On the Integration of Harware-based Trust in Embedded Devices

Pedro Larbig, Nicolai Kuntze, Carsten Rudolph, Andreas Fuchs

Fraunhofer Institute für Sichere Informationstechnologie (SIT)
Rheinstrasse 75, 64295 Darmstadt, Germany
{pedro.larbig|nicolai.kuntze|carsten.rudolph|andreas.fuchs}@sit.fraunhofer.de
http://www.sit.fraunhofer.de

Zusammenfassung Während pro Jahr ca. 200 Mio. PCs produziert werden, entstehen gleichzeitig ca. 2 Mrd. Geräte mit eingebetteten Rechner-Kernen. Viele dieser Geräte beeinflussen zentrale und kritische Bereiche der Gesellschaft. Damit gelten oft hohe Anforderungen an Zuverlässigkeit, Integrität und Sicherheit der entsprechende Computersysteme des täglichen Umgangs, die durch aktuell verwendete eingebettete Systeme nicht ausreichend berücksichtigt werden. Die Sicherheit inhomogener Computer- und Netzstrukturen wird zwar von Fachleuten aus Industrie und Forschung seit langem als Problem verstanden, aber in der Praxis kaum im Hardwaredesign umgesetzt. Der Einsatz standardisierter Hardware-Sicherheitsmodule wie des Trusted Plattform Module (TPM) der Trusted Computing Group erlaubt eine einfache Integration fortschrittlicher Sicherheitstechnik. Ein TPM adressiert hierbei einen zuverlässigen Selbstschutz eines Gerätes sowie neuartige Konzepte zur externen Verifikation des Systemzustandes aus der Entfernung. Durch Integration von TPMs in solche Plattformen ist es möglich deutlich vertrauenswürdigere Systeme zu entwerfen und sogar strengsten Zulassungskriterien Rechnung zu tragen.
Diese Publikation diskutiert die relevanten Sicherheitsanforderung zur Identifikation und Integritätsprüfung solcher Systeme und bildet sie auf die Konzepte von TPM basierten Trusted Computing ab. Darauf basierend wird eine Möglichkeit der technischen Realisierung in Form der Integration in eine ARM-basierte Architektur beschrieben. Es wird dabei auf die nötigen Änderungen an der Hardware sowie Aspekte der Software und des Systemstarts durch U-Boot eingegangen, die nötig sind um einen TPM zuverlässig in Betrieb zu nehmen.

1 Einleitung

Eingebettete Systeme, wie sie beispielsweise in Industriesteueranlagen eingesetzt werden, zeichnen sich insbesondere dadurch aus, dass keine direkte Benutzerinteraktion stattfindet. Die eingesetzte Software auf solchen Anlangen ist typischerweise fest vorgegeben und konstant. Insbesondere ist es nicht vorgesehen, dass dynamisch Software durch Nutzer auf einem solchen Gerät installiert wird. Diese Eigenheiten erzeugen oft einen falschen Eindruck von Sicherheit, da diese

eingebetteten Geräte nichtsdestotrotz veränderbar bleiben, von dieser Möglichkeit lediglich im normalen Ablauf kein Gebrauch gemacht wird. Auch wenn es keine dokumentierten Wege zur Softwareinstallation gibt, unterstützen die meisten Geräte Firmwareupdates oder administrative oder Debug Zugänge zum Dateisystem [1]. Auch vereinfacht die standardisierte und bekannte Software auf den eingebetteten Systemen eventuelle Einbruchsversuche. Ein Angreifer kann bereits vorab infizierte Ersatzsoftware vorbereiten ohne Kenntnisse über die konkreten Benutzereinstellungen zu benötigen. Nach dem Einspielen einer solchen veränderten Software wären auch für einen Administrator keine Änderungen sichtbar.

Im Entwurf von vertrauenswürdigen Systemen kann zwischen einbruchssicheren (tamper proof) und einbruchsdokumentierenden (tamper evident) Systemen unterschieden werden. Ein tamper proof System kann auf Grund seiner Konstruktion nicht unautorisiert verändert werden. Generell gilt, dass es teuer und sehr kompliziert ist ein solches Gerät zu bauen oder in vielen Fällen gar nicht einsetzbar, da solche Geräte oft nur sehr unflexible verwendet werden können. Ein tamper evident System hingegen erlaubt es Modifikationen eines Systems automatisiert zu erkennen oder durch externe Prüfer die Korrektheit eines Systems zu bestätigen [2]. Auch wenn ein Gerät modifiziert wurde, ist es dem externen Prüfer möglich die Änderung zu erkennen und so die *Gesundheit* des Gerätes zu ermitteln. Plomben und Siegel sind weitverbreitete Anwendungen des gleichen Konzeptes der tamper evidence.

