Non-Interactive Verifiable Computing

Outsourcing Computation to Untrusted Workers

Motivation

- Desire to outsource computing from weak clients to powerful cloud services - e.x. SETI@Home, Folding@Home

- Dishonest clients may modify their client software. To return plausible results without performing actual work

- Could address this with redundancy, though a waste of resources, and doesn’t defend against colluding users

- Applications outsourced to cloud services may be critical that accidental errors have to be addressed. Financial incentive to return incorrect answers.

“Verifiable Computation”

- Enable computationally-weak clients to outsource computation of a function F on various dynamically-chosen inputs to N workers.

- Workers return the result of the function evaluation, as well as a proof that the computation was carried out correctly

- Verification of the proof should require substantially less computational effort than computing the function from scratch.

Ideas from Paper

- Can adapt Yao's Garbled Circuit Construction to allow a client to outsource the computation of a function on a *single input*

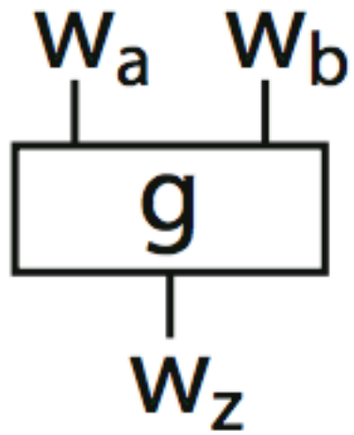
- Combine Yao's Garbled Circuit w/ a fully homomorphic encryption system to safely reuse garbled circuit for multiple inputs (new PK generated for every input)

Procedure

1. Alice constructs circuit representation from function
2. Alice garbles the circuit (Yao's garbled circuit technique)
3. Alice garbles input
4. Alice sends garbled input and garbled version of circuit to Bob (worker)
5. Bob attempts to decrypt garbled circuit with given input to get an output
6. Bob sends output result back to Alice
7. Alice verifies the result by comparing to original circuit truth table

On subsequent inputs, Bob could just reuse the same response!

1. Processing - Garbling



(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.*

2. Transmit $\mathscr{e}$ and x

F → C → $\mathscr{e}$

X → x

$\mathscr{e}$

$\text{Enc}_{a_0}(\text{Enc}_{b_0}(z_0))$

$\text{Enc}_{a_0}(\text{Enc}_{b_1}(z_0))$

$\text{Enc}_{a_1}(\text{Enc}_{b_0}(z_0))$

$\text{Enc}_{a_1}(\text{Enc}_{b_1}(z_1))$

$x = a_1 b_0$

3. Evaluate the garbled circuit

i) Decrypt with a_1

ii) Decrypt with b_0

$\text{Enc}_{a_0}(\text{Enc}_{b_0}(z_0))$

$\text{Enc}_{a_0}(\text{Enc}_{b_1}(z_0))$

z_0

$\text{Enc}_{b_1}(z_1)$

4. Verify the result y

If $y = z_0$

y ← 0

If $y = z_1$

y ← 1

Else

Reject!

$z_i \in \{0, 1\}^\lambda$

Security Intuition:

1. Probability to guess z_0 or z_1 is $\frac{1}{2^\lambda}$
2. Encryption scheme guarantees secrecy of the "other" z_i

Ideas from Paper

- Can adapt Yao's Garbled Circuit Construction to allow a client to outsource the computation of a function on a *single input*

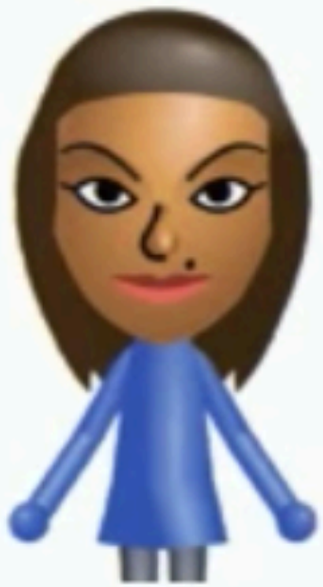
- Combine Yao's Garbled Circuit w/ a fully **homomorphic encryption** system to safely reuse garbled circuit for multiple inputs (new PK generated for every input)

