# Arguments of Knowledge in hidden order groups

Steve Thakur

**Abstract**

We study non-interactive arguments of knowledge (AoKs) in groups of hidden order. We provide protocols whereby a Prover can demonstrate certain relations between committed sets/multisets with succinct proofs. These proofs can be publicly verified against the constant-sized commitments to the sets/multisets. In particular, we provide AoKs for the disjointness of sets/multisets in cryptographic accumulators, with a view toward applications to verifiably outsourcing data storage and sharded stateless blockchains.

Recent work ([DGS20]) suggests that the hidden order groups need to be substantially larger in size that previously thought. Thus, in order to keep the communication complexity between the Prover and the the Verifier to a minimum, we have designed the protocols so that the proofs consist of a constant number of group elements, independent of the number of the committed sets/multisets rather than just independent of the sizes of these set/multisets. We build on the techniques from [BBF19] and [Wes18].

If the underlying group of hidden order is an imaginary quadratic class group or a genus three Jacobian, the argument systems are transparent (trustless). Furthermore, since all challenges are public coin, the protocols can be made non-interactive using the Fiat-Shamir heuristic.

## 1 Introduction

Finite abelian groups of hidden order have seen a surging interest within cryptography in the last few years. The adaptive root assumption in such groups yields a cryptographic accumulator which is universal and dynamic with batchable membership and non-membership proofs ([BBF19]). One of the best known verifiable delay functions is that constructed in [Wes18], which can be instantiated with any group of unknown order in which the adaptive root assumption holds. Such groups also form the basis for the transparent polynomial commitment constructed in [BFS19]. This is a polynomial commitment with logarithmic sized proofs and verification time and can be instantiated with any group of hidden order.

In this paper, we explore non-interactive arguments of knowledge in groups of hidden order. We provide protocols whereby a Prover who stores data in the form of sets/multisets can prove relationships between these sets/multisets with communication complexity independent of the size of this data. These proofs can be publicly verified against the constant-sized commitments held by the Verifier. Our primary goal is to provide protocols for proofs of disjointness of committed sets/multisets in crptographic accumulators. The primary use case for these AoKs is potential applications to sharded blockchains and to verifiable outsourcing of data.

As was the case with previously studied arguments of knowledge in hidden order groups ([BBF19], [CFGNK20] etc.), the proofs consist of elements of the group $\mathbb{G}$ and $\lambda$-bit integers, where $\lambda$ is the security parameter. The proofs are succinct in the sense that the proof sizes are independent of the size of the committed data sets/multisets and. Recent work ([DGS20]) suggests that the hidden order groups need to be substantially larger in size that previously

thought, in order to ensure the desired security level. Furthermore, the two known candidates for transparent hidden order groups - imaginary quadratic class groups and genus three Jacobians - are not as well studied as RSA groups when it comes to potential attacks against the adaptive root and strong-RSA assumptions. Hence, it is conceivable that these groups might need to be even larger than presently believed. Bearing this in mind and with a view toward keeping the communication complexity between the Prover and the the Verifier to a minimum, we have designed the protocols so that the proofs consist of a constant number of group elements, independent of the number of committed sets/multisets involved, rather than just independent of the sizes of these sets/multisets.

### 1.1 Candidates for hidden order groups

At the moment, there are only three known families of finite abelian groups of unknown order. We briefly discuss them here.

1. **RSA groups:** For distinct 1536-bit primes $p$, $q$, define $N := pq$. The group $(\mathbb{Z}/N\mathbb{Z})^*$ has order $\phi(N) = (p-1)(q-1)$ which can only be computed by factorizing $N$. The strong-RSA assumption is believed to hold in the RS group. However, the group does contain the element $-1 \pmod N$ of a known order 2. For the adaptive root assumption to hold, the group has to be replaced by its quotient group $(\mathbb{Z}/N\mathbb{Z})^*/\{\pm 1\}$ of order $\frac{(p-1)(q-1)}{2}$.

The RSA groups suffer from the need for a trusted setup. In practice, this is usually mitigated by a one-time secure multi-party computation. At the moment, a 3300-bit RSA modulus yields a security level of 128-bits.

2. **Class groups:** Computing the class group of a number field is a long-standing problem in algorithmic number theory. Hence, class groups are natural candidates for hidden order groups. At the moment, the only class groups that allow for efficient group operations are those of imaginary quadratic fields.

For a square-free integer $d > 0$, the field $\mathbb{Q}(\sqrt{-d})$ has a class group of size roughly $\sqrt{d}$. This group is believed to fulfill the strong-RSA assumption. Furthermore, if $d$ is a prime $\equiv 3 \pmod 4$, the 2-torsion group is trivial, which eliminates the possibility of known elements of order 2. Such a group is believed to fulfill the adaptive root assumption unless the integer $d$ lies within a certain thin set of integers.

A 6656-bit discriminant $d$ yields a security level of 128-bits at the moment. Unlike RSA groups, class groups allow for a transparent (trustless) setup. The downside is that for the same level of security, the group operations are roughly 10 times slower than modular multiplication.

3. **Jacobians:** Recently, the group of $\mathbb{F}_p$-valued points of the Jacobian of a genus three hyperelliptic curve over a prime field $\mathbb{F}_p$ has been proposed as a candidate ([DGS20]). While this idea needs more scrutiny, it seems promising because of the transparent setup, the smaller key sizes and the fact that the group operations are 28 times faster than those in class groups for the same level of security.

For an irreducible polynomial $f(X) \in \mathbb{Z}[X]$ of degree 7 with Galois group $\mathbf{S}_7$ and a prime $p$ such that $f(X) \pmod p$ is separable, the hyperelliptic curve $C : Y^2 = f(X)$ over $\mathbb{F}_p$ yields a Jacobian that is resistant to the known attacks. At the moment, such a genus three hyperelliptic Jacobian over a prime field $\mathbb{F}_p$ of bit-size 1100 allows for a security level of 128-bits. This group $\mathrm{Jac}(C)(\mathbb{F}_p)$ is roughly of size $p^3$.

Unfortunately, the adaptive root assumption fails in the group $\mathrm{Jac}(C)(\mathbb{F}_p)$. However, the group obtained by replacing it by an appropriate quotient group appears to satisfy the adaptive root assumption and the weaker assumptions such as the fractional root and low order assumptions.

# 2 Preliminaries

We first state some definitions and notations used in this paper.

**Notations:** We denote the security parameter by $\lambda$ and the set of all polynomial functions by $\mathtt{poly}(\lambda)$. A function $\epsilon(\lambda)$ is said to be *negligible* - denoted $\epsilon(\lambda) \in \mathtt{negl}(\lambda)$ - if it vanishes faster than the reciprocal of any polynomial. An algorithm $\mathcal{A}$ is said to be a probabilistic polynomial time (PPT) algorithm if it is modeled as a Turing machine that runs in time $\mathtt{poly}(\lambda)$. We denote by $y \leftarrow \mathcal{A}(x)$ the process of running $\mathcal{A}$ on input $x$ and assigning the output to $y$. For a set $S$, $\#$ or $|S|$ denote its cardinality and $x \xleftarrow{\$} S$ denotes selecting $x$ uniformly at random over $S$. For a positive integer $n$, we write $[n] := \{0, 1, \cdots, n-1\}$. $\mathtt{NextPrime}(n)$ denotes the smallest prime $\geq n$. For statements $\mathbf{A}$, $\mathbf{B}$ we say that $\mathbf{A}$ implies $\mathbf{B}$ with overwhelming probability (denoted by $\mathbf{A} \overset{\text{o.p.}}{\Longrightarrow} \mathbf{B}$) if

$$1 - \mathbf{Prob}(\mathbf{B} \mid \mathbf{A}) = \mathtt{negl}(\lambda).$$

## 2.1 Cryptographic assumptions

The cryptographic protocols make extensive use of groups of unknown order, i.e., groups for which the order cannot be computed efficiently. Concretely, we require groups for which two hardness assumptions hold. The Strong RSA Assumption [BP97] roughly states that it is hard to take arbitrary roots of random elements. Secondly, the much newer Adaptive Root Assumption [Wes19] is the dual to the Strong RSA Assumption and states that it is hard to take random roots of arbitrary group elements. Both of these assumptions hold in generic groups of unknown order [DK02, BBF19, DGS20].

**Assumption 2.1.** *We say that the* ***adaptive root assumption*** *holds for a group $\mathbb{G}$ if there is no efficient probabilistic polynomial time (PPT) adversary $(\mathcal{A}_0, \mathcal{A}_1)$ that succeeds in the following task. $\mathcal{A}_0$ outputs an element $w \in \mathbb{G}$ and some state. Then a random prime $\ell$ is chosen and $\mathcal{A}_1(\ell, \text{state})$ outputs $w^{1/\ell} \in \mathbb{G}$.*

**Assumption 2.2.** *For a set $\mathcal{S}$ of rational primes, we say $\mathbb{G}$ satisfies the $\mathcal{S}$-**strong RSA** assumption if given a random group element $g \in \mathbb{G}$ and a prime $\ell \notin \mathcal{S}$, no PPT algorithm $\mathcal{A}$ is able to compute (except with negligible probability) the $\ell$-th root of a chosen element $w \in \mathbb{G}$. When $\mathcal{S} = \emptyset$, it is called the **strong RSA** assumption.*

**Assumption 2.3.** *We say $\mathbb{G}$ satisfies the **low order assumption** if no PPT algorithm can generate (except with negligible probability) an element $a \in \mathbb{G} \setminus \{1\}$ and an integer $n < 2^{\mathrm{poly}(\lambda)}$ such that $a^n = 1$.*

**Assumption 2.4.** *For a set $\mathcal{S}$ of rational primes, we say $\mathbb{G}$ satisfies the $\mathcal{S}$-**fractional root assumption** if for a randomly generated element $g \in \mathbb{G}$, a PPT algorithm cannot output $h \in \mathbb{G}$ and $d_1, d_2 \in \mathbb{Z}$ such that*

$$g^{d_1} = h^{d_2} \ \wedge \ \gcd(d_1, d_2) = 1 \ \wedge \ d_2 \text{ has a prime divisor outside } \mathcal{S}$$

*except with negligible probability. When $\mathcal{S} = \emptyset$, it is called the **fractional root assumption**.*

Clearly, if $\mathcal{S}_0 \subseteq \mathcal{S}$, the $\mathcal{S}_0$-**fractional root assumption** implies the $\mathcal{S}$-**fractional root assumption**. For instance, class groups of imaginary quadratic fields are believed to fulfill the $\{2\}$-fractional root assumption although they do not fulfill the (stronger) fractional root assumption. This is because there is a well-known algorithm to compute square roots in imaginary quadratic class groups ([BS96]). The assumptions bear the following relations:

$$\{\text{Adaptive root assumption}\} \implies \{\text{Low order assumption}\} \implies \{\text{Fractional root assumption}\},$$

$$\{\text{Strong-RSA assumption}\} \implies \{\text{Fractional root assumption}\}.$$

We refer the reader to the appendix of [BBF19] for further details.

### 2.1.1 Generic group models for hidden order groups

We will use the generic group model for groups of unknown order as defined by [DK02] and [BBF19]. The group is parametrized by two integer public parameters $A, B$. The order of the group is sampled uniformly from the interval $[A, B]$. The group $\mathbb{G}$ is defined by a random injective function $\sigma : \mathbb{Z}_{|\mathbb{G}|} \longrightarrow \{0,1\}^n$ for some $n >> \log_2(|\mathbb{G}|)$. A generic group algorithm $\mathcal{A}$ is a probabilistic algorithm. Let $\mathcal{L}$ be a list that is initialized with the encodings given to $A$ as input. The algorithm can query two generic group oracles:

• $\mathcal{O}_1$ samples a random $r \in \mathbb{Z}_{\mathbb{G}}$ and returns $\sigma(r)$, which is appended to the list of encodings $\mathcal{L}$.

• When $\mathcal{L}$ has size $q$, the second oracle $\mathcal{O}_2(i, j, \pm)$ takes two indices $i, j \in \{1, \cdots, q\}$ and a sign bit and returns $\sigma(x_i \pm x_j)$ which is appended to $\mathcal{L}$.

## 2.2 Argument Systems

Interactive arguments are interactive proofs [GMR85] in which security holds only against a computationally bounded prover. In an interactive argument of knowledge for a relation $\mathcal{R}$, the prover convinces the verifier that it knows a witness $w$ for a statement $x$ such that $(x, w) \in R$. In this paper, the term *knowledge* means that the argument has *witness-extended emulation*.

## 2.3 Multiset notations and operations

We first recall/introduce a few basic definitions and notations concerning multisets. For a multiset $\mathcal{M}$, we denote by $\mathtt{Set}(\mathcal{M})$ the underlying set of $\mathcal{M}$. For any element $x$, we denote by $\text{mult}(\mathcal{M}, x)$ the multiplicity of $x$ in $\mathcal{M}$. Thus, $\mathcal{M} = \{\text{mult}(\mathcal{M}, x) \times x : \ x \in \mathtt{Set}(\mathcal{M})\}$. For brevity, we write

$$\Pi(\mathcal{M}) := \prod_{x \in \mathtt{Set}(\mathcal{M})} x^{\text{mult}(\mathcal{M}, x)}.$$

For two multisets $\mathcal{M}, \mathcal{N}$, we have the following operations:

- The sum $\mathcal{M} + \mathcal{N} := \{(\text{mult}(\mathcal{M}, x) + \text{mult}(\mathcal{N}, x)) \times x : \ x \in \mathtt{Set}(\mathcal{M}) \cup \mathtt{Set}(\mathcal{N})\}$
- The union $\mathcal{M} \cup \mathcal{N} := \{\max(\text{mult}(\mathcal{M}, x), \text{mult}(\mathcal{N}, x)) \times x : \ x \in \mathtt{Set}(\mathcal{M}) \cup \mathtt{Set}(\mathcal{N})\}$
- The intersection $\mathcal{M} \cap \mathcal{N} := \{\min(\text{mult}(\mathcal{M}, x), \ \text{mult}(\mathcal{N}, x)) \times x : \ x \in \mathtt{Set}(\mathcal{M}) \cup \mathtt{Set}(\mathcal{N})\}$
- The difference $\mathcal{M} \setminus \mathcal{N} := \{\min(\text{mult}(\mathcal{M}, x) - \text{mult}(\mathcal{N}, x), \ 0) \times x : \ x \in \mathtt{Set}(\mathcal{M}) \cup \mathtt{Set}(\mathcal{N})\}$.

The function $\Pi(\cdot)$ clearly has the following properties:

- $\Pi(\mathcal{M} + \mathcal{N}) = \Pi(\mathcal{M}) \cdot \Pi(\mathcal{N})$
- $\Pi(\mathcal{M} \cup \mathcal{N}) = \mathbf{lcm}(\Pi(\mathcal{M}), \Pi(\mathcal{M}))$
- $\Pi(\mathcal{M} \cap \mathcal{N}) = \mathbf{gcd}(\Pi(\mathcal{M}), \Pi(\mathcal{M}))$
- $\Pi(\mathcal{M} \setminus \mathcal{N}) = \Pi(\mathcal{M}) / \Pi(\mathcal{M} \cap \mathcal{N})$

For a multiset $\mathcal{M}$ represented by $\lambda$-bit primes and a hidden order group $\mathbb{G}$, a $\mathbb{G}$-*commitment to a multiset* $\mathcal{M}$ is a pair $(g, h) \in \mathbb{G}^2$ such that $g^{\Pi(\mathcal{M})} = h$. The hardness of the discrete logarithm problem implies that this commitment is *hiding* in the sense that no PPT algorithm can compute $\mathcal{M}$ from the pair $[g, h]$. The low order assumption implies that it is *binding* in the sense that no PPT algorithm can compute another multiset $\mathcal{M}'$ with the same commitment.

### 2.4 Cryptographic Accumulators

A cryptographic accumulator [Bd94] is a primitive that produces a short binding commitment to a set (or multiset) of elements together with short membership and/or non-membership proofs for any element in the set. These proofs can be publicly verified against the commitment. Broadly, there are three known types of accumulators at the moment:

- Merkle trees
- pairing-based (aka bilinear) accumulators
- accumulators based on groups of unknown order, which we study in this paper.

Let $\mathbb{G}$ be a group of hidden order and fix an element $g \in \mathbb{G}$. Let $\mathcal{M} = \{\mathfrak{m}_1 \times d_1, \cdots, \mathfrak{m}_m \times d_m\}$ be a multiset, where $\{d_1, \cdots, d_n\}$ is the underlyling set $\texttt{Set}(\mathcal{M})$ of $\mathcal{M}$ and $\mathfrak{m}_i$ is the multiplicity of $d_i$. Using an appropriate hashing algorithm, we may assume the elements $d_i$ are distinct $\lambda$-bit primes. The **accumulated digest** of $\mathcal{M}$ is given by

$$\mathbf{Acc}(\mathcal{M}) := g^{\Pi(\mathcal{M})},$$

where

$$\Pi(\mathcal{M}) = \prod_{x \in \texttt{Set}(\mathcal{M})} x^{\text{mult}(\mathcal{M}, x)}.$$

Let $\mathcal{M}_0$ be a multiset contained in $\mathcal{M}$, so that $\mathfrak{m}_{0,i} \leq \mathfrak{m}_i \; \forall \; i$. The element

$$w(\mathcal{M}_0) := g^{\prod_{x \in \texttt{Set}(\mathcal{M})} x^{\text{mult}(\mathcal{M}, x) - \text{mult}(\mathcal{M}_0, x)}} \in \mathbb{G}$$

is called *the membership witness* of $\mathcal{M}_0$. Given this element, a Verifier can verify membership of $\mathcal{M}_0$ in $\mathcal{M}$ by verifying the equation

$$w(\mathcal{M}_0)^{\Pi(\mathcal{M}_0)} \stackrel{?}{=} \mathbf{Acc}(\mathcal{D}).$$

When the multiset $\mathcal{M}_0$ is large, this verification can be sped up using Wesolowki's *Proof of Exponentiation* (`PoE`) protocol ([Wes18]).

Shamir's trick allows for aggregation of membership witnesses in accumulators based on hidden order groups. This is not possible with Merkle trees, which is the primary reason other families of accumulators have been explored as authentication data structures for stateless blockchains. With bilinear accumulators, aggregation of membership witnesses has a linear runtime complexity, which is impractical for most use cases. Thus, accumulators based on hidden order groups have a major advantage in this regard.

These accumulators also allow for non-membership proofs [LLX07]. In [BBF19], the authors used a non-interactive argument of knowledge to compress batched non-membership proofs into constant-sized proofs, i.e. independent of the number of elements involved. This yields the first known Vector Commitment with constant-sized openings as well as constant-sized public parameters.

**Hashing the data to primes:** The security of cryptographic accumulators and vector commitments hinges on the assumption that for disjoint data sets $\mathcal{D}, \mathcal{E}$, the integers $\Pi(\mathcal{D}), \Pi(\mathcal{E})$

are relatively prime. The easiest way to ensure this is to map the data elements to distinct $\lambda$-bit primes. This is usually done by hashing the data to $\lambda$-bit integers and subjecting the output to a probabilistic primality test such as the Miller-Rabin test. The prime number theorem states that the number of primes less than $n$ is $\mathbf{O}(\frac{n}{\log(n)})$ and hence, implies that the expected runtime for finding a prime is $\mathbf{O}(\lambda)$.

Dirichlet's theorem on primes in arithmetic progressions combined with the prime number theorem implies that for relatively prime integers $k, r$ and an integer $n$, the number of primes less than $n$ that are $\equiv r \pmod{k}$ is roughly $\frac{n}{\log(n)\phi(k)}$. Thus, we can modify the hashing algorithm so that for any element inserted into the accumulator, the prime reveals the position in which it was inserted. We proceed as follows.

1. Fix a prime $p$ of size $\frac{\lambda}{2}$.

2. For a string inserted in position $i$, we map the string to the first prime of size $\lambda$ which is $\equiv i \pmod{p}$. This (pseudo-)prime is obtained by subjecting the integers $p\mathbb{Z}+i$ to the probabilistic Miller-Rabin test.

The number of such primes is roughly $\frac{2^\lambda}{\lambda(p-1)}$ and hence, the expected runtime is $\mathbf{O}(\lambda)$.

### 2.5 Aggregating and disaggregating membership witnesses

**Shamir's trick:** Given elements $a_1, a_2, A \in \mathbb{G}$ and integers $d_1, d_2$ such that $a_1^{d_1} = a_2^{d_2} = A$, Shamir's trick allows us to compute the $\mathbf{lcm}(d_1, d_2)$-th root of $A$ as follows.

1. Compute integers $e_1, e_2$ such that
$$e_1 d_1 + e_2 d_2 = \mathbf{gcd}(d_1, d_2).$$

2. Set $a_{1,2} := a_1^{e_2} a_2^{e_1}$.

Then
$$a_{1,2}^{d_1 d_2} = A^{d_2 e_2 + d_1 e_1} = A^{\mathbf{gcd}(d_1, d_2)}$$
and hence,
$$a_{1,2}^{\mathbf{lcm}(d_1, d_2)} = A.$$
More generally, given elements $a_1, \cdots, a_n$ such that
$$a_1^{d_1} = \cdots = a_n^{d_n} = A,$$
we can use Shamir's trick repeatedly to compute an element $a \in \mathbb{G}$ such that
$$a^{\mathbf{lcm}(d_1, \cdots, d_n)} = A.$$
The runtime for this algorithm is $\mathbf{O}(n \log(n))$.

The most important special case is when $A$ is the accumulated digest $g^{\Pi(\mathcal{M})}$ for a multiset $\mathcal{M}$ and $w_1, \cdots, w_n$ are membership witnesses for multisets $\mathcal{M}_1, \cdots, \mathcal{M}_n \subseteq \mathcal{M}$. Shamir's trick allows us to compute a membership witness for the union $\bigcup_{i=1}^{n} \mathcal{M}_i$.

**The RootFactor Algorithm:** Given elements $a, A \in \mathbb{G}$ and integers $d_1, \cdots, d_n$, $D$ such that
$$D = \prod_{i=1}^{n} d_i \ , \ a^D = A,$$
the RootFactor algorithm ([BBF19], [STY01]) allows us to compute elements $a_1, \cdots, a_n$ such that
$$a_1^{d_1} = \cdots = a_n^{d_n} = A$$
in runtime $\mathbf{O}(\log(D) \log(\log(D)))$. naïvely, this would take runtime $\mathbf{O}(\log^2(D))$, which would be impractical for many applications.

### 2.6 $\mathbb{Z}_{(\lambda)}$-integers and the equivalence relation ($\equiv_\lambda$)

**Definition 2.1.** *For elements $a, b \in \mathbb{G}$ and a rational $\alpha \in \mathbb{Q}$, we say $a^\alpha = b$ with respect to a Prover $\mathcal{P}$ if $\mathcal{P}$ verifiably possesses integers $d_1, d_2 \in \mathbb{Z}$ such that $\alpha = \frac{d_1}{d_2}$ and $a^{d_1} = b^{d_2}$.*

Note that if there exists an element $a \in \mathbb{G}$ and distinct rationals $\frac{d_1}{d_2}, \frac{d_3}{d_4}$ ($d_i \in \mathbb{Z}$) such that

$$a^{\frac{d_1}{d_2}} = a^{\frac{d_3}{d_4}},$$

then $a^{d_1 d_4 - d_2 d_3} = 1$ and $d_1 d_4 - d_2 d_3 \neq 0$. So the low order assumption implies that a PPT algorithm cannot generate such a tuple $(a, d_1, d_2, d_3, d_4)$ except with negligible probability. Furthermore, by Shamir's trick, the condition is equivalent to the Prover $\mathcal{P}$ being able to compute an element $a_0 \in \mathbb{G}$ and co-prime integers $d_1, d_2$ such that

$$\alpha = \frac{d_1}{d_2} \ , \ a_0^{d_2} = a \ , \ a_0^{d_1} = b \ ,$$

**Localization at a set of primes:** For a set $\mathcal{S}$ of rational primes, we denote by $\mathbb{Z}_{\mathcal{S}}$ the localization

$$\mathbb{Z}_{\mathcal{S}} := \left\{ \frac{a}{\prod\limits_{p \in \mathcal{S}} p^{e_p}} \ : \ a \in \mathbb{Z},\ e_p \in \mathbb{Z},\ e_p = 0 \text{ for all but finitely many } p \right\}$$

of $\mathbb{Z}$ at all primes in $\mathcal{S}$. This is a principal ideal domain whose prime ideals are those generated by rational primes outside the set $\mathcal{S}$. The group of units of $\mathbb{Z}_{\mathcal{S}}$ is given by

$$\mathbb{Z}_{\mathcal{S}}^\times := \left\{ \pm \prod_{p \in \mathcal{S}} p^{e_p} \ : e_p \in \mathbb{Z},\ e_p = 0 \text{ for all but finitely many } p \right\}.$$

For an element $\alpha \in \mathbb{Z}_{\mathcal{S}}$, we may uniquely write $\alpha$ as

$$\alpha = \frac{\alpha_1}{\alpha_2} \text{ where } \alpha_1, \alpha_2 \in \mathbb{Z} \ , \ \alpha_2 > 0 \ , \ \mathbf{gcd}(\alpha_1, \alpha_2) = 1.$$

Similarly, for two elements, $\alpha, \beta$, we write

$$\alpha = \frac{\alpha_1}{\alpha_2} \ , \ \beta = \frac{\beta_1}{\beta_2}$$

and define $\mathbf{gcd}_{\mathcal{S}}(\alpha, \beta) := \mathbf{gcd}(\alpha_1, \beta_1)$. Thus, $\mathbf{gcd}_{\mathcal{S}}(\alpha, \beta)$ is the unique non-negative integer that has no prime divisors in $\mathcal{S}$ and generates the ideal $\mathbb{Z}_{\mathcal{S}}\alpha + \mathbb{Z}_{\mathcal{S}}\beta \subseteq \mathbb{Z}_{\mathcal{S}}$.

