Overview

• Describe a system for secure 2-party computation using garbled circuits that is much more *scalable* and significantly *faster* than best prior work

• Applications:
  – *Face recognition*: Hamming distance
  – *Genomics*: Edit distance, Smith-Waterman
  – *Private encryption*: Oblivious AES evaluation
program Millionaires {
    type int = Int<4>; // 4-bit integer
    type AliceInput = int;
    type BobInput = int;
    type AliceOutput = Boolean;
    type BobOutput = Boolean;
    type Output = struct {
        AliceOutput alice, BobOutput bob;
    };
    type Input = struct {
        AliceInput alice, BobInput bob;
    };
    function Output out(Input inp) {
        out.alice = inp.alice > inp.bob;
        out.bob = inp.bob > inp.alice;
    }
}
Problems?

An alternative approach ... would have been to apply Yao’s generic secure two-party protocol.... This would have required expressing the algorithm as a circuit ... and then sending and computing that circuit.... [We] believe that the performance of our protocols is significantly better than that of applying generic protocols.

Margarita Osadchy, Benny Pinkas, Ayman Jarrous, Boaz Moskovich.


---

[Generic SFE] is very fast ... but the circuit size is extremely large.... Our prototype circuit compiler can compile circuits for problems of size (200, 200) but uses almost 2 GB of memory to do so.... larger circuits would be constrained by available memory for constructing their garbled versions.

Somesh Jha, Louis Kruger, Vitaly Shmatikov.

The Fallacy

program Millionaires {
    type int = Int<4>; // 4-bit integer
    type AliceInput = int;
    type BobInput = int;
    type AliceOutput = Boolean;
    type BobOutput = Boolean;
    type Output = struct {
        AliceOutput alice, BobOutput bob;
    };
    type Input = struct {
        AliceInput alice, BobInput bob;
    };
    function Output out(Input inp) {
        out.alice = inp.alice > inp.bob;
        out.bob = inp.bob > inp.alice;
    }
}
Faster Garbled Circuits

Gates can be evaluated as they are generated: **pipelining**
Benefits of *Pipelining*

- Allows GC to scale to circuits of arbitrary size
  
  We ran circuits with over a billion gates, at a rate of roughly 10 µs per gate.

- Improves the time efficiency
Problems in Existing (SFDL) Compilers

Resource-demanding SFDL compilation

It takes hours on a 40GB memory server to compile a SFDL program that implements AES.

Many optimization opportunities are missed

Circuit level
Minimize bitwidth
Reduce the number of non-free gates

Program level
Treat public and secret values differently
## Some Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Best Previous Result</th>
<th>Our Result</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hamming Distance</strong> (Face Recognition, Genetic Dating) – two 900-bit vectors</td>
<td>213s [SCiFI, 2010]</td>
<td>0.051s</td>
<td>4176x</td>
</tr>
<tr>
<td><strong>Levenshtein Distance</strong> (genome, text comparison) – two 200-character inputs</td>
<td>534s [Jha+, 2008]</td>
<td>18.4s</td>
<td>29x</td>
</tr>
<tr>
<td><strong>Smith-Waterman</strong> (genome alignment) – two 60-nucleotide sequences</td>
<td>[Not Implementable]</td>
<td>447s</td>
<td>-</td>
</tr>
<tr>
<td><strong>AES Encryption</strong></td>
<td>3.3s [Henecka, 2010]</td>
<td>0.2s</td>
<td>16.5x</td>
</tr>
</tbody>
</table>

**Scalable:** 1 billion gates evaluated at ≈100,000 gates/second on regular PCs

Comparisons are aligned to the same security level in the semi-honest model.
Our Results

Scalability

Performance

Fairplay [PSSW09]  TASTY  Here

max gates

non-free gates/s

Billions

0 0.2 0.4 0.6 0.8 1 1.2

Fairplay  [PSSW09]  TASTY  Here

x 10000

0 2 4 6 8 10

Fairplay  [PSSW09]  TASTY  Here
## Timing Results

<table>
<thead>
<tr>
<th>Time</th>
<th>Results</th>
<th>Best previous</th>
<th>Here</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Hamming distance (900 bits)</td>
<td>4176x faster</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Hamming distance (900 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>Hamming distance (900 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>Hamming distance (900 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>Hamming distance (900 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>Hamming distance (900 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>Hamming distance (900 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>edit distance (200 256-bit chars)</td>
<td>29x faster</td>
<td></td>
</tr>
</tbody>
</table>

References:

[SCiFI, 2010]

[Jha+, 2008]
Conclusion

- **Pipelining** enables garbled-circuit technique to scale to large problem sizes
- **Circuit-level optimizations** can dramatically reduce performance overhead

Privacy-preserving applications can run orders of magnitude faster than previously thought.
Thanks!

