context of [63].

Note that different from [63] (where the algebraic group model (AGM) is assumed), we assume the plain model in this paper due to recent discussions [47, 70] over AGM. That is: when a proof of knowledge for a group element is required in the scheme, we do not assume it automatically exists (as in AGM).

Definition 1. *Given a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$. For a multi-set $A = \{a_1, ..., a_n\} \in \mathbb{F}^n$, define $p_A(X) = \prod_{i=1}^{n}(X - a_i)$ be its* vanishing polynomial. *Given a key $\left(\left([(s_1)^i]_1\right)_{i=0}^{q}, [s_2]_1\right)$, the bilinear accumulator of A is denoted as $\mathbf{acc}_A = [p_A(s_1)]_1$. Let $r \xleftarrow{\$} \mathbb{Z}_p^*$ be the opening, its zero knowledge accumulator (zk-set) is defined as $\mathbf{zkset}_A = \mathbf{acc}_A + r[s_2]_1$.*

Mutating Zero Knowledge Set and Proof It may be tempting to build the zk-subset proof upon the zk-polynomial evaluation proof via a simple application of Schwartz-Zippel. The problem is that: if a different random point is used, then all the three evaluation proofs have to be re-computed, which is costly in our problem domain.[5] We follow another technical route, mainly by deriving the zk-batch membership proof [63, Section 7.1]. Our goal is to re-use the published subset of AC-DFA states/transitions and their subset proofs, without leaking which subset we use in the 2-phase proof scheme. In Figure 3 we present the $\Sigma_{\texttt{subset}}$ protocol.

We first give an intuition of `PrvSubset`. Assume that there exists a proof for: $q(X)$ is a factor polynomial of $p(X)$ (hiding behind zk-VPD commitments $\mathbf{C}_q$ and $\mathbf{C}_p$ respectively). There exists a witness polynomial $w(X)$ s.t. $p(X) = q(X)w(X)$. The zk-VPD commitment of $q(X)$ and proof of knowledge are pre-computed, and so are the KZG commitment to $w(X)$. The job of `PrvSubset` is to generate a new commitment $\mathbf{C}'_q$ from $\mathbf{C}_q$ and the subset proof, without disclosing $\mathbf{C}_q$. This is done by padding a blinding factor to them, e.g., $\mathbf{C}'_q = \mathbf{C}_q + r'_q[s_2]_1$, at negligible cost.

Zero Knowledge Subset Proof We build a zk-subset proof between two zero knowledge sets based on the batch-membership proof [63, Section 7.1]. Our scheme provides improved concrete cost by cutting a Σ-product proof.

Given $p(X) = q(X)w(X)$ and $\mathbf{acc}_p = [p(s_1)]_1$, $\mathbf{C}_q = [q(s_1)]_1 + r_q[s_2]_1$ and $\mathbf{C}_w = [w(s_1)]_1 + r_w[s_2]_1$. The basic idea of [63] is to build a proof for Equation 1 which is eventually reduced to: $[p(s_1)]_1 \cdot [1]_2 = [q(s_1)]_1 \cdot [w(s_1)]_2$ that asserts $p(X) = q(X)w(X)$.

$$\mathbf{acc}_p \cdot [1]_2 + \mathbf{C}_1 \cdot [s_2]_2 = \mathbf{C}_q \cdot \mathbf{C}_w \tag{1}$$

Here $\mathbf{C}_1$ is introduced to balance off the extra cross-terms caused by the openings of $\mathbf{C}_q$ and $\mathbf{C}_w$. Let $r_1 = r_q$, $r_2 = r_w$, and $r_3 = -r_q r_w$. One can verify that: $\mathbf{C}_1 = r_1\mathbf{C}_w + r_2\mathbf{C}_q + r_3[s_2]_1$.

In [63], Schnorr style zk-proofs are used to establish prover's knowledge of random exponents r_1, r_2, and r_3. In particular, the product relation: $r_3 = -r_1 r_2$. Our observation is that: the zk-proof for the product relation is not needed, given the proof of knowledge for $\mathbf{C}_q$ and $\mathbf{C}_w$.

The `PrvSubset()` operation in Figure 3 provides the details. First, the $\mathbf{acc}_p$ in [63] is converted to a fully hiding zk-VPD commitment $\mathbf{C}_p$ in our context. The major difference from the proof in [63] is that we just need to provide a DLOG proof for proving the knowledge of the four exponents of $\mathbf{C}_1$, thus, cutting the Σ-product proof used in [63]. Formally, the `VerSubset()` algorithm can be regarded as a Σ-protocol, and we denote it as $\Sigma_{\texttt{subset}}$. It proves that the polynomial hiding behind $\mathbf{C}_q$ is a factor of the polynomial behind $\mathbf{C}_p$.

[5] In [21] one proof asserts evaluation at multiple points. It can be used for batching proofs in our domain. However, it is not perfectly hiding. Extra Schnorr-style proofs are needed to cancel terms generated by blinding factors.

Lemma 2. *Under the DL, q-PKE, q-SDH, and q-CPDH assumptions, $\Sigma_{\mathtt{subset}}$ is perfectly complete, computational sound, and HVZK.*

Appendix E presents the proof for Lemma 2. The following Lemma allows to use the polynomial product relation for proving subsets.

Lemma 3. *Let $p(X)$ be the vanishing polynomial for a multi-set S of elements in $\mathbb{Z}_p$, and let $\mathbf{C}_p$ be a valid commitment to $p(X)$ with proof of knowledge. For any $\mathbf{C}_q$ if there exists $\pi_{q\subseteq p}$ s.t. $\mathtt{VerSubset}(\mathbf{C}_p, \mathbf{C}_q, \pi_{q\subseteq p}, \sigma_\Sigma) = 1$, then the polynomial behind $\mathbf{C}_q$ vanishes at a subset of S.*

Efficiency: The prover has to perform $O(n\mathsf{log}(n))$ field operations (for computing $w(X)$), and $O(n)$ group operations, where n is the degree of $p(X)$ behind $\mathbf{C}_p$. The proof consists of 5 $\mathbb{G}_1$, 1 $\mathbb{G}_2$ and 4 $\mathbb{Z}_p$ elements. The verifier spends 9 pairings and 5 $\mathbb{G}_1$ multiplications. Given n zk-subset claims and proofs, it is possible to aggregate them into one single proof of $\mathsf{log}(n)$ size, using inner pairing product argument [24]. Details are show in Appendix G.

4 Arithmetic Circuit

The arithmetic circuit in `zkreg` takes a *secret* witness stream of input characters (4-bit nibbles) and its acceptance path by the AC-DFA. It enforces the AC-DFA semantics (e.g., the fail-edges), performs encryption and hash operations, generates the vanishing polynomials for states and transitions, and evaluates them at a given random point. We introduce several design decisions that optimize the prover performance.

An AC-DFA is a tuple (Σ, S, s_0, F, T) where $\Sigma = \{i\}_{i=0}^{|\Sigma|-1}$ is the alphabet, $S = \{i\}_{i=0}^{|S|-1}$ and $F = \{i\}_{i=0}^{|F|-1}$ are the set of states and final states. $s_0 \in S$ (its encoded value being 0) is the initial state and each transition in T is a tuple (s, c, t, b) where $s, t \in S$ are the source/destination states, $c \in \Sigma$ is the input character, and b is a Boolean flag indicating if the transition is a fail-edge. Given $\mathbb{Z}_p$ of a bilinear group (usually $p > 2^{252}$), we define an encoding function $\rho : T \to \mathbb{Z}_p$, letting $m = \lceil \mathsf{log}(|S|) \rceil$ and $n = \lceil \mathsf{log}(|\Sigma|) \rceil$:

$$\rho((s, c, t, b)) = b + 2c + 2^{n+1}s + 2^{m+n+1}t + 2^{2m+n+2} \qquad (2)$$

In practice for ClamAV, we take $m = 25$ and $n = 4$. Note that the circuit has to perform a range check on all elements of a transition, e.g., b is really a Boolean flag. The last item 2^{2m+n+2} in Equation 2 is used to separate the sets of transitions and states, thus saving the range proof cost for states.

4.1 Support Set

Let U denote the set of transitions along an acceptance path. It is a multi-set, i.e., one element may appear multiple times. When the prover needs to argue in the

Σ-protocol that it is a subset of all possible transitions of the AC-DFA, its *support set* $\hat{U}$ (i.e., the set of the distinct elements in U) is needed by the Σ-protocol. It is possible to encode the relation between U and $\hat{U}$ using the switch-network technique in TinyRAM [15], however, the circuit size is $|U|\mathsf{log}(|U|)$. In the following, we introduce a technique that incurs $O(|U|\mathsf{log}^2(|U|))$ field operations (due to half-GCD) for supplying the circuit witness, but it results in $O(|U|)$ circuit size. The scheme runs faster in concrete time. For instance, for a malware-free file of size $1MB$ and depth 10, the half-GCD field operation costs 662 seconds of CPU time, which is less than 2.7% of the total CPU time spent on the proof.

Given U and $\hat{U}$, let Let p_U and $p_{\hat{U}}$ be the vanishing polynomials. Let p'_U be the derivative of p_U (thus monic and non-zero). It is known that these polynomials satisfy the following [32, Ch. 9] for fields of a large prime order.

$$\frac{p_U}{\gcd(p_U, p'_U)} \equiv p_{\hat{U}} \tag{3}$$

Example 41 Let $U = \{1, 1, 2\}$ and $\hat{U} = \{1, 2\}$. Clearly, $p_U(X) = (X-1)^2(X-2)$, and $p'_U(X) = (X-1)(2(X-2)+(X-1))$. Then, $\gcd(p_U, p'_U) = (X-1)$ and $\frac{p_U}{\gcd(p_U,p'_U)}$ is $(X-1)(X-2)$, which is identical to $p_{\hat{U}}$. □

According to Bézout's Lemma, for any $f, g \in \mathbb{Z}_p[X]$: $\gcd(f, g) = 1$ if and only if there are $a, b \in \mathbb{Z}_p[X]$ s.t. $af + bg = 1$. We denote a and b as $\mathtt{BZ}_1(f, g)$, and $\mathtt{BZ}_2(f, g)$.

Definition 2. *Let A be a multi-set of field elements of $\mathbb{Z}_p$, let p_A be its vanishing polynomial, and p'_A the formal derivative of p_A. The* evidence polynomials *for p_A, denoted as $\mathtt{Evi}(p_A)$, is defined as the following tuple of seven polynomials:*

$$\left(\begin{array}{l} p_A, p'_A, \gcd(p_A, p'_A), \frac{p_A}{\gcd(p_A,p'_A)}, \frac{p'_A}{\gcd(p_A,p'_A)}, \\ \mathtt{BZ}_1\left(\frac{p_A}{\gcd(p_A,p'_A)}, \frac{p'_A}{\gcd(p_A,p'_A)}\right), \mathtt{BZ}_2\left(\frac{p_A}{\gcd(p_A,p'_A)}, \frac{p'_A}{\gcd(p_A,p'_A)}\right) \end{array}\right)$$

Note that in Definition 2, the 4th polynomial, i.e., $\frac{p_A}{\gcd(p_A,p'_A)}$ is the vanishing polynomial for the support set, i.e., $p_{\hat{A}}$. Given the coefficients of the polynomials in $\mathtt{Evi}(p_A)$, one can use arithmetic circuit to efficiently check p'_A is the derivative of p_A. For the rest of the check on $\mathtt{Evi}(p_A)$, we present a Feedback Commit-and-Prove (FB-CP) scheme in Section 5.2 to take advantage of Schwartz-Zippel.

4.2 Circuit Specification

We present the specification of $\mathbb{CIRC}_{n,\mathbf{A}}$ in Figure 4. Here, n is the input length and $\mathbf{A}$ is the AC-DFA that it encodes. $\mathbb{CIRC}_{n,\mathbf{A}}$ is defined as a function that takes public input wires $\boldsymbol{I}$ and witness input wires that are split into two segments: $\overrightarrow{w_1}$ and $\overrightarrow{w_2}$. All wire values are elements of $\mathbb{Z}_p$. Intuitively, the circuit performs consistency checks on the input wires, and produces an output wire H where $H = \mathsf{hash}(\mathsf{encrypt}(\boldsymbol{c}, k))$ for a secret key k and input string $\boldsymbol{c}$. The commitment to $\overrightarrow{w_2}$ will be taken from the Groth16 system and used to connect with Σ protocols.

Circuit Specification: $H \leftarrow \mathbb{CIRC}_{n,\mathbf{A}}(\boldsymbol{I}, \overrightarrow{w_1}, \overrightarrow{w_2})$
Parse $\boldsymbol{I}$ as $\{r\}$. Parse $\overrightarrow{w_1}$ as $(\boldsymbol{T}, \texttt{Evi}(p_{\boldsymbol{u}}), k)$ where for each $i \in [0, n)$:
$\boldsymbol{T}_i = (\boldsymbol{s}_i, \boldsymbol{c}_i, \boldsymbol{s}_{i+1}, \boldsymbol{b}_i)$, and $\boldsymbol{u} = \boldsymbol{s} \cup \rho(\boldsymbol{T})$. Parse $\overrightarrow{w_2} = (v, r_2, \boldsymbol{s}_1, \boldsymbol{s}_n)$.
Abort if any of the following assertion fails.

