Non-Interactive Verifiable Aggregation

Ojaswi Acharya
University of Massachusetts Amherst
MA, USA
oacharya@umass.edu

Suvasree Biswas
The George Washington University
DC, USA
suvasree@gwu.edu

Weiqi Feng
University of Massachusetts Amherst
MA, USA
weiqifeng@umass.edu

Adam O'Neill
University of Massachusetts Amherst
MA, USA
adamoneill@umass.edu

Arkady Yerukhimovich
The George Washington University
DC, USA
arkady@gwu.edu

ABSTRACT

Consider a weak analyst that wishes to outsource data collection and computation of aggregate statistics over a a potentially large population of (also weak) clients to a powerful server. For flexibility and efficiency, we consider *public-key* and *non-interactive* protocols, meaning the clients know the analyst's public key but do not share secrets, and each client sends at most one message. Furthermore, the final step should be *silent*, whereby the analyst simply downloads the (encrypted) result from the server when needed. To capture this setting, we define a new primitive we call *Non-Interactive Verifiable Aggregation* (NIVA). We require both privacy and robustness for a NIVA protocol to be deemed secure. Namely, our security notion for NIVA ensures that the clients' data remains private to both the server and the analyst, while also ensuring that malicious clients cannot skew the results by providing faulty data.

We propose a secure NIVA protocol, which we call PEAR (for **P**rivate, **E**fficient, **A**ccurate, **R**obust), which can validate inputs according to any NP validity rule. PEAR is based on a novel combination of *functional encryption for inner-products* (Abdalla *et al.*, PKC 2015) and *fully-linear probabilistically-checkable proofs* (Boneh *et al.*, Crypto 2019). We emphasize that PEAR is non-interactive, public-key, and makes black-box use of the underlying cryptographic primitives. Additionally, we devise substantial optimizations of PEAR for practically-relevant validity rules. Finally, we implement PEAR to show feasibility for such validity rules, conducting a thorough performance evaluation. In particular, we compare PEAR to two more straightforward or "off-the-shelf" NIVA protocols and show performance gains, demonstrating the merit of our new approach. The bottleneck in our protocol comes from the fact that we require the underlying IPFE scheme to be "unrestricted" over a large field. As more efficient such schemes are developed, they can be immediately be plugged into PEAR for further gains.

KEYWORDS

Secure aggregation, Input validation, Inner-product functional encryption, Fully-linear PCPs

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–18

https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

1.1 Background and Motivation

Motivating applications. There are numerous applications in which an 'analyst' wishes to conduct a study on a large population of 'clients' to learn aggregate statistics, but input collection from the clients is too burdensome for the analyst. Therefore, the analyst wishes to *outsource* the collection and aggregation of the data to a (potentially untrusted) 'server' who aggregates the inputs and returns the result to the analyst. Here are some examples:

- **Population studies:** Suppose that think tanks or policy-makers wish to collect statistics such as average age, *etc.* about a certain segment of the population. The think tank may not have the intensive computational resources to collect all the survey data, and may choose to outsource this to the cloud. The cloud server can then collect and aggregate the data and give the result to the think tank.
- **Sensor networks:** Suppose a researcher wishes to deploy sensors to measure the environment (*e.g.*, temperature readings) in some remote setting. Rather than set up their own network to collect the readings, the researcher can use a cloud server to collect and aggregate the readings before providing a summary. Note that as the sensors may be placed in sensitive locations, privacy of the collected readings may need to be protected.
- **Pharmaceutical research:** Pharmaceutical companies may wish to collect statistics about genomic sequences of certain cohorts of hospital patients. Rather than sending the data directly, the hospitals may instead upload data to a server and want to ensure that the pharmaceutical company only gets aggregate information rather than data about any individual patient.
- **Disease outbreak detection:** To help detect disease outbreaks, disease control centers would like to collect aggregate statistics about symptoms and diagnoses from hospitals' patient records. Similar to the pharmaceutical research use-case, aggregating this data requires a server to produce the aggregate without leaking the individual patient data.

Abstracting away the details of the above examples, we are concerned with outsourcing the learning of aggregate statistics about *sensitive* data. Thus, to ensure privacy, we want the clients' inputs to remain private from other clients and from the server, as well as from the analyst (to the extent possible given that the analyst

learns the final result of the computation). In tandem, it is crucial to defend against *data poisoning*, where some clients' data is corrupted either due to unintentional error or intentional malice, which is a well-known concern. To address this, we want the server to be able to verify that the clients' inputs satisfy some analyst-specified validity rule before aggregating them. Of course, this validation check must still preserve the privacy of the inputs.

Crucially for the applications we consider, we want anybody to be able to provide input to the aggregation – i.e., to serve as a client. To enable this, we operate in a *public key* setting where we assume the clients can obtain the authentic public keys of the analyst and server, but do not share any secret information with either. To achieve all of these goals, we propose a new primitive called (public-key, outsourced) *Non-Interactive Validated Aggregation* (NIVA). An overview of the NIVA functionality is given in Figure 1.

Before outlining the performance and security requirements of NIVA, we wish to highlight one critical design goal.

On the importance of being non-interactive. We require that all NIVA schemes be non-interactive meaning that each client sends *a single message* to the server to provide input for the aggregation and after validating and aggregating the inputs, the server sends one message to the analyst. No additional rounds of communication are allowed either among the clients or between the clients and the server or analyst.

For the case of client-server communication, such non-interactive protocols are crucial in settings of low connectivity such as sensor networks. While client devices may be able to perform local computation, they can only communicate with the server when online and so need to do so in short bursts without waiting for a response from the server.

Moreover, by not requiring any interaction NIVA allows for clients to remain unknown to the server (and analyst) prior to submitting input. This is critical in settings such as public population surveys allowing anybody to participate.

We note that this restriction to unknown clients does place some limitation on the security that can be achieved against a malicious server. Since the analyst does not know the set of clients, a malicious server can pretend to be a client adding inputs of its choice into the aggregation or drop clients who have supplied valid inputs. Similar to prior work (e.g., [18, 32]), we thus only aim for privacy, but not integrity, against a malicious server.

For the case of server-analyst interaction, prohibiting communication between the server and analyst ensures that the analyst does not need to be online during data collection and only needs to receive the results, which can be stored for him.

1.2 NIVA Performance Requirements

To achieve the goals outlined above, NIVA has a unique set of performance requirements that we detail below.

- The analysts' cost must be independent of the number of clients. In fact, we require that the analyst not even know the identity of the clients, thus precluding any sort of secret-key setup between them.
- We require that each client's computation and communication be independent of the number of clients. Moreover, a client should only send a *single message* to the server and have no communication directly with the analyst or other clients. Thus, clients cannot pre-register before providing input.
- The server should run the most computationally intensive component of our protocol, namely validation and aggregation of the clients' inputs. Thus, the server's computation and communication grows linearly in the number of clients. But, we require that both of these steps be non-interactive (*i.e.*, performed locally by the server).
- The validity rule must not depend on the underlying cryptographic tools. This means that when proving the validity of their inputs, the clients must prove the statement "my input satisfies the validity equation" rather than "my ciphertext is an encryption of a valid input."

1.3 NIVA Security Requirements

To meet the needed security for a secure (outsourced) aggregation protocol, NIVA must satisfy the following security conditions.

- A malicious server must not learn the content of the clients' inputs except for whether or not they satisfy the validity constraint. Additionally, she should learn nothing about the final aggregate result unless it is made public by the analyst. This is true even if she colludes with a subset of the clients. As discussed previously, we only guarantee integrity (*i.e.*, that the analyst gets the correct output) against a semi-honest server.
- Malicious clients who do not collude with the server must not be able to get the server to accept any inputs that do not satisfy the validity rule. This must be true even if all clients are malicious. That is, we guarantee correctness against malicious clients. Moreover, malicious clients should not be able to block or tamper with the input of an honest client.
- A semi-honest analyst, who does not collude with the server and thus is assumed to not see the clients' (plaintext) communication with the server, should learn nothing about individual client inputs other than what is revealed by the final aggregated information. In particular, he should not learn which of the clients provided valid or invalid inputs. In fact, we achieve a somewhat stronger guarantee that as long as the analyst runs the Setup procedure honestly[1], then security is maintained even against an otherwise malicious analyst.

1.4 Technical Overview

We propose a NIVA protocol supporting summation, which we call PEAR (for **P**rivate, **E**fficient, **A**ccurate, **R**obust). Summation allows for a wide range of applications, in particular all those supported by PRIO [19] using their "affine aggregatable encodings." Moreover, PEAR achieves all of our desiderata described above. In particular, to our knowledge it is the first system for non-interactive aggregation with a single server and input validation where the validity rule does *not* depend on the underlying cryptographic primitives.

Intuition. The design of PEAR is based on a novel combination of additively-homomorphic inner-product functional encryption

[1] Note that this procedure is run only once before any clients submit their inputs.

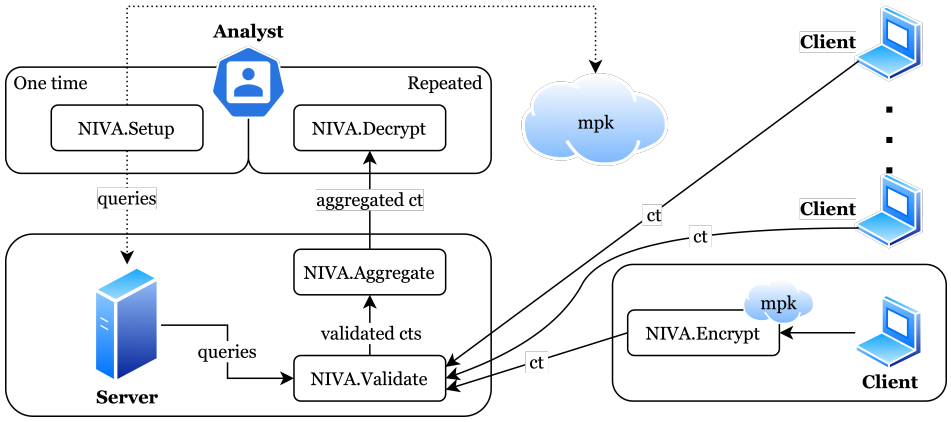

Figure 1: An overview of NIVA functionality.

(IPFE) [1] and fully-linear probabilistically-checkable proofs (FL-PCPs) [12]. In IPFE, anyone with mpk can encrypt a vector $\mathbf{x}$. Furthermore, a secret key $\mathsf{sk}_\mathbf{y}$ associated to a vector $\mathbf{y}$ can be generated using msk. While msk can decrypt ciphertexts as usual, an evaluation algorithm run on $\mathsf{ct}_\mathbf{x}$ and $\mathsf{sk}_\mathbf{y}$ yields $\langle \mathbf{x}, \mathbf{y} \rangle$ in the clear. Our key observation is that the verification procedure of an FL-PCP consists of a secret linear step on the instance $\mathbf{x}$ and proof $\boldsymbol{\pi}$ followed by a non-linear step on the result. As proven by [12], the output of the linear step is in fact zero-knowledge, allowing for the non-linear step to be performed in the clear. Thus, to preserve privacy, it is sufficient to perform the secret linear step under the hood of the IPFE. Moreover, existing IPFE schemes are additively homomorphic (although this property is not usually exploited in prior work), allowing to then homomorphically sum up the valid input vectors.

However, for this to work with the FL-PCP, we need IPFE supporting "unrestricted" inner-product, without a polynomial upper-bound on its magnitude. While initial schemes did not have this property as they recover inner-product in the exponent, we use a scheme based on class groups by Castagnos *et al.* [17], which exploits a subgroup where computing discrete log becomes easy.

OUR PROTOCOL. In more detail, PEAR works as follows for input-validity specified by some language $\mathcal{L} \in \mathsf{NP}$:

(1) The analyst generates keypair $(\mathsf{mpk}, \mathsf{msk})$ for IPFE and publishes mpk. Additionally, he chooses ℓ random vectors $\mathbf{q}_1, \ldots, \mathbf{q}_\ell$ (which will serve as FL-PCP query points) and generates IPFE secret keys $sk_1, \ldots, \mathsf{sk}_\ell$ for each of these vectors, then sends them privately to the server.
(2) On input $\mathbf{x}_i \in \mathcal{L}$ and a witness $\mathbf{w}_i$ for $\mathbf{x}_i$, client C_i generates an FL-PCP proof $\boldsymbol{\pi}_i$ to prove that $\mathbf{x}_i \in \mathcal{L}$ and encrypts $\mathbf{x}_i || \boldsymbol{\pi}_i$ under IPFE public-key mpk to get ciphertext ct_i, which it sends to the server.
(3) Upon receiving ciphertext ct_i, the server computes IPFE evaluation on ct_i with each of $\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell$ to perform the linear step of the FL-PCP verification procedure. Following this, the server can run the remainder of the procedure locally to determine whether $\mathbf{x} \in \mathcal{L}$. If so, the server stores ct_i and otherwise discards it.
(4) Upon receiving an *a priori* agreed-upon number (n) of ciphertexts, the server uses the additive homomorphism of the IPFE to aggregate all the valid ciphertexts producing a single aggregate ciphertext ct_{agg} encrypting the corresponding sum, which it then sends to the analyst.
(5) The analyst uses msk to decrypt ct_{agg} and recover the result.

We remark that the above can easily be extended to *weighted* sum aggregation assuming the IPFE supports homomorphic multiplication-by-constant, as is the case for the scheme of [17].

1.5 Comparison to Other Approaches for NIVA

Here we discuss other more straightforward or "off-the-shelf" approaches to building a secure NIVA protocol and why our new approach is superior.

NIVA BASED ON HE+NIZK. A secure NIVA protocol can also be constructed in a generic manner from additively homomorphic encryption (HE) and non-interactive zero-knowledge proofs (NIZKs) for languages in NP [22]. The protocol works as follows. Fix an analyst-specified NP language $\mathcal{L}$. First, the analyst generates a keypair $(\mathsf{pk}, \mathsf{sk})$ for HE and broadcasts pk. Given input $\mathbf{x}_i \in \mathcal{L}$ with witness $\mathbf{w}_i$, client C_i encrypts $\mathbf{x}_i$ under pk to get ct'_i and attaches to it a proof $\boldsymbol{\pi}_i$ that ct'_i encrypts a member of $\mathcal{L}$. Upon receiving $\mathsf{ct}_i = \mathsf{ct}'_i \| \boldsymbol{\pi}_i$, the server verifies $\boldsymbol{\pi}_i$ and discards ct'_i if this check fails. The server then homomorphically aggregates the remaining ct'_i and sends the result to the analyst, who decrypts it under sk.

However, this protocol is *non-blackbox* because the statement proven in ZK depends on the encryption circuit. While it is certainly possible to design optimized encryption schemes and proofs to reduce the overhead of these proofs, as done in some prior work — e.g., [7] — this limits the flexibility of the resulting NIVA protocol as it is not possible to replace the encryption scheme by a more efficient one developed down the line without reworking the proof. In particular, since optimizing performance is not necessarily the same as optimizing the "proof-cost" of a cryptographic primitive, improvements in building blocks may not yield improvements in the resulting construction. Optimizations to the protocol notwithstanding, to better demonstrate the overhead of the above non-blackbox solution we implement "an off-the-shelf" version of it based on Bulletproofs [15] and (the additively-homomorphic version of) El Gamal encryption [21]. We then compare its performance to PEAR, demonstrating significant gains. See Section 6 for details.

NIVA based on FHE+FL-PCP. As another point of comparison, we consider a secure NIVA protocol based around *fully*-homomorphic encryption (FHE) [23]. The idea is that we use an FL-PCP as before but the entire validation of the ciphertexts and aggregation of the valid ones is done completely "under the hood" of FHE. Note that while this approach is black-box in the underlying cryptography, it does *not* take advantage of the fact that the output of a "G-gate" in the validation circuit is ZK and the remainder of the computation can be done in the clear. Indeed, we demonstrate impracticality of this FHE-based NIVA protocol for an "off-the-shelf" implementation of it using the SEAL library [39]. Specifically, verifying the FL-PCP validation result under the hood of FHE, without client assistance, is highly expensive, and the server must repeat this task for each client ciphertext. Our approach is to exponentiate the FL-PCP validation bit to the size of the plaintext space, ensuring that zero remains zero while all other values become one, thereby filtering out invalid ciphertexts by flipping the bit. See Section 6 for further details.

1.6 Adding Differential Privacy

As our protocol is described above, the analyst learns nothing about clients' inputs except what is revealed by the output. Of course, this does not necessarily suffice to guarantee the privacy of clients' inputs since the output may reveal information about such inputs.

