Verifiable Computation with Reduced Informational Costs and Computational Costs

Gang Xu, George T. Amariucai, and Yong Guan

Iowa State University, Ames, Iowa, USA
{gxu,gamari,guan}@iastate.edu

Abstract. Outsourcing computation is a fundamental principle of the new cloud computing paradigm. Among its various aspects, the correctness of the computation result remains paramount. This motivates the birth of verifiable computation, which aims at efficiently checking the result for general-purpose computation. The common goal of recently sprouted verifiable computation protocols is to reduce the costs associated with verification at both prover and verifier. Unfortunately, the high computation and communication costs of verification still keep general verifiable computation away from practicality. Besides the computational costs, we observe that another type of verification cost has been generally ignored until now –the informational costs, namely, the information required for the verification. In particular, in the context of the third-party verification, this cost implies the information leakage of sensitive information regarding the computational task and its results. In this paper, we introduce the new verifiable-computation protocol RIVER, which reduces the computational costs of the verifier and of the prover, comparing to the most recent alternative protocols, and (for the first time in the context of verifiable computation) addresses and decreases informational costs.

Keywords: verifiable computing, QAPs, PCPs, clouds, informational costs, privacy.

1 Introduction

In the era of cloud computing, outsourcing computation becomes a new trend. Instead of purchasing, maintaining, and updating expensive computing assets for local computation, users can outsource computation and other IT services from relatively weaker devices to a professional service provider (like the Cloud). But while enjoying the appealing benefits of outsourcing computation – such as reduced financial, personnel and computational burdens – a critical security issue arises: Cloud servers may be error-prone or otherwise not trustworthy. This motivates a great body of research on verifiable computation. Many works focus on specific computational tasks, exploiting their special structure to provide efficient verification [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]. However, when it comes to verifying the results of general computation, most efforts seem to be

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part I, LNCS 8712, pp. 292–309, 2014.

concentrated around classic proof systems in theoretical computer science. In this framework, the server plays the role of a *prover*, trying to convince the client, who plays the role of a *verifier*, that the result returned from the server is correct. The verification procedure usually consists of the verifier issuing randomly-chosen queries, and the prover providing answers.

Interactive proof (IP) systems [13] have recently been introduced to the context of delegation of computation [14] [15] [16]. They are efficient in terms of asymptotic complexity. As a new encoding scheme for arithmetic circuits, Quadratic Arithmetic Programs (QAPs, [17]) have shown great potential for the design of verifiable computation protocols [17] [18]. The third notable direction in verifiable computation is based on probabilistically checkable proof (PCP) systems [19] [20]. *Argument systems* [21], one variant of the PCP model, seem to be appropriate for practical verifiable computation. Argument systems hold a more practical assumption that, in addition to the verifier being polynomial-time probabilistic, the prover is also computationally bounded. Combining PCP with a homomorphic cryptographic machinery, recent works on argument systems, such as IKO [22], Pepper [23], Ginger [24], XAG [25], and Zaatar [26], have brought PCP-based approaches closer to practicality – they sharply reduced the number of queries, and the overhead required to prevent the falsification of the proof string.

1.1 Motivation

The state-of-the-art Zaatar [26] showed the connection between Linear PCP and QAP. However, unlike other QAP-based designs [17] [18], Zaatar relies only on standard cryptographic assumptions. It applies QAP into the framework of PCP and generates a novel verifiable computation scheme. Moreover, as an appealing verifiable computation scheme, Zaatar makes the prover more efficient than any other PCP-based designs. However, Zaatar does not bring major improvements to the verifier's computational cost. As in the recent PCP-based works [22] [27] [23] [24], once the prover has committed to the proof, the most computationally-intensive part for the verifier in Zaatar is the generation of queries. The high costs of the verifier are hence lowered by reusing some of the queries for multiple instances of the *same* problem – or *batching*. For computational tasks that can tolerate large batch sizes, the costs of verification in Zaatar can be driven down by amortizing. However, for tasks that require low investment on the verification and tolerate only small batch sizes, a new, more efficient protocol is needed.

Besides the computation and communication costs that have been concerned in existing research of verifiable computation, another type of verification cost has been generally ignored until now. We call this the *informational cost* – the cost associated with information required for verification on both sides. The verifier's information required for verification usually consists of verification keys and the full knowledge of the computation task. The prover's informational costs usually consists of the proof vector.

The informational cost generally has a strong impact on the adoption of the verification algorithms. On one hand, the size of the required information

directly influences the memory cost for verification, and the speed of verification. For the memory cost, verifiers in existing research keep the required information in a large memory and frequently access it. For the speed, the length of the proof vector determines the cost of generating, and responding to, the queries while verifying. On the other hand, the informational cost implies the privacy/confidentiality issues. One obvious risk is that, storing this information itself introduces potential leakage of sensitive information about the computational task and its results. A more serious risk occurs in the context of the third party verification. (For example, disputes between the server and the client can be solved by an arbitrator who plays the role of the third-party verifier. Similar verifications may be required by government agencies, nonprofit organizations, and consumer organization, for the purpose of quality evaluation, project management, etc.) In such scenarios, information cost implies the computation task and its results being delivered to the third party. However, once delivered, the information is out of the control of the client and the client can never call them back. This security issue, in turn, may limit the outsourcing of verification [25].

As far as informational costs are concerned, all recent PCP-based works [22] [27] [23] [24], [26] require the verifier to have full knowledge of the computation circuit while performing the verification.

1.2 Our Contributions

In this paper, we introduce **RIVER**, a **R**educed-**i**nvestment **ver**ified computation protocol, whose improvement further enhances the practicality of argument systems in verifiable computation. Our contributions are summarized as follows:

- RIVER reduces the verifier's workload that needs to be amortized. Namely, the number of batched instances of the protocol, required for amortization, will largely decrease. Instead of batching over instances of the same circuit as in existing works, RIVER makes more parts of costs amortized over instances of *all different circuits of the same size*. We model the costs and compare the costs using a typical computation such as matrix multiplication, showing that RIVER is 28% better than state-of-the-art Zaatar at the verifier side.
- As a side effect, RIVER reduces the prover's non-amortized cost. As in the typical computation of matrix multiplication, RIVER achieves 55% better than Zaatar at the prover's non-amortized cost. Although RIVER introduces amortized cost to the prover side, this cost can be amortized over instances of *all different circuits of the same size*.
- RIVER reduces the informational cost of the verifier, by removing the requirement that the verifier has to access the circuit description during query generating. Thus, a third-party verifier can help generating the queries without knowing the computation task details.
- RIVER adopts one of our theoretical findings. We show that under certain assumptions, the Single-Commit-Multi-Decommit protocol provides the inherent linearity tests. Thus, a modified Single-Commit-Multi-Decommit protocol will make the linearity tests obsolete and reduce verification costs.

2 System Model

In the context of cloud computing, we propose a computation architecture involving two parties: the client $\mathcal{V}$, who is computationally weak, has computation tasks to be delegated to the cloud; the cloud server $\mathcal{P}$, who is computationally powerful, provides computing services to the client. The computation tasks are formalized into the *arithmetic circuit* – i.e., the computation task is performed over an arithmetic circuit. This is pretty natural, since arithmetic circuits can be easily mapped to real-world computation tasks[1]. Let Ψ be a $|\Psi|$-gate arithmetic circuit. The client $\mathcal{V}$ is providing the prover $\mathcal{P}$ with Ψ and input $X \in \mathbb{F}^n$, and expects $\mathcal{P}$ to return the correct output $Y \in \mathbb{F}^{n'}$. Then $\mathcal{P}$ tries to convince $\mathcal{V}$ that Y is correct. $\mathcal{P}$ will hold a proof Z, which is a *correct assignment* Z –the concatenation of the input X, output Y with all the intermediate results W inside the circuit, $(Z = X||Y||W)$ and has length $|Z| = m$, where W is the intermediate result vector $W \in \mathbb{F}^{m-n-n'}$ of the circuit Ψ.