  - ***Homomorphic encryption:*** *allows computation on cypher texts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext.*

Procedure

1. Alice constructs circuit representation from function
2. Alice garbles the circuit (Yao's garbled circuit technique)
3. Alice garbles input
4. Alice sends **homomorphically encrypted garbled input + PubKey** and garbled version of circuit to Bob (worker)
5. **Bob uses PubKey to homomorphically encrypt garbled circuit**
6. **Bob applies encrypted circuit on encrypted input to get encrypted output**
7. ~~Bob attempts to decrypt garbled circuit with given input to get an output~~
8. Bob sends output result back to Alice
9. **Alice decrypts output using her private key**
10. Alice verifies the result by comparing to original circuit truth table
11. **For new inputs Alice uses a new public key, preventing Bob from reusing his response since it wouldn't decrypt correctly**

Combine with One-Time Verifiable Scheme



e

x PubKey

y

$y \leftarrow e(x)$

F → C → e

x → x

w PubKey

$t \leftarrow e(w)$

y → y = F(x)

t

w → w

y

Reject!

As a Protocol/Scheme

1. **KeyGen**$(F,\lambda) \rightarrow (PK,SK)$: Represent F as a circuit C. Following Yao's Circuit Construction (see Section 2), choose two values, $w_i^0, w_i^1 \xleftarrow{R} \{0,1\}^\lambda$ for each wire w_i. For each gate g, compute the four ciphertexts $(\gamma_{00}^g, \gamma_{01}^g, \gamma_{10}^g, \gamma_{11}^g)$ described in Equation 1. The public key PK will be the full set of ciphertexts, i.e., $PK \leftarrow \cup_g(\gamma_{00}^g, \gamma_{01}^g, \gamma_{10}^g, \gamma_{11}^g)$, while the secret key will be the wire values chosen: $SK \leftarrow \cup_i(w_i^0, w_i^1)$.
2. **ProbGen**$_{SK}(x) \rightarrow \sigma_x$: Run the fully-homomorphic encryption scheme's key generation algorithm to create a new key pair: $(PK_{\mathcal{E}}, SK_{\mathcal{E}}) \leftarrow$ **KeyGen**$_{\mathcal{E}}(\lambda)$. Let $w_i \subset SK$ be the wire values representing the binary expression of x. Set the public value $\sigma_x \leftarrow (PK_{\mathcal{E}},$ **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, w_i))$ and the private value $\tau_x \leftarrow SK_{\mathcal{E}}$.

As a Protocol/Scheme Cont.

3. **Compute**$_{PK}(\sigma_x) \rightarrow \sigma_y$: Calculate **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, \gamma_i)$. Construct a circuit Δ that on input w, w', γ outputs $D_w(D_{w'}(\gamma))$, where D is the decryption algorithm corresponding to the encryption E used in Yao's garbling (therefore Δ computes the appropriate decryption in Yao's construction). Calculate **Evaluate**$_{\mathcal{E}}(\Delta,$ **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, w_i),$ **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, \gamma_i))$ repeatedly, to decrypt your way through the ciphertexts, just as in the evaluation of Yao's garbled circuit. The result is $\sigma_y \leftarrow$ **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, \bar{w}_i)$, where $\bar{w}_i$ are the wire values representing $y = F(x)$ in binary.
4. **Verify**$_{SK}(\sigma_y) \rightarrow y \cup \perp$: Use $SK_{\mathcal{E}}$ to decrypt **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, \bar{w}_i)$, obtaining $\bar{w}_i$. Use SK to map the wire values to an output y. If the decryption or mapping fails, then output $\perp$.