Trusted Computing, wie es durch die Trusted Computing Group (TCG) [3] spezifiziert wird, stellt Mechanismen zur Verfügung, mit denen unter anderem tamper evidence ermöglicht wird, bzw. verlässlich Informationen bezüglich der Konfiguration eines Gerätes zu ermitteln und andere Geräte darüber zu informieren. Eine Konfiguration besteht hierbei aus der Software, die auf dem Gerät gestartet wurde, sowie deren Parametern. Diese Informationen ermöglichen es dann einem externen Prüfer den Zustand des Gerätes zu beurteilen und gegebenenfalls zu reagieren, falls diese von einem als vertrauenswürdig bekannten Zustand abweicht. Die Konzepte der TCG wurden ursprünglich den PC Bereich definiert und sind bisher auch hauptsächlich hier im Einsatz. Im Folgenden wird erklärt, wie diese Technologie für die Nutzung in eingebetteten Geräte mit einem ARM Prozessor verwendet werden kann. Dieses Einsatzgebiet ist bisher nicht durch die bestehenden Standards der TCG abgedeckt, obwohl sie zum Umsetzung von Anwendungsszenarien wie [4] oder [5] nötig sind.

Diese Publikation präsentiert einen Ansatz zur Integration eines TCG-definierten Hardware-basierten Vertrauensankers – Trusted Platform Module (TPM) – in eingebettete Systeme und somit die technische Verankerung des Vertrauens, das diesen Geräten zuteil wird. Erste Ansätze hierzu wurden bereits in [6] veröffentlicht. Um ein TPM [7] zur Beurteilung von Gerätekonfigurationen verwenden zu können, müssen einige Voraussetzungen seitens der Plattformarchitektur des eingebetteten Gerätes erfüllt sein. Insbesondere der so-genannte Root of Trust for Measurement (RTM) wird im folgenden Beitrag in der Form eines modifizierten U-Boot [8] realisiert.

2 Sicherheitsanforderungen

Jede Form der Interaktion von vernetzten Geräten, insbesondere Fernwartung und Betrieb von eingebetteten Geräten, hat besondere Bedürfnisse bezüglich der Sicherheitsbeurteilung des Gerätes. Bei lokalem Zugriff auf Geräte wird die Echtheit des Gerätes, dessen Authentizität, meist durch physische Eigenschaften wie Siegel und Beschriftungen festgestellt und der Zugang wird durch Türen und Schlösser geregelt. Für vernetzte Geräte sind diese Maßnahmen nicht ausreichend.

Sicherheitsanforderungen für Industriesteuerungssysteme oder verteilte Netze sowie Sensornetze beinhalten insbesondere die Anforderung der Systembetreiber Geräte sicher zu identifizieren und mit diesen Geräten vertrauenswürdig zu kommunizieren. Darüber hinaus sollen verlässlich Zustandsinformationen über ein Gerät eingeholt werden können. Insbesondere darf es dabei nicht möglich sein, dass ein Gerät (sei es extern oder intern) die Identität eines anderen Gerätes duplizieren kann oder dass ein Gerät Statusinformationen eines anderen Geräte anstelle seiner selbst ausliefert.

3 Einführung in Trusted Computing

Das Prinzip von Trusted Computing wird durch die Trusted Computing Group anhand einer Reihe von Ankern für Vertrauen in einem System definiert. Für die vorliegende Arbeit wichtig sind dabei insbesondere der *Root of Trust for Measurement (RTM)*, der *Root of Trust for Storage*, sowie der *Root of Trust for Reporting*.

Die Aufgabe des *Root of Trust for Measurement (RTM)* ist es, als Anker für die Erhebung der Konfiguration einer Plattform zu dienen. Bereits beim Starten des Gerätes – noch vor Starten des Betriebssystems oder anderer Anwendungssoftware – wird dieser RTM initialisiert. Das verhalten des RTM muss unveränderbar sein. Beim Starten des RTM misst dieser die Konfiguration der Hardware-Plattform während diese Initialisiert wird, sowie die erste gestartete Software-Komponente. Danach ist der RTM beendet und führt keine weiteren Aktionen mehr durch. Hiermit lassen sich also alle Änderungen an der Plattform oder der zuerst gestarteten Software-Komponente erkennen.