3 Preliminaries

3.1 PCP and Commitments

Now we briefly review the line of verifiable computation based on argument systems. To make argument systems efficient, current implementations rely on *probabilistically checkable proofs* (PCPs). However, PCP algorithms assume that the proof is computed by the prover, and fixed before the interaction with the verifier begins. The same assumption cannot be made in the context of argument systems. To bridge the gap between arguments and PCPs, an additional protocol is required, in which the prover commits to the proof before starting the PCP protocol with the verifier. Consequently, an argument is generally formed by joining together two protocols: a *PCP* and a *commitment protocol*.

To avoid the need for convoluted short PCP proofs, as well as the uncertain security of practical hashing primitives, [22] takes a new approach to argument systems: maintain a large (exponential-size) proof, and base the commitment on (computationally) provably-secure encryption primitives – public-key primitives.

The protocols of [22] are restricted to linear PCPs ([28], Section 6). It is shown how SAT problems, formulated in the context of a boolean circuit, can be readily addressed by a simple linear PCP [22]. To form the argument system, [22] complemented the linear PCP with the notion of *commitment with linear decommitment*, which is instantiated with a simple public-key-based protocol. Once the proof is committed, $\mathcal{V}$ will check the proof in the linear PCP fashion.

In Ishai et al.'s original commitment design [22], one query is accompanied by an auxiliary query which is associated to a commitment. This requires many commitments, therefore increases the overhead. As in [22], where the PCP proof is represented by the vector d, the prover $\mathcal{P}$ commits to a certain proof, in a

[1] Existing compilers can turn high-level programs into arithmetic circuits [24], [26], [18]. For simplicity, we omit these techniques.

Table 1. The Single-Commit-Multi-Decommit Design [24]

$\mathcal{P}$**'s Input:** a vector $z \in \mathbb{F}^n$, a linear function $\pi : \mathbb{F}^{n^2+n} \mapsto \mathbb{F}$ where $\pi(\cdot) = \langle Z \| z \otimes z, \cdot \rangle$, n is the length of a correct assignment z.
$\mathcal{V}$**'s Input:** arity n, security parameter k of the encryption.
Commitment
Step 1: $\mathcal{V}$ generates the key pair: $(pk, sk) \leftarrow Gen(1^k)$.
$\mathcal{V}$ randomly generates a vector: $r = (r_1, r_2, \cdots, r_{n^2+n}) \in_R \mathbb{F}^{n^2+n}$.
$r_i \in \mathbb{F}, i = 1, 2, \cdots, n^2 + n$. $\mathcal{V}$ encrypts each entry of the vector r.
He sends $\mathtt{Enc}(pk, r_1), \cdots, \mathtt{Enc}(pk, r_{n^2+n})$ to $\mathcal{P}$.
Step 2: Using the homomorphism, $\mathcal{P}$ gets: $e = \mathtt{Enc}(pk, \langle r, z \rangle)$ $\mathcal{P}$ sends e to $\mathcal{V}$.
Step 3: $\mathcal{V}$ decrypts e. He gets $s = \langle r, z \rangle = Dec(sk, e)$.
Decommitment
Step 1: $\mathcal{V}$ picks μ secrets $\alpha_1, \cdots, \alpha_\mu \in \mathbb{F}$
$\mathcal{V}$ queries $\mathcal{P}$ with $q_1, \cdots, q_\mu$ and $t = r + \alpha_1 q_1 + \cdots + \alpha_\mu q_\mu$.
Step 2: $\mathcal{P}$ returns $(a_1, \cdots, a_\mu, b)$ where $a_i = \pi(q_i)$ for $i = 1, \cdots, \mu$ and $b = \pi(t)$
Step 3: $\mathcal{V}$ checks whether $b = s + \alpha_1 a_1 + \cdots \alpha_\mu a_\mu$ holds.
If so, $\mathcal{V}$ outputs $a_1, \cdots, a_\mu$. Otherwise, he rejects and output $\perp$.

manner that assures the verifier $\mathcal{V}$ that the proof he later queries is the original proof, and not some adaptively-modified false proof. Several recent works build upon the ideas developed in [22]. Of these, [23] introduces several contributions, including a *single-commit-multi-decommit* protocol. In [23], one auxiliary query is made, which is a random linear combination of all the PCP queries and the secret information that is associated to the commitment. In this design, one decommitment could guarantee many PCP queries are bound to the committed function. This sharply reduced the computational cost of generating the commitment information (although remaining cost is still very high.). The Single-Commit-Multi-Decommit design is demonstrated in Table 1. For more details regarding Table 1, refer to [22] and [23].

3.2 Quadratic Programs and Zaatar

Recently Gennaro, Gentry, Parno and Raykova introduced a new characterization of the NP complexity class – the *Quadratic Span Programs (QSPs)* (and *Quadratic Arithmetic Programs (QAPs)*) [17] [18]. They showed that NP can be defined as the set of languages with proofs that can be efficiently verified by QSPs (or QAPs). Similar to PCPs – another characterization of NP, which has already been widely used to obtain verifiable computation schemes – QSPs/QAPs are considered to be well-suited for verifiable computation and zero-knowledge schemes. One limitation of QSPs is that they inherently compute boolean circuits. But since arithmetic circuits are more natural and efficient in real-world computation tasks, we focus on QAPs, the counterpart of QSPs dealing with arithmetic circuit evaluation.

Definition 1 (Quadratic Arithmetic Programs)([17]) *A QAP Q over field $\mathbb{F}$ contains three sets of $m + 1$ polynomials: $\{A_i(t)\}$, $\{B_i(t)\}$, $\{C_i(t)\}$, for $i \in$*

$\{0, 1, \cdots, m\}$, *and a target polynomial* $D(t)$. *For function* $\Psi : \mathbb{F}^n \mapsto \mathbb{F}^{n'}$, *we say* Q *computes* Ψ *if the following holds:* $(z_1, z_2, \cdots, z_{n+n'}) \in \mathbb{F}^{n+n'}$ *is a valid assignment of* Ψ*'s inputs and outputs, if and only if there exist coefficients* $z_{n+n'+1}, \cdots, z_m$ *such that* $D(t)$ *divides* $P(t)$, *where* $P(t) = \left(\sum_{i=1}^{m} z_i \cdot A_i(t) + A_0(t)\right) \cdot \left(\sum_{i=1}^{m} z_i \cdot B_i(t) + B_0(t)\right) - \left(\sum_{i=1}^{m} z_i \cdot C_i(t) + C_0(t)\right)$. *In other words, there exists a polynomial* $H(t)$ *such that* $D(t) \cdot H(t) = P(t)$.

Given an arithmetic circuit, its QAP can be constructed by polynomial interpolation. Consider the set of circuit wires corresponding to the input and output of the circuit, and the outputs of all multiplication gates. Each one of these wires is assigned three interpolation polynomials in Lagrange form, encoding whether the wire is a left input, right input, or output of each multiplication gate. The resulting set of polynomials is a complete description of the original circuit.

