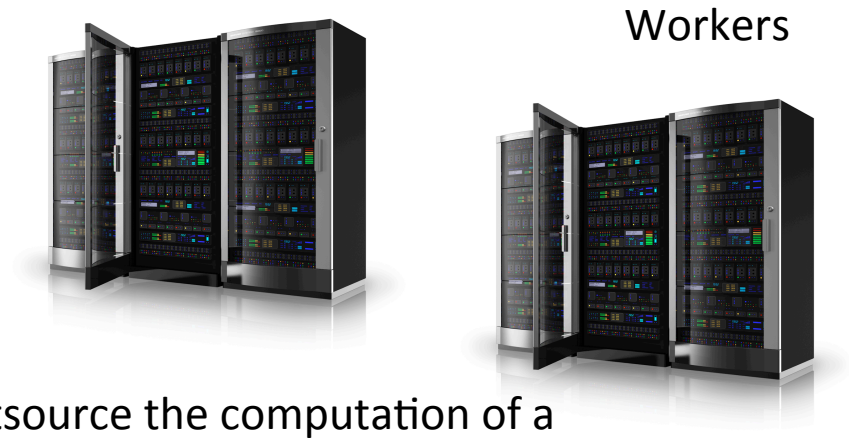
Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers

Rosario Gennaro et al.

Presenter: Xi Chen

Verifiable Computation



Client

outsource the computation of a function F on various dynamically-chosen inputs $x_1, \ldots, x_k$

return the result of the function evaluation (e.g.,y_i=F(x_i))as well as a proof

Prior Work

- Audit-based solutions [24,9]
- Secure co-processors [28,33]
- Trusted Platform Modules(TPMs) [29]
- Wallets with observers [10]
- Interactive proofs [6,15]
- Micali's CS Proofs [22]

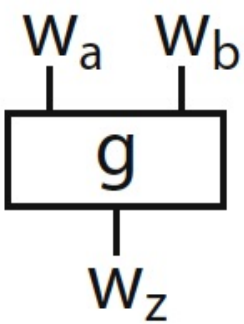
Verifiable Computation Scheme

- Preprocessing. A one-time stage in which the client computes some auxiliary information associated with F.
- Input Preparation. The client prepares some public and private information about x and sends the public part to the worker.
- Output Computation and Verification. The worker computes an encoding of the value F(x), and then returns it to the client. The client can compute the value F(x) from the returned value and verify its correctness.

Yao's Garbled Circuit Construction

- Compute F with private inputs a and b
- Alice converts F into a boolean circuit C
- Alice prepares G(C) and G(a) and sends them to Bob
- Alice and Bob engage in oblivious transfers so that Bob obtains G(b)
- Bob then derives G(F(a,b)).
- Alice translates this into F(a,b)

Yao's Garbled Circuit Construction



(a)

w_a	w_b	w_z
0	0	g(0,0)
0	1	g(0,1)
1	0	g(1,0)
1	1	g(1,1)

(b)

w_a	w_b	w_z
k_a^0	k_b^0	$k_z^{g(0,0)}$
k_a^0	k_b^1	$k_z^{g(0,1)}$
k_a^1	k_b^0	$k_z^{g(1,0)}$
k_a^1	k_b^1	$k_z^{g(1,1)}$

(c)

w_a	w_b	w_z
k_a^0	k_b^0	$E_{k_a^0}(E_{k_b^0}(k_z^{g(0,0)}))$
k_a^0	k_b^1	$E_{k_a^0}(E_{k_b^1}(k_z^{g(0,1)}))$
k_a^1	k_b^0	$E_{k_a^1}(E_{k_b^0}(k_z^{g(1,0)}))$
k_a^1	k_b^1	$E_{k_a^1}(E_{k_b^1}(k_z^{g(1,1)}))$

(d)

Fig. 1. Yao's Garbled Circuits. *The original binary gate* **(a)** *can be represented by a standard truth table* **(b)**. *We then replace the 0 and 1 values with the corresponding randomly chosen* λ*-bit values* **(c)**. *Finally, we use the values for* w_a *and* w_b *to encrypt the values for the output wire* w_z **(d)**. *The random permutation of these ciphertexts is the garbled representation of gate g.*

$$\gamma_{ij} = E_{k_a^i}(E_{k_b^j}(k_z^{g(i,j)})), \text{ where } i \in \{0,1\}, j \in \{0,1\}$$

Homomorphic Encryption

- KeyGen_{ξ}
- Encrypt_{ξ}
- Decrypt_{ξ}
- Evaluate_{ξ}

Problem Definition

- Verifiable computation scheme (VC)

1. KeyGen $(F,\lambda) \longrightarrow (PK,SK)$
2. $ProbGen_{SK}(x) \longrightarrow (\sigma_x,\tau_x)$. σ_x: public value ; τ_x: secret value;
3. $Compute_{PK}(\sigma_x) \longrightarrow \sigma_y$
4. $Verify_{SK}(\tau_x, \sigma_y) \longrightarrow y$ or $\perp$

An efficient verifiable-computation scheme with input and output privacy

1. KeyGen(F,λ) → (PK,SK): PK ← $\cup_g(\gamma_{00}^g,\gamma_{01}^g,\gamma_{10}^g,\gamma_{11}^g)$; SK ← $\cup_i(w_i^0,w_i^1)$.

2. $\text{ProbGen}_{SK}(x)$ → σ_x: Run the fully−homomorphic encryption scheme's key generation algorithm to create a new key pair: $(PK_\xi, SK_\xi) \leftarrow \text{KeyGen}_\xi(\lambda)$. Let $w_i \subset SK$ be the wire values representing x. Set $\sigma_x \leftarrow (PK_\xi, \text{Encrypt}_\xi(PK_\xi, w_i))$ and $\tau_x \leftarrow SK_\xi$

An efficient verifiable-computation scheme with input and output privacy

3.Compute$_{PK}(\sigma_x)$ → σ_y: Caculate Encrypt$_\xi$(PK$_\xi$, γ_i). Construct a circuit Δ that on input w, w', γ outputs $D_w(D_{w'}(\gamma))$. D is the decryption algorithm corresponding to the encryption E in Yao's garbling. Calculate Evaluate$_\xi$(Δ,Encrypt$_\xi$(PK$_\xi$,w_i), Encrypt$_\xi$(PK$_\xi$, γ_i)) to get Encrypt$_\xi$(PK$_\xi$, $\underline{w}_i$).

$\underline{W}_i$ is the wire value representing y=F(x)

4.Verify$_{SK}(\sigma_y)$ → y or $\perp$: Use SK$_\xi$ to decrypt Encrypt$_\xi$(PK$_\xi$, $\underline{w}_i$), obtaining $\underline{w}_i$. Use SK to map the wire values to an output y. If the decryption or mapping fails, then output $\perp$.

How to handle cheating workers

- Suppose that the adversary wants to determine the first bit w_{11}^b of the first label.
- Run Compute as before, obtaining ciphertexts that encrypt the bits $\underline{w}_i$ of a label for the output wire.
- XOR w_{11}^b with the first bit of $\underline{w}_i$ to obtain $\underline{w}_i'$, and sends $\underline{w}_i'$ as its response.
- If Verify outputs $\perp$, then w_{11}^b must have been a 1; otherwise, it is a 0 with overwhelming probability.

Questions?