Pinocchio

Nearly Practical Verifiable Computation

Guilhermo Mocelim   Felipe Vicentin

IC - Unicamp

June 26, 2024

Outline

Motivation

- Computational power is often asymmetric.
- Client may want to outsource to powerful *workers*.
  - Cloud computing;
  - Grid computing;
  - Distributed computing.
- Client needs to *verify* computations.
  - Fast verification;
  - Minimize overhead.

Verifiable Computation (VC)

- A VC scheme allows a client to outsource the evaluation of $F(u)$.
- Then, the client can verify the correcteness of $F(u)$.

Definition (Public Verifiable Computation)

A public verifiable computation scheme $\mathcal{VC}$ consists of a set of three polynomial-time algorighms (KeyGen, Compute, Verify) defined as follows.

- $(\mathsf{EK}_F, \mathsf{VK}_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$: Outputs a *public* evaluation key EK_F and a public verification key VK_F.
- $(y, \pi_y) \leftarrow \mathsf{Compute}(\mathsf{EK}_F, u)$: Outputs $y \leftarrow F(u)$ and a proof π_y.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{VK}_F, u, y, \pi_y)$: Uses VK_F and outputs whether $F(u) = y$.

Correctness: Verify always outputs 1 if $y = F(u)$.
Security: Verify has negligible probability of outputting 1 for a wrong evaluation.
Efficiency: KeyGen is a one-time operation that is cheaper than F.

Quadratic Arithmetic Programs (QAP)

Definition

A QAP Q over field $\mathbb{F}$ contains three sets of $m+1$ polynomials $\mathcal{V} = \{v_k(x)\}$, $\mathcal{W} = \{w_k(x)\}$, $\mathcal{Y} = \{y_k(x)\}$, for $k \in \{0, \ldots, m\}$, and a target polynomial $t(x)$.

Suppose F is a function that takes as input n elements of $\mathbb{F}$ and outputs n' elements, for a total of $N = n + n'$ I/O elements.
Then, we say that Q computes F if: $(c_1, \ldots, c_N) \in \mathbb{F}^N$ is a valid assignment of F's inputs and outputs iff there exist coefficients $(c_{N+1}, \ldots, c_m)$ such that $t(x)$ divides $p(x)$ where:

$$p(x) = \left(v_0(x) + \sum_{k=1}^{m} c_k v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} c_k w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} c_k y_k(x)\right)$$

In other words, there must exist $h(x)$ such that $h(x) \cdot t(x) = p(x)$.

Quadratic Arithmetic Programs (QAP)

Let us build a QAP from an arithmetic circuit C.

- For each *multiplicative* gate, pick an arbitrary root $r_g \in \mathbb{F}$.
- Define the target polynomial as $t(x) = \prod_g (x - r_g)$.
- Associate an index $k \in [m] = \{1, \ldots, m\}$ to each *input* and *output* from a *multiplication gate* (not addition gates).
- Define $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{Y}$ enconding the left input, right input and output of each multiplication gate, respectively.
  - $v_k(r_g) = 1$ if the k-th wire is a left input to gate g, and $v_k(r_g) = 0$ otherwise.
  - $w_k(r_g) = 1$ if the k-th wire is a right input to gate g, and $w_k(r_g) = 0$ otherwise.
  - $y_k(r_g) = 1$ if the k-th wire is a output to gate g, and $y_k(r_g) = 0$ otherwise.
- This way, we have a nice simplification:

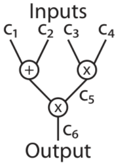
$$\left(v_0(x) + \sum_{k=1}^{m} c_k v_k(r_g)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} c_k w_k(r_g)\right)$$

$$= \left(\sum_{k \in I_{\text{left}}} c_k\right) \cdot \left(\sum_{k \in I_{\text{right}}} c_k\right) = c_g y_k(r_g) = c_g \implies p(r_g) = 0$$

Quadratic Arithmetic Programs (QAP)

Example

Let $F(c_1, c_2, c_3, c_4) = (c_1 + c_2) \times (c_3 \times c_4)$



	(r_5, r_6)		(r_5, r_6)		(r_5, r_6)
$v_1(r_i)$	(0,1)	$w_1(r_i)$	(0,0)	$y_1(r_i)$	(0,0)
$v_2(r_i)$	(0,1)	$w_2(r_i)$	(0,0)	$y_2(r_i)$	(0,0)
$v_3(r_i)$	(1,0)	$w_3(r_i)$	(0,0)	$y_3(r_i)$	(0,0)
$v_4(r_i)$	(0,0)	$w_4(r_i)$	(1,0)	$y_4(r_i)$	(0,0)
$v_5(r_i)$	(0,0)	$w_5(r_i)$	(0,1)	$y_5(r_i)$	(1,0)
$v_6(r_i)$	(0,0)	$w_6(r_i)$	(0,0)	$y_6(r_i)$	(0,1)

$$t(x) = (x - r_5)(x - r_6)$$

Quadratic Arithmetic Programs (QAP)

- **Strong QAPs**: The same set of c_i must be applied to all sets of polynomials.
- It is possible to convert any regular QAP to a strong QAP.
- QAP's degree increases to $3d + 2N$ (tripling what we had before).

Bilinear maps

- Let $\mathbb{G}$ be a group of order q for some large prime p.
- A bilinear map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ is a function such that:

$$e(g^a, g^b) = e(g^{ab}, g)$$
$$\frac{e(g^{ab}, g)}{e(g^c, g)} = e(g^{ab-c}, g)$$

QAP $\rightarrow$ VC: KeyGen

Let F be a function with N I/O values from $\mathbb{F}$.