A standard way to guarantee that the output of the aggregation does not reveal too much about the individual inputs is to guarantee differential privacy (DP) [20]. We can add DP to PEAR in the typical case that input-validity checks for bounded norm, by simply having the server and the clients each add Laplacian noise to the final aggregation that is tuned to this maximum value. The server can add noise homomorphically, while the clients can add noise in the clear. Since the server and the clients each add their noise independently, this protocol achieves DP wrt. either of them. Since we see DP as being largely orthogonal to the design of our protocol, we do not elaborate on it further in the remainder of the paper.

1.7 Performance Evaluation

We implement PEAR using the unrestricted IPFE scheme by Castagnos et al. [17] and the FL-PCP by Boneh at al. [12]. We construct two validation circuits: one for verifying whether an input is a binary number and another for ensuring that the L_2 norm of an input vector is bounded. We benchmark PEAR and compare against a naive approach that combines off-the-shelf additively-homomorphic encryption with non-interactive zero-knowledge proofs of ciphertext validity as described earlier. In terms of client runtime, PEAR achieves a speedup of 4.5× to 20×, while for server runtime, it achieves a speedup of 1.2× to 3.6×. The server-side speedup could be significantly improved if the performance of the unrestricted IPFE scheme were comparable to that of restricted IPFE schemes, such as the optimized one proposed by Kim *et al.* [30]. We leave the development of a more efficient unrestricted IPFE scheme as an intriguing open problem. We also compare PEAR with the approach of using FHE with FL-PCP and observe that its server runtime for the binary number validation is 20× to over 100× slower.

2 RELATED WORK

Secure Aggregation. Secure aggregation, introduced by [11], allows a server to privately sum the inputs of a potentially large number of clients. These protocols use secret-shared masks between clients to enable efficient aggregation, *without* verifying validity of inputs. In SA, the server learns the sum; there is no separate analyst. Since then, various works have also considered how to extend these protocols to build *verifiable* secure aggregation as in our setting. In the single-server setting, various protocols have been proposed that require several rounds of communication between the server and clients [7, 18, 32]. Another line of work shows how to achieve secure aggregation in a multi-server setting relying on (interactive) secure computation between the servers [3, 6, 13, 19, 37, 40]. There is also another line of work that adds additional roles to some distinguished clients and necessitates additional communication between any client and these distinguished ones. An example is the committee-based setting of [8, 28, 31]. In comparison, PEAR does not require that any of the clients be known to anyone else. In Table 1, we compare PEAR against popular single-server, verifiable SA protocols. Here n is number of clients.

Scheme	PEAR	ACORN [7]	RoFL [32]	EiFFeL [18]
Number of rounds	1	≥ 5	≥ 5	≥ 5
Mal. clients threshold	$n-1$	$n/3$	t neighbors	$(n-1)/3$
Client comp	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
Server comp	$O(n)$	$O(n \log n)$	$O(n)$	$O(n^2)$
Proof system	FL-PCP	Bulletproofs	Bulletproofs	SNIP
Key setup	public	pairwise	pairwise	pairwise

Table 1: Comparing our NIVA protocol PEAR **to SA protocols with input validation.**

Of course, compared to existing single-server, verifiable SA protocols, PEAR operates in a setting where there is an additional analyst, who is separate from the server. Besides being natural from an application standpoint, we chose this modification because there are fundamental limitations to the security of such non-interactive protocols when the server learns the aggregation result herself. Even without public keys, the server can perform a so-called 'residual attack' if it corrupts clients. In a residual attack, the server locally re-runs the protocol many times, changing the corrupted clients' inputs each time, to learn more and more about other clients' inputs. In our public-key setting the limitations are even more severe, because the server does not need to corrupt any clients to run such attacks. By having separate server and analyst we simply avoid such issues. Note that [28] (which does not consider input validation) avoids them by having a committee in addition to a server.

Efficient Zero-Knowledge Proofs. Our protocol crucially relies on efficient non-interactive zero-knowledge (NIZK) proofs to prove validity of the clients' inputs. Zero-knowledge proofs were originally introduced by Goldwasser *et al.* [24] with the non-interactive variant introduced by Feige et al. [22]. Since then, a number of works (*e.g.* [5, 15, 25, 29, 33, 36]) have investigated how to make such proofs shorter and more efficient. In PEAR, we require proofs

that allow proving validity of an input unknown to the verifier. For this purpose, we build off of the work of Boneh *et al.* [12], who consider proofs over secret-shared inputs.

Inner Product Functional Encryption (IPFE). Another crucial tool for our construction is inner-product functional encryption (IPFE) which allows computing the inner product between an encrypted vector and a vector in the (functional) evaluation key. IPFE was first introduced by Abdalla *et al.* [1], who showed simple constructions based on DDH and LWE. Further variants of IPFE were given by *e.g.* [2, 4, 10]. In PEAR, we use the IPFE scheme due to Castagnos *et al.* [17] based on the hard subgroup membership (HSM) assumption in certain class groups. Importantly for us, this IPFE scheme is *unrestricted* in that it allows efficient evaluation when an inner product may output an *arbitrary* field element, rather than restricting the output to small set that can be brute-forced. This is needed for compatibility with the FL-PCP.

3 PRELIMINARIES

3.1 Notation and Conventions

If $\mathbf{v}$ is a vector then $|\mathbf{v}|$ is its length (the number of its coordinates) and $\mathbf{v}[i]$ is its i-th coordinate. Strings are identified with vectors over $\{0,1\}$, so that $|Z|$ denotes the length of a string Z and Z_i denotes its i-th bit. By ε we denote the empty string or vector. By $x\|y$ we denote the concatenation of strings x, y. For two vectors $\mathbf{x}, \mathbf{y}$ of the same length over the same field, we denote their inner-product by $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i \in [|\mathbf{x}|]} \mathbf{x}_i \cdot \mathbf{y}_i$.

If S is a finite set, then $|S|$ denotes its size. If X is a finite set, we let $x \leftarrow\!\!\$\ X$ denote picking an element of X uniformly at random and assigning it to x. Algorithms may be randomized unless otherwise indicated. Running time is the worst case, which for an algorithm with access to oracles means across all possible replies from the oracles. We use $\perp$ (bot) as a special symbol to denote rejection, and it is assumed to not be in $\{0,1\}^*$.

A function $\nu\colon \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p\colon \mathbb{N} \to \mathbb{R}$ there is a $\lambda_p \in \mathbb{N}$ such that $\nu(\lambda) \leq 1/p(\lambda)$ for all $\lambda \geq \lambda_p$. "PT" stands for "polynomial time," whether for randomized or deterministic algorithms. By 1^λ we denote the unary representation of the integer security parameter $\lambda \in \mathbb{N}$.

Games. We use the code-based game-playing framework of BR [9]. By $\Pr[\mathrm{G} \Rightarrow y]$ we denote the probability that the execution of game G results in this output being y. In games, integer variables, set variables, boolean variables, and string variables are assumed initialized, respectively, to 0, $\emptyset$, false, and $\perp$. For an adversary $\mathcal{A}$ playing game G, we may write another adversary $\mathcal{B}$ in the same format as G, with the understanding that $\mathcal{B}$ runs this game with $\mathcal{A}$.

3.2 Fully-Linear PCP Proof Systems

Binary relations. For $d, h \in \mathbb{N}$, let $\mathcal{R} \subseteq \mathbb{F}^d \times \mathbb{F}^h$ be a binary relation over a finite field $\mathbb{F}$. Additionally, let $\mathcal{R}_p \subseteq \mathbb{F}_p^d \times \mathbb{F}_p^h$ be a family of binary relations over finite fields $\mathbb{F}_p$ parameterized by the field size p. For simplification, we drop the subscript p once it is fixed. We define the corresponding language $\mathcal{L}_\mathcal{R}$ such that $\mathbf{x} \in \mathcal{L}_\mathcal{R}$ if and only if there exists an $\mathbf{w} \in \mathbb{F}^h$ such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$. We also define an associated arithmetic circuit $C_\mathcal{R}$ over the field $\mathbb{F}$ such that

Variable name	Meaning
λ	security parameter
$\mathbf{x}$	input instance vector
d	input instance vector length
$\mathbf{w}$	witness vector
h	witness vector length
$\boldsymbol{\pi}$	proof vector
m	proof length
$\mathbf{q}$	FL-PCP query
ℓ	number of FL-PCP queries

Table 2: Notation used throughout the paper.

$C_\mathcal{R}(\mathbf{x}, \mathbf{w}) = 0$ if and only if $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$. A message $\mathbf{x} \in \mathcal{L}_\mathcal{R}$ is a valid message. We stress that in our notation if $C_\mathcal{R}$ outputs 0 that means the input is *valid*, and any non-zero value means the input is *invalid*.

Syntax. The following definition is adapted from [12]. A fully linear probabilistically checkable proof system (FL-PCP) for $\mathcal{R}$ over $\mathbb{F}$ with proof length m and query complexity ℓ is a tuple of algorithms $\Pi_\mathcal{R} = (\mathsf{ProofGen}, \mathsf{Query}, \mathsf{Decision})$ where the prover runs the ProofGen algorithm and the verifier runs the Query and Decision algorithms described below. Additionally, all algorithms also have an implicit input aux which contains relevant information about the corresponding relation $\mathcal{R}$.

- $\mathsf{Setup}(1^\lambda, \mathbb{F})$: On input a unary encoding of the security parameter 1^λ and a field $\mathbb{F}$ st $|\mathbb{F}| \in \Omega(2^\lambda)$, it updates the internal state.
- $\mathsf{ProofGen}(1^\lambda, \mathbf{x}, \mathbf{w})$: On input a unary encoding of the security parameter 1^λ, an instance $\mathbf{x}$, and its witness $\mathbf{w}$, the proof generation algorithm outputs a proof $\boldsymbol{\pi} \in \mathbb{F}^m$.
- $\mathsf{Query}(1^\lambda, 1^d)$: On input a unary encoding of the security parameter 1^λ and the length of an input instance 1^d, the query algorithm outputs ℓ queries $\mathbf{q}_1, \ldots, \mathbf{q}_\ell \in \mathbb{F}^{d+m}$ and state information st.
- $\mathsf{Decision}(\mathsf{st}, a_1, \ldots, a_\ell)$: On input state information st and ℓ scalar values $a_1, \ldots, a_\ell \in \mathbb{F}$, the decision algorithm outputs a bit b.

Without loss of generality, we assume that the state of the query algorithm consists of the coins of the algorithm. Thus, the query algorithm can also be written as $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell) \leftarrow \mathsf{Query}_\mathcal{R}(1^\lambda, 1^d; \mathsf{st}_\Pi)$.

The above algorithms satisfy the following properties:

Completeness. For all $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, the verifier accepts a valid proof. Formally, an FL-PCP $\Pi_\mathcal{R}$ satisfies completeness if for all $\lambda, d \in \mathbb{N}$ and all $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$:

$$\Pr[\mathsf{Decision}(\mathsf{st}, \langle (\mathbf{x}\|\boldsymbol{\pi}), \mathbf{q}_1 \rangle, \ldots, \langle (\mathbf{x}\|\boldsymbol{\pi}), \mathbf{q}_\ell \rangle) \Rightarrow 1] = 1 \,,$$

where the probability is over

$$\boldsymbol{\pi} \leftarrow\!\!\$\ \mathsf{ProofGen}(1^\lambda, \mathbf{x}, \mathbf{w})\ ;\ (\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \mathsf{st}) \leftarrow\!\!\$\ \mathsf{Query}(1^\lambda, 1^d)\ .$$

Soundness. For all $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$, the verifier does not accept any proof. Formally, a FL-PCP $\Pi_\mathcal{R}$ satisfies γ-soundness for a function $\gamma(\cdot)$ if for all $\lambda, d \in \mathbb{N}$ and all $\mathbf{x}^* \notin \mathcal{L}_\mathcal{R}$ and a $\boldsymbol{\pi}^* \in \mathbb{F}^m$:

$$\Pr[\mathsf{Decision}(\mathsf{st}, \langle (\mathbf{x}^*\|\boldsymbol{\pi}^*), \mathbf{q}_1 \rangle, \ldots, \langle (\mathbf{x}^*\|\boldsymbol{\pi}^*), \mathbf{q}_\ell \rangle) \Rightarrow 1] \leq \gamma(\lambda)$$

where the probability is over $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \mathsf{st}) \leftarrow\!\!\$\ \mathsf{Query}(1^\lambda, 1^d)$.

Strong HVZK. Intuitively, the strong honest-verifier zero knowledge (HVZK) property states that the verifier does not learn anything about the instance $\mathbf{x}$ other than if $\mathbf{x} \in \mathcal{L}_\mathcal{R}$. A fully-linear PCP $\Pi_\mathcal{R}$ satisfies strong HVZK if there exists a simulator $S_{\Pi,\mathcal{R}}$ such that for all $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ and $\lambda, d \in \mathbb{N}$, the distribution of $S_{\Pi,\mathcal{R}}(1^\lambda, 1^d)$ is identical to the following:

$$(\mathsf{st}, \langle(\mathbf{x}||\boldsymbol{\pi}), \mathbf{q}_1\rangle, \ldots, \langle(\mathbf{x}||\boldsymbol{\pi}), \mathbf{q}_\ell\rangle),\ (\mathbf{q}_1, \ldots, \mathbf{q}_\ell)$$

where $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \mathsf{st}) \leftarrow\!\!\$\ \mathsf{Query}(1^\lambda, 1^d)$ and $\boldsymbol{\pi} \leftarrow\!\!\$\ \mathsf{ProofGen}(1^\lambda, \mathbf{x}, \mathbf{w})$.

3.3 Inner-Product Functional Encryption

Syntax. A public-key inner product functional encryption IPFE scheme is a tuple of algorithms IPFE = (Setup, KeyGen, Encrypt, Eval, Decrypt) described as follows:

- $\mathsf{Setup}(1^\lambda, 1^d)$: On input unary encodings of security parameter λ and dimension d, the setup algorithm outputs a master public key mpk (in particular containing d and description of $\mathbb{F}$), a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y})$: On input a master secret key msk and a function $\mathbf{y} \in \mathbb{F}^d$, the key generation algorithm outputs a functional decryption key $\mathsf{sk}_\mathbf{y}$.
- $\mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x})$: On input a master public key mpk and a message $\mathbf{x} \in \mathbb{F}^d$, the encryption algorithm outputs a ciphertext $\mathsf{ct}_\mathbf{x}$.
- $\mathsf{Eval}(\mathsf{sk}_\mathbf{y}, \mathsf{ct}_\mathbf{x})$: On input a functional decryption key $\mathsf{sk}_\mathbf{y}$, and a ciphertext $\mathsf{ct}_\mathbf{x}$, the evaluation algorithm outputs a string $z \in \mathbb{F}$ or a special symbol $\perp$.
- $\mathsf{Decrypt}(\mathsf{msk}, \mathsf{ct}_\mathbf{x})$: On input a master secret key msk and a ciphertext $\mathsf{ct}_\mathbf{x}$, the decryption algorithm outputs a message $\mathbf{x} \in \mathbb{F}^d$ or a special symbol $\perp$.

We use standard notions of correctness and security for IPFE which are provided in appendix A.

Additive Homomorphism. We use the notion of additive homomorphism given in [14]. We say that IPFE is *additively homomorphic* if it has an additional algorithm IPFE.Add as follows such that $\mathsf{Add}(\mathsf{mpk}, \mathsf{ct}_{\mathbf{x}_1}, \mathsf{ct}_{\mathbf{x}_2})$: On inputs a master public key mpk and two ciphertexts $\mathsf{ct}_{\mathbf{x}_1}, \mathsf{ct}_{\mathbf{x}_2}$, the Add algorithm outputs a ciphertext ct. Moreover, the following ciphertexts are identically distributed for all $\lambda, d \in \mathbb{N}$, and all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{F}^d$: $\mathsf{Add}(\mathsf{mpk}, \mathsf{Enc}(\mathsf{pk}, \mathbf{x}_1), \mathsf{Enc}(\mathsf{pk}, \mathbf{x}_2))$ and $\mathsf{Enc}(\mathsf{pk}, \mathbf{x}_1 + \mathbf{x}_2)$, where $(\mathsf{pk}, \mathsf{msk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^d)$. Note that this means that the ciphertexts output by Add are compact in size. For notational convenience, by $\mathsf{Add}(\mathsf{mpk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ we mean that the Add algorithm is run repeatedly to generate a single ciphertext. It is easy to check that existing IPFE schemes are additively homomorphic.

Unrestricted. We say that IPFE is *unrestricted* if its evaluation algorithm runs in polynomial time (PT). The IPFE schemes proposed in [1, 4] are not PT because they recover the inner product in the exponent. As a result, they restrict the message space so that the number of possible inner product values remains sufficiently small. The lattice-based IPFE scheme proposed in [4] (though over a small field) and the hard subgroup membership (HSM)-based scheme in [17] (which we use in our implementation) are unrestricted.

Henceforth, we assume that IPFE is always unrestricted.