**Note:** The localization $\mathbb{Z}_{\mathcal{S}}$ is not to be confused with the non-archimedean *completion* of the localization, especially when $\mathcal{S}$ consists of a single prime.

**Definition 2.2.** *An integer is said to be $\lambda$-**smooth** is all of its prime divisors are $\leq 2^{\lambda-1}$. An integer is said to be $\lambda$-**rough** is all of its prime divisors are $> 2^{\lambda-1}$. We say a set/multiset $\mathcal{M}$ of primes is $\lambda$-rough if the integer $\Pi(\mathcal{M})$ is $\lambda$-rough.*

Clearly, $\mathcal{M}$ being $\lambda$-rough is equivalent to each prime of $\mathcal{M}$ being larger than $2^{\lambda-1}$. The properties of $\lambda$-smoothness and $\lambda$-roughness are clearly preserved under products, greatest common divisors and least common multiples. Furthermore, any positive integer $n$ is uniquely expressible as a product $n_{\lambda,s} n_{\lambda,r}$ of a $\lambda$-smooth integer $n_{\lambda,s} \geq 0$ and a $\lambda$-rough integer $n_{\lambda,r} \geq 0$.

**Definition 2.3.** *For a security parameter $\lambda$, we denote by $\mathbb{Z}_{(\lambda)}$ the integral domain obtained by localizing $\mathbb{Z}$ away from all primes $\geq 2^{\lambda-1}$.*

Thus,

$$\mathbb{Z}_{(\lambda)} = \left\{ \frac{\alpha}{\beta} : \ \alpha, \beta \in \mathbb{Z}, \ \mathbf{gcd}(\alpha, \beta) = 1, \ \beta \text{ is } \lambda\text{-smooth} \right\}.$$

Note that $\mathbb{Z}_{(\lambda)}$ inherits the structure of a principal ideal domain. The group of units of $\mathbb{Z}_{(\lambda)}$ is given by

$$\mathbb{Z}_{(\lambda)}^{\times} := \left\{ \frac{\alpha}{\beta} : \ \alpha, \beta \in \mathbb{Z}, \ \mathbf{gcd}(\alpha, \beta) = 1, \ \alpha, \beta \text{ are } \lambda\text{-smooth} \right\}.$$

The prime ideals of $\mathbb{Z}_{(\lambda)}$ are the principal ideals generated by rational primes larger than $2^{\lambda-1}$.

**Definition 2.4.** *For $\mathbb{Z}_{(\lambda)}$-integers $d_1, d_2$ we say $d_1 \equiv_\lambda d_2$ if $\frac{d_1}{d_2}$ is a unit in $\mathbb{Z}_{(\lambda)}$.*

This is clearly a homomorphic equivalence relation.

**Definition 2.5.** *For $\mathbb{Z}_{(\lambda)}$-integers $d_1, d_2$, we denote by $\mathbf{gcd}_\lambda(d_1, d_2)$ the largest $\lambda$-rough integer that divides both $d_1$ and $d_2$ in the principal ideal domain $\mathbb{Z}_{(\lambda)}$. Similarly, we denote by $\mathbf{lcm}_\lambda(d_1, d_2)$ the smallest $\lambda$-rough integer divisible by $d_1$ and $d_2$ in $\mathbb{Z}_{(\lambda)}$.*

**Definition 2.6.** *For elements $a, b$ in a hidden order group $\mathbb{G}$, we say*

$$a \equiv_\lambda b$$

*with respect to a Prover $\mathcal{P}$ if $\mathcal{P}$ verifiably possesses relatively prime $\lambda$-smooth integers $d_1, d_2$ such that $a^{d_1} = b^{d_2}$.*

Because of Shamir's trick, this is equivalent to $\mathcal{P}$ being able to generate an element $a_0 \in \mathbb{G}$ and relatively prime $\lambda$-smooth integers $d_1, d_2$ such that

$$a_0^{d_2} = a, \ a_0^{d_1} = b.$$

It is easy to see that this an equivalence relation.

**Proposition 2.1.** *The relation $(\equiv_\lambda)$ is an equivalence relation.*

*Proof.* Since the reflexivity and the symmetry are obvious, it suffices to show that the relation is transitive.
(Transitivity): Suppose $a \equiv_\lambda b$ and $b \equiv_\lambda c$ for elements $a, b, c \in \mathbb{G}$. Then $\mathcal{P}$ possesses $\lambda$-smooth integers $d_1, d_2, d_3, d_4$ such that

$$a^{d_1} = b^{d_2} \ , \ b^{d_3} = c^{d_4} \ , \ \mathbf{gcd}(d_1, d_2) = \mathbf{gcd}(d_3, d_4) = 1.$$

Now,

$$a^{d_1 d_3} = b^{d_2 d_3} = c^{d_2 d_4}$$

and clearly, the integers $d_1 d_3, d_2 d_4$ are $\lambda$-smooth. Set $d := \mathbf{gcd}(d_1 d_3, d_2 d_4)$ and $e_1 := d_1 d_3 / d, \ e_2 := d_2 d_4 / d$. Then $e_1, e_2$ are co-prime and $\lambda$-smooth and

$$a^{e_1} = c^{e_2}.$$

Thus, $a \equiv_\lambda c$. □

For elements $a, b \in \mathbb{G}$ the following are equivalent:

1. $a^d \equiv_\lambda b$ for some integer $d$.
2. $a^d \equiv_\lambda b$ for some $\lambda$-rough integer $d$.
3. $b = a^{d_1}$ for some $\mathbb{Z}_{(\lambda)}$-integer $d_1$.

Furthermore, if a PPT algorithm is able to output an element $a \in \mathbb{G}$ and integers $d_1, d_2$ such that $a^{d_1} \equiv_\lambda a^{d_2}$, then with overwhelming probability, $\frac{d_1}{d_2} \in \mathbb{Z}^\times_{(\lambda)}$. In particular, no PPT algorithm can output an element $a \in \mathbb{G}$ and distinct $\lambda$-rough integers $d_1, d_2$ such that $a^{d_1} \equiv_\lambda a^{d_2}$.

We note, however, that the relation $(\equiv_\lambda)$ is not homomorphic, meaning that $a_1 \equiv_\lambda a_2$, $b_1 \equiv_\lambda b_2$ does not imply $a_1 a_2 \equiv_\lambda b_1 b_2$. But the relation is *partly* homomorphic in the sense that for any integer $d$,

$$a \equiv_\lambda b \iff a^d \equiv_\lambda b^d..$$

**Non-membership proofs in accumulators:** The best-known application of the knowledge of exponent protocol is constant-sized batched non-membership proofs in accumulators ([BBF19]). We discuss the implications of replacing equality of $\mathbb{G}$-elements with the equivalence relation $\equiv_\lambda$ in this regard.

Let $g \in \mathbb{G}$ denote the genesis state of the accumulator, $\mathcal{D}$ the inserted data set and $A = g^{\Pi(\mathcal{D})}$ the accumulated digest. Given a data set $\mathcal{D}_0$ disjoint with $\mathcal{D}$, the Prover demonstrates non-membership for all elements of $\mathcal{D}_0$ by sending the following to the Verifier:

- Elements $w, A_1 \in \mathbb{G}$ such that $w^{\Pi(\mathcal{D}_0)} A_1 = g$.
- A non-interactive proof for $\texttt{PoKE}[A,\ A_1]$.

Suppose, instead of $\texttt{PoKE}[A,\ A_1]$, the Prover proves the weaker statement that he possesses an integer $k$ such that $A^k \equiv_\lambda A_1$. By definition, there exist an integer $k$ and a $\lambda$-smooth integer $e$ such that $\mathbf{gcd}(k, e) = 1$ and $A^k = A_1^e$. Write $w = g^x$. Then

$$x\Pi(\mathcal{D}_0) + \frac{k\Pi(\mathcal{D})}{e} = 1$$

and hence,

$$ex\Pi(\mathcal{D}_0) + k\Pi(\mathcal{D}) = e.$$

Thus, $\mathbf{gcd}(\Pi(\mathcal{D}_0), \Pi(\mathcal{D}))$ divides $e$ which is a $\lambda$-smooth integer. Since each element of $\mathcal{D}$ is a $\lambda$-bit prime, it follows that $\mathcal{D} \cap \mathcal{D}_0 = \emptyset$, despite $\frac{k}{e}$ possibly not being an integer. Thus, the equivalence relation $\equiv_\lambda$ is compatible with the nonmembership protocol of [BBF19].

### 2.7 Some preliminary lemmas

We will need the next two lemmas repeatedly in the subsequent protocols. We briefly explain the motivation for these lemmas here and provide further details in the next section. As before, for a set $\mathcal{S}$ of rational primes, we denote by $\mathbb{Z}_{\mathcal{S}}$ the localization

$$\mathbb{Z}_{\mathcal{S}} := \left\{ \frac{a}{\prod\limits_{p \in \mathcal{S}} p^{e_p}} \ :\ a \in \mathbb{Z},\ e_p \in \mathbb{Z},\ e_p = 0 \text{ for all but finitely many } p \right\}$$

of $\mathbb{Z}$ at all primes in $\mathcal{S}$. This is a principal ideal domain and, in particular, is integrally closed.

Consider a setting where a Verifier possesses commitments

$$a_i = a^{d_i}$$

to $\mathbb{Z}_{\mathcal{S}}$-elements $d_i$ where $a \in \mathbb{G}$ is the common base and $\mathcal{S}$ is a set of rational primes. Suppose the Prover - who stores these integers - needs to demonstrate that the $d_i$ are integers rather than merely rationals. A straightforward way to do this would be for the Prover to send the elements $g^{d_i}$ along with proofs that the discrete logarithm between $g, g^{d_i}$ is the same as that between $a, a_i$. The $\mathcal{S}$-fractional root assumption would then imply that the $d_i$ are $\mathcal{S}$-integers.

But such a proof would entail $\mathbf{O}(n)$ elements of $\mathbb{G}$. Since the group elements are rather large, we would prefer to send a proof that contains a constant number of group elements. To this end, the Prover can instead demonstrate that for a randomly generated integer $\gamma$, the rational $\sum_{i=1}^{n} d_i^k \gamma^i$ is an element of $\mathbb{Z}_{\mathcal{S}}$ for some $k \geq n\lambda$. The next two lemmas show that this implies that all the $d_i$ are elements of $\mathbb{Z}_{\mathcal{S}}$.

In a setting where we are dealing with accumulators and the accumulation of primes $< 2^{\lambda-1}$ is disallowed, it suffices for the Prover to show that the rational $\sum_{i=1}^{n} d_i \gamma^i$ is an element of $\mathbb{Z}_{\mathcal{S}}$.

**Lemma 2.2.** *Let $p$ be a prime and let $f(X)$ be a monic univariate degree $n$ polynomial in $\mathbb{Z}[X]$. For a randomly generated integer $\gamma$, the probability that $f(\gamma) \equiv 0 \pmod{p^{n\lambda}}$ is* $\texttt{negl}(\lambda)$.

*Proof.* Let $F$ be a splitting field of $f(X)$, $p\mathcal{O}_F$ its ring of integers and let

$$f(X) = \prod_{i=1}^{n} (X - \alpha_i)$$

be the factorization of $f(X)$ over $F$. Let $\mathfrak{p}_1, \cdots, \mathfrak{p}_g$ be the distinct primes of $F$ lying over $p$. Since the extension $F/\mathbb{Q}$ is Galois, we have

$$p\mathcal{O}_F = \prod_{i=1}^{g} \mathfrak{p}_i^e = \bigcap_{i=1}^{g} \mathfrak{p}_i^e$$

where $e \geq 1$ is the ramification degree and the Galois group $\mathrm{Gal}(F/\mathbb{Q})$ acts transitively on the set $\{\mathfrak{p}_1, \cdots, \mathfrak{p}_g\}$.

We note that for any integer $k \geq 1$, $\mathfrak{p}_1^{ek} \cap \mathbb{Z} = p^k\mathbb{Z}$. The inclusion $p^k\mathbb{Z} \subseteq \mathfrak{p}_1^{ek} \cap \mathbb{Z}$ is obvious. For the reverse inclusion, let $x \in \mathfrak{p}_1^{ek} \cap \mathbb{Z}$. For any index $i$, there exists an automorphism $\sigma_i \in \mathrm{Gal}(F/\mathbb{Q})$ such that $\sigma_i(\mathfrak{p}_1) = \mathfrak{p}_i$. So $x = \sigma(x) \in \mathfrak{p}_i^e$. Hence, $x \in \bigcap_{i=1}^{g} \mathfrak{p}_i^{ek} = p^k\mathbb{Z}$.

For any two integers $x_1, x_2 \in \mathbb{Z}$, we have

$$x_1 - x_2 \in \mathfrak{p}_1^{e\lambda} \iff x_1 - x_2 \in \mathfrak{p}_1^{e\lambda} \cap \mathbb{Z} = p^\lambda\mathbb{Z}.$$

Hence, the set

$$\mathbf{S}_\lambda := \{x + \mathfrak{p}_1^{e\lambda} : \ x \in \mathbb{Z}\} \subseteq \mathcal{O}_F/\mathfrak{p}_1^{e\lambda}$$

has cardinality $p^\lambda$. Now, for any integer $\gamma$,

$$f(\gamma) \equiv 0 \pmod{p^{n\lambda}} \iff f(\gamma) \equiv 0 \pmod{\mathfrak{p}_1^{en\lambda}} \implies \gamma \equiv \alpha_i \pmod{\mathfrak{p}_1^{e\lambda}} \text{ for at least one index } i.$$

Since $\gamma$ is randomly generated, $\gamma \pmod{\mathfrak{p}_1^{e\lambda}}$ is randomly and uniformly distributed over the set $\mathbf{S}_\lambda$. Hence,

$$\mathbf{Prob}\big(f(\gamma) \equiv 0 \pmod{p^{n\lambda}}\big) \leq \frac{n}{p^\lambda} = \texttt{negl}(\lambda),$$

which completes the proof. □

**Lemma 2.3. 1**. *For rationals $d_1, \cdots, d_n \in \mathbb{Q}$ and a randomly generated $\lambda$-bit integer $\gamma$, if*

$$\sum_{i=1}^{n} d_i \gamma^i \ \in \ \mathbb{Z}_{(\lambda)},$$

*with overwhelming probability, $(d_1, \cdots, d_n) \in \mathbb{Z}_{(\lambda)}^n$.*

**2**. *Let $k$ be any integer $\geq n\lambda$. Let $\mathcal{S}$ be a set of rational primes and let $\mathbb{Z}_{\mathcal{S}}$ be the localization of $\mathbb{Z}$ at all primes in $S$. For rationals $d_1, \cdots, d_n \in \mathbb{Q}$ and a randomly generated $\lambda$-bit integer $\gamma$, if*

$$\sum_{i=1}^{n} d_i^k \gamma^i \ \in \ \mathbb{Z}_{\mathcal{S}},$$

*with overwhelming probability, $(d_1, \cdots, d_n) \in \mathbb{Z}_{\mathcal{S}}^n$.*

*Proof.* **1**. Let $D$ be the least common denominator for $d_1, \cdots, d_n$ and write

$$d_i = \frac{c_i}{D} \ , \ c_i \in \mathbb{Z} \quad (i = 1, \cdots, n).$$

Suppose, by way of contradiction that $(d_1, \cdots, d_n) \notin \mathbb{Z}_{(\lambda)}^n$. Then $D$ is divisible by some prime $p > 2^{\lambda - 1}$ and

$$\sum_{i=1}^{n} c_i \gamma^i \equiv 0 \pmod{p}.$$

Now, the polynomial $\sum\limits_{i=1}^{n} c_i X^i \in \mathbb{F}_p[X]$ has at most $n$ distinct zeros in $\mathbb{F}_p$ and since $\gamma$ is uniformly distributed modulo $p$, the probability of this occurring is $\mathtt{negl}(\lambda)$, a contradiction.

**2**. Let $D$ be the least common denominator for $d_1, \cdots, d_n$ and write

$$d_i = \frac{c_i}{D} \ \ (c_i \in \mathbb{Z}) \quad \text{for } i = 1, \cdots, n.$$

Suppose, by way of contradiction that $(d_1^k, \cdots, d_n^k) \notin \mathbb{Z}_{\mathcal{S}}^n$ and let $p \notin \mathcal{S}$ be a prime dividing $D$. Then

$$\sum_{i=1}^{n} c_i^k \gamma^i \equiv 0 \pmod{p^k}.$$

Now, the polynomial $f(X) := \sum\limits_{i=1}^{n} c_i^k X^i$ has degree $n$ and by the preceding lemma,

$$\mathbf{Prob}\big(f(\gamma) \equiv 0 \pmod{p^k}\big) = \mathtt{negl}(\lambda).$$

Thus, with overwhelming probability, the rationals $d_i^k$ lie in $\mathbb{Z}_{\mathcal{S}}$. Since the integral domain $\mathbb{Z}_{\mathcal{S}}$ is integrally closed, this implies that the $d_i$ lie in $\mathbb{Z}_{\mathcal{S}}$. □

In particular,

$$\mathbf{Prob}\big((d_1, \cdots, d_n) \notin \mathbb{Z}_{(\lambda)}^n \ \Big| \ \sum_{i=1}^{n} d_i \gamma^i \ \in \ \mathbb{Z}\big) = \mathtt{negl}(\lambda).$$

Similarly,

$$\mathbf{Prob}\big((d_1, \cdots, d_n) \notin \mathbb{Z}^n \ \Big| \ \sum_{i=1}^{n} d_i^k \gamma^i \ \in \ \mathbb{Z}\big) = \mathtt{negl}(\lambda) \quad \text{for any } k \geq n\lambda.$$

### 2.7.1 Representations of group elements

Following the terminology of [BBF19], the event DLOG refers to a generic PPT algorithm $\mathcal{A}$ generating random elements $g_1, \cdots g_n$ by making queries to the oracle $\mathcal{O}_1$ and generating integers $x_1, \cdots, x_n$ such that

$$\prod_{i=1}^{n} g_i^{x_i} = 1.$$

As shown in Lemma 4 of the Appendix in [BBF19], the probability of this event is negligible. The next lemma follows directly from the techniques in the appendix of [BBF19]. We provide a proof for the reader's convenience.

**Lemma 2.4.** *Let $a, b$ be elements of a generic hidden order group $\mathbb{G}$. Let $\mathcal{A}$ be a generic PPT algorithm that succeeds at the following task:*

*- The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\ell$.*

*- $\mathcal{A}$ generates an element $Q \in \mathbb{G}$ and an integer $r \in [\ell]$ such that $Q^\ell a^r = b$.*

*Then with overwhelming probability, $\mathcal{A}$ can generate integers $d_1, d_2$ such that*

$$a^{d_1} = b^{d_2} \ \wedge \ d_1 \equiv r d_2 \pmod{\ell}.$$

*Proof.* By the element representation lemma of [Sho97] (see the Appendix of [BBF19] for additional details), every element $\mathcal{A}$ obtains in response to an $\mathcal{O}_2$ query is expressible as a linear combination $\prod_{i=1}^{k} g_i^{x_i}$ where the $g_1, \cdots, g_k$ are outputs of $\mathcal{O}_1$ and the $x_i$ are integers. Let

$$a = \prod_{i=1}^{k} g_i^{\alpha_i} \ , \ b = \prod_{i=1}^{k} g_i^{\beta_i}, \quad (\alpha_i, \beta_i \in \mathbb{Z}).$$

By lemma 4 of ([BBF19], Appendix), except with negligible probability, $\mathcal{A}$ cannot generate integers $\gamma_1, \cdots \gamma_k$ (with at least one of them non-zero) such that

$$\prod_{i=1}^{k} g_i^{\gamma_i} = 1.$$

Now, for a randomly generated $\lambda$-bit prime $\ell$, $\mathcal{A}$ is able to output $Q \in \mathbb{G}$, $r \in [\ell]$ such that $Q^\ell a^r = b$. Hence,

$$\frac{\beta_1}{\alpha_1} \equiv \cdots \equiv \frac{\beta_k}{\alpha_k} \equiv r \pmod{l}.$$

Since $\ell$ was randomly generated, this implies that with overwhelming probability,

$$\frac{\beta_1}{\alpha_1} = \cdots = \frac{\beta_k}{\alpha_k}.$$

Thus, $a^{\beta_1} = b^{\alpha_1}$. □

**Lemma 2.5.** *Let $a, \ b_1, \cdots, b_n$ be elements of a generic hidden order group $\mathbb{G}$. Let $\mathcal{A}$ be a generic PPT algorithm that succeeds at the following task:*

*- The Fiat-Shamir heuristic generates a $\lambda$-bit challenge $\gamma$ and a $\lambda$-bit prime $\ell$.*

*- $\mathcal{A}$ generates an element $Q \in \mathbb{G}$ and an integer $r \in [\ell]$ such that $Q^\ell a^r = \prod_{i=1}^{n} b_i^{\gamma^i}$.*

*Then with overwhelming probability, $\mathcal{A}$ can generate rationals $d_i$ such that $a^{d_i} = b_i \ \forall \ i$.*

*Proof.* As before, the algorithm $\mathcal{A}$ extracts integers $\alpha_j$ $(j = 1, \cdots, k)$, $\beta_{i,j}$ $(i = 1, \cdots, n\ ,\ j = 1, \cdots, k)$ such that

$$a = \prod_{j=1}^{k} g_j^{\alpha_j} \ \ , \ \ b_i = \prod_{j=1}^{k} g_j^{\beta_{i,j}}, \quad (\alpha_i, \beta_{i,j} \in \mathbb{Z})$$

where $g_1, \cdots, g_k$ are the $\mathbb{G}$-elements obtained from queries to the oracle $\mathcal{O}_1$. Now, for a $\lambda$-bit challenge $\gamma$, $\mathcal{A}$ generates $Q \in \mathbb{G}$, $r \in [\ell]$ such that $Q^\ell a^r = \prod_{i=1}^{n} b_i^{\gamma^i}$. By the preceding lemma, $\mathcal{A}$ can generate a rational $\widetilde{d}$ such that $a^{\widetilde{d}} = \prod_{i=1}^{n} b_i^{\gamma^i}$. Since the event DLOG occurs with at most negligible probability, it follows that $\widetilde{d} = \sum_{i=1}^{n} \frac{\beta_{i,j}}{\alpha_j} \gamma^i$ for $j = 1, \cdots, k$. Since $\gamma$ is randomly generated, it follows that with overwhelming probability, we have

$$\frac{\beta_{i,1}}{\alpha_1} = \cdots = \frac{\beta_{i,n}}{\alpha_n}.$$

Setting $d_i := \frac{\beta_{i,1}}{\alpha_1}$, we have $a^{d_i} = b_i \ \forall\ i$. □

# 3 Arguments of Knowledge

In this section, we discuss arguments of knowledge of exponents in hidden order groups and of relations between these exponents. This is equivalent to the knowledge of relations between the committed sets or multisets. The proofs can be publicly verified against the succinct commitments held by the Verifier.

## 3.1 Preliminaries

We briefly review the protocol `PoKE` (from [BBF19]) which we will need repeatedly in this paper.

**Protocol 3.1.** *Proof of Knowledge of the Exponent* (`PoKE`)

**Parameters:** $\mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda)$, $g \in \mathbb{G}$

**Inputs:** $u, w \in \mathbb{G}$.

**Claim:** The Prover posseses an integer $x$ such that $u^x = w$.

1. The Prover $\mathcal{P}$ computes $z := g^x$ and sends it to the Verifier $\mathcal{V}$.
2. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\ell$.
3. $\mathcal{P}$ computes the integers $q, r$ such that

$$x = q\ell + r, \ \ r \in [\ell].$$

4. $\mathcal{P}$ computes $Q := u^q$, $Q' = g^q$ and sends $(Q, Q', g^x, r)$ to $\mathcal{V}$.
5. $\mathcal{V}$ accepts if and only if

$$r \in [\ell], \ \ Q^\ell u^r w, \ \ Q'^\ell g^r = z.$$ □

The part where $\mathcal{P}$ computes $g^x$ and sends it to $\mathcal{V}$ *before* receiving the challenge $\ell$ is necessary for the security of the protocol. Without this step, a malicious Prover could convince the Verifier that $x$ is an integer, which might not necessarily be the case.

Clearly, the relation *Knowledge of the Exponent* is transitive in the sense that for elements $a_1, a_2, a_3 \in \mathbb{G}$, if a prover $\mathcal{P}$ possesses integers $d_1, d_2$ such that $a_1^{d_1} = a_2$ , $a_2^{d_2} = a_3$, then he possesses the integer $d_1 d_2$ which fulfills the equation $a_1^{d_1 d_2} = a_3$. Henceforth, we denote the proof of knowledge of the discrete logarithm between $a, b \in \mathbb{G}$ by $\texttt{PoKE}[a,\ b]$.

