Practical Secure Function Evaluation by Logic Synthesis and Optimization: Challenges and Opportunities

Farinaz Koushanfar
Professor and Henry Booker Faculty ECE Dept.,
University of California, San Diego
What is Secure Function Evaluation?
DNA report: Alice wants to learn about her DNA report without revealing her DNA to Bob. Bob also doesn’t want to share his genetic database or model.

How to achieve the DNA report w/o a third party while keeping the information private?
Goal: Compute function $F()$ on private inputs $X$ and $Y$

Our focus:

- 2-party computation based on (Yao’s) Garbled Circuits (GC) Protocol [Yao1986]
- Circuit generation and garbling as a primitive (inline with JustGarbled [])
Theoretical SFE Protocols

- **Circuit-Based protocols**
  - Operates on Boolean representation of function
  - Yao’s Garbled Circuit (GC) Protocol
  - Goldreich-Micali-Wigderson (GMW) Protocol

Example:
Yao’s Millionaires problem circuit
3-bit comparison

Yao’s Protocol: Garbled Circuits (GC)

Alice (Circuit Generator)

Bob (Circuit evaluator)

\[ X = (110)_2 \]

\[ Y = (100)_2 \]

\[ X = (110)_2 \]

\[ Y = (100)_2 \]

Generate Logic Circuit C

<table>
<thead>
<tr>
<th>Garbled inputs ( x_i )</th>
<th>Garbled tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i = 0 )</td>
<td>( x_1^0 )</td>
</tr>
<tr>
<td></td>
<td>( y_1^0 )</td>
</tr>
<tr>
<td></td>
<td>( w_5^0 )</td>
</tr>
<tr>
<td>( x_i = 1 )</td>
<td>( x_1^1 )</td>
</tr>
<tr>
<td></td>
<td>( y_1^1 )</td>
</tr>
<tr>
<td></td>
<td>( w_5^0 )</td>
</tr>
</tbody>
</table>

e.g. \( F(X < Y) \)

Generate Garbled Circuit \( \tilde{C} \)
Adversary model

- Honest but curious
- Malicious
Can Yao’s GC Protocol be practical?
Custom high level procedural language
SFDL (Secure Function Definition Language)
compiled into a circuit description language,
SHDL (Secure Hardware Description Language)
Main Approach for GC

- **Compiler-based**
  - Compile high-level description of functionality to optimized circuits
  - e.g., FairPlay, TASTY, KSMB, etc.

- **Library-based**
  - custom-libraries with special functions for emitting Boolean circuits, built-in boolean circuits
  - e.g., FastGC, VMCrypt, etc.

- **Hardware-assisted**
  - GPU based, AES-NI
Shortcomings of prior approaches

- Manual circuit-level optimizations
- Combinational logic
  - Prevents synthesis of large control-intensive circuits (e.g., SHA3)
- Poor Scalability
  - Memory exhaustion
  - Circuit generation/evaluation time may exceed real-time constraints
  - Loops unrolling and subroutines inlining
  - High-level programming abstraction: circuits not compact or optimized
- Lack of practical utility
  - Users cannot comprehend the final circuit organization and therefore cannot apply finer circuit optimizations
- Only moderate size circuits are handled
  - Some circuit sizes not feasible for embedded devices
TinyGarble\textsuperscript{[1]}:

Generating super compact and scalable circuits by

- Sequential logic description for functionality
- Introducing new transforms/libraries to enable adapting classic HW synthesis techniques
- Improving best reported results by several orders of magnitude
- Enabling implementation of circuits never reported before

\textsuperscript{[1]} Songhori (Koushanfar) et al., IEEE S&P ‘15
Garbled Circuit (GC) optimizations

- **Row-reduction**
  - Reduce size of garbled truth table for non-XOR gates by 25% \([1]\)

- **Free-XOR**
  - No garbled truth table for XOR gate needed\([2][3]\)

