

TIVA: Trusted Integrity Verification Architecture

Mahadevan Gomathisankaran & Akhilesh Tyagi

Electrical & Computer Engineering
Iowa State University

DRMTICS 2005

Outline

- 1 Problem Identification
- 2 Our Solution
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate
- 3 Analysis
 - Obfuscation Strength
 - Attack Scenarios
 - Flexibility
- 4 Related Work
- 5 Conclusion

Outline

- 1 Problem Identification
- 2 Our Solution
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate
- 3 Analysis
 - Obfuscation Strength
 - Attack Scenarios
 - Flexibility
- 4 Related Work
- 5 Conclusion

Outline

- 1 Problem Identification
- 2 Our Solution
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate
- 3 Analysis
 - Obfuscation Strength
 - Attack Scenarios
 - Flexibility
- 4 Related Work
- 5 Conclusion

Outline

- 1 Problem Identification
- 2 Our Solution
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate
- 3 Analysis
 - Obfuscation Strength
 - Attack Scenarios
 - Flexibility
- 4 Related Work
- 5 Conclusion

Outline

- 1 Problem Identification
- 2 Our Solution
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate
- 3 Analysis
 - Obfuscation Strength
 - Attack Scenarios
 - Flexibility
- 4 Related Work
- 5 Conclusion

Problem Identification

Pervasive Computing

- *Sensitive to Confidential* information
- Interact frequently with the insecure world

Integrity Verification

- As and when required
- With respect to predefined snapshot image

Reverse Engineering

- Determines the embedded IP
- Leads to software piracy

Problem Identification

Pervasive Computing

- *Sensitive to Confidential* information
- Interact frequently with the insecure world

Integrity Verification

- As and when required
- With respect to predefined snapshot image

Reverse Engineering

- Determines the embedded IP
- Leads to software piracy

Problem Identification

Pervasive Computing

- *Sensitive to Confidential* information
- Interact frequently with the insecure world

Integrity Verification

- As and when required
- With respect to predefined snapshot image

Reverse Engineering

- Determines the embedded IP
- Leads to software piracy

Problem Identification

Example

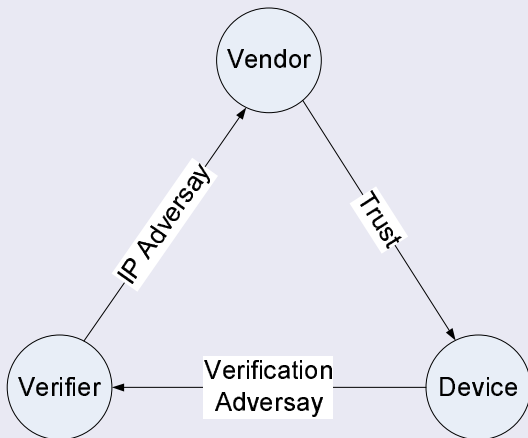
- Field officer verifying her GPS
- Executive verifying her PDA
- Organization verifying its routers

Three Dimensions

Distribution	\mathcal{D} distributes I to \mathcal{V} to verify \mathcal{E}
IP Protection	\mathcal{D} would like to ensure that IP of I is protected despite its distribution to \mathcal{V}
Verification	\mathcal{V} verifies I of \mathcal{E}

Problem Identification

Adversary Model



Proposed Solution

Basic Properties

- Image I & Time to verify \mathcal{T}

Possible Approaches

- Hash \Rightarrow **Replay, Spoofing**
- Variable Hash \Rightarrow **IP**
- Keyed Hash \Rightarrow **IP**
- No Shared Secret \Rightarrow **Spoofing**

Requirements

- Challenge-response based
- Shared secret between \mathcal{V} and \mathcal{E}
- Response as a function of challenge
- Fast and efficient

Proposed Solution

Basic Properties

- Image I & Time to verify \mathcal{T}

Possible Approaches

- Hash \Rightarrow **Replay, Spoofing**
- Variable Hash \Rightarrow **IP**
- Keyed Hash \Rightarrow **IP**
- No Shared Secret \Rightarrow **Spoofing**

Requirements

- Challenge-response based
- Shared secret between \mathcal{V} and \mathcal{E}
- Response as a function of **challenge**
- Fast and efficient