Preimage sampleability. The inner product function is *preimage sampleable* as defined in [34]. This means that given a set of n vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and their inner product values $a_1, \ldots, a_n$ with some unknown vector $\mathbf{y}$, we can compute the vector $\mathbf{y}$ by solving the system of linear equations given that at least one such vector $\mathbf{y}$ exists. Free variables in the system can be assigned arbitrarily. This is used in our security games as an algorithm VecSamp that inputs $((\mathbf{x}_1, a_1), \ldots, (\mathbf{x}_n, a_n))$ and outputs a consistent vector $\mathbf{v}$. Note that this is a property of the inner product function and, therefore, applies to any inner-product functional encryption scheme.

4 NON-INTERACTIVE VALIDATED AGGREGATION

4.1 Syntax and Correctness

Syntax. A *non-interactive validated aggregation (NIVA)* protocol for a relation $\mathcal{R} \subseteq \mathbb{F}^d \times \mathbb{F}^h$ for a field $\mathbb{F}$ and $d, h, n \in \mathbb{N}$, is a protocol consisting of three kinds of participants: an analyst, a server and n clients and a tuple of algorithms $\mathsf{NIVA}_\mathcal{R}$ = (Setup, Encrypt, Validate, Aggregate, Decrypt) defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^d)$: On input unary encodings of the security parameter λ and an input length d, the setup algorithm outputs a public key mpk, a master secret key msk, and a validation key vk.
- $\mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x}, \mathbf{w})$: On input a public key mpk, an input $\mathbf{x}$, and a witness $\mathbf{w}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Validate}(\mathsf{vk}, \mathsf{ct})$: On input a validation key vk and a ciphertext ct, the validation algorithm outputs a bit b.
- $\mathsf{Aggregate}(\mathsf{mpk}, \{\mathsf{ct}_j\}_{j\in[n]})$: On input a master public key mpk and a set of ciphertexts $\{\mathsf{ct}_j\}_{j\in[n]}$ for some $n \in \mathbb{N}$, the aggregation algorithm outputs an aggregated ciphertext ct_{agg}.
- $\mathsf{Decrypt}(\mathsf{msk}, \mathsf{ct})$: On input a master secret key msk and a ciphertext ct, the decryption algorithm outputs $\mathbf{y} \in \mathbb{F}^d$.

The protocol, which is also depicted in Fig. 1, is described next.

- The analyst runs the Setup algorithm and sends (mpk, vk) to the server and keeps msk to itself. In addition, the server sends mpk to each client when they join the system.
- Each client runs the Encrypt algorithm using the mpk and its input-witness pair $(\mathbf{x}, \mathbf{w})$ and sends the resulting ct to the server.
- The server upon receiving some number of ciphertexts from clients, runs the Validate algorithm on each ciphertext and discards invalid ones.
- The server uses the Aggregate algorithm to aggregates the n valid ciphertexts where n is less than or equal to the number of clients. Then, the server sends the aggregate ciphertext $\mathsf{ct}_{\mathsf{agg}}$ to the analyst when requested.
- The analyst decrypts the aggregate ciphertext using the Decrypt algorithm and its msk.

CORRECTNESS. We require *validation correctness* meaning that for all $\lambda, d, h \in N$ and all $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$

$$\Pr[\mathsf{Validate}(\mathsf{vk}, \mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x}, \mathbf{w})) \Rightarrow 1] = 1$$

where $(\mathsf{mpk}, \mathsf{msk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^d)$ and the probability is over the coins of the Setup and Encrypt algorithms. We say that $\mathsf{NIVA}_\mathcal{R}$ has *aggregation error* γ if for all $\lambda, n \in \mathbb{N}$, all $(\mathbf{x}_1, \mathbf{w}_1), \ldots, (\mathbf{x}_n, \mathbf{w}_n) \in \mathcal{R}$:

$$\Pr\left[\mathsf{Decrypt}\left(\mathsf{msk}, \mathsf{Aggregate}(\mathsf{mpk}, \{\mathsf{ct}_j\}_{j\in[n]})\right) = \sum_{j\in[n]} \mathbf{x}_j\right] \geq 1-\gamma(\lambda)$$

where $(\mathsf{mpk}, \mathsf{msk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^d)$, $\mathsf{ct}_j \leftarrow\!\!\$\ \mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x}_j, \mathbf{w}_j)$ for all $j \in [n]$, and the probability is over the coins of the Setup and Encrypt algorithms.

4.2 Security

PRIVACY AGAINST A MALICIOUS SERVER. We define an indistinguishability-based notion of privacy for $\mathsf{NIVA}_\mathcal{R}$ via the game in Figure 2. Here, an adversary receives a master public key and a validation key and has access to an encryption oracle. The adversary is modeled after a malicious server. She must make encryption queries with *valid* message and witness pairs. Then, she receives ciphertexts for the left or the right tuple and wins the game if she can determine whether the left or the right tuples are being encrypted. Note that we do not consider privacy of invalid inputs as that is equivalent to a malicious client. Formally, for an adversary $\mathcal{A}$, for every $\lambda, d \in \mathbb{N}$ we let its ind-advantage be:

$$\mathbf{Adv}^{\text{ind}}_{\mathsf{NIVA},\mathcal{A}}(\lambda, d) = 2 \cdot \Pr[\mathbf{G}^{\text{ind}}_{\mathsf{NIVA},\mathcal{A}}(\lambda, d) \Rightarrow 1] - 1.$$

We say that $\mathcal{A}$ is *valid* if every encryption query $((\mathbf{x}^0, \mathbf{w}^0), (\mathbf{x}^1, \mathbf{w}^1))$ comprises of valid message-witness pair i.e., $(\mathbf{x}^0, \mathbf{w}^0), (\mathbf{x}^1, \mathbf{w}^1) \in \mathcal{R}$. Intuitively, a malicious server should not be able to get any information about honest client data except for their validity status even when deviating from the protocol.

Game $\mathbf{G}^{\text{ind}}_{\mathsf{NIVA},\mathcal{A}}(\lambda, d)$

INITIALIZE$(1^\lambda, 1^d)$:
1 $b \leftarrow\!\!\$\ \{0,1\}$
2 $(\mathsf{mpk}, \mathsf{msk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^d)$
3 Return $(\mathsf{mpk}, \mathsf{vk})$

ENCO $((\mathbf{x}^0, \mathbf{w}^0), (\mathbf{x}^1, \mathbf{w}^1))$:
4 $\mathsf{ct} \leftarrow\!\!\$\ \mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x}^b, \mathbf{w}^b)$
5 Return ct

FINALIZE(b'):
6 Return $(b = b')$

Figure 2: Game defining NIVA privacy against the server.

PRIVACY AGAINST THE ANALYST. We also define a notion of privacy against the analyst for $\mathsf{NIVA}_\mathcal{R}$ via the game in Figure 3. An adversary submits two sets of inputs such that the valid inputs in each set have the same sum and receives an aggregated ciphertext and a master secret key. The adversary is modeled after a semi-honest analyst. The adversary wins the game if he can determine whether the left or the right set of inputs are validated and aggregated. Formally, for an adversary $\mathcal{A}$, for every $\lambda, d \in \mathbb{N}$, we let its ind-advantage be:

$$\mathbf{Adv}^{\text{analyst-ind}}_{\mathsf{NIVA},\mathcal{A}}(\lambda, d) = 2 \cdot \Pr[\mathbf{G}^{\text{analyst-ind}}_{\mathsf{NIVA},\mathcal{A}}(\lambda, d) \Rightarrow 1] - 1.$$

We say that $\mathcal{A}$ is *valid* if her submission $((\mathbf{x}^0_i, \mathbf{w}^0_i)_{i\in[n]}, (\mathbf{x}^1_i, \mathbf{w}^1_i)_{i\in[n]})$ is such that the following holds:

$$\sum_{i\in n:\ \mathbf{x}^0_i \in \mathcal{L}_\mathcal{R}} \mathbf{x}^0_i = \sum_{i\in n:\ \mathbf{x}^1_i \in \mathcal{L}_\mathcal{R}} \mathbf{x}^1_i.$$

We say that $\mathsf{NIVA}_\mathcal{R}$ is *analyst-private* if for all $d \in \mathbb{N}$, $\mathbf{Adv}^{\text{ind}}_{\mathsf{NIVA},\mathcal{A}}(\cdot, d)$ is negligible for all valid PT $\mathcal{A}$.

We want to capture the idea that an analyst does not get any information about the individual client data including the number of valid inputs or validity status of individual data points and only receives the aggregated result of valid inputs.

Game $\mathbf{G}^{\text{analyst-ind}}_{\mathsf{NIVA},\mathcal{A}}(\lambda, d)$

INITIALIZE$((\mathbf{x}^0_i, \mathbf{w}^0_i)_{i\in[n]}, (\mathbf{x}^1_i, \mathbf{w}^1_i)_{i\in[n]})$:
1 $b \leftarrow\!\!\$\ \{0,1\};\ S \leftarrow \emptyset$
2 $(\mathsf{mpk}, \mathsf{msk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^d)$
3 For $i \in [n]$:
4 $\quad \mathsf{ct}_i \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x}^b_i, \mathbf{w}^b_i)$
5 $\quad$ If $\mathsf{Validate}(\mathsf{vk}, \mathsf{ct}_i) : S \leftarrow S \cup \{\mathsf{ct}_i\}$
6 $\mathsf{ct}_b \leftarrow \mathsf{Aggregate}(\mathsf{mpk}, S)$
7 Return $(\mathsf{msk}, \mathsf{ct}_b)$

FINALIZE(b'):
8 Return $(b = b')$

Figure 3: Game defining NIVA privacy against the analyst.

INTEGRITY. We define a notion of integrity for $\mathsf{NIVA}_\mathcal{R}$ via the game in Figure 4. An adversary on receiving a master public key and has access to a validate oracle. The adversary is modeled after a malicious client. She submits ciphertexts to the validate oracle and receives a bit indicating if the ciphertext verified or not. The adversary submits a final ciphertext and wins the game if this ciphertext encrypts an invalid instance but passes the validity check. Concretely, for an adversary $\mathcal{A}$, for every $\lambda, d \in \mathbb{N}$ we let its int-advantage be:

$$\mathbf{Adv}^{\text{int}}_{\mathsf{NIVA},\mathcal{A}}(\lambda, d) = \Pr[\mathbf{G}^{\text{int}}_{\mathsf{NIVA},\mathcal{A}}(\lambda, d) \Rightarrow 1].$$

A PT adversary is *valid* if she submits a ct such that size of ct is $|\mathsf{Encrypt}(\mathsf{mpk}, \mathbf{v})|$ where $|\mathbf{v}| = d + m$.

We say that a $\mathsf{NIVA}_\mathcal{R}$ scheme has integrity if the advantage of any valid PT adversary is negligible in the security parameter λ.

Intuitively, we want to capture the fact that it is difficult for malicious clients to get their invalid ciphertexts accepted and aggregated with valid ciphertexts even if they know the validity status of previous ciphertexts.

4.3 Reusability in FL-PCPs

REUSABLE SOUNDNESS. First, we define an extension of soundness for FL-PCPs that we call *reusable* soundness, which involves reusing the verifier's queries across multiple proofs. Intuitively, this property states that reusing the state and queries of the FL-PCP across multiple proofs does not significantly degrade the soundness of the protocol, which will be important in our application.

Game $\mathbf{G}^{\text{int}}_{\text{NIVA},\mathcal{A}}(\lambda, d)$

INITIALIZE:
1 $(\text{mpk}, \text{msk}, \text{vk}) \leftarrow\$ \text{Setup}(1^\lambda, 1^d)$
2 Return mpk

VALIDATEO(ct):
3 Return Validate(vk, ct)

FINALIZE(ct*):
4 If Validate(vk, ct*) :
5 $\quad \mathbf{y}^* \leftarrow \text{Decrypt}(\text{msk}, \text{ct}^*)$
6 $\quad$ If $C_{\mathcal{R}}(\mathbf{y}^*) \neq 0$: Return 1
7 Return 0

Figure 4: Game defining NIVA integrity.

For $\lambda, d \in \mathbb{N}$ and an adversary $\mathcal{A}$ interacting with the game given in Figure 5, we let its reusable soundness advantage be as follows:

$$\mathbf{Adv}^{\text{reuse-sound}}_{\Pi,\mathcal{A}}(\lambda, d) = \Pr[\mathbf{G}^{\text{reuse-sound}}_{\text{NIVA},\mathcal{A}}(\lambda, d) \Rightarrow 1] .$$

We say that $\Pi_{\mathcal{R}}$ satisfies γ-reusable soundness if for all adversaries making Q REPEATO queries, and all $\lambda, d \in \mathbb{N}$ its reusable-soundness advantage is bounded as follows:

$$\mathbf{Adv}^{\text{reuse-sound}}_{\Pi,\mathcal{A}}(\lambda, d) \leq \gamma(\lambda, Q) .$$

The FL-PCP of Boneh *et al.* [12], BBG$^+$, and our optimized FL-PCP in Section 5 fulfill this property, which we prove in the appendix B.1.

Game $\mathbf{G}^{\text{reuse-sound}}_{\Pi,\mathcal{A}}(\lambda, d)$

INITIALIZE:
1 $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \text{st}) \leftarrow\$ \text{Query}(1^\lambda, 1^d)$

REPEATO($\mathbf{x} \| \boldsymbol{\pi}$):
2 For $i \in [\ell] : a_i \leftarrow \langle (\mathbf{x} \| \boldsymbol{\pi}), \mathbf{q}_i \rangle$
3 $b \leftarrow \text{Decision}(\text{st}, a_1, \ldots, a_\ell)$
4 Return b

FINALIZE($\mathbf{x}^* \| \boldsymbol{\pi}^*$):
5 If $\mathbf{x}^* \in \mathcal{R}$: Return $\perp$
6 For $i \in [\ell] : a^*_i \leftarrow \langle (\mathbf{x}^* \| \boldsymbol{\pi}^*), \mathbf{q}_i \rangle$
7 $b^* \leftarrow \text{Decision}(\text{st}, a^*_1, \ldots, a^*_\ell)$
8 Return b^*

Figure 5: Game defining reusable soundness property for FL-PCP.

REUSABLE ZERO-KNOWLEDGE. We also define a corresponding extension of strong HVZK for FL-PCP that we call *reusable* strong HVZK using the games in Figure 6. Intuitively, the property states that reusing the state and queries of the FL-PCP across multiple instances of proofs does not reveal any information about valid input instances.

For $\lambda, d \in \mathbb{N}$ and an adversary $\mathcal{A}$ interacting with the game given in Figure 6, we let its reusable strong HVZK advantage be as follows:

$$\mathbf{Adv}^{\text{rshvzk}}_{\Pi,\mathcal{A}}(\lambda, d) = \Pr[\mathbf{G}^{\text{rshvzk-1}}_{\Pi,\mathcal{A}}(\lambda, d) \Rightarrow 1] - \Pr[\mathbf{G}^{\text{rshvzk-0}}_{\Pi,\mathcal{A},\mathcal{S}}(\lambda, d) \Rightarrow 1] .$$

We say that $\Pi_{\mathcal{R}}$ satisfies reusable strong HVZK if for all adversaries and all $\lambda, d \in \mathbb{N}$ its reusable strong HVZK advantage is negligible in λ.

The FL-PCPs BBG$^+$ and our optimized FL-PCP in Section 5 fulfill this property, which we prove in the appendix B.2.

Game $\mathbf{G}^{\text{rshvzk-1}}_{\Pi,\mathcal{A}}(\lambda, d)$

INITIALIZE:
1 $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \text{st}) \leftarrow\$ \text{Query}(1^\lambda, 1^d)$
2 Return st

REPEATO($\mathbf{x}, \mathbf{w}$):
3 $\boldsymbol{\pi} \leftarrow\$ \text{ProofGen}(1^\lambda, \mathbf{x}, \mathbf{w})$
4 For $i \in [\ell] : a_i \leftarrow \langle (\mathbf{x} || \boldsymbol{\pi}), \mathbf{q}_i \rangle$
5 Return $(a_1, \ldots, a_\ell)$

FINALIZE(b):
6 Return b

Game $\mathbf{G}^{\text{rshvzk-0}}_{\Pi,\mathcal{A},\mathcal{S}}(\lambda, d)$

INITIALIZE:
1 $\text{st} \leftarrow\$ S_1(1^\lambda, 1^d)$
2 Return st

REPEATO ($\mathbf{x}, \mathbf{w}$):
3 $(a_1, \ldots, a_\ell) \leftarrow\$ S_2(1^\lambda, 1^d, \text{st})$
4 Return $(a_1, \ldots, a_\ell)$

FINALIZE(b):
5 Return b

Figure 6: Games defining reusable strong honest-verifier zero knowledge property for FL-PCP.

