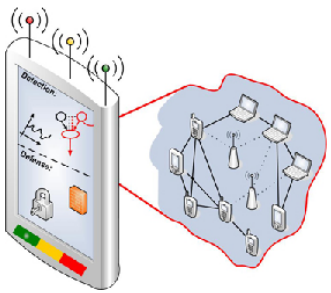
Code Attestation with Compressed Instruction Code

Benjamin Vetter (benjamin.vetter@haw-hamburg.de)
Prof. Dr. Dirk Westhoff (westhoff@informatik.haw-hamburg.de)

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik

Stand: 3. Mai 2011

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences



Gliederung

1 Einführung

2 Related Work
- Code Attestation
- On-The-Fly Decompression

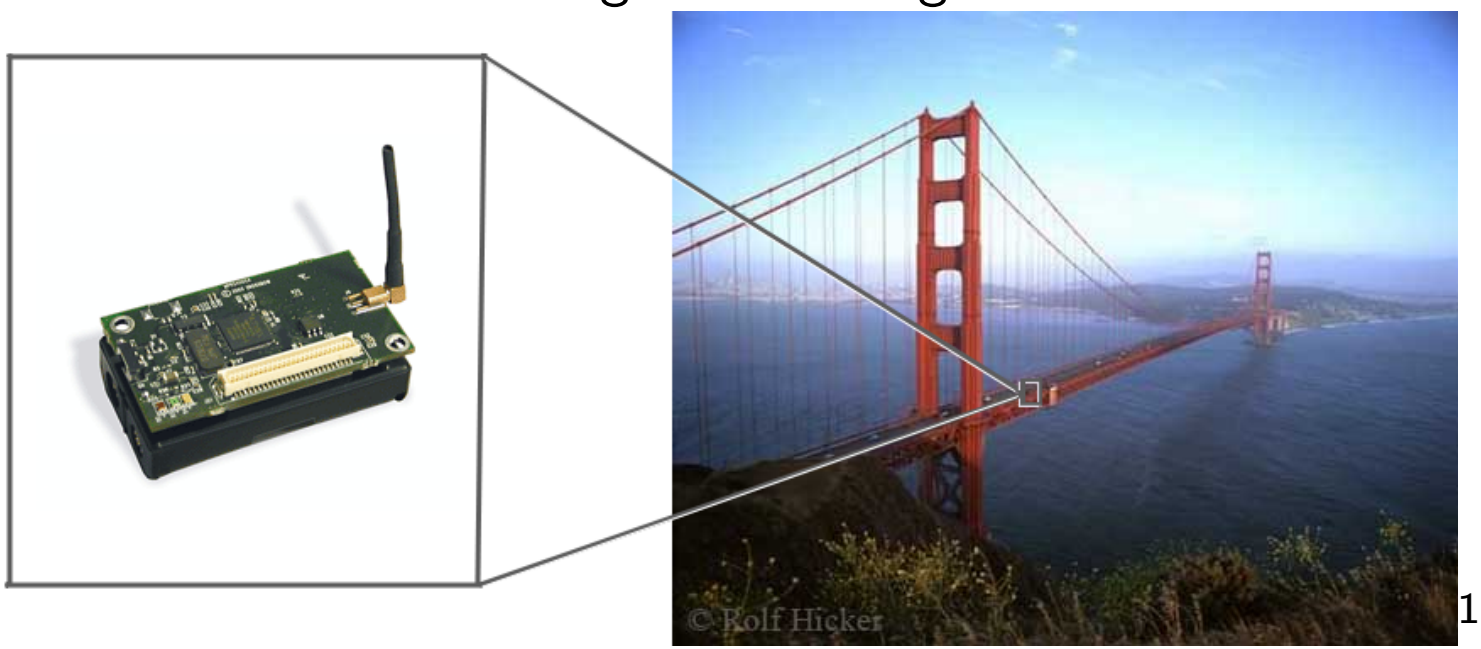
3 Einordnung und Abgrenzung

4 Zusammenfassung

5 Referenzen

Was wollen wir?

