Verifiable Computation with reduced Informational Costs and Computational Costs

(ESORICS 2014)

Gang Xu, George Amariucai, and Yong Guan

Department of Electrical and Computer Engineering
Iowa State University

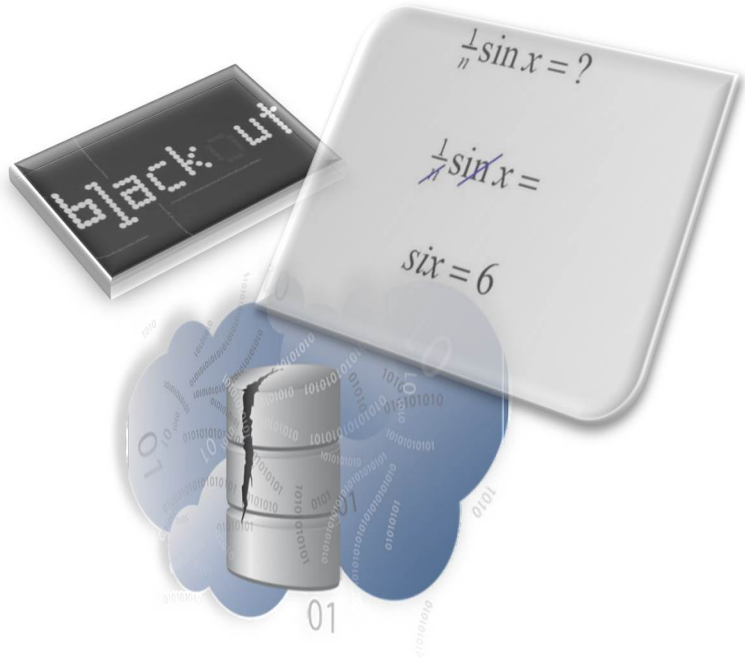
Background

The Cloud Computing provides

- savings in computing infrastructure;
- expert technical consulting.

However, the Cloud

- may be error-prone;
- may not be fully trustable.



Verifiable Computation

An immediate need for *result assurance* naturally arises.

- Some recent works focus on specific problems;
- Others strive for verifying the result of *general* computation:
  - Interactive proof (IP) systems
  - Probabilistically checkable proof (PCP)
  - Variants, such as argument systems.
- The goal: to ensure that verifying the result is much cheaper (from client's perspective) than computing the result from scratch.
- Problems:
  - The prover's workload increases substantially;
  - The client's verification task remains quite expensive;
  - Relatively high communication costs.

Verification Schemes

- Recomputing
- Interactive verification schemes
  - NP proof
  - IP-based
  - PCP-based
    - Problem is represented as a circuit (binary or arithmetic).
    - Proof represented as $[z, z \otimes z]$, where vector z contains values of all circuit wires .
    - Proof is tested at random points by sending queries q of size equal to size of $[z, z \otimes z]$; Expected answers are $\langle q, [z, z \otimes z] \rangle$.
    - Linearity Test.
    - Quadratic Consistency Test.
    - Circuit Correctness Test.

Argument Systems

- Rely on the PCP framework, complemented with a commitment to the proof.
- First argument system (IKO) is proposed by Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky [Ishai et. al., CCC'07].
  - Tools: additive homomorphic encryption $\{\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec}\}$.
  - One commitment and one decommitment for every query.
- Pepper [Setty et. al., NDSS'12] and Ginger [Setty et. al., Usenix'12] refine IKO commitment protocol , such that multiple queries share one commitment.

$\mathcal{V}$: $\alpha_1, \alpha_2, \cdots, \alpha_\mu \in_R \mathbb{F}$.
$\mathcal{V} \rightarrow \mathcal{P}$: $q_1, q_2, \cdots, q_\mu, r + \sum_{i=1}^{\mu} \alpha_i q_i$.
$\mathcal{P} \rightarrow \mathcal{V}$: $(a_1, a_2, \cdots, a_\mu, b)$
$\mathcal{V}$: determines whether $b = s + \sum_{i=1}^{\mu} \alpha_i a_i$.

Single-Commit-Multi-Decommit Design

$\mathcal{P}$**'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 $\perp$.

Quadratic Arithmetic programs (QAPs)

Definition

(Quadratic Arithmetic Programs)[Genaro et. al, Eurocrypt 13] 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 $D(t) \cdot H(t) = P(t)$.