4.4 PEAR: Our Black-Box NIVA Protocol

As discussed in the Introduction, a NIVA protocol that is non-blackbox in the underlying cryptography can be constructed from homomorphic encryption and NIZK. We now present our black-box solution.

CONSTRUCTION. Let Δ = (Setup, KeyGen, Encrypt, Eval, Decrypt) be an additively-homomorphic IPFE scheme. Let $\mathcal{R}_p$ be a family of relations over finite fields $\mathbb{F}_p$ parameterized by the order p, a prime. Let $\Pi_{\mathcal{R}}$ =(ProofGen, Query, Decision) be an FL-PCP system for a [2] relation $\mathcal{R}_p \subseteq \mathbb{F}_p^d \times \mathbb{F}_p^h$. We define the associated NIVA protocol $\text{NIVA}_{\mathcal{R}}[\Delta, \Pi_{\mathcal{R}}]$ = (Setup, Encrypt, Validate, Aggregate, Decrypt). Henceforth, we denote it by PEAR and describe it below with formal details given in Figure 7.

- During setup, the analyst runs the IPFE setup algorithm and generates query vectors for the FL-PCP. Next, the analyst generates function keys for the queries and sends the IPFE public key along with the verification key—consisting of the function keys and a state—to the server. The master secret key for IPFE is retained by the analyst.
- During encryption, a client generates a proof for its message and witness with respect to the validation circuit of the relation. The client then encrypts the message, concatenated with its proof, using the IPFE public key that is broadcasted by the server, and sends the resulting ciphertext to the server.
- During validation, the server receives a ciphertext and evaluates the inner product of the ciphertext with each query vector. It then runs the decision algorithm of the FL-PCP using the state of the query algorithm and the computed inner product values.
- During aggregation, the server combines n valid ciphertexts using IPFE's homomorphic addition algorithm. Furthermore, it homomorphically adds a mask to the second component of the ciphertexts to obscure the sum of the proofs. The server then sends the aggregate ciphertext to the analyst.
- Upon receiving an aggregate ciphertext, the analyst decrypts it using the IPFE master secret key.

[2]We only support relations over finite fields of prime order because the IPFE scheme we use is over fields of prime order.
[2]mpk has the description of $\mathbb{F}$
[3]following the syntax of Δ as in section 3.3

Setup($1^\lambda, 1^d$):

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \Delta.\mathsf{Setup}(1^\lambda, 1^d)$
$\Pi_{\mathcal{R}}.\mathsf{Setup}(1^\lambda, \mathsf{mpk})$ [2]
$(\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \mathsf{st}_\Pi) \leftarrow \Pi_{\mathcal{R}}.\mathsf{Query}(1^\lambda, 1^d)$
For $j \in [\ell]$:
  $\mathsf{sk}_{\mathbf{q}_j} \leftarrow \Delta.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{q}_j)$
$\mathsf{vk} \leftarrow \left(\{\mathsf{sk}_{\mathbf{q}_j}\}_{j\in[\ell]}, \mathsf{st}_\Pi\right)$
Return $(\mathsf{mpk}, \mathsf{msk}, \mathsf{vk})$

Encrypt(mpk, $\mathbf{x}$, $\mathbf{w}$):

$\boldsymbol{\pi} \leftarrow\$ \Pi_{\mathcal{R}}.\mathsf{ProofGen}(1^\lambda, \mathbf{x}, \mathbf{w})$
$\mathsf{ct} \leftarrow\$ \Delta.\mathsf{Encrypt}(\mathsf{mpk}, (\mathbf{x} \| \boldsymbol{\pi}))$ [3]
Return ct

Validate(ct, vk):

$\left(\{\mathsf{sk}_{\mathbf{q}_j}\}_{j\in[\ell]}, \mathsf{st}_\Pi\right) \leftarrow \mathsf{vk}$
For $j \in [\ell]$:
  $a_j \leftarrow \Delta.\mathsf{Eval}(\mathsf{sk}_{\mathbf{q}_j}, \mathsf{ct})$
$b \leftarrow \Pi_{\mathcal{R}}.\mathsf{Decision}(\mathsf{st}_\Pi, a_1, \ldots, a_\ell)$
Return b

Aggregate(mpk, $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$):

$\mathbf{r} \leftarrow\$ \mathbb{F}^m$
$\mathsf{ct}_{n+1} \leftarrow \Delta.\mathsf{Encrypt}(\mathsf{mpk}, (0^d \| \mathbf{r}))$
If $n = 0$: Return ct_{n+1}
$\mathsf{ct}_{\mathsf{agg}} \leftarrow \Delta.\mathsf{Add}(\mathsf{mpk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_{n+1})$
Return $\mathsf{ct}_{\mathsf{agg}}$

Decrypt(msk, ct):

$\mathbf{y} \leftarrow \Delta.\mathsf{Decrypt}(\mathsf{msk}, \mathsf{ct})$
Return $\mathbf{y}[1 : d]$

Figure 7: PEAR**: Our NIVA construction.**

Correctness. PEAR is validation correct which follows from perfect evaluation correctness of the IPFE Δ and completeness of the FL-PCP $\Pi_{\mathcal{R}}$.

PEAR also has aggregation error $\gamma(\lambda) = 0$ which follows from additive homomorphism and perfect evaluation correctness of the IPFE Δ.

Privacy against a malicious server. We show that PEAR is private against a malicious server under the indistinguishability-based security notion defined in Section 4.2.

Theorem 1. *Let* PEAR *be as given in Fig. 7. Then* PEAR *is an ind-secure* NIVA *protocol if the IPFE scheme Δ is ind-secure and the FL-PCP protocol $\Pi_{\mathcal{R}}$ satisfies reusable strong HVZK. Namely, for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that:*

$$\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{PEAR},\mathcal{A}}(\lambda, d) \leq \mathbf{Adv}^{\mathsf{ind}}_{\mathsf{IPFE},\mathcal{B}}(\lambda, d) .$$

The running-time of $\mathcal{B}$ is about the same as $\mathcal{A}$.

We prove the above theorem in a game-playing format. The first game is the privacy against the server game of NIVA. In the second game, we swap the vector that is encrypted with IPFE with another vector that has the same inner product values using a vector sampling algorithm. This is possible because inner-product is a pre-image sampleable function. An adversary that can distinguish between these two games can break indistinguishability-based security of the IPFE scheme. After that, we swap the query vector generated during the setup procedure with queries generated by a reusable strong honest-verifier zero-knowledge simulator for the FL-PCP. The inner product values generated during each encryption are also replaced with outputs from the simulator. Here, these two games are indistinguishable because of the reusable strong honest-verifier zero-knowledge property of the FLPCP. A formal proof of the theorem including the hybrid games and the adversary is provided in Appendix C.

Privacy against the analyst. Privacy of PEAR against the analyst follows directly from the property of the underlying IPFE Δ that homomorphically aggregated ciphertexts are distributed identically to freshly generated ones.

Integrity. The integrity of PEAR follows directly from reusable soundness of the underlying FL-PCP as defined in 3.2. We summarize it in the following theorem.

Theorem 2. *Let* PEAR *be as constructed in Fig. 7. Then,* PEAR *is an int-secure* NIVA *protocol. Formally, for any adversary $\mathcal{A}$ for all $\lambda, d \in N$:*

$$\mathbf{Adv}^{\mathsf{int}}_{\mathsf{PEAR},\mathcal{A}}(\lambda, d) \leq \gamma(\lambda)$$

where $\gamma(\cdot)$ is the reusable soundness error of $\Pi_{\mathcal{R}}$.

5 OPTIMIZED FL-PCP FOR CONJUCTION-OF-PREDICATES

5.1 Conjunction-of-Predicates Relations

A common structure of an input validity check in practice involves verifying that the input satisfies multiple predicates at once. Namely, let $\mathcal{R} \subseteq \mathbb{F}^d \times \mathbb{F}^h$ for $d, h \in \mathbb{N}$. Let $C_1, \ldots, C_k$ be arithmetic circuits and define

$$\mathcal{R}_{\mathsf{AND}}[C_1, \ldots, C_k](\mathbf{x}, \mathbf{w}) = (C_1(\mathbf{x}, \mathbf{w}) = 0) \ \wedge \ \ldots \ \wedge \ (C_k(\mathbf{x}, \mathbf{w}) = 0)$$

which we call a *conjunction-of-predicates* relation [3]. For simplicity, we assume that each C_i has input length $d + h$. We give two main examples of such relations before discussing our optimization of the FL-PCP of [12] for them.

Binary validation. We first consider the binary validation relation which checks that an input is a binary vector. Formally, $\mathcal{R}_{\mathsf{BV}}(\mathbf{x}) = 1$ if for all $i \in [d]$, $x_i \in \{0, 1\}$ where $h = 0$. This relation does not use the witness. Then there exist arithmetic circuits $C_1, \ldots, C_d$ such that $\mathcal{R}_{\mathsf{BV}} = \mathcal{R}_{\mathsf{AND}}[C_1, \ldots, C_d]$, namely $C_i(\mathbf{x}) = x_i \cdot (x_i - 1)$ for all $i \in [d]$.

L_2-Norm bound validation. We next consider the L_2-norm bound validation relation parameterized by $b \in \mathbb{N}$ such that $2^{2b} < |\mathbb{F}|/d$, which checks that the square of an input vector's L_2-norm *over the integers* is bounded by 2^{2b}. Note that it would not suffice to simply check the L_2-norm *over the field* because of potential wrap-around. To ensure that there is no wrap-around we need to check that each component of the input $\mathbf{x}$ is bounded by 2^b. (Thus, the trivial bound on the square of the L_2-norm is $d \cdot 2^{2b}$, but we check that it is actually much less; furthermore, our approach generalizes to any bound that is a power of two.) A simple way to check this, and the aforementioned bound on the square of the L_2-norm, is to look at their binary decomposition. Specifically, if the decomposition of x_i has at most b bits, then $x_i < 2^b$. Thus, for our L_2-norm bound validation we provide these binary decompositions as witnesses.

[3] What follows later as $C_{\mathcal{R}}$ is the same as the C_i for some $i \in k$ here.

Namely, for input $\mathbf{x} \in \mathbb{F}^d$, let witness $\mathbf{w} = (\mathbf{u}, \mathbf{v}) \in \mathbb{F}^{db} \times \mathbb{F}^{2b}$ the L_2-norm bound validation relation $\mathcal{R}_{\mathsf{NBV}}[b](\mathbf{x}, \mathbf{w}) = 1$ if:

(1) $\mathbf{u}$ and $\mathbf{v}$ are binary vectors.
(2) $\mathbf{u} = (\mathbf{u}_1 \| \ldots \| \mathbf{u}_d)$ is a vector of binary decompositions of the components of $\mathbf{x}$ where each $\mathbf{u}_i$ is of length b.
(3) $\mathbf{v}$ is the binary representation of $\|\mathbf{x}\|_2^2$

Observe that our witness w has length $h = db + 2b$. Again, we claim that $\mathcal{R}_{\mathsf{NBV}}[b] = \mathcal{R}_{\mathsf{AND}}[C_1, \ldots, C_k]$ for $k = h + d + 1$ for arithmetic circuits $C_1, \ldots, C_k$. The circuits are as follows:

(1) For $j \in [h]$: $C_j(\mathbf{x}, \mathbf{w}) = w_j \cdot (w_j - 1)$.
(2) For $i \in [d]$: $C_{h+i} = x_i - \left(\sum_{j=1}^{b} \mathbf{u}_{i \cdot b + j} \cdot 2^{j-1}\right)$.
(3) $C_{h+d+1} = \left(\sum_{i=1}^{d} x_i^2\right) - \left(\sum_{j=1}^{2b} \mathbf{v}_j \cdot 2^{j-1}\right)$.

5.2 Overview of Our Optimization

Existing FL-PCP. To explain our optimization, we first describe some details of BBG$^+$, [12]'s FL-PCP and why it does not directly support conjunction-of-predicates relations optimally. Recall that their construction works on arithmetic circuits containing many instances of the same sub-circuit, where each sub-circuit takes L inputs and has one output. These sub-circuits are called "G-gates". The construction guarantees that there exists an FL-PCP with strong HVZK for the arithmetic circuit $C_{\mathcal{R}}$.

The idea of the FL-PCP is that the prover computes $C(\mathbf{x}, \mathbf{w})$ and enumerates all the inputs and outputs of the G-gates as interpolated polynomials. The verifier checks that the polynomials are correctly generated by evaluating them at a uniform random point, which can be done via inner-product. The construction is provided in full Iin Fig. 14 for completeness and further details can be found in [12].

Supporting conjuction-of-predicates. As BBG$^+$ works with arithmetic circuits, it does not directly support conjunctions-of-predicates relations in an efficient manner. Although, a boolean AND gate can be simulated by an arithmetic circuit in principle because both computational models are complete, this incurs a significant blowup in the circuit size. A slightly better idea is to modify BBG$^+$ by letting the output of each C_i be the output of a G-gate, and having the prover reveal the output of the *last* k G-gates, corresponding to the outputs of $C_i(\mathbf{x}, \mathbf{w})$, instead of just the last G-gate. Note that since the output of $C_i(\mathbf{x}, \mathbf{w})$ on valid inputs is equal to 0 for all $i \in [k]$, revealing the output of these k gates instead of just one is still zero knowledge. On the other hand, if the input is invalid, zero knowledge is not required to hold. We omit details since this is not our final construction. Indeed, the problem is that revealing the output of the last k G-gates requires k additional linear queries by the verifier, which in our NIVA construction corresponds to k decryptions under the IPFE, a very expensive operation.

Our Optimization. To describe our optimization, we first observe that the verifier computes an affine transformations on the proof, thus it is possible to take linear combinations of these transformations to compute a linear combination of the output of the transformations. Thus, we could compute $\sum_{i=1}^{k} C_i(\mathbf{x}, \mathbf{w})$ using a single linear query by simply adding up the queries that compute each $C_i(\mathbf{x}, \mathbf{w})$. However, as previously stated it does not suffice to check that $\sum_{i=1}^{k} C_i(\mathbf{x}, \mathbf{w})$ is equal to 0.

Our main observation is that checking that a *random* linear combination of $C_i(\mathbf{x}, \mathbf{w})$ equals to 0, where the randomness is *chosen by the verifier* allows to use just *one* query while ensuring soundness with overwhelming probability. In more detail, the verifier computes $\sum_{i=1}^{k} \psi_i \cdot C_i(\mathbf{x}, \mathbf{w})$ and checks that it is equal to 0, where $\psi_i \leftarrow\!\!\$\ \mathbb{F}$. It is unlikely for this sum to be equal to 0 when $C_i(\mathbf{x}, \mathbf{w})$ are not all equal to 0 individually. Finally, note that this sum can indeed be computed by a linear query equal to a weighted sum of the linear queries that compute $C_i(\mathbf{x}, \mathbf{w})$. We stress that this check only requires opening the output of this one value, instead of having to open all k G-gates as proposed earlier.

We note that for the naive method of opening the last k G-gates, the total query size and decision time increase quadratically with the input length. This quadratic growth happens because more G-gates must be opened as the input length increases [4], and each query becomes longer. In contrast, our optimization results in a linear increase in both total query size and decision time. It requires only a single query to recover the random linear combination of all the G-gates, with only the query length growing as the input length increases.

5.3 Details of the Optimized FL-PCP

We now give our optimized FL-PCP in detail, which we denote by $\Pi' = (\mathsf{ProofGen}, \mathsf{Query}, \mathsf{Decision})$, in Fig. 8. We show below that it fulfills completeness, soundness, and strong honest-verifier zero-knowledge properties. For simplicity, assume that the output of the last k G-gates are the outputs of $C_1, \ldots, C_k$. Here, the verifier wants to check that $C_i(\mathbf{x}, \mathbf{w}) = 0$ for each $i \in [k]$. Since all the coefficients of the polynomial p are a part of the proof vector, the verifier can compute $p(i)$ for any i by taking the inner product of the coefficient vector with the following vector: $(i^0, i^1 \ldots i^{deg})$ where deg is the degree of p. To compute $\sum_{i=1}^{k} \psi_i \cdot C_i(\mathbf{x}, \mathbf{w})$ the verifier computes $\sum_{i=1}^{k} \psi_i \cdot p(M - i + 1)$ using the query vector on line 19 of Fig. 8.

While we assume that the output of each C_i is the output of a G-gate, we can also accommodate circuits that consist only of affine gates. In particular, in our L_2-norm bound validation, circuits that check if a binary decomposition is correct are affine circuits. Since the output of this type of circuit is not the output of a G-gate, it cannot be revealed using polynomial evaluation. However, its output is simply a linear query on the instance-proof vector. We show that this modified version of FL-PCP satisfies Completeness, Soundness, and Strong honest-verifier zero-knowledge in the Appendix D.