- **Garbling with fixed-key block cipher**
  - No additional keys for gate output (unique tweak T per gate)\([4]\)

- **Execution optimization**
  - Fast table lookups, pipelining\([5][6]\)

---

\([1]\) Naor et al. ACMES‘99
\([2]\) Kolesnikov et al. ICALP‘08
\([3]\) Kolesnikov et al. Crypto‘14
\([4]\) Bellare et al. S&P‘13
\([5]\) Järvinen et al. CHES‘10
\([6]\) Haung et al. USENIX‘11
TinyGarble: Superfolding Garbled Circuits
Sequential vs. Combinational Circuits

- **Combinational (Boolean) circuit**: outputs are only functions of inputs

\[ X, Y := F(A, B, C) \]

- **Sequential circuit**: outputs are functions of both inputs and circuit states kept in memory elements, e.g., Flip Flops (FF)

\[ X, Y := F(A, B, FF \text{ state}) \]
Simple Example: N-bit Binary Adder

Combinational n-bit adder: 1 HA and (n-2) FAs → n-dependent

Sequential n-bit adder: 1 FA and 1 FF (feedback to re-use same FA)

- Sequential circuit size independent of input size (n)
- More compact and scalable than combinational design
Nearest Neighbor Search (NNS): Mapped perfectly to sequential circuit

FF keeps index and value of closest element to x at every clock cycle

size = \( O(1) \) independent of database size

compact and scalable circuit representation
Global Flow

- High Level Synthesis
  - (*.c/*.cpp)
  - (*.vhdl/*.v)

- HDL Logic Synthesis
  - User synthesis constraints
  - netlist (*.v)

- Scheduler
  - Simple Circuit Description (*.scd)

- Garbling

- Evaluation

Superfolding Customized Synthesis Library

Code Snippet
```
#include <ap_int.h>

module hamming(Clk, rst, a, b, h);

module hamming(Clk, rst, a, b, h):
  input clk, rst, a, b;
  output [15:0] h;

  DFF c_reg[0] (.D(n3), .CLK(clk), .RST(rst), .Q(h[0]));

  XNOR U1 (.A(a), .B(b), .Z(n2));
  NANDN U3 (.A(n2), .B(n3), .Z(n3));

endmodule
```
✓ Offline synthesis → exploits established logic optimization techniques, as opposed to online generation, garbling and evaluation.
Towards Building a Universal Secure Processor\textsuperscript{[1][2][3]}...
Private Function Evaluation PFE-SFE

- Alice has a *private function*\(^A(.)\)
- Bob has \(data_B\)
- Bob wants to learn *private function*\(^A(data_B)\) without telling Alice what \(data_B\) is
- Alice does not want Bob to know *function*\(^A(.)\)

[1] Gentry et al., CRYPTO’10
MIPS for PFE-SFE

- MIPS is a RISC μP
- Low overhead: relatively small # of gates (~13K)

The first scalable implementation of a PFE-SFE on a real processor architecture 😊

*Plasma project in opencoures.org
Can we bridge the PF-SFE / SFE gap?
Efficient and optimized frameworks are too difficult to work for non-expert users.
  - Requires logic circuit design knowledge

High-level GC languages and compiler are extremely inefficient.
  - Compared to a hand and logic synthesis optimized circuits

Bridge the gap

Between efficient-and-hard and inefficient-and-easy GC framework
Previous Work and Challenge

- Garbling a processor [TinyGarble S&P’15]
  - To solve Private Function SFE (PF-SFE), where along with input data, the function is also private.
  - Programmed with high-level languages and conventional compilers

- How about using it for SFE
- Challenges
  - Extremely costly since hides the function (PF-SFE)
Bridge the gap with GarbledCPU

- Support for SFE without paying PF-SFE cost
  - Performance-privacy trade-off
    - By relaxing privacy and improving performance
  - Supports private, semi-private, or public functions
- While enjoying simplicity of programming a processor
- Hardware implementation of GC protocol
  - Increase performance