The very recent work of [26] observes that QAPs can also be viewed as linear PCPs. By re-designing the PCP query generation and replacing the quadratic consistency checks and circuit correctness checks with the divisibility check of a QAP, they successfully fit QAPs into the framework of Ginger [24]. The result is the novel protocol *Zaatar*, which significantly reduces the prover's workload. The key observation of Zaatar is that the evaluation of the polynomial $P(t)$ in (1) at the point $t = \tau$ can be simply written as:

$$P(\tau) = (\langle Z, q\rangle + A_0(\tau)) \cdot (\langle Z, q'\rangle + B_0(\tau)) - (\langle Z, q''\rangle + C_0(\tau)), \tag{1}$$

where $Z = (z_1, \cdots, z_m)$, $q = (A_1(\tau), \cdots, A_m(\tau))$, $q' = (B_1(\tau), \cdots, B_m(\tau))$, $q'' = (C_1(\tau), C_2(\tau), \cdots, C_m(\tau))$. Thus, $P(\tau)$ can be evaluated through three standard PCP queries to the oracle $\pi_Z(\cdot) = \langle Z, \cdot\rangle$. If we represent the polynomials $H(t)$ explicitly: $H(t) = h_{|C_Z|}t^{|C_Z|} + \cdots + h_1 t + h_0$ (where C_Z is the set of constraints in Zaatar), similar observations on $H(\tau)$ can be made: $H(\tau) = \langle K_H, q_H\rangle$, where $K_H = (h_0, h_1, \cdots, h_{|C_Z|})$ and $q_H = (1, \tau, \tau^2, \cdots, \tau^{|C_Z|})$. Thus, $H(\tau)$ can also be evaluated through one PCP query to the oracle $\pi_H(\cdot) = \langle K_H, \cdot\rangle$.

If Z consists of the input X with width $|X| = n$, output Y with width $|Y| = n'$ and intermediate results W with width $|W| = m - (n + n')$, then to guarantee that Y is the correct output when the input is X, the verifier needs to compute a part of $\langle Z, q\rangle$, and also a part of $\langle Z, q'\rangle$ and $\langle Z, q''\rangle$, by himself. Consequently, $\mathcal{V}$ only queries the linear function oracle $\pi_W(\cdot) = \langle W, \cdot\rangle$, instead of $\pi_Z(\cdot)$.

4 A Technique: A Commitment Providing Inherent Linearity Tests

In the line of linear-PCP fashion verifiable computation designs, once the prover is committed to a proof, the verifier has to perform laborious linearity tests to ensure the proof is linear. In fact, the number of queries required to perform linearity tests dominate the number of overall queries of the protocol. Thus, the cost caused by linearity test is still one of the bottlenecks of current protocols.

Up to now, in the context of Single-Commit-Multi-Decommit protocol (refer to Section 3), whether the linearity tests are necessary was still an open question. In this section, we propose our theoretical result, showing that under an assumption, the Single-Commit-Multi-Decommit protocol will provides inherent linearity tests. Thus, if linear PCP is combined with this commitment protocol, the linearity tests are obsolete. We will adopt the following theoretical results in our protocol design and thus achieve cost savings.

Theorem 1. *The Single-Commit-Multi-Decommit protocol ensures that, if the secret commit information is generated by the prover using an affine function (analytically defined), (or equivalent to the cases that is generated by the verifier himself), for all query tuples, unless the sender (or prover) replies to all queries with the same* linear function *and he knows the analytic description of this linear function, the prover will not pass the decommitment test except with probability* $\frac{1}{|\mathbb{F}|} + \epsilon_S$*, where the probability is over the randomness of the prover and the verifier in the decommitment phase.*

Proof. Let $\pi()$ denote the proof in the PCP sense. The prover knows that, when the verifier generates the commitment information $\pi(r)$, he uses the linear function $\mathtt{F}_1(\cdot)$, such that $\pi(r) = \mathtt{F}_1(r)$. We claim that in this scenario, the prover has to answer all queries with the same linear function $\mathtt{F}_1$ – otherwise the probability that the prover passes the decommitment test is less than $\frac{1}{|\mathbb{F}|} + \epsilon_S$.

To prove this, we assume that there exists a PPT prover $\mathcal{P}^*$, and queries $q_1, q_2, \cdots, q_\mu$, such that once committed, with these queries, the probability that $\mathcal{P}^*$ answers the μ queries with a function $f(\cdot)$ – such that there exists at least one index k for which $f(q_k) \neq \mathtt{F}_1(q_k)$ – and passes the decommitment test, is more than $\frac{1}{|\mathbb{F}|} + \epsilon_S$, where the probability is over the randomness of the prover and the verifier in the decommitment phase.

We can now modify $\mathcal{P}^*$ and make it into an algorithm $\mathcal{P}^\dagger$ which can solve the problem stated in Lemma 1 with probability more than $\frac{1}{|\mathbb{F}|} + \epsilon_S$:

1. $\mathcal{P}^\dagger$ has inputs: $\mathtt{Enc}(\mathtt{r}), r + \alpha q_k, q_k$
2. $\mathcal{P}^\dagger$ uses $\mathtt{Enc}(\mathtt{r})$ as inputs and runs $\mathcal{P}^*$'s commitment phase.
3. $\mathcal{P}^*$ outputs with $\mathtt{Enc}(\mathtt{F}_1(r)$ which $\mathcal{P}^\dagger$ will neglect.
4. $\mathcal{P}^\dagger$ uses the set of queries $\{q_1, \ldots, q_\mu\}$. He produces a set of coefficients $\{\alpha_1, \cdots, \alpha_{k-1}, \alpha_{k+1}, \cdots, \alpha_{\mu-1}, \alpha_\mu\}$, and runs $\mathcal{P}^*$'s decommitment phase with following: $\{q_1, q_2, \ldots, q_{\mu-1}, q_\mu, (r + \alpha q_k) + \sum_{i=1}^{k-1} \alpha_i q_i + \sum_{i=k+1}^{\mu} \alpha_i q_i\}$..
5. $\mathcal{P}^*$ outputs $\{f(q_1), \ldots, f(q_\mu), \mathtt{F}_1(r) + \alpha f(q_k) + \sum_{i=1}^{k-1} \alpha_i f(q_i) + \sum_{i=k+1}^{\mu} \alpha_i f(q_i)\}$. This will pass the decommitment test with probability more than $\frac{1}{|\mathbb{F}|} + \epsilon_S$.
6. Having access to $\{\alpha_1, \cdots, \alpha_{k-1}, \alpha_{k+1}, \cdots, \alpha_{\mu-1}, \alpha_\mu\}$, and $f(q_1), \ldots, f(q_\mu)$, $\mathcal{P}^\dagger$ can now obtain an equation of the form $\mathtt{F}_1(r) + \alpha f(q_k) = b$ (where $b \in \mathbb{F}$ is easily calculated).

Recall that $\mathcal{P}^\dagger$ has knowledge of $r + \alpha q_k$, which yields a group of n linearly-independent linear equations in the form of $r + \alpha q_k = a$. Given that $\mathtt{F}_1(q_k) \neq$

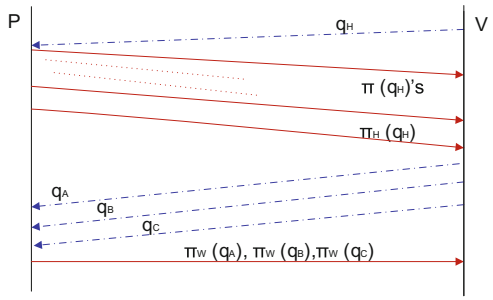

Fig. 1. PCP querying

$f(q_k)$, the equation $\mathtt{F}_1(r) + \alpha f(q_k) = b$ is linearly independent of the former n equations $r + \alpha q_k = a$. Thus, $\mathcal{A}$ can solve for α from these $n + 1$ linearly independent equations. This will contradict Lemma 1:

Lemma 1. *(from [22]) For any probabilistic polynomial time algorithm $\mathcal{A}$, any $q \in \mathbb{F}^n$, and any uniformly-randomly picked $r \in \mathbb{F}^n$ we have $Pr[\mathcal{A}(\mathtt{Enc}(\mathtt{r}), r + \alpha q, q) = \alpha] \leq \frac{1}{|\mathbb{F}|} + \epsilon_S$, where ϵ_S is from the semantic security.*

5 A Reduced-Investment Verifiable Computation Protocol: RIVER

In this section, we introduce RIVER(*reduced-investment verifiable computation protocol*), an improvement of Zaatar [26], aimed at reducing the amortized cost of the verifier – or equivalently, the number of instances required before amortization can be considered complete. We accomplishes this by deferring some of the verifier's amortizable computation to the prover. In doing so, two other benefits are achieved as side effects. First, the overall cost for the verifier is decreased when compared to Zaatar (and implicitly also to Ginger). This is despite deferring some of the amortized computation to the prover (the deferred part is almost negligible when compared to the construction of the proof). Second, RIVER enables the verifier to generate queries independently – that is, the query generating stage does not require full knowledge of the circuit. We detail RIVER as follows.