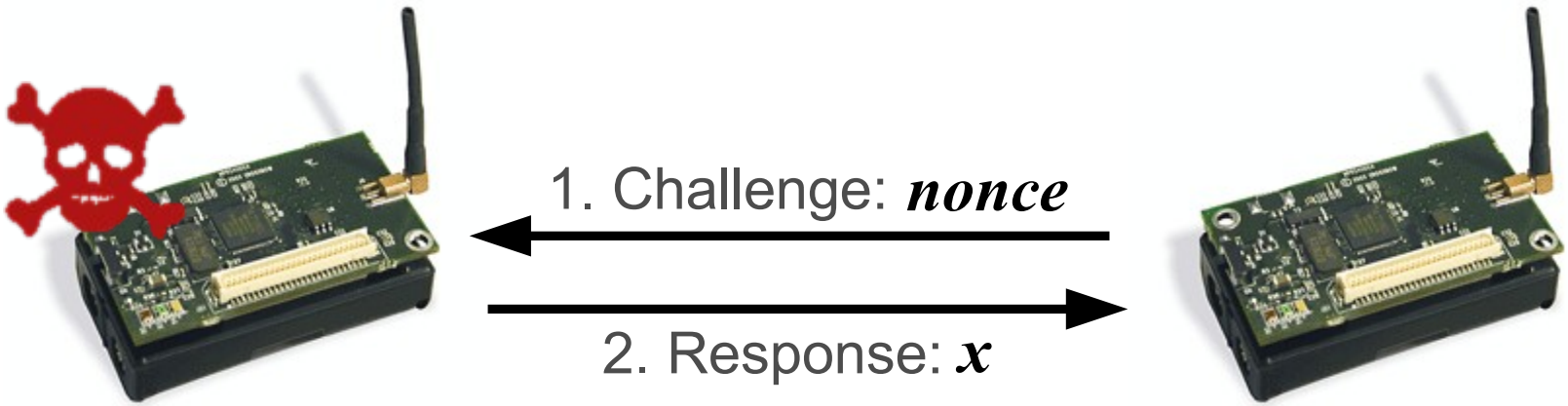
Wir wollen sicherstellen, dass ein Sensorknoten noch das korrekte, vertrauenswürdige Code Image ausführt.



[1]

[1] http://www.hicker.de/

Attestation mit Challenge Response Protokoll



Challenge Response Protokoll:

- Die Basis-Station schickt einem Knoten eine Challenge *nonce*
- Der Sensor berechnet als Response eine Checksum seines Program-Memories samt Code Image *CI*
- Korrekte Response $\Longrightarrow$ Der Knoten ist vertrauenswürdig
- Falsche Response $\Longrightarrow$ Der Knoten ist kompromittiert

Compression Attack

[CFPS09] auf der CCS '09:

- Kompression von CI mit verlustfreiem Kompressionsverfahren durch den Angreifer (z.B. CHE), d.h. $C(CI)$
- Im so gewonnenen Program Memory kann der Angreifer das Schadecode Image $\widetilde{CI}$ ablegen
- zur Attestation Zeit: On-the-fly Dekompression von $C(CI)$, d.h. $C^{-1}(C(CI))$
- Überlistet Code Attestation Protokolle, da x korrekt ist

Unser Lösungsvorschlag

Code Attestation with Compressed Instruction Code

Bereits komprimiertes CI, d.h. $C(CI)$, auf den Knoten laden, um zu verhindern, dass ein Angreifer CI wesentich komprimieren kann

$$x = h(nonce||C(CI)||...) \tag{1}$$

Unsere Kompression erhöht den Aufwand für einen Angreifer 'by orders of magnitude', so dass der Overhead leicht zu entdecken ist

Gliederung

1 Einführung

2 Related Work
- Code Attestation
- On-The-Fly Decompression

3 Einordnung und Abgrenzung

4 Zusammenfassung

5 Referenzen

Gliederung: Code Attestation Protokolle

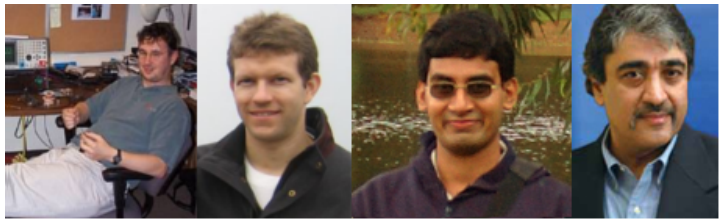
Gliederung: Code Attestation Protokolle

- SWATT [SPvDK04]
  - Pseudorandom Memory Traversal
  - Angriff auf SWATT [CFPS09]
- SCUBA [SLP+06]
  - Angriff auf SCUBA [CFPS09]
- Noise Filling [YWZC07]
  - Angriff auf Noise Filling [CFPS09]
- Attestation mittels Obfuskierung [SMKK05]

