TPM 2.0 Policies in der Praxis

Einfach zum sicheren Rechtemanagement für Embedded Systems

Markus Wamser, Mixed Mode GmbH

Trusted Platform Modules (TPMs) sind seit vielen Jahren fest am Markt etabliert. Mittlerweile haben Module nach der aktuellen Version 2.0 des Standards ältere Module weitgehend abgelöst. Dennoch bleiben viele der Neuerungen und Funktionen dieser Module ungenutzt.
Ein prominentes Beispiel ist das Konzept der Extended Authorization Policies. Damit sind nicht nur sichere und vertrauenswürdige Boot- und Update-Konzepte möglich. Mit wenig Aufwand lässt sich auch ein Konzept zum Rechte- bzw. Lizenzmanagement umsetzen, zum Beispiel in einem Fahrzeug.

Im Alltag verbinden viele den Begriff Digitales Rechtemanagement (DRM) mit dem Kauf und Konsum von Mediendateien. Tatsächlich macht Videostreaming mehr als die Hälfte des gesamten Internetverkehrs aus [1], davon ist ein Großteil per DRM geschützt. Im Embedded-Umfeld sind die Einsatzmöglichkeiten für ein Rechte- und Lizenzmanagement aber weit vielfältiger. So lassen sich Nutzungsrechte für einzelne Softwarefunktionen kryptographisch sicher an eine Autorisierung binden. Diese Autorisierungen wiederum können mit einem TPM 2.0 Modul flexibel und nahezu beliebig komplex gestaltet werden. Gleichzeitig ist aber die Implementierung im eingebetteten System kostengünstig realisierbar, da lediglich ein TPM 2.0 Modul als hardwarebasierter Vertrauensanker vorhanden sein muss.

Dieser Zugewinn an Flexibilität und Sicherheit entstammt wesentlich dem Konzept der Extended Authorization und darauf aufbauender Policies, das beim Übergang zu Version 2.0 Einzug in den TPM-Standard gefunden hat.

TPM 2.0: Motivation und Änderungen zu TPM 1.2

Trusted Platform Modules (TPMs) wurden aus der Not geboren, einen möglichst günstigen, aber dennoch sicheren Vertrauensanker in Geräten zu haben. Einfache, auf eine Hauptplatine auflötbare Chips, die grundlegende Funktionen zur Generierung und zur sicheren Aufbewahrung von kryptographischem Schlüsselmaterial boten, fanden kurz nach der Jahrtausendwende zum ersten Mal weite Verbreitung. Inkompatibilitäten im Detail verhinderten aber eine tatsächliche intensivere Verwendung, mit Ausnahme einzelner großer Unternehmen mit homogenen Umgebungen. Ein Wendepunkt war die Verabschiedung der Version 1.2 des TPM-Standards durch die Trusted Computing Group (TCG) im Jahr 2009. Sie brachte neben Verbesserungen am Sicherheitskonzept vor allem eine weitgehende Harmonisierung von Schnittstellen und Funktionen. Trotz des kleinen Versionssprunges war dies ein Meilenstein für die Akzeptanz und Verwendung von TPMs. Eines des prominentesten Anwendungsszenarien – TPM-gestützte Festplattenverschlüsselung – wurde fester Bestandteil von Betriebssystemen und kann seither ohne weitere Software oder aufwändige Konfiguration genutzt werden.

Die intensivere Nutzung von TPMs offenbarte aber auch Schwächen dieser Version: Der Standard war über mehrere Versionen entsprechend der Wünsche von Anwendern gewachsen, behielt aber stets eine Abwärtskompatibilität. Dadurch war er komplex und schwer lesbar geworden, was die Entwicklung weitere

Einsatzszenarien hemmte. Ebenso wirkte sich das Fehlen weiterer Funktionen hemmend aus. Als größter Mangel offenbarte sich schließlich die enge Festlegung auf bestimmte (wenige) kryptographische Verfahren. Sollte sich auch nur eines dieser Verfahren als schwach oder unsicher erweisen, so gab es keine Möglichkeit auf Alternativen auszuweichen. Dass diese Furcht nicht nur theoretischer Natur war, zeigte sich noch während der Entwicklung der Version 2.0. Erste praktikable Angriffe auf SHA-1, das in Version 1.2 festgeschriebene Hashverfahren, zeichneten sich am Horizont ab.

Die Entwicklung von Version 2.0 des Standards war deswegen von drei grundlegenden Gedanken geprägt:

- Der Standard sollte von Grund auf neu konzipiert und geschrieben werden, um die Lesbarkeit zu verbessern und Mehrdeutigkeiten oder Widersprüche zu vermeiden.
- Im Standard sollten statt konkreter kryptographischer Primitiven nur Klassen von Primitiven definiert werden. Diese *Agilität* ermöglicht eine schnelle und effiziente Reaktion, falls neue Angriffe auf die verwendeten Algorithmen bekannt werden. Ebenso erhöht es die Einsetzbarkeit von TPMs, etwa im staatlichen Umfeld, wo regional verschiedene Verschlüsselungsalgorithmen zum Einsatz kommen. Agilität in der Auswahl der kryptographischen Primitive war der primäre Grund für Renovierung des Standards,
- Das Konzept von Administrator- und Benutzerrollen sollte stark überarbeitet werden um mehr Einsatzszenarien zu ermöglichen. Gleichzeitig sollten alle mit einem TPM der Version 1.2 realisierbaren Use Cases weiterhin darstellbar bleiben.

