Correct Execution Environment: Hardware-Assisted Verifiable Computation

Junghee Lee, Korea University
(j_lee@korea.ac.kr)
Your Speaker

• **Education**
  – Ph.D. Georgia Institute of Technology (2013)
  – B.S. Seoul National University (2000)

• **Appointments**
  – Assistant/Associate Professor
    Korea University (2019-Present)
  – Assistant Professor
    University of Texas at San Antonio (2014-2019)
  – Engineer
    Samsung Electronics (2003-2008)

• **Research area**
  – Hardware security (processor, memory, non-volatile memory, storage, dedicated hardware)
Verifiable Computation

Server

\[ f(x) \]

\[ y = f(x) \]

Client

\[ y = f(x) \]
Cryptographic VC

\[ \text{EK, VK} \leftarrow \text{KeyGen}(1^\lambda) \]

Prover

\[ y, \pi \leftarrow \text{compute(} \text{EK, F, x} \text{)} \]

Verifier

\[ \text{verify(} \text{VK, F, x, y, } \pi \text{)} == 1 \]
Hardware-Assisted VC

\[ y^P = f^R(x^P) \]

\[ x^V = x^P \]
\[ y^V = y^P \]
\[ f^I = f^R \]
Contents

• Introduction
• Background and Motivation
• Correct Execution Environment
• Evaluation
• Conclusions
VC Construction

• For $y = f(x)$
• Three algorithms
  – $(E_K, V_K) \leftarrow \text{KeyGen}(1^\lambda)$
  – $(y, \pi) \leftarrow \text{Compute}(E_K, f, x)$
  – $\{0,1\} \leftarrow \text{Verify}(V_K, f, x, y, \pi)$
• Guaranteed properties
  – Completeness
  – Soundness
Cryptographic Approach

- Verifies every step of computation
  - Checks the hash of the previous state
  - Generates the proof of every instruction

- Extremely slow
  - 10,000 ~ 100,000 times slower
Trusted Hardware

• Trusted Execution Environment (TEE)
  – Hardware guarantees the correct execution of a protected application by isolation and attestation

• TEE for VC
  – If the hardware guarantees correct execution, we do not have to verify every step

• Formality
  – We need to define what exactly the hardware guarantees and what should be included to the proof
Contents

• Introduction
• Background and Motivation
• Correct Execution Environment
• Evaluation
• Conclusions
Correct Execution Environment

\[ S_0^R = S_0^I \]

By proof

\[ S_1^R = T^R(S_0^R) \]

\[ S_1^I = T^I(S_0^I) \]

\[ S_2^R = T^R(S_1^R) \]

\[ S_2^I = T^I(S_1^I) \]

... \[ T^R = T^I \]

States are preserved

By TEE

\[ S_n^R = T^R(S_{n-1}^R) \]

\[ S_n^I = T^I(S_{n-1}^I) \]
State Preservation

$S_0^R$

$S_1^R = T^R(S_0^R)$

$S_1^R$

$S_2^R = T^R(S_1^R)$

$S_2^R$

...

$S_n^R = T^R(S_{n-1}^R)$

$S_n^R$

Preventing memory access to physical pages used by the protected application

$\Rightarrow$ OS services cannot be used

A shared memory region may be allowed

$\Rightarrow$ Its integrity should be managed by the developer
CEE VC Construction

• Digital signature scheme
  – (SK, PK) \xleftarrow{} Gen(1^\lambda)
  – σ \xleftarrow{} Sig(m, SK)
  – \{0,1\} \xleftarrow{} Ver(m, σ, PK)

• CEE VC construction
  – KeyGen
    • (EK, VK) \xleftarrow{} Gen
  – Compute
    • π \xleftarrow{} Sig(S_0||S_n, EK)
  – Verify
    • \{0,1\} \xleftarrow{} Ver(S_0||S_n, π, VK)
CEE VC Scheme

(S1) Random number
(S2) Public key
(S3) Initial state
(S4) Write input and code to the memory regions
(S5) Initialize hardware
(S6) Start
(S6) Start
(S7) Finish
(S7) Finish
(S8) Read signature
(S9) Send signature
Contents

• Introduction
• Background and Motivation
• Correct Execution Environment
• Evaluation
• Conclusions
Prototype

• By modifying AMBER processor
  – ARM-compatible open-source processor written in Verilog

• Tools
  – Xilinx ISE Verilog simulator
  – Synopsys Design Compiler
# MiBench Suite