1. Assert $\forall i \in [0, n) : \boldsymbol{b}_i \in \{0, 1\}$.
2. Assert $\forall i \in [0, n) : \boldsymbol{c}_i \in \Sigma$.
3. Assert $\boldsymbol{s}_n < |F|$.
4. Fail-edge semantics: $\forall i \in [0, n-1) : \boldsymbol{b}_i = 1 \Rightarrow \boldsymbol{c}_{i+1} = \boldsymbol{c}_i$.
5. $\boldsymbol{t} \leftarrow \Big(\rho(\boldsymbol{T}_i)\Big)_{i=0}^{n-1}$. $v_1 \leftarrow (\prod_{i=0}^{n-1}(\boldsymbol{t}_i - r))(\prod_{i=0}^{n}(\boldsymbol{s}_i - r))$. Parse $\texttt{Evi}(p_{\boldsymbol{u}}) = \big(p_i\big)_{i=1}^{7}$. Assert $p_1(r) = v_1$, and verify $\texttt{Evi}(p_{\boldsymbol{u}})$ is valid: $p_2 = p_1' \ \wedge \ p_1(r) = p_3(r)p_4(r) \ \wedge \ p_2(r) = p_3(r)p_5(r) \ \wedge \ p_6(r)p_4(r) + p_7(r)p_5(r) = 1$.
6. Assert $v = p_4(r)$.

Recollect input chars from $\boldsymbol{c}$ by skipping those for fail edges. Let them be $\boldsymbol{d}$.
Return $\mathsf{hash}(\mathsf{encrypt}(\boldsymbol{d}, k))$.

Fig. 4. $\mathbb{CIRC}_{n,\mathbf{A}}$ for AC-DFA $\mathbf{A}$ with Input Size n

As shown in Figure 4, there is only one public input wire: r. It is used as the random input point for evaluating polynomials. The first witness segment $\overrightarrow{w_1}$ consists of the acceptance path of the secret input string, evidence polynomials, and a symmetric encryption key k. $\boldsymbol{T}$ is a vector of transitions, where each element is written as a tuple $(\boldsymbol{s}_i, \boldsymbol{c}_i, \boldsymbol{s}_{i+1}, \boldsymbol{b}_i)$. Let $\boldsymbol{u} = \boldsymbol{s} \cup \rho(\boldsymbol{T})$, i.e., the encoding of states and transitions. Let $\hat{\boldsymbol{u}}$ be its support set. The second component of $\boldsymbol{w_1}$ is $\texttt{Evi}(p_{\boldsymbol{u}})$, and its correctness will be validated in the circuit.

Let $v \leftarrow p_{\hat{\boldsymbol{u}}}(r)$, and $r_2 \overset{\$}{\leftarrow} \mathbb{Z}_p^*$. Witness segment 2 consists of $(v, r_2, \boldsymbol{s}_1, \boldsymbol{s}_n)$. Its Pedersen commitment is used to bridge with Σ-protocols. s_1 and s_n are the beginning and last state of the acceptance path, which is used to connect the proofs for consecutive chunks of a big file. r_2 is the opening (blinding factor) of the Pedersen commitment.

In Figure 4, actions (1)-(4) encode the AC-DFA machinery, mainly the logic of fail-edges. Note that the range proof for states can be saved due to encoding tricks introduced earlier. In action (5), the multi-set of transition encoding and states are built. The circuit computes its evaluation at random point r, and then use it to compare with the $p_1(X)$ in $\texttt{Evi}(p_{\boldsymbol{u}})$, thus establishing that the p_1 in $\texttt{Evi}(p_{\boldsymbol{u}})$ is a vanishing polynomial of the multi-set of transitions and states. Then the rest of the check verifies Bézout's identity relation and others for the validity of $\texttt{Evi}(p_{\boldsymbol{u}})$. This establishes that the $p_4(X)$ in $\texttt{Evi}(p_{\boldsymbol{u}})$ is the vanishing polynomial for the support set, i.e., $p_{\hat{\boldsymbol{u}}}$. Then it verifies the v in segment 2 is equal to $p_4(r)$. The circuit assumes that all witness wires are committed before the random input r is supplied as input. This is addressed in Section 5.

1 **Trusted Set-up:** $(\sigma_G, \tau) \leftarrow \text{Setup}(R)$
Parse $R = \big(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \ell, \{u_i(X), \mathsf{v}_i(X), w_i(X)\}_{i=0}^{m}, t(X), \{b_j\}_{j=0}^{k}\big)$. Sample α, β, γ, $\{\delta_i\}_{i=1}^{k}$, x from $\mathbb{Z}_p^*$.
For $1 \leq j \leq k$, compute $\kappa_j \leftarrow \left\{\frac{\beta u_i(x)+\alpha \mathsf{v}_i(x)+w_i(x)}{\delta_j}\right\}_{i=b_{j-1}}^{b_j-1}$. Define simulator trapdoor $\tau = \big(\alpha, \beta, \gamma, \{\delta_i\}_{i=1}^{k}, x\big)$. Generate σ_1, σ_2 as below:

$$\sigma_1 \leftarrow \begin{pmatrix} \alpha, \beta, \{\delta_i\}_{i=1}^{k}, \left\{x^i\right\}_{i=0}^{n-1}, \{\kappa_j\}_{j=1}^{k}, \\ \left\{\frac{\beta u_i(x)+\alpha \mathsf{v}_i(x)+w_i(x)}{\gamma}\right\}_{i=0}^{\ell}, \left\{\frac{x^i t(x)}{\delta_k}\right\}_{i=0}^{n-2} \end{pmatrix}, \text{ and } \sigma_2 \leftarrow \begin{pmatrix} \beta, \gamma, \{\delta_i\}_{i=1}^{k}, \\ \left\{x^i\right\}_{i=0}^{n-1} \end{pmatrix}$$

Compute $\sigma_G \leftarrow ([\sigma_1]_1, [\sigma_2]_2])$. Return (σ_G, τ).

2 **Prove:** $([A]_1, [B]_2, \{[C_i]_1\}_{i=1}^{k}) \leftarrow \text{Prove}((a_0, \ldots, a_m), R, ([\sigma_1]_1, [\sigma_2]_2))$.
Sample r, s, and $\{r_j\}_{j=1}^{k}$ from $\mathbb{Z}_p^*$. Compute $\pi = \big([A]_1, [B]_2, \{[C_j]_1\}_{j=1}^{k}\big)$ where

$$[A]_1 \leftarrow \left[\alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta_k\right]_1 \qquad [B]_2 \leftarrow \left[\beta + \sum_{i=0}^{m} a_i \mathsf{v}_i(x) + s\delta_k\right]_2$$

For each $1 \leq j \leq k-1$: $[C_j]_1 \leftarrow \left[\left(\sum_{i=b_{j-1}}^{b_j-1} a_i \left(\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta_j}\right)\right) + r_j\delta_k\right]_1$.
For the last segment, $[C_k]_1 \leftarrow$
$\left[\left(\sum_{i=b_{k-1}}^{b_k-1} a_i \left(\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta_k}\right)\right) + \frac{h(x)t(x)}{\delta_k} + As + Br - rs\delta_k - \sum_{i=1}^{k-1} r_i\delta_i\right]_1$.

3 **Verify:** $0/1 \leftarrow \text{verify}((a_0, \ldots, a_\ell), \pi, R, ([\sigma_1]_1, [\sigma_2]_2]))$
Parse π as $\big([A]_1, [B]_2, \{[C_j]_1\}_{j=1}^{k}\big)$. Return 1 if and only if:
$[A]_1 \cdot [B]_2 = [\alpha]_1 \cdot [\beta]_2 + \sum_{i=0}^{\ell} a_i \left[\frac{\beta u_i(x)+\alpha \mathsf{v}_i(x)+w_i(x)}{\gamma}\right]_1 \cdot [\gamma]_2 + \sum_{i=1}^{k}[C_i]_1 \cdot [\delta_i]_2$.

4 **Simulation:** $\pi \leftarrow \text{simulate}((a_0, \ldots, a_\ell), \tau, R)$
Sample A, B, $\{C_i\}_{i=1}^{k-1}$ from $\mathbb{Z}_p^*$. Compute
$C_k \leftarrow \frac{AB-\alpha\beta-\left(\sum_{i=0}^{\ell} a_i(\beta u_i(x)+\alpha \mathsf{v}_i(x)+w_i(x))\right)-\sum_{i=1}^{k-1}\delta_i C_i}{\delta_k}$. Return
$\pi = \big([A]_1, [B]_2, \{[C_i]_1\}_{i=1}^{k}\big)$.

Fig. 5. *k*-Segment ccGro16 (k-ccGro16) Protocol

5 2-Phase Proof Scheme

In this section, we provide a modified Groth16 [44] zkSNARK system, based on which, we develop a 2-phase proof strategy to exploit the locality of AC-DFA. We design a Feedback Commit-and-Prove (FB-CP) scheme for realizing the 2-phase strategy, by connecting *Σ*-protocols with Groth16.

5.1 *k*-ccGro16

The *k*-segment *commitment-carrying* Groth16 scheme (k-ccGro16) is a generalization of the ccGro16 proof system in LegoSNARK [29, Appendix H.5], by extending it from one committed segment to multiple.

Definition 3. *A k-segment quadratic arithmetic program (k-QAP) is a tuple*

$$R = \left(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \ell, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X), \{b_j\}_{j=0}^{k}\right)$$

where its statement is $(a_0, \ldots, a_\ell) \in \mathbb{Z}_p^\ell$ *with* $a_0 = 1$*, and witness* $(a_{\ell+1}, \ldots, a_m) \in \mathbb{Z}_p^{m-\ell}$*. Let* n *be the degree of* $t(X)$*, the witness is accepted if and only if there exists a* $n-2$ *degree polynomial* $h(X)$ *s.t.*

$$\sum_0^m a_i u_i(X) \sum_0^m a_i v_i(X) = \sum_0^m a_i w_i(X) + h(X)t(X)$$

A witness tuple is divided into k *segments, and their scope is defined by a* boundary vector $\left\{b_j\right\}_{j=0}^{k}$ *in ascending order, where* $b_0 = \ell + 1$ *and* $b_k = m + 1$*. Each segment* i *corresponds to a slice of the witness, i.e.,* $(a_{b_{i-1}}, \ldots, a_{b_i - 1})$*. It is required that for each segment* $j \in [1, k-1]$*:* $\left\{u_i(x)\right\}_{i=b_{j-1}}^{b_j - 1}$ *are* linearly independent.[6]

Algorithm 5 presents the details of k-ccGro16. We briefly explain the design idea here and refer readers to [29, Appendix H.5] for details, from which k-ccGro16 is derived. The trusted setup generates prover keys $([\sigma_1]_1, [\sigma_2]_2)$ and simulator trap-door τ. Compared with the standard Groth16 [44], for each witness segment j, there is an additional trapdoor parameter δ_j (in contrast to one δ in Groth16). The $\left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}\right\}$ component in Groth16 is similarly "segmented" into k segments (denoted as κ_j for each $j \in [1, k]$) in the new scheme. k-ccGro16 generates a proof $\pi = \left([A]_1, [B]_2, \{[C_j]_1\}_{j=1}^{k}\right)$, which splits the $[C]_1$ in Groth16 into k pieces, i.e., $\{[C_j]_1\}_{j=1}^{k}$. A blinding component $r_j \delta_k$ is added to each C_j for zero knowledge. δ_j ensures knowledge extraction and avoids inter-mix of values from different segments. Its correctness is stated in Theorem 1.

Theorem 1. *The protocol given in Algorithm 5 is perfectly complete, perfectly zero knowledge, and statistical knowledge sound against adversaries using a polynomial number of generic bilinear group operations.*

5.2 Feedback Commit-and-Prove (FB-CP)

In Figure 8, we present FB-CP for realizing the 2-phase proof strategy. The basic idea is to fix part of the arithmetic circuit inputs first by committing to them, recompute the polynomial input point by applying Fiat-Shamir, and then compute the rest of Groth16 proof. Note that the $\texttt{CP}_{\texttt{lnk}}$ in LegoSnark [28] cannot replace FB-CP as it is needed by $\mathbb{CIRC}_{n,\mathbf{A}}$ for validating evidence polynomials.

[6] Linear independence can be ensured by adding dummy constraints when compiling QAP [14, Lemma 2.4], or directly reasoning about relation between R1CS variables.