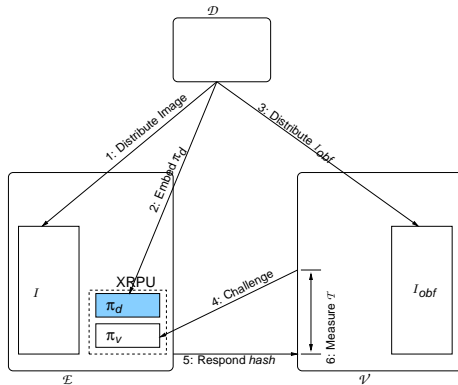
Outline

- 1 Problem Identification
- 2 **Our Solution**
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate
- 3 Analysis
 - Obfuscation Strength
 - Attack Scenarios
 - Flexibility
- 4 Related Work
- 5 Conclusion

Integrity Verification Architecture

Solution

- For every $(\mathcal{V}, \mathcal{E})$, \mathcal{D} generates π_d and gives $(\pi_d(I), \mathcal{T})$ to \mathcal{V}
- \mathcal{D} embeds π_d in \mathcal{E}
- \mathcal{V} generates π_v , finds $\mathcal{F}(\pi_v(\pi_d(I)))$ and challenges \mathcal{E}
- \mathcal{E} generates hash using π_v and π_d
- \mathcal{V} measures \mathcal{T}
- \mathcal{V} verifies signature



Integrity Verification Architecture

Characteristics

- Permutation of N values $\Rightarrow N! \approx 2^N \Rightarrow$
brute force
- Shared secret $\pi_d \Rightarrow$ spoofing
- Unique $\pi_v \Rightarrow$ replay
- $I_{obf} \Rightarrow$ IP

Outline

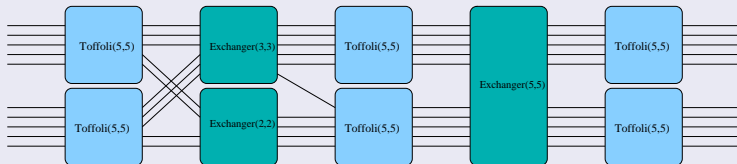
- 1 Problem Identification
- 2 Our Solution**
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit**
 - Area Delay Estimate
- 3 Analysis
 - Obfuscation Strength
 - Attack Scenarios
 - Flexibility
- 4 Related Work
- 5 Conclusion

Reconfigurable Permutation Unit

Permutation

- **Bijjective** or **reversible** function
- Composition of **reversible** gates
- **Toffoli** gates are reversible
- Should be reconfigurable

RPU

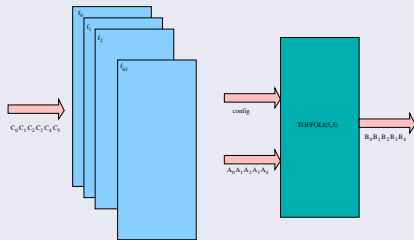


Reconfigurable Permutation Unit

Design

- Toffoli gates with LUT
- Configurations of LUT should have $T \cap C = \emptyset$
- Unique functions = $4 \binom{5}{1} + 3 \binom{5}{2} + 2 \binom{5}{3} + \binom{5}{4}$
- Permutation space (ignoring $|C| = 1$) = $(55)^6 \approx 2^{34}$
- Exchanger = *swap*
- Configuration selection \Rightarrow 39 bits

Config Selection

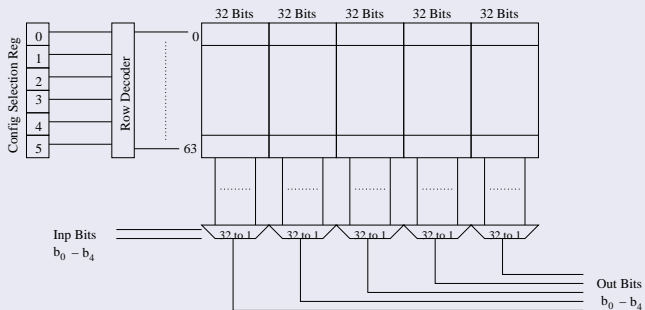


Outline

- 1 Problem Identification
- 2 Our Solution**
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate**
- 3 Analysis
 - Obfuscation Strength
 - Attack Scenarios
 - Flexibility
- 4 Related Work
- 5 Conclusion