5.1 PCP Querying

Our main observation is that the PCP query generation in Zaatar is somewhat redundant. RIVER removes the redundancy by employing three rounds of PCP querying and one round of decision making. The logical procedure is demonstrated in Figure 1.

5.2 PCP Querying of RIVER

Let $l = |C_R|$. We represent the QAP polynomials $A_i(t)$, $B_i(t)$, $C_i(t)$, with $i = 0, 1, \cdots, m$ explicitly as:

$$A_i(t) = a_l^{(i)} t^l + a_{l-1}^{(i)} t^{l-1} + \cdots + a_1^{(i)} t + a_0^{(i)} \tag{2}$$

$$B_i(t) = b_l^{(i)} t^l + b_{l-1}^{(i)} t^{l-1} + \cdots + b_1^{(i)} t + b_0^{(i)} \tag{3}$$

$$C_i(t) = c_l^{(i)} t^l + c_{l-1}^{(i)} t^{l-1} + \cdots + c_1^{(i)} t + c_0^{(i)} \tag{4}$$

Evaluation of any one of these polynomials at the point $t = \tau$ can be expressed as a linear function: $A_i(\tau) = \pi_A^{(i)}(q_H) = \langle K_A^{(i)}, q_H \rangle$, $B_i(\tau) = \pi_B^{(i)}(q_H) = \langle K_B^{(i)}, q_H \rangle$, $C_i(\tau) = \pi_C^{(i)}(q_H) = \langle K_C^{(i)}, q_H \rangle$, where $q_H = (1, \tau, \tau^2, \cdots, \tau^l)$ and

$$K_A^{(i)} = (a_l^{(i)}, a_{l-1}^{(i)}, \cdots, a_1^{(i)}, a_0^{(i)}) \tag{5}$$

$$K_B^{(i)} = (b_l^{(i)}, b_{l-1}^{(i)}, \cdots, b_1^{(i)}, b_0^{(i)}) \tag{6}$$

$$K_C^{(i)} = (c_l^{(i)}, c_{l-1}^{(i)}, \cdots, c_1^{(i)}, c_0^{(i)}). \tag{7}$$

We can simply express the PCP queries of Zaatar as

$$q_A = (\pi_A^m(q_H), \pi_A^{m-1}(q_H), \ldots, \pi_A^{n+n'+1}(q_H)) \tag{8}$$

$$q_B = (\pi_B^m(q_H), \pi_B^{m-1}(q_H), \ldots, \pi_B^{n+n'+1}(q_H)) \tag{9}$$

$$q_C = (\pi_C^m(q_H), \pi_C^{m-1}(q_H), \ldots, \pi_C^{n+n'+1}(q_H)). \tag{10}$$

In RIVER, the verifier constructs q_A, q_B, q_C by querying linear functions π_A^i, π_B^i, π_C^i $(i = 0, \cdots, m)$ by a single query q_H.

Similarly, we can express $H(t)$ and $D(t)$ as:

$$H(t) = h_l t^l + h_{l-1} t^{l-1} + \cdots + h_1 t + h_0 \tag{11}$$

$$D(t) = d_l t^l + d_{l-1} t^{l-1} + \cdots + d_1 t + d_0, \tag{12}$$

and define: $\pi_H(\cdot) = \langle K_H, \cdot \rangle$, where $K_H = (h_0, h_1, \cdots, h_l)$, and $\pi_D(\cdot) = \langle K_D, \cdot \rangle$, where $K_D = (d_0, d_1, \cdots, d_l)$. Zaatar points out that the evaluation of $H(\tau)$ can be viewed as querying an oracle $\pi_H(\cdot)$ with q_H. Here, we argue that the same holds for the evaluation of $D(\tau)$ – querying the oracle $\pi_D(\cdot)$ with q_H. The idea is detailed in Table 2. Note that by comparison, Zaatar requires the queries q_A, q_B, q_C, along with $D(\tau)$ to be entirely computed by $\mathcal{V}$. It should be mentioned that computing these queries by querying another set of proofs requires additional commitments and testing. However, the procedure can be simplified by removing all linearity tests for these $3m+4 = 3(m+1)+1$ proofs. The reason this works is that, according to Theorem 1, our decommitment already provides an inherent linearity test. In the second round of our design, $\mathcal{V}$ issues queries q_H as in Table 3. In the third round, $\mathcal{V}$ issues queries q_A, q_B, q_C, q_D as in Table 4. After $\mathcal{V}$ collects all responses, he makes the decision as in Table 5.

Table 2. The First Round of Our QAP-based Linear PCP

For every π in the set of $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, $(i = 0, 1, \cdots m)$ π_D, perform the following:

- Divisibility queries generation. $\mathcal{V}$ randomly selects $\tau \in_R \mathbb{F}$. $\mathcal{V}$ takes $q_H \leftarrow (1, \tau, \tau^2, \cdots, \tau^l)$.
- Querying. $\mathcal{V}$ sends out q_H and gets back $\pi(q_H)$.

If all these proofs pass all linearity tests, $\mathcal{V}$ will have: $\pi_D(q_H)$ and

- $\pi_A^{(m)}(q_H), \pi_A^{(m-1)}(q_H), \cdots, \pi_A^{(0)}(q_H)$,
- $\pi_B^{(m)}(q_H), \pi_B^{(m-1)}(q_H), \cdots, \pi_B^{(0)}(q_H)$,
- $\pi_C^{(m)}(q_H), \pi_C^{(m-1)}(q_H), \cdots, \pi_C^{(0)}(q_H)$,

Table 3. The Second Round of Our QAP-based Linear PCP

$\mathcal{V}$ queries π_H.

- Linearity queries generation. $\mathcal{V}$ selects $q_2, q_3 \in_R \mathbb{F}^l$. Take $q_4 \leftarrow q_3 + q_2$. Perform ρ_{lin} iterations in total.
- QAP queries generation. $\mathcal{V}$ takes $q_H \leftarrow (1, \tau, \tau^2, \cdots, \tau^l)$ and $q_1 \leftarrow (q_H + q_2)$.
- Querying π_H. $\mathcal{V}$ sends out $q_1, q_2, \cdots, q_{1+3\rho}$ and gets back $\pi_H(q_1), \pi_H(q_2), \cdots, \pi_H(q_{1+3\rho})$.
- Linearity tests. Check whether following holds: $\pi_H(q_4) = \pi_H(q_3) + \pi_H(q_2)$ and likewise for all other $\rho - 1$ iterations. If not, reject.

At the end of this phase, if π_H passes all linearity tests, $\mathcal{V}$ will have: $\pi_H(q_H)$.