1 Setup: $(\mathbb{R}, \sigma_\Sigma, \sigma_c, \{(\mathbf{A}_i, \pi_{\mathbf{A}_i})\}_{i=0}^{u}, \{(\mathbf{W}_i, \pi_{\mathbf{W}_i}, \mathbf{W}_{i,2})\}_{i=0}^{u}) \leftarrow \texttt{Setup}(\lambda, \mathbf{A})$

1. Parse AC-DFA $\mathbf{A}$ as (Σ, S, s_0, F, T). Compute $\mathbf{S} = S \cup \rho(T)$. Let u be the max depth of $\mathbf{S}$. Compute $\{\mathbf{S}_i\}_{i=1}^{u}$, where $\mathbf{S}_i$ is the subset of $\mathbf{S}$ bounded by depth i. Build $\mathbb{CIRC}_{n,\mathbf{A}}$ from $\mathbf{A}$, and let $\mathbb{R}$ be its QAP. Run $G \leftarrow \mathcal{G}(1^\lambda)$.
2. $(\sigma_c, \tau_c) \leftarrow$ k-ccGro16.$\texttt{Setup}(\mathbb{R})$ and $\sigma_\Sigma \leftarrow \Sigma_{\texttt{univar\_zk\_vpd}}.\texttt{Setup}(1^\lambda, G, |\mathbf{S}|)$.
3. For each $i \in [1, u]$: compute $p_{\mathbf{S}_i}$, and let $w_i(x) = p_{\mathbf{S}}/p_{\mathbf{S}_i}$. Compute: $(\mathbf{A}_i, \pi_{\mathbf{A}_i}) \leftarrow \Sigma_{\texttt{univar\_zk\_vpd}}.\texttt{CommitPoly}(p_{\mathbf{S}_i}, 0, \sigma_\Sigma)$, and $(\mathbf{W}_i, \pi_{\mathbf{W}_i}) \leftarrow \Sigma_{\texttt{univar\_zk\_vpd}}.\texttt{CommitPoly}(w_i, 0, \sigma_\Sigma)$. $\mathbf{W}_{i,2} = [w_i(s_1)]_2$. Note that $\mathbf{S}_u = \mathbf{S}$ and $\mathbf{A}_u$ is the accumulator for $\mathbf{S}$. Return $(\mathbb{R}, \sigma_\Sigma, \sigma_c, \{(\mathbf{A}_i, \pi_{\mathbf{A}_i})\}_{i=0}^{u}, \{(\mathbf{W}_i, \pi_{\mathbf{W}_i}, \mathbf{W}_{i,2})\}_{i=0}^{u})$.

2 Prove: $\pi \leftarrow \texttt{Prove}(\mathbf{A}, s, k, \sigma)$

1. Parse σ as $(\mathbb{R}, \sigma_\Sigma, \sigma_c, \{(\mathbf{A}_i, \pi_{\mathbf{A}_i})\}_{i=0}^{u}, \{(\mathbf{W}_i, \pi_{\mathbf{W}_i}, \mathbf{W}_{i,2})\}_{i=0}^{u})$. Run input s over $\mathbf{A}$, and extract multi-set of transitions and states $\boldsymbol{T}$, and compute its vanishing poly $p_{\hat{\mathbf{T}}}$. Sample $r_{\hat{\mathbf{T}}}$ from $\mathbb{Z}_p^*$ and compute: $(\mathbf{C}_{\hat{\mathbf{T}}}, \pi_{\mathbf{C}_{\hat{\mathbf{T}}}}) \leftarrow \Sigma_{\texttt{univar\_zk\_vpd}}.\texttt{CommitPoly}(p_{\hat{\mathbf{T}}}, r_{\hat{\mathbf{T}}}, \sigma_\Sigma)$; $\overrightarrow{w_1} = \{\boldsymbol{T}, \texttt{Evi}(p_{\boldsymbol{T}}), k\}$. Let $b_0 = 3$ and $b_1 = |\overrightarrow{w_1}| + 3$ (because QAP public inputs are $\{1, r, H\}$). Compute the following using k-ccGro16.$\texttt{Prove}$ in Figure 5:
$$\mathbf{C}_1 \leftarrow \left[\left(\sum_{i=b_0}^{b_1-1} \overrightarrow{w_1}[i-3] \left(\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta_1}\right)\right) + r_1\delta_3\right]_1 .$$
2. Apply Fiat-Shamir: $r \leftarrow \mathsf{hash}(\mathbf{C}_1, \mathbf{C}_{\hat{\mathbf{T}}})$. Sample r_2 from $\mathbb{Z}_p^*$. Let s_1 and s_n be the first and last state of T. Define $\boldsymbol{I} = \{r\}$, $\overrightarrow{w_2} = \{p_{\hat{\mathbf{T}}}(r), r_2, s_1, s_n\}$. Compute $H = \mathbb{CIRC}_{n,\mathbf{A}}(\boldsymbol{I}, \overrightarrow{w_1}, \overrightarrow{w_2})$. Convert inputs of $\mathbb{CIRC}_{n,\mathbf{A}}$ to R1CS, and then QAP witness, letting it be $\boldsymbol{a}$. Now apply the full k-ccGro16 (where $\mathbf{C}_1' = \mathbf{C}_1$ for honest prover).
$$\big(\mathbf{A}, \mathbf{B}, \mathbf{C}_1', \mathbf{C}_2, \mathbf{C}_3\big) \leftarrow \text{k-ccGro16}.\texttt{Prove}(\boldsymbol{a}, \mathbb{R}, \sigma_c).$$
3. Compute $w_{\hat{\mathbf{T}}}(X) \leftarrow p_{\mathbf{S}_d}(X)/p_{\hat{\mathbf{T}}}(X)$; $(\mathbf{W}_{\hat{\mathbf{T}}}, \pi_{\mathbf{W}_{\hat{\mathbf{T}}}}) \leftarrow \Sigma_{\texttt{univar\_zk\_vpd}}.\texttt{CommitPoly}(w_{\hat{\mathbf{T}}}, 0, \sigma_\Sigma)$, and $\mathbf{W}_{\hat{\mathbf{T}},2} = [w_{\hat{\mathbf{T}}}(s_1)]_2$. Generate subset and zk-kzg proofs:
$$(\mathbf{C}'_{\mathbf{A}_d}, \pi_{\mathbf{A}_d \subset \mathbf{A}_u}) \leftarrow \texttt{PrvSubset}(\mathbf{A}_u, \mathbf{A}_d, \pi_{\mathbf{A}_d}, \mathbf{W}_d, \pi_{\mathbf{W}_d}, \mathbf{W}_{d,2}, 0, 0, \sigma_\Sigma, \texttt{T});$$
$$(\mathbf{C}_{\hat{\mathbf{T}}}, \pi_{\hat{\mathbf{T}} \subset \mathbf{A}_d}) \leftarrow \texttt{PrvSubset}(\mathbf{C}'_{\mathbf{A}_d}, \mathbf{C}_{\hat{\mathbf{T}}}, \pi_{\hat{\mathbf{T}}}, \mathbf{W}_{\hat{\mathbf{T}}}, \pi_{\mathbf{W}_{\hat{\mathbf{T}}}}, \mathbf{W}_{\hat{\mathbf{T}},2}, r'_{\mathbf{A}_d}, r_{\hat{\mathbf{T}}}, \sigma_\Sigma, \texttt{F}),$$
$$\text{where } r'_{\mathbf{A}_d} \text{ is the opening of } \mathbf{C}'_{\mathbf{A}_d}$$
$$(\mathbf{C}_y, \pi_y) \leftarrow \Sigma_{\mathsf{univar\_zk\_vpd}}.\texttt{Open}(p_{\hat{\mathbf{T}}}, r_{\hat{\mathbf{T}}}, r, \sigma_\Sigma)$$
$$\pi_{y,2} \leftarrow \pi_{\mathsf{SAME}}(\mathbf{C}_y, \mathbf{C}_2)\{(y, r_y, r_1, r_2, r_3, r_4) : \mathbf{C}_y = y[1]_1 + r_y[s_2]_2 \ \wedge$$
$$\mathbf{C}_2 = y[\kappa_2[0]]_1 + r_1[\kappa_2[1]]_1 + r_2[\kappa_2[2]]_1 + r_3[\kappa_2[3]]_1 + r_4[\delta_3]_1\}$$

Return $\big(r, (\mathbf{A}, \mathbf{B}, \mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3), (\mathbf{C}'_{\mathbf{A}_d}, \pi_{\mathbf{A}_d \subset \mathbf{A}_u}), (\mathbf{C}_{\hat{\mathbf{T}}}, \pi_{\hat{\mathbf{T}} \subset \mathbf{A}_d}), (\mathbf{C}_y, \pi_y, \pi_{y,2})\big)$.

3 Verify: $1/0 \leftarrow \texttt{Verify}(\mathbb{R}, H, \pi, \sigma)$

Parse π as $\big(r, (\mathbf{A}, \mathbf{B}, \mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3), (\mathbf{C}'_{\mathbf{A}_d}, \pi_{\mathbf{A}_d \subset \mathbf{A}_u}), (\mathbf{C}_{\hat{\mathbf{T}}}, \pi_{\hat{\mathbf{T}} \subset \mathbf{A}_d}), (\mathbf{C}_y, \pi_y, \pi_{y,2})\big)$. Return 1 iff. all of the following checks pass:

1. k-ccGro16.$\texttt{Verify}((1, r, H), (\mathbf{A}, \mathbf{B}, \mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3), \mathbb{R}, \sigma_c) \ \wedge \ r = \mathsf{hash}(\mathbf{C}_1, \mathbf{C}_{\hat{\mathbf{T}}})$.
2. $\Sigma_{\texttt{subset}}.\texttt{VerSubset}(\mathbf{A}_u, \mathbf{C}_{\mathbf{A}'_d}, \pi_{\mathbf{A}_d \subset \mathbf{A}_u}, \sigma_\Sigma) \ \wedge \ \Sigma_{\texttt{subset}}.\texttt{VerSubset}(\mathbf{C}_{\mathbf{A}'_d}, \mathbf{C}_{\hat{\mathbf{T}}}, \pi_{\hat{\mathbf{T}} \subset \mathbf{A}_d}, \sigma_\Sigma)$.
3. $\Sigma_{\texttt{univar\_zk\_vpd}}.\texttt{Verify}(\mathbf{C}_{\hat{\mathbf{T}}}, \mathbf{C}_y, \pi_y, \sigma_\Sigma) \ \wedge \ \texttt{CheckSAME}(\mathbf{C}_y, \mathbf{C}_2, \pi_{y,2}, \sigma_\Sigma)$.

Fig. 6. Feedback Commit-and-Prove (FB-CP)

Its one-time setup has three steps, where step (2) needs to be carried out by a trusted party. The setup first generates σ_Σ and σ_c, the prover/verifier keys for Σ protocols and k-ccGro16. Then for the given AC-DFA, it publishes $\mathbf{S}$, the encoded states and transitions, and then the corresponding subsets bounded by depth, denoted as $\mathbf{S}_i$ for depth i. For each subset, its bilinear accumulator $\mathbf{A}_i$ and the corresponding proof $\mathbf{W}_i$ are pre-computed, to speed up zk-subset proofs later. For $\mathbf{A}_i$, its knowledge proof is $\pi_{\mathbf{A}_i}$, and likewise $\pi_{\mathbf{W}_i}$ for $\mathbf{W}_i$.

The prover is given an input string s, an encryption key k, and her job is to prove that $s \in L(\mathbf{A})$. Note that s will not be visible to verifier, who can only see $\mathsf{hash}(\mathsf{encrypt}(s, k))$. The prover proceeds in three steps.

In the first step, the prover runs s over the AC-DFA, generates its acceptance path and let $\boldsymbol{T}$ be the encoded states and transitions and $\hat{\mathbf{T}}$ its support set. She then generates $\mathtt{Evi}(p_{\boldsymbol{T}})$, as the proof for $\hat{\mathbf{T}}$ being the support set of $\boldsymbol{T}$, using these polynomials she prepares $\overrightarrow{w_1} = (\boldsymbol{T}, \mathtt{Evi}_{\boldsymbol{T}}, k)$ as the first segment of witness for $\mathbb{CIRC}_{n,\mathbf{A}}$. She then uses the Groth16 prover algorithm to compute $\mathbf{C}_1$ for the first segment (but not all of the Groth16 proof). Then, she computes $\mathbf{C}_{\hat{\mathbf{T}}}$, the zk-VPD commitment to $\hat{\mathbf{T}}$. These two commitments are first presented to the verifier to fix these inputs.

In the second step, the verifier generates a random nonce r. This is simulated using Fiat-Shamir by setting r as a hash on $\mathbf{C}_{\hat{\mathbf{T}}}$ and $\mathbf{C}_1$. Then the prover takes r, feed it to $\mathbb{CIRC}_{n,\mathbf{A}}$, and generates the rest of the intermediate and output wires. Write $\mathtt{Evi}(p_{\boldsymbol{T}})$ as $\mathtt{Evi}_{\boldsymbol{T}}$. Recall that the circuit evaluates all the polynomials in $\mathtt{Evi}_{\boldsymbol{T}}$ and checks their relations (e.g., Bézout's identity). As all polynomial coefficients are already fixed in $\mathbf{C}_1$, given that r is random, the scheme is sound with overwhelming probability by Schwartz-Zippel. For the second segment $\overrightarrow{w_2} = (v, r_2, s_1, s_n)$, the circuit asserts that $v = \mathtt{Evi}_{\boldsymbol{T}}[3](r)$ (i.e., the $p_{\hat{\mathbf{T}}}$ for honest prover) and s_1 and s_n are the first and last states in the acceptance path of s.