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Executed</th>
<th>Program size</th>
<th>Input size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADPCM</td>
<td>121,672</td>
<td>5,116 B</td>
<td>3,072 B</td>
</tr>
<tr>
<td>BitCount</td>
<td>115,895</td>
<td>4,828 B</td>
<td>292 B</td>
</tr>
<tr>
<td>BlowFish</td>
<td>372,860</td>
<td>4,820 B</td>
<td>4,258 B</td>
</tr>
<tr>
<td>CRC32</td>
<td>137,460</td>
<td>4,576 B</td>
<td>1,136 B</td>
</tr>
<tr>
<td>QuickSort</td>
<td>126,712</td>
<td>4,320 B</td>
<td>528 B</td>
</tr>
<tr>
<td>SHA</td>
<td>239,833</td>
<td>5,968 B</td>
<td>672 B</td>
</tr>
<tr>
<td>StringSearch</td>
<td>167,549</td>
<td>4,444 B</td>
<td>2,852 B</td>
</tr>
</tbody>
</table>
# Prover Overhead

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Original</th>
<th>Proposed</th>
<th>No-limit&lt;sup&gt;1)&lt;/sup&gt;</th>
<th>Limit&lt;sup&gt;2)&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADPCM</td>
<td>1.82 ms</td>
<td>2.01 ms</td>
<td>885.50 h</td>
<td>3.00 h</td>
</tr>
<tr>
<td>BitCount</td>
<td>1.99 ms</td>
<td>2.06 ms</td>
<td>843.45 h</td>
<td>2.86 h</td>
</tr>
<tr>
<td>BlowFish</td>
<td>5.41 ms</td>
<td>5.68 ms</td>
<td>2,713.59 h</td>
<td>N/A</td>
</tr>
<tr>
<td>CRC32</td>
<td>2.55 ms</td>
<td>2.67 ms</td>
<td>1,000.40 h</td>
<td>3.39 h</td>
</tr>
<tr>
<td>QuickSort</td>
<td>1.91 ms</td>
<td>1.98 ms</td>
<td>922.18 h</td>
<td>3.13 h</td>
</tr>
<tr>
<td>SHA</td>
<td>3.62 ms</td>
<td>3.75 ms</td>
<td>1,745.45 h</td>
<td>6.58 h</td>
</tr>
<tr>
<td>StringSearch</td>
<td>2.45 ms</td>
<td>2.60 ms</td>
<td>1,219.38 h</td>
<td>4.60 h</td>
</tr>
</tbody>
</table>

## Verifier Overhead

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Original</th>
<th>Proposed</th>
<th>No-limit&lt;sup&gt;1)&lt;/sup&gt;</th>
<th>Limit&lt;sup&gt;2)&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADPCM</td>
<td>1.82 ms</td>
<td>1.43 ms</td>
<td>59.69 ms</td>
<td>43.26 ms</td>
</tr>
<tr>
<td>BitCount</td>
<td>1.99 ms</td>
<td>1.56 ms</td>
<td>57.67 ms</td>
<td>41.19 ms</td>
</tr>
<tr>
<td>BlowFish</td>
<td>5.41 ms</td>
<td>1.52 ms</td>
<td>57.61 ms</td>
<td>N/A</td>
</tr>
<tr>
<td>CRC32</td>
<td>2.55 ms</td>
<td>1.73 ms</td>
<td>55.90 ms</td>
<td>44.14 ms</td>
</tr>
<tr>
<td>QuickSort</td>
<td>1.91 ms</td>
<td>1.24 ms</td>
<td>54.10 ms</td>
<td>41.37 ms</td>
</tr>
<tr>
<td>SHA</td>
<td>3.62 ms</td>
<td>1.30 ms</td>
<td>65.67 ms</td>
<td>41.48 ms</td>
</tr>
<tr>
<td>StringSearch</td>
<td>2.45 ms</td>
<td>1.35 ms</td>
<td>54.97 ms</td>
<td>43.09 ms</td>
</tr>
</tbody>
</table>


Hardware Cost
## Hardware Cost

<table>
<thead>
<tr>
<th>Component</th>
<th>Original</th>
<th>Modified</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register file</td>
<td>0.023 mm²</td>
<td>0.032 mm²</td>
<td>41.47 %</td>
</tr>
<tr>
<td>Execution stage</td>
<td>0.021 mm²</td>
<td>0.025 mm²</td>
<td>19.86 %</td>
</tr>
<tr>
<td>Cache controller</td>
<td>0.002 mm²</td>
<td>0.006 mm²</td>
<td>209.71 %</td>
</tr>
<tr>
<td>Signature</td>
<td>-</td>
<td>0.257 mm²</td>
<td>-</td>
</tr>
<tr>
<td>Interface</td>
<td>-</td>
<td>0.004 mm²</td>
<td>-</td>
</tr>
<tr>
<td>Unchanged</td>
<td>0.085 mm²</td>
<td>0.085 mm²</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>0.131 mm²</td>
<td>0.409 mm²</td>
<td>213.72 %</td>
</tr>
</tbody>
</table>
Conclusions

• A trusted hardware-based verifiable computation scheme is proposed.
• It offers order-of-magnitude shorter execution time compared to cryptographic approaches.
• The required properties for the hardware and security properties guaranteed by the hardware are formally defined.