Table 4. The Second Round of Our QAP-based Linear PCP

$\mathcal{V}$ queries π_W. Remember $\pi_W(\cdot) = \langle W, \cdot \rangle$, where $W = (z_m, z_{m-1}, \cdots, z_{N+1})$

- Linearity queries generation. $\mathcal{V}$ select $q_4, q_5 \in_R \mathbb{F}^{m-N}$. Take $q_6 \leftarrow q_4 + q_5$. Perform ρ_{lin} iterations in total.
- QAP queries generation. $\mathcal{V}$ takes:
  - $q_A \leftarrow (\pi_A^{(m)}(q_H), \pi_A^{(m-1)}(q_H), \cdots, \pi_A^{(n+n'+1)}(q_H))$, and $q_1 \leftarrow (q_A + q_4)$.
  - $q_B \leftarrow (\pi_B^{(m)}(q_H), \pi_B^{(m-1)}(q_H), \cdots, \pi_B^{(n+n'+1)}(q_H))$, and $q_2 \leftarrow (q_B + q_4)$.
  - $q_C \leftarrow (\pi_C^{(m)}(q_H), \pi_C^{(m-1)}(q_H), \cdots, \pi_C^{(n+n'+1)}(q_H))$, and $q_3 \leftarrow (q_C + q_4)$.
- Querying π_W. $\mathcal{V}$ sends out $q_1, q_2, \cdots, q_{3+3\rho}$ and gets back $\pi_W(q_1), \pi_W(q_2), \cdots, \pi_W(q_{3+3\rho})$.
- Linearity tests. Check whether following holds: $\pi_W(q_6) = \pi(q_4) + \pi(q_5)$ and likewise for all other $\rho - 1$ iterations. If not, reject. Otherwise, accept and output $\pi_W(q_A) \leftarrow \pi_W(q_1) - \pi_W(q_4)$, $\pi_W(q_B) \leftarrow \pi_W(q_2) - \pi_W(q_4)$, $\pi_W(q_C) \leftarrow \pi_W(q_3) - \pi_W(q_4)$.

Table 5. The Decision Making Stage of Our QAP-based Linear PCP

Decision Making: (Note: $(z_1, z_2, \cdot, z_{n+n'}) = X\|\|Y$.)
– $\mathcal{V}$ computes: • $p_A \leftarrow \sum_{i=1}^{(n+n')} z_i \cdot \pi_A^{(i)}(q_H) + \pi_A^{(0)}(q_H)$ • $p_B \leftarrow \sum_{i=1}^{(n+n')} z_i \cdot \pi_B^{(i)}(q_H) + \pi_B^{(0)}(q_H)$ • $p_C \leftarrow \sum_{i=1}^{(n+n')} z_i \cdot \pi_C^{(i)}(q_H) + \pi_C^{(0)}(q_H)$ – Divisibility Test. $\mathcal{V}$ checks whether the following holds: $\pi_D(q_H) \cdot \pi_H(q_H) = (\pi_Z(q_A) + p_A) \cdot (\pi_Z(q_B) + p_B) - (\pi_Z(q_C) + p_C)$.

5.3 Commit, Decommit and Consistency Verification of RIVER

To ensure the security of the protocol, $\mathcal{P}$ commits to all the linear functions mentioned above. Similarly to Zaatar, our design inherits the single-commit-multiple-decommit protocol from Ginger. For π_H and π_W, $\mathcal{V}$ and $\mathcal{P}$ run the IKO-style single-commit-multi-decommit protocol to generate the commitment. This part is omitted for simplicity.

For $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, $(i = 0, 1, \cdots, m)$ and π_D, the case is a bit more complex. We note that in addition to the commitments and decommitments, $\mathcal{V}$ has to also verify the consistency of the polynomials' coefficient vectors corresponding to $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, for $i = 1, \ldots, m$, and π_D. Namely, $\mathcal{V}$ needs to make sure that $\mathcal{P}$ eventually uses $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, for $i = 1, \ldots, m$, and π_D to answer $\mathcal{V}$'s queries.

To accomplish this, we use the technique in Section 4 and come up with the commitment/decommitment protocol as follows: Before sending $\mathcal{P}$ his computation task, $\mathcal{V}$ secretly generates a random number r and computes by himself the values $A_i(r)$, $B_i(r)$, $C_i(r)$ $(i = 0, 1, \cdots m)$ and $D(r)$, each of which represents, respectively, the commitment for $\pi_A^{(i)}()$, $\pi_B^{(i)}()$, $\pi_C^{(i)}()$, for $i = 0, 1, \ldots, m$, and $\pi_D()$. The algorithm to compute these values is demonstrated in Section 6.1. These values are stored for future decommitment. This setup computation is done only once for different values of τ. In comparison with Zaatar, where the setup requires the verifier to evaluate the queries associated with different values of τ, a single r suffices for all τ's in RIVER, since the verifier outsources extra computation to the prover. As in Table 6, our commitment design guarantees the consistency of the polynomials' coefficient vectors with the linear functions to which $\mathcal{P}$ commits.

Theorem 2. *Let π be any of the linear functions $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$ and π_D. By performing our protocol, the commitment to π is guaranteed to be bound to a linear function $\tilde{\pi}$, and the probability that $\pi \neq \tilde{\pi}$ is at most $1/|\mathbb{F}|$. The probability is over all the randomness of the prover.*

Proof. Given that our protocol performs the single-commit-multi-decommit protocol when querying π, the response to the query is guaranteed to be bound to a linear function $\tilde{\pi}$. This feature is provided by the underlying single-commit-multi-decommit protocol. If $\pi \neq \tilde{\pi}$ but $\tilde{\pi}$ still passes the decommitment, $\tilde{\pi}(r) = \pi(r)$

Table 6. Decommit Design for $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, $(i = 0, 1, \cdots, m)$ and π_D

$\mathcal{P}$**'s Input:** linear functions π_D, $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, for $i = 1, \ldots, m$.
$\mathcal{V}$**'s Input:** $A_i(r)$, $B_i(r)$, $C_i(r)$, $i = 0, \cdots, m$ and $D(r)$, $t = (1, r, r^2, \cdots, r^l)$. $q_1, \cdots, q_\mu$
Commitment
The verifier generates the commitment information as in Section 6.1.
Decommitment
Step 1: $\mathcal{V}$ picks μ secrets $\alpha_1, \cdots, \alpha_\mu \in \mathbb{F}$ $\mathcal{V}$ queries $\mathcal{P}$ with $q_1, \cdots, q_\mu$ and $T = t + \alpha_1 q_1 + \cdots + \alpha_\mu q_\mu$.
Step 2: $\mathcal{P}$ returns $(\pi_A^{(i)}(q_1), \cdots, \pi_A^{(i)}(q_\mu), \pi_A^{(i)}(T))$, $(\pi_B^{(i)}(q_1), \cdots, \pi_B^{(i)}(q_\mu), \pi_B^{(i)}(T))$, $(\pi_C^{(i)}(q_1), \cdots, \pi_C^{(i)}(q_\mu), \pi_C^{(i)}(T))$, where $i = 0, \cdots, m$ and $(\pi_D(q_1), \cdots, \pi_D(q_\mu), \pi_D(T))$.
Step 3: $\mathcal{V}$ checks whether $\pi_A^{(i)}(T) = A_i(r) + \sum_{j=1}^{\mu} \alpha_j \pi_A^{(i)}(q_j)$ and whether $\pi_B^{(i)}(T) = B_i(r) + \sum_{j=1}^{\mu} \alpha_j \pi_B^{(i)}(q_j)$ and whether $\pi_C^{(i)}(T) = C_i(r) + \sum_{j=1}^{\mu} \alpha_j \pi_C^{(i)}(q_j)$, $i = 0, \cdots, m$ and $\pi_D(T) = D(r) + \sum_{j=1}^{\mu} \alpha_j \pi_D(q_j)$ hold. If so, $\mathcal{V}$ accepts. Otherwise, he rejects and output $\perp$.

must hold. For all possible choices of $\tilde{\pi}$, only $1/|\mathbb{F}|$ of them can satisfy this equation. However, r is unknown by the prover. Thus, the probability that a dishonest prover chooses a $\tilde{\pi} \neq \pi$ that passes the decommitment is at most $1/|\mathbb{F}|$.