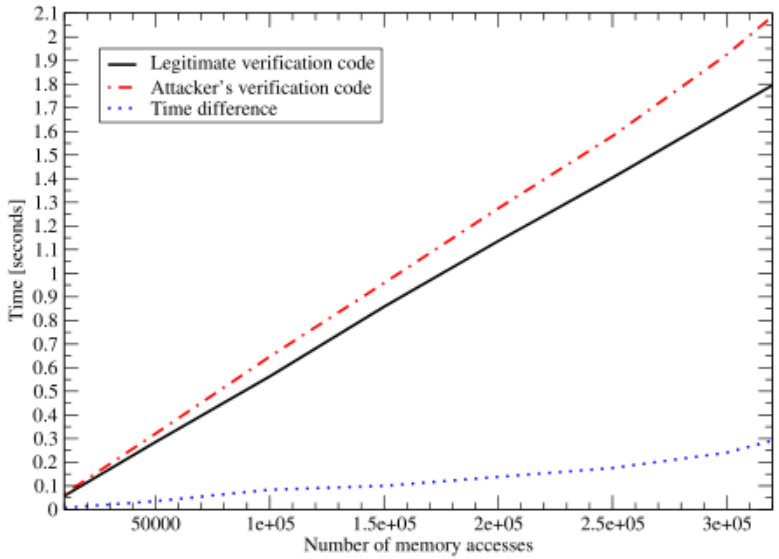
SWATT

SWATT: [SPvDK04]

- Der Angreifer muss bei der Berechnung von x für jede Speicherstelle überprüfen, ob er sie überschrieben hat
- Entspricht zusätzlichem *if*()
- Pseudrandom Memory Traversal: *nonce* als PRNG-Seed $\Longrightarrow$ zufällige Wahl des nächsten Bytes
- Die Response-Berechnung solange laufen lassen, bis der Angreifer durch den Aufwand der extra *if*()'s entdeckt ist



L. van Doorn A. Perrig A. Seshadri P. Khosla



Angriff auf SWATT

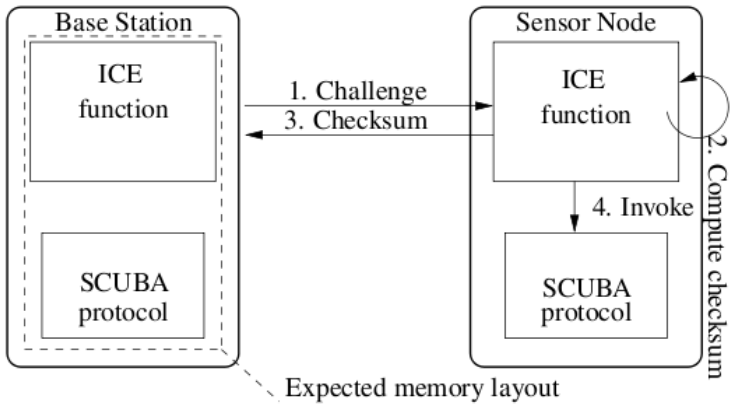
C. Castelluccia A. Francillion

Angriff auf SWATT: [CFPS09]

- 2 Weak-Assumptions in SWATT
  - SWATT hat den effizientesten Algorithmus um x zu berechnen
  - Für die Berechnung des Overheads der *if*()'s wurde vom effizentesten Algrithmus ausgegangen
- Hack: Eine oder beide Assumptions widerlegen
- Castelluccia und Francillion haben für *test-and-redirect* statt der angenommenen 3 cycles nur 2 cycles benötigt

$\Longrightarrow$ So enges, korrektes Timing ist grundsätzlich schwierig

SCUBA

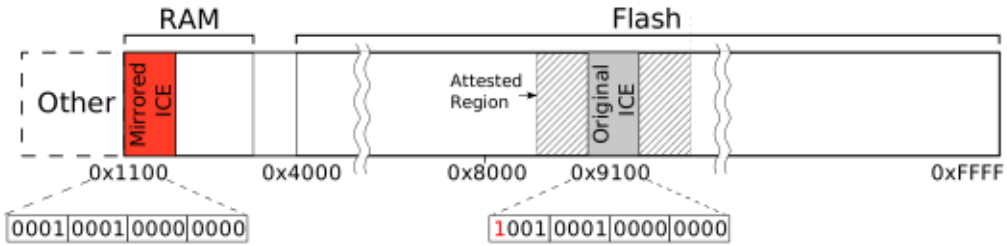


SCUBA: [SLP+06]