Area Delay Estimate

LUT Schema



Area Delay Estimate

Estimates

- Area estimate using **CACTI**

Technology <i>nm</i>	Area <i>mm²</i>
180	1.4526
130	0.7578
70	0.2196

- Delay using **HSPICE**

$$T_{RPU} = 3 \times T_{32-to-1 \text{ MUX}} + 2 \times T_{2-to-1 \text{ MUX}}$$

Technology <i>nm</i>	T_{RPU} <i>ns</i>
180	1.140
70	0.780

Area Delay Estimate

XRPU

Processor	% Area Inc	Delay <i>ns</i>	Latency <i>cyc</i>
PXA 255	0.50	3.367	2
PXA 26x	0.86	3.367	2
PXA 27x	0.86	3.147	2
PXA 800F	0.53		
PPC 750FX	0.17		
PPC 750CXe	0.20	3.367	3

Outline

- 1 Problem Identification
- 2 Our Solution
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate
- 3 Analysis**
 - **Obfuscation Strength**
 - Attack Scenarios
 - Flexibility
- 4 Related Work
- 5 Conclusion

Obfuscation Strength

Quantification

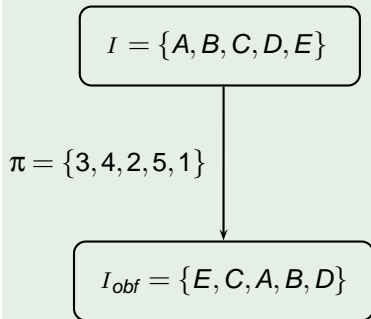
- **Measure** \Rightarrow How **dissimilar** the CFG is
- **Complete** \Rightarrow Perturbs **all nodes and edges** of CFG
- Basic blocks are the nodes of CFG
- Limited measure

$S_n^j \rightarrow$ Sequence of instructions i_1, i_2, \dots, i_n in I
from j^{th} position where $1 \leq j \leq (N - n + 1)$

$OS_n =$ % of S_n^j not in I_{obf}

Obfuscation Strength

Example



- $|I| = |I_{obf}| = N = 5$
- S_2^1 exists unobfuscated
- $OS_2 = 75\%$
- OS_n for $n > 2$ is 100%

Obfuscation Strength

Result

Table: Average *Obfuscation Strength* for 2^{20} runs

Seed	OS ₅	OS ₆	OS ₇	OS ₈	OS ₉	OS ₁₀
0x031245f8	94.74	95.96	96.80	97.35	97.83	98.29
0x7fc5a2d5	94.75	95.98	96.82	97.37	97.84	98.30
0x015e8f8c	94.73	95.97	96.80	97.36	97.84	98.29
0x00231eea	94.74	95.97	96.80	97.36	97.83	98.29
0x0153d22e	94.75	95.97	96.81	97.36	97.84	98.30

Outline

- 1 Problem Identification
- 2 Our Solution
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate
- 3 **Analysis**
 - Obfuscation Strength
 - **Attack Scenarios**
 - Flexibility
- 4 Related Work
- 5 Conclusion

Attack Scenarios

- Malicious \mathcal{E} is able to generate $(\mathcal{H}, \mathcal{T})$
- An impostor system generates $(\mathcal{H}, \mathcal{T})$
- Can $\pi_d \cdot \pi_v$ be obtained ?
- I_{obf} leakage \Rightarrow **spoofing** attack
- π_v resists **replay** attack
- π_d resists **IP** attacks

Outline

- 1 Problem Identification
- 2 Our Solution
 - Integrity Verification Architecture
 - Reconfigurable Permutation Unit
 - Area Delay Estimate
- 3 **Analysis**
 - Obfuscation Strength
 - Attack Scenarios
 - **Flexibility**
- 4 Related Work
- 5 Conclusion

Flexibility of TIVA

- Memory size ?
- 32 or 16 or 8 bit ?
- Von Neumann or Harvard ?

Related Work

SWATT

- Software only Attestation
- Probabilistic memory access
- $O(n \log n)$ memory accesses for memory size n
- Battery power ?
- **Spoofing** attack

Related Work

Genuinity

- Software only solution for autonomous verification of remote systems
- Applicable only to systems which expose architectural parameters like TLB miss counters, etc
- Architectural side-effects are **not** sufficient

IBM's IMA

- Requires TPM, OS support
- Integrity verification is done only at the loading point

Conclusion

- TIVA uses a hardware component to aid the verification
- TIVA is deterministic
- TIVA uses a shared secret between the embedded device and verifier to make simulating/emulating the device very difficult.
- TIVA uses permutation function to achieve both IP protection and randomness in hash generation function
- TIVA overhead is minimal 1% for area, 2 cycles for delay

Thank You!

Questions ?