6 Performance Analysis

For the informational cost, it is straightforward to see that, once committed, all the queries in the verification are not depending on the circuit description. Namely, during the query generating of the verification stage, the verifier does not need to access the circuit information any more. Our design separates the verification workload that involves only non-sensitive information from the verification workload that involves sensitive information (e.g. the circuit information). In the scenarios with a third-party verifier, the verifier can undertake the workload involving only non-sensitive information (e.g. query generating) without knowing the secrecy of the computation task.

In the following, we derive the computational cost of our RIVER design and compare it with previous work. In the process, we show that, similarly to Ginger and Zaatar, our protocol batches many instances for one *same* circuit to reduce the cost per instance. But RIVER can amortize more parts of amortized cost over *all different circuits of the same size.*

6.1 The Verifier

This section performs an analysis of the verifier's cost. A comparison with the verifier's costs in two other the state-of-the-art designs is given in Table 7.

Setup. The cost that RIVER incurs upon the commitment is $(|W_R| + |C_R|) \cdot (e + c)/\beta$. This is because RIVER needs two commitment query constructions.

Table 7. Comparison of Cost for Verifier in Each Instance

	Ginger	Zaatar	RIVER
Setup: Commit	$\|W_G\| \cdot e/(\beta \cdot \gamma)$	$(\|W_Z\|+\|C_Z\|)\cdot e/(\beta\cdot\gamma)$	$(\|W_R\| + \|C_R\|) \cdot e/(\beta \cdot \gamma) + (f_{div} + 5f) \cdot \|C_R\|/(\beta \cdot \gamma)$
Linearity Query Generation	$\rho\cdot\rho_{lin}\cdot 2\cdot(\|C_G\|+\|C_G\|^2)\cdot c/(\beta \cdot \gamma)$	$\rho \cdot \rho_{lin} \cdot 2 \cdot (\|W_Z\| + \|C_Z\|) \cdot c/(\beta \cdot \gamma)$	$\rho \cdot \rho_{lin} \cdot 2 \cdot (\|W_R\| + \|C_R\|) \cdot c/(\beta \cdot \gamma)$
Other PCP Query Generation	$\rho \cdot (c \cdot \|C_G\| + f \cdot K)/\beta$	$\rho \cdot [c + (f_{div} + 5f) \cdot \|C_Z\| + f \cdot K + 3f \cdot K_2]/\beta$	$\rho \cdot \|C_R\| \cdot f/(\beta \cdot \gamma)$
Decommitment Query Generation	$\rho \cdot L \cdot f/\beta$	$\rho \cdot (\rho_{lin} \cdot 3 \cdot (\|W_Z\| + \|C_Z\|) + (3\|W_Z\| + \|C_Z\|)) \cdot f/\beta$	$\rho\cdot(\rho_{lin}\cdot 3\cdot(\|W_R\|+\|C_R\|) + (3\|W_R\| + \|C_R\|)) \cdot f/\beta$
Decommitment Test	$d + \rho \cdot L \cdot f$	$d + \rho \cdot (\rho_{lin} \cdot 6 + 4) \cdot f$	$2d+\rho\cdot(\rho_{lin}\cdot 6+4)\cdot f+\rho\cdot(3m+4)\cdot f/\beta$
Decision Making	$\rho \cdot (\|X\| + \|Y\|) \cdot f$	$\rho \cdot (3\|X\| + 3\|Y\|) \cdot f$	$\rho \cdot (2\|X\| + \|Y\|) \cdot f$
Total non-amortized cost	$d+\rho\cdot(L+\|X\|+\|Y\|)\cdot f$	$2d+\rho\cdot f\cdot(3\|X\|+3\|Y\|+\rho_{lin}\cdot 6+4)$	$2d+\rho\cdot f\cdot(2\|X\|+\|Y\|+\rho_{lin}\cdot 6+4)$
Total amortized cost	$\|W_G\|\cdot e/(\beta\cdot\gamma)+\rho\cdot\rho_{lin}\cdot 2\cdot(\|C_G\|+\|C_G\|^2)\cdot c/(\beta\cdot\gamma)+\rho\cdot c\cdot\|C_G\|/\beta+\rho\cdot(L+K)\cdot f/\beta$	$(\|W_Z\|+\|C_Z\|)\cdot e/(\beta\cdot\gamma)+\rho\cdot\rho_{lin}\cdot 2\cdot(\|W_Z\|+\|C_Z\|)\cdot c/(\beta\cdot\gamma)+\rho\cdot[c+(f_{div})\cdot\|C_Z\|]/\beta+(\rho_{lin}\cdot 3\cdot(\|W_Z\|+\|C_Z\|)+(3\|W_Z\|+6\|C_Z\|+K+3K_2))\cdot\rho\cdot f/\beta$	$(\|W_R\| + \|C_R\|) \cdot e/(\beta \cdot \gamma) + \rho \cdot \rho_{lin} \cdot 2 \cdot (\|W_R\| + \|C_R\|) \cdot c/(\beta \cdot \gamma) + (\rho_{lin} \cdot 3 \cdot (\|W_R\| + \|C_R\|) + (3\|W_R\| + \|C_R\|) + 3m + 4) \cdot \rho \cdot f/\beta + ((f_{div}+5f)\cdot\|C_R\| + \rho \cdot \|C_R\| \cdot f)/(\beta \cdot \gamma)$

C_G: set of constraints in Ginger
$|W_G|$: number of variables in the constraints (excluding inputs and outputs) in Ginger
C_Z: set of constraints in Zaatar
$|W_Z|$: number of variables in the constraints (excluding inputs and outputs) in Zaatar
C_R: set of constraints in our design
$|W_R|$: number of variables in the constraints (excluding inputs and outputs) in our design
$|X|$: number of input
$|Y|$: number of output
g: cost of addition over $\mathbb{F}$
L: number of PCP queries in Ginger
β: number of batching
γ: number of circuits of the same size.
ρ: number of iteration of verification for one instance
ρ_{lin}: number of iterations of linearity tests in one iteration of verification.
f_{div}: cost of division over $\mathbb{F}$
f: cost of multiplication over $\mathbb{F}$
c: cost of pseudorandomly generating an element in $\mathbb{F}$
d: cost of decryption over $\mathbb{F}$
e: cost of encryption over $\mathbb{F}$
K: number of additive terms in the constraints of Ginger
K_2: number of distinct additive degree-2 terms in the constraints of Ginger

One is for π_H, and incurs a cost of $|C_R| \cdot (e+c)/\beta$, while the other is for π_W, and incurs a cost of $|W_R| \cdot (e+c)/\beta$. This total cost is the same as that of Zaatar.

It is apparent that RIVER introduces additional workload to the setup stage. $\mathcal{V}$ has to evaluate $A_i(r)$, $B_i(r)$, $C_i(r)$ and $D(r)$. However, we have discovered that a large part of the computation cost is independent of the underlying circuits.