The knowledge of exponents can easily be aggregated when they share a common base. Given elements

$$a \in \mathbb{G}\ ,\ (b_1, \cdots, b_n) \in \mathbb{G}^n,$$

and a radomly generate integer $\gamma$, if the Prover possesses a rational $\widetilde{d}$ such that

$$a^{\widetilde{d}} = \prod_{i=1}^{n} b_i^{\gamma^i},$$

then the Prover, with overwhelming probability, possesses rationals $d_i$ such that $a^{d_i} = b_i\ \forall\ i$. In the special case where $a = g$, a randomly generated element of the group $\mathbb{G}$, the fractional root assumption implies that the $d_i$ must be integers. In the more general case, we use lemma 2.3 to show that the $d_i$ are integers rather than merely rationals.

We start out with a fairly simple protocol. We show how a Prover could probabilistically demonstrate that two discrete logarithms are equal, with a constant-sized proof. In other words, the protocol allows a Prover to show that two pairs $[a_1, b_1]$, $[a_2, b_2]$ of $\mathbb{G}$-elements are commitments to the same set/multiset. We provide an argument of knowledge for the following relation:

$$\mathcal{R}_{\texttt{EqDLog}}[(a_1, b_1),\ (a_2, b_2)] = \left\{ \begin{array}{l} ((a_1, b_1),\ (a_2, b_2) \in \mathbb{G}^2 \\ d \in \mathbb{Z} : \\ (b_1, b_2) = (a_1^d, a_2^d) \end{array} \right\}$$

The protocol hinges on the observation that for two integers $d_1, d_2$, if we have $d_1 \equiv d_2 \pmod{\ell}$ for a randomly generated $\lambda$-bit prime $\ell$, then with overwhelming probability, $d_1 = d_2$.

**Protocol 3.2.** *Proof of equality of discrete logarithms* (PoEqDLog) :

**Parameters:** $\mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda),\ g \in \mathbb{G}$.

**Inputs:** $a_1, a_2, b_1, b_2 \in \mathbb{G}$.

**Claim:** The Prover possesses an integer $d$ such that $a_1^d = b_1$ and $a_2^d = b_2$.

1. The Prover $\mathcal{P}$ sends $\widetilde{g} := g^d$ to the Verifier $\mathcal{V}$.
2. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\ell$.
3. $\mathcal{P}$ computes the integers $q, r$ such that $d = q\ell + r$, $r \in [\ell]$ and the group elements

$$Q_1 := a_1^q,\ \ Q_2 := a_2^q,\ \ \breve{g} := g^q.$$

He sends $(Q_1, Q_2, \breve{g}, r)$ to $\mathcal{V}$.

4. $\mathcal{V}$ verifies the equations

$$r \in [\ell],\ \ Q_1^\ell a_1^r \stackrel{?}{=} b_1,\ \ Q_2^\ell a_2^r \stackrel{?}{=} b_2,\ \ (\breve{g})^\ell g^r \stackrel{?}{=} \widetilde{g}.$$

He accepts if and only if all equations hold. □

Thus, the proof consists of four $\mathbb{G}$-elements and one $\lambda$-bit integer.

**Proposition 3.3.** *The protocol* $\texttt{EqDLog}[(a_1, b_1),\ (a_2, b_2)]$ *is an argument of knowledge for the relation* $\mathcal{R}_{\texttt{EqDLog}}$ *in the generic group model.*

*Proof.* Since the protocol PoKE is secure ([BBF19]), the validity of the equations

$$Q_1^l a_1^r \stackrel{?}{=} b_1, \;\; Q_2^l a_2^r \stackrel{?}{=} b_2, \;\; (\breve{g})^l g^r \stackrel{?}{=} \widetilde{g}$$

proves that $\mathcal{P}$ possesses integers $d_1, d_2$ such that $a_1^{d_1} = b_1$, $a_2^{d_2} = b_2$. Suppose, by way of contradiction, $d_1 \neq d_2$. The adaptive root assumption implies that with overwhelming probability,

$$d_1 \equiv r \equiv d_2 \pmod{\ell}.$$

But since the $\lambda$-bit prime $\ell$ is randomly generated, the integer $d_1 - d_2$ is randomly and uniformly distributed modulo $\ell$ and hence,

$$\mathbf{Prob}\big(d_1 \equiv d_2 \pmod{\ell} \;\Big|\; d_1 \neq d_2\big) = \frac{1}{\ell} = \mathtt{negl}(\lambda).$$

Thus, with overwhelming probability $d_1 = d_2$. □

We can also generalize the protocol EqDLog as follows. For a public polynomial $f(X) \in \mathbb{Z}[X]$, an honest Prover can provide a constant-sized proof that he possesses integers $d_1, d_2$ such that

$$a_1^{d_1} = b_1 \;,\; a_2^{d_2} = b_2 \;,\; f(d_1) = d_2.$$

We provide an argument of knowledge for the relation

$$\mathcal{R}_{\mathtt{PolyDLog}}[(a_1, b_1),\ (a_2, b_2),\ f] = \left\{ \begin{array}{l} ((a_1, b_1), (a_2, b_2) \in \mathbb{G}^2,\ f \in \mathbb{Z}[X]); \\ (d_1, d_2) \in \mathbb{Z}^2 : \\ b_1 = a_1^{d_1} \bigwedge b_1 = a_1^{d_1} \bigwedge d_2 = f(d_1) \end{array} \right\}$$

**Protocol 3.4.** *Proof of Polynomial relation between discrete logarithms* (PoPolyDLog) :
**Parameters:** $\mathbb{G} \stackrel{\$}{\leftarrow} \mathrm{GGen}(\lambda),\ g \in \mathbb{G}$.
**Inputs:** Elements $a_1, b_1, a_2, b_2 \in \mathbb{G}$, a public polynomial $f(X) \in \mathbb{Z}[X]$.
**Claim:** The Prover possesses integers $d_1$, $d_2$ such that:
-$a_1^{d_1} = b_1,\ a_2^{d_2} = b_2$
-$f(d_1) = d_2$

1. The Prover $\mathcal{P}$ computes $\widetilde{g}_1, \widetilde{g}_2$ and sends them to the Verifier $\mathcal{V}$.
2. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\ell$.
3. $\mathcal{P}$ computes elements $q_1, q_2, r_1, r_2$ such that

$$d_1 = q_1\ell + r_1,\ d_2 = q_1\ell + r_1, \;\; r_1, r_2 \in [\ell].$$

4. $\mathcal{P}$ computes the elements $Q_1 := a_1^{q_1},\ Q_2 := a_2^{q_2},\ g_1 := g^{q_1},\ g_2 := g^{q_2} \in \mathbb{G}$ and sends them to $\mathcal{V}$ along with the integer $r_1$.
5. The Verifer verifies that $r_1 \in [\ell]$ and independently computes $r_2 := f(r_1) \pmod{\ell}$.
6. $\mathcal{V}$ verifies the equations

$$Q_1^\ell a_1^{r_1} \stackrel{?}{=} b_1 \;\bigwedge\; Q_2^\ell a_2^{r_2} \stackrel{?}{=} b_2 \;\bigwedge\; (g_1)^\ell g^{r_1} \stackrel{?}{=} \widetilde{g}_1 \;\bigwedge\; (g_2)^\ell g^{r_2} \stackrel{?}{=} \widetilde{g}_2$$

and accepts the validity of the claim if and only if all equations hold. □

Thus, the proof consists of six elements of $\mathbb{G}$ and one $\lambda$-bit integer.

**Proposition 3.5.** *The protocol* PoPolyDLog *is an argument of knowledge in the generic group model.*

*Proof.* This is a special case of the protocol PoMultPolyDLog, which we provide a security proof for in the next section. □

In the next section, we will generalize this protocol to multivariate polynomial relations for multiple discrete logarithms. We briefly discuss an application of the last protocol.

### 3.1.1 Underlying sets of committed multisets

Let $a_1, a_2$ be elements of $\mathbb{G}$. Let $\mathcal{M}, \mathcal{N}$ be multisets of rational primes. Let

$$A_1 := \texttt{Com}(g, \mathcal{M}) = a_1^{\Pi(\mathcal{M})}\ ,\ A_2 := \texttt{Com}(g, \mathcal{N}) = a_2^{\Pi(\mathcal{N})}$$

be the commitments to $\mathcal{M}$, $\mathcal{N}$ with bases $a_1, a_2 \in \mathbb{G}$.

Clearly, the relation $\mathcal{N} \subseteq \mathcal{M}$ can be demonstrated by the protocol $\texttt{PoKE}[A_2, A_1]$. We now show that the protocol PolyDLog allows a Prover to succinctly demonstrate the following relations between the underlying sets of $\mathcal{M}$, $\mathcal{N}$, the proofs for which can be publicly verified against the commitments to $\mathcal{M}$ and $\mathcal{N}$.

1. $\texttt{Set}(\mathcal{M}) \subseteq \texttt{Set}(\mathcal{N})$.
2. $\texttt{Set}(\mathcal{M}) \nsubseteq \texttt{Set}(\mathcal{N})$.
3. $\texttt{Set}(\mathcal{M}) = \texttt{Set}(\mathcal{N})$

Before we describe the protocols, we note a few basic facts. Clearly, we have

$$\texttt{Set}(\mathcal{M}) = \texttt{Set}(\mathcal{N}) \iff \texttt{Set}(\mathcal{M}) \subseteq \texttt{Set}(\mathcal{N}) \bigwedge \texttt{Set}(\mathcal{N}) \subseteq \texttt{Set}(\mathcal{M}).$$

Furthermore, with notations as before, we have

$$\texttt{Set}(\mathcal{M}) \subseteq \texttt{Set}(\mathcal{N}) \iff \exists\ N :\ \Pi(\mathcal{M})^N \equiv 0 \pmod{\Pi(\mathcal{N})}.$$

Likewise, to show that $\texttt{Set}(\mathcal{M}) \nsubseteq \texttt{Set}(\mathcal{N})$, it suffices to show that there exists an integer $p$ such that

$$p \notin \{-1, 1\} \bigwedge \Pi(\mathcal{M}) \equiv 0 \pmod{p} \bigwedge \mathbf{gcd}(\Pi(\mathcal{N}), p) = 1.$$

**Protocol 3.6.** *Protocol for the containment of underlying sets* (PoConSets).

**Parameters:** $\mathbb{G} \xleftarrow{\$} \text{GGen}(\lambda),\ g \in \mathbb{G}$.
**Input:** Elements $a_1, a_2 \in \mathbb{G}$; commitments $A_1 = a_1^{\Pi(\mathcal{M})}\ ,\ A_2 = a_2^{\Pi(\mathcal{N})}$ to multisets $\mathcal{M}$, $\mathcal{N}$.
**Claim:** $\texttt{Set}(\mathcal{N}) \subseteq \texttt{Set}(\mathcal{M})$.

1. The Prover $\mathcal{P}$ computes $N := \max\{\text{mult}(\mathcal{N}, x) : x \in \mathcal{N}\}$ and

$$A_3 := a_1^{\Pi(\mathcal{M})^N}.$$

He sends $A_3$ and $N$ to the Verifier $\mathcal{V}$.

2. $\mathcal{P}$ computes a non-interactive proof for $\texttt{PoPolyDLog}[(a_1, A_1),\ (a_2, A_3),\ X^N]$ and sends it to $\mathcal{V}$.
3. $\mathcal{P}$ computes a non-interactive proof for $\texttt{PoKE}[A_2, A_3]$ and sends it to $\mathcal{V}$.
4. $\mathcal{V}$ verifies the two proofs and accepts if and only if both are valid. □

**Protocol 3.7.** *Protocol for the non-containment of underlying sets* (PoNonConSets).

**Parameters:** $\mathbb{G} \xleftarrow{\$} \text{GGen}(\lambda),\ g \in \mathbb{G}$.

**Input:** Elements $a_1, a_2 \in \mathbb{G}$; commitments $A_1 := \texttt{Com}(a_1, \mathcal{M}) = a_1^{\Pi(\mathcal{M})}$ , $A_2 := \texttt{Com}(a_2, \mathcal{N}) = a_2^{\Pi(\mathcal{N})}$ to multisets $\mathcal{M}$, $\mathcal{N}$.

**Claim:** $\texttt{Set}(\mathcal{M}) \nsubseteq \texttt{Set}(\mathcal{N})$.

1. The Prover chooses an integer $p$ such that $p \in \texttt{Set}(\mathcal{M}) \setminus \texttt{Set}(\mathcal{N})$. and computes $b_1 := a_1^p$. He sends $b_1$ to the Verifier $\mathcal{V}$ along with a non-interactive proof for $\texttt{PoKE}[b_1, A_1]$.

2. $\mathcal{P}$ computes a non-interactive proof for $\texttt{RelPrimeDLog}[(a_1, b_1),\ (a_2, A_2)]$ and sends it to $\mathcal{V}$.

3. $\mathcal{V}$ verifies that $b_1 \notin \{a_1, a_1^{-1}\}$ and the proofs for $\texttt{PoKE}[b_1, A_1]$, $\texttt{RelPrimeDLog}[(a_1, b_1),\ (a_2, A_2)]$. He accepts if and only if both proofs are valid. □

In both cases, the proofs consists of a constant number of $\mathbb{G}$-elements and $\lambda$-bit integers. In particular, the proof size is independent of the sizes of $\mathcal{M}$ and $\mathcal{N}$.

### 3.2 Aggregating the knowledge of multiple exponents

In this section, we discuss protocols for aggregating the proofs of knowledge of multiple exponents and proofs of certain relations between these exponents. This amounts to demonstrating relations between multiple sets/multisets through non-interactive proofs that can be publicly verified against the commitments to these sets/multisets. The proofs consist of elements of the hidden order $\mathbb{G}$ and $\lambda$-bit integers arising from the remainders of the exponents modulo the prime challenges. Using Lemma 2.3 and a few more techniques, we have designed the protocols so that the number of $\mathbb{G}$-elements is constant and hence, independent of the number of exponents involved. The proof sizes are $\mathbf{O}(n)$ since they consist of $\mathbf{O}(n)$ $\lambda$-bit integers. However, in practice, this is a lot more efficient in terms of the communication complexity than proofs with $\mathbf{O}(n)$ group elements.

The first protocol in this section allows us demonstrate the knowledge of multiple integer exponents when they share a common base. We call this protocol the *Proof of Aggregated Knowledge of the Exponents* 1 or `PoAggKE-1` for short. We provide an argument of knowledge for the relation:

$$\mathcal{R}_{\texttt{AggKE-1}}[a,\ \mathcal{A}] = \left\{ \begin{array}{l} (a \in \mathbb{G}\ ,\ \mathcal{A} = (a_1, \cdots, a_n) \in \mathbb{G}^n); \\ (d_1, \cdots, d_n) \in \mathbb{Z}^n) : \\ a_i = a^{d_i}\ \forall\ i \end{array} \right\}$$

In addition to the cryptographic assumptions for generic groups, the security of the protocol hinges on the Schartz-Zippel lemma, which we state here.

**Lemma 3.8.** (Schwartz-Zippel) : *Let $F$ be a field and let $f \in F[X_1, \cdots, X_n]$ be a polynomial. Let $r_1, \cdots, r_n$ be selected randomly and uniformly from a subset $S \subseteq F$. Then*

$$\textbf{Prob}[f(r_1, \cdots, r_n) = 0] \leq \frac{\deg(f)}{|S|}.$$

**Protocol 3.9.** *Proof of aggregated knowledge of exponents* 1 (`PoAggKE-1`):

**Parameters:** $\mathbb{G} \xleftarrow{\$} \text{GGen}(\lambda),\ g \in \mathbb{G}$.

**Inputs:** Elements $a \in \mathbb{G}$, $(b_1, \cdots, b_n) \in \mathbb{G}^n$ for some integer $n \geq 1$.

**Claim:** The Prover possesses integers $d_1, \cdots, d_n$ such that $a^{d_i} = b_i$ for $i = 1, \cdots, n$.

1. The Fiat-Shamir heuristic generates $\lambda$-bit challenge $\gamma$.

2. The Prover $\mathcal{P}$ computes

$$p := \texttt{NextPrime}(n\lambda) \ , \ \widetilde{g} := g^{\sum_{i=1}^{n} d_i^p \gamma^i}$$

and sends $\widetilde{g}$ to the Verifier $\mathcal{V}$.

3. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\ell \not\equiv 1 \pmod{p}$.

4. $\mathcal{P}$ computes the integers $r_i := d_i \pmod{\ell}$ and the integers $q$, $r$, $\widetilde{q}$, $\widetilde{r}$ such that

$$\sum_{i=1}^{n} d_i \gamma^i = q\ell + r \ , \ \sum_{i=1}^{n} d_i^p \gamma^i = \widetilde{q}\ell + \widetilde{r}, \quad r, \widetilde{r} \in [\ell]$$

and

$$Q := a^q \ , \ \breve{g} := g^{\widetilde{q}}.$$

He sends $Q$, $\breve{g}$, $(r_1, \cdots, r_n)$ to the Verifier.

5. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\gamma_0$.

6. $\mathcal{P}$ computes integers $q_0, r_0$ such that

$$\sum_{i=1}^{n} d_i \gamma_0^i = q_0\ell + r_0 \ , \ r_0 \in [\ell].$$

He computes $Q_0 := a^{q_0}$ and sends it to $\mathcal{V}$.

7. $\mathcal{V}$ verifies that $(r_1, \cdots, r_n) \in [\ell]^n$ and independently computes

$$b := \prod_{i=1}^{n} b_i^{\gamma^i} \ , \ b_0 := \prod_{i=1}^{n} b_i^{\gamma_0^i} \ ,$$

$$\widetilde{r} := \sum_{i=1}^{n} r_i^p \gamma^i \pmod{\ell} \ , \ r := \sum_{i=1}^{n} r_i \gamma^i \pmod{\ell} \ , \ r_0 := \sum_{i=1}^{n} r_i \gamma_0^i \pmod{\ell}.$$

8. $\mathcal{V}$ verifies the equations

$$Q^\ell a^r \stackrel{?}{=} b \bigwedge (Q_0)^\ell a^{r_0} \stackrel{?}{=} b_0 \bigwedge (\breve{g})^\ell g^{\widetilde{r}} \stackrel{?}{=} \widetilde{g}.$$

He accepts if and only if all equations hold. □

Thus, the proof consists of three $\mathbb{G}$-elements and $n$ $\lambda$-bit integers. In particular, the number of $\mathbb{G}$-elements is constant-sized and independent of the number of exponents. For the security of the protocol, it is necessary that the challenge $\gamma_0$ is generated *after* the remainders $r_1, \cdots, r_n$ have been committed. In a non-interactive setting, this means the hashing algorithm that generates $\gamma_0$ takes the vector $(r_1, \cdots, r_n) \in [\ell]^n$ of remainders modulo $\ell$ as one of its inputs. Hence, the remainders $r_i := d_i \pmod{\ell}$ must be honestly computed by the Prover in order to succeed at the additional task of computing the element $Q_0 \in \mathbb{G}$ such that $(Q_0)^\ell a^{r_0} = b_0$.

In the special case where $a$ is randomly generated by the oracle, the subprotocol where the Prover computes the element $\widetilde{g}$ and sends it to the Verifier is redundant. So the proof would be smaller and the Prover's computational burden would be substantially lower in this special case.

When $a$ is not randomly generated, the most expensive part of the proof generation is co mputing the element $\widetilde{g}$. The effective runtime for this can be mitigated by the Prover pre-computing the set

$$\{g^{2^i} : 1 \leq i \leq N\}$$

for some appropriately large integer $N$.

**Proposition 3.10.** *The protocol* `PoAggKE-1` *is an argument of knowledge for the relation* $\mathcal{R}_{\texttt{AggKE}-1}$ *in the generic group model.*

*Proof.* (Sketch) Since the equation $Q_0^{\ell} a^{r_0} = b_0$ holds, lemma 2.5 implies that with overwhelming probability, $\mathcal{P}$ possesses rationals $d_1, \cdots, d_n$ such that

$$a^{d_i} = b_i \ \ , \ \ \sum_{i=1}^{n} d_i \gamma_0^i \equiv \sum_{i=1}^{n} r_i \gamma_0^i \pmod{\ell}.$$

Since the challenge $\gamma_0$ is randomly generated after the integers $r_1, \cdots, r_n$ have been sent, the Schwartz-Zippel lemma implies that with overwhelming probability, $d_i \equiv r_i \pmod{\ell} \ \forall \ i$.

Furthermore, we have $\widetilde{g} = (\breve{g})^{\ell} g^{\widetilde{r}}$, which implies that with overwhelming probability, $\mathcal{P}$ possesses a rational $\widetilde{d}$ such that

$$\widetilde{g} = g^{\widetilde{d}} \ \ , \ \ \widetilde{d} \equiv \sum_{i=1}^{n} r_i^p \gamma^i \pmod{\ell}.$$

The fractional root assumption then implies that with overwhelming probability, $\widetilde{d} \in \mathbb{Z}$. Furthermore, since $\ell$ is randomly generated after $\widetilde{g}$ has been sent, we have

$$\widetilde{d} \equiv \sum_{i=1}^{n} d_i^p \gamma^i \pmod{\ell} \quad \stackrel{\text{o.p.}}{\Longrightarrow} \quad \widetilde{d} = \sum_{i=1}^{n} d_i^p \gamma^i.$$

Since $p > n\lambda$, Lemma 2.3 now implies that with overwhelming probability, $d_i \in \mathbb{Z} \ \forall \ i$ □

In the next protocol, we generalize the protocol `PolyDLog` to multiple discrete logarithms. We provide an argument of knowledge for the relation:

$$\mathcal{R}_{\texttt{MultPolyDLog}}[a, \ (b_1, \cdots, b_n), \ (f_1, \cdots, f_k)] = \left\{ \begin{array}{l} (a \in \mathbb{G}, \ (b_1, \cdots, b_n) \in \mathbb{G}^n); \\ (f_1, \cdots, f_k) \in \mathbb{Z}[X_1, \cdots, X_n]^k; \\ (d_1, \cdots, d_n) \in \mathbb{Z}^n) \ : \\ b_i = a^{d_i} \ \forall \ i \ \bigwedge \\ f_j(d_1, \cdots, d_n) = 0 \ \forall \ j \end{array} \right\}$$

**Protocol 3.11.** *Proof of multivariate polynomial relations between discrete logarithms* (`PoMultPolyDLog`) :

**Parameters:** $\mathbb{G} \stackrel{\$}{\leftarrow} \text{GGen}(\lambda), \ g \in \mathbb{G}$.

**Inputs:** Elements $a \in \mathbb{G}$, $(b_1, \cdots, b_n) \in \mathbb{G}^n$ for some integer $n \geq 1$; public $n$-variate polynomials $f_1, \cdots, f_k \ \in \ \mathbb{Z}[X_1, \cdots, X_n]$.

**Claim:** The Prover possesses integers $d_1, \cdots, d_n$ such that:
- $a^{d_i} = b_i$ for $i = 1, \cdots, n$.
- $f_j(d_1, \cdots, d_n) = 0$ for $j = 1, \cdots, k$.

1. The Fiat-Shamir heuristic generates a $\lambda$-bit integer $\gamma$.

2. $\mathcal{P}$ computes

$$p := \texttt{NextPrime}(n\lambda) \ \ , \ \ \widetilde{g} := g^{\sum_{i=1}^{n} d_i^p \gamma^i}$$

and sends $\widetilde{g}$ it to the Verifier $\mathcal{V}$.

3. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\ell \not\equiv 1 \pmod{p}$.

4. $\mathcal{P}$ computes the integers $r_i := d_i \pmod{\ell} \ \ (i = 1, \cdots, n)$ and the integers $\widetilde{q}, q, \widetilde{r}, r$ such that

$$\sum_{i=1}^{n} d_i^p \gamma^i = \widetilde{q}\ell + \widetilde{r} \ , \ \sum_{i=1}^{n} d_i \gamma^i = q\ell + r \ , \quad \widetilde{r}, r \in [\ell].$$

5. $\mathcal{P}$ computes $Q := a^q$ , $\breve{g} := g^{\widetilde{q}}$ and sends $Q$, $\breve{g}$, $(r_1, \cdots, r_n)$ to $\mathcal{V}$.

6. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\gamma_0$.

7. $\mathcal{P}$ computes the integers $q_0, r_0$ such that

$$\sum_{i=1}^{n} d_i \gamma_0^i = q_0\ell + r_0 \ , \ r_0 \in [\ell].$$

He computes $Q_0 := a^{q_0}$ and sends $Q_0$ to $\mathcal{V}$.

8. $\mathcal{V}$ verifies that $(r_1, \cdots, r_n) \in [\ell]^n$ and independently computes

$$\widetilde{r} := \sum_{i=1}^{n} r_i^p \gamma^i \pmod{\ell} \ , \ r := \sum_{i=1}^{n} r_i \gamma^i \pmod{\ell} \ , \ r_0 := \sum_{i=1}^{n} r_i \gamma_0^i \pmod{\ell}.$$

9. $\mathcal{V}$ computes

$$b := \prod_{i=1}^{n} b_i^{\gamma^i} \ , \ b_0 := \prod_{i=1}^{n} b_i^{\gamma_0^i}.$$

10. $\mathcal{V}$ verifies the equations

$$Q^\ell a^r \stackrel{?}{=} b \ \bigwedge \ (Q_0)^\ell a^{r_0} \stackrel{?}{=} b_0 \ \bigwedge \ (\breve{g})^\ell g^{\widetilde{r}} \stackrel{?}{=} \widetilde{g} \ \bigwedge \ \Big( \bigwedge_{j=1}^{k} f_j(r_1, \cdots, r_n) \stackrel{?}{\equiv} 0 \pmod{\ell} \Big).$$

He accepts the validity of the claim if and only if all equations hold. □

Thus, the proof consists of five $\mathbb{G}$-elements and $n$ $\lambda$-bit integers. We note that the additional challenge $\gamma_0$ is necessary for the security of the protocol. A malicious Prover $\mathcal{P}_{\text{mal}}$ could forge a fake proof as follows.