[Songhori et al., GarbledCPU, DAC’16]
Talk this Wednesday 1:30-3pm Session
Overview of GarbledCPU
Evaluations of TinyGarble
Evaluation: Benchmark Functions

- Combinational circuit implementations
- Sequential circuit implementations
  - Compared with equivalent combinational circuit
  - Compared with sequential circuit implementations of different circuit folding
  - Compared with reference circuit implementations in other works
- Report functions not implementable earlier
  - e.g. SHA3 function
Measuring Performance

Circuit Size Efficiency (CSE):

\[
CSE = \frac{\text{size of ref circuit}}{\text{Size of TinyGarbled}} = \frac{CS_0}{CS} \times 100
\]

Garbling Time: number of permutation function calls for non-XOR gates (PFC)

PFC = \(4 \times (\#\text{non-XOR}) \times c\) \approx \text{estimated garbling (communication and computation) time}

Permutation function \(\pi\) called 4 times for each non-XOR gate

\(c\) : number of sequential cycles

\(c = 1\) for combinational circuits

PFC\(_0\) = PFC for reference circuit

\[
\frac{PFC - PFC_0}{PFC_0} \times 100 = \text{PFD}
\]

**negative** PFD indicates reduced circuit garbling time in comparison to reference circuit
## Evaluation of TinyGarble combinational circuits compared with [1]

<table>
<thead>
<tr>
<th>Function</th>
<th>CSE</th>
<th>PFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>16384-bit Compare</td>
<td>1.49</td>
<td>-49%</td>
</tr>
<tr>
<td>160-bit Hamming</td>
<td>3.55</td>
<td>-58%</td>
</tr>
<tr>
<td>128-bit Sum</td>
<td>2.28</td>
<td>-63%</td>
</tr>
<tr>
<td>256-bit Sum</td>
<td>2.32</td>
<td>-65%</td>
</tr>
<tr>
<td>1024-bit Sum</td>
<td>2.35</td>
<td>-66%</td>
</tr>
<tr>
<td>64-bit Mult</td>
<td>9.26</td>
<td>-84%</td>
</tr>
<tr>
<td>128-bit Mult</td>
<td>8.88</td>
<td>-84%</td>
</tr>
<tr>
<td>256-bit Mult</td>
<td>7.30</td>
<td>-60%</td>
</tr>
</tbody>
</table>

Relative improvement in Memory footprint

% Reduction in # of non-XORs: Communicated labels (BW)

Our combinational circuit size for 64-bit multiplication is **9.26 times smaller** than results reported in KSMB

Garbling time is reduced by **84%** for the same circuit as opposed to KSMB [due to reduced number of non-XOR gates]

[1] Kreuter et al., USENIX’13
TinyGarble memory footprint (CSE): Evaluation of 64-bit multiplication

- **64-bit Mult. Sequential (this work)**: $c = 16$, CSE = 15.5
- **64-bit Mult. Combinational (this work)**: $c = 1$, CSE = 15.5
- **64-bit Mult. (KSMB)**: CSE$^{KSMB} = 143.5$

Total Number of Gates

- Non-XOR Gates
- XOR Gates
### TinyGarble evaluation

**Metrics:**

- Relative improvement in Memory footprint (CSE)
- % Reduction in # of non-XORs: Communicated labels (PFD)

<table>
<thead>
<tr>
<th>Multi.</th>
<th>64-bit</th>
<th>128-bit</th>
<th>( I_{PCF} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total gates</td>
<td>358</td>
<td>11439</td>
<td>105880</td>
</tr>
<tr>
<td>Non-XOR</td>
<td>122</td>
<td>3925</td>
<td>24766</td>
</tr>
<tr>
<td>CS (B)</td>
<td>8592</td>
<td>274536</td>
<td>2541120</td>
</tr>
<tr>
<td>PFC</td>
<td>31232</td>
<td>15700</td>
<td>99064</td>
</tr>
<tr>
<td>CSE</td>
<td>32.0</td>
<td>15.5</td>
<td>1.0</td>
</tr>
<tr>
<td>PFD</td>
<td>99%</td>
<td>27%</td>
<td>0%</td>
</tr>
</tbody>
</table>