In the third step, the prover computes Σ-proofs. The π_{SAME} proof $(\pi_{y,2})$ establishes that $\mathbf{C}_2$ and $\mathbf{C}_y$ as Pedersen commitment hide the same value y (this is verified using $\mathtt{CheckSAME}$ in step (3) of $\mathtt{Verify}$, which is derived from $\mathtt{CheckDLOG}$). Then the $\Sigma_{\mathsf{univar\_zk\_vpd}}$ proof establishes that $y = p_{\hat{\mathbf{T}}}(r)$, where $\mathbf{C}_{\hat{\mathbf{T}}}$ is the commitment to $p_{\hat{\mathbf{T}}}$. This now links the Σ-protocols with k-ccGro16, i.e., the $p_{\hat{\mathbf{T}}}$ behind $\mathbf{C}_{\hat{\mathbf{T}}}$ is indeed the $\mathtt{Evi}_{\boldsymbol{T}}[3]$ in $\overrightarrow{w_1}$ of $\mathbb{CIRC}_{n,\mathbf{A}}$.

Then the two zk-subset proofs establishes that $\hat{\mathbf{T}}$ is indeed a subset of allowed state/transition set $\mathbf{S}$. This is accomplished by proving that $\hat{\mathbf{T}}$ (letting its depth be d) is a subset of $\mathbf{S}_d$, and then proving $\mathbf{S}_d$ is a subset of $\mathbf{S}$. Due to the use of $\Sigma_{\mathsf{subset}}.\mathtt{PrvSubset}$, $\mathbf{C}'_{\mathbf{A}_d}$ is used (instead of $\mathbf{A}_d$), thus retaining zero knowledge.

The verifier is given a hash H, the prover/verifier key σ, and a proof π. By running the $\mathtt{Verify}()$ algorithm, she can be convinced that there exists a string s and a key k s.t. $H = \mathsf{hash}(\mathsf{encrypt}(s, k))$ and $s \in L(\mathbf{A}))$ where $\mathbf{A}$ is the AC-DFA that the prover and verifier agrees upon (for which σ is the prover/verifier key).

Theorem 2. *The protocol given in Algorithm 8 is perfectly complete, perfectly zero knowledge, and computational sound in random oracle model.*

Efficiency: Let $|s|$ be the input string length, d the depth of its acceptance path, and $|\mathbf{S}_d|$ the size of the corresponding subset bounded by depth d. The verifier cost is apparently $O(1)$. The prover has to pay $O(|s|\mathsf{log}^2(|s|) + |\mathbf{S}_d|\mathsf{log}(|\mathbf{S}_d|))$ field operations because of half-GCD algorithm and polynomial division, and $O(|s|\mathsf{log}(|s|))$ group operations for R1CS witness to QAP and $O(|\mathbf{S}_d|)$ group operations for Σ-proofs, and the Groth16 itself costs $O(|s|)$ group operations.

6 Implementation and Evaluation

6.1 System Architecture and Data-set

We provide a full implementation of `zkreg`, based upon `Arkworks` [7] for field-/group arithmetic and Rust OpenMPI [56] for distributed processing. We use an instrumented JSnark [49] for converting arithmetic circuit to R1CS, which is fed to a distributed QAP and Groth16 system implemented in Rust. The `zkreg` implementation consists of 33k LOC of Rust and 15k LOC of Java. Our evaluation uses a cluster of 4 CentOS 7.1 servers (each with 112 vCPU and 448 GB of RAM). The actual GCP network bandwidth is 85Gbps, with average ping time 0.1ms. We use BLS12-381 curve, which provides 128-bit security.

The AC-DFA is generated from the hex-signature database of ClamAV, which consists of 101634 signature strings, with 89% being fixed string patterns that can be directly handled by AC-DFA. [7] There are 11040 regex patterns, which will cause state explosion when compiled to DFA. We take a conservative approximation approach, where we treat regex operators as separators and split each regex signature as a collection of fixed pattern strings. A file is regarded as a virus if it contains any of these patterns. The approximation is "conservative" in the sense that it never reports false-negative for a real virus, but it might report false positives. This is mainly caused by very short patterns resulted from regex signature sets. We removed 8553 such pattern strings. This results in a pattern string collection of 110692, based on which the AC-DFA is built. The AC-DFA contains 19 million states and 342 million transitions. We then run the AC-DFA over all the 2479 ELF files (object and executable files) in a Linux CentOs 7.1, and then there is no false positive reported. The ELF data set has a total size of $747MB$. It exhibits good locality. 96% files have depth less than 40 (which accounts for 22% of the AC-DFA states), with max-depth 252.

6.2 Distributed Processing

We first implement a distributed vector structure using Rust OpenMPI [56], using which coefficients of a polynomial can be stored over a cluster of computer nodes. Then we take the algorithm presented in DIZK [66] and implement distributed FFT operations. Compared with DIZK, we made the following additions, mainly for generating Σ-protocol proofs. We implement polynomial division using Hensel lifting [64] with $O(n\mathsf{log}(n))$ complexity, and also the half-GCD algorithm [61] with $O(n\mathsf{log}^2(n))$ complexity.

[7] Retrieved 03/28/2022. All hex signatures are contained in `main.ndb`

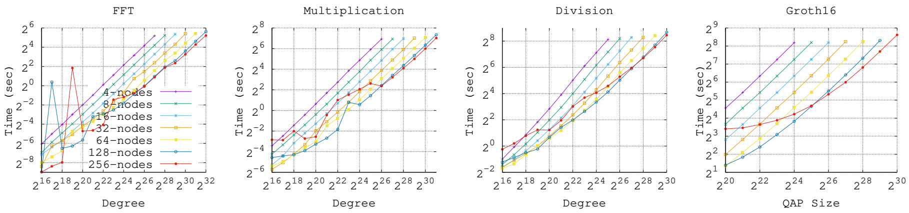


Fig. 7. Distributed System Performance over BLS12-381

Figure 7 presents the performance and scalability of `zkreg`. Distributed FFT is the basis of all. Compared with DIZK, our FFT operation over BN-254 achieves 34× speed-up at degree 2^{28} (2.6 seconds vs 90.5 seconds by DIZK). For curve BLS12-381, it costs 37 seconds for FFT of 2^{32} degree. Half-GCD is only needed at lower degree, However, it does not scale well with MPI. At the degree of 2^{20}, single thread half-GCD costs 450 seconds, running with 32 (256) MPI-nodes costs 310 (401) seconds. We mainly achieve parallelism of GCD via batch processing proofs. We provide a distributed implementation of the Groth16 system similarly. Compared with DIZK [66], with 256-executors for QAP size of 2^{30} over BN-254, our system needs 266 seconds of prover time (107× faster than $2^{14.8}$ seconds needed by DIZK [66, Figure 5(b)]), and for BLS12-381 it needs 393 seconds.

Arithmetic circuit is encoded using the JSnark library [49]. Let n be the length of the input string in terms of bytes, the total cost of the circuit is $124n$ R1CS constraints ($30n$ for AC-DFA, $34n$ for encryption/hash, $60n$ for support-set). For instance, an input file of 1008 bytes needs 125263 R1CS constraints. The circuit cost is high, even when bilinear friendly hash and encryption such as Poseidon [42] and MiMC [4] are used. Also addressing support-set is expensive. An alternative solution is to apply the technique in [35]. However our estimate is that it costs greater ($448n$ R1CS constraints), as the comparison to verify sorted list is expensive. Its advantage is the better scalability than the half-GCD approach. To improve performance in practice, all of circuit generation, conversion to R1CS and QAP are distributed in `zkreg`.

6.3 Proving Linux CentOs 7 Malware Free

To demonstrate the scalability of `zkreg`, we prove all ELFs in a Linux CentOs 7.1 malware free. We need to run a one-time set-up for keys. We generate the subset for depth $D = \{10, 15, 20, 30, 40, 50, 300\}$. The entire set-up takes less than 1 hour on the HPC cluster. Due to limit of RAM resources, we chunk larger files into $1MB$ pieces, and provide additional Schnorr style DLOG proofs to connect the last/first states of consecutive chunks. We thus have 2954 chunks resulted from 2479 ELF files.

Table 1 shows the prover cost break-down for several sample files (each is padded to a size of closest power of 2 and zk-subset proofs are generated for

	File1 Depth: 10 Size: 2^{16}	File2 Depth: 10 Size: 2^{20}	File3 Depth: 300 Size: 2^{20}
	R1CS: 8.3×10^6 Poly: 2.9×10^7	R1CS: 1.3×10^8 Poly: 2.9×10^7	R1CS: 1.3×10^8 Poly: 3.5×10^8
Step	Cost (sec)	Cost (sec)	Cost (sec)
(1) 1-thread half-GCD.	69.6	662.9	700.6
(2) Load Witness.	1.0	9.3	10.1
(3) Groth16 Step 1.	0.3	2.1	2.1
(4) Apply Fiat-Shamir.	0.2	0.3	0.4
(5) QAP Witness.	2.6	17.1	15.8
(6) Groth16 Step 2.	2.5	21.4	21.1
(7) Σ_{subset} Proof 1.	0.1	0.01	0.01
(8) Σ_{subset} Proof 2.	30.3	31.6	242.9
(9) $\Sigma_{\mathsf{univar\_zk\_vpd}}$ Proof.	0.2	3.0	3.4
Turnaround Time	106.8	747.7	996.4
Adjusted Total	37.6	94.5	314.2

All file size in bytes. R1CS performance "faster" than reported in Figure 7 because many witness wire inputs (e.g., states and transitions) are 56-bit numbers.

Table 1. Cost Breakdown of Prover Time

the closest depth in D). We also show the problem size such as the number of R1CS constraints and the highest degree of polynomials in Σ-protocols. Except for step (1) which is executed with a single thread, all others are run with 256 OpenMPI processes. In practice, step (1) is concurrently processed for multiple jobs of the same size configuration to cut down cost. We thus have two ways to compute the sum of total cost. The turnaround time stands for the duration from the moment a proof job is submitted to the moment the proof is generated. The adjusted total reflects the "actual" cost of a job by dividing the HGCD cost by the number of concurrent jobs.

According to Lifewire, [8] the average email size is $75kb$. For $64kb$ files, including the single-thread half-GCD cost, its turn-around proof time is 106 seconds. This implies that the `zkreg` scheme is practical for proving encrypted emails are malware free. The system also demonstrates good performance for $1MB$ files (with depth 10). Its adjusted total is 94 seconds. For files with depth 300, the prover time is significantly higher, however, such files are rare. The peak system memory usage is $880GB$.

For 2479 ELF files ($747MB$) collected, we generate 2954 proofs (2016 bytes each) using 54 hours on the HPC cluster, which costs 1350 dollars on GCP. Each proof can be verified by a single-thread verifier in 36 milliseconds. We further apply the inner product pairing product technique [24] to aggregating all proofs. The details are presented in Appendix G. All 2954 proofs are aggregated into

[8] `https://www.lifewire.com/what-is-the-average-size-of-an-email-message-1171208`

one single proof of $1.96MB$ in 727 seconds, which can be verified in 6.5 (16) seconds with 8 (1) threads.

7 Conclusion

We present an efficient zero knowledge proof system for regular language and demonstrate its performance by proving CentOS 7 malware free.

References

1. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-Preserving Signatures and Commitments to Group Elements. *J. Cryotology*, 2:363–421, 2016.
2. S. Agrawal, C. Ganesh, and P. Mohassel. Non-interactive zero-knowledge proofs for composite statements. In *CRYPTO*, pages 643–673, 2018.
3. A. V. Aho and M. J. Corasick. Efficient string hatching: an aid to bibliographic search. *Communications of ACM*, 18:333–340, 1975.
4. M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In *ASIACRYPT*, pages 191–219, 2016.
5. S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104, 2017.
6. D. Aranha, E. M. Bennedsen, M. Campanelli, C. Ganesh, C. Orlandi, and A. Takahashi. ECLIPSE: Enhanced Compiling Method for Pedersen-Committed zk-SNARK Engines. In *PKC*, pages 584–614, 2022.
7. arkworks contributors. `arkworks` zksnark ecosystem, 2022.
8. N. Baric and B. Pfitzmann. Collision-free accmulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494, 1997.
9. E. Ben-Sasson, I. Bentov, I. Horesh, and M. Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *ICALP*, pages 14:1–14:17, 2018.
10. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *ICAR Cryptology ePrint Archieve*, 46, 2018.
11. E. Ben-Sasson, A. Chiesa, C. Garman andM. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *SSP*, pages 459–474, 2014.
12. E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKS for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *CRYPTO*, pages 90–108, 2013.
13. E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for r1cs. In *EUROCRYPT*, pages 103–128, 2019.
14. E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *USENIX*, pages 781–796, 2014.
15. E. Ben-Sasson, Al Chiesa, D. Genkin, E. Tromer, and M. Virza. Tinyram architecture specification. *available at* `http://www.scipr-lab.org/doc/TinyRAM-spec-0.991.pdf`.