Veränderungen, die nicht direkt durch den RTM erfasst werden können, da sie beispielsweise Teil späterer Software-Komponenten wie dem Betriebssystem oder Anwendungssoftware sind, müssen jedoch auch "gemessen" werden. Hierfür muss jede Software-Komponente die jeweils als nächstes zu startende Software-Komponente oder Konfiguration messen, bevor diese Einfluss auf das System nehmen können. Dieses Vorgehen wird als *Chain of Trust for Measurement* bezeichnet. Der RTM stell hierbei den Beginn der Kette dar.

Die Messwerte dieser Kette werden von den jeweiligen Elementen der *Chain of Trust for Measurement* an den *Root of Trust for Reporting (RTR)* weitergegeben. Der RTR speichert die Messwerte und ist in der Lage im Rahmen einer Überprüfung über den Systemzustand Auskunft zu geben. Im Falle eines Trusted Platform Module (TPM) in seiner Rolle als RTR werden die Messwerte

in dem sogenannten Platform Configuration Registeres (PCR) gespeichert. Aufgrund der Limitierung der Hardware eines TPMs werden diese Messwerte jedoch nicht alle individuell gespeichert. Jeweils der nächste Messwert wird mit dem aktuellen Wert durch eine Hashoperation zu dem neuen Wert des PCR Registers verarbeitet.

$$PCR_{t0} = 0 \qquad PCR_{tn} = SHA1(PCR_{t(n-1)}|measurement).$$

Durch Beschränkungen beim Zugriff auf die PCRs im TPM und die Verwendung eines sicheren Hashverfahrens im TPM kann diese Operation nicht rückgängig gemacht werden. Sofern dem RTR/TPM also eine Messung mitgeteilt wurde, wird diese bis zum Ausschalten des Gerätes gesichert.

Zur Prüfung eines Endgerätes bildet der TPM eine digitale Signatur über PCR Werte unter Verwendung eines Schlüssels, der durch den TPM als *Root of Trust for Storage (RTS)* geschützt wird. Der TPM bietet mit dem RTS eine SmartCard ähnliche Funktion zur Signaturerstellung an und schützt die eingesetzten Schlüssel gegen Offenlegung. Der RTS bietet dadurch auch die Möglichkeit ein Endgerät sicher zu identifizieren. Einer dieser Schlüssel, der Endorsement Key (EK), ist speziell für diesen Zweck vorgesehen. Zum Erstellen eines Measurement Reports – Attestation genannt, kommen sogenannte Attestation Identity Keys (AIK) zum Einsatz, die an den EK gebunden. Sie erlauben es, als Pseudonym eine Abstraktionsschicht zwischen der eindeutigen Geräteidentität (in Form des EK) und der Attestation einzufügen, was in Desktop-PCs primär dem Privatsphärenschutz dient.

3.1 Geräteprüfung

Bei einem solchen Gerät ist es dann einer entfernten Partei, normalerweise ein anderes gerät im Netz oder der Systembetreuer, den Gesundheitszustand anhand einer solchen die *Attestierung* überprüfen. Eine Attestierung besteht aus den Hashwerten (Messungen) von jedem Teil der ausgeführten Programme und benötigten Konfigurationsdateien seit dem Start des Geräts und kann auch Dateien im Massenspeicher oder weitere Hardware abbilden. In der so entstandenen Messkette, die vom Firmware Loader bis zur Anwendungssoftware reicht, sind alle aktiven Software-Komponenten erfasst. Dies hat zur Konsequenz, dass schadhafte Programme zwar gestartet werden können, aber dann auch in der Messliste aufgeführt werden. Auch können sich solche Programme nicht mehr selbst aus der Messkette entfernen, da immer zuerst gemessen und erst dann gestartet wird. Außerdem lässt sich die Operation des Hinzufügens eines Messwertes zu einem PCR nur durch einen Neustart rückgängig machen, wodurch ein Verstecken von Schadprogramme nicht möglich ist.