1. $\mathcal{P}_{\text{mal}}$ computes integers $r_1, \cdots, r_n \in [\ell]$ such that

$$\sum_{i=1}^{n} d_i^p \gamma^i \equiv \sum_{i=1}^{n} r_i^p \gamma^i \pmod{\ell} \ , \ \bigwedge_{j=1}^{k} f_j(r_1, \cdots, r_n) \equiv 0 \pmod{\ell}$$

but $d_i \not\equiv r_i \pmod{\ell}$ for some or all indices $i$. The malicious Prover can succeed in this task with non-negligible probability.

2. $\mathcal{P}_{\text{mal}}$ then sends $(r_1, \cdots, r_n)$ to the Verifier.

3. The Verifier is thus tricked into believing that $f_j(d_1, \cdots, d_n) = 0$, which might not necessarily be the case.

Now, in our protocol, $\gamma_0$ is randomly generated by the Fiat-Shamir heuristic *after* the Prover sends $(r_1, \cdots, r_n)$. In a non-interactive setting, this means the hashing algorithm that generates the challenge $\gamma_0$ takes the $\lambda$-bit integers $(r_1, \cdots, r_n)$ as one of its inputs. Hence,

$$\mathbf{Prob}\Big( \sum_{i=1}^{n} d_i \gamma_0^i \equiv \sum_{i=1}^{n} r_i \gamma_0^i \pmod{\ell} \ \Big| \ d_i \not\equiv r_i \pmod{\ell} \text{ for some } i \Big) = \mathtt{negl}(\lambda).$$

So the elements $(r_1, \cdots, r_n)$ must be honestly computed in order to succeed at the additional task of computing the element $\widehat{Q}_0$ such that

$$(\widehat{Q}_0)^{\ell} a^{\widehat{r}_0} = \prod_{i=1}^{n} a_i^{\gamma_0^i}$$

with non-negligible probability.

An important special case is where $f$ is the $(n+1)$-variate polynomial

$$f(X_1, \cdots, X_n, X_{n+1}) := \big(\prod_{i=1}^{n} X_i\big) - X_{n+1}.$$

We will need this case for some of the subsequent protocols for demonstrating pairwise disjointness of committed data sets/multisets.

As was the case with `AggKE-1`, in the special case where $a = g$, the subprotocol where the Prover computes the element $\widetilde{g}$ and sends it to the Verifier is redundant. So the proof would be smaller and the Prover's computational burden would be substantially lower in this special case.

**Proposition 3.12.** *The protocol* `PoMultPolyDLog` *is an argument of knowledge for the relation* $\mathcal{R}_{\texttt{MultPolyDLog}}$ *in the generic group model.*

*Proof.* (Sketch) The Verifier independently computes the elements $r_0 := \sum_{i=1}^{n} r_i \gamma_0^i \pmod{\ell}$. Since the equation $Q_0^{\ell} a^{r_0} = \prod_{i=1}^{n} b_i^{\gamma_0^i}$ holds, lemma 2.5 implies that with overwhelming probability, the Prover possesses rationals $d_1, \cdots, d_n$ such that:

- $a^{d_i} = b_i$ for $i = 1, \cdots, n$
- $\sum_{i=1}^{n} d_i \gamma_0^i \equiv \sum_{i=1}^{n} r_i \gamma_0^i \pmod{\ell}$.

Since the challenge $\gamma_0$ is randomly generated after the Prover has sent $(r_1, \cdots, r_n)$, $\gamma_0$ is randomly and uniformly distributed modulo $\ell$. Hence, the Schwart-Zippel lemma implies that with overwhelming probability, $d_i \equiv r_i \pmod{\ell}$ for every index $i$. Now, since the prime $\ell$ is randomly generated,

$$\textbf{Prob}\Big(f_j(d_1, \cdots, d_n) \equiv 0 \pmod{\ell} \;\Big|\; f_j(d_1, \cdots, d_n) \neq 0\Big) = \texttt{negl}(\lambda).$$

Hence,

$$f_j(d_1, \cdots, d_n) \equiv 0 \pmod{\ell}\ \forall\ j \quad \overset{\text{o.p.}}{\Longrightarrow} \quad f_j(d_1, \cdots, d_n) = 0\ \forall\ j.$$

It remains to verify that the rationals $d_i$ are integers. The Verifier independently computes

$$\widetilde{r} := \sum_{i=1}^{n} r_i^p \gamma^i \equiv \sum_{i=1}^{n} d_i^p \gamma^i \pmod{\ell}.$$

So, the equation $\widetilde{g} = (\breve{g})^{\ell} g^{\widetilde{r}}$ and lemma 2.4 imply that the Prover possesses a rational $\widetilde{d}$ such that

$$\widetilde{g} = g^{\widetilde{d}} \ , \ \ \widetilde{d} \equiv \widetilde{r} \equiv \sum_{i=1}^{n} r_i^p \gamma^i \equiv \sum_{i=1}^{n} d_i^p \gamma^i \pmod{\ell}.$$

If $\widetilde{d} \notin \mathbb{Z}$, the tuple $(g, \widetilde{g}, \widetilde{d})$ would violate the fractional root assumption. So $\widetilde{d}$ is an integer, except with negligible probability. Since the $\lambda$-bit prime $\ell$ is randomly generated after $\widetilde{g}$ has been sent by the Prover, the congruence

$$\widetilde{d} \equiv \sum_{i=1}^{n} d_i^p \gamma^i \pmod{\ell}$$

implies that with overwhelming probability, $\widetilde{d} = \sum_{i=1}^{n} d_i^p \gamma^i$. Hence, $\sum_{i=1}^{n} d_i^p \gamma^i \in \mathbb{Z}$ and Lemma 2.3 then implies that $d_i \in \mathbb{Z}\ \forall\ i$ except with negligible probability. □

We now discuss a relation (and its argument of knowledge) that is a dual to the relation `AggKE-1`. Instead of the multiple exponents having a common base, we consider the case where they exponentiate to the same power. We provide an argument of knowledge for the following relation:

$$\mathcal{R}_{\texttt{AggKE-2}}[(a_1, \cdots, a_n),\ A] = \left\{ \begin{array}{l} ((a_1, \cdots, a_n) \in \mathbb{G}^n,\ A \in \mathbb{G}) \\ (d_1, \cdots, d_n) \in \mathbb{Z}^n) : \\ A = a_i^{d_i}\ \forall\ i \end{array} \right\}$$

Given elements $a_1, \cdots, a_n$ such that

$$A = a_1^{d_1} = \cdots = a_n^{d_n}$$

where the integers $d_i$ are known to him, the Prover can efficiently compute $d := \mathbf{lcm}(d_1, \cdots, d_n)$ and using Shamir's trick, an element $a \in \mathbb{G}$ such that $a^d = A$ in runtime $\mathbf{O}(n \log(n))$. Now, the protocol $\texttt{PoAggKE-1}[a,\ (a_1, \cdots, a_n)]$ and $\texttt{PoKE}[a,\ A]$ would demonstrate that the Prover possesses the discrete logarithms between $a_i$ and $A$ for every $i$. However, these protocols do not prove that these discrete logarithms are, in fact, integers. To that end, a Prover needs to demonstrate that the rationals $d/d_i$ $(i = 1, \cdots, n)$ are integers. This can be achieved by the Prover producing the element

$$\widetilde{g} := g^{\sum_{i=1}^{n} d_i^p \gamma^i}$$

for some randomly generated $\lambda$-bit integer $\gamma$. Lemma 2.3 and the fractional root assumption then imply that with overwhelming probability, the $d/d_i$ are integers.

**Protocol 3.13.** *Proof of Aggregated knowledge of exponents* 2 (`PoAggKE-2`):

**Parameters:** $\mathbb{G} \xleftarrow{\$} \text{GGen}(\lambda),\ g \in \mathbb{G}$

**Inputs:** $(a_1, \cdots, a_n) \in \mathbb{G}^n$, $A \in \mathbb{G}$.

**Claim:** The Prover posseses integers $d_1, \cdots, d_n$ such that $a_i^{d_i} = A$.

1. The Prover $\mathcal{P}$ computes the integers

$$D := \mathbf{lcm}(d_1, \cdots, d_n)\ ,\ \widehat{d_i} := D/d_i\ (i = 1, \cdots, n).$$

Using Shamir's trick, he computes an element $a \in \mathbb{G}$ such that $a^D = A$. He sends $a$ to the Verifier $\mathcal{V}$ along with a non-interactive $\texttt{PoKE}[a,\ A]$.

2. The Fiat-Shamir heuristic generates a $\lambda$-bit integer $\gamma$.

3. $\mathcal{P}$ computes

$$p := \texttt{NextPrime}(n\lambda)\ \ ,\ \ \widetilde{g} := g^{\sum_{i=1}^{n} d_i^p \gamma^i}$$

and sends it to the Verifier $\mathcal{V}$.

4. $\mathcal{P}$ computes a non-interactive proof for $\texttt{AggKE-1}[a,\ (a_1, \cdots, a_n)]$ and sends it to $\mathcal{V}$.

5. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\ell \not\equiv 1 \pmod{p}$.

6. $\mathcal{P}$ computes $R := D \pmod{\ell}$ and $\breve{a} := a^{(D-R)/\ell}$. He sends $\breve{a}, R$ to $\mathcal{V}$.

7. $\mathcal{P}$ computes the integers $\widehat{r}_i := \widehat{d_i} \pmod{\ell}$ $(i = 1, \cdots, n)$ and the integers $\widehat{q}, \widehat{r}$ such that

$$\sum_{i=1}^{n} \widehat{d}_i \gamma^i = \widehat{q}\ell + \widehat{r} \ , \ \ \widehat{r} \in [\ell].$$

He computes $\widehat{Q} := a^{\widehat{q}}$ and sends $\widehat{Q},\ (\widehat{r}_1, \cdots, \widehat{r}_n)$ to $\mathcal{V}$.

8. $\mathcal{P}$ computes the integers $r_i := d_i \pmod{\ell}$ $(i = 1, \cdots, n)$ and the integers $q, r, \widetilde{q}, \widetilde{r}$ such that

$$\sum_{i=1}^{n} d_i \gamma^i = q\ell + r \ , \ \ \sum_{i=1}^{n} d_i^p \gamma^i = \widetilde{q}\ell + \widetilde{r} \ , \quad r, \widetilde{r} \in [\ell]$$

He computes $Q := a^q$, $\breve{g} := g^{\widetilde{q}}$ and sends $Q, \breve{g}$ to $\mathcal{V}$.

9. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\gamma_0$.

10. $\mathcal{P}$ computes the integers $\widehat{q}_0, \widehat{r}_0$ such that

$$\sum_{i=1}^{n} \widehat{d}_i \gamma^i = \widehat{q}_0 \ell + \widehat{r}_0 \ , \ \widehat{r}_0 \in [\ell].$$

He computes $\widehat{Q}_0 := a^{q_0}$ and sends $Q_0$ to $\mathcal{V}$.

11. $\mathcal{V}$ verifies that $(\widehat{r}_1, \cdots, \widehat{r}_n, R) \in [\ell]^{n+1}$ and independently computes $r_i \equiv \widehat{r}_i^{-1} R \pmod{\ell}$ $(i = 1, \cdots, n)$ and

$$\widehat{r} := \sum_{i=1}^{n} \widehat{r}_i \gamma^i \pmod{\ell} \ , \ \widetilde{r} := \sum_{i=1}^{n} r_i^p \gamma^i \pmod{\ell} \ , \ \widehat{r}_0 := \sum_{i=1}^{n} \widehat{r}_i \gamma_0^i \pmod{\ell}$$

12. $\mathcal{V}$ verifies the equations

$$(\breve{a})^\ell a^R \stackrel{?}{=} A \bigwedge (\widehat{Q})^\ell a^{\widehat{r}} \stackrel{?}{=} \prod_{i=1}^{n} a_i^{\gamma^i} \bigwedge (\widehat{Q}_0)^\ell a^{\widehat{r}_0} \stackrel{?}{=} \prod_{i=1}^{n} a_i^{\gamma_0^i} \bigwedge (\breve{g})^\ell g^{\widetilde{r}} \stackrel{?}{=} \widetilde{g}.$$

He accepts the validity of the claim if and only if all equations hold and the proofs for $\texttt{PoKE}[a,\ A]$, $\texttt{AggKE-1}[a,\ (a_1, \cdots, a_n)]$ are valid. □

Thus, the proof consists of a constant number of $\mathbb{G}$-elements and $2n + \mathbf{O}(1)$ $\lambda$-bit integers. We note that the additional challenge $\gamma_0$ is necessary for the security of this protocol. The Prover commits the integer $\sum_{i=1}^{n} d_i^p \gamma^i$ by computing $\widetilde{g} := g^{\sum_{i=1}^{n} d_i^p \gamma^i}$ and sending it to the Verifier *before* the challenge $\ell$ is generated by the Fiat-Shamir heuristic. However, a malicious Prover $\mathcal{P}_{\text{mal}}$ could forge a fake proof as follows:

1. $\mathcal{P}_{\text{mal}}$ chooses integers $e_1, \cdots, e_n$ and sends $g^{\sum_{i=1}^{n} e_i \gamma^i}$ to the Verifier instead of $g^{\sum_{i=1}^{n} d_i^p \gamma^i}$

2. $\mathcal{P}_{\text{mal}}$ chooses integers $r_1, \cdots, r_n \in [\ell]$ such that

$$\sum_{i=1}^{n} (D d_i^{-1}) \gamma^i \equiv \sum_{i=1}^{n} (D r_i^{-1}) \gamma^i \pmod{\ell} \ , \ \sum_{i=1}^{n} e_i \gamma^i \equiv \sum_{i=1}^{n} r_i \gamma^i \pmod{\ell},$$

but $d_i \not\equiv r_i \pmod{\ell}$ for some or all indices $i$. The Prover $\mathcal{P}_{\text{mal}}$ can do so with non-negligible probability.

3. Thus, the Verifier is tricked into believing that $\sum_{i=1}^{n} d_i^p \gamma^i$ is an integer, which might not necessarily be the case. In fact, even if the Fiat-Shamir heuristic outputs the additional challenge

$\gamma_0$ before the remainders $(r_1, \cdots, r_n, R)$ are committed, $\mathcal{P}_{\text{mal}}$ can forge a fake proof with non-negligible probability.

Now, in our protocol, $\gamma_0$ is randomly generated by the Fiat-Shamir heuristic *after* the Prover sends $(\widehat{r}_1, \cdots, \widehat{r}_n, R)$. Hence, we have

$$\textbf{Prob}\Big( \sum_{i=1}^{n} d_i \gamma_0^i \equiv \sum_{i=1}^{n} r_i \gamma_0^i \pmod{\ell} \;\Big|\; d_i \not\equiv r_i \pmod{\ell} \text{ for some } i \Big) = \texttt{negl}(\lambda).$$

Hence, the elements $(r_1, \cdots, r_n)$ must be honestly computed in order to succeed at the additional challenge of computing the element $\widehat{Q}_0$ such that

$$\widehat{Q}_0^{\ell} a^{\widehat{r}_0} = \prod_{i=1}^{n} a_i^{\gamma_0^i}$$

with non-negligible probability.

The most expensive part of the proof generation is computing the element $\widetilde{g}$. The effective runtime could be reduced if the Prover pre-computes the set

$$\{g^{2^i} : 1 \leq i \leq N\}$$

for an appropriately large integer $N$.

**Proposition 3.14.** *The protocol* PoAggKE-2 *is an argument of knowledge for the relation* $\mathcal{R}_{\texttt{AggKE}-2}$ *in the generic group model.*

*Proof.* (Sketch) The subprotocol PoAggKE-1$[a,\ (a_1, \cdots, a_n)]$ demonstrates that with overwhelming probability, the Prover possesses integers $\widehat{d}_1, \cdots, \widehat{d}_n$ such that

$$a_i = a^{\widehat{d}_i} \ (i = 1. \cdots, n)$$

Furthermore, since the equation

$$\widehat{Q}_0^{\ell} a^{\widehat{r}_0} = \prod_{i=1}^{n} a_i^{\gamma_0^i}$$

holds, the adaptive root assumption implies that with overwhelming probability,

$$\sum_{i=1}^{n} \widehat{r}_i \gamma_0^i \equiv \sum_{i=1}^{n} \widehat{d}_i \gamma_0^i \pmod{\ell}.$$

Since the $\lambda$-bit challenge $\gamma_0$ is randomly generated, the Schwartz-Zippel lemma implies that with overwhelming probability, $\widehat{r}_i \equiv \widehat{d}_i \pmod{\ell}\ \forall\ i$.

The equation $(\breve{a})^{\ell} a^R = A$ implies that with overwhelming probability, the Prover possesses a rational $D \equiv R \pmod{\ell}$ such that $a^D = A$. Thus, with overwhelming probability, the rationals $D\widehat{d}_1^{\ -1}, \cdots, D\widehat{d}_n^{-1}$ satisfy

$$D\widehat{d}_i^{\ -1} \equiv R\widehat{r}_i^{-1} \pmod{\ell} \text{ for every } i.$$

This, in turn, implies that with overwhelming probability,

$$a_i^{D\widehat{d}_i^{\ -1}} = A \text{ for every } i.$$

Now, the Verifier independently computes the $\lambda$-bit integers

$$r_i := R\widehat{r}_i^{-1} \equiv D\widehat{d}_i^{-1} \pmod{\ell} \quad (i = 1, \cdots, n)$$

$$r := \sum_{i=1}^{n} r_i \gamma^i \equiv \sum_{i=1}^{n} D\widehat{d}_i^{-1} \pmod{\ell} \ , \ \widetilde{r} := \sum_{i=1}^{n} r_i^p \gamma^i \equiv \sum_{i=1}^{n} (D\widehat{d}_i^{-1})^p \gamma^i \pmod{\ell}.$$

Hence, the equation $(\breve{g})^\ell g^{\widetilde{r}} = \widetilde{g}$ implies that with overwhelming probability, the Prover possesses a rational $\widetilde{d}$ such that $\widetilde{g} = g^{\widetilde{d}}$. The adaptive root assumption implies that with overwhelming probability,

$$\widetilde{d} \equiv \widetilde{r} \equiv \sum_{i=1}^{n} r_i^p \gamma^i \equiv \sum_{i=1}^{n} \left(D\widehat{d}_i^{-1}\right)^p \gamma^i \pmod{\ell}$$

Since the $\lambda$-bit prime $\ell$ is randomly generated after $\widetilde{g}$ has been sent by the Prover, it follows that with overwhelming probability,

$$\widetilde{d} = \sum_{i=1}^{n} \left(D\widehat{d}_i^{-1}\right)^p \gamma^i.$$

Now, if $\widetilde{d}$ were not an integer, the tuple $(g, \widetilde{g}, \widetilde{d})$ would violate the fractional root assumption. Thus, with overwhelming probability, $\widetilde{d}$ is an integer. Since $p > n\lambda$, Lemma 2.3 now implies that with overwhelming probability, the rationals $D\widehat{d}_i^{-1}$ are integers. □

## 4 Protocols for arguments of disjointness

The goal of this section is to provide protocols for demonstrating disjointness of multiple data sets/multisets. The proofs can be publicly verified against the succinct commitments to these multisets. To that end, we first describe a protocol whereby an honest Prover can show that the GCD of two discrete logarithms equals a third discrete logarithm while keeping the communication complexity constant. One obvious application is proving disjointness of sets in accumulators instantiated with hidden order groups. We formulate an argument of knowledge for the relation

$$\mathcal{R}_{\texttt{GCD}}[(a_1, b_1),\ (a_2, b_2),\ (a_3, b_3)] = \{((a_i, b_i \in \mathbb{G});\ d_i \in \mathbb{Z}) \ :\ b_i = a_i^{d_i},\ \mathbf{gcd}(d_1, d_2) = d_3\}.$$

We construct a protocol that has communication complexity independent of the elements $a_i, b_i$. The protocol rests on the simple observation that

$$d_3 = \mathbf{gcd}(d_1, d_2) \iff (d_1 \equiv d_2 \equiv 0 \pmod{d_3}) \bigwedge \left(\exists\, (x_1, x_2) \in \mathbb{Z}^2 :\ d_3 = x_1 d_1 + x_2 d_2\right).$$

**Protocol 4.1.** *Proof of the greatest common divisor* (PoGCD):

**Parameters:** $\mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda),\ g \in \mathbb{G}$.

**Inputs:** Elements $a_1, a_2, a_3, b_1, b_2, b_3 \in \mathbb{G}$.

**Claim:** The Prover possesses integers $d_1$, $d_2$, $d_3$ such that:
- $a_1^{d_1} = b_1,\ a_2^{d_2} = b_2,\ a_3^{d_3} = b_3$
- $\mathbf{gcd}(d_1, d_2) = d_2$

1. The Prover $\mathcal{P}$ computes $b_{1,2} := a_1^{d_2}$, $b_{1,3} := a_1^{d_3}$ and sends them to the Verifier $\mathcal{V}$.

2. $\mathcal{P}$ generates non-interactive proofs for $\texttt{EqDLog}[(a_2, b_2),\ (a_1, b_{1,2})]$, $\texttt{EqDLog}[(a_3, b_3),\ (a_1, b_{1,3})]$ and sends them to $\mathcal{V}$.

3. $\mathcal{P}$ generates non-interactive proofs for $\texttt{PoKE}[b_{1,3},\ b_1]$ and $\texttt{PoKE}[b_{1,3},\ b_{1,2}]$ and sends them to $\mathcal{V}$.

4. $\mathcal{P}$ uses the Eulidean algorithm to compute integers $e_1, e_2$ such that

$$e_1 d_1 + e_2 d_2 = d_3 \ , \ |e_1| < d_2 \ , \ |e_2| < d_1.$$

5. $\mathcal{P}$ computes

$$\widetilde{b}_1 := b_1^{e_1},\ \ \widetilde{b}_{1,2} := b_{1,2}^{e_2}$$

and sends them to $\mathcal{V}$. He generates non-interactive proofs for $\texttt{PoKE}[b_1,\ \widetilde{b}_1]$ and $\texttt{PoKE}[b_{1,2},\ \widetilde{b}_{1,2}]$. He sends these proofs to $\mathcal{V}$.

6. $\mathcal{V}$ verifies all of the proofs he receives in addition to the equation $\widetilde{b}_1\widetilde{b}_{1,2} \stackrel{?}{=} b_{1,3}$. He accepts the validity of the claim if and only if all of these proofs are valid. □

An important special case is where $\mathbf{gcd}(d_1, d_2) = 1$. In this case, Step 3 is redundant and hence, the proof size is smaller. We call this special case the Protocol for *Relatively Prime Discrete Logarithms* or `RelPrimeDLog` for short:

$$\mathcal{R}_{\texttt{RelPrimeDLog}}[(a_1, b_1),\ (a_2, b_2)] = \{((a_i, b_i \in \mathbb{G});\ d_i \in \mathbb{Z})\ :\ b_i = a_i^{d_i},\ \mathbf{gcd}(d_1, d_2) = 1\}.$$

**Protocol 4.2.** *Proof of Relatively Prime Discrete Logarithms* (`PoRelPrimeDLog`):

**Parameters:** $\mathbb{G} \stackrel{\$}{\leftarrow} \mathrm{GGen}(\lambda),\ g \in \mathbb{G}$.

**Inputs:** Elements $a_1, a_2, b_1, b_2 \in \mathbb{G}$.

**Claim:** The Prover possesses integers $d_1,\ d_2$ such that:

- $a_1^{d_1} = b_1,\ a_2^{d_2} = b_2$
- $\mathbf{gcd}(d_1, d_2) = 1$

1. The Prover $\mathcal{P}$ computes $b_{1,2} := a_1^{d_2}$ and sends it to the Verifier $\mathcal{V}$.

2. $\mathcal{P}$ computes a non-interactive proof for $\texttt{EqDLog}[(a_2, b_2),\ (a_1, b_{1,2})]$ and sends it to $\mathcal{V}$.

3. $\mathcal{P}$ uses the Euclidean algorithm to compute integers $e_1, e_2$ such that $e_1d_1 + e_2d_2 = 1$.

4. $\mathcal{P}$ computes

$$\widetilde{b}_1 := b_1^{e_1}\ \ ,\ \ \widetilde{b}_{1,2} := b_{1,2}^{e_2}$$

and sends them to $\mathcal{V}$.

5. $\mathcal{P}$ generates non-interactive proofs for $\texttt{PoKE}[b_1,\ \widetilde{b}_1]$ and $\texttt{PoKE}[b_{1,2},\ \widetilde{b}_{1,2}]$ and sends them to $\mathcal{V}$.