16. J. Benaloh and M. de Mare. One-way accmulators: A decentralized alternative to digital signatures. In *EUROCRYPT*, pages 274–285, 1993.
17. D. Benarroch, M. Campanelli, D. Fiore, K. Gurkan, and D. Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. *available at* `https://eprint.iacr.org/2019/1255.pdf`, 2019.
18. D. Benarroch, M. Campanelli, D. Fiore, K. Gurkan, and D. Kolonelos. Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular. In *Financial Cryptogrpahy*, pages 393–414, 2021.
19. D. Boneh and X. Boyen. Short signatures without random oracles. In *Eurocrypt*, pages 56–73, 2004.
20. D. Boneh and X. Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *J. Cryptology*, 21:149–177, 2008.
21. D. Boneh, J. Drake, B. Fisch, and A. Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. IACR Cryptol. ePrint Arch., 2020.
22. J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT*, pages 327–357, 2016.
23. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *SSP*, pages 315–334, 2018.
24. B. Bünz, M. Maller, P. Mishara, N. Tyagi, and P. Vesely. Proofs for Inner Pairing Products and Applications. In *ASIACRYT*, pages 65–97, 2021.
25. J. Camenisch and A. Lysyanskaya. Dynamic acculators and applications to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76, 2002.
26. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO*, LNCS 1296, pages 410–424, 1997.
27. M. Campanelli, A. Faonio, D. Fiore, A. Querol, and H. Rodriguez. Lunar: A Toolbox for More Efficient Universal and Updatable zkSNARKs and Commit-and-Prove Extensions. In *ASIACRYPT*, pages 3–33, 2021.
28. M. Campanelli, D. Fiore, and A. Querol. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. In *CCS*, pages 2075–2092, 2019.
29. M. Campanelli, D. Fiore, and A. Querol. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. IACR Cryptol. ePrint Arch. 2019:142, 2019.
30. M. Chase, C. Ganesh, and P. Mohassel. Efficient Zero-Knowledge Proof of Algebraic and Non-Algebraic Statements with Applications to Privacy Preserving Credentials. In *CRYPTO*, pages 499–530, 2016.
31. A. Ciesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. P. Ward. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *EUROCRYPT*, pages 738–768, 2020.
32. J. S. Cohen. *Computer Algeba and Symbolic Computation.* A K Peters, 2003.
33. L. Eagen, D. Fiore, and A. Gabizon. cq: Cached quotients for fast lookups. IACR Cryptol. ePrint Arch., 2022.
34. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
35. N. Franzese, J. Katz, S. Lu, R. Ostrovsky, X. Wang, and C. Weng. Constant-Overhead Zero-Knowledge for RAM Programs. In *CCS*, pages 178–191, 2021.
36. A. Gabizon and Z. J. Williamson. Plookup: A simplified polynomial protocol for lookup tables. IACR Cryptol. ePrint Arch., 2020.
37. R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succint nizks without pcps. In *EUROCRYPT*, pages 626–645, 2013.

38. E. Ghosh, O. Ohrimenko, D. Papadopoulos, R. Tamassia, and N. Triandopolous. Zero-knowledge accmulators and set algebra. In *ASIACRYPT (2)*, pages 67–100, 2016.
39. I. Giacomelli, J. Madsen, and C. Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security*, 2016.
40. O. Goldreich, S. Micali, and A. Wigderson. How to prove all np statements in zero-knowledge and a methodology of cryptographic protocol design. In *CRYPTO*, pages 171–185, 1986.
41. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems (extended abstract). In *STOC*, pages 291–304, 1985.
42. L. Grassi, D. Kales, D. Khovratovich, A. Roy, C. Rechberger, and M. Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security*, 2021.
43. J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.
44. J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.
45. P. Grubbs, A. Arun, Y. Zhang, J. Bonneau, and M. Walfish. Zero-Knowledge Middleboxes. In *USENIX Security Seposium 2022*, pages 4255–4272, 2022.
46. A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, pages 177–194, 2010.
47. J. Katz, C. Zhang, and H. Zhou. An Analysis of the Algebraic Group Model. In *ASIACRYPT*, pages 310–322, 2022.
48. E. Kiltz and H. Wee. Quasi-adaptive NIZK for Linear Subspaces Revisited. In *EUROCRYPT*, pages 101–128, 2015.
49. A. Kosba, Z. Chao, A. Miller, Y. Qian, T. H. Chan, C. Papamanthou, R. Pass, and a. shelat. *cøø*: A framework for building composable zero-knowledge proofs. *Cryptology ePrint Archive. 2015/1093*, 2015.
50. H. Lipmaa. Secure accmulators from euclidean rings without trusted setup. In *ACNS*, pages 224–240, 2012.
51. H. Liu and L. Ning. Zero-knowledge authentication protocol based on alternative mode in rfid systems. *IEEE Sensors Journals*, 11:3235–3245, 2011.
52. T. Liu, X. Xie, and Y. Zhang. zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. In *CCS*, pages 2268–2985, 2021.
53. A. Luthra, J. Cavanaugh, H. R. Olcese, R. M. Hirsch, and X. Fu. Zeroaudit. In *ACSAC*, pages 798–812, 2020.
54. I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *SSP*, pages 397–411, 2013.
55. P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *SSP*, pages 19–38, 2017.
56. Rust MPI. MPI binding for Rust, 2022.
57. N. Narula, W. Vasquez, and M. Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *NSDI*, pages 65–80, 2018.
58. L. Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, pages 275–292, 2005.
59. B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *SSP*, pages 238–252, 2013.
60. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.
61. A. SchÖnhage. Schnelle berechnung von kettenbruchentwicklugen. *Acta Informatica*, 1:139–144, 1971.

62. S. Setty. Spartan: Efficient and general-purpose zkSNARKS without trusted setup. In *CRYPTO*, pages 704–737, 2020.
63. S. Srinivasan, I. Karantaidou, F. Baldimtsi, and C. Papamanthou. Batching, Aggregation, and Zero-Knowledge Proofs in Bilinear Accumulators. In *CCS*, pages 2719–2733, 2022.
64. M. Sudan. 6.S897 Algebra and Computation. Polynomial Division. `http://people.csail.mit.edu/madhu/ST12/scribe/lect06.pdf`, 2012.
65. J. Thaler. Proofs, Arguments, and Zero-Knowledge. `https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf`, 2012.
66. H. Wu, W. Zheng, Z. Chiesa, R. A. Popa, and I. Stoica. Dizk: A distributed zero knowledge proof system. In *USENIX Security*, pages 675–692, 2018.
67. T. Xie, J. Zhang, Y. Zhang, and C. Papamanthou annd D. Song. Libra: Succint Zero-Knowledge Proofs with Optimal Prover Computation. In *CRYPTO*, pages 733–764, 2019.
68. T. H. Yuen, S.-F. Sun, J. K. Liu, M. H. Au, M. F. Esgin, Q. Zhang, and D. Gu. Ringct 3.0 for blockchain confidential transaction: Shofter size and stronger security. *available at* `https://eprint.iacr.org/2019/508.pdf`.
69. A. Zapico, V. Buterin, D. Khovratovich, M. Maller, A. Nitulescu, and M. Simkin. Caulk: Lookup Arguments in Sublinear Time. In *CCS*, pages 3121–3134, 2022.
70. M. Zhandry. To Lable, or Not to Label (in Generic Groups). In *CRYPTO*, pages 66–96, 2022.
71. Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. A Zero-Knowledge Version of vSQL. IACR Cryptol. ePrint Arch. 2017:1146, 2017.
72. Y. Zhang, J. Katz, and C. Papamanthou. An expressive (zero-knowledge) set accumulator. In *EuroS&P*, pages 158–173, 2017.

A Security Assumptions

In the following we use $\mathbb{G}$ to denote any of the three groups in $\{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T\}$ and use p as their group order, if context is clear.

Definition 4 (Discrete Logarithm (DL) Assumption [46]). *Let* $\mathbf{g}$ *be a generator of* $\mathbb{G}$*, and* $a \xleftarrow{\$} \mathbb{Z}_p^*$*, for every probabilistic polynomial time (PPT) adversary* $\mathcal{A}$*:*

$$Pr[\mathcal{A}(\mathbf{g}, a\mathbf{g}) = a] \approx 0.$$

Definition 5 (q-Strong Diffie-Hellman (q-SDH) [19, 46]). *Let* $s \xleftarrow{\$} \mathbb{Z}_p^*$*. For any PPT adversary* $\mathcal{A}$ *and any* $c \in \mathbb{Z}_p \setminus \{-s\}$*:*

$$Pr\Big[\mathcal{A}\Big(([s^i]_1)_{i=0}^q, [1]_2, [s]_2\Big) = (c, [\frac{1}{s+c}]_1)\Big] \approx 0$$

Definition 6 (q-Power Knowledge of Exponent (q-PKE) Assumption [43]). *For all PPT adversary* $\mathcal{A}$ *there exists PPT extractor* $\mathcal{X}$ *so that:*

$$Pr\left[\begin{array}{l} B = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda); \\ \mathbf{g} \xleftarrow{\$} \mathbb{G} \setminus \{1\}; \alpha, s \xleftarrow{\$} \mathbb{Z}_p^*; \\ \sigma = \Big(B, \big(s^i\mathbf{g}\big)_{i=0}^q, \big(\alpha s^i\mathbf{g}\big)_{i=0}^q\Big); \\ \Big(c_1, c_2; \boldsymbol{a} \in \mathbb{Z}_p^{q+1}\Big) \leftarrow (\mathcal{A}||\mathcal{X})(\sigma) \end{array} : \begin{array}{l} c_2 = \alpha c_1 \ \wedge \\ c_1 \neq \sum_{i=0}^q \big(\boldsymbol{a}_i s^i \mathbf{g}\big) \end{array}\right] \approx 0$$

Here $(y; z) \leftarrow (\mathcal{A}||\mathcal{X})(x)$ denotes that $\mathcal{A}$ outputs y given input x, and $\mathcal{X}$ outputs z given the same input x and the random tape of $\mathcal{A}$.

Definition 7 (q-computational power Diffie-Hellman (q-CPDH) Assumption [43]). *For all PPT adversary $\mathcal{A}$ and for all $j \in [0, q]$:*

$$Pr\left[\begin{array}{l} (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda); \\ \mathbf{g} \xleftarrow{\$} \mathbb{G} \setminus \{1\}; \alpha, s \xleftarrow{\$} \mathbb{Z}_p^*; \\ y = \mathcal{A}\left(\begin{array}{l} (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e), \\ \left(\alpha s^i \mathbf{g}\right)_{i=0}^{j-1}, \left(\alpha s^i \mathbf{g}\right)_{i=j+1}^{q}, \\ \left(s^i \mathbf{g}\right)_{i=0}^{q} \end{array}\right); \end{array} : y = \alpha s^j \mathbf{g}\right] \approx 0$$

B Definition of Σ-Protocol

We use $\langle \mathcal{P}, \mathcal{V} \rangle(x)$ to denote the *transcript* of interactions between the two parties for input x. If the verifier accepts, we write it as $\langle \mathcal{P}, \mathcal{V} \rangle(x) = \text{accept}$.

Definition 8. *An interactive proof system $(\mathcal{P}, \mathcal{V})$ for NP relation R is an argument of knowledge system if $\mathcal{P}$ and $\mathcal{V}$ are both PPT that satisfy the following:*

- Perfect Completeness: $\forall (x, w) \in R$: $\Pr[\langle \mathcal{P}(w), \mathcal{V} \rangle(x) = \mathit{accept}] = 1$,
- Knowledge Soundness: *We say that the system has knowledge error ϵ if there is a PPT extractor $\mathcal{X}$ for all $\mathcal{P}$ and for all x s.t.*
  $\Pr[w = \mathcal{X}(\mathcal{P}, x) \;\wedge\; (x, w) \in R] > \Pr[\langle \mathcal{P}(w), \mathcal{V} \rangle(x) = \mathit{accept}] - \epsilon$
- Perfect Honest Verifier Zero Knowledge (HVZK): *there is a PPT simulator $\mathcal{S}$ s.t. $\forall (x, w) \in R$: given x, $\mathcal{S}$ generates a transcript whose distribution is identical to that of $\langle \mathcal{P}(w), \mathcal{V} \rangle(x)$.*

C Univariate Instantiation of zk-VPD [71]

Definition 9. *[71, 67] Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be a bilinear group for a given security parameter λ, and $q \in poly(\lambda)$. A zero-knowledge* univariate *polynomial delegation scheme (zk-VPD) for polynomials in $\mathbb{Z}_p[X]$ consists of the following algorithms:*

- $(\sigma_\Sigma, \tau) \leftarrow$ `Setup`$(1^\lambda, q)$ *generates prover/verifier key σ_Σ and simulator trapdoor τ.*
- $(\mathbf{C}_{p,1}, \mathbf{C}_{p,2}) \leftarrow$ `CommitPoly`(p, r_p, σ_Σ) *computes the commitment $\mathbf{C}_{p,1}$ to $p(X)$ with opening (randomness) r_p. $\mathbf{C}_{p,2}$ is the knowledge extraction proof for $\mathbf{C}_{p,1}$.*
- $1/0 \leftarrow$ `Check`$(\mathbf{C}_{p,1}, \mathbf{C}_{p,2}, \sigma_\Sigma)$ *verifies that $\mathbf{C}_{p,2}$ is valid knowledge extraction proof for $\mathbf{C}_{p,1}$ based on q-PKE.*
- $(\mathbf{C}_y, \pi) \leftarrow$ `Open`$(p, r_p, t, \sigma_\Sigma)$ *produces an evaluation proof π for certifying that $p(X)$ evaluates to a secret value y at t and $\mathbf{C}_y$ is a Pedersen commitment to y.*
- $1/0 \leftarrow$ `Verify` $(\mathbb{C}_{p,1}, \mathbb{C}_y, t, \pi, \sigma_\Sigma)$ *verifies the evaluation proof is correct.*