Questions?

Download framework and Android demo application from MightBeEvil.com
Secure Two-Party Computation

Bob’s Genome: ACTG...
Markers (≈1000): [0, 1, ..., 0]

Alice’s Genome: ACTG...
Markers (≈1000): [0, 0, ..., 1]

$x = f(g_A, g_B)$

Can Alice and Bob compute a function of their private data, without exposing anything about their data besides the result?
Secure Function Evaluation

Alice (circuit generator)

Holds $a \in \{0,1\}^s$

Agree on $f(a, b) \to x$

Bob (circuit evaluator)

Holds $b \in \{0,1\}^t$

Garbled Circuit Protocol

Outputs $x = f(a, b)$
without revealing $a$
to Bob or $b$ to Alice.

Andrew Yao, 1986
## Yao’s Garbled Circuits

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**AND**

\[ a \quad \quad \quad \quad \quad \quad b \quad \quad \quad \quad \quad \quad x \]
Computing with Meaningless Values?

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_0$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
</tbody>
</table>

$a_i$, $b_i$, $x_i$ are random values, chosen by the circuit generator but meaningless to the circuit evaluator.
Computing with Garbled Tables

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_0$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
</tbody>
</table>

$a_i, b_i, x_i$ are random values, chosen by the circuit generator but meaningless to the circuit evaluator.

Bob can only decrypt one of these!

Garbled And Gate

- $Enc_{a_0,b_1}(x_0)$
- $Enc_{a_1,b_1}(x_1)$
- $Enc_{a_1,b_0}(x_0)$
- $Enc_{a_0,b_0}(x_0)$

Randomly permute
Chaining Garbled Circuits

Can do *any* computation privately this way!
Threat Model

Semi-Honest *(Honest-but-Curious)* Adversary

Adversary follows the protocol as specified (!), but tries to learn more from the protocol execution transcript

May be good enough for some scenarios

We are working on efficient solutions for malicious adversaries
for (int i = 1; i < a.length; ++i)  
    for (int j = 1; j < b.length; ++j) {
        T = (a[i] == b[j]) ? 0 : 1;
        D[i][j] = min(D[i-1][j]+1, D[i][j-1]+1, 
                      D[i-1][j-1] + T);
    }
Circuit Optimization – Edit Distance

\[ D[i][j] \]

AddOneBit

AddOneBit

2-Min

T

AddOneBit

2-Min

D[i-1][j]

D[i][j-1]

D[i-1][j-1]
Circuit Optimization – Edit Distance

\[ D[i-1][j] \quad D[i][j-1] \quad D[i-1][j-1] \]

2-Min

AddOneBit

1

AddOneBit

2-Min

\[ D[i][j] \]
Circuit Optimization – Edit Distance

\[ D[i-1][j] \quad D[i][j-1] \quad D[i-1][j-1] \]

- 2-Min
- AddOneBit
- Mux
- T

Saves about 28% of gates
Through custom circuit design and the use of optimal circuit components, we strive to minimize the number of *non-free* gates.

Ease of Use

• Our framework assumes no expert knowledge of cryptography

• Need basic ideas of Boolean circuits

• Circuit designs converted directly to Java programs
Use the Framework

Traditional Java Application

Critical Component

Critical Component

Critical Component

Rest of the Java Program

javac

Circuit Generator

Circuit Evaluator

Java code

Library Circuit

Custom Circuit

Library Circuit

Use the Framework
Leveraging an existing ASIC design for AES allows us to reduce the state-of-the-art AES circuit by 30% of non-free gates, compared to [PSSW09] and [HKSSW10].

Example: AES SBox