### Performance compared to reference circuit from

[Kreuter et al., USENIX 2013]

Evaluation of benchmark functions, e.g., sum, Hamming, RSA, and compare show a similar improvement in memory footprint (several orders of magnitude), and in number of communicated labels (3-4 times).
TinyGarble memory footprint: SHA3

Total Number of Gates

- 1600-bit SHA3 (c = 24)
- 1600-bit SHA3 (c = 12)
- 1600-bit SHA3 (c = 6)
- 1600-bit SHA3 (c = 1)

Legend:
- Non XOR Gates
- XOR Gates
TinyGarble cache effect

- Garbling Time
- CS

CPU cycle (×10⁶) vs. Sequential Cycle (c)
PF-SFE results: MIPS

- Lite MIPS VI with support of simple instructions plus \textit{mult} and \textit{shift} and \textit{256B instruction ROM} and \textit{256B Data RAM}.
  - # of non-XOR = 3,536
  - # of XOR = 526

- Hamming Distance Benchmark:
  - Distance of two arrays with the length of \( l \) (\( A[l] \) and \( B[l] \))
  - # of instruction: \( m = 9 + 9*l \)
  - For example when \( l = 32 \):
    - Total \( \pi \) function calls: \( m \times \text{# of non-XOR} = 1M \)
    - Total network communication: \( m \times \text{# of non-XOR} \times \text{sizeof(garbled table)} = 50MB \)
Spectrum of exciting applications!
Secure K-NN\textsuperscript{[1]}

- The first efficient, scalable, and practical privacy-preserving k-nearest neighbors (k-nn) search
- None of the parties reveal their information while they can still cooperatively find the nearest matches
- The circuits small enough to fit within an embedded processor
- Garbled search for $n=128$, and $k=8$ within a few seconds

\textsuperscript{[1]} Songhori (Koushanfar) et al., DAC 2015
Privacy-preserving fingerprint authentication [1]

- The first efficient, reliable and provably secure two-party privacy-preserving fingerprint matching
- Adopted the NIST standardized Bozorth algorithm so it’s amenable to GC optimizations
- Devised a new protocol
- Results show the ability to authenticate a garbled fingerprint within a fraction of a second
  - No loss of accuracy compared to original Bozorth

[1] Zhang and Koushanfar, HOST’16
Privacy-preserving genome matching [1]

- Human Leukocyte Antigen (HLA) analysis which is a crucial test in organ transplantation
- Patient holds her whole genome sequence
- First scalable and efficient solution for secure organ transplantation compatibility testing
- Designing sub-linear size circuit for HLA compatibility testing
- Testing can be done within a few seconds on an embedded processor

[1] Riazi (Koushanfar) et al., HOST’16
Secure automotive location discovery [1]

- The car Q is lost due to unavailability or malfunction of GPS, e.g., military settings
- It sends request to three nearby cars A, B, and C for assistance in computing its location
- The three assisting cars then engage in a privacy-preserving localization protocol
- A new library with new functions required to generate GC optimized netlists
- Could locate ~0.5s on an embedded processor

[1 Hussain and Koushanfar, DAC’16
Talk on Tuesday, 1:30-3pm]
Summary

1. Minimizing computing/storage/comm cost of a broad class of iterative big/dense data analytics
   - To the limits of data structure and pertinent platform
   - Enables HW acceleration and stream processing
   - Benefits costly privacy-preserving computing

2. Novel scalable solutions for privacy preserving computing by Yao's Garbled Circuit (GC)
   - Enables addressing classical challenges and new apps

Evaluations show great efficiency compared with prior art, often by orders of magnitude!
Selected refs (related to logic optimization)


Thank You!
Questions?

farinaz@ucsd.edu