They satisfy the following security properties:

- Completeness: *For $r_p, t \in \mathbb{Z}_p^*$ and polynomial $p(X)$, the following probability is* 1*.*

$$\Pr\left[\begin{array}{c} (\sigma_\Sigma, \tau) \leftarrow \texttt{Setup}(1^\lambda, q) \\ (\mathbf{C}_{p,1}, \mathbf{C}_{p,2}) \leftarrow \\ \texttt{CommitPoly}(p, r_p, \sigma_\Sigma) \\ (\mathbf{C}_y, \pi) \leftarrow \texttt{Open}(p, r_p, t, \sigma_\Sigma) \end{array} : \begin{array}{c} \texttt{Check}(\mathbf{C}_{p,1}, \mathbf{C}_{p,2}, \sigma_\Sigma) = 1 \\ \texttt{Verify}(\mathbf{C}_{p,1}, \mathbf{C}_y, t, \pi, \sigma_\Sigma) = 1 \end{array}\right]$$

- Binding: *For any PPT adversary $\mathcal{A}$, the following probability is negligible.*

$$\Pr\left[\begin{array}{c} (\sigma_\Sigma, \tau) \leftarrow \texttt{Setup}(1^\lambda, q) \\ \begin{pmatrix} \mathbf{C}_{p,1}, \mathbf{C}_{p,2}, \mathbf{C}_y, \\ p, r_p, \\ t, y, r_y, \\ \pi \end{pmatrix} \leftarrow \mathcal{A}(\sigma_\Sigma) \end{array} : \begin{array}{c} \texttt{Check}(\mathbf{C}_{p,1}, \mathbf{C}_{p,2}, \sigma_\Sigma) = 1 \\ \texttt{Verify}(\mathbf{C}_{p,1}, \mathbf{C}_y, t, \pi, \sigma_\Sigma) = 1 \\ (\mathbf{C}_{p,1}, \mathbf{C}_{p,2}) \\ = \texttt{CommitPoly}(p, r_p, \sigma_\Sigma) \\ \mathbf{C}_y = \texttt{CommitPed}(y, r_y, \sigma_\Sigma) \\ y \neq p(t) \end{array}\right]$$

- Zero Knowledge: *Let $\texttt{Real}_{\mathcal{A},p}$ and $\texttt{Ideal}_{\mathcal{A},\mathcal{S}}$ be two games for polynomial p, adversary $\mathcal{A}$ and simulator $\mathcal{S}$. We use δ to denote the trapdoor information that is used for generating prover/verifier key σ_Σ.*

```
$\texttt{Real}_{\mathcal{A},p}(1^\lambda)$ :
 $\sigma_\Sigma \leftarrow \texttt{Setup}(1^\lambda, q)$
 $r_p \xleftarrow{\$} \mathbb{Z}_p$
 $(\mathbf{C}_{p,1}, \mathbf{C}_{p,2}) \leftarrow \texttt{CommitPoly}(p, r_p, \sigma_\Sigma)$
 $k \leftarrow \mathcal{A}(1^\lambda, \mathbf{C}_{p,1}, \mathbf{C}_{p,2}, \sigma_\Sigma)$
 for i in 1..k repeat
   $t_i \leftarrow \mathcal{A}\begin{pmatrix} 1^\lambda, \mathbf{C}_{p,1}, \mathbf{C}_{p,2}, \\ (\mathbf{C}_{y_j})_{j=0}^{i-1}, (\pi_j)_{j=0}^{i-1} \end{pmatrix}$
   $(\mathbf{C}_{y_i}, \pi_i) \leftarrow \texttt{Open}(p, t_i, r_p, \sigma_\Sigma)$
 $b \leftarrow \mathcal{A}\begin{pmatrix} 1^\lambda, (\mathbf{C}_{y_i})_{i=0}^{k}, \\ (\pi_i)_{i=1}^{k}, \sigma_\Sigma \end{pmatrix}$
 output
```

```
$\texttt{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$ :
 $(\sigma_\Sigma, \mathbf{C}_{p,1}, \mathbf{C}_{p,2}, \delta) \leftarrow \mathcal{S}(1^\lambda, q)$
 $k \leftarrow \mathcal{A}(1^\lambda, \mathbf{C}_{p,1}, \mathbf{C}_{p,2}, \sigma_\Sigma)$
 for i in 1..k repeat
   $t_i \leftarrow \mathcal{A}\begin{pmatrix} 1^\lambda, \mathbf{C}_{p,1}, \mathbf{C}_{p,2}, \\ (\mathbf{C}_{y_j})_{j=0}^{i-1}, \\ (\pi_j)_{j=0}^{i-1} \end{pmatrix}$
   $(\mathbf{C}_{y_i}, \pi_i, \delta) \leftarrow \mathcal{S}(t_i, \sigma_\Sigma, \delta)$
 $b \leftarrow \mathcal{A}\begin{pmatrix} 1^\lambda, (\mathbf{C}_{y_i})_{i=0}^{k}, \\ (\pi_i)_{i=1}^{k}, \sigma_\Sigma \end{pmatrix}$
 output b
```

For any PPT adversary $\mathcal{A}$ and any polynomial p, there exists PPT simulator $\mathcal{S}$ such that: $\Pr[\texttt{Real}_{\mathcal{A},p}(1^\lambda) = 1] \approx \Pr[\texttt{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda) = 1]$

In the following, we present the univariate instantiation of zk-VPD construction in [71, Section 3], as a baseline of comparison.

Let prover key be $(([s_1^i]_1)_{i=0}^{q}, ([\alpha s_1^i]_1)_{i=0}^{q}, [s_2]_1, [\beta s_2]_1, [\alpha s_2]_1, [\alpha]_2, [\beta]_2)$, where $(s_1, s_2, \alpha, \beta)$ is the trap-door. We here do not distinguish between prover and verifier key for convenience of presentation. Given $p(X)$ and blinding randomness r_p, its zk-VPD commitment $(\mathbf{C}_{p,1}, \mathbf{C}_{p,2})$ is defined as: $([p(s_1) + r_p s_2]_1, [\alpha(p(s_1) + r_p s_2)]_1)$. A commitment can be validated (proved of knowledge extraction) via checking $\mathbf{C}_{p,1} \cdot [\alpha]_2 = \mathbf{C}_{p,2} \cdot [1]_2$.

Given a validated commitment $\mathbf{C}_{p,1}$ and assume that the prover wants to prove that $p(t) = y$ for a given point t. The proof is produced as follows. The prover samples r_y, r_1, r_2 from $\mathbb{Z}_p^*$, and produces the following:

1. $(\mathbf{C}_{y,1}, \mathbf{C}_{y,2}) = ([y + r_y s_2]_1, [\beta(y + r_y s_2)]_1)$.
2. Compute $q_1(X) = (p(X){-}y)/(x{-}t)$, and $(\mathbf{C}_{1,1}, \mathbf{C}_{1,2}) = ([q_1(s_1){+}r_1 s_2]_1, [\alpha(q_1(s_1){+} r_1 s_2)]_1)$.
3. $(\mathbf{C}_{2,1}, \mathbf{C}_{2,2}) = ([(r_p - r_y) - r_1(s_1 - t)]_1, [\alpha((r_p - r_y) - r_1(s_1 - t))]_1)$

Intuitively, the claim is that given $(\mathbf{C}_{p,1}, \mathbf{C}_{y,1}, t)$, the prover knows the y behind Pedersen Commitment $\mathbf{C}_{y,1}$ s.t. $y = p(t)$ where $p(X)$ is the polynomial behind $\mathbf{C}_{p,1}$. Here $\mathbf{C}_{p,1}$ is assumed to be already validated (knowledge extraction proof provided somewhere else), to make a fair comparison with Section 3.1. The proof consists of $\mathbf{C}_{y,2}$, $\mathbf{C}_{1,1}$, $\mathbf{C}_{1,2}$, $\mathbf{C}_{2,1}$, $\mathbf{C}_{2,2}$. The verification consists of the following pairing checks:

1. $\mathbf{C}_{y,1} \cdot [\beta]_2 = \mathbf{C}_{y,2} \cdot [1]_2$.
2. $\mathbf{C}_{2,1} \cdot [\alpha]_2 = \mathbf{C}_{2,2} \cdot [1]_2$.
3. $\mathbf{C}_{1,1} \cdot [\alpha]_2 = \mathbf{C}_{1,2} \cdot [1]_2$.
4. $(\mathbf{C}_{p,1} - \mathbf{C}_{y,1}) \cdot [1]_2 = \mathbf{C}_{2,1} \cdot [s_2]_2 + \mathbf{C}_{1,1} \cdot [s_1 - t]_2$.

The verification takes 9 pairings, but the first 4 pairings (i.e., for knowledge extraction proof of $\mathbf{C}_{y,1}$ and $\mathbf{C}_{2,1}$ can be replaced by Schnorr-style DLOG proofs with minor cost in proof size in exchange for verification speed as operations over $\mathbb{G}_1$ are much faster than pairings), as a fair comparison with Section 3.1. Thus, the univariate zk-VPD scheme needs 5 pairings for verification. The difference in our $\Sigma_{\texttt{univar\_zk\_vpd}}$ scheme in Section 3.1 is that the knowledge extraction for $\mathbf{C}_{1,1}$ can be saved, and we also change the last equation to save one more pairing, thus cutting the number of pairings to 2.

D Proof for Lemma 1

Proof. The proof for completeness is apparent. For zero knowledge: the simulator, with the trap-door information s_1, can sample $\mathbf{C}_y, \mathbf{C}_1$ from $\mathbb{G}_1$ and then compute $\mathbf{C}_2 \leftarrow (s_1 - t)\mathbf{C}_1 + \mathbf{C}_y - \mathbf{C}_p$. Then run simulators for the DLOG proof for $\mathbf{C}_y$ and $\mathbf{C}_2$. It thus generates a transcript indistinguishable from ideal ones.

For binding, we will show that if a PPT adversary $\mathcal{A}$ is able to craft a fake evaluation proof with non-negligible probability, then one can build a PPT $\mathcal{B}$ that breaks the q-SDH assumption (Definition 5).

Let $\mathcal{B}$ be given a q-SDH challenge [20]: given $\left([s^i]_1\right)_{i=0}^{q}$, $[1]_2$, and $[s]_2$, she needs to present a tuple $(c, [\frac{1}{c+s}]_1)$. From the q-SDH instance, $\mathcal{B}$ builds an instance of $\Sigma_{\texttt{univar\_zk\_vpd}}$ for $\mathcal{A}$ first. Write $s_1 = s$. $\mathcal{B}$ samples s_2 and α, and computes the rest of the keys needed for $\Sigma_{\texttt{univar\_zk\_vpd}}$. For instance, $[s_1 s_2]_1$ is computed as $s_2[s_1]_1$ (as she does not know s_1 but knows s_2). $\mathcal{B}$ then samples t and generates a random polynomial $p(X)$, and passes the $\Sigma_{\texttt{univar\_zk\_vpd}}$ instance to $\mathcal{A}$.