6. $\mathcal{V}$ verifies the equation $\widetilde{b}_1\widetilde{b}_{1,2} \stackrel{?}{=} a_1$ and the proofs for $\texttt{EqDLog}[(a_2, b_2),\ (a_1, b_{1,2})]$, $\texttt{PoKE}[b_1,\ \widetilde{b}_1]$ and $\texttt{PoKE}[b_{1,2},\ \widetilde{b}_{1,2}]$. He accepts the validity of the claim if and only if all of these proofs are valid. □

**Proposition 4.3.** *The Protocols* `PoGCD`, `PoRelPrimeDLog` *are arguments of knowledge for the relations* $\mathcal{R}_{\texttt{GCD}}$, $\mathcal{R}_{\texttt{RelPrimeDLog}}$ *respectively in the generic group model.*

*Proof.* Since the relation `RelPrimeDLog` is a special case of the relation `GCD`, it suffices to show that the protocol `PoGCD` is correct and sound. Furthermore, since we showed that `PoEqDLog` is correct and sound, we may assume without loss of generality that - with notations as in the protocol `PoGCD` -

$$a_1 = a_2 = a_3\ ,\ b_{1,2} = b_2\ ,\ b_{1,3} = b_3.$$

Now, the protocols $\texttt{PoKE}[b_3, b_1]$, $\texttt{PoKE}[b_3, b_2]$, $\texttt{PoKE}[a_1, b_3]$ imply that with overwhelming probability, the Prover $\mathcal{P}$ possesses integers $d_1, d_2, d_3$ such that

$$a_1^{d_1} = b_1\ ,\ a_1^{d_2} = b_2\ ,\ a_1^{d_3} = b_3\ ,\ \mathbf{gcd}(d_1, d_2) \equiv 0\ (\mathrm{mod}\ d_3).$$

Furthermore, the Prover $\mathcal{P}$ sends elements $\widetilde{b}_1, \widetilde{b}_2 \in \mathbb{G}$ such that $\widetilde{b}_1\widetilde{b}_2 = b_3$ along with the non-interactive proofs for $\texttt{PoKE}[b_1,\ \widetilde{b}_1]$, $\texttt{PoKE}[b_2,\ \widetilde{b}_2]$. Hence, with overwhelming probability, the Prover possesses integers $e_1, e_2$ such that

$$a_1^{e_1 d_1 + e_2 d_2} = b_1^{e_1} b_2^{e_2} = b_3 = a_1^{d_3}.$$

The low order assumption then implies that with overwhelming probability, $e_1 d_1 + e_2 d_2 = d_3$, which, in turn, implies that $\mathbf{gcd}(d_1, d_2) = d_3$. □

It is easy to see that the `PoGCD` may be combined with the protcol `PoMultPolyDLog` to provide an argument of knowledge for the relation

$$\mathcal{R}_{\texttt{LCM}}[(a_1, b_1), (a_2, b_2),\ (a_3, b_3)] = \{((a_i, b_i \in \mathbb{G});\ d_i \in \mathbb{Z})\ :\ b_i = a_i^{d_i},\ \mathbf{lcm}(d_1, d_2) = d_3\}.$$

This argument of knowledge can demonstrate that for data multisets $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$, we have

$$\mathcal{D}_3 = \mathcal{D}_1 \cup \mathcal{D}_2$$

by setting

$$d_i = \prod_{d \in \mathcal{D}_i} x \ \ (i = 1, 2, 3).$$

**An application:** Consider a setting where a client node stores succinct commitments to two large data sets/multisets $\mathcal{D}_1, \mathcal{D}_2$ and outsources these sets/mutisets to a server node that performs the necessary updates. If the client node, at any point, needs to verify that $\mathcal{D}_1, \mathcal{D}_2$ are disjoint, the server node can send a constant-sized proof using the protocol `PoRelPrimeDLog`. The proof is verifiable against the succinct commitments held by the client node.

### 4.1 Protocols for aggregated arguments of disjointness

We now use the protocols `AggKE-1` and `AggKE-2` and a few more techniques to generalize the protocol `RelPrimeDLog` to multiple discrete logarithms. Consider a setting where we have $n$ accumulators $\mathbf{Acc}_1, \cdots, \mathbf{Acc}_n$ instantiated in the same group $\mathbb{G}$ and with the common genesis state $g \in \mathbb{G}$. Let $\mathcal{D}_i$ denote the data inserted into $\mathbf{Acc}_i$ and let $A_i$ denote the accumulated digest of $\mathbf{Acc}_i$. Thus,

$$A_i = g^{\Pi(\mathcal{D}_i)}.$$

Suppose a Prover needs to demonstrate to a Verifier (with access to the accumulated digests) that the data sets/multisets $\mathcal{D}_i$ are pairwise disjoint, while keeping the communication complexity to a bare minimum. In particular, the Verifier should not need to access the data sets/multisets $\mathcal{D}_i$ which might be too large for the storage capacity of the verifying node. A straightforward way would be to provide the $\binom{n}{2}$ proofs of pairwise disjointness using the protocol `PoRelPrimeDLog`. But this would entail $\mathbf{O}(n^2)$ group elements and $\mathbf{O}(n^2)$ $\lambda$-bit integers, which we would like to avoid. Instead, we provide a protocol whereby the Prover can demonstrate the pairwise disjointness with a constant number of $\mathbb{G}$-elements and $2n + \mathbf{O}(1)$ $\lambda$-bit integers.

We call the next protocol the *Aggregated Knowledge of Relatively Prime Exponents* 1 or `AggRelPrimeDLog-1` for short. We provide an argument of knowledge for the relation:

$$\mathcal{R}_{\texttt{AggRelPrimeDLog-1}}[a,\ \mathcal{A}] = \left\{ \begin{array}{l} (a \in \mathbb{G},\ \mathcal{A} := (a_1, \cdots, a_n) \in \mathbb{G}^n); \\ (d_1, \cdots, d_n) \in \mathbb{Z}^n) : \\ a_i = a^{d_i}\ \forall\ i\ \ ,\ \ \mathbf{gcd}(d_i, d_j) = 1)\ \forall\ i \neq j \end{array} \right\}$$

The protocol rests on the following elementary lemma.

**Lemma 4.4.** *Let $d_1, \cdots, d_n$ be non-zero integers. Set*

$$D := \prod_{i=1}^{n} d_i\ ,\ \widehat{d}_i := \frac{D}{d_i}\ (i = 1, \cdots, n)\ ,\ \widehat{D} := \sum_{i=1}^{n} \widehat{d}_i.$$

*Then*

$$\mathbf{gcd}(d_i, d_j) = 1 \; \forall \; i \neq j \iff \mathbf{gcd}(D, \widehat{D}) = 1.$$

*Proof.* First, suppose there exists a pair $i, j$ such that $\mathbf{gcd}(d_i, d_j) > 1$. Then $\mathbf{gcd}(d_i, d_j)$ divides $\widehat{d}_k$ for every index $k$ and in particular, $\mathbf{gcd}(d_i, d_j)$ divides $\widehat{D}$. Hence, $\mathbf{gcd}(D, \widehat{D})$ is divisible by $\mathbf{gcd}(d_i, d_j)$.

Conversely, suppose $\mathbf{gcd}(d_i, d_j) = 1 \; \forall \; i \neq j$. Then for every index $i$, $\widehat{D} \equiv \widehat{d}_i \pmod{d_i}$ and hence, $\mathbf{gcd}(\widehat{D}, d_i) = \mathbf{gcd}(\widehat{d}_i, d_i) = 1$. Thus, $\mathbf{gcd}(D, \widehat{D}) = 1$. □

Recall that given integers $d_1, \cdots, d_n$ and elements $a, A \in \mathbb{G}$ such that

$$a^D = a^{\prod_{i=1}^{n} d_i} = A,$$

the **RootFactor** algorithm allows us to compute elements $a_i$ such that $a_i^{d_i} = A$ in runtime $\mathbf{O}(\log(D)\log(\log(D)))$, whereas the naïve approach would take runtime $\mathbf{O}(\log^2(D))$.. Thus, a Prover can compute the element

$$\widehat{A} := \prod_{i=1}^{n} a_i$$

in runtime $\mathbf{O}(\log(D)\log(\log(D)))$ with the **RootFactor** algorithm followed by $n$ group multiplications.

**Protocol 4.5.** *Proof of Aggregated Knowledge of Relatively Prime Discrete Logarithms* 1 (`PoAggRelPrimeDLog-1`) :

**Parameters:** $\mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda), \; g \in \mathbb{G}$.

**Inputs:** Element $a \in \mathbb{G}$, $(a_1, \cdots, a_n) \in \mathbb{G}^n$.

**Claim:** The Prover possesses integers $d_1, \cdots, d_n$ such that:

- $a^{d_i} = a_i$ for $i = 1, \cdots, n$.
- $\mathbf{gcd}(d_i, d_j) = 1$ for every pair $i \neq j$.

1. The Prover $\mathcal{P}$ computes the integers

$$D := \prod_{i=1}^{n} d_i \;\; , \;\; \widehat{D} := \sum_{i=1}^{n} \prod_{\substack{1 \leq j \leq n \\ j \neq i}} d_j.$$

2. $\mathcal{P}$ computes $A := a^D, \; \widehat{A} := a^{\widehat{D}}$ (the latter using the **RootFactor** algorithm) and sends $A, \widehat{A}$ to the Verifier $\mathcal{V}$.

3. $\mathcal{P}$ computes a non-interactive proof for $\texttt{MultPolyDLog}[a, \; (a_1, \cdots, a_n, A, \widehat{A}), \; (f, \widehat{f})]$ where

$$f(X_1, \cdots, X_{n+2}) := \big(\prod_{i=1}^{n} X_i\big) - X_{n+1} \;\; , \;\; \widehat{f}(X_1, \cdots, X_{n+2}) := \big(\sum_{i=1}^{n} \prod_{\substack{1 \leq j \leq n \\ j \neq i}} X_j\big) - X_{n+2}$$

and sends the proof to $\mathcal{V}$.

4. $\mathcal{P}$ generates a non-interactive proof for $\texttt{RelPrimeDLog}[(a, A), \; (a, \widehat{A})]$ and sends it to $\mathcal{V}$.

5. $\mathcal{V}$ verifies the three proofs and accepts the validity of the claim if and only if all proofs are valid. □

Recall that the protocol PoMultPolyDLog$[a,\ (a_1,\cdots,a_n,A,\widehat{A}),\ (f,\widehat{f})]$ contains PoAggKE-1$[a,\ (a_1,\cdots,a_n)]$ as a subprotocol. So the protocol PoAggRelPrimeDLog-1 demonstrates that there exist $n$ integers $d_i$ such that $a^{d_i}=a_i$ and the product $\prod_{i=1}^{n} d_i$ is relative prime with the integer given by the $(n-1)$-th elementary symmetric function

$$\sum_{i=1}^{n}\prod_{\substack{1\le j\le n\\ j\neq i}} d_j.$$

By the preceding lemma, this is equivalent to the integers $d_i$ being pairwise co-prime. Thus, the proof consists of a constant number of $\mathbb{G}$-elements and $2n+\mathbf{O}(1)$ $\lambda$-bit integers.

In the special case where $a$ is an element randomly generated by the oracle, the fractional root assumption implies that it is infeasible to compute any roots of $a$. Hence, the subprotocol of PoAggKE-1 or PoMultPolyDLog where the Prover computes the element

$$\widetilde{g} := g^{\sum_{i=1}^{n} d_i^p \gamma^i}$$

and sends it to the Verifier is redundant. So the proof would be a bit smaller and the Prover's computational burden would be substantially lower in this special case.

**Proposition 4.6.** *The protocol* PoAggRelPrimeDLog-1 *is an argument of knowledge for the relation* $\mathcal{R}_{\texttt{AggRelPrimeDLog}-1}$ *in the generic group model.*

*Proof.* (Sketch) The two MultPolyDLog proofs imply that with overwhelming probability, $\mathcal{P}$ possesses integers $D$, $\widehat{D}$, $d_1,\cdots,d_n$ such that

$$D=\prod_{i=1}^{n} d_i\ ,\ \widehat{D}=\sum_{i=1}^{n}\frac{D}{d_i}\ ,\ a^{d_i}=a_i\ \forall\ i\ ,\ a^D=A\ ,\ a^{\widehat{D}}=\widehat{A}.$$

Furthermore, the proof for RelPrimeDLog$[(a,A),\ (a,\widehat{A})]$ implies that $\mathbf{gcd}(D,\widehat{D})=1$. Hence, by the preceding lemma, the integers $d_i$ are pairwise co-prime. □

Given elements $a_1,a_2\in\mathbb{G}$ and equations

$$a_1^{d_1}=b_1,\cdots,a_1^{d_m}=b_m\ \ ,\ \ a_2^{e_1}=c_1,\cdots,a_2^{e_n}=c_n,$$

a Prover may provide a proof that he possesses the integers $d_1,\cdots,d_m,\ e_1,\cdots,e_n$ and that every pair $d_i,e_j$ is relatively prime. Clearly, the latter part is equivalent to the the integers $d:=\prod_{i=1}^{m} d_i$, $e:=\prod_{j=1}^{n} e_j$ being relatively prime. Our approach is to compute elements $B=a_1^d,\ C:=a_2^e$. We then use the protocols AggKE-1 and AggKE-2 to provide arguments of knowledge that $d,e$ are divisible by $\{d_1,\cdots,d_n\}$, $\{e_1,\cdots,e_n\}$ repectively. We then use the protocol RelPrimeDLog to show that $\mathbf{gcd}(d,e)=1$.

We call the next protocol the *Aggregated Knowledge of Relatively Prime Exponents* 2 or AggRelPrimeDLog-2 for short. We provide an argument of knowledge for the following relation:

$$\mathcal{R}_{\texttt{AggRelPrimeDLog-2}}[a_1,a_2,\ \mathcal{B},\mathcal{C}]=\left\{\begin{array}{l}((a_1,a_2)\in\mathbb{G}^2,\\ \mathcal{B}:=(b_1,\cdots,b_m)\in\mathbb{G}^m,\ \mathcal{C}:=(c_1,\cdots,c_n)\in\mathbb{G}^n);\\ ((d_1,\cdots,d_m)\in\mathbb{Z}^m,\ (e_1,\cdots,e_n)\in\mathbb{Z}^n)\ :\\ (b_i=a_1^{d_i}\ \bigwedge\ c_j=a_2^{e_j}\ \bigwedge\ \mathbf{gcd}(d_i,e_j)=1)\ \forall\ i,j\end{array}\right\}$$

**Protocol 4.7.** *Proof of Aggregated Knowledge of Relatively Prime Discrete Logarithms 2* (`PoAggRelPrimeDLog-2`) :

**Parameters:** $\mathbb{G} \stackrel{\$}{\leftarrow} \text{GGen}(\lambda),\ g \in \mathbb{G}$.
**Inputs:** Elements $a_1, a_2 \in \mathbb{G}$; Elements $\mathcal{B} = (b_1, \cdots, b_m),\ \mathcal{C} = (c_1, \cdots, c_n)$ of $\mathbb{G}^n$.
**Claim:** The Prover possesses integers $d_1, \cdots, d_m,\ e_1, \cdots, e_n$ such that:
- $a_1^{d_i} = b_i$ for $i = 1, \cdots, m$.
- $a_2^{e_j} = c_j$ for $j = 1, \cdots, n$.
- $\mathbf{gcd}(d_i, e_j) = 1$ for every pair $i, j$.

1. The Prover $\mathcal{P}$ computes
$$d := \prod_{i=1}^{m} d_m \ \ , \ \ e := \prod_{j=1}^{n} e_n.$$

2. $\mathcal{P}$ computes $B := a_1^d,\ \ C := a_2^e\ \in \mathbb{G}$ and sends $B, C$ to the Verifier $\mathcal{V}$.

3. $\mathcal{P}$ generates a non-interactive proof for
$$\texttt{MultPolyDLog}[a_1,\ (b_1, \cdots, b_m, B),\ (\prod_{i=1}^{m} X_i) - X_{m+1}]$$
and sends it to $\mathcal{V}$.

4. $\mathcal{P}$ generates a non-interactive proof for
$$\texttt{MultPolyDLog}[a_2,\ (c_1, \cdots, c_n, C),\ (\prod_{j=1}^{n} X_j) - X_{n+1}]$$
and sends it to $\mathcal{V}$.

5. $\mathcal{P}$ generates a non-interactive proof for $\texttt{RelPrimeDLog}[(a_1, B),\ (a_2, C)]$ and sends it to $\mathcal{V}$.

6. $\mathcal{V}$ accepts the validity of the claim if and only if all three proofs are valid. □

Note that the protocol $\texttt{PoMultPolyDLog}[a_1,\ (b_1, \cdots, b_m, B),\ (\prod_{i=1}^{m} X_i) - X_{m+1}]$ contains $\texttt{AggKE-1}[a_1,\ \mathcal{B}]$ as a subprotocol. Similarly, $\texttt{PoMultPolyDLog}[a_2,\ (c_1, \cdots, c_n, C),\ (\prod_{j=1}^{n} X_i) - X_{n+1}]$ contains $\texttt{AggKE-1}[a_2,\ \mathcal{C}]$ as a subprotocol. Thus, the proof consists of a constant number of $\mathbb{G}$-elements and $2(m+n) + \mathbf{O}(1)$ $\lambda$-bit integers.

**Proposition 4.8.** *The Protocol* `AggRelPrimeDLog-2` *is an argument of knowledge for the relation* $\mathcal{R}_{\texttt{AggRelPrimeDLog-2}}$ *in the generic group model.*

*Proof.* (Sketch) Since each of the subprotocols was shown to be correct and sound in the preceding seciton, this follows immediately. □

**An example:** We discuss an example of an application of this last protocol. Consider two families
$$\mathcal{A}_1 = (A_{1,1}, \cdots, A_{1,m}) \ \ , \ \ \mathcal{A}_2 = (A_{2,1}, \cdots, A_{2,n})$$
of accumulators instantiated using the same group $\mathbb{G}$ of hidden order. As usual, each data element is represented by a $\lambda$-bit prime. Let $g_1$, $g_2$ be the genesis states for all accumulators in $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

As usual, each data element is represented by a distinct $\lambda$-bit prime. Let $\mathcal{D}_{1,i}$ ($\mathcal{D}_{2,j}$) denote the data set/multiset inserted into the accumulator $A_{1,i}$ (respectively, $A_{2,j}$) and write

$$\mathcal{D}_1 := \bigcup_{i=1}^{m} \mathcal{D}_{1,i} \ , \ \mathcal{D}_2 = \bigcup_{j=1}^{n} \mathcal{D}_{2,j}.$$

Suppose a Verifier (with access to the accumulated digests) wants to verify that the unions are disjoint, i.e. $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$. A Prover with access to the data sts/multisets could simply provide a non-interactive proof for the protocol `AggRelPrimeDLog-2`$[(g_1, \mathcal{A}_1),\ (g_2, \mathcal{A}_2)]$. In particular, the proof can be verified without access to the data sets $\mathcal{D}_1, \mathcal{D}_2$.

### 4.2 Protocols for disjointness of multisets in a single accumulator

We now discuss a dual to the protocol `PoAggRelPrimeDLog-1`. Consider a setting where we have data sets/multisets $\mathcal{D}_1, \cdots, \mathcal{D}_n$ inserted into an accumulator. Let $A$ denote the accumulated digest, $w_i$ the witness for $\mathcal{D}_i$ and $d_i := \Pi(\mathcal{D}_i)$. Suppose a Prover needs to demonstrate that the multisets $\mathcal{D}_i$ are pairwise disjoint to a Verifier who has access to the witnesses $w_1, \cdots, w_n$ but not the data multisets. A straightforward approach would be to provide a proof for `RelPrimeDLog`$[(w_i, A),\ (w_j, A)]$ for each pair $i, j$. But such a proof would entail $\mathbf{O}(n^2)$ $\mathbb{G}$-elements and $\mathbf{O}(n^2)$ $\lambda$-bit integers, which is impractical for larger values of $n$.

Instead, we provide a protocol whereby the proof consists of a constant number of $\mathbb{G}$-elements and $n$ $\lambda$-bit integers. The protocol rests on two simple observations. First, note that for integers $d_1, \cdots, d_n$,

$$\mathbf{gcd}(d_i, d_j) = 1 \ \forall \ i \neq j \iff \prod_{i=1}^{n} d_i = \mathbf{lcm}(d_1, \cdots, d_n),$$

as can be easily proved by induction. Secondly, if an element $w \in \mathbb{G}$ can be expressed in the form

$$w = \prod_{i=1}^{n} w_i^{x_i}, \ \ (x_1, \cdots, x_n) \in \mathbb{Z}^n,$$

then

$$w^{\mathbf{lcm}(d_1, \cdots, d_n)} = A^k \quad \text{where } k := \sum_{i=1}^{n} x_i \frac{\mathbf{lcm}(d_1, \cdots, d_n)}{d_i}.$$

Furthermore, the Prover can efficiently compute the integers

$$d := \prod_{i=1}^{n} d_i = \mathbf{lcm}(d_1, \cdots, d_n) \ , \ \widehat{d_i} := \prod_{\substack{1 \leq j \leq n \\ j \neq i}} d_j \quad (i = 1, \cdots, n) \ \ , \ \ \widehat{d} := \sum_{i=1}^{n} \widehat{d_i}.$$

Now, $d$ is relatively prime to $\widehat{d}$ by lemma 4.4. Hence, the Prover can efficiently compute integers $e, \widehat{e}$ such that

$$de + \widehat{d}\widehat{e} = 1 \ , \ A^e (\prod_{i=1}^{n} w_i)^{\widehat{e}} = w.$$

In particular, since $\prod_{i=1}^{n} w_i$ is publicly computable, the Prover can demonstrate - with constant communication complexity - that $w$ is expressible as a product $\prod_{i=1}^{n} w_i^{x_i}$ where the $x_i$ are integers known to him. If the Prover can also demonstrate that

$$w^{\prod_{i=1}^{n} d_i} = A,$$

(with a subprotocol virtually identical to `MultPolyDLog`), then this implies that $\mathbf{lcm}(d_1, \cdots, d_n)$ divides the product $\prod_{i=1}^{n} d_i$, which forces equality between these two integers. In what follows, we provide an argument of knowledge for the relation

$$\mathcal{R}_{\texttt{AggRelPrimeDLog-3}}[(w_1, \cdots, w_n),\ A] = \left\{ \begin{array}{l} (A \in \mathbb{G},\ (w_1, \cdots, w_n)\ \in \mathbb{G}^n); \\ ((d_1, \cdots, d_n) \in \mathbb{Z}^n)\ : \\ w_i^{d_i} = A\ \forall\ i \\ \mathbf{gcd}(d_i, d_j) = 1\ \forall\ i, j : i \neq j \end{array} \right\}$$

**Protocol 4.9.** *Proof of Aggregated Knowledge of Relatively Prime Discrete Logarithms* 3 (`PoAggRelPrimeDLog-3`):

**Parameters:** $\mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda),\ g \in \mathbb{G}$.

**Inputs:** Elements $(w_1, \cdots, w_n) \in \mathbb{G}^n$, $A \in \mathbb{G}$

**Claim:** The Prover possesses integers $d_1, \cdots, d_n$ such that:
- $w_i^{d_i} = A$ for $i = 1, \cdots, n$.
- $\mathbf{gcd}(d_i, d_j) = 1$ for every pair $i \neq j$.

1. The Prover $\mathcal{P}$ computes the integers

$$D := \prod_{i=1}^{n} d_i\ \ ,\ \ \widehat{d_i} = \prod_{\substack{1 \leq j \leq n \\ j \neq i}} d_j\ (i = 1, \cdots, n)\ \ ,\ \ \widehat{D} := \sum_{i=1}^{n} \widehat{d_i}.$$

2. Using Shamir's trick, $\mathcal{P}$ computes an element $w \in \mathbb{G}$ such that $w^D = A$ and sends $w$ to the Verifier $\mathcal{V}$.

3. $\mathcal{P}$ uses the Euclidean algorithm to compute integers $e, \widehat{e}$ such that $eD + \widehat{e}\widehat{D} = \mathbf{gcd}(D, \widehat{D}) = 1$.

4. $\mathcal{P}$ computes

$$A_0 := A^e\ ,\ W := \big(\prod_{i=1}^{n} w_i\big)^{\widehat{e}}.$$

He sends $A_0, W$ to $\mathcal{V}$ along with non-interactive proofs for $\texttt{PoKE}[A,\ A_0]$ and $\texttt{PoKE}[(\prod_{i=1}^{n} w_i),\ W]$.

5. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\gamma$.

6. $\mathcal{P}$ computes

$$p := \texttt{NextPrime}(n\lambda)\ \ ,\ \ \widetilde{g} := g^{\sum_{i=1}^{n} d_i^p \gamma^i}$$

and sends $\widetilde{g}$ to $\mathcal{V}$.

7. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\ell \not\equiv 1 \pmod{p}$.

8. $\mathcal{P}$ computes

$$R := D \pmod{\ell}\ ,\ \breve{w} := w^{(D-R)/\ell}$$

and sends $\widehat{w}$ to $\mathcal{V}$.

9. $\mathcal{P}$ computes the integers

$$\widehat{r_i} := \widehat{d_i} \pmod{\ell}\ ,\ r_i := d_i \pmod{\ell}$$

and sends $(r_1, \cdots, r_n)$ to $\mathcal{V}$.

10. $\mathcal{P}$ computes the integers $\widetilde{q}$, $\widehat{q}$, $\widetilde{r}$, $\widehat{r}$ such that

$$\sum_{i=1}^{n} d_i^p \gamma^i = \widetilde{q}\ell + \widetilde{r} \ , \ \sum_{i=1}^{n} \widehat{d}_i \gamma^i = \widehat{q}\ell + \widehat{r} \ , \quad r, \widehat{r} \in [\ell].$$

11. $\mathcal{P}$ computes

$$\widehat{Q} := w^{\widehat{q}} \ , \ \breve{g} := g^{\widetilde{q}}$$

and sends $\widehat{Q}, \breve{g}$ to $\mathcal{V}$.

12. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\gamma_0$.