Wenn eine Überprüfung des Gerätes angefordert wird, signiert der TPM eine Menge an PCRs unter Verwendung eines AIKs, dessen privater Schlüssel durch den TPM gegen Auslesen und Falsch- sowie Fremdverwendung geschützt ist. Schadprogramme können die Antwort nicht fälschen, da hierzu das Wissen über den durch das TPM geschützten Schlüssel benötigt wird, um den PCR zu signieren.

Auch beinhaltet die Attestierung einen 160-bit Zufallswert (Nonce), der in die Antwort eingebettet wird, so dass der Prüfer wiedereingespielte Attestierungen sicher erkennen kann. Zusammen mit dem PCR Wert und der Signatur des PCR Wertes wird das sog. Stored Measurement Log (SML) übermittelt. In diesem Log sind alle Einzelmessungen der Measurement Chain enthalten. Anhand des SML lassen sich die gestarteten Programme bewerten sowie auf Unveränderheit prüfen. Aus dem SML lassen sich dann die Werte der PCRs nachberechnen, um diese dann mit den im TPM in die Signatur eingegangenen Werten abzugleichen.

4 Anpassung auf den Einsatz in eingebetteten Geräten

Im Gegensatz zu Desktop oder Server PCs erfüllen eingebettete Geräte spezialisierte Aufgaben. Diese Spezialisierung betrifft die Variabilität der in solchen Szenarien eingesetzen Software, sowie die Menge der möglichen Systemzustände. Allerdings sind auch eingebettete Systeme universelle Rechner, so dass ihre Funktionalität nicht wie bei einer spezialisierten Hardwareimplementierung (ASIC) eingeschränkt ist. Deshalb ist es auch bei diesen spezialisierten Geräten sinnvoll und nötig Gerätekonfiguration und Software auf Veränderungen zu prüfen.

Die Spezialisierung von Software schlägt sich auch in der Gerätearchitektur nieder. Oft werden beispielsweise Read-Only Firmwares benutzt (z.B. squashfs [9]) und Konfiguration und Zustand getrennt von der Software gespeichert. Wenn ein Gerät so aufgebaut ist, kann die oben beschriebenen Vorgehensweise vereinfacht werden. Es muss dann keine komplexe *Trusted Chain of Measurement* aufgebaut werden. Stattdessen kann ein RTM die komplette Firmware messen bevor sie gestartet wird. Dies reduziert Auswirkungen einer Trusted Computing Lösung auf die Systemsoftware, da das Betriebsystem nicht angepasst werden muss um Messungen gestarteter Software vorzunehmen und sich auch das Laufzeitverhalten nicht verändert. Auch ist es nicht nötig jede Software-Komponente einzeln zu messen und in einer SML zu speichern. Stattdessen kann das gesamte Firmware-Image gemessen und vom Prüfer gegen einen Referenzwert abgeglichen werden.

Sehr häufig kommen in eingebetteten Systemen keine x86 basierten Computer mit BIOS oder UEFI sondern ARM basierte Geräte mit U-Boot zum Einsatz. Bisherige Spezifikationen der Trusted Computing Group beziehen sich aber insbesondere auf eine Implementierung des RTM im pre-BIOS bzw. im UEFI in denen der RTM besonders zu schützen ist. Auf ARM Plattformen gibt es in vielen Fällen allerdings bereits ohne Bezug auf Trusted Computing die Möglichkeit eines gesicherten Starts von Software (z.B. ARM Secure Boot [10]) oder von nur einmal beschreibbarem Speicher, der auch gegen physische Manipulationen geschützt ist.

Um im präsentierten Konzept keine Veränderungen an bestehender Hardware vornehmen zu müssen, wurde auf eine Implementierung des RTM in Hardware verzichtet. Stattdessen wird der RTM als Teil des Bootloaders ausgeführt und durch die Secure Boot Fähigkeiten des Chips oder Flash Fuses geschützt. So kann der Hersteller eines solchen Geräts den Bootloader mit einer digitalen Signatur

versehen und den Verifikationsschlüssel in der CPU verankern. Bei jedem Start überprüft die CPU dann den Bootloader. Nur der Gerätehersteller ist somit in der Lage ein neues Bootimage zu erstellen, da hierzu der private Schlüssel zur Signaturerstellung benötigt wird. Insbesondere ist es möglich, den privaten Schlüssel nach der Programmierung des Bootloaders zu vernichten oder alternativ die Überschreibbarkeit zu deaktivieren um eine noch höhere Sicherheit gegen Manipulation zu gewährleisten. Dieser nicht modifizierbare Bootloader garantiert dann die Funktion des RTM.