Rather, the computation only depends on the size of the circuit. This implies that this part of the computation can be amortized over many different circuits, which only share the same size, rather than over many different instances of the same circuit. To see this, first notice that the target polynomial $D(t) = \prod_{k=1}^{|C_R|}(t - \sigma_k)$ does not depend on the circuit details, but rather $D(t)$ is determined by the circuit size. Hence, we can compute $D(r)$ once for all circuits of the same size, where r is the secret as in Section 5. If given in the form of generalized Newton's interpolation formula ([29], 4.6.4), $D(r)$ can be evaluated in time $|C_R| \cdot f$. Second, we express $A_i(t)$, $B_i(t)$, $C_i(t)$ in the form of Lagrange Polynomial interpolation: $A_i(t) = \sum_{j=1}^{|C_R|} a_{ij} \cdot l_j(t)$, $B_i(t) = \sum_{j=1}^{|C_R|} b_{ij} \cdot l_j(t)$, $C_i(t) = \sum_{j=1}^{|C_R|} c_{ij} \cdot l_j(t)$, where $l_j(t) = \prod_{1 \le k \le |C_R|, k \ne j} \frac{(t-\sigma_k)}{(\sigma_j - \sigma_k)}$ are Lagrange basis polynomials. We can represent these Lagrange basis polynomials as follows: $l_j(t) = \frac{D(t)}{(t-\sigma_j) \cdot \frac{1}{v_j}}$, where $v_j = 1/\prod_{0 \le k \le |C|, k \ne j}(\sigma_j - \sigma_k)$. If we choose these σ_k $(k = 1, \cdots, |C_R|)$ to follow an arithmetic progression [26], $l_j(r)$ $(j = 1, \cdots, |C_R|)$ can be evaluated in total time of $(f_{div} + 4f)|C_R|$. (Computing $1/v_{j+1}$ from $1/v_j$ requires only two operations and computing $1/v_0$ uses $|C_R|$ multiplication. Recall that $D(r)$ is computed already. Finally, to get each $l_j(r)$, a multiplication and one division are needed.) Given that both the evaluation of $D(r)$ and $l_j(r)$ are independent of the underlying circuit, we can amortize the cost of the evaluation into all circuits of the same size.

The remaining work is to evaluate $A_i(r)$, $B_i(r)$, $C_i(r)$ from the Lagrange polynomials $l_j(r)$ $(j = 1, \cdots, |C_R|)$. But this is reduced to merely several additions of $l_j(r)$ polynomials – note that the coefficients a_{ij}, b_{ij}, c_{ij} are all either 0 or 1. The number of wires in the circuit that can contribute to the multiplication gates is at most $2|C_R|$. The total number of additions to evaluate $A_i(r)$ and $B_i(r)$ is at most the number of wires in the circuit that can contribute to the multiplication gates. Then, the total number of additions to evaluate $A_i(r)$ and $B_i(r)$ is at most $2|C_R|$. The total number of additions to evaluate $C_i(r)$ is $(|W_R| + |Y|)$, since it takes $(|W_R| + |C_R|) \cdot (e + c)$ to generate the commitment queries (where, the whole cost of setup is at most $(|W_R| + |C_R|) \cdot (e + c)/\beta + (f_{div} + 5f) \cdot |C_R|/\beta + (2|C_R| + |W_R| + |Y|) \cdot g/\beta)$, where g is the cost of addition over a finite field. Since g is small, we omit addition cost in the tables of cost, as Zaatar [26] does.

Compared to Zaatar, RIVER introduces an extra cost of $(f_{div} + 5f) \cdot |C_R|/\beta + (2|C_R| + |W_R| + |Y|) \cdot g/\beta$ to the total cost of setup. However, notice that this part of the computation can be amortized over many different circuits, which only share the same size, rather than over many different instances of the same circuit. Thus, RIVER actually introduces a negligible cost in the setup phase.

Linearity Query Generation. The cost of generating the linearity queries for π_H is $\rho \cdot \rho_{lin} \cdot 2 \cdot |C_R| \cdot c/(\beta \cdot \gamma)$. Another group of linearity queries are for the proof π_W. The cost of generating these linearity queries is $\rho \cdot \rho_{lin} \cdot 2 \cdot |W_R| \cdot c/(\beta \cdot \gamma)$. Thus, the total cost of generating linearity queries amounts to $\rho \cdot \rho_{lin} \cdot 2 \cdot (|C_R| + |W_R|) \cdot c/(\beta \cdot \gamma)$.

Divisibility Query Generation, Decommitment Query Generation and Decommitment Test. These are straight-forward, we omit these for simplicity.

Non-amortized Costs. From the construction above, we draw the following observations:

- For $i = 1, \cdots, n$, we have $C_i(t) = 0$ for any $t \in \mathbb{F}$, since the inputs of the circuit cannot be outputs of multiplication gates.
- For $i = n+1, \cdots, n+n'$, we have $A_i(t) = 0$ for any $t \in \mathbb{F}$, since the outputs of the circuit Ψ' cannot be inputs to multiplication gates.
- For $i = n+1, \cdots, n+n'$, we have $B_i(t) = 0$ for any $t \in \mathbb{F}$, since the outputs of the circuit Ψ' cannot be inputs to multiplication gates.

Thus, the verifier's cost in the decision making stage (computing p_A, p_B, p_C) is merely $\rho \cdot (2|X| + |Y|) \cdot f$.

Comparison with Zaatar. We list the amortized and non-amortized cost of both RIVER and Zaatar in Table 7. At this time, it is useful to take $W_R = W_Z$ and $C_R = C_Z$.

We can see that, both the amortized and non-amortized cost of RIVER are less than Zaatar. For the amortized part, which is known as the *investment*, even for cases when $\beta = 1$ and $\gamma = 1$, the cost of RIVER is less than Zaatar. (To have a clear picture, we look at a real example: computing xA where the input x is a $1 \times M$ vector and A is a fixed $M \times M$ matrix. This is widely used in all kinds of scientific computing such as communications, signal processing, and control systems, and is a basic operation of many computations. We use previously published models ([26]) and instantiate the costs as in Table 7. From the instance, for $M > 5000$, We see the amortized cost in RIVER is at least 28% less than that in Zaatar. For $M < 5000$, the improvement is even greater.) Since the same part of the amortized cost in RIVER and Zaatar is dominated by linearity test queries, if we apply the query compressing technique in Ginger ([30]), RIVER will have a more significant improvement compared to Zaatar.

6.2 The Prover

The method to construct the proof vector is the same as that in Zaatar. The cost is $T + 3f \cdot |C_R| \cdot log^2|C_R|$. We omit the details here. The remaining cost is from the fact that the prover needs to compute the coefficients of $A_i(t)$, $B_i(t)$, and $C_i(t)$, $(i = 0, 1, \cdots, m)$. However, this could be amortized. First, remember that each of $A_i(t)$, $B_i(t)$ and $C_i(t)$, $(i = 0, 1, \cdots, m)$ are sums of several Lagrange basis polynomials. The cost to get the coefficients of the Lagrange basis polynomials is independent of the underlying circuit and can be amortized into all circuits of the same size and is negligible. Second, similarly to Section 5.2, the number of additions of Lagrange basis polynomials is at most $2|\Psi| + |Y|$. Each Lagrange basis polynomials has degree at most $|C_R|$. Thus, for each instance, the cost of computing the coefficients is at most $(2|\Psi| + |Y|) \cdot |C_R| \cdot g/\beta$, where g is the cost of addition over the field $\mathbb{F}$. As in Zaatar [26], we omit the addition cost.

When the prover issues the PCP responses, he needs to respond to not only queries for π_W and π_H, but also queries for $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, $(i = 0, 1, \cdots, m)$ and π_D. The cost for the former is $(h + 1) \cdot (|W_R| + |C_R|) \cdot f + \rho \cdot (3|W_R| + |C_R|) \cdot f + \rho_{lin} \cdot 3 \cdot (|W_R| + |C_R|) \cdot \rho \cdot f$. Given that the length of the latter is $|C_R|$ and these responses do not depend on underlying circuit or the proof vector π_W, the cost to compute the responses for the latter can be amortized into all instances of the same circuit size. This cost is $[h + \rho \cdot (3m + 4) \cdot f] \cdot |C_R|/(\beta \cdot \gamma)$.