Let $y = p(t)$. Assume $\mathcal{A}$ with non-negligible probability can create a fake evaluation proof $(\mathbf{C}'_1, \mathbf{C}'_2, \pi_{\mathsf{DLOG}y'}, \pi_{\mathsf{DLOG}2'})$ for some $y' \neq p(t)$. By completeness $\mathcal{A}$ can produce a valid proof, and let it be: $(\mathbf{C}_1, \mathbf{C}_2, \pi_{\mathsf{DLOG}y}, \pi_{\mathsf{DLOG}2})$. As both pass $\Sigma_{\texttt{univar\_zk\_vpd}}.\texttt{Verify}$, we have the following (using its third equation): (1)

$(\mathbf{C}_p - \mathbf{C}_y + \mathbf{C}_2)\cdot[1]_2 = \mathbf{C}_1\cdot[s_1 - t]_2$; and (2) $(\mathbf{C}_p - \mathbf{C}_{y'} + \mathbf{C}'_2)\cdot[1]_2 = \mathbf{C}'_1\cdot[s_1 - t]_2$. Subtracting them leads to the following:

$$(\mathbf{C}'_y - \mathbf{C}_y + \mathbf{C}_2 - \mathbf{C}'_2)\cdot[1]_2 = (\mathbf{C}_1 - \mathbf{C}'_1)\cdot[s_1 - t]_2 \tag{4}$$

Apply the knowledge extractors in $\pi_{\mathsf{DLOG}y}$, $\pi_{\mathsf{DLOG}y'}$, $\pi_{\mathsf{DLOG}2}$, $\pi_{\mathsf{DLOG}2'}$, and write $\mathbf{C}_y = [y + s_2 r_y]_1$, $\mathbf{C}_{y'} = [y' + s_2 r'_y]_1$, $\mathbf{C}_2 = [s_2((s-t)r_1 + r_2)]_1$, $\mathbf{C}_{2'} = [s_2((s-t)r'_1 + r'_2)]_1$, where $\mathcal{A}$ knows y, y', r_y, r'_y, r_1, r'_1, r_2, r'_2 (but not s_1 and s_2). Rewrite Equation 4 with the known values by $\mathcal{A}$. We have:

$$\begin{bmatrix} (y' - y) + \\ s_2(r'_y - r_y + r_2 - r'_2) \end{bmatrix}_1 \cdot [1]_1 = \begin{pmatrix} \mathbf{C}_1 - \mathbf{C}'_1 \\ +(r'_1 - r_1)[s_2]_1 \end{pmatrix} \cdot [s_1 - t]_2 \tag{5}$$

Write $\Delta y = y' - y$, $A = r'_y - r_y + r_2 - r'_2$, and $\mathbf{h}_1 = \mathbf{C}_1 - \mathbf{C}'_1 + (r'_1 - r_1)[s_2]_1$. These are all known to $\mathcal{A}$. Then Equation 5 is simplified to the following:

$$[\Delta y + s_2 A]_1 \cdot [1]_2 = \mathbf{h}_1 \cdot [s_1 - t]_2 \tag{6}$$

First of all, note that $\Delta y \neq 0$ by the assumption ($y' \neq y$). This leads to $\Pr[\Delta y + s_2 A = 0] \approx 0$. To see it, assume that $\Delta y + s_2 A = 0$. Since $\Delta y \neq 0$, we have $s_2 A \neq 0$ and thus $A \neq 0$. Then $\mathcal{A}$ can extract $s_2 = -\Delta y / A$, thus breaking the DL assumption.

Now since $\Delta y + s_2 A \neq 0$, and $\mathcal{B}$ knows s_2. $\mathcal{B}$ can compute $\Delta y + s_2 A$ and find its inverse $\frac{1}{\Delta y + s_2 A}$. Plug it into Equation 6, we have $\frac{1}{\Delta y + s_2 A}\mathbf{h}_1 = [\frac{1}{s_1 - t}]_1$. $\mathcal{B}$ can submit $(-t, \frac{1}{\Delta y + s_2 A}\mathbf{h}_1)$ to break q-SDH. This concludes the proof.

E Proof for Lemma 2

Proof. The completeness is apparent. The HVZK proof is also straight-forward and is given in Appendix F. For soundness (binding), since $\mathbf{C}_p$, $\mathbf{C}_q$, and $\mathbf{C}_w$ all have proof of knowledge, apply their knowledge extractors and re-write them as: $\mathbf{C}_p = [p(s_1)]_1 + r_p[s_2]_1$, $\mathbf{C}_q = [q(s_1)]_1 + r_q[s_2]_1$, $\mathbf{C}_w = [w(s_1)]_1 + r_w[s_2]_1$. Similarly, write $\mathbf{C}_1 = r_1\mathbf{C}_q + r_2\mathbf{C}_w + r_3[s_2]_1 + r_4[1]_1$. Here, the prover knows: $p(X)$, $q(X)$, $w(X)$, r_p, r_q, r_w, $\big(r_i\big)_{i=1}^4$.

Now rewrite the last equation in the `VerSubset` algorithm: $\mathbf{C}_p \cdot [1]_2 + \mathbf{C}_1 \cdot [s_2]_2 = \mathbf{C}_q \cdot \mathbf{C}_{w,2}$. This leads to:

$$\begin{pmatrix} (s_2)^2\big(r_1 r_q + r_2 r_w + r_3 - r_q r_w\big) + \\ s_2\big((r_p + r_4) + (r_1 - r_w)q(s_1) + (r_2 - r_q)w(s_1)\big) \\ \big(p(s_1) - q(s_1)w(s_1)\big) \end{pmatrix} = 0 \tag{7}$$

Consider the LHS of Equation 7 as a Laurent polynomial with s_1 and s_2 as variables (controlled by the set-up), and all the others as known coefficients (controlled by the adversary prover). All coefficients of each term $(s_2)^i$ have to be 0, by Schwartz-Zippel. Then for $(s_2)^0$:

$$p(s_1) - q(s_1)w(s_1) = 0$$

If $p(X) - q(X)w(X)$ is not a zero polynomial, it implies that the prover knows a non-trivial polynomial which evaluates to 0 at secret point s_1. It breaks q-CPDH according to Lemma 1 in [43]. The DL and q-SDH are needed for zk-VPD used.

F HVZK Proof of Lemma 2

For HVZK: in the ideal scenario, i.e., the prover is honest and the verifier uses public coin, $\mathbf{C}_p$, $\mathbf{C}_q$ and $\mathbf{C}_w$ are all uniformly distributed and independent from each other. They uniquely determine the rest of the proof except π_1, i.e. (π_w, $\mathbf{C}_{w,2}$, $\mathbf{C}_1$). In another word, given that $\mathbf{C}_p$ and $\mathbf{C}_q$ are already fixed in the statement, $\mathbf{C}_w$ uniquely determines $\mathbf{C}_1$ and the rest of the proof (except π_1), due to the last equation in `VerSubset`:

$$\mathbf{C}_p \cdot [1]_2 + \mathbf{C}_1 \cdot [s_2]_2 = \mathbf{C}_q \cdot \mathbf{C}_{w,2}$$

Now consider the simulator, who has trap-door information s_1 and s_2 and can compute $\frac{1}{s_2}$ from the trapdoor. The simulator samples w and r_w from $\mathbb{Z}_p^*$ and computes $\mathbf{C}_w \leftarrow [w(s_1) + r_w s_2]_1$, $\mathbf{C}_{w,2} \leftarrow [w(s_1) + r_w s_2]_2$, and $\pi_w \leftarrow [\alpha(w(s_1) + r_w s_2)]_1$. Given $\mathbf{C}_p$, $\mathbf{C}_q$, and the above, the simulator computes $\mathbf{C}_1$ as below:

$$\mathbf{C}_1 \leftarrow \frac{1}{s_2}\big((w(s_1) + r_w s_2)\mathbf{C}_q - \mathbf{C}_p\big)$$

Then run the Schnorr DLOG proof simulator for π_1. Now the simulated proof $(\mathbf{C}_1, \pi_1, \mathbf{C}_w, \pi_w, \mathbf{C}_{w_2})$ is statistically indistinguishable from an ideal conversation.

G Proof Aggregation

We use inner-pairing product argument (GIPA) [24] for aggregating proofs. Intuitively GIPA relies on the commitment schemes in [1] for committing to a vector of group/field elements. It provides a log-size proof for asserting that the inner product between two vectors of group/field elements is the claimed value. We formally define the commitment scheme below and refer users to [24] for its constructions.

Definition 10. *[24] Let $B = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be a bilinear group for a given security parameter λ. The generalized inner pairing product argument (GIPA) provides the following algorithms: $\sigma_G = \textit{Setup}(1^\lambda, B, q)$ generates prover/verifier key σ_G for degree bound q. $\mathbf{h}_1 \leftarrow \mathtt{CM}_1(\boldsymbol{a}, \sigma_G)$ generates the commitment for a group element vector $\boldsymbol{a} \in \mathbb{G}_1^n$. Similarly are $\mathtt{CM}_2$ and $\mathtt{CM}_{\mathbb{Z}_p}$ defined for input vectors of $\mathbb{G}_2^n$ and $\mathbb{Z}_p^n$. When the context is clear we omit the commitment key σ_G in notations, e.g., $\mathtt{CM}_1(\boldsymbol{a})$.*

GIPA has three instantiations: $\mathtt{TIPP}$, $\mathtt{MIPP}_u$, and $\mathtt{MIPP}_k$, each provides two functions: $\pi \leftarrow \textit{Prove}(\boldsymbol{a}, \boldsymbol{b}, r, \sigma_G)$: generates inner product argument. $1/0 \leftarrow \textit{Verify}(\mathbf{C}_{\boldsymbol{a}}, \mathbf{C}_{\boldsymbol{b}}, r, Z, \pi, \sigma_G)$: verifies the proof that asserts the following relation. There are three variations: $\mathtt{TIPP}$: $\boldsymbol{a} \in \mathbb{G}_1^n$, $\boldsymbol{b} \in \mathbb{G}_2^n$, $\mathbf{C}_{\boldsymbol{a}} = \mathtt{CM}_1(\boldsymbol{a})$, $\mathbf{C}_{\boldsymbol{b}} = \mathtt{CM}_2(\boldsymbol{b})$, $Z = \sum_{i=0}^{n-1}\big(r^{2i}(\boldsymbol{a}_i \cdot \boldsymbol{b}_i)\big)$. $\mathtt{MIPP}_u$: $\boldsymbol{a} \in \mathbb{G}_1^n$, $\boldsymbol{b} \in \mathbb{Z}_p^n$, $\mathbf{C}_{\boldsymbol{a}} = \mathtt{CM}_1(\boldsymbol{a})$, $\mathbf{C}_{\boldsymbol{b}} = \mathtt{CM}_{\mathbb{Z}_p}(\boldsymbol{b})$, $Z = \sum_{i=0}^{n-1}\big(r^{2i}\boldsymbol{b}_i\boldsymbol{a}_i\big)$. $\mathtt{MIPP}_k$ differs from $\mathtt{MIPP}_u$ in that $\mathbf{C}_{\boldsymbol{b}} = \boldsymbol{b}$.

We show how to aggregate zk-subset proofs and the aggregation for $\Sigma_{\texttt{univar\_zk\_vpd}}$ is similar. We start with several toy examples. Consider two committed polynomials $p_L(X)$ and $p_R(X)$ of degree n and let their coefficient vectors be $\boldsymbol{L}$ and $\boldsymbol{R}$. To prove $\boldsymbol{L} = \boldsymbol{R}$, the verifier samples r, and the prover shows $p_L(r) = p_R(r)$, e.g., using KZG commitment. By Schwartz-Zippel, with overwhelming probability $\boldsymbol{L} = \boldsymbol{R}$. Note that evaluating $p_L(X)$ at r is essentially applying a factor of r^i to all equations, i.e., $\sum_{i=0}^{n} r^i \boldsymbol{L}_i = \sum_{i=0}^{n} r^i \boldsymbol{R}_i$.

We can extend the idea to aggregating knowledge extraction proofs for n KZG commitments in batch:

$$\forall i \in [0, n) : \mathbf{C}_i \cdot [s_2]_2 = \boldsymbol{\pi}_i \cdot [1]_2 \tag{8}$$

Here $\mathbf{C}$ and $\boldsymbol{\pi}$ are the vectors of KZG commitments and their proofs. Let $\mathbf{C}_{\overrightarrow{[s_2]_2}}$ be $\texttt{CM}_2(\overrightarrow{[s_2]_2})$, and $\mathbf{C}_{\overrightarrow{[1]_2}}$ be $\texttt{CM}_2(\overrightarrow{[1]_2})$. These can be pre-computed in set-up. The prover computes $\mathbf{C}_{\mathbf{C}} = \texttt{CM}_1(\mathbf{C})$ and $\mathbf{C}_{\boldsymbol{\pi}} = \texttt{CM}_1(\boldsymbol{\pi})$, an $Z = \sum_{i=0}^{n-1}(r^{2i}\mathbf{C}_i \cdot [s_2]_2)$. Then the prover produces: $\pi_1 = \texttt{TIPP.Prove}(\mathbf{C}, \overrightarrow{[s_2]_2}, r, \sigma_G)$, and $\pi_2 = \texttt{TIPP.Prove}(\boldsymbol{\pi}, \overrightarrow{[1]_2}, r, \sigma_G)$. The verifier runs $\texttt{TIPP.Verify}$ on both the proofs, and the same Z for both sides establishes the proof by Schwartz-Zippel.

G.1 Aggregation of ZK-Subset Proofs

Figure 2 presents the algorithm for aggregating zk-subset proofs. The proof generation algorithm takes n zk-subset proofs, and it outputs an aggregated proof of size $\mathsf{log}(n)$, and a constant size commitment to the vector of statements (denoted as $\mathbf{C}_{\mathbb{X}}$). The verification algorithm, given $\mathbf{C}_{\mathbb{X}}$, verifies that there exists a vector of zk-subset statements behind $\mathbf{C}_{\mathbb{X}}$, each of which has a valid proof.

