Verifiable Computing: Outsourcing Computation to Malicious Workers
Bryan Parno (CMU), Rosario Gennaro (IBM), Craig Gentry (IBM)

1. Why Outsource?
- Distributed Computing
  - Many weak clients replace supercomputers
- Computing on Demand
  - Pay for what you use
  - Quickly scale up/down
- Mobile Devices
  - Outsource resource-intensive tasks

Can you trust it?

2. Computing \( y \leftarrow F(x) \)
\[ F \rightarrow C \]
\[ G(C), G(x) \]
\[ G(x) \]
\[ \downarrow \]
\[ G(C) \]
\[ G(y) \]

Garble

Verify \( G(y) \) is “correct”

3. Algorithms

- **KeyGen** \((F, \lambda) \rightarrow PK, SK\)
  \(O(|F|)\)
- **ProbGen** \(SK(x) \rightarrow G(x)\)
  \(O(|x|)\)
- **Compute** \(PK(G(x)) \rightarrow G(y)\)
  \(O(|F|)\)
- **Verify** \(SK(G(y)) \rightarrow y \text{ or } \bot\)
  \(O(|y|)\)

- **Correctness**: \( y = F(x) \)
- **Security**: Adversary wins if \( \text{Verify} \rightarrow \hat{y} \) s.t. \( \hat{y} \neq \bot \) and \( \hat{y} \neq F(x) \)

4. Garbling a Gate

Choose a nonce/bit value:

\[ a_i, b_i, z_i \in \{0,1\}^\lambda \]

\[ a_0, a_1, b_0, b_1 \]

\[ Z \]

\[ Z_0, Z_1 \]

\[ g \]

\[ A \]

\[ B \]

\[ a_0, a_1, b_0, b_1 \]

\[ E_{00}, E_{01}, E_{10}, E_{11} \]

\[ E_{00} \leftarrow E_{a_0}(E_{b_0}(Z_1)) \]

\[ E_{01} \leftarrow E_{a_0}(E_{b_1}(Z_1)) \]

\[ E_{10} \leftarrow E_{a_1}(E_{b_0}(Z_1)) \]

\[ E_{11} \leftarrow E_{a_1}(E_{b_1}(Z_0)) \]

\[ [Yao82] \]

Accept if:
\[ 2 = z_0 \text{ or } 2 = z_1 \]

Can't be reused!

5. Evaluating a Gate

\[ a_i, b_k \]

\[ E_{10}, E_{11}, E_{01}, E_{00} \]

\[ E_{10} \leftarrow E_{a_i}(E_{b_k}(Z_1)) \]

\[ E_{11} \leftarrow E_{a_i}(E_{b_k}(Z_0)) \]

\[ \hat{z} \]

\[ \hat{z} = z_0 \text{ or } \hat{z} = z_1 \]

\[ G(g) \]

\[ Z \]

\[ Z_0, Z_1 \]

\[ g \]

\[ A \]

\[ B \]

\[ a_0, a_1, b_0, b_1 \]

\[ E_{00}, E_{01}, E_{10}, E_{11} \]

\[ E_{00} \leftarrow E_{a_0}(E_{b_0}(Z_1)) \]

\[ E_{01} \leftarrow E_{a_0}(E_{b_1}(Z_1)) \]

\[ E_{10} \leftarrow E_{a_1}(E_{b_0}(Z_1)) \]

\[ E_{11} \leftarrow E_{a_1}(E_{b_1}(Z_0)) \]

\[ [Yao82] \]

Accept if:
\[ 2 = z_0 \text{ or } 2 = z_1 \]

Can't be reused!

6. Multi-Use Circuits

- Use fully-homomorphic encryption (FE) to make circuits reusable
- For input \( x \), Alice gives out \( FE_K(G(x)) \)
  - Chooses a new key \( K \) for FE
  - Encrypts the wire values \( G(x) \) representing \( x \)
- Worker uses homomorphism to evaluate \( G(C) \) and obtain an encryption of the output: \( FE_K(G(y)) \)

- Achieves Correctness, Security, & (theoretically) Efficiency
- Also provides Input Privacy