Basierend auf diesem Konzept ist es möglich die Anforderungen der sicheren Authentisierung und Gesundheitsprüfung unter Verwendung des TPM als RTS und RTR sowie eines abgesicherten Bootloader als RTM zu erfüllen.

5 Implementierung

Im Folgenden werden Details zur Implementierung gezeigt und ein Demonstrator auf der Basis eines BeagleBoards vorgestellt. Der benötigte Source Code kann bei den Autoren angefragt werden.

5.1 Hardware-Anbindung des TPM

Die Implementierung des Trusted Computing Konzepts für eingebettete Geräte basiert auf dem Einsatz eines TPM, der über den I2C Bus angebunden ist. Der TPM realisiert den Root for Trust und Roots for Storage und Reporting. Der Root of Trust for Measurement wird durch eine Erweiterung von U-Boot um die nötigen TPM Kommandos etabliert. Hierdurch ist das U-Boot in der Lage Messungen über Konfigurationsdaten, Shell Kommandos, etc. in die vorgesehenen PCR Register zu schreiben. Es wird angenommen, dass das U-Boot durch einen Secure Boot geschützt ist.

5.2 Erweiterung von U-Boot zum Root of Trust for Measurements

Um eine maximale Flexibilität des Einsatzes von Trusted Computing in eingebetteten Geräten zu ermöglichen, misst der erweiterte U-Boot nicht automatisch die nachfolgend startende Software. Stattdessen wird durch den Firmwarentwickler bestimmt welche Software-Komponenten gemessen werden sollen – nur der folgende Kernel, der Kernel zusammen mit der zugehörigen Initrd oder gesamte oder Teile des Flashs. Um Angriffe auf der Ebene der U-Boot Shell zu verhindern werden die Kommandos auf der Shell ebenfalls erfasst und in einem PCR hinterlegt.

Jedes Kommando das in der U-Boot Shell nach dem Start des Systems ausgeführt wird, wird gemessen und in einem PCR entsprechend der Reihenfolge der Aufrufe erfasst. Hierdurch ist es einem Angreifer nicht möglich die RTM Funktionalität zu umgehen. Dies wird durch eine Veränderung am Kommandointerpreter erreicht. Bevor ein Kommando ausgeführt wird, wird es gehasht (SHA-1) und der Hash an einen PCR Register übermittelt. Das Kommando wird ausgeführt,

sobald der TPM die PCR Extend Operation bestätigt hat. Falls der TPM dies nicht bestätigt wird das Kommando nicht ausgeführt, so dass der Angreifer den RTM nicht umgehen kann.

Die Funktionsweise der bereits bestehenden Kommandos von U-Boot wurde nicht verändert. Zum Messen von Daten die auf den erweiterten U-Boot oder das nachfolgende System Einfluss haben, wurde stattdessen ein *measure* Kommando zur Verfügung hinzugefügt, das es erlaubt bestimmte Speicherbereiche zu messen. Es ist die Aufgabe des Bootskript Erstellers alle relevanten *measure* Kommandos in das U-Boot Startskript einzufügen.

Im Falle eines Angriffs bei dem das U-Boot Skript verändert oder per Shell die *measure* Kommandos umgangen werden, ist es dennoch nicht möglich im Nachhinein zu simulieren, dass eigentlich ein nicht schadhaftes System geladen wurde, da die Reihenfolge der Kommandos im U-Boot ein *measure* Kommando vor dem *boot* Kommando vermissen ließen. Eine solche Veränderung der erwarteten Reihenfolge würde in der Prüfung einer Attestierung auffallen.

5.3 Demonstration

Der beschriebene Ansatz wurde auf der Basis eine BeagleBoards mit einem Infineon I2C TPM und einem modifizierten U-Boot basierend auf der Version 2011.12 (siehe Abbildung 1) realisiert. Der Start des initialen U-Boot ist geschützt durch die Secure Boot Techniken der CPU. Heutige ARM Prozessoren bieten verschiedene Ansätze wie High Assurance Boot process [12] oder Secure Boot, wie z.B. in den iMX.51 CPUs[1] implementiert.