The aggregation algorithm essentially encodes the `VerSubset` algorithm of $\Sigma_{\texttt{subset}}$ in Figure 3. The following checks are performed for each instance. Equations 9-11 are encoded mainly using $\texttt{MIPP}_u$ and `TIPP`, and then checked in step C4 and C5.

$$\mathbf{C}_w \cdot [\alpha]_2 = \pi_w \cdot [1]_2 \ \wedge \ \mathbf{C}_q \cdot [\alpha]_2 = \pi_q \cdot [1]_2 \ \wedge \tag{9}$$

$$\mathbf{C}_w {\cdot} [1]_2 = [1]_1 {\cdot} \mathbf{C}_{w,2} \ \wedge \tag{10}$$

$$\wedge \ \ \mathbf{C}_p \cdot [1]_2 + \mathbf{C}_1 \cdot [s_2]_2 = \mathbf{C}_q \cdot \mathbf{C}_{w,2} \tag{11}$$

$$\wedge \ \ \texttt{CheckDLOG}(\mathbf{C}_1, \pi_1, (\mathbf{C}_q, \mathbf{C}_w, [s_2]_1, [1]_1)) = 1 \tag{12}$$

The handling of `CheckDLOG` (Equation 12) is different. The main concern here is that when Fiat-Shamir transform is applied, encoding many hash operations in zk-proof can be prohibitively expensive. In this case, we regenerate all `CheckDLOG` proofs in batch so that a same public-coin verifier challenge can be used. Thus, `AggPrv` needs auxiliary information $\boldsymbol{a}$, the discrete logarithm for Equation 12 of each instance.

Concretely, for each instance i: let $x_1, ..., x_4$ be the secret discrete logarithm from aux information. The prover samples r_1, r_2, r_3, r_4 from $\mathbb{Z}_p$ and creates $\boldsymbol{R}[i] = r_1\overrightarrow{\mathbf{C}_q}[i] + r_2\overrightarrow{\mathbf{C}_w}[i] + r_3[s_2]_1 + r_4[1]_1$. Given challenge c, the prover has to respond with $u_j = cx_j + r_j$ for each $j \in [1, 4]$, so that $u_1\overrightarrow{\mathbf{C}_q}[i] + u_2\overrightarrow{\mathbf{C}_w}[i] + u_3[s_2]_1 + u_4[1]_1 = \boldsymbol{R}[i] + c(\overrightarrow{\mathbf{C}_1}[i])$. A parallel composition (conjunction) of all instances results in Part 2 of `AggPrv` in Figure 2. Then the proof is checked in Step C6 of `AggVer`.

The algorithm presented Figure 2 scarifies efficiency for convenience of presentation. For instance, in implementation, all duplicate data items are removed from proof. C2 in `AggVer` can also be pre-computed, thus resulting in $\mathsf{log}(n)$ verifier work.

1 **Aggregate ZK-Subset Proof:** $(\mathbf{C}_{\overrightarrow{\mathbb{X}}}, \pi_a) \leftarrow \mathtt{AggPrv}(\overrightarrow{\mathbb{X}}, \boldsymbol{\pi}, \boldsymbol{a}, \sigma_G)$:

# **Input**: $\overrightarrow{\mathbb{X}}$: statements, $\boldsymbol{\pi}$: proofs, $\boldsymbol{a}$: aux info, σ_G: GIPA key. Let $n = |\overrightarrow{\mathbb{X}}| = |\boldsymbol{\pi}|$. **Output**: $\mathbf{C}_{\overrightarrow{\mathbb{X}}}$: commitment to statements, and π_a: aggregated proof.

Part 1 (Regenerate Batch DLOG Proofs): Parse $\overrightarrow{\mathbb{X}}_i$ as $(\overrightarrow{\mathbf{C}_p}[i], \overrightarrow{\mathbf{C}_q}[i])$. $\overrightarrow{\mathbf{C}_p}$ denotes $\left(\overrightarrow{\mathbf{C}_p}[i]\right)_{i=0}^{n-1}$, and others are similar. Parse $\overrightarrow{\pi}_i$ as $(\overrightarrow{\pi_q}[i]$, $\overrightarrow{\mathbf{C}_1}[i]$, $\overrightarrow{\pi_1}[i]$, $\overrightarrow{\mathbf{C}_w}[i]$, $\overrightarrow{\pi_w}[i]$, $\overrightarrow{\mathbf{C}_{w,2}}[i])$. Parse $\boldsymbol{a}_i$ as $(\overrightarrow{x_1}[i], \overrightarrow{x_2}[i], \overrightarrow{x_3}[i], \overrightarrow{x_4}[i])$.

$\forall i \in [0, n)$: define $\mathbf{h}_i = \left(\overrightarrow{\mathbf{C}_q}[i], \overrightarrow{\mathbf{C}_w}[i], [s_2]_1, [1]_1\right)$. Note that $\forall i \in [0, n)$:
$\overrightarrow{\mathbf{C}_1}[i] = \sum_{j=1}^{4} \overrightarrow{x_j}[i]\mathbf{h}_i[j-1]$.

Sample $\overrightarrow{r_1}$, $\overrightarrow{r_2}$, $\overrightarrow{r_3}$, $\overrightarrow{r_4}$ from $\mathbb{Z}_p^n$ and create $\boldsymbol{R}$ s.t. $\forall i \in [0, n)$: $\boldsymbol{R}[i] \leftarrow \sum_{j=1}^{4} \overrightarrow{r_j}[i]\mathbf{h}_i[j-1]$.

Compute $\mathbf{C}_{\boldsymbol{R}} \leftarrow \mathtt{CM}_1(\boldsymbol{R}, \sigma_G)$.

Apply Fiat-Shamir and let $c \leftarrow \mathsf{hash}\,(\mathbf{C}_{\boldsymbol{R}})$. Compute $\overrightarrow{u_1}, \overrightarrow{u_2}, \overrightarrow{u_3}, \overrightarrow{u_4}$ in $\mathbb{Z}_p^n$ s.t.
$\forall i \in [1, 4], j \in [0, n)$: $\overrightarrow{u_i}[j] = c\overrightarrow{x_i}[j] + \overrightarrow{r_i}[j]$.

Part 2 (Generate Aggregated Proof): Let $\overrightarrow{1} = (1)_{i=0}^{n-1}$, $\overrightarrow{[1]_1} = \left([1]_1\right)_{i=0}^{n-1}$, $\overrightarrow{[s_2]_2} = \left([s_2]_2\right)_{i=0}^{n-1}$, $\overrightarrow{[1]_2} = \left([1]_2\right)_{i=0}^{n-1}$.

Define: $\begin{pmatrix} \overrightarrow{\mathbb{Z}} = (\overrightarrow{u_1}, \overrightarrow{u_2}, \overrightarrow{u_3}, \overrightarrow{u_4}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}) \\ \overrightarrow{\mathbb{G}_{1M}} = (\overrightarrow{\mathbf{C}_w}, \overrightarrow{\mathbf{C}_q}, \overrightarrow{[s_2]_1}, \overrightarrow{[1]_1}, \overrightarrow{\mathbf{C}_q}, \overrightarrow{\pi_q}, \overrightarrow{\mathbf{C}_w}, \overrightarrow{\pi_w}, \overrightarrow{\mathbf{C}_1}, \overrightarrow{\mathbf{C}_p}, \boldsymbol{R}) \end{pmatrix}, \begin{pmatrix} \overrightarrow{\mathbb{G}_{1T}} = (\overrightarrow{[1]_1}, \overrightarrow{\mathbf{C}_q}) \\ \overrightarrow{\mathbb{G}_2} = (\overrightarrow{\mathbf{C}_{w,2}}, \overrightarrow{\mathbf{C}_{w,2}}) \end{pmatrix}$.

Let $m = |\overrightarrow{\mathbb{Z}}|$. Compute $\left(\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{Z}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1T}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_2}}}\right) \leftarrow$
$\left(\left(\mathtt{CM}_{\mathbb{Z}_p}(\overrightarrow{\mathbb{Z}}[i], \sigma_G)\right)_{i=0}^{m-1}, \left(\mathtt{CM}_1(\overrightarrow{\mathbb{G}_{1M}}[i], \sigma_G)\right)_{i=0}^{m-1}, \left(\mathtt{CM}_1(\overrightarrow{\mathbb{G}_{1T}}[i], \sigma_G)\right)_{i=0}^{1}, \left(\mathtt{CM}_2(\overrightarrow{\mathbb{G}_2}[i], \sigma_G)\right)_{i=0}^{1}\right)$.

Apply Fiat-Shamir transform: $r \leftarrow \mathsf{hash}\left(\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{Z}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1T}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_2}}}\right)$

Compute: $\left(\overrightarrow{Z_M}, \overrightarrow{\pi_M}, \overrightarrow{Z_T}, \overrightarrow{\pi_T}\right) \leftarrow$
$\begin{pmatrix} \left(\sum_{j=0}^{n-1} r^{2j}\, \overrightarrow{\mathbb{Z}}[i]_j \overrightarrow{\mathbb{G}_{1M}}[i]_j\right)_{i=0}^{m-1}, \left(\mathtt{MIPP}_u.\mathtt{Prove}(\overrightarrow{\mathbb{G}_{1M}}[i], \overrightarrow{\mathbb{Z}}[i], r, \sigma_G)\right)_{i=0}^{m-1}, \\ \left(\sum_{j=0}^{n-1} r^{2j} \overrightarrow{\mathbb{G}_{1T}}[i]_j \cdot \overrightarrow{\mathbb{G}_2}[i]_j\right)_{i=0}^{1}, \left(\mathtt{TIPP}.\mathtt{Prove}(\overrightarrow{\mathbb{G}_{1T}}[i], \overrightarrow{\mathbb{G}_2}[i], r, \sigma_G)\right)_{i=0}^{1} \end{pmatrix}$

Let $\mathbf{C}_{\overrightarrow{\mathbb{X}}} = \left(\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[9], \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[1]\right)$. Let $\pi_a = \left(c, r, \left(\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{Z}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1T}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_2}}}\right), \left(\overrightarrow{Z_M}, \overrightarrow{\pi_M}, \overrightarrow{Z_T}, \overrightarrow{\pi_T}\right)\right)$.

Return $(\mathbf{C}_{\overrightarrow{\mathbb{X}}}, \pi_a)$.

2 **Verify Aggregated ZK-Subset Proof:** $1/0 \leftarrow \mathtt{AggVer}\left(\mathbf{C}_{\overrightarrow{\mathbb{X}}}, \pi_a\right)$:

Parse $\overrightarrow{\mathbb{X}}$ as $(\mathbf{C}_{\overrightarrow{\mathbf{C}_p}}, \mathbf{C}_{\overrightarrow{\mathbf{C}_q}})$. Parse π_a as $\left(c, r, \left(\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{Z}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1T}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_2}}}\right), \left(\overrightarrow{Z_M}, \overrightarrow{\pi_M}, \overrightarrow{Z_T}, \overrightarrow{\pi_T}\right)\right)$. Let $m = |\overrightarrow{\mathbb{Z}}|$. Return 1 if and only if (C0)-(C7) pass.

(C0) $\mathbf{C}_{\overrightarrow{\mathbf{C}_p}} = \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[9] \quad \wedge \quad \mathbf{C}_{\overrightarrow{\mathbf{C}_q}} = \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[1]$.

(C1) $(\forall 4 \leq i, j \leq 10$: $\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{Z}}}}[i] = \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{Z}}}}[j]) \quad \wedge \quad \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1T}}}}[0] = \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[3]$
$\wedge \quad \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[1] = \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[4] = \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1T}}}}[1] \quad \wedge \quad \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[0] = \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[6]$.

(C2) $\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[2] = \mathtt{CM}_1\left(\overrightarrow{[s_2]_1}, \sigma_G\right) \quad \wedge \quad \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[3] = \mathtt{CM}_1\left(\overrightarrow{[1]_1}, \sigma_G\right) \quad \wedge \quad \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{Z}}}}[4] = \mathtt{CM}_{\mathbb{Z}_p}\left(\mathbf{1}, \sigma_G\right)$.

(C3) $\forall i \in [0, m)$: $\mathtt{MIPP}_u.\mathtt{Verify}\left(\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[i], \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{Z}}}}[i], \overrightarrow{Z_M}[i], \overrightarrow{\pi_M}[i], \sigma_G\right) = 1 \quad \wedge \ \forall i \in [0, 1]$:
$\mathtt{TIPP}.\mathtt{Verify}\left(\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1T}}}}[i], \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_2}}}[i], \overrightarrow{Z_T}[i], \overrightarrow{\pi_T}[i], \sigma_G\right) = 1$.

(C4) $\forall (i, j) \in \{(4, 5), (6, 7)\}$: $\overrightarrow{Z_M}[i] \cdot [\alpha]_2 = \overrightarrow{Z_M}[j] \cdot [1]_2$, and $\overrightarrow{Z_M}[6] \cdot [1]_2 = \overrightarrow{Z_T}[0]$.

(C5) $\overrightarrow{Z_M}[9] \cdot [1]_2 + \overrightarrow{Z_M}[8] \cdot [s_2]_2 = \overrightarrow{Z_T}[1]$.

(C6) $\sum_{i=0}^{3} \overrightarrow{Z_M}[i] = \overrightarrow{Z_M}[10] + c\overrightarrow{Z_M}[8]$.

(C7) $c = \mathsf{hash}\left(\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}[10]\right)$ and $r = \mathsf{hash}\left(\overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{Z}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1M}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_{1T}}}}, \overrightarrow{\mathbf{C}_{\overrightarrow{\mathbb{G}_2}}}\right)$.

Fig. 8. Aggregate Zk-Subset Proof