6 EXPERIMENTAL EVALUATION

In this section, we present an experimental evaluation of PEAR.[5] For the underlying unrestricted IPFE scheme, we utilize the construction from [17], which we refer to as CLT. Details of this construction are provided in Appendix E for completeness. We implement circuits for the Binary Validation $\mathcal{R}_{\mathsf{BV}}$ and L_2-Norm Bound Validation $\mathcal{R}_{\mathsf{NBV}}$ relations as previously defined in Section 5.1.

We establish a baseline for comparison using the straightforward NIVA protocol based on additively homomorphic encryption (HE) combined with a non-interactive zero-knowledge proof of knowledge (NIZK). Recall that in this approach, the client encrypts their

[4] increase in input length is inclusive of the increase in depth
[5] We will open-source our implementation upon publication.

ProofGen$(1^\lambda, \mathbf{x}, \mathbf{w})$:

Evaluate $C_1(\mathbf{x}, \mathbf{w}), \ldots, C_k(\mathbf{x}, \mathbf{w})$
For $i \in [L]$:
  $\mathbf{f}[i][0] \leftarrow\$ \mathbb{F}$
  For $j \in [M]$:
    $\mathbf{f}[i][j] \leftarrow i^{th}$ input to the j^{th} G-gate
Use polynomial interpolation for $i \in [L]$ as follows:
  $\{(j, \mathbf{f}[i][j])\}_{j=0}^{M}$ to compute f_i
$p \leftarrow G(f_1, \ldots, f_L)$
Let $\mathbf{c}_p$ be the coefficients of p
$\boldsymbol{\pi} \leftarrow (\mathbf{w} || \mathbf{f}[1][0] \ldots \mathbf{f}[L][0] || \mathbf{c}_p)$
Return $\boldsymbol{\pi}$

Query$(1^\lambda, 1^d)$:

$r \leftarrow\$ \mathbb{F} \backslash [M]$
Compute linear queries for polynomial evaluation as follows:
  compute $\boldsymbol{\lambda}$ to evaluate $p(r)$
  For $i \in [L]$: compute $\boldsymbol{\lambda}_i$ to evaluate $f_i(r)$
$\boldsymbol{\psi} \leftarrow\$ \mathbb{F}^k$
$\mathbf{q}_1 \leftarrow \boldsymbol{\lambda}$
$\mathbf{q}_2, \ldots, \mathbf{q}_L \leftarrow \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_L$
$\mathbf{q}_{L+1} \leftarrow (\mathbf{0}^{d+h+L} || \sum_{i=1}^{k} \psi_i \cdot i^0, \ldots, \sum_{i=1}^{k} \psi_i \cdot i^{deg})$
st $\leftarrow (r, \boldsymbol{\psi})$
Return $(\mathbf{q}_1, \ldots, \mathbf{q}_{L+1}, \text{st})$

Decision(st, $a_1, \ldots, a_{L+1}$):

If $a_1 = G(a_2, \ldots, a_L)$ and $a_{L+1} = 0$: Return 1
Return 0

Figure 8: Optimized FL-PCP for Conjunction-of-Predicates.

plaintexts with the HE scheme and uses the NIZK to prove that the ciphertext is well-formed and satisfies the required constraints. The client then sends both the ciphertext and the proof to the server, who verifies the proof and aggregates the validated ciphertexts. In the following discussion, we refer to this approach as NIZK+HE.

On FHE+FL-PCP. We also attempted to compare PEAR with a second "off-the-shelf" NIVA protocol based on FHE+FL-PCP, where FL-PCP validation is performed under the hood of FHE. Recall that the FL-PCP decision algorithm computes a G-gate and compares the output with another value, denoting their difference as t. The decision algorithm determines the proof is valid if $t = 0$. It is straightforward for IPFE to check the value of t as it obtains the value in the clear. However, checking the same for the FHE approach is significantly more challenging, since the server does not interact with the client. We take the approach of exponentiating the FHE ciphertext of t to the order of the plaintext space to obtain a bit, where a zero bit indicates "valid". We then flip the bit and use it to multiply the corresponding ciphertext to enable aggregation without decrypting the intermediate result.

We implement this approach using the BFV scheme from the SEAL library [39], written in C++. The FL-PCP scheme should operate over a 128-bit field; however, the BFV scheme in SEAL naively supports a plaintext modulus of at most 2^{60}. Although longer integers could be managed by initializing multiple schemes and concatenating their ciphertexts, we omit this step and instead use a smaller plaintext size of 2^{60} for BFV to provide it with some advantage and demonstrate that PEAR already offers a significant speedup while also uses a much larger field. For the $\mathcal{R}_{\text{BV}}$ relation, the server runtime of the FHE+FL-PCP approach is 20× to over 100× slower than that of PEAR in our experiments. Therefore, we exclude the FHE+FL-PCP approach from our detailed analysis below.

Implementation setup. We implement PEAR in Sage [38], which uses a C backend, leveraging the PARI/GP library [26] to provide robust support of ideal class groups (for the CLT scheme). It also supports efficient polynomial computations over a field (for the FL-PCP). For the NIZK+HE approach, we use Bulletproofs and ElGamal implementations from the Tongsuo library [35], also written in C. We assume a network latency of 30 ms and a bandwidth of 100 Mbps. All experiments were conducted on a cloud server equipped with an Intel Xeon Platinum 8160 CPU (96 cores, 2.10 GHz) and 64 GB of DDR4 memory, running Ubuntu 22.04.

6.1 Server-Side Optimizations

We introduce server-side optimizations to PEAR tailored to the relations we consider and to the IPFE scheme CLT. We observe that, for both the $\mathcal{R}_{\text{BV}}$ and $\mathcal{R}_{\text{NBV}}$ relations, the G-gate used is a single multiplication gate. Additionally, to recover the inner product r of a ciphertext $\text{ct}_{\mathbf{x}}$ and a function key $\text{sk}_{\mathbf{y}}$, CLT.Eval first computes g^r, where g is a generator of a group. It then uses a trapdoor to take the discrete logarithm and recover the exponent, a process that remains significantly more expensive than basic group operations such as multiplication and exponentiation.

However, to validate the FL-PCP proof, recovering the exact values of $f_1(r)$ and $f_2(r)$ is unnecessary; the goal is simply to verify that their product *equals* $p(r)$. This can be done by instead checking that $g^{f_1(r) \cdot f_2(r)}$ equals $g^{p(r)}$. However, at least in the CLT scheme, there is no direct way to compute $f_1(r) \cdot f_2(r)$ in the exponent. Thus, we first recover the value of $f_1(r)$ as described above. Then during the decryption of $f_2(r)$, we skip the final step of using the trapdoor to take the discrete logarithm and compute $(g^{f_2(r)})^{f_1(r)}$ to obtain $g^{f_1(r) \cdot f_2(r)}$. Similarly, when decrypting $p(r)$, we skip the discrete logarithm step and just retrieve $g^{p(r)}$, then check equality of the resulting group elements. This optimization avoids two trapdoor discrete logarithm computations per input validation.

Another server-side optimization applies to the aggregation step. In the aggregation step of PEAR, we are only interested in the aggregate value of the first part of the input vectors. In the IPFE CLT, a ciphertext ct of a vector $(\mathbf{x} \| \boldsymbol{\pi})$ consists of components $(\text{ct}_0, \text{ct}_1, \ldots, \text{ct}_{d+m})$ where $d + m$ is the length of $(\mathbf{x} \| \boldsymbol{\pi})$. Here, ct_0 is the randomness used during encryption and ct_i is a function of ct_0 and $\mathbf{x}_i$. Thus, instead of adding the entire vectors, we can input the components $(\text{ct}_0, \text{ct}_1, \ldots, \text{ct}_d)$ to CLT.Add. This optimization reduces both server runtime and bandwidth.

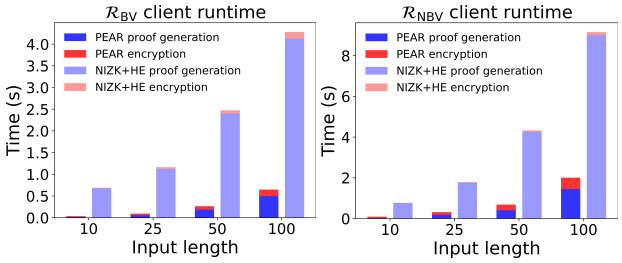

Figure 9: Client runtimes.

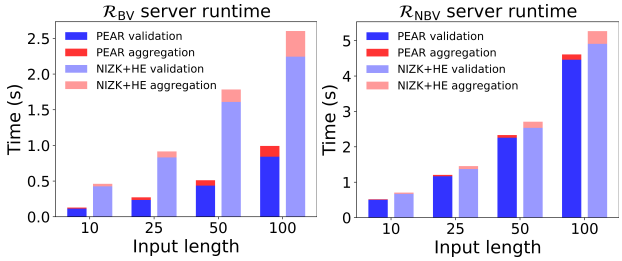

Figure 10: Server runtimes.

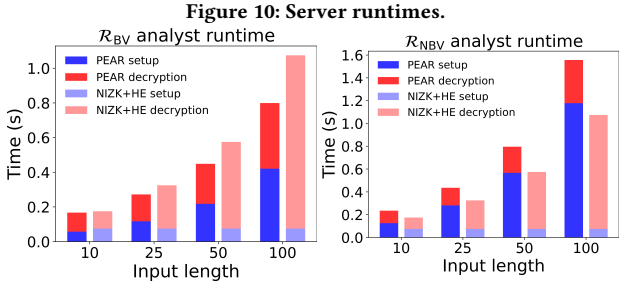

Figure 11: Analyst runtimes.

6.2 PEAR Evaluation

We benchmark the performance of PEAR and NIZK+HE with the $\mathcal{R}_{BV}$ and $\mathcal{R}_{NBV}$ validations. For the $\mathcal{R}_{NBV}$ relation, we assume that each x_i is bounded by 2^{10}, and the L_2-norm is bounded by 2^{20}.

Client performance. In Figure 9, client runtimes are split into two components: proof generation time and encryption time. The proof generation time for PEAR is significantly shorter than for NIZK+HE, primarily because the FL-PCP proof generation operates solely over the field. While the most computationally expensive step is interpolating polynomials over the field, we optimize this process to $O(n \log n)$ using the Fast Fourier Transform [27]. In contrast, Bulletproof's proof generation involves elliptic curve operations, which are inherently slower. On the other hand, encryption time for PEAR is slower than that of NIZK+HE. This is because in the case of the former the plaintext to be encrypted is longer, as it includes a proof and witness. However, since proof generation time dominates the client's computation, PEAR still achieves a speedup of 4.5× to 20× in client computation time compared to the NIZK+HE method.

Server performance. In Figure 10, the server's runtime is split into validation time and aggregation time. We note that validation time for NIZK+HE is larger since this requires verifying correctness of encryption as well as validity of the input. However, the relative cost of this decreases as the validity circuit grows. The reported aggregation time corresponds to adding two ciphertexts. In both approaches, aggregation time is independent of the circuit complexity, depending only on the input length. Overall, PEAR achieves a performance speedup of 1.2× to 3.6× in server computation time.

Analyst performance. Figure 11 presents the analyst's runtime which consists of running the setup procedure and decrypting the aggregated result. For PEAR, the Setup time is longer because it also involves executing CLT.KeyGen to generate queries for FL-PCP proof validation. Note that for both of these approaches, however, the runtime of the analyst is independent of the number of clients, which may be the dominating term in practice. Moreover, if more efficient unrestricted IPFE is developed in future work, it can be immediately plugged into our protocol to make the analyst's runtime much smaller, likely outperforming that for NIZK+HE.

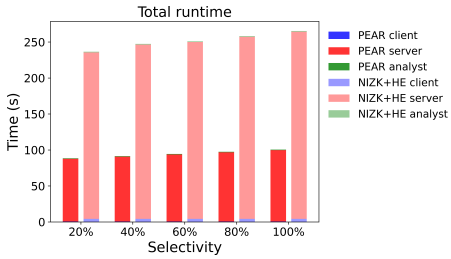

Figure 12: Total runtimes.

Overall Protocol. Figure 12 illustrates the total runtime of the protocol. The client time corresponds to a single client generating a proof for the $\mathcal{R}_{BV}$ relation and encrypting data with an input length of 100. The server time is measured with 100 participating clients, where selectivity refers to the percentage of valid client ciphertexts. Regardless of selectivity, the overall communication time remains constant. Under our network setting of 30 ms latency and 100 Mbps bandwidth, the analyst in both PEAR and NIZK+HE can retrieve the aggregated results within 40 ms. When running the entire protocol, server time dominates the total runtime; specifically, for both PEAR and NIZK+HE, server time contributes to over 99% of the total runtime. Hence, the overall speedup PEAR achieves is similar to the speedup it achieves in server time. We stress that our server runtime can be trivially reduced by parallel processing (as ciphertext validation and aggregation are independent across clients), which is not reflected in our experiments.

7 CONCLUSION

We have shown the first instantiation of a NIVA protocol based on IPFE and FL-PCP. Our result shows basic feasibility for this primitive and demonstrates that, with further optimizations of the underlying primitives, this can become practical for real-world use.

A number of open questions remain for further extensions of this primitive. We only consider the setting where the server is semi-honest. It is an open question if one can build a non-interactive, black-box protocol where the server is allowed to be malicious. A particularly challenging case is guaranteeing integrity when a malicious server may collude with an input client.

A second open question to consider is whether we can remove the analyst instead providing output directly to the server. This seems difficult to achieve in the non-interactive setting since the server could mix and match ciphertexts to learn more information about the messages, i.e., so called residual attacks. Handling *residual attacks* in a non-interactive setting where server and malicious clients collude remains an interesting open question.

ACKNOWLEDGMENTS

Arkady Yerukhimovich is supported in part by NSF grants CNS-2144798 (CAREER) and CNS-1955620.

REFERENCES

[1] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. 2015. Simple Functional Encryption Schemes for Inner Products. In *PKC 2015 (LNCS, Vol. 9020)*, Jonathan Katz (Ed.). Springer, Heidelberg, 733–751. https://doi.org/10.1007/978-3-662-46447-2_33

[2] Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. 2017. Multi-input Inner-Product Functional Encryption from Pairings. In *EUROCRYPT 2017, Part I (LNCS, Vol. 10210)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). Springer, Heidelberg, 601–626. https://doi.org/10.1007/978-3-319-56620-7_21

[3] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. 2022. Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares. In *Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13409)*, Clemente Galdi and Stanislaw Jarecki (Eds.). Springer, 516–539. https://doi.org/10.1007/978-3-031-14791-3_23

[4] Shweta Agrawal, Benoît Libert, and Damien Stehlé. 2016. Fully Secure Functional Encryption for Inner Products, from Standard Assumptions. In *CRYPTO 2016, Part III (LNCS, Vol. 9816)*, Matthew Robshaw and Jonathan Katz (Eds.). Springer, Heidelberg, 333–362. https://doi.org/10.1007/978-3-662-53015-3_12

[5] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*. 2087–2104.

[6] Laasya Bangalore, Mohammad Hossein Faghihi Sereshgi, Carmit Hazay, and Muthuramakrishnan Venkitasubramaniam. 2023. Flag: A Framework for Lightweight Robust Secure Aggregation. In *ASIACCS 23*, Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik (Eds.). ACM Press, 14–28. https://doi.org/10.1145/3579856.3595805

[7] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. 2023. ACORN: Input Validation for Secure Aggregation. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 4805–4822. https://www.usenix.org/conference/usenixsecurity23/presentation/bell

[8] James Bell-Clark, Adrià Gascón, Baiyu Li, Mariana Raykova, and Phillipp Schoppmann. 2024. Willow: Secure Aggregation with One-Shot Clients. Cryptology ePrint Archive, Paper 2024/936. https://eprint.iacr.org/2024/936

[9] Mihir Bellare and Phillip Rogaway. 2006. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In *EUROCRYPT 2006 (LNCS, Vol. 4004)*, Serge Vaudenay (Ed.). Springer, Heidelberg, 409–426. https://doi.org/10.1007/11761679_25

[10] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. 2015. Function-Hiding Inner Product Encryption. In *ASIACRYPT 2015, Part I (LNCS, Vol. 9452)*, Tetsu Iwata and Jung Hee Cheon (Eds.). Springer, Heidelberg, 470–491. https://doi.org/10.1007/978-3-662-48797-6_20

[11] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 1175–1191. https://doi.org/10.1145/3133956.3133982

[12] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2019. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In *Annual International Cryptology Conference*. Springer, 67–97.