1. Convert F into a QAP $Q = (t(x), \mathcal{V}, \mathcal{W}, \mathcal{Y})$ of size m and degree d.
2. Let $I_{\text{mid}} = \{N+1, \ldots, m\}$ be the non-IO-related indices.
3. Let e be a non-trivial bilinear map $e \colon \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, and let $g \in \mathbb{G}$.
4. Choose $s, \alpha, \beta_v, \beta_w, \beta_y, \gamma \xleftarrow{R} \mathbb{F}$.
5. Construct the public keys as:

$$\begin{aligned}\mathrm{EK}_F = (\quad & \{g^{v_k(s)}\}_{k \in I_{\text{mid}}}, \{g^{w_k(s)}\}_{k \in [m]}, \{g^{y_k(s)}\}_{k \in [m]}, \\ & \{g^{\alpha v_k(s)}\}_{k \in I_{\text{mid}}}, \{g^{\alpha w_k(s)}\}_{k \in [m]}, \{g^{\alpha y_k(s)}\}_{k \in [m]}, \\ & \{g^{s^i}\}_{i \in [d]}, \{g^{\alpha s^i}\}_{i \in [d]} \quad)\end{aligned}$$

$$\begin{aligned}\mathrm{VK}_F = (\quad & g^1, g^\alpha, g^\gamma, g^{\beta_v \gamma}, g^{\beta_w \gamma}, g^{\beta_y \gamma}, \\ & g^{t(s)}, \{g^{v_k(s)}\}_{k \in [N]}, g^{v_0(s)}, g^{w_0(s)}, g^{y_0(s)} \quad)\end{aligned}$$

QAP $\to$ VC: Compute

1. On input u, the worker evaluates the circuit for F to obtain $y \leftarrow F(u)$; it also learns all c_i values.
2. Worker has Q, so it knows $p(x)$.
3. The worker computes $v(s)$, $w(s)$ and $y(s)$ using exponents.

$$g^{v_0(s)} \cdot \prod_{k \in [m]} \left(g^{v_k(s)}\right)^{c_k} = g^{v_0(s) + \sum_{k \in [m]} c_k v_k(s)} = g^{v(s)}$$

$$g^{\alpha v_0(s)} \cdot \prod_{k \in I_{\text{mid}}} \left(g^{\alpha v_k(s)}\right)^{c_k} = g^{\alpha(v_0(s) + \sum_{k \in I_{\text{mid}}} c_k v_k(s))} = g^{\alpha v_{\text{mid}}(s)}$$

4. It can also compute $h(s)$.

$$\frac{p(x)}{t(x)} = h(x) = h_0 + h_1 x + h_2 x^2 + \cdots \implies h(s) = h_0 + h_1 s + h_2 s^2 + \ldots$$

$$\implies \prod_{i \in [d]} \left(g^{s^i}\right)^{h_i} = g^{\sum_{i \in [d]} h_i s^i} = g^{h(s)}$$

5. Finally, it outputs the proof π_y as:

$$\begin{array}{lllll} (& g^{v_{\text{mid}}(s)}, & g^{w(s)}, & g^{y(s)}, & g^{h(s)}, \\ & g^{\alpha v_{\text{mid}}(s)}, & g^{\alpha w(s)}, & g^{\alpha y(s)}, & g^{\alpha h(s)}, \\ & g^{\beta_v v(s)+\beta_w w(s)+\beta_y y(s)} & & &) \end{array}$$

QAP $\rightarrow$ VC: Verify

1. Check correctness of α and β proofs.

$$e\left(g^{v_{\text{mid}}(s)}, g^{\alpha}\right) = e\left(g^{\alpha v_{\text{mid}}(s)}, g\right)$$

2. Divisibility check for the QAP: compute $g^{v_{\text{io}}(s)} = \prod_{k \in [N]} \left(g^{v_k(s)}\right)^{c_k}$ and check:

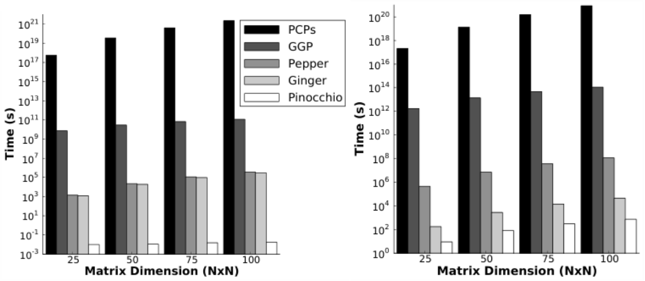
$$\frac{e\left(g^{v_0(s)} g^{v_{\text{io}}(s)} g^{v_{\text{mid}}(s)}, g^{w_0(s)} g^{w(s)}\right)}{e\left(g^{y_0(s)} g^{y(s)}, g\right)} = e\left(g^{h(s)}, g^{t(s)}\right)$$

QAP $\rightarrow$ VC: Security

- If the adversary manages to provide a proof of a false statement that verifies, then these polynomials must not actually correspond to a QAP solution.
- So, either $p(x)$ is not actually divisible by $t(x)$ (in this case we break $2q$-SDH) or $v(x) = v_{\text{io}}(x) + v_{\text{mid}}(x)$, $w(x)$ and $y(x)$ do not use the same linear combination (in this case we break q-PDH because in the proof we chose β in a clever way).

Conclusion: Pinocchio

- Uses regular QAPs.
- Implemented a basic compiler that translates C code to QAPs.
- Fast (about 10 ms).
- Proof is of constant length (little overhead).



(a) Per-Instance Verification Latency (b) Worker Latency

Questions?