The comparison in terms of the prover's cost is in Table 8. We also use the computation example in Section 6.1 to demonstrate the improvement. For any $M > 100$, RIVER's non-amortized cost of the prover is at least 55% better than that of Zaatar. We demonstrate the results using $M = 10000$. Although RIVER introduces amortized cost, this cost becomes negligible since it can be amortized into all instances of the same circuit size.

Table 8. Comparison of Cost for Prover in Each Instance

	Ginger	Zaatar	RIVER
Construct proof	$T + f \cdot \|W_G\|^2$	$T + 3f \cdot \|C_Z\| \cdot log^2\|C_Z\|$	$T + 3f \cdot \|C_R\| \cdot log^2\|C_R\|$
	$32s + 3.2 \times 10^9 s$	$32s + 3.2 \times 10^4 s$	$32s + 3.2 \times 10^4 s$
Issue PCP responses	$(h+(\rho \cdot L+1) \cdot f) \cdot (\|C_G\| + \|C_G\|^2)$	$(h + (\rho \cdot L' + 1) \cdot f) \cdot (\|C_Z\| + \|W_Z\|)$	$(h+1) \cdot (\|W_R\| + \|C_R\|) \cdot f + \rho \cdot (3\|W_R\| + \|C_R\|) \cdot f + \rho_{lin} \cdot 3 \cdot (\|W_R\| + \|C_R\|) \cdot \rho \cdot f + [h + \rho \cdot (3m + 4) \cdot f] \cdot \|C_R\|/(\beta \cdot \gamma)$
	$2.9 \times 10^{12} s$	$9.0 \times 10^4 s$	$4.0 \times 10^4 s + \frac{7.7 \times 10^{10}}{(\beta \cdot \gamma)} s$

T: cost of computing the task h: cost of ciphertext add plus multiply
$L = 3\rho_{lin} + 2$: number of (high order) PCP queries in Ginger
$L' = 6\rho_{lin} + 4$: number of PCP queries in Zaatar

7 Conclusions

The state-of-the-art designs such as Pepper/Ginger/Zaatar combine a commitment protocol to a linear PCP, achieving breakthroughs in verifiable computation. However, the high computation, communication and storage costs still keep general verifiable computation away from practicality. In this paper, we presented a new verifiable-computation protocol called RIVER. We show that RIVER reduces the amortized computational costs of the verifier and the non-amortized cost of the prover. Namely, the number of batched instances of the protocol, required for amortization, will largely decrease. RIVER introduces only a negligible increase in the prover's costs. However, this increased cost can be amortized over instances of different circuits of the same size. Thus, this part can be done only once, but used for all possible verifications.

In addition, for the first time in the context of verifiable computation, we address the problem of reducing the informational costs. RIVER removes the requirement that the verifier has to access the circuit description during query generating. Furthermore, this feature of RIVER can be viewed as a first step

towards applying QAP-based arguments to the secure outsourcing of verification introduced in [25].

References

1. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable delegation of computation over large datasets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 111–131. Springer, Heidelberg (2011)
2. Boneh, D., Freeman, D.M.: Homomorphic signatures for polynomial functions. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 149–168. Springer, Heidelberg (2011)
3. Ergun, F., Kumar, S.R.: Approximate checking of polynomials and functional equations. In: Proceedings of the 37th Annual Symposium on Foundations of Computer Science, pp. 592–607. IEEE Computer Society, Washington, DC (1996)
4. Golle, P., Mironov, I.: Uncheatable distributed computations. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 425–440. Springer, Heidelberg (2001)
5. Karame, G.O., Strasser, M., Čapkun, S.: Secure remote execution of sequential computations. In: Qing, S., Mitchell, C.J., Wang, G. (eds.) ICICS 2009. LNCS, vol. 5927, pp. 181–197. Springer, Heidelberg (2009)
6. Sion, R.: Query execution assurance for outsourced databases. In: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB 2005, pp. 601–612. VLDB Endowment (2005)
7. Thompson, B., Haber, S., Horne, W.G., Sander, T., Yao, D.: Privacy-preserving computation and verification of aggregate queries on outsourced databases. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 185–201. Springer, Heidelberg (2009)
8. Wang, C., Ren, K., Wang, J.: Secure and practical outsourcing of linear programming in cloud computing. In: INFOCOM, pp. 820–828. IEEE (2011)
9. Wang, C., Ren, K., Wang, J., Urs, K.M.R.: Harnessing the cloud for securely solving large-scale systems of linear equations. In: Proceedings of the 2011 31st International Conference on Distributed Computing Systems, ICDCS 2011, pp. 549–558. IEEE Computer Society, Washington, DC (2011)
10. Atallah, M.J., Frikken, K.B.: Securely outsourcing linear algebra computations. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010, pp. 48–59. ACM, New York (2010)
11. Garofalakis, M.: Proof sketches: Verifiable in-network aggregation. In: IEEE Internation Conference on Data Engineering, ICDE (2007)
12. Przydatek, B., Song, D., Perrig, A.: Sia: secure information aggregation in sensor networks. In: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003, pp. 255–265. ACM, New York (2003)
13. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. 18(1), 186–208 (1989)
14. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC 2008, pp. 113–122. ACM, New York (2008)
15. Canetti, R., Riva, B., Rothblum, G.N.: Two 1-round protocols for delegation of computation. Cryptology ePrint Archive, Report 2011/518 (2011), `http://eprint.iacr.org/`

16. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS 2012, pp. 90–112. ACM, New York (2012)
17. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013)
18. Parno, B., Gentry, C., Howell, J., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: The IEEE Symposium on Security and Privacy, IEEE S&P 2013 (2013)
19. Arora, S., Safra, S.: Probabilistic checking of proofs; a new characterization of np. In: Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, SFCS 1992, pp. 2–13. IEEE Computer Society, Washington, DC (1992)
20. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, STOC 1991, pp. 21–32. ACM, New York (1991)
21. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (1988)
22. Ishai, Y., Kushilevitz, E., Ostrovsky, R.: Efficient arguments without short pcps. In: Proceedings of the Twenty-Second Annual IEEE Conference on Computational Complexity, CCC 2007, pp. 278–291. IEEE Computer Society, Washington, DC (2007)
23. Setty, S., McPherson, R., Blumberg, A.J., Walfish, M.: Making argument systems for outsourced computation practical (sometimes). In: NDSS (2012)
24. Setty, S., Vu, V., Panpalia, N., Braun, B., Blumberg, A.J., Walfish, M.: Taking proof-based verified computation a few steps closer to practicality. In: USENIX Security (2012)
25. Xu, G., Amariucai, G., Guan, Y.: Delegation of computation with verification outsourcing: Curious verifiers. In: Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2013. ACM (2013)
26. Setty, S., Braun, B., Vu, V., Blumberg, A.J., Parno, B., Walfish, M.: Resolving the conflict between generality and plausibility in verified computation. In: Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys 2013, pp. 71–84. ACM, New York (2013)
27. Setty, S., Blumberg, A.J., Walfish, M.: Toward practical and unconditional verification of remote computations. In: Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems, HotOS 2013, p. 29. USENIX Association, Berkeley (2011)
28. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM 45(3), 501–555 (1998)
29. Knuth, D.E.: Seminumerical Algorithms, the art of computer programming, 3rd edn. Addison-Wesley (2007)
30. Setty, S., Vu, V., Panpalia, N., Braun, B., Ali, M., Blumberg, A.J., Walfish, M.: Taking proof-based verified computation a few steps closer to practicality (extended version). Cryptology ePrint Archive, Report 2012/598 (2012), `http://eprint.iacr.org/`