[13] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight Techniques for Private Heavy Hitters. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 762–776. https://doi.org/10.1109/SP40001.2021.00048

[14] Florian Bourse. 2017. *Functional encryption for inner-product evaluations*. Theses. Université Paris sciences et lettres. https://theses.hal.science/tel-01665276

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

[16] Guilhem Castagnos and Fabien Laguillaumie. 2015. Linearly Homomorphic Encryption from DDH. In *CT-RSA 2015 (LNCS, Vol. 9048)*, Kaisa Nyberg (Ed.). Springer, Heidelberg, 487–505. https://doi.org/10.1007/978-3-319-16715-2_26

[17] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. 2018. Practical Fully Secure Unrestricted Inner Product Functional Encryption Modulo p. In *ASIACRYPT 2018, Part II (LNCS, Vol. 11273)*, Thomas Peyrin and Steven Galbraith (Eds.). Springer, Heidelberg, 733–764. https://doi.org/10.1007/978-3-030-03329-3_25

[18] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. 2022. EIFFeL: Ensuring Integrity for Federated Learning. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, 2535–2549. https://doi.org/10.1145/3548606.3560611

[19] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, Aditya Akella and Jon Howell (Eds.). USENIX Association, 259–282. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs

[20] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC 2006 (LNCS, Vol. 3876)*, Shai Halevi and Tal Rabin (Eds.). Springer, Heidelberg, 265–284. https://doi.org/10.1007/11681878_14

[21] Taher ElGamal. 1984. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO'84 (LNCS, Vol. 196)*, G. R. Blakley and David Chaum (Eds.). Springer, Heidelberg, 10–18.

[22] Uriel Feige, Dror Lapidot, and Adi Shamir. 1990. Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String (Extended Abstract). In *31st FOCS*. IEEE Computer Society Press, 308–317. https://doi.org/10.1109/FSCS.1990.89549

[23] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, Michael Mitzenmacher (Ed.). ACM Press, 169–178. https://doi.org/10.1145/1536414.1536440

[24] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *17th ACM STOC*. ACM Press, 291–304. https://doi.org/10.1145/22145.22178

[25] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer, 305–326.

[26] The PARI Group. 2024. PARI/GP Library. http://pari.math.u-bordeaux.fr. 2.17.1.

[27] Michael Hecht and Ivo F. Sbalzarini. 2019. Fast Interpolation and Fourier Transform in High-Dimensional Spaces. In *Intelligent Computing*, Kohei Arai, Supriya Kapoor, and Rahul Bhatia (Eds.). Springer International Publishing, Cham, 53–75.

[28] Harish Karthikeyan and Antigoni Polychroniadou. 2024. OPA: One-shot Private Aggregation with Single Client Interaction and its Applications to Federated Learning. Cryptology ePrint Archive, Paper 2024/723. https://eprint.iacr.org/2024/723

[29] Joe Kilian. 1992. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 723–732.

[30] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J Wu. 2018. Function-hiding inner product encryption is practical. In *Security and Cryptography for Networks: 11th International Conference, SCN 2018, Amalfi, Italy, September 5–7, 2018, Proceedings 11*. Springer, 544–562.

[31] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. 2023. LERNA: Secure Single-Server Aggregation via Key-Homomorphic Masking. In *ASIACRYPT 2023, Part I (LNCS, Vol. 14438)*, Jian Guo and Ron Steinfeld (Eds.). Springer, Heidelberg, 302–334. https://doi.org/10.1007/978-981-99-8721-4_10

[32] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. 2023. RoFL: Robustness of Secure Federated Learning. In *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 453–476. https://doi.org/10.1109/SP46215.2023.10179400

[33] Silvio Micali. 2000. Computationally sound proofs. *SIAM J. Comput.* 30, 4 (2000), 1253–1298.

[34] Adam O'Neill. 2010. Definitional Issues in Functional Encryption. Cryptology ePrint Archive, Report 2010/556. https://eprint.iacr.org/2010/556.

[35] OpenAtom. 2024. Tongsuo Library. https://www.tongsuo.net/en/docs/. 8.4.0.

[36] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 238–252. https://doi.org/10.1109/SP.2013.47

[37] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. 2023. ELSA: Secure Aggregation for Federated Learning with Malicious Actors. In *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1961–1979. https://doi.org/10.1109/SP46215.2023.10179468

[38] Sagemath. 2024. Sage Library. https://www.sagemath.org/development-ack.html. 10.5.0.

[39] SEAL 2023. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[40] Jianan Su, Laasya Bangalore, Harel Berger, Jason Yi, Alivia Castor, Micah Sherr, and Muthuramakrishnan Venkitasubramaniam. 2024. SCIF: Privacy-Preserving Statistics Collection with Input Validation and Full Security. Cryptology ePrint Archive, Paper 2024/1821. https://eprint.iacr.org/2024/1821

A IPFE CORRECTNESS AND SECURITY

Correctness. We require IPFE to be correct meaning it has:

(1) Evaluation correctness: For all $\lambda, d \in \mathbb{N}$, $\mathbf{x}, \mathbf{y} \in \mathbb{F}^d$:

$$\Pr[\mathsf{IPFE.Eval}(\mathsf{IPFE.KeyGen}(\mathsf{msk}, \mathbf{y}), \mathsf{IPFE.Encrypt}(\mathsf{mpk}, \mathbf{x})) = \langle \mathbf{x}, \mathbf{y} \rangle] = 1$$

where $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{IPFE.Setup}(1^\lambda, 1^d)$,

(2) Decryption correctness:

$$\Pr[\mathsf{IPFE.Decrypt}(\mathsf{msk}, \mathsf{IPFE.Encrypt}(\mathsf{mpk}, \mathbf{x})) = \mathbf{x}] = 1$$

for all $\lambda, d \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{msk}, \mathsf{pp}) \leftarrow\!\$\ \mathsf{IPFE.Setup}(1^\lambda, 1^d)$, $\mathbf{x} \in \mathbb{F}^d$. The probability is taken over the coins of the encryption algorithm.

Security. The ipfe-ind game is defined in Figure 13.

Game $\mathbf{G}^{\mathsf{ind}}_{\mathsf{IPFE},\mathcal{A}}$

Initialize$(1^\lambda, 1^d)$:
$b \leftarrow\!\$\ \{0,1\}$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{IPFE.Setup}(1^\lambda, 1^d)$
$V \leftarrow \emptyset$; $W \leftarrow \emptyset$
Return mpk

KeyGen$(\mathbf{y})$:
$V \leftarrow V \cup \{\mathbf{y}\}$
$\mathsf{sk}_\mathbf{y} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}, \mathbf{y})$
Return $\mathsf{sk}_\mathbf{y}$

Enc$(\mathbf{x}_0, \mathbf{x}_1)$:
$W \leftarrow W \cup \{(\mathbf{x}_0, \mathbf{x}_1)\}$
$\mathsf{ct}_{\mathbf{x}_b} \leftarrow\!\$\ \mathsf{IPFE.Encrypt}(\mathsf{mpk}, \mathbf{x}_b)$
Return $\mathsf{ct}_{\mathbf{x}_b}$

Finalize(b'):
if $\exists\ \mathbf{y} \in V$ and $(\mathbf{x}_0, \mathbf{x}_1) \in W$ such that:
  $\langle \mathbf{y}, \mathbf{x}_0 \rangle \neq \langle \mathbf{y}, \mathbf{x}_1 \rangle$
  then return 0
return $(b' = b)$

Figure 13: Indistinguishability-based security game for IPFE.

We consider a multi-challenge setting. For an adversary $\mathcal{A}$ interacting with the experiment in Figure 13, the advantage is defined as follows:

$$\mathbf{Adv}^{\mathsf{ind\text{-}multi}}_{\mathsf{IPFE},\mathcal{A}}(\lambda) = 2 \cdot \Pr[\mathbf{G}^{\mathsf{ind}}_{\mathsf{IPFE},\mathcal{A}}(\lambda) \Rightarrow 1] - 1\ .$$

B REUSABILITY OF BBG$^+$

B.1 Soundness

Lemma 1. *The FL-PCP* BBG$^+$*, satisfies γ-reusable soundness as defined in Section 4.3 where*

$$\gamma \leq \frac{Q(d+1)}{|\mathbb{F}| - (M + Qd)}$$

and Q is the number of queries made to Repeat*, M is the number of G-gates in the validation circuit, and d is the arithmetic degree of the gate G.*

Proof. We include BBG$^+$ here for completeness in Figure 14. We omit specifics of the QueryVecs algorithm for simplification. Here $\mathcal{R}$ is a relation such that $C_\mathcal{R}$ is the corresponding validation circuit that consists of G-gates with in-degree L. Additionally, let $\ell = L + 2$ be the query complexity.

We present the reusable soundness game modified to include a bad flag in Figure 15. The modifications are given in red. First, we show that the bad flag is set to 1 with negligible probability. Then, we show that if the bad flag is not set to 1 when we reach the Finalize procedure, any adversary $\mathcal{A}$ has a negligible advantage using arguments similar to the proof of soundness of BBG$^+$.

FL-PCP $\Pi_\mathcal{R}$

ProofGen$(1^\lambda, \mathbf{x}, \mathbf{w})$:
For $i \in [L]$:
  $z_i \leftarrow\!\$\ \mathbb{F}$
Run $C_\mathcal{R}(\mathbf{x}, \mathbf{w})$
Define $f_1, \ldots, f_L$ such that:
  $f_i(j)$ is the i^{th} input to the j^{th} G-gate
  z_i is the constant term of f_i
Define $p = G(f_1, \ldots, f_L)$
Define $\mathbf{c}_p$ vector of coefficients of p
Return $(\mathbf{w}, \mathbf{z}, \mathbf{c}_p)$

Query$(1^\lambda, 1^d)$:
st $\leftarrow\!\$\ \mathbb{F} \setminus [M]$
$(\mathbf{q}_1, \ldots, \mathbf{q}_\ell) \leftarrow \mathsf{QueryVecs}(\mathsf{st}, C_\mathcal{R}, G)$
Return $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \mathsf{st})$

Decision$(\mathsf{st}, a_1, \ldots, a_\ell)$:
If $G(a_1, \ldots, a_{\ell-2}) = a_{\ell-1}$ **and** $a_\ell = 0$:
Return 1
Else: Return 0

Figure 14: Fully Linear PCP BBG$^+$ **[12].**

Game $\mathbf{G}^{\mathsf{reuse\text{-}sound}}_{\Pi,\mathcal{A}}(\lambda, d)$

Initialize:
st $\leftarrow\!\$\ \mathbb{F} \setminus [M]$; bad $\leftarrow 0$
$(\mathbf{q}_1, \ldots, \mathbf{q}_\ell) \leftarrow \mathsf{QueryVecs}(\mathsf{st}, C_\mathcal{R}, G)$

Repeat$(\mathbf{x} \| \boldsymbol{\pi})$:
For $i \in [\ell]$: $a_i \leftarrow \langle (\mathbf{x} \| \boldsymbol{\pi}), \mathbf{q}_i \rangle$
If $G(a_1, \ldots, a_{\ell-2}) = a_{\ell-1}$ and $a_\ell = 0$: $b \leftarrow 1$
ELse $b \leftarrow 0$
bad $\leftarrow$ CheckBadEvent$(\mathsf{st}, \mathbf{x} \| \boldsymbol{\pi})$
Return b

Finalize$(\mathbf{x}^* \| \boldsymbol{\pi}^*)$:
If $\mathbf{x}^* \in \mathcal{R}$: Return $\perp$
For $i \in [\ell]$: $a^*_i \leftarrow \langle (\mathbf{x}^* \| \boldsymbol{\pi}^*), \mathbf{q}_i \rangle$
If $G(a^*_1, \ldots, a^*_{\ell-2}) = a^*_{\ell-1}$ and $a^*_\ell = 0$: $b^* \leftarrow 1$
Else $b^* \leftarrow 0$
Return b^*

CheckBadEvent$(\mathsf{st}, \mathbf{x} \| \boldsymbol{\pi})$:
Compute polynomial $p - G(f_1, \ldots, f_{\ell-2})$ from $\mathbf{x} \| \boldsymbol{\pi}$
If $p - G(f_1, \ldots, f_{\ell-2})$ is not identically zero:
  Compute roots of the polynomial as $r_1, \ldots, r_d$
  If there exists $i \in [d]$ such that $r_i = \mathsf{st}$: Return 1
Else: Return 0

Figure 15: Game defining reusable soundness property for BBG$^+$**. Modifications to include a bad flag are given in red.**

First, we analyze the input $(\mathbf{x}^* \| \boldsymbol{\pi}^*)$ to the Finalize procedure. A valid input to this procedure must contain an input $\mathbf{x}^*$ such that $\mathbf{x}^* \notin \mathcal{R}$. We parse $\boldsymbol{\pi}^*$ as $\mathbf{w}^* \| z^*_1, \ldots z^*_L \| \mathbf{c}_{p^*}$ where:

(1) $\mathbf{w}^*$ is the witness for $\mathbf{x}^*$.
(2) $z^*_1, \ldots, z^*_L$ are some constants.
(3) $\mathbf{c}_{p^*}$ is a vector of coefficients of a polynomial p^*.

For $(\mathbf{x}^* \| \boldsymbol{\pi}^*)$, let $f^*_1, \ldots, f^*_{\ell-2}$ and p^* be the polynomials purported to be associated with the input and output wires of the evaluation of $C_\mathcal{R}(\mathbf{x}^* \| \mathbf{w}^*)$. Additionally, let a^*_i be equal to $\langle (\mathbf{x}^* \| \boldsymbol{\pi}^*), \mathbf{q}_i \rangle$ for each $i \in [\ell]$.

If the Finalize procedure outputs 1 when $\mathbf{x}^* \notin \mathcal{R}$, this means that both $G(a^*_1, \ldots, a^*_{\ell-2}) = a^*_{\ell-1}$ and $a^*_\ell = 0$. However, since there

does not exist a witness $\mathbf{w}^*$ such that $C(\mathbf{x}^*, \mathbf{w}^*) = 0$, it must be that p^* represented by the coefficient vector $\mathbf{c}_{p^*}$ is not the same as $G(f_1^*, \ldots, f_{\ell-2}^*)$. Hence, $p^* - G(f_1^*, \ldots, f_{\ell-2}^*)$ is not an identically zero polynomial and the state value st must be a root of this polynomial.

Next, consider the queries made to the RepeatO procedure. Note that in the RepeatO oracle, we do not impose a restriction on whether the submitted $\mathbf{x}_i$ must be in $\mathcal{R}$. The adversary can submit any combination of valid and invalid input $\mathbf{x}$, witness $\mathbf{w}$, and proof $\boldsymbol{\pi}$. However, the adversary can only get more information about the state st if the polynomial $p^* - G(f_1^*, \ldots, f_{\ell-2}^*)$ is not identically zero similar to the case of the Finalize procedure discussed above. Additionally, we can assume that an adversary submits queries such that this polynomial has distinct roots (including roots of polynomials corresponding to previous queries) to maximize information gain.

Claim: The probability of bad flag being set to 1 when we reach the Finalize procedure is given as follows:

$$\Pr[\text{bad} = 1 \text{ at } \textsc{Finalize}] \leq \frac{Qd}{|\mathbb{F}| - (M + Qd)}$$

where the probability is taken over the coins of the Initialize procedure.

The bad flag is set to 1 when we reach the Finalize procedure if for at least one $j \in [Q]$, the state st is a root of the polynomial $p^j - G(f_1^j, \ldots, f_{\ell-2}^j)$ corresponding to the j^{th} query $(\mathbf{x}_j \| \boldsymbol{\pi}_j)$ made by the adversary $\mathcal{A}$ to the RepeatO oracle. The probability of this event can be calculated as follows:

$$\begin{aligned}
\Pr[\text{bad} = 1 \text{ at } \textsc{Finalize}] &= \sum_{j \in [N]} \Pr[\text{bad} = 1 \text{ for the first time at } j] \\
&= \sum_{j \in [Q]} (\Pr[\text{bad} = 0 \text{ at } j-1] \cdot \Pr[\text{bad} = 1 \text{ at } j]) \\
&= \sum_{j \in [Q]} \left(\prod_{k<j} \left(1 - \frac{d}{|\mathbb{F}| - (M + kd)}\right) \cdot \frac{d}{|\mathbb{F}| - (M + jd)} \right) \\
&\leq \sum_{j \in [Q]} \frac{d}{|\mathbb{F}| - (M + jd)} \\
&\leq \frac{Qd}{|\mathbb{F}| - M - Qd} \,.
\end{aligned}$$

The third equality follow from the fact that at query j, the probability that the state is a root of the corresponding polynomial $p^j - G(f_1^j, \ldots, f_{\ell-2}^j)$ is given by $\frac{d}{|\mathbb{F}| - (M + jd)}$.

Claim: For an adversary $\mathcal{A}$, the following holds:

$$\Pr[\mathbf{G}^{\text{reuse-sound}}_{\Pi,\mathcal{A}}(\lambda, d) \Rightarrow 1 | \text{bad} = 0 \text{ at } \textsc{Finalize}] \leq \frac{d}{|\mathbb{F}| - (M + Qd)}$$

where Q is the total number of queries made to the RepeatO oracle and the probability is taken over the coins of the Initialize procedure.