- Basiert auf ICE (Indisputable Code Execution) um einen Vertrauensanker zu bootstrappen
- ICE soll garantieren, dass die Attestation-Routine unmodifiziert ausgeführt wird
- Es wird ein Untampered Execution Environment gebootstrapped, indem der Attestation-Code ausgeführt wird

Angriff auf SCUBA

C. Castelluccia A. Francillion



$$C_j = C_j + PC \oplus mem[current\_address] + j \oplus C_{j-1} + x \oplus current\_address + C_{j-2} \oplus SR$$

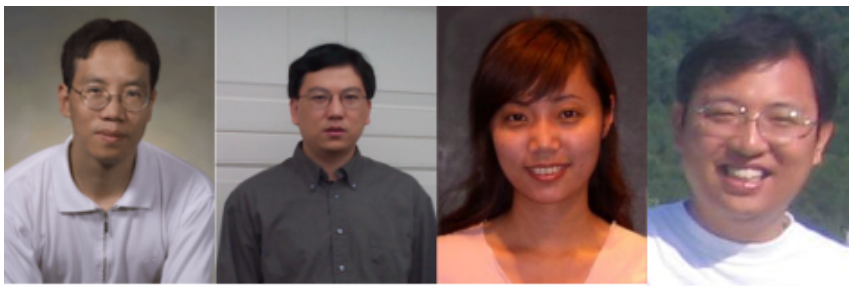
Angriff auf SCUBA: [CFPS09]

- *Most Significant Bit* im *PC* und jeder $mem[current\_address]$ kippen $\Longleftrightarrow$ ICE-Kopie an anderer Adresse ausführen
- Das Kippen des *Most Significant Bit* zweier Eingabewerte hebt sich bei Addition auf, und Carry wird weggeworfen: $C_{j-1} + PC \oplus mem[current\_address] + ...$

Beispiel, Original: $0b1... + 0b1... \oplus 0b0... = 0b0...$

Gekippt: $0b1... + 0b0... \oplus 0b1... = 0b0...$ ☺

Noise Filling



Noise Filling: [YWZC07]

- Ungenutztes Program Memory mit pseudozufälligen Daten *PRW* füllen; Der Angreifer kann *PRW* nicht überschreiben
- Inhalt von *PRW* in die Checksumme fließen lassen

Angriff auf Noise Filling: [CFPS09]

- Code Image komprimieren: $C(CI)$ um Platz für $\widetilde{CI}$ zu schaffen
- Während der Attestation $C(CI)$ dekomprimieren: $C^{-1}(C(CI))$ um die Response x korrekt zu berechnen

Attestation mittels Obfuskierung

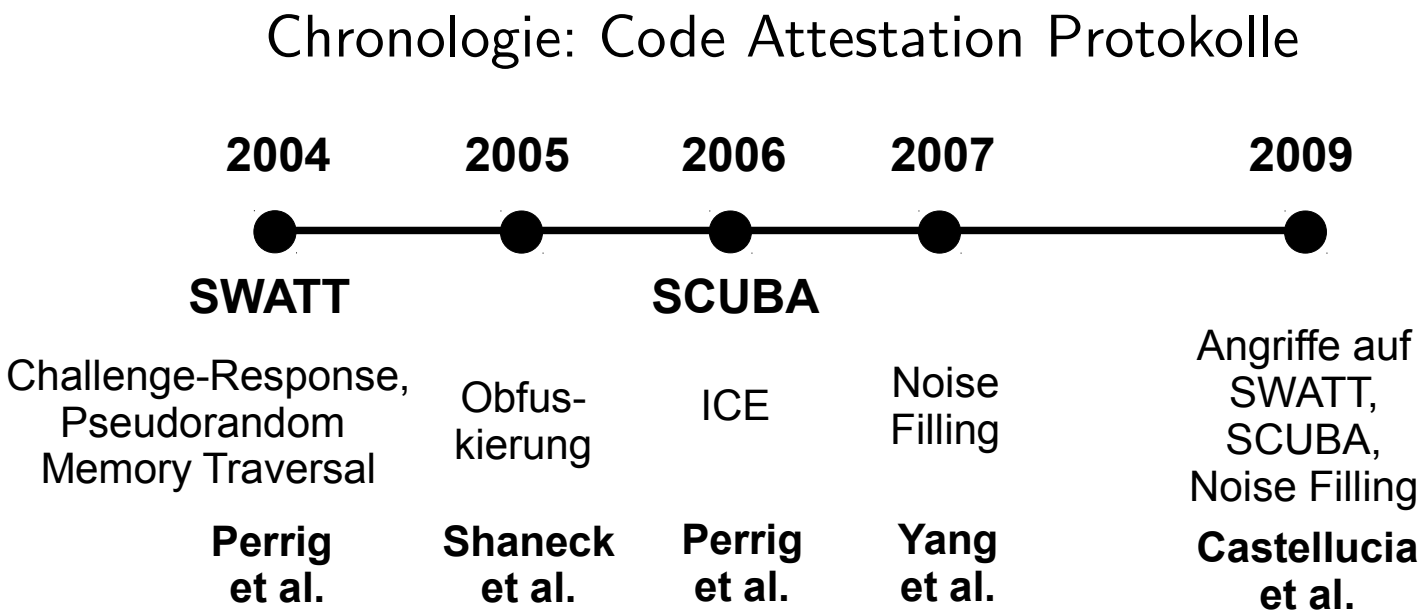
Attestation mittels Obfuskierung: [SMKK05]