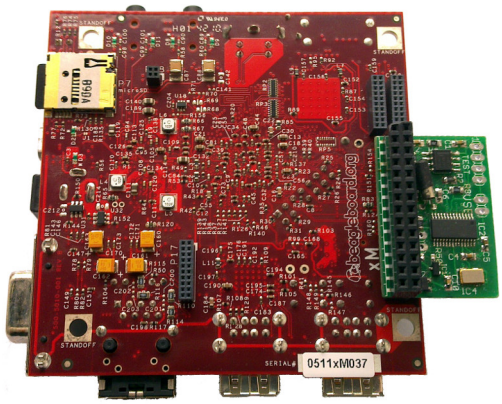

Abbildung 1. Demonstration Board

Der modifizierte U-Boot kündigt sich als RTM an:

[1] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX515

```
Hit any key to stop autoboot:  0
This is a secured shell, all commands will get measured!
```

Das U-Boot startet und initialisiert den TPM über die I2C Schnittstelle mit der folgenden Sequenz an Befehlen:

```
1. tis_init: Initialize I2C Bus and probe TPM
2. tis_open: Register TPM, read device ID and locality
3. TPM_Startup
4. TSC_PhysicalPresence
5. TPM_Enable
6. TPM_PhysicalSetDeactivated: Set to 0 (activate)
```

Das Ergebnis wird berichtet. Jedes OK repräsentiert hierbei einen Schritt aus der vorher gezeigten Sequenz.

```
Initializing TPM: OK OK OK OK OK OK
```

Danach startet das U-Boot das Lesen und Ausführen des Setup Skript, wenn der Benutzer nicht die Shell benutzt.

```
reading uEnv.txt
```

Das Beispiel-Skript misst das uImage und startet dann den Kernel.

```
reading uImage
4073824 bytes read
SHA1  : 93cea9bf2c8fea43327e25838087faf7536673ad
PCR 01: 77c03ffe4adf4b91d2f04b72635ec6fadba48c98
Booting from mmc ...
## Booting kernel from Legacy Image at 80200000 ...
```

Nach dem Start des Linux Kernel findet sich in PCR-01 der Messwert für das Kernelimage, die U-Boot Kommando Sequenz wurde nach PCR-06 gemessen und der Messwert des Linux Kernel wurde nach PCR-10 berichtet.

```
# cat /sys/devices/platform/omap/omap_i2c.2/i2c-2/2-00200/pcrs
PCR-00:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-01:77 C0 3F FE 4A DF 4B 91 D2 F0 4B 72 63 5E C6 FA DB A4 8C 98
...
PCR-06:5E AF 6D B6 88 4B 8C 8D 85 CA 19 56 B3 02 AE 9F 6C 31 80 3C
...
PCR-10:60 17 FC E3 4E 6C 4C D0 B9 A0 8C E5 54 C5 3F 47 E0 0D 8C 68
...
```

Die U-Boot Erweiterung zur Kommunikation mit dem TPM aus der U-boot shell bietet die folgenden Kommandos:

```
# help tpm
This is a secured shell, all commands will get measured!
```

```
Initializing TPM: OK OK OK OK OK OK
tpm - Subsystem for TPM startup and extending PCRs

Usage:
tpm send <bytes>          - write stream of hex bytes to TPM and
                            read response
tpm startup <state>       - Send Startup command to make TPM start
                            in the given state (1=clear, 2=saved,
                            3=deactivate)
tpm presence <bits>       - Sets lifetime and physical presence
                            bits, refer to TPM Specs:
                            TSC_PhysicalPresence
tpm enable                - Enables the TPM
tpm activate              - Activates the TPM by clearing the
                            Deactivated Flag
tpm selftest              - Instructs the TPM to run a
                            comprehensive self test
tpm bootup                - Boots up TPM by issuing Startup,
                            Presence, Enable and Activate commands
tpm readpcr [index]       - Display contents of PCR. If no index
                            is given, all registers will be
                            displayed
tpm extend <index> <hash> - Extend PCR with the supplied SHA1 hash
                            in hex
tpm measure <index> <addr> <len> - Read len bytes at addr and
                            extend PCR with SHA1 hash of that area
```

Dies erlaubt auch einen manuellen Zugriff auf den TPM durch das U-Boot, wenn das Betriebssystem keine Kommunikation zum TPM unterstützt.

```
# tpm readpcr 6
PCR 06: 295f8f267ab67e0a6f7dbd14a870a31cd1ea4901

# tpm measure 1 ${loadaddr} 0x${filesize}
SHA1  : 93cea9bf2c8fea43327e25838087faf7536673ad
PCR 01: 77c03ffe4adf4b91d2f04b72635ec6fadba48c98
```