In game $\mathbf{G}^{\text{reuse-sound}}_{\Pi,\mathcal{A}}$, when the output is 1 the state value st is the root of the polynomial $p^* - G(f_1^*, \ldots, f_{\ell-2}^*)$ corresponding to $\mathbf{x}^* \| \boldsymbol{\pi}^*$, as previously established. From the construction of the game, we know that st is sampled uniformly at random from the set $\mathbb{F} \setminus [M]$. If the bad flag is 0, then the adversary did not find the state st during one of its RepeatO queries. In more detail, there does not exist a query $(\mathbf{x}_j \| \boldsymbol{\pi}_j)$ to the RepeatO oracle such that the state st is a root of the corresponding polynomial $p^j - G(f_1^j, \ldots, f_{\ell-2}^j)$. Thus, using similar arguments as the proof of soundness of BBG$^+$ and the Schwartz-Zippel lemma, the probability of this event is given by $\frac{d}{|\mathbb{F}| - (M+Qd)}$.

Finally, combining the two scenarios of the value of bad at Finalize, we get that the soundness parameter γ is $\frac{Q(d+1)}{|\mathbb{F}| - (M+Qd)}$ as claimed. □

Remark B.1. *The FL-PCP described in Section 5 and Fig. 8 satisfies reusable soundness the proof of which is identical to the above proof.*

B.2 Strong Honest-Verifier Zero-Knowledge

Lemma 2. BBG$^+$ *satisfies reusable strong HVZK as defined in Section 4.3.*

Proof. In order to prove that BBG$^+$ satisfies reusable strong HVZK, we first provide a simulator $\mathcal{S}_{\Pi,\mathcal{R}} = (\mathcal{S}_1, \mathcal{S}_2)$ in Figure 16. Then, we show that when using this simulator, the output of the two experiments are identically distributed.

Simulator $\mathcal{S}$

$\mathcal{S}_1(1^\lambda, 1^d)$:
1 $r \leftarrow\$ \mathbb{F}\setminus[M]$
2 st $\leftarrow r$
3 Return st

$\mathcal{S}_2(1^\lambda, 1^d, \text{st})$:
4 If st $\notin \mathbb{F}\setminus[M]$:
5 Return $\perp$
6 $a_1, \ldots, a_{\ell-2} \leftarrow\$ \mathbb{F}$
7 $a \leftarrow G(a_1, \ldots, a_{\ell-1}) \in \mathbb{F}$
8 Return $(a_1, \ldots, a_{\ell-2}, a, 0) \in \mathbb{F}^{\ell+1}$

Figure 16: Simulator for reusable strong HVZK for BBG$^+$.

The output of the Initialize procedure in the game $\mathbf{G}^{\text{rshvzk-1}}_{\Pi,\mathcal{A}}$ being distributed identically to that in the game $\mathbf{G}^{\text{rshvzk-0}}_{\Pi,\mathcal{A}}$ follows from the fact that the output state st is sampled uniformly at random from the same domain $\mathbb{F} \setminus [M]$ in both games.

Next, we want to show that the responses from the Repeat oracle are also identically distributed in both games. This can be done using an argument similar to the one made in the proof of zero-knowledge in BBG$^+$. First, we recall that an honest verifier samples r in order to check that the polynomials p and $G(f_1, \ldots, f_{\ell-2})$ are constructed correctly by evaluating them at r using query vectors $\mathbf{q}_1, \ldots, \mathbf{q}_\ell$. Among $(a_1, \ldots, a_{\ell-2}, a_{\ell-1}, a_\ell)$, $a_\ell = 0$ and $a_{\ell-1} = G(a_1, \ldots, a_{\ell-2})$ in the output of both games when $\mathbf{x} \in \mathcal{R}$. Since these values are identical, it suffices to show that the distribution of $(a_1, \ldots, a_{\ell-2})$ is identical in the two games.

Here, $(a_1, \ldots, a_{\ell-2})$ in the game $\mathbf{G}^{\text{rshvzk-0}}_{\Pi,\mathcal{A}}$ simulates the values $(f_1(r), \ldots, f_{\ell-2}(r))$ in the game $\mathbf{G}^{\text{rshvzk-1}}_{\Pi,\mathcal{A}}$. For every $i \in [\ell - 2]$, $f_i(r)$ can be written in terms of the Lagrange interpolating polynomials:

$$f_i(r) = \lambda_0(r) \cdot f_i(0) + \sum_{j=1}^{M} \lambda_j(r) \cdot f_i(j) \,.$$

When $r \notin [M]$, the value of the zero-th interpolating polynomial $\lambda_0(r)$ is non-zero. We also know that each $f_i(0)$ is sampled uniformly from $\mathbb{F}$ in the ProofGen algorithm for each query $(\mathbf{x}, \mathbf{w})$. Thus,

$\lambda_0(r)\cdot f_i(0)$ is uniform random and the distribution of $f_i(r)$ is uniform random as well.

On the other hand, the simulator in $\mathcal{S}_2$ samples $a_1,\ldots,a_{\ell-2}\in\mathbb{F}$ uniformly at random. Therefore, the distribution of $(a_1,\ldots,a_{\ell-2})$ is identical in the two games.

Hence, the responses from the Repeat oracle are identically distributed in both games and the advantage of any adversary is 0. □

Remark B.2. *The FL-PCP described in Section 5 and Figure 8 satisfies reusable strong HVZK, the proof of which is identical to the above except that $\mathcal{S}_1$ also samples an additional vector $\boldsymbol{\psi}\leftarrow\!\!\$\ \mathbb{F}^k$ and returns it as part of its output.*

C PEAR PROOF OF PRIVACY

Here we prove Theorem 1 showing that PEAR is private against a malicious server. Let $\mathcal{A}$ be an adversary against the privacy of PEAR. Consider the game sequence in Fig. 17. The differences between consecutive games are highlighted in red. The first game $\mathbf{G}^1_{\mathsf{PEAR},\mathcal{A}}$ is the indistinguishability game from Fig. 2. In the second game $\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}$, instead of encrypting $(\mathbf{x}^b\|\boldsymbol{\pi})$, we compute its inner product with each query vector, sample a vector $\mathbf{v}^*$ that has the same inner products with the query vectors, and encrypt that vector instead. The transition between these two games is justified using security of the IPFE scheme Δ and preimage sampleability of the inner product functionality. Next, in game $\mathbf{G}^3_{\mathsf{PEAR},\mathcal{A},\mathcal{S}}$, we use the FL-PCP simulator to generate the inner product values and sample a vector that has these inner products with the query vectors. This game is independent of the bit b.

The following lemmas correspond to the games:

Lemma 3. *For all $\lambda,d\in\mathbb{N}$, there exists an adversary $\mathcal{B}$ such that:*

$$\Pr[\mathbf{G}^1_{\mathsf{PEAR},\mathcal{A}}(\lambda,d)\Rightarrow 1]-\Pr[\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}(\lambda,d)\Rightarrow 1]=\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{IPFE},\mathcal{B}}(\lambda,d).$$

Lemma 4. *There exists a simulator $\mathcal{S}_\Pi$ such that the following equation holds:*

$$\Pr[\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}(\lambda,d)\Rightarrow 1]=\Pr[\mathbf{G}^3_{\mathsf{PEAR},\mathcal{A},\mathcal{S}}(\lambda,d)\Rightarrow 1].$$

Finally, we can see that in game $\Pr[\mathbf{G}^3_{\mathsf{PEAR},\mathcal{A},\mathcal{S}_\Pi}(\lambda)]$, the output of the game is independent of the bit b. Thus,

$$\Pr[\mathbf{G}^3_{\mathsf{PEAR},\mathcal{A},\mathcal{S}}(\lambda,d)\Rightarrow 1]=\frac{1}{2}.$$

Next, we combine this fact with the lemmas as follows:

$$\begin{aligned}\Pr[\mathbf{G}^{\mathsf{ind}}_{\mathsf{PEAR},\mathcal{A}}(\lambda)\Rightarrow 1]&=\Pr[\mathbf{G}^1_{\mathsf{PEAR},\mathcal{A}}(\lambda)\Rightarrow 1]\\&=\Big(\Pr[\mathbf{G}^1_{\mathsf{PEAR},\mathcal{A}}(\lambda)\Rightarrow 1]-\Pr[\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}(\lambda)\Rightarrow 1]\Big)\\&\quad+\Pr[\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}(\lambda)\Rightarrow 1]\\&=\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{IPFE},\mathcal{B}}(\lambda,d)+\Pr[\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}(\lambda)\Rightarrow 1]\\&=\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{IPFE},\mathcal{B}}(\lambda,d)+\Pr[\mathbf{G}^3_{\mathsf{PEAR},\mathcal{A},\mathcal{S}}(\lambda)\Rightarrow 1]\\&=\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{IPFE},\mathcal{B}}(\lambda,d)+\frac{1}{2}.\end{aligned}$$

Finally,

$$\begin{aligned}\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{PEAR},\mathcal{A}}(\lambda,d)&=2\cdot\Pr[\mathbf{G}^{\mathsf{ind}}_{\mathsf{PEAR},\mathcal{A}}(\lambda)\Rightarrow 1]-1\\&=\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{IPFE},\mathcal{B}}(\lambda,d).\end{aligned}$$

Now, it suffices to prove Lemmas 3 and 4.

Proof of Lemma 3. First, consider adversary $\mathcal{B}$ provided in Fig. 18. The adversary simulates the encryption oracle for the adversary $\mathcal{A}$ by generating both messages $(\mathbf{x}^b||\boldsymbol{\pi})$ and $\mathbf{v}^*$ corresponding to the two games and making an IPFE encryption query with them. By inspection, we can see that when $\mathcal{B}$ has its challenge bit bit = 0, it perfectly simulates game $\mathbf{G}^1_{\mathsf{PEAR},\mathcal{A}}$ in Fig. 17.

On the other hand, by definition of VecSamp, the inner products of vector $\mathbf{v}^*$ with every vector $\mathbf{q}_j$ is equal to the inner products of vector $(\mathbf{x}^b||\boldsymbol{\pi})$ with every $\mathbf{q}_j$ satisfying the validity requirement of the encryption oracle. Here we use preimage sampleability of the inner product function in the VecSamp to compute the vector $\mathbf{v}^*$. Thus, when $\mathcal{B}$ has its challenge bit bit = 0, it perfectly simulates $\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}$ for $\mathcal{A}$. Finally,

$$\Pr[\mathbf{G}^1_{\mathsf{PEAR},\mathcal{A}}(\lambda,d)\Rightarrow 1]-\Pr[\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}(\lambda,d)\Rightarrow 1]=\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{IPFE},\mathcal{B}}(\lambda,d).$$

□

Proof of Lemma 4. The difference between games $\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}$ and $\mathbf{G}^3_{\mathsf{PEAR},\mathcal{A},\mathcal{S}_\Pi}$ is in the generation of the state st_Π and the inner product values $a_1,\ldots,a_\ell$. The existence of a simulator $\mathcal{S}_{\Pi,\mathcal{R}}=(\mathcal{S}_1,\mathcal{S}_2)$ whose output is identically distributed to the real state st_Π and inner product values $a_1,\ldots,a_\ell$ is guaranteed by the reusable strong HVZK property when the input instances are valid. Thus, if the distribution of these values is identical, then the output of the games is also identical for any adversary $\mathcal{A}$. Hence,

$$\Pr[\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}(\lambda,d)\Rightarrow 1]=\Pr[\mathbf{G}^3_{\mathsf{PEAR},\mathcal{A},\mathcal{S}}(\lambda,d)\Rightarrow 1].$$

□

D FL-PCP OPTIMIZATION PROOFS

Here, we show that this modified version of the FL-PCP instance satisfies Completeness, Soundness, and Strong honest-verifier zero-knowledge.

Claim D.1. *(Completeness.) Π' is complete. That is, for all $(\mathbf{x},\mathbf{w})\in\mathcal{R}$, the verifier always accepts the proof. More precisely, for all $(\mathbf{x},\mathbf{w})\in\mathcal{R}$, $\Pr[\mathsf{Decision}(\mathsf{st},a_1,\ldots,a_{L+1})\Rightarrow 1]=1$ where $\boldsymbol{\pi}\leftarrow\!\!\$\ \mathsf{ProofGen}(1^\lambda,\mathbf{x},\mathbf{w})$, $(\mathbf{q}_1,\ldots,\mathbf{q}_{L+1},\mathsf{st})\leftarrow\!\!\$\ \mathsf{Query}(1^\lambda,1^d)$, and $a_i\leftarrow\langle(\mathbf{x},\boldsymbol{\pi}),\mathbf{q}_i\rangle$ for all $i\in[L+1]$. The probability is taken over the coins of the* ProofGen *and* Query *algorithms.*

Proof. First, we know that by construction of the output polynomial p, if the prover honestly generates the polynomials $f_1,\ldots,f_L,p$, then the first L linear checks pass. Additionally, we can see that if $x_i\in\mathcal{R}$, then for all $i\in[k]: C_i(\mathbf{x},\mathbf{w})=0$. Hence, $\psi_i\cdot C_i(\mathbf{x},\mathbf{w})=0$ for any ψ_i, and thus the sum always equals 0 as required. □

Claim D.2. *(Soundness.) Π' is sound. More precisely, for $(\mathbf{x},\mathbf{w})\in\mathbb{F}^d\times\mathbb{F}^h$, for any claimed proof $\boldsymbol{\pi}$ such that $|\boldsymbol{\pi}|=|\mathsf{ProofGen}(1^\lambda,\mathbf{x},\mathbf{w})|$, $(\mathbf{q}_1,\ldots,\mathbf{q}_{L+1},\mathsf{st})\leftarrow\!\!\$\ \mathsf{Query}(1^\lambda,1^d)$, and $a_i\leftarrow\langle(\mathbf{x},\boldsymbol{\pi}),\mathbf{q}_i\rangle$ for all $i\in[L+1]$ if $\mathsf{Decision}(\mathsf{st},a_1,\ldots,a_{L+1})$ outputs 1, then $(\mathbf{x},\mathbf{w})\notin\mathcal{R}$ with probability at most $\frac{degG}{|\mathbb{F}|-M}+\frac{1}{|\mathbb{F}|}$. The probability is taken over the coins of the* ProofGen *and* Query *algorithms.*

Game $\mathbf{G}^1_{\mathsf{PEAR},\mathcal{A}}(\lambda, d)$

INITIALIZE$(1^\lambda, 1^d)$:
1 $b \leftarrow\$ \{0,1\}$
2 $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \Delta.\mathsf{Setup}(1^\lambda, 1^d)$
3 $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \mathsf{st}_\Pi) \leftarrow \Pi_\mathcal{R}.\mathsf{Query}(1^\lambda, 1^d)$
4 For $j \in [\ell]$:
5 $\quad \mathsf{sk}_{\mathbf{q}_j} \leftarrow \Delta.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{q}_j)$
6 $\mathsf{vk} \leftarrow \left(\{\mathsf{sk}_{\mathbf{q}_j}\}_{j\in[\ell]}, \mathsf{st}_\Pi\right)$
7 Return $(\mathsf{mpk}, \mathsf{vk})$

ENC$((\mathbf{x}^0, \mathbf{w}^0), (\mathbf{x}^1, \mathbf{w}^1))$:
8 $\boldsymbol{\pi} \leftarrow\$ \Pi_\mathcal{R}.\mathsf{ProofGen}(1^\lambda, \mathbf{x}^b, \mathbf{w}^b)$
9 $\mathsf{ct} \leftarrow\$ \Delta.\mathsf{Encrypt}(\mathsf{mpk}, (\mathbf{x}^b \| \boldsymbol{\pi}))$
10 return ct

FINALIZE(b'):
11 return $(b = b')$

Game $\mathbf{G}^2_{\mathsf{PEAR},\mathcal{A}}(\lambda, d)$

INITIALIZE$(1^\lambda, 1^d)$:
1 $b \leftarrow\$ \{0,1\}$
2 $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \Delta.\mathsf{Setup}(1^\lambda, 1^d)$
3 $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \mathsf{st}_\Pi) \leftarrow \Pi_\mathcal{R}.\mathsf{Query}(1^\lambda, 1^d)$
4 For $j \in [\ell]$:
5 $\quad \mathsf{sk}_{\mathbf{q}_j} \leftarrow \Delta.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{q}_j)$
6 $\mathsf{vk} \leftarrow \left(\{\mathsf{sk}_{\mathbf{q}_j}\}_{j\in[\ell]}, \mathsf{st}_\Pi\right)$
7 Return $(\mathsf{mpk}, \mathsf{vk})$