13. $\mathcal{P}$ computes the integers $\widehat{q}_0, \widehat{r}_0$ such that

$$\sum_{i=1}^{n} \widehat{d}_i \gamma^i = \widehat{q}_0 \ell + \widehat{r}_0 \ , \ \widehat{r}_0 \in [\ell]$$

He computes $\widehat{Q}_0 := w^{\widehat{q}_0}$ and sends it to $\mathcal{V}$.

14. $\mathcal{V}$ verifies that $(r_1, \cdots, r_n) \in [\ell]^n$ and independently computes

$$R := \prod_{i=1}^{n} r_i \pmod{\ell} \ , \ \widehat{r}_i = R r_i^{-1} \pmod{\ell} \ \ (i = 1, \cdots, n),$$

$$\widetilde{r} := \sum_{i=1}^{n} r_i^p \gamma^i \pmod{\ell} \ , \ \widehat{r} := \sum_{i=1}^{n} \widehat{r}_i \gamma^i \pmod{\ell} \ , \ \widehat{r}_0 := \sum_{i=1}^{n} \widehat{r}_i \gamma_0^i \pmod{\ell}.$$

15. $\mathcal{V}$ verifies the equations

$$(\widehat{Q})^\ell w^{\widehat{r}} \stackrel{?}{=} \prod_{i=1}^{n} w_i^{\gamma^i} \bigwedge (\widehat{Q}_0)^\ell w^{\widehat{r}_0} \stackrel{?}{=} \prod_{i=1}^{n} w_i^{\gamma_0^i} \bigwedge A_0 \cdot W \stackrel{?}{=} w \bigwedge (\breve{w})^\ell w^R \stackrel{?}{=} A \bigwedge (\breve{g})^\ell g^{\widetilde{r}} \stackrel{?}{=} \widetilde{g}$$

and the two PoKEs from Step 4. He accepts if and only if all five equations hold and the two PoKEs are valid. □

The proof consists of a constant number of $\mathbb{G}$-elements and $n + \mathbf{O}(1)$ $\lambda$-bit integers.

**Proposition 4.10.** *The protocol* PoAggRelPrimeDLog-3 *is an argument of knowledge for the relation* $\mathcal{R}_{\texttt{AggRelPrimeDLog}-3}$ *in the generic group model.*

*Proof.* (Sketch) The equations verified in step 15 imply that with overwhelming probability, the Prover possesses rationals $d_1, \cdots, d_n, D$ such that

$$d_i \equiv r_i \pmod{\ell} \ , \ w_i^{d_i} = A \ \ (\forall\, i), \ \ D \equiv R \pmod{l} \ , \ w^D = A.$$

In particular, $D \equiv R \equiv \prod_{i=1}^{n} r_i \equiv \prod_{i=1}^{n} d_i \pmod{\ell}$ and since the $\lambda$-bit prime $\ell$ is randomly generated after the Prover sends $w$, it follows that with overwhelming probability, $D = \prod_{i=1}^{n} d_i$. Furthermore, the equation $(\breve{g})^\ell g^{\widetilde{r}} = \widetilde{g}$ implies that with overwhelming probability, the Prover possesses a rational $\widetilde{d}$ such that

$$\widetilde{g} = g^{\widetilde{d}} \ , \ \widetilde{d} \equiv \widetilde{r} \equiv \sum_{i=1}^{n} r_i^p \gamma^i \equiv \sum_{i=1}^{n} d_i^p \gamma^i \pmod{\ell}.$$

Since the $\lambda$-bit prime $\ell$ is randomly generated after $\widetilde{g}$ has been sent by the Prover, it follows that with overwhelming probability,

$$\widetilde{d} = \sum_{i=1}^{n} d_i^p \gamma^i.$$

The fractional root assumption implies that $\widetilde{d}$ is an integer except with negligible probability. So $\sum_{i=1}^{n} d_i^p \gamma^i \in \mathbb{Z}$ and lemma 2.4 implies that with overwhelming probability, the $d_i$ are all integers.

Furthermore, since we have $A_0 \cdot W = w$ and the proofs for $\texttt{PoKE}[A,\ A_0]$ and $\texttt{PoKE}[(\prod_{i=1}^{n} w_i),\ W]$, it follows that, in particular, $w$ is expressible as a product

$$w = \prod_{i=1}^{n} w_i^{x_i}\ ,\ (x_1, \cdots, x_n) \in \mathbb{Z}^n.$$

Hence,

$$w^{\mathbf{lcm}(d_1,\cdots,d_n)} = A^k = w^{k \prod_{i=1}^{n} d_i}$$

for some integer $k$ known to the Prover. Now, the low order assumption implies that with overwhelming probability, $\mathbf{lcm}(d_1, \cdots, d_n)$ is divisible by the product $\prod_{i=1}^{n} d_i$, which forces equality between these two integers. Hence, the integers $d_i$ are pairwise co-prime. □

Next, we discuss a dual to the Protocol `AggRelPrimeDLog-2`. Given elements $B, C \in \mathbb{G}$ and subsets

$$\mathcal{B} = \{b_1, \cdots, b_m\} \in \mathbb{G}^m\ \ ,\ \ \mathcal{C} = \{c_1, \cdots, c_n\} \in \mathbb{G}^n,$$

an honest Prover may provide a proof that he possesses integers

$$\{d_1, \cdots, d_m\}\ ,\ \{e_1, \cdots, e_n\}$$

such that $b_i^{d_i} = B,\ c_j^{e_j} = C$ and every pair $d_i, e_j$ is relatively prime. We call this relation the *Aggregated Relatively Prime Discrete Logarithms* 4 or `AggRelPrimeDLog-4` for short. We provide an argument of knowledge for the following relation:

$$\mathcal{R}_{\texttt{AggRelPrimeDLog-4}}[\mathcal{B}, \mathcal{C}, B, C] = \left\{ \begin{array}{l} ((B, C) \in \mathbb{G}^2, \\ \mathcal{B} = (b_1, \cdots, b_m) \in \mathbb{G}^m\ ,\ \mathcal{C} = (c_1, \cdots, c_n) \in \mathbb{G}^n); \\ ((d_1, \cdots, d_m) \in \mathbb{Z}^m,\ (e_1, \cdots, e_n) \in \mathbb{Z}^n)\ : \\ (B = b_i^{d_i},\ C = c_j^{e_j}\ \bigwedge\ \mathbf{gcd}(d_i, e_j) = 1)\ \forall\ i, j \end{array} \right\}$$

**An example:** Consider the case where $B$, $C$ are accumulated digests for accumulators $\mathbf{Acc}_1$ and $\mathbf{Acc}_2$ respectively. Let $\mathcal{D}_1, \cdots, \mathcal{D}_m$ and $\mathcal{E}_1, \cdots, \mathcal{E}_m$ be data sets inserted into the two accumulators. Let $w_i,\ u_j$ denote the membership witnesses for $\mathcal{D}_i,\ \mathcal{E}_j$ and let $d_i,\ e_j$ denote the products of elements of $\mathcal{D}_i,\ \mathcal{E}_j$ respectively$(1 \leq i \leq m,\ 1 \leq j \leq n)$. Then

$$w_i^{d_i} = B\ ,\ u_j^{e_j} = C.$$

Suppose a Prover needs to prove the disjointness of the unions

$$\mathcal{D} := \bigcup_{i=1}^{m} \mathcal{D}_i\ \ ,\ \ \mathcal{E} := \bigcup_{j=1}^{n} \mathcal{E}_j$$

to a Verifier with access to the witnesses

$$\mathcal{W} := \{w_1, \cdots, w_m\} \ , \ \mathcal{U} := \{u_1, \cdots, u_n\}.$$

A straightforward approach would be to provide $m \cdot n$ distinct proofs that $\mathbf{gcd}(d_i, e_j) = 1$ for every pair $d_i, e_j$ using the protocol `PoRelPrimeDLog`. But such a proof would entail $\mathbf{O}(mn)$ elements of $\mathbb{G}$ in addition to $\mathbf{O}(mn)$ $\lambda$-bit integers. Instead, the Prover could simply send a non-interactive proof for the relation `AggRelPrimeDLog-4`$[(\mathcal{W}, B),\ (\mathcal{U}, C)]$. The proof consists of a constant number of $\mathbb{G}$-elements and $2(m+n) + \mathbf{O}(1)$ $\lambda$-bit integers.

**Protocol 4.11.** *Proof of Aggregated Knowledge of Relatively Prime Discrete Logarithms* 4 (`PoAggRelPrimeDLog-4`) :

**Parameters:** $\mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda),\ g \in \mathbb{G}$.
**Inputs:** Elements $B, C \in \mathbb{G},\ \mathcal{B} = (b_1, \cdots, b_m) \in \mathbb{G}^m,\ \mathcal{C} = (c_1, \cdots, c_n) \in \mathbb{G}^n$
**Claim:** The Prover possesses integers $d_1, \cdots, d_m,\ e_1, \cdots, e_n$ such that:
- $b_i^{d_i} = B$ for $i = 1, \cdots, m$.
- $c_j^{e_j} = C$ for $j = 1, \cdots, n$.
- $\mathbf{gcd}(d_i, e_j) = 1$ for every pair $i, j$.

1. Using Shamir's trick, $\mathcal{P}$ computes elements $b, c \in \mathbb{G}$ such that

$$b^{\mathbf{lcm}(d_1, \cdots, d_m)} = B \ , \ c^{\mathbf{lcm}(e_1, \cdots, e_n)} = C$$

and sends $b, c$ to the Verifier $\mathcal{V}$.

2. $\mathcal{P}$ generates non-interactive proofs for `AggKE-2`$[\mathcal{B},\ B]$ and `AggKE-2`$[\mathcal{C},\ C]$ and sends the proofs to $\mathcal{V}$.

3. $\mathcal{P}$ generates non-interactive proofs for `AggKE-1`$[b,\ \mathcal{B}]$ and `AggKE-1`$[c,\ \mathcal{C}]$ and sends them to $\mathcal{V}$.

4. $\mathcal{P}$ generates a non-interactive proof for `RelPrime`$[(b, B),\ (c, C)]$ and sends the proof to $\mathcal{V}$.

5. $\mathcal{V}$ verifies all of these proofs and accepts the validity of the claim if and only if all proofs are valid. □

Thus, the proof for `AggRelPrimeDLog-4` consists of a constant number of $\mathbb{G}$-elements and $2(m+n) + \mathbf{O}(1)$ $\lambda$-bit integers.

**Proposition 4.12.** *The protocol* `AggRelPrimeDLog-4` *is an argument of knowledge for the relation* $\mathcal{R}_{\texttt{AggRelPrimeDLog-4}}$ *in the generic group model.*

*Proof.* (Sketch) The proofs for `AggKE-2`$[\mathcal{B}, B]$ and `AggKE-1`$[b, \mathcal{B}]$ imply that, with overwhelming probability, the Prover possesses integers $d,\ d_1, \cdots, d_m,\ \widehat{d}_1, \cdots, \widehat{d}_m$ such that

$$b^d = B \ , \ b^{\widehat{d_i}} = b_i \ , \ b_i^{d_i} = B \ \forall\ i.$$

Similarly, the proofs for `AggKE-2`$[\mathcal{C},\ C]$ and `AggKE-1`$[c,\ \mathcal{C}]$ imply that with overwhelming probability, the Prover possesses integers $e,\ e_1, \cdots, e_n,\ \widehat{e}_1, \cdots, \widehat{e}_n$ such that

$$c^e = C \ , \ c^{\widehat{e}_j} = c_j \ , \ c_j^{e_j} = C \ \forall\ j.$$

The low order assumption implies that

$$d\widehat{d}_i = d \ \forall\ i \ \ , \ \ e\widehat{e}_j = e \ \forall\ j.$$

Lastly, the proof for `RelPrimeDLog`$[(b, B),\ (c, C)]$ implies that with overwhelming probability, $\mathbf{gcd}(d, e) = 1$. Hence, $\mathbf{gcd}(d_i, e_j) = 1\ \forall\ i, j$. □

# 5 Applications

## 5.1 Verifiably outsourcing storage

The protocols we have developed so far allow us to build a mechanism whereby a client can verifiably outsource data multisets to a server node. The client stores constant-sized commitments to these multisets and can query the server for information regarding these data sets. The server node, in turn, submits this information with proofs that can be publicly verified against the constant-sized commitments stored by the client.

As before, $\mathbb{G}$ is a group of hidden order in which we assume the adaptive root and strong-RSA assumptions to hold. The (fixed) element $g \in \mathbb{G}$ is a randomly generated element of $\mathbb{G}$. For a multiset $\mathcal{M}$, the commitment to $\mathcal{M}$ is given by the element

$$\texttt{Com}(g, \mathcal{M}) := g^{\Pi(\mathcal{M})} \in \mathbb{G}$$

where $\Pi(\mathcal{M})$ is the product of all elements of $\mathcal{M}$, with the appropriate multiplicities.

**Proof of storage:** To ask the server $\mathcal{P}$ to prove that he is storing the data multiset $\mathcal{M}$, the client $\mathcal{V}$ can generate a random element $g_1 \in \mathbb{G}$ and ask the server to provide the element

$$\texttt{Com}(g_1, \mathcal{M}) := g_1^{\Pi(\mathcal{M})} \in \mathbb{G}$$

along with a non-interactive proof for $\texttt{EqDLog}[(g, \texttt{Com}(g, \mathcal{M})),\ (g_1, \texttt{Com}(g_1, \mathcal{M}))]$. The communication complexity is constant and in particular, is independent of the size of $\mathcal{M}$.

If the client needs the proof of storage for multiple data multisets $\mathcal{M}_1, \cdots, \mathcal{M}_n$, they may proceed as follows:

1. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\gamma$ and a group element $g_1$.
2. The Prover $\mathcal{P}$ computes

$$h := \prod_{i=1}^{n} \texttt{Com}(g, \mathcal{M}_i)^{\gamma^i} \quad , \quad h_1 := g_1^{\sum_{i=1}^{n} \Pi(\mathcal{M}_i)\gamma^i}$$

and sends $h_1$ to the Verifier $\mathcal{V}$ along with a non-interactive proof for $\texttt{EqDLog}[(g, h),\ (g_1, h_1)]$.

3. $\mathcal{V}$ independently computes $h$ and accepts if and only if the $\texttt{EqDLog}$ proof is valid.

Thus, the proof is constant-sized irrespective of the number of data sets/multisets or their sizes.

**Updates:** When the data multiset is to be updated, the client sends the changes to the server. The server stores these changes and sends back the updated commitment to the multiset, along with a non-interactive proof of exponentiation ($\texttt{PoE}$) so that the client can efficiently verify that the new commitment is the correct one.

**Multiset sums:** For multisets $\mathcal{M}_1, \cdots, \mathcal{M}_n$, the server node can verifiably send the client the commitment $A_\Pi$ for the sum

$$\sum_{i=1}^{n} \mathcal{M}_i = \Big\{ \big( \sum_{i=1}^{n} \text{mult}(\mathcal{M}_i, x) \big) \times x : \ x \in \bigcup_{i=1}^{n} \texttt{Set}(\mathcal{M}_i) \Big\}$$

using the Protocol

$$\texttt{PoMultPolyDLog}[g,\ (A_1, \cdots, A_n, A_\Pi),\ (\prod_{i=1}^{n} X_i) - X_{n+1})].$$

The proof consists of a constant number of $\mathbb{G}$-elements and $2n$ $\lambda$-bit integers.

**Multiset differences:** For multisets $\mathcal{M}, \mathcal{N}$, the difference $\mathcal{M} \setminus \mathcal{N}$ has commitment

$$\texttt{Com}(g, \mathcal{M} \setminus \mathcal{N}) = g^{\frac{\Pi(\mathcal{M})}{\Pi(\mathcal{M} \cap \mathcal{N})}}.$$

So the Prover can combine the Protocols `PoGCD` and `PoMultPolyDLog` to verifiably send the commitment to $\mathcal{M} \setminus \mathcal{N}$.

### 5.1.1 Multiset intersections

Consider a setting where a client $\mathcal{V}$ who stores commitments $A_i := \texttt{Com}(g, \mathcal{M}_i)$ for data multisets $\mathcal{M}_1, \cdots, \mathcal{M}_n$ needs a commitment $\texttt{Com}(g, \mathcal{M}_\cap)$ to the intersection

$$\mathcal{M}_\cap := \bigcap_{i=1}^{n} \mathcal{M}_i.$$

In keeping with the rest of this paper, we would like to design a protocol that allows the Prover to do so while keeping the communication complexity to a minimum. Note that

$$d = \mathbf{gcd}(d_1, \cdots, d_n) \iff (d | d_i \; \forall \; i) \bigwedge \exists \, (e_1, \cdots, e_n) \in \mathbb{Z}^n : \sum_{i=1}^{n} e_i d_i = d.$$

Furthermore,

$$d = \mathbf{gcd}(d_1, \cdots, d_n) \iff \left(\frac{d_1}{d}, \cdots, \frac{d_n}{d}\right) \in \mathbb{Z}^n \bigwedge \mathbf{gcd}\left(\frac{d_n}{d}, \cdots, \frac{d_n}{d}\right) = 1.$$

Hence, by changing the base from $g$ to $a := g^d$, we can reduce this to the case where the GCD of the $n$ integers is 1. So, the Prover can verifiably send the commitment for $\mathcal{M}_\cap$ as follows:

**Protocol 5.1.** *Protocol for the intersection of multisets.*
**Parameters:** $\mathbb{G} \xleftarrow{\$} \text{GGen}(\lambda), \;\; g \in \mathbb{G}$.
**Inputs:** Commitments $A_i = \texttt{Com}(g, \mathcal{M}_i) := g^{\Pi(\mathcal{M}_i)}$ for multisets $\mathcal{M}_i$ whose elements are $\lambda$-bit primes; an element $A_\cap \in \mathbb{G}$
**Claim:** $A_\cap = \texttt{Com}(g, \bigcap_{i=1}^{n} \mathcal{M}_i) := g^{\Pi(\bigcap_{i=1}^{n} \mathcal{M}_i)}$.

1. The Prover $\mathcal{P}$ computes the integers

$$d_i := \frac{\Pi(\mathcal{M}_i)}{\Pi(\bigcap_{i=1}^{n} \mathcal{M}_i)} \;\; (i = 1, \cdots, n).$$

2. The Prover $\mathcal{P}$ generates a non-interactive proof for $\texttt{AggKE-1}[A_\cap, \; (A_1, \cdots, A_n)]$ and sends it to the Verifier $\mathcal{V}$.

3. $\mathcal{P}$ uses the Euclidean algorithm to compute integers $e_1, \cdots, e_n$ such that $\sum_{i=1}^{n} e_i d_i = 1$.

4. $\mathcal{P}$ computes the elements $\breve{A}_i := a^{e_i}$ $(i = 1, \cdots, n)$ and sends them to the $\mathcal{V}$.

5. $\mathcal{P}$ generates a non-interactive proof for $\texttt{MultPolyDLog}[A_\cap, \; (A_1, \cdots, A_n, \breve{A}_1, \cdots, \breve{A}_n), \; f]$ where

$$f(X_1, \cdots, X_{2n}) := \sum_{i=1}^{n} X_i X_{n+i} - 1$$

and sends the proof to $\mathcal{V}$.

6. $\mathcal{V}$ verifies the proofs and accepts if and only if they are all valid. □

This proof entails $n + \mathbf{O}(1)$ group elements and $\mathbf{O}(n)$ $\lambda$-bit integers. As before, we would like to keep the number of group elements constant. To this end, the Prover can sample $\lambda$-bit integers $\gamma$ until he finds one such that

$$\mathbf{gcd}(\sum_{i=1}^{n} d_i \gamma^i, d_1) = 1.$$

He can then send a non-interactive proof for

$$\texttt{RelPrimeDLog}[(A_\cap, \prod_{i=1}^{n} A_i^{\gamma^i}),\ (A_\cap, A_1)].$$

When the elements of the multisets $\mathcal{M}_i$ are all $\lambda$-bit primes, the integers $\Pi(\mathcal{M}_i)$ are $\lambda$-rough and hence, finding an appropriate $\gamma$ takes runtime $\mathbf{O}(1)$. This is because for an arbitrary $\gamma$ and any $\lambda$-bit prime $p$, the Schwartz-Zippel lemma implies that

$$\mathbf{Prob}\big(\sum_{i=1}^{n} d_i \gamma^i \equiv 0 \pmod{p}\big) = \texttt{negl}(\lambda).$$

As before, we say a multiset $\mathcal{M}$ is $\lambda$-**rough** if the integer $\Pi(\mathcal{M})$ is $\lambda$-rough or equivalently, if all elements of $\mathcal{M}$ are primes $> 2^{\lambda-1}$.

**Protocol 5.2.** *Protocol for the intersection of $\lambda$-rough multisets.*

**Parameters:** $\mathbb{G} \xleftarrow{\$} \text{GGen}(\lambda),\ g \in \mathbb{G}$.

**Inputs:** Commitments $A_i = \texttt{Com}(g, \mathcal{M}_i) := g^{\Pi(\mathcal{M}_i)}$ for $\lambda$-rough multisets $\mathcal{M}_i$ whose elements are $\lambda$-bit primes; an element $A_\cap \in \mathbb{G}$

**Claim:** $A_\cap = \texttt{Com}(g, \bigcap_{i=1}^{n} \mathcal{M}_i) := g^{\Pi(\bigcap_{i=1}^{n} \mathcal{M}_i)}$.

1. The Prover $\mathcal{P}$ computes the integers

$$d_i := \frac{\Pi(\mathcal{M}_i)}{\Pi(\bigcap_{i=1}^{n} \mathcal{M}_i)}\ \ (i = 1, \cdots, n).$$

2. $\mathcal{P}$ generates non-interactive proofs for $\texttt{AggKE-1}[A_\cap,\ (A_1, \cdots, A_n)]$, $\texttt{PoKE}[g,\ A_\cap]$ and sends them to the Verifier $\mathcal{V}$.

3. $\mathcal{P}$ samples $\lambda$-bit primes $\gamma$ until he finds one such that

$$\mathbf{gcd}(\prod_{i=1}^{n} d_i, \sum_{i=1}^{n} d_i \gamma^i) = 1.$$

He sends $\gamma$ to $\mathcal{V}$.

4. $\mathcal{P}$ computes $\widetilde{A} := g^{\prod_{i=1}^{n} d_i}$ and sends $\widetilde{A}$ to $\mathcal{V}$ along with a non-interactive proof for $\texttt{MultPolyDLog}[A_\cap,\ (A_1, \cdots, A_n, \widetilde{A}),\ (\prod_{i=1}^{n} X_i) - X_{n+1}]$.

5. $\mathcal{P}$ generates a non-interactive proof for the relation $\texttt{RelPrimeDLog}[(A_\cap, \widetilde{A}),\ (A_\cap, \prod_{i=1}^{n} A_i^{\gamma^i})]$ and sends it to $\mathcal{V}$.

6. $\mathcal{V}$ verifies all the proofs he receives and accepts if and only if they are all valid. □

Thus, the proof consists of a constant number of $\mathbb{G}$-elements and $2n + \mathbf{O}(1)$ $\lambda$-bit integers.

### 5.1.2 Multiset unions

The techniques in the last protocol also allow a server node to verifiably send over a commitment for the union of multisets $\mathcal{M}_1, \cdots, \mathcal{M}_n$. The proof that this commitment is valid can be publicly verified against the commitments $\texttt{Com}(g, \mathcal{M}_i)$ $(i = 1, \cdots, n)$ which the client stores. The basic idea here is as follows.

**Lemma 5.3.** *For integers $d_1, \cdots, d_n$, set $\widehat{d_j} := \prod_{\substack{1 \le i \le n \\ i \ne j}} d_i$ $(j = 1, \cdots, n)$. Then we have*

$$\mathbf{lcm}(d_1, \cdots, d_n) \cdot \mathbf{gcd}(\widehat{d_1}, \cdots, \widehat{d_n}) = \prod_{i=1}^{n} d_i.$$

We omit the proof since it is straightforward. Thus, the following are equivalent:

- $\mathbf{lcm}(d_1, \cdots, d_n) = d$
- $d_i \big| d\ \forall\ i$ and there exist integers $\widehat{e}_1, \cdots, \widehat{e}_n$ such that $\prod_{i=1}^{n} d_i = d \cdot \sum_{i=1}^{n} \widehat{e}_i \widehat{d}_i$.
- The rationals $\frac{d}{d_i}$ are integers and $\mathbf{gcd}(\frac{d}{d_1}, \cdots, \frac{d}{d_n}) = 1$.

**Protocol 5.4.** *Protocol for the union of $\lambda$-rough multisets.*

**Parameters:** $\mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda),\ g \in \mathbb{G}$.

**Inputs:** Commitments $A_i = \texttt{Com}(g, \mathcal{M}_i) := g^{\Pi(\mathcal{M}_i)}$ for $\lambda$-rough multisets $\mathcal{M}_i$; an element $A_\cup \in \mathbb{G}$

**Claim:** $A_\cup = \texttt{Com}(g, \bigcup_{i=1}^{n} \mathcal{M}_i) := g^{\Pi(\bigcup_{i=1}^{n} \mathcal{M}_i)}$.

1. The Prover $\mathcal{P}$ computes the integers

$$d := \Pi(\bigcup_{i=1}^{n} \mathcal{M}_i)\ ,\ \breve{d}_i := \frac{\Pi(\bigcup_{i=1}^{n} \mathcal{M}_i)}{\Pi(\mathcal{M}_i)}\ \ (i = 1, \cdots, n).$$

2. $\mathcal{P}$ generates a non-interactive proof for the relation $\texttt{AggKE-2}[(A_1, \cdots, A_n),\ A_\cup]$ and sends it to the Verifier $\mathcal{V}$.