- Der Attestation-Algorithmus wird zu Begin übertragen
- Der Angreifer muss den Algorithmus analysieren
- Obfuskierung durch *Self-Modifying Code*, *Randomization*, etc.

Probleme:

- *Self-Modifying Code* ist nicht auf Harvard-Arch. möglich
- Kompliziertes, fragwürdiges Konzept (Komplexität, Reverse Engineering VS endlich viele Algorithmus-Varianten)

Chronologie: Code Attestation Protokolle

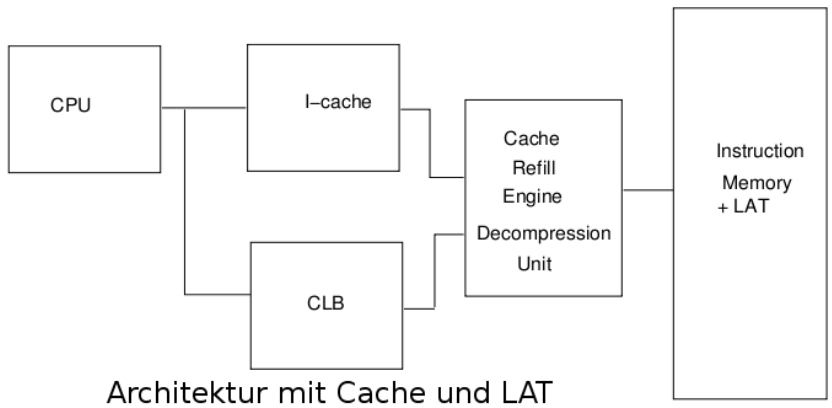


Gliederung: On-The-Fly Decompression

Gliederung: On-The-Fly Decompression

- Block-basiert
  - Cache und LAT
  - Cache im RAM
- Dictionary- oder Decoding Table basiert
  - Grundsätzlich
  - Neue Optimierungen
    - Instruction Splitting
    - Re-encoding

Block-basiert

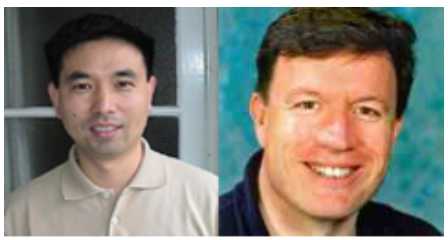


Architektur mit Cache und LAT

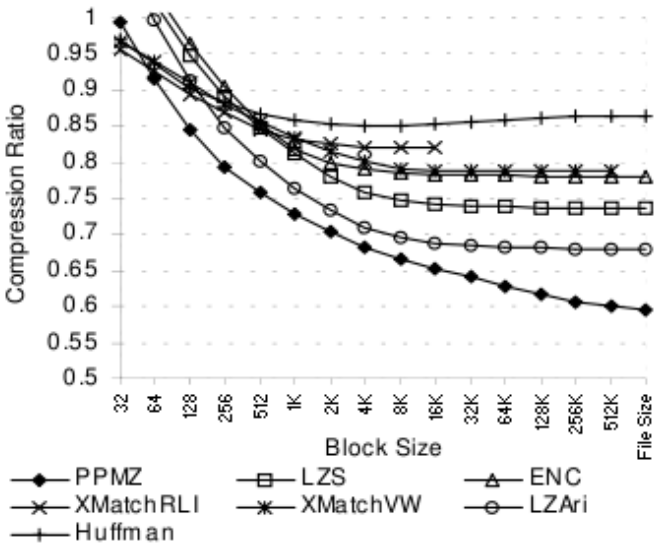
Block-basiert: [WC92]