Zaatar – A linear PCP based on QAPs [Setty et.al., Usenix Security 2012]

$\mathcal{V}$ queries an oracle $\pi_W(\cdot) = \langle W, \cdot \rangle$, where $W = (z_m, \cdots, z_{n+n'+1})$ is the intermediate results of the circuit computation.
loop ρ times the steps 1 to 5:

1. Linearity queries generation. $\mathcal{V}$ selects $q_5, q_6 \in_R \mathbb{F}^{m-(n+n')}$ and $q_8, q_9 \in_R \mathbb{F}^{|C_Z|+1}$. He takes $q_7 \leftarrow q_5 + q_6$ and $q_{10} \leftarrow q_8 + q_9$. Perform ρ_{lin} iterations in total. 2. QAP queries generation. $\mathcal{V}$ selects $\tau \in_R \mathbb{F}$ and takes:

- $q_A \leftarrow (A_{(m)}(\tau), A_{(m-1)}(\tau), \cdots, A_{(n+n'+1)}(\tau))$, and $q_1 \leftarrow (q_A + q_5)$.
- $q_B \leftarrow (B_{(m)}(\tau), B_{(m-1)}(\tau), \cdots, B_{(n+n'+1)}(\tau))$, and $q_2 \leftarrow (q_B + q_5)$.
- $q_C \leftarrow (C_{(m)}(\tau), C_{(m-1)}(\tau), \cdots, C_{(n+n'+1)}(\tau))$, and $q_3 \leftarrow (q_C + q_5)$.
- $q_H \leftarrow (1, \tau, \tau^2, \cdots, \tau^{|C_Z|})$, and $q_4 \leftarrow q_H + q_8$

3. Querying π_W. $\mathcal{V}$ sends out $q_1, q_2, \cdots, q_{4+6\rho}$ and gets back $\pi_W(q_1), \cdots, \pi_W(q_{4+6\rho})$.
4. Linearity tests. Check whether following holds: $\pi_W(q_7) = \pi_W(q_6) + \pi_W(q_5)$, $\pi_W(q_{10}) = \pi_W(q_9) + \pi_W(q_8)$ and likewise for all other $\rho - 1$ iterations.
5. Divisibility test: $\mathcal{V}$ takes:

- $A_\tau = (\pi_W(q_1) - \pi_W(q_5) + \sum_{i=1}^{n+n'} z_i \cdot A_i(\tau) + A_0(\tau))$
- $B_\tau = (\pi_W(q_2) - \pi_W(q_5) + \sum_{i=1}^{n+n'} z_i \cdot B_i(\tau) + B_0(\tau))$
- $C_\tau = (\pi_W(q_3) - \pi_W(q_5) + \sum_{i=1}^{n+n'} z_i \cdot C_i(\tau) + C_0(\tau))$

and $\mathcal{V}$ checks whether the following holds: $D(\tau) \cdot (\pi_H(q_4) - \pi_H(q_8)) = A_\tau \cdot B_\tau - C_\tau$.
If $\mathcal{V}$passes all the tests above, accept.

Zaatar – Costs and Redundancy

- Zaatar views QAPs in the context of linear PCPs;
- Zaatar proceeds as follows:
  - Commitment
  - Linearity testing
  - Divisibility testing
  - Decommitment
- Ideally, we would like to say that the commitment with multiple decommitments precludes linearity testing.
- Can we discard linearity testing in the general case? – so far, not supported by conclusive arguments.

Linearity Testing

- We introduce a weaker result:

Theorem

The Single-Commit-Multi-Decommit protocol ensures that, if the secret commit information is generated by the prover using an affine function (analytically defined and a priori known to the prover), (the same holds if commitment is generated by the verifier himself), then for all query tuples, unless the prover replies to all queries with the same linear function (linear part of the affine function), the prover will not pass the decommitment test except with probability $\frac{1}{|\mathbb{F}|} + \epsilon_S$ *(probability is over randomness of prover and verifier in decommitment phase).*

RIVER – Reduced-Investment Verifiable Computation Protocol

Contributions of RIVER:

- RIVER reduces the verifier's workload that needs to be amortized. Instead of batching over instances of the same circuit (as in existing works), RIVER can amortize more costs than Zaatar over *all different circuits of the same size*.
- Although RIVER introduces additional amortized costs 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 requirement that verifier has to access the circuit description during query generation – this helps with third-party verification.