6 Fazit

Das vorgestellte und prototypisch implementierte Konzept liefert eine Adaption der Trusted Computing Technologie der Trusted Computing Group an eingebettete Geräte. Neben der Hardware-seitigen Integration des TPM Chips in das eingebettete Gerät ist die wichtigste Entwicklung dabei das Konzept zum Messen und Berichten des Systemzustands. Dieses Konzept wurde durch eine Erweiterung von U-Boot umgesetzt, die flexibel ist und verschiedene Bootverfahren und Verzweigungen im Bootskript unterstützt. Das Bootskript kann durch den

Gerätebetreiber angepasst und für verschiedene Firmwareversionen angepasst werden, so dass auch Fallback-Modi unterstützt werden können. Hierdurch behält der Gerätebetreiber eine größtmögliche Flexibilität beim Betrieb, ohne auf die Möglichkeiten einer sicheren Attestierung verzichten zu müssen.

Die Anwendbarkeit dieses Konzepts wurde durch den beschriebenen Demonstrator gezeigt, der auf einem handelsüblichen BeagleBoard in Kombination mit einem Infineon TPM über I2C lauffähig ist. Die Vertrauenskette startet im Secure Boot und wird dann über das U-Boot auf die gesamte Software ausgeweitet um eine externe Prüfung des Gesamtsystems zu ermöglichen. Dies ermöglicht ein Gerät sicher zu identifizieren, sowie dessen Gesundheit zu überprüfen. In einem entsprechenden Gesamtkonzept könnten darauf basierend Maßnahmen zur Abwehr von Gefahren bei einer Veränderung eingesetzt werden.

7 Danksagung

Diese Arbeit wurde mit der Unterstützung des deutschen Forschungsprojektes ANSII sowie des EU FP7 Projektes SecFutur durchgeführt. Weiterhin hat Infineon die Arbeiten mit Hardware und Software aktiv unterstützt.

Literatur

1. Kuntze, N., Rudolph, C., Cupelli, M., Liu, J., Monti, A.: Trust Infrastructures for Future Energy Networks. In: Power and Energy Society General Meeting - Power Systems Engineering in Challenging Times. (2010)
2. Kuntze, N., Rudolph, C., Bente, I., Vieweg, J., von Helden, J.: Interoperable device Identification in Smart-Grid Environments. In: Power and Energy Society General Meeting, 2011 IEEE. (July 2011)
3. Mitchell, C.: Trusted Computing, Institution of Electrical Engineers (2005)
4. Kuntze, N., Rudolph, C., Paatero, J.: Establishing Trust between Nodes in Mobile Ad-Hoc Networks. In Mitchell, C., Tomlinson, A., eds.: Trusted Systems. Volume 7711 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 48–62
5. Kuntze, N., Rudolph, C.: On the Automatic Establishment of Security Relations for Devices. In: IFIP/IEEE International Symposium On Integrated Network Management. (2013)
6. Winter, J.: Trusted Computing Building Blocks for Embedded Linux-based ARM TrustZone Platforms. In: Proceedings of the 3rd ACM workshop on Scalable trusted computing, ACM (2008) 21–30
7. Kinney, S.: Trusted Platform Module Basics: Using TPM in Embedded Systems. Newnes (2006)
8. Ding, X., Liao, Y., Fu, J., Huang, H., Liu, W.: Analysis of Bootloader and Transplantation of U-Boot Based on S5PC100 Processor. In: Proceedings of the 2011 Third International Conference on Intelligent Human-Machine Systems and Cybernetics - Volume 01. IHMSC '11, Washington, DC, USA, IEEE Computer Society (2011) 61–64
9. Pavlov, A.I., Cecchetti, M.: SquashFS HOWTO (2005)

10. Dietrich, K., Winter, J.: Secure boot revisited. In: Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for, IEEE (2008) 2360–2365
11. Sparks, E.R., Sparks, E.R.: A Security Assessment of Trusted Platform Modules. Technical Report TR2007-597, Department of Computer Science Dartmouth College (2007)
12. Ashkenazi, A.: Security Features in the i. MX31 and i. MX31L Multimedia Applications Processors. Freescale Semiconductor, Online (2005)