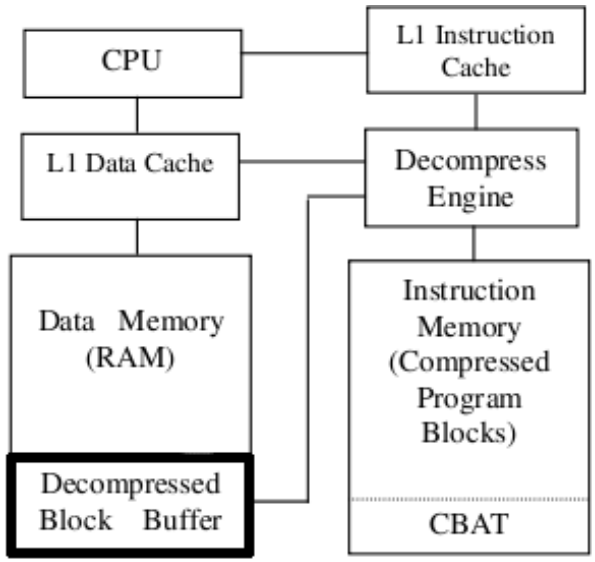
- Block-basierte On-The-Fly Decompression mit Line Address Table (LAT) und Cache
- Die LAT hält die Offsets der komprimierten Blöcke bereit
- Der Cache speichert den aktuellen Block zwischen
- Kleiner Cache, moderate Kompressionraten, Huffman-Coding

Cache im RAM



X. Xu S. Jones



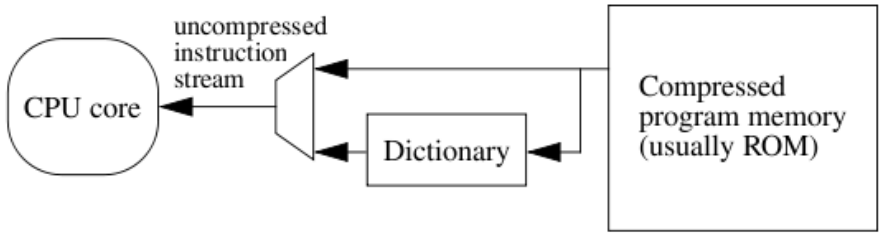


Arch. mit Cache im RAM

Cache im RAM: [XJ03]

- Die Kompressionsrate ist für größere Blockgrößen besser
- Cache-Größen von bspw. 32-bytes reichen nicht aus
- Die Blöckgröße kann deutlich größer ausfallen, wenn das RAM für den Cache verwendet wird

Dictionary-basiert



Dictionary-basiert: [LBCM97]

- Häufig vorkommende Codesequenzen werden in einem Dictionary gespeichert
- Im Code werden die Sequenzen durch kürzere Codewörter ersetzt $\Longrightarrow$ Kompression
- Schnelle Dekompression, aber für jedes Dictionary-Verfahren existiert beweisbar ein mindestens so gutes statistisches Verfahren [BCW90]
- Kaum freie Wahl des Kompressionsverfahrens