3. $\mathcal{P}$ uses Bezout's algorithm to compute integers $\breve{e}_1, \cdots, \breve{e}_n$ such that

$$\sum_{i=1}^{n} \breve{e}_i \breve{d}_i = 1.$$

4. $\mathcal{P}$ computes the elements $\breve{A}_i := g^{\breve{e}_i}$ and sends them to $\mathcal{V}$.

5. $\mathcal{P}$ generates a non-interactive proof for $\texttt{MultPolyDLog}[g,\ (A_1, \cdots, A_n, \breve{A}_1, \cdots, \breve{A}_n, A_\cup),\ \breve{f}]$ where

$$\breve{f}(X_1, \cdots, X_{2n+1}) := X_{2n+1}\Big(\sum_{i=1}^{n} X_{n+i}\big(\prod_{\substack{1\le j\le n\\ j\neq i}} X_j\big)\Big) - \prod_{i=1}^{n} X_i$$

and sends the proof to $\mathcal{V}$.

6. $\mathcal{V}$ verifies all of the proofs he receives and accepts if and only if they are valid. □

The proof entails $n + \mathbf{O}(1)$ group elements and $2n + \mathbf{O}(1)$ $\lambda$-bit integers. As was the case with multiset intersections, when the multisets are $\lambda$-rough, the protocol can be modified so that the number of group elements is constant.

**Protocol 5.5.** *Protocol for the union of multisets.*

**Parameters:** $\mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda),\ g \in \mathbb{G}$.

**Inputs:** Commitments $A_i = \texttt{Com}(g, \mathcal{M}_i) := g^{\Pi(\mathcal{M}_i)}$ for $\lambda$-rough multisets $\mathcal{M}_i$; an element $A_\cup \in \mathbb{G}$

**Claim:** $A_\cup = \texttt{Com}(g, \bigcup_{i=1}^{n} \mathcal{M}_i) := g^{\Pi(\bigcup_{i=1}^{n} \mathcal{M}_i)}$.

1. The Prover $\mathcal{P}$ computes the integers

$$d := \Pi(\bigcup_{i=1}^{n} \mathcal{M}_i)\ ,\ \breve{d}_i := \frac{\Pi(\bigcup_{i=1}^{n} \mathcal{M}_i)}{\Pi(\mathcal{M}_i)}\ \ (i = 1, \cdots, n).$$

2. $\mathcal{P}$ generates a non-interactive proof for the relation $\texttt{AggKE-2}[(A_1, \cdots, A_n),\ A_\cup]$ and sends it to the Verifier $\mathcal{V}$.

3. $\mathcal{P}$ samples $\lambda$-bit integers $\gamma$ until he finds one such that

$$\mathbf{gcd}(d, \sum_{i=1}^{n} \breve{d}_i \gamma^i) = 1.$$

He sends $\gamma$ to $\mathcal{V}$.

4. $\mathcal{P}$ computes

$$\breve{A} := g^{\sum_{i=1}^{n} \breve{d}_i \gamma^i}$$

and sends $\breve{A}$ to $\mathcal{V}$ along with a non-interactive proof for $\texttt{MultPolyDLog}[g,\ (A_1, \cdots, A_n, A_\cup, \breve{A}), \breve{f}]$ where

$$\breve{f}(X_1, \cdots X_{n+2}) := X_{n+1}\Big(\sum_{i=1}^{n} \gamma^i\big(\prod_{\substack{1\le j\le n\\ j\neq i}} X_j\big)\Big) - X_{n+2}\prod_{i=1}^{n} X_i.$$

5. $\mathcal{P}$ generates a non-interactive proof for $\texttt{RelPrimeDLog}[(g, A_\cup),\ (g, \prod_{i=1}^{n} A_i^{\gamma^i})]$ and sends it to $\mathcal{V}$.

6. $\mathcal{V}$ verifies all the proofs he receives and accepts if and only if they are all valid. □

The proof consists of a constant number of $\mathbb{G}$-elements and $2n+\mathbf{O}(1)$ $\lambda$-bit integers. A server node storing the data multisets $\mathcal{M}_i$ can use this protocol to verifiably send the succinct commitment for the union $\bigcup_{i=1}^{n} \mathcal{M}_i$. The validity of this commitment can be verified against the commitments to the $\mathcal{M}_i$ held by the client.

**Disjointness:** The server node can verifiably demonstrate that the multisets $\mathcal{M}_1, \cdots, \mathcal{M}_n$ are pairwise disjoint using the protocol `AggRelPrimeDLog-1`$[g,\ (A_1, \cdots, A_n)]$. Similarly, for any subset $I \subseteq \{1, \cdots, n\}$ of indices, the server node can use the protocol `AggRelPrimeDLog-2` to verifiably show that the multisets

$$\widetilde{\mathcal{M}}_1 = \bigcup_{i \in I} \mathcal{M}_i \ , \ \widetilde{\mathcal{M}}_2 = \bigcup_{j \in \{1,\cdots,n\} \setminus I} \mathcal{M}_j$$

are disjoint. The proofs in both cases consist of $\mathbf{O}(1)$ group elements and $2n+\mathbf{O}(1)$ $\lambda$-bit integers.

**Underlying sets:** Given commitments to two multisets $\mathcal{M}$, $\mathcal{N}$, if the client needs to know whether the underlying sets coincide or if one is contained in the other, the server node can demonstrate this using the protocol `PoConSets` or `PoNonConSets`. The proof is constant-sized in each case.

Thus, to summarize, the protocols in this paper allow the server node to verifiably send the client succinct commitments to unions, intersections, sums (and combinations thereof) of multisets for which the client holds succinct commitments.

### 5.1.3 Frequencies of elements

Consider a setting where a client node needs to keep track of the occurrences of a certain keyword or certain blocks of keywords in the files that he possesses. If the client node suffers from a low storage capacity or weak computational power, he would prefer to outsource the files to an untrusted server node. As before, he stores succinct commitments to the data sets/multisets derived from hashing the files. This allows the server node storing the data to send publicly verifiable proofs about the occurrences of batches of elements in the data sets/multisets.

Let $\mathcal{M}_1, \cdots, \mathcal{M}_n$ be data multisets and let

$$A_i := \mathtt{Com}(g, \mathcal{M}_i) = g^{\Pi(\mathcal{M}_i)} \ \ (i = 1, \cdots, n)$$

be the commitments with the same base $g \in \mathbb{G}$. Suppose the server node storing the data for the client needs to identify the data multiset with the highest frequency of a data set $\mathcal{D}$. The protocols we have developed so far allows him to do so with a proof that the client can verify against the commitments $A_1, \cdots, A_n$.

The protocol hinges on the simple observation that for integers $d, d_1, d_2$, the following are equivalent:

1. For every prime $p$ dividing $d$, $\mathrm{val}_p(d_1) > \mathrm{val}_p(d_2)$.
2. There exists an integer $e$ such that $de$ divides $d_1$ and $\gcd(de, d_2) = e$.

**Protocol 5.6.** *Protocol for frequency of elements in multisets* 1.

**Parameters:** $\mathbb{G} \xleftarrow{\$} \mathrm{GGen}(\lambda)$, $g \in \mathbb{G}$.

**Inputs:** Commitments $A_i := g^{\Pi(\mathcal{M}_i)}$ for multisets $\mathcal{M}_i$; a data set $\mathcal{D}$.

**Claim:** Each element of $\mathcal{D}$ occurs with a higher frequency in $\mathcal{M}_1$ than in $\mathcal{M}_i$ $\forall\ i \geq 2$.

1. The Prover $\mathcal{P}$ computes

$$\widehat{A}_1 := g^{\Pi\left(\bigcup_{i=2}^{n} \mathcal{M}_i\right)}$$

and sends it to the verifier $\mathcal{V}$ along with a non-interactive proof for $\texttt{AggKE-2}[(A_2, \cdots, A_n), \widehat{A}_1]$.

2. The Prover computes the group elements

$$B_1 := g^{\prod_{x \in \mathcal{D}} x^{\text{mult}\left(\bigcup_{i=2}^{n} \mathcal{M}_i, x\right)}}, \; B_2 := B_1^{\Pi(\mathcal{D})} \in \mathbb{G}.$$

3. $\mathcal{P}$ sends $B_1, B_2$ to $\mathcal{V}$ along with a non-interactive $\texttt{PoE}$ for the equation $B_2 = B_1^{\Pi(\mathcal{D})}$.

4. $\mathcal{P}$ compute non-interactive proofs for $\texttt{PoKE}[B_2, \; A_1]$, $\texttt{PoGCD}[(g, \widehat{A}_1), \; (g, B_2), \; (g, B_1)]$ and sends them to $\mathcal{V}$.

5. $\mathcal{V}$ verifies the three proofs and accepts if and only if all of them are valid. □

Similarly, the following protocol allows the server to prove that every element of a certain data set $\mathcal{D}$ occurs with a lower frequency in $\mathcal{M}_1$ than in any of the multisets $\mathcal{M}_2, \cdots, \mathcal{M}_n$.

**Protocol 5.7.** *Protocol for frequency of elements in multisets* 2.

**Parameters:** $\mathbb{G} \xleftarrow{\$} \text{GGen}(\lambda), \; g \in \mathbb{G}$.
**Inputs:** Commitments $A_i := g^{\Pi(\mathcal{M}_i)}$ for multisets $\mathcal{M}_i$; a data set $\mathcal{D}$.
**Claim:** Each element of $\mathcal{D}$ occurs with a lower frequency in $\mathcal{M}_1$ than in $\mathcal{M}_i \; \forall \; i \geq 2$.

1. The Prover $\mathcal{P}$ computes

$$B_1 := g^{\prod_{x \in \mathcal{D}} x^{\text{mult}(\mathcal{M}_1, x)}}, \; B_2 = B_1^{\Pi(\mathcal{D})}.$$

2. $\mathcal{P}$ sends $B_1, B_2$ to $\mathcal{V}$ along with a non-interactive $\texttt{PoE}$ for the equation $B_2 = B_1^{\Pi(\mathcal{D})}$.

3. $\mathcal{P}$ generates non-interactive proofs for $\texttt{AggKE-1}[B_2, \; (A_2, \cdots, A_n)]$ and $\texttt{PoGCD}[(g, A_1), \; (g, B_2), \; (g, B_1)]$ and sends them to $\mathcal{V}$.

4. $\mathcal{V}$ verifies the three proofs and accepts if and only if they are all valid. □

The next two protocols are duals to the last two. They allow the server node to verifiably identify the multiset with the highest/lowest occurrence of a data set in a setting where the client stores the membership witnesses for the multisets with respect to a single accumulated digest.

**Protocol 5.8.** *Protocol for frequency of elements in multisets* 3.

**Parameters:** $\mathbb{G} \xleftarrow{\$} \text{GGen}(\lambda), \; g \in \mathbb{G}$.
**Inputs:** Membership witnesses $w_i$ for multisets $\mathcal{M}_i$ with respect to an accumulated digest $A$; a data set $\mathcal{D}$.
**Claim:** Each element of $\mathcal{D}$ occurs with a higher frequency in $\mathcal{M}_1$ than in $\mathcal{M}_i \; \forall \; i \geq 2$.

1. The Prover $\mathcal{P}$ uses Shamir's trick to compute an element $\breve{w}_1 \in \mathbb{G}$ such that

$$\breve{w}_1^{\Pi(\bigcup_{i=2}^{n} \mathcal{M}_i)} = A$$

and sends it to the Verifier $\mathcal{V}$.

2. $\mathcal{P}$ computes the elements

$$\widetilde{w}_1 := w_1^{\prod\limits_{x\in\mathcal{D}} x^{\text{mult}(\bigcup\limits_{i=2}^{n}\mathcal{M}_i,x)}},\ \widetilde{w}_1^{\Pi(\mathcal{D})}\ \in\ \mathbb{G}$$

and sends them to $\mathcal{V}$ along with a non-interactive PoE.

3. $\mathcal{P}$ generates a non-interactive proof for $\texttt{AggKE-1}[\breve{w}_1,\ (w_2,\cdots,w_n)]$ and sends it to $\mathcal{V}$.

4. $\mathcal{P}$ generates non-interactive proofs for $\texttt{PoGCD}[(\breve{w}_1, A),\ (w_1, \widetilde{w}_1^{\Pi(\mathcal{D})}),\ (w_1, \widetilde{w}_1)]$ and $\texttt{PoKE}[\widetilde{w}_1^{\Pi(\mathcal{D})}, A]$ and sends them to $\mathcal{V}$.

5. $\mathcal{V}$ verifies the three proofs and accepts if and only if they are all valid. □

**Protocol 5.9.** *Protocol for frequency of elements in multisets* 4.

**Parameters:** $\mathbb{G} \xleftarrow{\$} \text{GGen}(\lambda),\ g \in \mathbb{G}$.
**Inputs:** Membership witnesses $w_i$ for multisets $\mathcal{M}_i$ with respect to an accumulated digest $A$; a data set $\mathcal{D}$.
**Claim:** Each element of $\mathcal{D}$ occurs with a lower frequency in $\mathcal{M}_1$ than in $\mathcal{M}_i\ \forall\ i \geq 2$.

1. The Prover $\mathcal{P}$ computes an element $\widehat{w}_1 \in \mathbb{G}$ such that

$$\widehat{w}_1^{\Pi(\bigcap\limits_{i=2}^{n}\mathcal{M}_i)} := A$$

and sends $\widehat{w}_1$ to the Verifier $\mathcal{V}$ along with a non-interactive proof for $\texttt{AggKE-2}[(w_2,\cdots,w_n),\ \widehat{w}_1]$.

2. $\mathcal{P}$ computes

$$\widehat{A} := \widehat{w}_1^{\prod\limits_{x\in\mathcal{D}} x^{\text{mult}(\mathcal{M}_1,x)}},\ \widehat{A}^{\Pi(\mathcal{D})}\ \in\mathbb{G}$$

and sends them to $\mathcal{V}$ along with a non-interactive PoE for the exponentiation $\widehat{A}^{\Pi(\mathcal{D})}$.

3. $\mathcal{P}$ generates non-interactive proofs for $\texttt{PoKE}[\widehat{w}_1^{\Pi(\mathcal{D})},\ A]$ and $\texttt{PoGCD}[(w_1, A),\ (w_1, \widehat{A}^{\Pi(\mathcal{D})}),\ (w_1, \widehat{A})]$ and sends them to $\mathcal{V}$.

4. $\mathcal{V}$ verifies the three proofs and accepts if and only if they are all valid. □

### 5.1.4 Updates

As before, let $\mathcal{M}_1,\cdots,\mathcal{M}_n$ be multisets a client $\mathcal{V}$ outsources to a server node $\mathcal{P}$. The client stores succinct commitments

$$\texttt{Com}(g, \mathcal{M}_i) = g^{\Pi(\mathcal{M}_i)},$$

where $g$ is a randomly generated element of $\mathbb{G}$. The multisets are dynamic and hence, the commitments need to be updated in response to changes.

**Inserts:** When the multiset $\mathcal{M}_1$ changes to $\mathcal{M}_1 + \mathcal{M}_1'$, the server changes the commitment from $[g,\ g^{\Pi(\mathcal{M}_1)}]$ to $[g,\ g^{\Pi(\mathcal{M}_1+\mathcal{M}_1')}]$. He sends $g^{\Pi(\mathcal{M}_1+\mathcal{M}_1')}$ to the client along with a non-interactive PoE for the equation

$$(g^{\Pi(\mathcal{M}_1)})^{\Pi(\mathcal{M}_1')} = g^{\Pi(\mathcal{M}_1+\mathcal{M}_1')}.$$

**Deletes:** Let $\mathcal{M}_1'$ be a multiset contained in $\mathcal{M}_1$ and suppose $\mathcal{M}_1'$ is to be deleted from $\mathcal{M}_1$. Broadly there are three ways of handling deletes, each with some tradeoffs. We discuss them here.

1. The server node changes the commitment from $[g,\ g^{\Pi(\mathcal{M}_1)}]$ to $[g,\ g^{\Pi(\mathcal{M}_1 \setminus \mathcal{M}'_1)}]$. He sends $g^{\Pi(\mathcal{M}_1 \setminus \mathcal{M}'_1)}$ to the client along with a non-interactive `PoE` for the equation

$$(g^{\Pi(\mathcal{M}_1 \setminus \mathcal{M}'_1)})^{\Pi(\mathcal{M}'_1)} = g^{\Pi(\mathcal{M}_1)}.$$

While this is probably the simplest way of handling deletions, the computational burden is linear in the size of $\mathcal{M}_1 \setminus \mathcal{M}'_1$.

2. The server node changes the commitment from $[g,\ g^{\Pi(\mathcal{M}_1)}]$ to $[g^{\Pi(\mathcal{M}'_1)},\ g^{\Pi(\mathcal{M}_1)}]$. He sends $g_{\text{new}} := g^{\Pi(\mathcal{M}'_1)}$ to the client along with a non-interactive `PoE` for the equation

$$g^{\Pi(\mathcal{M}'_1)} = g_{\text{new}}.$$

The advantage is that the runtime complexity is $\mathbf{O}(\#\mathcal{M}'_1)$. The downside is that the client needs to keep track of the different bases for the commitments as opposed to a single base $g$. This increases the communication complexity and the Prover's work when the client asks for an argument of knowledge for any relation between the multisets $\mathcal{M}_1, \cdots, \mathcal{M}_n$.

3. The server node changes the commitment for $\mathcal{M}_1$ from $[g,\ g^{\Pi(\mathcal{M}_1)}]$ to $[g^{\Pi(\mathcal{M}'_1)},\ g^{\Pi(\mathcal{M}_1)}]$. Furthermore, he updates the commitments for $\mathcal{M}_i$ $(i = 2, \cdots, n)$ from $[g,\ g^{\Pi(\mathcal{M}_i)}]$ to $[g^{\Pi(\mathcal{M}'_1)},\ g^{\Pi(\mathcal{M}_i + \mathcal{M}'_1)}]$. He sends

$$g_{\text{new}} := g^{\Pi(\mathcal{M}'_1)}\ ,\ g^{\Pi(\mathcal{M}_i + \mathcal{M}'_1)}\ (i = 2, \cdots, n)$$

to the client along with the relevant non-interactive `PoE`s.

This preserves a common base for the $n$ commitments. The runtime complexity is $\mathbf{O}(n \cdot \#\mathcal{M}'_1)$, but the $n$ exponentiations are completely parallelizable.

### 5.2 Sharded stateless blockchains

We briefly discuss the concept of a stateless blockchain and the need for it. Currently, in every existing blockchain, every full node in the system needs to store the entire state of the blockchain in order to validate incoming transactions. This has already become cumbersome as the size of the state grows. To this end, [Tod16] suggested the concept of a *stateless* blockchain. In this proposed model, every node stores the data relevant to itself and the accumulated digest. The miners no longer need to store the state since the concerns of state storage and consensus are decoupled.

Recently, the idea of introducing shards has been gaining ground within the blockchain ecosystem. While a sharded blockchain would theoretically have a higher throughput, it would be less secure since each individual shard would have fewer validators than the entire blockchain. To prevent collusion, an idea that has been floated is to periodically rotate the validators assigned to the shards. Such a model makes stateless validation highly desirable since a stateful model would make it cumbersome for a validator to download the data for a new shard he gets assigned to.

While such a stateless model would drastically alleviate the problem of state bloat, the big tradeoff would be that in such a model, a node sending over transactions must also send over proofs attesting to the validity of these transactions. At the moment, the authentication data structure used by blockchains is that of a Merkle tree and the membership proofs are what we call Merkle branches/paths. Despite the several advantages that Merkle trees provide, a drawback is that the membership proofs cannot be batched or aggregated. Thus, a stateless model that continues to use Merkle trees as the accumulator would suffer from a bandwidth bottleneck.

To address this problem, [BBF19] etc. have proposed using a cryptographic accumulator with batchable/aggregable membership and non-membership proofs for a UTXO model. An accounts based model such as Ethereum could use a Vector Commitment with similar properties. In this regard, accumulators hinging on groups of unknown order have an important advantage over bilinear accumulators in that they are dynamic (constant runtime updates), have constant-sized public parameters and are transparent if the underlying hidden order group is a class group or a genus three Jacobian.

Consider the setting of a stateless sharded blockchain that, instead of a Merkle tree, hinges on a cryptographic accumulator instantiated with a hidden order group $\mathbb{G}$ ([BBF19]). Let $g$ be a randomly selected element of $\mathbb{G}$ and $\mathbf{S}_1, \cdots, \mathbf{S}_n$ the distinct shards. Let $\mathcal{D}_i$ denote the data in shard $\mathbf{S}_i$ and $\mathcal{D} := \bigcup_{i=1}^{n} \mathcal{D}_i$. Then the accumulated digest (the analog of the Merkle root hash) of $\mathbf{S}_i$ is given by

$$A_i := g^{\Pi(\mathcal{D}_i)}.$$

The accumulated digest of the blockchain is given by

$$A := g^{\Pi(\mathcal{D})}.$$

In order to demonstrate that the data sets in distinct shards are pairwise disjoint, a Prover (such as a miner or an untrusted server node) can provide a non-interactive proof for the relation $\mathcal{R}_{\texttt{AggRelPrimeDLog-1}}[a,\ (A_1, \cdots, A_n)]$. The proof consists of $\mathbf{O}(1)$ $\mathbb{G}$-elements and $2n + \mathbf{O}(1)$ $\lambda$-bit integers.

**Disjointness verifiable against membership witnesses:** Now consider the setting of a single shard. Let $\mathcal{V}_1, \cdots, \mathcal{V}_n$ be verifiers (such as light nodes) on the network. Let $\mathcal{E}_i$ denote the data set corresponding to $\mathcal{V}_i$. Suppose the verifiers need to verify that the data sets $\mathcal{E}_i$ are pairwise disjoint, but do not have access to the data sets outside their shards. A Prover $\mathcal{P}$ (such as a miner or an untrusted server node) can prove this pairwise disjointness as follows.

1. $\mathcal{V}_i$ $(i = 1, \cdots, n)$ broadcasts the membership witness $w_i$ for $\mathcal{D}_i$ to the other $n-1$ nodes $\mathcal{V}_j$ $(j \neq i)$.
2. $\mathcal{V}_i$ sends the data $\mathcal{D}_i$ to the prover $\mathcal{P}$.
3. $\mathcal{P}$ computes

$$d_i := \prod_{d \in \mathcal{D}_i} d \ \ (i = 1, \cdots, n)$$

and generates a non-interactive proof for the protocol $\texttt{PoAggRelPrimeDLog-3}[(w_1, \cdots, w_n),\ A]$, which he then broadcasts to the Verifiers.

The proof consists of $\mathbf{O}(1)$ $\mathbb{G}$-elements and $2n+\mathbf{O}(1)$ $\lambda$-bit integers. In particular, the Verifiers do not need access to the data sets $\mathcal{D}_i$ to verify this proof of disjointness.

**Pre-computation:** Although the argument systems in this paper have certain advantages such as transparency and succinctness, an important drawback is that the Prover's computational burden is rather high. This is primarily because exponentiations in hidden order groups are expensive. Furthermore, these exponentiations are conjectured to be almost purely sequential, which means there is no way to make them faster, aside from better hardware. However, the effective runtime in several cases can be reduced using pre-computations on the Prover's part. In most of the protocols, the most expensive part is the computation

$$\widetilde{g} := g^{\sum_{i=1}^{n} d_i^{n\lambda}\gamma^i}$$

where the $d_i$ are usually the products $\Pi(\mathcal{M}_i)$ of all elements of a data set/multiset and $\gamma$ is a $\lambda$-bit integer randomly generated by the Fiat-Shamir heuristic. We use this step in quite a few of the protocols in this paper with the sole purpose of demonstrating that the $d_i$ are integers rather than merely rationals. The Prover can reduce the effective runtime of this computation by pre-computing and storing the set

$$\{g^{2^i} \ : \ 1 \leq i \leq N\}$$

for an appropriately large integer $N$.

**Conclusion:** We hope that at least some the techniques will find more applications than what we have discussed in this paper. To this end, we have tried to keep the setting general, in the hope of finding a wider variety of applications. Several open questions remain, the foremost of which is whether the computational burden of the Prover can be alleviated. Furthermore, most of our proofs consist of a constant number of group elements and $\mathbf{O}(n)$ $\lambda$-bit integers, where $n$ is the number of comitted sets/multisets involved. It would be desirable to compress the proof sizes further so that the proofs are genuinely constant-sized, independent of $n$.

The protocols involving disjointness of the committed sets/multisets inherently rely on the Prover having access to the entirety of the data. It would be interesting to see if we can modify the protocols so that they are amenable to proof generation in a distributed setting.

A closely related line of research is to further explore class groups and Jacobians as candidates for transparent hidden order groups. The adaptive root assumption in these groups and the weaker assumptions such as low order and fractional root need further scrutiny.

**Acknowledgements:** The author thanks Benedikt Bünz and Dimitris Kolonelos for helpful feedback.

## References

[BBF19] D. Boneh, B. Bünz, B. Fisch, *Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains.* In Alexandra Boldyreva and Daniele Micciancio, editors, Advances in Cryptology – CRYPTO 2019, pages 561–586, Cham, 2019. Springer International Publishing.

