Establishing Software Root of Trust Unconditionally
(or, a First Rest Stop on the Never-Ending Road to Provable Security)

Virgil D. Gligor
Maverick S.-L. Woo

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

NDSS 2019
San Diego, CA
February 27, 2019
Outline

I. What is it?
   - Definition & relationships
   - Unconditional solution

II. Why is it hard?
   - 3 Problems
   - RoT ≠ software-based, crypto attestation

III. How to do it?
   - randomized polynomials
     - k-independent (almost) universal hash families; and
     - space-time optimal in cWRAM; and
   - scalable optimal bounds

IV. Q & A

Full Paper is the CMU-CyLab TR 18-003
https://www.cylab.cmu.edu/_files/pdfs/tech_reports/CMUCyLab18003.pdf
I. What is it?
Device State:

- Content of processor registers (R) and (persistent) memories (M)

System State:

- Union of all device states at time T

Controlled by a Powerful Adversary

Don’t Care

Persistent malware is unknown
Root of Trust (RoT) Establishment

- **Verifier**
- **Bus System**
- **GPU**
- **Sys Mgt**
- **CPU**
- **M**
- **I/O**
- **P**
- **D**
- **NIC**
- **Memory**
- **Don’t Care**
- **USB controller**
- **Disk controller**

A Verifier initializes an untrusted system state to chosen content. Check ensures that the state has all & only chosen content & PC values.
Secure State: *RoT state* (chosen content) satisfies security predicate $P$

**Verifiable boot:**

- either boot code in a *secure state*
- or detect unknown content

Verifiable boot \(\Rightarrow\) Secure State \(\Rightarrow\) *RoT State*

Trusted Recovery \(\Rightarrow\) . . .

Access Control Models \(\Rightarrow\) . . .

. . .
Unconditional Solution*

- no Secrets, no Trusted HW Modules, no Bounds on Adversary’s Power

- need only
  
  - random bits
  
  - device specifications.

Importance?

- no dependencies on the unknown & unknowable
- a defender has a provable advantage over any adversary
- outlives technology advances.

*I know of no other unconditional solution to any software security problem
I. What is it?

II. Why is it hard?
1. space-time optimal $C_{m,t} \leq$ malware-free Device

- non-asymptotic bounds
- on Device Specs; e.g., ISA ++

(a realistic model of computation?)

Complexity theory?

- non-asymptotic bounds? Very few
- on Device Specs? None

e.g., Horner’s rule for polynomial evaluation uniquely optimal in infinite fields: $2d (\times, +)$
not optimal in finite fields,
not on any Device ISA++
1. space-time optimal $C_{m,t} \preceq$ malware-free Device

- non-asymptotic bounds
- on Device Specs
1. space-time optimal $C_{m,t} \leq$ malware-free Device

- non-asymptotic bounds
- on Device Specs
- adversary execution?

Complexity Theory?
- no help.
- how could it help?
  e.g., malware beats m-t bounds
  $\Rightarrow C_{\text{nonce}}(v)$ becomes *unpredictable*

Engineering Solution?
  e.g., see - segmented memory
1. space-time optimal $C_{m,t} \preceq$ malware-free Device

- non-asymptotic bounds
- on Device Specs
- adversary execution
1. space-time optimal $C_{m,t}$ $\leq$ malware-free Device
- non-asymptotic bounds
- on Device Specs
- adversary execution

Reduction is insufficient!
1. space-time optimal $C_{m,t}$ \leq malware-free Device $\checkmark$

- non-asymptotic bounds
- on Device Specs
- adversary execution

Reduction is insufficient!

Solution?

control flow integrity after $C_{\text{nonce}}$ ends

$\Rightarrow$

control flow integrity before $C_{\text{nonce}}$ starts!
1. space-time optimal $C_{m,t}$ \(\preceq\) malware-free Device ✔
- non-asymptotic bounds
- on Device Specs
- adversary execution

2. Verifiable Control Flow ✔

3. Two Devices, or more?
- sequential verification fails

nonce\textsubscript{j}  
\hline  
\textbf{Cm}\textsubscript{j},\textsubscript{t}\textsubscript{j}  
Device j  
\hline  

nonce\textsubscript{i}  
\textbf{Cm}\textsubscript{i},\textsubscript{t}\textsubscript{i}  
Device i  
\hline  

Device \textsubscript{i} corrupts verified Device j  

restores

\textbf{Cm}\textsubscript{i},\textsubscript{t}\textsubscript{i}  

\hline  

\hline  

\textbf{time gap}
- ordinary concurrency fails

\[ nonce_j \]

Device \( j \)

\[ C_{m,j,t_j} \]

ends early

Slow

Fast

\[ nonce_i \]

Device \( i \)

\[ C_{m,i,t_i} \]

for Device \( j \)

restores

for Device \( j \)
- ordinary concurrency fails

\[ \text{nonce}_j \]
\[ C_{m_j, t_j} \]
Device j

\[ \text{nonce}_i \]
\[ C_{m_i, t_i} \]
Device i

ends early

“verify”

Slow

Fast
- concurrent verification w/ scalable bounds

\[ \text{Device } j \]

\[ C_{m_j, t_j} \]

\[ \delta_{\text{start}} \]

\[ \text{Device } i \]

\[ C_{m_i, t_i} \]

\[ \delta_{\text{end}} \]

\[ \text{Device } i \]

\[ C_{m_i, t'_i} \]

\[ t_i < t'_i \]

\[ \text{nonce}_j \]

\[ \text{nonce}_i \]
Protocol Atomicity

- verifiable control flow
- concurrent transaction order and duration

Unpredictable result
- code composition
- scalable bounds

Code Optimality in Adversary Execution

Legend: ← dependency
Time-measurement security

Protocol Atomicity

- verifiable control flow
- concurrent transaction order and duration

unpredictable result

Code Optimality in Adversary Execution

- caches? TLB?
- clock jitter?
- multi-processor interference?
- remote proxy?