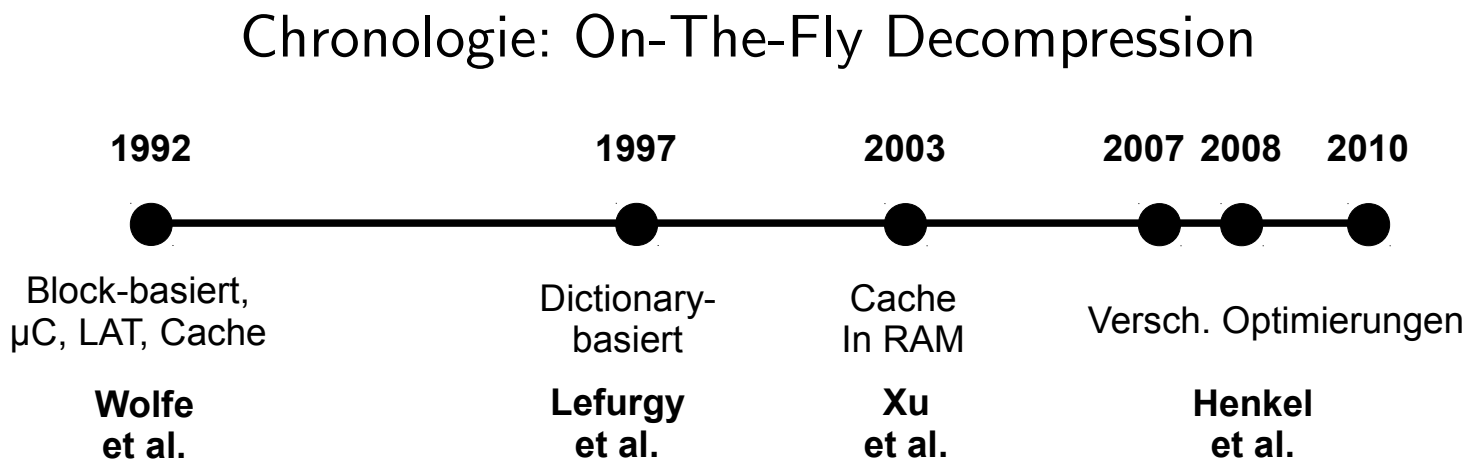
Optimierung durch Instruction Splitting

J. Henkel T. Bonny

Optimierung durch Instruction Splitting: [BH07]

- Dictionary/Table Verfahren kämpfen mit großen Dictionaries
- Unique Instructions brauchen unverhältnismäßig Platz im Dict.
- [BH07]: Unique Instr. werden in beliebige 'Patterns' zerlegt
- Jedes 'Pattern' soll insgesamt möglichst häufig vorkommen
- Huffman-Kompression der 'Pattern'-Code-Wörter
- Weitere Optimierungen von Henkel et al.: Instruction Re-encoding [BH08], Splitting plus Re-encoding [BH10]

Chronologie: On-The-Fly Decompression



Gliederung

1 Einführung

2 Related Work
- Code Attestation
- On-The-Fly Decompression

3 Einordnung und Abgrenzung

4 Zusammenfassung

5 Referenzen

Einordnung

Einordnung

- Challenge-Response Protokoll [SPvDK04]
- Pseudorandom Memory Traversal [SPvDK04]
- Timing-basert, aber nur grobe Nutzung [SPvDK04]
- Noise Filling [YWZC07]
- Block-basierte Kompression mit LAT, Cache [WC92]
- Cache im RAM [XJ03]

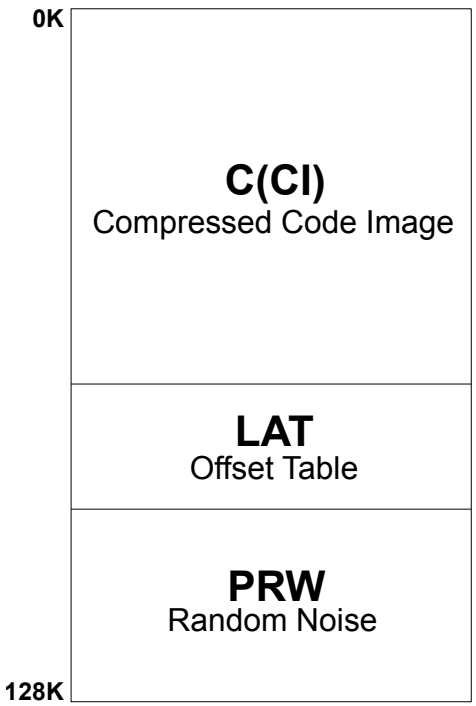
Abgrenzung

Abgrenzung

- Kompression zum Zwecke der Sicherheit
- ICE-basierte Verfahren [SLP+06]
- Attestation mittels Obfuskierung [SMKK05]
- Dictionary-basierte Kompressionverfahren [LBCM97]