ENC$((\mathbf{x}^0, \mathbf{w}^0), (\mathbf{x}^1, \mathbf{w}^1))$:
8 $\boldsymbol{\pi} \leftarrow\$ \Pi_\mathcal{R}.\mathsf{ProofGen}(1^\lambda, \mathbf{x}^b, \mathbf{w}^b)$
9 For $i \in [\ell]$: $a_i \leftarrow \langle (\mathbf{x}^b \| \boldsymbol{\pi}), \mathbf{q}_i \rangle$
10 $\mathbf{v}^* \leftarrow \mathsf{VecSamp}((\mathbf{q}_1, a_1), \ldots, (\mathbf{q}_\ell, a_\ell))$
11 $\mathsf{ct} \leftarrow\$ \Delta.\mathsf{Encrypt}(\mathsf{mpk}, \mathbf{v}^*)$
12 return ct

FINALIZE(b'):
13 return $(b = b')$

VecSamp$((\mathbf{y}_1, a_1), \ldots, (\mathbf{y}_\ell, a_\ell))$:
14 Let $\mathbf{v}^* \in \mathbb{F}^{d+m}$
15 *Solve the system of linear equations for* $\mathbf{v}^*$:
16 $\quad (\mathbf{y}_1 \| \ldots \| \mathbf{y}_\ell)^\top \cdot \mathbf{v}^* = (a_1 \| \ldots \| a_\ell)$
17 return $\mathbf{v}^*$

Game $\mathbf{G}^3_{\mathsf{PEAR},\mathcal{A},\mathcal{S}}(\lambda, d)$

INITIALIZE$(1^\lambda, 1^d)$:
1 $b \leftarrow\$ \{0,1\}$
2 $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \Delta.\mathsf{Setup}(1^\lambda, 1^d)$
3 $\mathsf{st}_\Pi \leftarrow\$ \mathcal{S}_1(1^\lambda, 1^d)$
4 $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell) \leftarrow \Pi_\mathcal{R}.\mathsf{Query}(1^\lambda, 1^d; \mathsf{st}_\Pi)$
5 For $j \in [\ell]$:
6 $\quad \mathsf{sk}_{\mathbf{q}_j} \leftarrow \Delta.\mathsf{KeyGen}(\mathsf{msk}, \mathbf{q}_j)$
7 $\mathsf{vk} \leftarrow \left(\{\mathsf{sk}_{\mathbf{q}_j}\}_{j\in[\ell]}, \mathsf{st}_\Pi\right)$
8 Return $(\mathsf{mpk}, \mathsf{vk})$

ENC$((\mathbf{x}^0, \mathbf{w}^0), (\mathbf{x}^1, \mathbf{w}^1))$:
9 $(a_1, \ldots, a_\ell) \leftarrow\$ \mathcal{S}_2(1^\lambda, 1^d, \mathsf{st}_\Pi)$
10 $\mathbf{v}^* \leftarrow \mathsf{VecSamp}((\mathbf{q}_1, a_1), \ldots, (\mathbf{q}_\ell, a_\ell))$
11 $\mathsf{ct} \leftarrow\$ \Delta.\mathsf{Encrypt}(\mathsf{mpk}, \mathbf{v}^*)$
12 return ct

FINALIZE(b'):
13 return $(b = b')$

VecSamp$((\mathbf{y}_1, a_1), \ldots, (\mathbf{y}_\ell, a_\ell))$:
14 Let $\mathbf{v}^* \in \mathbb{F}^{d+m}$
15 *Solve the system of linear equations for* $\mathbf{v}^*$:
16 $\quad (\mathbf{y}_1 \| \ldots \| \mathbf{y}_\ell)^\top \cdot \mathbf{v}^* = (a_1 \| \ldots \| a_\ell)$
17 return $\mathbf{v}^*$

Figure 17: Games for proof of privacy for Theorem 1.
Changes between consecutive games are depicted in red.

Adversary $\mathcal{B}^{\mathrm{KEYGEN}'(\cdot), \mathrm{ENC}'(\cdot,\cdot)}(1^\lambda, 1^d, \mathsf{mpk})$

INITIALIZE:
1 $b \leftarrow\$ \{0,1\}$
2 $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell, \mathsf{st}_\Pi) \leftarrow\$ \Pi_\mathcal{R}.\mathsf{Query}(1^\lambda, 1^d)$
3 For $j \in [\ell]$:
4 $\quad \mathsf{sk}_{\mathbf{q}_j} \leftarrow \mathrm{KEYGEN}'(\mathbf{q}_j)$
5 $\mathsf{vk} \leftarrow \left(\{\mathsf{sk}_{\mathbf{q}_j}\}_{j\in[\ell]}, \mathsf{st}_\Pi\right)$
6 Return $(\mathsf{mpk}, \mathsf{vk})$

ENC$((\mathbf{x}^0, \mathbf{w}^0), (\mathbf{x}^1, \mathbf{w}^1))$:
7 $\boldsymbol{\pi} \leftarrow \Pi_\mathcal{R}.\mathsf{ProofGen}(1^\lambda, \mathbf{x}^b, \mathbf{w}^b)$
8 For $i \in [\ell]$: $a_i \leftarrow \langle (\mathbf{x}^b \| \boldsymbol{\pi}), \mathbf{q}_i \rangle$
9 $\mathbf{v}^* \leftarrow \mathsf{VecSamp}((\mathbf{q}_1, a_1), \ldots, (\mathbf{q}_\ell, a_\ell))$
10 $\mathsf{ct} \leftarrow\$ \mathrm{ENC}'((\mathbf{x}^b \| \boldsymbol{\pi}), \mathbf{v}^*)$
11 return ct

FINALIZE(b'):
12 return $(b = b')$

VecSamp$((\mathbf{y}_1, a_1), \ldots, (\mathbf{y}_\ell, a_\ell))$:
13 Let $\mathbf{v}^* \in \mathbb{F}^{d+m}$
14 *Solve the system of linear equations for* $\mathbf{v}^*$:
15 $\quad (\mathbf{y}_1 \| \ldots \| \mathbf{y}_\ell)^\top \cdot \mathbf{v}^* = (a_1 \| \ldots \| a_\ell)$
16 return $\mathbf{v}^*$

Figure 18: IND adversary for privacy proof for PEAR.

Proof. Using the soundness argument of BBG$^+$, by the Schwartz-Zippel Lemma, we know that the probability that $p(r) - G(f_1(r), \ldots, f_L(r)) = 0$ for $r \leftarrow\$ \mathbb{F}\backslash[M]$ is equal to $\frac{\deg G}{|\mathbb{F}|-M}$.

Next, consider the probability that:

$$\sum_{i=1}^{k} (\psi_i \cdot C_i(\mathbf{x}, \mathbf{w})) = 0$$

where at least one of the $C_i(\mathbf{x}, \mathbf{w}) \neq 0$. This probability can be computed as follows:

$$\Pr\left[\psi_k \cdot C_k(\mathbf{x}, \mathbf{w}) = -\sum_{i=1}^{k-1} \psi_i \cdot C_i(\mathbf{x}, \mathbf{w}) \middle| \ \psi_i \leftarrow\$ \mathbb{F} \ \ \forall i \in [k-1]\right] = \frac{1}{|F|}.$$

Thus, with probability $\frac{1}{|\mathbb{F}|}$, $\sum_{i=1}^{k} \psi_i \cdot C_i(\mathbf{x}, \mathbf{w}) = 0$, when $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$. The verifier accepts when $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$ with the total probability at most $\frac{\deg G}{|\mathbb{F}|-M} + \frac{1}{|\mathbb{F}|}$. □

Claim D.3. *(Strong honest-verifier zero-knowledge.)* Π' *is SHVZK. More precisely, there exists a simulator* S *such that for all* $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ *and* $\lambda, d \in \mathbb{N}$, *the distribution of* $S(1^\lambda, 1^d)$ *is identical to the following:*

$$(\mathsf{st}, \langle (\mathbf{x} || \boldsymbol{\pi}), \mathbf{q}_1 \rangle, \ldots, \langle (\mathbf{x} || \boldsymbol{\pi}), \mathbf{q}_{L+1} \rangle), \ (\mathbf{q}_1, \ldots, \mathbf{q}_{L+1}))$$

where $(\mathbf{q}_1, \ldots, \mathbf{q}_{L+1}, \mathsf{st}) \leftarrow\$ \mathsf{Query}(1^\lambda, 1^d)$ *and* $\boldsymbol{\pi} \leftarrow\$ \mathsf{ProofGen}(1^\lambda, \mathbf{x}, \mathbf{w})$.

Proof. Note that the queries of the verifier are determined by the state of the verifier and the circuits. So, it suffices to simulate $(\mathsf{st}, f_1(r), \ldots, f_L(r), \langle (\mathbf{x}, \pi), \mathbf{q}_{L+1} \rangle)$. The required simulator S is as follows:

(1) $r \leftarrow\$ \mathbb{F}\backslash[M]$.
(2) $\boldsymbol{\psi} \leftarrow\$ \mathbb{F}^k$.
(3) $a_1, \ldots, a_L \leftarrow\$ \mathbb{F}$.
(4) $\mathsf{st} \leftarrow (r, \boldsymbol{\psi})$
(5) Return $(\mathsf{st}, a_1, \ldots, a_L, 0) \in \mathbb{F}^{L+k+1}$.

The distribution of $r, a_1, \ldots, a_L$ sampled in the simulator is identical to the distribution of $r, f_1(r), \ldots, f_L(r)$ in the real interaction. This argument closely resembles the proof of strong honest verifier zero knowledge for the FL-PCP given in BBG$^+$. All the values are uniform random in their respective domains.

Next, we know that an honest verifier samples the vector $\boldsymbol{\psi}$ uniformly from $\mathbb{F}$, thus the query vector $\mathbf{q}_{L+1}$ is identically distributed to the real $(L+1)$-st query.

Finally, when $\mathbf{x}$ is valid, the final linear query outputs 0, thus the final output of the simulator is also perfect.

Hence, the simulator's entire output is identically distributed to the real interaction and Π' satisfies the SHVZK property. □

E UNRESTRICTED IPFE

The IPFE scheme must be unrestricted because the FL-PCP protocol's Query algorithm needs to sample random query vectors to verify the input. This scheme is proposed in [17] based on class groups, which we describe below. CLT operates within a DDH-hard group containing a DL-easy subgroup, meaning that there exists a polynomial time algorithm capable of solving discrete logarithm problem in this subgroup. Let Gen be the group generator. We specify that on inputs two parameters λ and μ, Gen outputs a tuple $(p, \tilde{s}, g, f, g_p, G, F, G^p)$ where, the set $(G, \cdot)$ is a cyclic group of order ps, s is an integer, $\tilde{s}$ is an upper bound of s, p is a μ-bit prime, and $\gcd(p, s) = 1$. The set $G^p = \{x^p, x \in G\}$ is the subgroup of order s of G, and F is subgroup of order p of G, so that $G = F \times G^p$. Finally f, g_p and $g = f \cdot g_p$ are generators of F, G^p and G, respectively. Since the discrete logarithm problem is easy in F, we let Solve be a deterministic polynomial time algorithm that solves the discrete logarithm problem in F.

Let $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_p^d$, we present the construction based on the HSM assumption in Figure 19, and demonstrate that it is additively homomorphic. In [17], CLT.KeyGen is designed as a stateful algorithm,

CLT.Setup$(1^\lambda, 1^\mu, 1^d)$:

1 $(p, \bar{s}, f, g_p, G, F, G^p) \leftarrow\$ \text{Gen}(1^\lambda, 1^\mu)$
2 $\mathbf{s} = (s_1, \ldots, s_d) \leftarrow\$ \mathcal{D}$
3 For $1 \le i \le d$:
4 $\quad h_i = g_p^{s_i}$
5 $\text{mpk} \leftarrow (p, \bar{s}, f, g_p, \{h_i\}_{i\in[d]})$
6 $\text{msk} \leftarrow \mathbf{s}$
7 Return (mpk, msk)

CLT.KeyGen(msk, $\mathbf{y}$):

8 $\mathbf{s} \leftarrow \text{msk}$
9 $\text{sk}_\mathbf{y} = \langle \mathbf{s}, \mathbf{y} \rangle$
10 Return $\text{sk}_\mathbf{y}$

CLT.Encrypt(mpk, $\mathbf{x}$):

11 $r \leftarrow\$ \mathcal{D}_p$
12 $\text{ct}_0 \leftarrow g_p^r$
13 For $1 \le i \le d$:
14 $\quad \text{ct}_i = f^{\mathbf{x}_i} \cdot h_i^r$
15 Return $(\text{ct}_0, \text{ct}_1, \ldots, \text{ct}_d)$

CLT.Eval(mpk, $\text{ct}_\mathbf{x}$, $\text{sk}_\mathbf{y}$):

16 $(\text{ct}_0, \text{ct}_1, \ldots, \text{ct}_d) \leftarrow \text{ct}_\mathbf{x}$
17 $\text{ct} = \left(\prod_{i=1}^d \text{ct}_i^{\mathbf{y}_i}\right) \cdot \text{ct}_0^{-\text{sk}_\mathbf{y}}$
18 Return Solve(ct)

CLT.Decrypt(msk, $\text{ct}_\mathbf{x}$):

19 $(\text{ct}_0, \text{ct}_1, \ldots, \text{ct}_d) \leftarrow \text{ct}_\mathbf{x}$
20 $\mathbf{s} \leftarrow \text{msk}$
21 For $1 \le i \le d$:
22 $\quad \mathbf{x}_i \leftarrow \text{Solve}(\text{ct}_i \cdot \text{ct}_0^{-\mathbf{s}_i})$
23 Return $(\mathbf{x}_1, \ldots, \mathbf{x}_d)$

CLT.Add$(\text{ct}_{\mathbf{x}_1}, \text{ct}_{\mathbf{x}_2})$:

24 $(\text{ct}_{1,0}, \text{ct}_{1,1}, \ldots, \text{ct}_{1,d}) \leftarrow \text{ct}_{\mathbf{x}_1}$
25 $(\text{ct}_{2,0}, \text{ct}_{2,1}, \ldots, \text{ct}_{2,d}) \leftarrow \text{ct}_{\mathbf{x}_2}$
26 For $0 \le i \le d$:
27 $\quad \text{ct}_i = \text{ct}_{1,i} \cdot \text{ct}_{2,i}$
28 Return $(\text{ct}_0, \text{ct}_1, \ldots, \text{ct}_d)$

Figure 19: Unrestricted IPFE construction instantiated by HSM in [17]

ensuring that keys cannot be queried for vectors that are linearly dependent over $\mathbb{Z}_p^d$ but independent over $\mathbb{Z}^d$. In our usage scenario, the analyst generates all inputs to CLT.KeyGen at once and can independently verify them before invoking CLT.KeyGen. Hence the stateful KeyGen is not necessary. Furthermore, we demonstrate that the scheme possesses the desired additive homomorphism property, as described in Section 3.3, enabling the server to aggregate ciphertexts and output their sum.

Additive Homomorphism of CLT. For all $\lambda, \mu, d \in \mathbb{N}$, $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{Z}_p^d$, and $(\text{mpk}, \text{msk}) \leftarrow\$ \text{CLT.Setup}(1^\lambda, 1^\mu, 1^d)$, we let $\text{ct}_{\mathbf{x}_1} \leftarrow \text{CLT.Encrypt}(\text{mpk}, \mathbf{x}_1)$, $\text{ct}_{\mathbf{x}_2} \leftarrow \text{CLT.Encrypt}(\text{mpk}, \mathbf{x}_2)$, and $\text{ct} \leftarrow \text{CLT.Add}(\text{ct}_{\mathbf{x}_1}, \text{ct}_{\mathbf{x}_2})$. We then have that $\text{ct}_0 = g_p^{r_1+r_2}$, $\text{ct}_i = f^{x_{1,i}+x_{2,i}} h_i^{r_1+r_2}$ for $1 \le i \le d$. Since r_1 and r_2 are randomly sampled, $\text{CLT.Add}(\text{ct}_{\mathbf{x}_1}, \text{ct}_{\mathbf{x}_2})$ is computationally indistinguishable from $\text{CLT.Encrypt}(\text{mpk}, \mathbf{x}_1 + \mathbf{x}_2)$.

Instantiation details. As shown in [17], in practice, the distributions $\mathcal{D}$ and $\mathcal{D}_p$ can be implemented from the output of Gen more efficiently. Let $\mathcal{D}_\sigma$ represents the discrete Gaussian distribution over $\mathbb{Z}$ with parameter σ centered at 0. We choose $\mathcal{D} = \mathcal{D}_{\tilde{s}\cdot p\cdot\sqrt{\lambda}}$, and $\mathcal{D}_p = \mathcal{D}_{\tilde{s}\cdot\sqrt{\lambda}}$. The CLT scheme is instantiated using class groups of imaginary quadratic fields. We refer to [16, 17] for a full description of the implementation. We set both λ the security parameter and μ the size of prime p to 128. This results in the hidden order group G_p having 924-bit elements and those in G having 2084-bit elements.