[BBBF18] D. Boneh, J. Bonneau, B. Bünz and B. Fisch, *Verifiable delay functions.* In Hovav Shacham and Alexandra Boldyreva, editors, CRYPTO 2018, Part I, volume 10991 of LNCS

[BBF18] D. Boneh, B. Bünz, and B. Fisch, *A survey of two verifiable delay functions.* Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712

[BFS19] B. Bünz, B. Fisch, A. Szepieniec, *Transparent SNARKs from DARK Compilers*, Cryptology ePrint Archive, Report 2019/1229, 2019. https://eprint.iacr.org/2019/1229

[BCM05] E. Bangerter, J. Camenisch, and U. Maurer. *Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order.* In Serge Vaudenay, editor, PKC 2005, volume 3386 of LNCS, Springer, Heidelberg, January 2005.

[BH01] J. Buchmann and S. Hamdy. *A survey on IQ cryptography*, In Public-Key Cryptography and Computational Number Theory.

[BP97] N. Bari and B. Pfitzmann. *Collision-free accumulators and fail-stop signature schemes without trees.* In Walter Fumy, editor, EUROCRYPT'97, volume 1233 of LNCS, pages 480-494. Springer, Heidelberg, May 1997.

[BS96] Wieb Bosma and Peter Stevenhagen. *On the computation of quadratic 2-class groups* In Journal de Theorie des Nombres, 1996.

[CF13] D. Catalano and D. Fiore. *Vector commitments and their applications*, In Kaoru Kurosawa and Goichiro Hanaoka, editors, PKC 2013, volume 7778 of LNCS, pages 55-72. Springer, Heidelberg, February/March 2013.

[CFGKN20] M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, L. Nizzardo, *Vector Commitment Techniques and Applications to Verifiable Decentralized Storage*

[CSV20] W. Castryck, J, Sotakova, F. Vercauteren, *Breaking the decisional Diffie-Hellman problem for class group actions using genus theory*

[Can87] D. Cantor. *Computing in the Jacobian of a hyperelliptic curve. Mathematics of computation*, 48(177):95–101, 1987.

[Can94] D. Cantor. *On the analogue of the division polynomials for hyperelliptic curves*, Crelle's Journal, 447:91–146, 1994.

[DGS20] S. Dobson, S. Galbraith, B. Smith, *Trustless Groups of Unknown Order with Hyperelliptic Curves*, https://eprint.iacr.org/2020/196

[Fis18] B. Fisch. *Tight Proofs of Space and Replication.* In Y. Ishai and V. Rijmen, editors, EUROCRYPT 2019, Part II, volume 11477 of LNCS, pages 324-348. Springer, Heidelberg, May 2019.

[FS87] A. Fiat, A. Shamir, *How to prove yourself: Practical solutions to identification and signature problems.* In Andrew M. Odlyzko, editor, CRYPTO'86, volume 263 of LNCS, pages 186–194. Springer, Heidelberg, August 1987

[KPZ17] N. Katz, C. Papamanthou, Y. Zhang, *An Expressive (Zero-Knowledge) Set Accumulator*, 2017 IEEE European Symposium on Security and Privacy

[LLX07] J. Li, N. Li, and R. Xue, *Universal accumulators with efficient nonmembership proofs* In Jonathan Katz and Moti Yung, editors, ACNS 07, volume 4521 of LNCS, pages 253-269. Springer, Heidelberg, June 2007.

[LM18] R. Lai and G. Malavolta, *Optimal succinct arguments via hidden order groups.* Cryptology ePrint Archive, Report 2018/705, 2018.

[Ngu05] L. Nguyen. *Accumulators from bilinear maps and applications.* CT- RSA, 2005.

[Sho97] V. Shoup, *Lower bounds for discrete logarithms and related problems.* In Walter Fumy, editor, EUROCRYPT'97, volume 1233 of LNCS, pages 256266. Springer, Heidelberg, May 1997.

[Sut07] A. Sutherland, *Order Computations in Generic Groups*, MIT Thesis, 2007

[STY01] T. Sander, A. Ta-Shma, M. Yung, *Blind, auditable membership proofs*, In Yair Frankel, editor, FC 2000, volume 1962 of LNCS, pages 5371. Springer, Heidelberg, February 2001.

[Th20] S. Thakur, *Constructing hidden order groups using genus three Jacobians*, https://eprint.iacr.org/2020/348

[Tod16] Peter Todd. *Making UTXO Set Growth Irrelevant With Low-Latency Delayed TXO Commitments.* https://petertodd.org/ 2016/delayed-txo-commitments, May 2016.

[Wes19] B. Wesolowski, *Efficient verifiable delay functions.* In Yuval Ishai and Vincent Rijmen, editors, Advances in Cryptology – Eurocrypt 2019, pages 379–407, Cham, 2019. Springer International Publishing.

Steve Thakur
Axoni Research Group
New York City, NY
Email: stevethakur01@gmail.com

# A List of symbols/abbreviations

$\mathbb{G}$: a group of hidden order in which we assume the adaptive root and strong-RSA assumptions to hold.

$\lambda$: The security parameter

$\texttt{negl}(\lambda)$: The set of functions negligible in $\lambda$.

$[n]$: The set of integers $\{0, 1, \cdots, n-1\}$

$\texttt{NextPrime}(n)$: the smallest prime $\geq n$

PPT: Probabilistic Polynomial Time

$a \equiv_\lambda b$: The equivalence of $a, b \in \mathbb{G}$ with respect to the relation $\equiv_\lambda$

$\mathbb{Z}_{\mathcal{S}}$: The localization of $\mathbb{Z}$ at a set $\mathcal{S}$ of rational primes

$\mathbf{gcd}_S(a, b)$: The GCD of elements $a, b \in \mathbb{Z}_{\mathcal{S}}$ in the PID $\mathbb{Z}_{\mathcal{S}}$

$\mathbb{Z}_{(\lambda)}$: The localization of $\mathbb{Z}$ at the set of all rational primes $\leq 2^{\lambda-1}$.

$\mathcal{P}$: The Prover

$\mathcal{P}_{\text{mal}}$: A malicious Prover

$\mathcal{V}$: The Verifier

$\stackrel{\text{o.p.}}{\Longrightarrow}$ : Implies with overwhelming probability

$\texttt{Set}(\mathcal{M})$: The underlying set of a multiset $\mathcal{M}$

$\Pi(\mathcal{M})$: The product of all elements of a multiset $\mathcal{M}$

$\text{mult}(\mathcal{M}, x)$: The multiplicity of an element $x$ in a multiset $\mathcal{M}$ of $\lambda$-bit primes

$\text{val}_p(n)$: The largest integer $k$ such that $p^k$ divides $n$

PoE: Proof of Exponentiation ([Wes18], [BBF19])

PoKE: Proof of Knowledge of the Exponent ([BBF19])

# B List of Protocols:

The following is a list of the protocols in this paper and the relations that the protocols are arguments of knowledge for, in the generic group model. In each of the protocols, we may replace $\mathbb{Z}$ by the localization $\mathbb{Z}_{\mathcal{S}}$ at any set $\mathcal{S}$ of rational primes.

1. PoEqDLog (*Proof of equality of discrete logarithms*)

$$\mathcal{R}_{\texttt{EqDLog}}[(a_1, b_1),\ (a_2, b_2)] = \left\{ \begin{array}{l} ((a_1, b_1),\ (a_2, b_2) \in \mathbb{G}^2 \\ d \in \mathbb{Z}) : \\ (b_1, b_2) = (a_1^d, a_2^d) \end{array} \right\}$$

2. PoPolyDLog (*Proof of polynomial relation between (two) discrete logarithms*)

$$\mathcal{R}_{\texttt{PolyDLog}}[(a_1, b_1),\ (a_2, b_2),\ f] = \left\{ \begin{array}{l} ((a_1, b_1),\ (a_2, b_2) \in \mathbb{G}^2,\ f \in \mathbb{Z}[X]); \\ (d_1, d_2) \in \mathbb{Z}^2 : \\ b_1 = a_1^{d_1} \bigwedge b_1 = a_1^{d_1} \bigwedge d_2 = f(d_1) \end{array} \right\}$$

3. PoAggKE-1 (*Proof of aggregated knowledge of exponents*-1)

$$\mathcal{R}_{\texttt{AggKE-1}}[a,\ (a_1,\cdots,a_n)] = \left\{ \begin{array}{l} (a \in \mathbb{G},\ (a_1,\cdots,a_n) \in \mathbb{G}^n); \\ (d_1,\cdots,d_n) \in \mathbb{Z}^n) : \\ a_i = a^{d_i}\ \forall\ i \end{array} \right\}$$

4. `PoAggKE-2` (*Proof of aggregated knowledge of exponents*-2)

$$\mathcal{R}_{\texttt{AggKE-2}}[(a_1,\cdots,a_n),\ A] = \left\{ \begin{array}{l} ((a_1,\cdots,a_n) \in \mathbb{G}^n,\ A \in \mathbb{G}) \\ (d_1,\cdots,d_n) \in \mathbb{Z}^n) : \\ A = a_i^{d_i}\ \forall\ i \end{array} \right\}$$

5. `PoMultPolyDLog` (*Proof of multivariate polynomial relations between discrete logs*)

$$\mathcal{R}_{\texttt{MultPolyDLog}}[a,\ (b_1,\cdots,b_n),\ (f_1,\cdots,f_k)] = \left\{ \begin{array}{l} (a \in \mathbb{G},\ (b_1,\cdots,b_n) \in \mathbb{G}^n); \\ (f_1,\cdots,f_k) \in \mathbb{Z}[X_1,\cdots,X_n]^k; \\ (d_1,\cdots,d_n) \in \mathbb{Z}^n)\ : \\ b_i = a^{d_i}\ \forall\ i\ \bigwedge \\ f_j(d_1,\cdots,d_n) = 0\ \forall\ j \end{array} \right\}$$

6. `PoGCD` (*Proof of GCD*)

$$\mathcal{R}_{\texttt{GCD}}[(a_1,b_1),\ (a_2,b_2),(a_3,b_3)] = \{((a_i,b_i \in \mathbb{G});\ d_i \in \mathbb{Z})\ :\ b_i = a_i^{d_i},\ \mathbf{gcd}(d_1,d_2) = d_3\}$$

7. `PoRelPrimeDLog` (*Proof of relatively prime discrete logs*; special case of `PoGCD`)

$$\mathcal{R}_{\texttt{RelPrimeDLog}}[(a_1,b_1),\ (a_2,b_2)] = \{((a_i,b_i \in \mathbb{G});\ d_i \in \mathbb{Z})\ :\ b_i = a_i^{d_i},\ \mathbf{gcd}(d_1,d_2) = 1\}.$$

8. `PoAggRelPrimeDLog-1` (*Aggregated proof of relatively prime discrete logarithms*-1)

$$\mathcal{R}_{\texttt{AggRelPrimeDLog-1}}[a,\mathcal{A}] = \left\{ \begin{array}{l} \big(a \in \mathbb{G},\ \mathcal{A} := (a_1,\cdots,a_n) \in \mathbb{G}^n); \\ (d_1,\cdots,d_n) \in \mathbb{Z}^n\big) : \\ a_i = a^{d_i}\ \forall\ i\ \ ,\ \ \mathbf{gcd}(d_i,d_j) = 1)\ \forall\ i \neq j \end{array} \right\}$$

9. `PoAggRelPrimeDLog-2` (*Aggregated proof of relatively prime discrete logarithms*-2)

$$\mathcal{R}_{\texttt{AggRelPrimeDLog-2}}[a_1,a_2,\mathcal{B},\mathcal{C}] = \left\{ \begin{array}{l} \big((a_1,a_2) \in \mathbb{G}^2, \\ \mathcal{B} := (b_1,\cdots,b_m) \in \mathbb{G}^m,\ \mathcal{C} := (c_1,\cdots,c_n) \in \mathbb{G}^n); \\ ((d_1,\cdots,d_m) \in \mathbb{Z}^m,\ (e_1,\cdots,e_n) \in \mathbb{Z}^n\big)\ : \\ (b_i = a_1^{d_i}\ \bigwedge\ c_j = a_2^{e_j}\ \bigwedge\ \mathbf{gcd}(d_i,e_j) = 1)\ \forall\ i,j \end{array} \right\}$$

10. `PoAggRelPrimeDLog-3` (*Aggregated proof of relatively prime discrete logarithms*-3)

$$\mathcal{R}_{\texttt{AggRelPrimeDLog-3}}[(w_1,\cdots,w_n),A] = \left\{ \begin{array}{l} \big(A \in \mathbb{G},\ (w_1,\cdots,w_n)\ \in \mathbb{G}^n); \\ ((d_1,\cdots,d_n) \in \mathbb{Z}^n\big)\ : \\ w_i^{d_i} = A\ \forall\ i\ \bigwedge \\ \mathbf{gcd}(d_i,d_j) = 1\ \forall\ i,j : i \neq j \end{array} \right\}$$

11. `PoAggRelPrimeDLog-4` (*Aggregated proof of relatively prime discrete logarithms*-4)

$$\mathcal{R}_{\texttt{AggRelPrimeDLog-4}}[\mathcal{B},\mathcal{C},B,C] = \left\{ \begin{array}{l} \big((B,C) \in \mathbb{G}^2, \\ \mathcal{B} = (b_1,\cdots,b_m) \in \mathbb{G}^m,\ \mathcal{C} = (c_1,\cdots,c_n) \in \mathbb{G}^n); \\ ((d_1,\cdots,d_m) \in \mathbb{Z}^m,\ (e_1,\cdots,e_n) \in \mathbb{Z}^n\big)\ : \\ (B = b_i^{d_i},\ C = c_j^{e_j}\ \bigwedge\ \mathbf{gcd}(d_i,e_j) = 1)\ \forall\ i,j \end{array} \right\}$$

12. *Protocol for the intersection of multisets*

$$\mathcal{R}_{\cap}[a,\ (A_1,\cdots,A_n),\ A_{\cap}] = \left\{ \begin{array}{l} (a \in \mathbb{G}; \\ (A_1,\cdots,A_n) \in \mathbb{G}^n\ ,\ A_{\cap} \in \mathbb{G}; \\ ((d_1,\cdots,d_n,d_{\cap}) \in \mathbb{Z}^{n+1}) : \\ A_i = g^{d_i}\ \forall\ i\ ,\ A_{\cap} = g^{d_{\cap}} \\ \mathbf{gcd}(d_1,\cdots,d_n) = d_{\cap} \end{array} \right\}$$

13. *Protocol for the union of multisets*

$$\mathcal{R}_{\cup}[a,\ (A_1,\cdots,A_n),\ A_{\cup}] = \left\{ \begin{array}{l} (a \in \mathbb{G}; \\ (A_1,\cdots,A_n) \in \mathbb{G}^n)\ ,\ ,A_{\cup} \in \mathbb{G}; \\ ((d_1,\cdots,d_n,d_{\cup}) \in \mathbb{Z}^{n+1}) : \\ A_i = g^{d_i}\ \forall\ i\ ,\ A_{\cup} = g^{d_{\cup}} \\ \mathbf{lcm}(d_1,\cdots,d_n) = d_{\cup} \end{array} \right\}$$

14. `PoConSets`/ `PoNonConSets` (*Protocol for the containment/non-containment of the underlying sets*)

If the Verifier has access to the commitments $\texttt{Com}(g, \mathcal{M}_i) = g^{\Pi(\mathcal{M}_i)}$ for multisets $\mathcal{M}_1, \mathcal{M}_2$, the Prover can verifiably show whether $\texttt{Set}(\mathcal{M}_1) \subseteq \texttt{Set}(\mathcal{M}_2)$ or $\texttt{Set}(\mathcal{M}_1) \nsubseteq \texttt{Set}(\mathcal{M}_2)$.

15. *Protocol for the highest/lowest frequency of elements in multisets*

If the Verifier has access to the commitments $\texttt{Com}(g, \mathcal{M}_i) = g^{\Pi(\mathcal{M}_i)}$ to multisets $\mathcal{M}_1, \cdots, \mathcal{M}_n$ and a data set $\mathcal{D}$, the Prover can show that every element of $\mathcal{D}$ occurs with a higher/lower frequency in $\mathcal{M}_1$ than in any $\mathcal{M}_i$ for $i \geq 2$.

Similarly, given a single accumulated digest $A$ and witnesses $w_i$ for multisets $\mathcal{M}_i$ such that $w_i^{\Pi(\mathcal{M}_i)} = A$, the Prover can show that every element of $\mathcal{D}$ occurs with a higher/lower frequency in $\mathcal{M}_1$ than in any $\mathcal{M}_i$ for $i \geq 2$.

16. `PoAggEqDLog` (*Proof of aggregated equality of discrete logarithms*)

$$\mathcal{R}_{\texttt{AggEqDLog}}[(a,b),\ (\mathcal{A},\ \mathcal{B})] = \left\{ \begin{array}{l} ((a,b) \in \mathbb{G}^2 \\ \mathcal{A} = (a_1,\cdots,a_n),\ \mathcal{B} = (b_1,\cdots,b_n) \in \mathbb{G}^n); \\ d \in \mathbb{Z}) : \\ b_i = a_1^d\ \forall\ i \end{array} \right\}$$

17. `PoEqDLogPairs` (*Proof of equality of discrete log pairs*)

$$\mathcal{R}_{\texttt{EqDLogPairs}}[(a_1,\mathcal{B}),\ (a_2,\mathcal{C})] = \left\{ \begin{array}{l} (a_1, a_2 \in \mathbb{G}; \\ (b_1,\cdots,b_n),\ (c_1,\cdots,c_n) \in \mathbb{G}^n); \\ (d_1,\cdots,d_n) \in \mathbb{Z}^n)\ : \\ b_i = a_1^{d_i}\ ,\ c_i = a_2^{d_i}\ \forall\ i \end{array} \right\}$$

## C More protocols

In what follows, we provide an argument of knowledge for the relation

$$\mathcal{R}_{\texttt{EqDLogPairs}}[(a_1,\mathcal{B}),\ (a_2,\mathcal{C})] = \left\{ \begin{array}{l} ((a_1,a_2)\in\mathbb{G}^2 \\ \mathcal{B}=(b_1,\cdots,b_n)\ ,\ \mathcal{C}=(c_1,\cdots,c_n)\ \in\mathbb{G}^n); \\ (d_1,\cdots,d_n)\in\mathbb{Z}^n): \\ b_i=a_1^{d_i}\ ,\ c_i=a_2^{d_i}\ \forall\ i \end{array} \right\}$$

**Protocol C.1.** *Proof of equalities of pairs of discrete logarithm* (`PoEqDLogPairs`) :

**Parameters:** $\mathbb{G}\xleftarrow{\$}\text{GGen}(\lambda),\ g\in\mathbb{G}$.

**Inputs :** $a_1,a_2\in\mathbb{G}\ ,(b_1,\cdots,b_n),\ (c_1,\cdots,c_n)\ \in\ \mathbb{G}^n$.

**Claim:** The Prover possesses integers $d_1,\cdots,d_n$ such that $a_1^{d_i}=b_i,\ a_2^{d_i}=c_i$.

1. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\gamma$.
2. The Prover $\mathcal{P}$ computes $p:=\texttt{NextPrime}(n\lambda)\ \ ,\ \ \widetilde{g}:=g^{\sum\limits_{i=1}^{n}d_i^{n\lambda}\gamma^i}$ and sends $\widetilde{g}$ to the Verifier $\mathcal{V}$.
3. $\mathcal{P}$ computes the elements

$$B:=\prod_{i=1}^{n}b_i^{\gamma^i},\ \ C:=\prod_{i=1}^{n}c_i^{\gamma^i}\ \in\ \mathbb{G}.$$

4. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\ell\not\equiv 1\pmod p$.
5. $\mathcal{P}$ computes $r_i:=d_i\pmod\ell$ and the integers $q,\widetilde{q},r,\widetilde{r}$ such that

$$\sum_{i=1}^{n}d_i^p\gamma^i=\widetilde{q}l+\widetilde{r}\ ,\ \sum_{i=1}^{n}d_i\gamma^i=ql+r,\quad r,\widetilde{r}\in[\ell]$$

6. $\mathcal{P}$ computes $\breve{g}:=g^{\widetilde{q}}$, $Q:=a_1^q$ and sends $\breve{g}$, $Q$, $(r_1,\cdots,r_n)$ to $\mathcal{V}$.
7. $\mathcal{P}$ computes a non-interactive proof for $\texttt{EqDLog}[(a_1,B),\ (a_2,C)]$ and sends it to $\mathcal{V}$.
8. The Fiat-Shamir heuristic generates a $\lambda$-bit prime $\gamma_0$.
9. $\mathcal{P}$ computes integers $q_0,r_0$ such that

$$\sum_{i=1}^{n}d_i\gamma_0^i=q_0l+r_0\ \ ,\quad r_0\in[\ell].$$

He computes $Q_0:=a_1^{q_0}$ and sends $Q_0$ to $\mathcal{V}$.

10. $\mathcal{V}$ verifies that $(r_1,\cdots,r_n)\in[\ell]^n$ and independently computes

$$\widetilde{r}:=\sum_{i=1}^{n}r_i^p\gamma^i\pmod\ell\ ,\ r:=\sum_{i=1}^{n}r_i\gamma^i\pmod\ell\ ,\ r_0:=\sum_{i=1}^{n}r_i\gamma_0^i\pmod\ell.$$

11. $\mathcal{V}$ independently computes the elements

$$B:=\prod_{i=1}^{n}b_i^{\gamma^i},\ \ B_0:=\prod_{i=1}^{n}b_i^{\gamma_0^i},\ \ C:=\prod_{i=1}^{n}c_i^{\gamma^i}\ \in\ \mathbb{G}.$$

12. $\mathcal{V}$ verifies the equations

$$Q^{\ell} a_1^r \stackrel{?}{=} B \bigwedge (Q_0)^{\ell} a_1^{r_0} \stackrel{?}{=} B_0 \bigwedge (\breve{g})^{\ell} g^{\widetilde{r}} \stackrel{?}{=} \widetilde{g}.$$

He then accepts the validity of the claim if and only if the proof for $\texttt{EqDLog}[(a_1, B),\ (a_2, C)]$ is valid and all equations hold. □

We now show how to generalize the last protocol for multiple discrete logarithms while keeping the communication complexity constant-sized and independent of the number of discrete logarithms. We call the following protocol the *Aggregated Equality of Discrete Logarithms* or `AggEqDLog` for short. We provide an argument of knowledge for the following relation:

$$\mathcal{R}_{\texttt{AggEqDLog}}[(a, b),\ (\mathcal{A},\ \mathcal{B})] = \left\{ \begin{array}{l} (\mathcal{A} = (a_1, \cdots, a_n),\ \mathcal{B} = (b_1, \cdots, b_n) \in \mathbb{G}^n); \\ d \in \mathbb{Z}) : \\ b_i = a_1^d \ \forall\ i \end{array} \right\}$$

**Protocol C.2.** *Proof of Aggregated Equal Discrete Logarithms* (`PoAggEqDLog`) :

**Parameters:** $\mathbb{G} \stackrel{\$}{\leftarrow} \text{GGen}(\lambda),\ g \in \mathbb{G}$.

**Inputs :** $a, b \in \mathbb{G}$, $(a_1, \cdots, a_n) \in \mathbb{G}^n$, $(b_1, \cdots, b_n) \in \mathbb{G}^n$.

**Claim:** The Prover possesses an integer $d$ such that $a^d = b$ and $a_i^d = b_i$ for $i = 1, \cdots, n$ .

1. The Fiat-Shamir heuristic generates a $\lambda$-bit integer $\gamma$.
2. The Prover $\mathcal{P}$ computes the elements

$$\widetilde{a} := \prod_{i=1}^{n} a_i^{\gamma^i},\ \ \widetilde{b} := \prod_{i=1}^{n} b_i^{\gamma^i} \quad \in \mathbb{G}.$$

3. $\mathcal{P}$ generates a non-interactive proof for $\texttt{EqDLog}[(a, b),\ (\widetilde{a}, \widetilde{b})]$ and sends it to the Verifier $\mathcal{V}$.
4. $\mathcal{V}$ independently computes the elements $\widetilde{a}, \widetilde{b}$ and accepts if and only if the proof for $\texttt{EqDLog}[(a, b),\ (\widetilde{a}, \widetilde{b})]$ is valid. □

Thus, the proof consists of four $\mathbb{G}$-elements and one $\lambda$-bit integer. In particular, it is constant-sized and independent of the cardinalities $|\mathcal{A}|$, $|\mathcal{B}|$.

**Proposition C.3.** *The protocol* `AggEqDLog` *is an argument of knowledge for the relation* $\mathcal{R}_{\texttt{AggEqDLog}}$ *in the generic group model.*

*Proof.* (Sketch) With notations as in in the Protocol, $\mathcal{V}$ accepts if and only if $\mathcal{P}$ proves possession of an integer $d$ such that

$$a^d = b,\ \ \widetilde{a}^d = \widetilde{b}$$

through the Protocol $\texttt{PoEqDLog}[(a, b),\ (\widetilde{a}, \widetilde{b})]$. Now, since the challenge $\gamma$ is randomly generated, it follows that

$$\textbf{Prob}\left( (a_1^d, \cdots, a_n^d) \neq (b_1, \cdots, b_n) \ \middle|\ \widetilde{a}^d = \widetilde{b} \right) \in \texttt{negl}(\lambda).$$

Since the Protocol $\texttt{EqDLog}[(a, b),\ (\widetilde{a}, \widetilde{b})]$ is secure under the strong-RSA and adaptive root assumptions, it follows that the Protocol $\texttt{AggEqDLog}[(a_1, b_1),\ (a_2, b_2)]$ is also secure under these assumptions. □