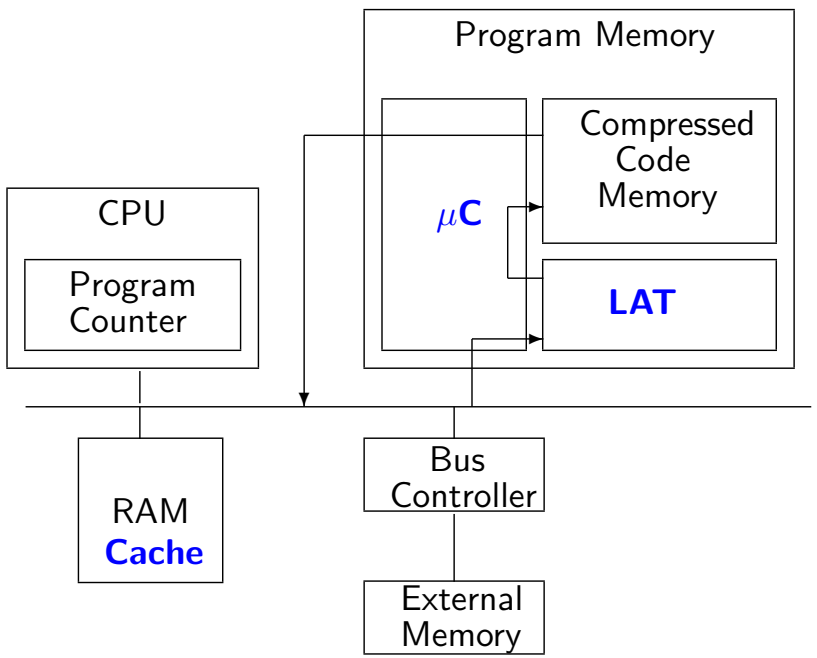
Unsere Wahl

Layout (Program Memory)



$$x = h(nonce||C(CI)||PRW||LAT)$$

Architektur (HW-Plattform)



Gliederung

1 Einführung

2 Related Work
- Code Attestation
- On-The-Fly Decompression

3 Einordnung und Abgrenzung

4 Zusammenfassung

5 Referenzen

Zusammenfassung

Zusammenfassung:

- Code Attestation
  - SWATT, Perrig et al. [SPvDK04]
  - SCUBA, Perrig et al. [SLP+06]
  - Noise Filling, Yang et al. [YWZC07]
  - Obfuskierung, Shaneck et al. [SMKK05]
  - Angriffe, Castelluccia et al. [CFPS09]
- On-The-Fly Decompression
  - Block-basiert, Wolfe et al. [WC92]
  - Cache im RAM, Xu et al. [XJ03]
  - Dictionary, Lefurgy et al. [LBCM97]
  - Versch. Optimierungen, Henkel et al. [BH07] [BH08] [BH10]
- Einordnung, Abgrenzung

Referenzen I


Timothy C. Bell, John G. Cleary, and Ian H. Witten.
Text compression.
Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.


T. Bonny and J. Henkel.
Instruction splitting for efficient code compression.
In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 646 –651, june 2007.


T. Bonny and J. Henkel.
Instruction re-encoding facilitating dense embedded code.
In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 770 –775, march 2008.

Referenzen II


Talal Bonny and Jörg Henkel.
Huffman-based code compression techniques for embedded processors.
ACM Trans. Des. Autom. Electron. Syst., 15:31:1–31:37, October 2010.


Claude Castelluccia, Aurélien Francillon, Daniele Perito, and Claudio Soriente.
On the difficulty of software-based attestation of embedded devices.
In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 400–409, New York, NY, USA, 2009. ACM.

Referenzen III


Charles Lefurgy, Peter Bird, I-Cheng Chen, and Trevor Mudge.
Improving code density using compression techniques.
In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, MICRO 30, pages 194–203, Washington, DC, USA, 1997. IEEE Computer Society.


Haris Lekatsas, Jörg Henkel, and Wayne Wolf.
Code compression for low power embedded system design.
In *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, pages 294–299, New York, NY, USA, 2000. ACM.

Referenzen IV

Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla.
Scuba: Secure code update by attestation in sensor networks.
In *Proceedings of the 5th ACM workshop on Wireless security*, WiSe '06, pages 85–94, New York, NY, USA, 2006. ACM.

Mark Shaneck, Karthikeyan Mahadevan, Vishal Kher, and Yongdae Kim.
Remote software-based attestation for wireless sensors.
In Refik Molva, Gene Tsudik, and Dirk Westhoff, editors, *Security and Privacy in Ad-hoc and Sensor Networks*, volume 3813 of *Lecture Notes in Computer Science*, pages 27–41. Springer Berlin / Heidelberg, 2005.

Referenzen V

10.1007/11601494_3.

A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla.
Swatt: software-based attestation for embedded devices.
In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 272 – 282, May 2004.

Andrew Wolfe and Alex Chanin.
Executing compressed programs on an embedded risc architecture.
SIGMICRO Newsl., 23:81–91, December 1992.

Referenzen VI


Xianhong Xu and S. Jones.
Code compression for the embedded arm/thumb processor.
In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2003. Proceedings of the Second IEEE International Workshop on*, pages 31 –35, 2003.


Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao.
Distributed software-based attestation for node compromise detection in sensor networks.
In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*, SRDS '07, pages 219–230, Washington, DC, USA, 2007. IEEE Computer Society.

Anhang: Table-Kompression