Software-based Attestation

has different goals
Protocol Atomicity

- verifiable control flow
- concurrent transaction order and duration

- unpredictable result
- code composition
- scalable bounds

Signatures/MACs based on secrets in HW

Cryptographic Attestation has different goals
I. What is it?

II. Why is it hard?

III. How to do it
Solution Overview

Randomized Polynomials

- k-independent uniform coefficients, independent of input x

- k-independent (almost) universal hash function family

- \((m, t)\)-optimal in the concrete Word Random Access Machine (cWRAM)

- optimal bounds \(m\) and \(t\) are scalable; e.g., no mandatory \(m \cdot t\) tradeoffs
Overview of the cWRAM ISA++

- **Constants:** \( w \)-bit word, up to 2 operands/instruction
  instructions execute in *unit time*

- **Memory:** \( M \) words
- **Processor registers** \( R \): GPRs, PC, PSW, Special Processor + Flag & I/O Registers
- **Addressing:** immediate, relative, direct, indirect
- **Architecture features:** caches, virtual memory, TLBs, pipelining, multi-core processors

- **ISA:** *all* (un)signed integer instructions
  - All Loads, Stores, Register transfers
  - All Unconditional & Conditional Branches, all branch types
    - *all predicates with 1 or 2 operands*
  - **Halt**
  - All **Computation** Instructions:
    - addition, subtraction, logic, \( \text{shift}_{r/l}(R_i, \alpha) \), \( \text{rotate}_{r/l}(R_i, \alpha) \), \ldots
    - *variable* \( \text{shift}_{r/l}(R_i, R_j) \), *variable* \( \text{rotate}_{r/l}(R_i, R_j) \), \ldots
    - multiplication (1 register output)\ldots
    - \( \text{mod} \) (aka., division-with-remainder) \ldots
\[ \{ r_0...r_{k-1}, x \} \xleftarrow{\$} \mathbb{Z}_p \]

\[
\begin{align*}
H_{r_0...r_{k-1}, x}(v) &= \sum_{i=0}^{\infty} (s_i \oplus v_i) \cdot x^i \pmod{p}, \\
s_i &= \sum_{j=0}^{k-1} r_j(i+1)^j \pmod{p} \\
d &= |v| - 1
\end{align*}
\]

k-independent almost universal hash function family

\[
C_{\text{nonce}}(v) = H_{r_0...r_{k-1}, x}(v) = H_{d,k,x}(v)
\]

m-t optimal bounds on cWRAM: \( m = k + 22, t = (6k - 4)6d \)

Scalable bounds: \( k \uparrow \Rightarrow m \uparrow, t \uparrow \) and \( d \uparrow \Rightarrow t \uparrow \)
Foundation

Theorem 1

Let \( w > 3 \), and \( p \) be a prime, \( 2 < p < 2^{w-1} \).

Horner’s rule for one-time honest evaluation of \( P_d(\cdot) \) in cWRAM

\[
P_d(\cdot) = \sum_{i=d}^{0} a_i \cdot x^i \pmod{p} = (\ldots (a_d \cdot x + a_{d-1}) \cdot x + \ldots + a_1) \cdot x + a_0 \pmod{p}
\]

is uniquely \((m, t)\)-optimal if the cWRAM execution space & time are simultaneously minimized; i.e., \( m = d+11 \), \( t = 6d \).

Answer to A. M. Ostrowski’s 1954 question:

“Is Horner’s rule optimal for polynomial evaluation?”

with non-asymptotic bounds in a realistic model of computation (cWRAM)
IV. Q & A
OK => malware-free Device $\rightarrow$ 2nd Pass w/ ordinary UHF
nonce₁ \rightarrow C_{m,t}\( \text{segment } M_1 \)

cnonce₁(v₁)\( \text{time}_1 \)

nonceᵢ \rightarrow C_{m,t}\( \text{segment } Mᵢ \)

cnonceᵢ(vᵢ)\( \text{time}_i \)

nonceᵦ \rightarrow C_{m,t}\( \text{segment } Mᵦ \)

cnonceᵦ(vᵦ)\( \text{time}_n \) \rightarrow \text{remote adversary}

\text{Local Verifier}

\text{random bits}

\text{Verifier’s random choice of segment i}
CPU 1
registers R1

CPUj
registers Rj

segment M₁
Cnonce₁ ↔ Cₘ,ₜ

... ...

segment M_j
Cnonce_j ↔ Cₘ,ₜ

... ...

segment Mₙ
Cnonceₙ ↔ Cₘ,ₜ

Memory M

nonce₁

Cnonce₁(v₁)
time₁

nonce_j

Cnonce_j(v_j)
time_j

nonceₙ

Cnonceₙ(vₙ)
timeₙ

round trip to remote adversary

Local Verifier

random bits

2/27/19
Implementation Notes
(Appendix C of CMU-CyLab TR 18-003)

Optimal Code: $\left( s_i \oplus v_i \right)$, loop control – simple on most real processors

Horner-rule step? (recall: $p$ is largest prime in $w$ bits)

- multiply
- mod $p$
- add
- mod $p$

- multiply
- div $p$
- multiply $p$ &
- subtract
- add
- div $p$
- multiply $p$ &
- subtract

$m, T$

$M, T$

$M, t$

different encodings => different results => SINGLE CHOICE!