PCP Querying in RIVER

- Recall in Zaatar $\mathcal{V}$ selects $\tau \in_R \mathbb{F}$ and calculates the queries:
  - $q_A \leftarrow (A_{(m)}(\tau), A_{(m-1)}(\tau), \cdots, A_{(n+n'+1)}(\tau))$
  - $q_B \leftarrow (B_{(m)}(\tau), B_{(m-1)}(\tau), \cdots, B_{(n+n'+1)}(\tau))$
  - $q_C \leftarrow (C_{(m)}(\tau), C_{(m-1)}(\tau), \cdots, C_{(n+n'+1)}(\tau))$
- In RIVER, $\mathcal{V}$ outsources this computation to $\mathcal{P}$:

$$A_i(\tau) = \pi_A^{(i)}(q_H) = \langle K_A^{(i)}, q_H \rangle$$
$$B_i(\tau) = \pi_B^{(i)}(q_H) = \langle K_B^{(i)}, q_H \rangle$$
$$C_i(\tau) = \pi_C^{(i)}(q_H) = \langle K_C^{(i)}, q_H \rangle$$

, where $q_H = (1, \tau, \tau^2, \cdots, \tau^l)$ and $K_A^{(i)}$, $K_B^{(i)}$, $K_C^{(i)}$ are coefficient vectors.

PCP Querying in RIVER

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

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

$\mathcal{V}$ 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: $\pi_H(q_H)$.

PCP Querying in RIVER

$\mathcal{V}$ 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 $\pi_W(q_A) \leftarrow \pi_W(q_1) - \pi_W(q_4)$, $\pi_W(q_B) \leftarrow \pi_W(q_2) - \pi_W(q_4)$, $\pi_W(q_C) \leftarrow \pi_W(q_3) - \pi_W(q_4)$.

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: $\pi_D(q_H) \cdot \pi_H(q_H) = (\pi_Z(q_A) + p_A) \cdot (\pi_Z(q_B) + p_B) - (\pi_Z(q_C) + p_C)$.

Commitment and Decommitment in RIVER

- Commitment and decommitment are generally similar to Zaatar.
- The exception – commitment and decommitment for $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$:

$\mathcal{P}$**'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 **??**.

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 $\perp$.

RIVER Informational Costs

- Once committed, all the queries in the verification are independent on the circuit description.
- During query generation in the verification stage, verifier does not need to access the circuit description.
- RIVER separates verification workload that involves only non-sensitive information from verification workload that involves sensitive information (e.g. the circuit information).
- A third-party verifier can undertake the workload involving only non-sensitive information.

RIVER Performance – The Verifier

- 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, a large part of this cost is independent of the underlying circuits, and only depends on the size of the circuit.
- This part of the computation can be amortized over many different circuits, which only share the same size:
  - Target polynomial $D(t) = \prod_{k=1}^{|C_R|}(t - \sigma_k)$ is determined by only the circuit size.
  - We can express $A_i(t)$ (same goes for $B_i(t)$, $C_i(t)$) in the form of Lagrange Polynomial interpolation: $A_i(t) = \sum_{j=1}^{|C_R|} a_{ij} \cdot l_j(t)$, where $l_j(t) = \prod_{1 \leq k \leq |C_R|, k \neq j} \frac{(t-\sigma_k)}{(\sigma_j - \sigma_k)}$ are Lagrange basis polynomials. We can represent these as: $l_j(t) = \frac{D(t)}{(t-\sigma_j) \cdot \frac{1}{v_j}}$ – hence depending only on circuit size.

RIVER Performance – The Prover

- Construction of the proof vector is the same as that in Zaatar.
- Additionally, prover needs to compute the coefficients of $A_i(t)$, $B_i(t)$, and $C_i(t)$, $(i = 0, 1, \cdots, m)$. This cost can be amortized, over multiple circuits of the same size, by writing $A_i(t)$, $B_i(t)$ and $C_i(t)$, as sums of Lagrange basis polynomials – recall these polynomials are independent of the underlying circuit (only depend on size)

Summary

- Under certain conditions, single-commit-multi-decommin protocol precludes linearity tests. This holds if the commitment is produced using an affine function known to prover.
- QAP polynomial evaluations in τ can be computed by prover, but corresponding commitment has to be computed by verifier. No linearity tests are required, and commitment costs can be amortized over multiple circuits of same size, and over multiple values of τ.
- Once QAP polynomial commitment is produced, verifier no longer requires circuit description – hence the remaining verification can be outsourced to a third party (*reduced informational costs*).

Thank you!

Gang Xu, gxu@iastate.edu
George T. Amariucai, gamari@iastate.edu
Yong Guan, guan@iastate.edu