Die prägendste Änderung war die Einführung symmetrische Kryptographie. Auf diese war in den 1.x-Versionen bewusst verzichtet worden, um einen Konflikt mit eventuellen Exportrestriktionen zu vermeiden. Flexibilität in der Auswahl von Kryptoverfahren und die Verwendung moderner Primitiven (etwa asymmetrischer Verfahren auf Basis von Elliptischen Kurven) machten die Einführung hybrider Verfahren und damit symmetrischer Primitiven notwendig. Zwischenzeitlich gelockerte Exportrestriktionen vereinfachten diese Entscheidung.

Die zweite zentrale und nicht zu unterschätzende Änderung war die Vereinheitlichung der verschiedenen Autorisierungsverfahren zur sogenannten *Enhanced Authorization*. Diese Vereinheitlichung erweiterte das Spektrum an Einsatzmöglichkeiten, etwa im Sinne von Extended Authorization Policies, welche in diesem Artikel näher beleuchtet werden. Gleichzeitig wurde der Implementierungsaufwand für Hersteller und Anwender von TPMs reduziert.

Das Ziel der einfachen Lesbarkeit wurde schließlich der Eindeutig untergeordnet. Im Gegenzug ist es möglich aus der Spezifikation einen TPM 2.0-Simulator zu erzeugen, dessen Verhalten letztendlich als autoritative Referenz herangezogen werden kann.

In Ergänzung zur Spezifikation [2] entstand schließlich *A Practical Guide to TPM 2.0* [3] als frei verfügbares Buch um den Zugang zu Technologie und Spezifikation zu vereinfachen. Kapitel 3 dieses Buchsnennt weitere, für diesen

Artikel nicht relevante Neuerungen in TPM 2.0, die im Wesentlichen der Benutzerfreundlichkeit dienen.

Extended Authorization Policies

Grundlagen

Kern der Extended Authorization Policies ist das Hash-extend-Verfahren um eine Art revisionssicheres Protokoll zu erzeugen. Allerdings werden nicht die zu protokollierenden Daten selbst gespeichert, sondern nur ein aktueller Statuswert. Durch das verwendete Hashverfahren ist sichergestellt, dass es praktisch unmöglich ist, einen zweites Protokoll zu erzeugen, das zu demselben Statuswert führt. Ebenso ist es nicht möglich, bei vorgegebenem aktuellem Statuswert das Protokoll *(beliebig)* so fortzuschreiben, dass ein bestimmter Statuswert erreicht wird. Dies entspricht, salopp gesprochen, einer in das TPM integrierten Blockchain, wobei nur der jeweils aktuelle Block gespeichert wird. Genauer handelt es sich sogar nur um dessen Hashwert.

Das Fortschreiben dieser Hash-Kette geschieht nun folgendermaßen: Ist im TPM ein Statuswert *S* hinterlegt und sollen die Daten *D* dort mit hinterlegt werden (die Spezifikation spricht hier davon, dass *D extended* wird), so wird zuerst *D* an *S* angehängt. Mit einer Hashfunktion *hash* wird dann ein neuer Statuswert *S‘* berechnet:

$$S' \leftarrow hash(S \,||\, D)$$

Diese Statuswerte werden meist in sogenannten Platform Configuration Registers (PCRs) hinterlegt. Namensgebend war dabei der ursprüngliche und auch heute noch häufigste Einsatzzweck, nämlich die Absicherung und Protokollierung des Bootvorganges (*Secure Boot* resp. *Trusted Boot*). Die PCRs eines TPMs lassen sich nicht direkt beschreiben. Ausgehend von einem wohldefinierten Anfangswert sind sie nur durch die *extend*-Operation veränderbar. Damit ist ein Zurückstellen auf einen vorherigen Wert oder das gezielte Schreiben eines Statuswertes praktisch unmöglich.

Die *extend*-Operation selbst ist auch ein zentraler Baustein der Extended Authorization Policies.

Konzept

Um einen störungsfreien und sicheren Betrieb zu gewährleisten, prüft das TPM bei jedem erhaltenen Befehl, ob der Sender des Befehls auch zu dessen Ausführung, genauer zur Nutzung der im Befehl referenzierten Ressourcen (entities), berechtigt ist. Im einfachsten Fall geschieht dieser Nachweis über die Eingabe eines Kennwortes oder das Verwenden eines mit dem TPM vereinbarten Sitzungsschlüssels.

Extended Authorization Policies, manchmal auch als *Enhanced Authorization Policies* und oft einfach kurz als *Policies* bezeichnet, stellen eine weitere und deutlich mächtigere Möglichkeit dar, Rechte zu vergeben und zu prüfen. Die einschlägige Literatur wagt sogar die Behauptung: „Clever policy designs can allow virtually any restriction on key use that you can envision,[…]“ [3, p. 33] Dies natürlich mit der Einschränkung, dass der benötigte Implementierungsaufwand eine natürliche Schranke darstellt.

Als einfaches Beispiel kann folgendes Szenario dienen: In einem Unternehmen werden die Schlüssel für die Kommunikation per eMail von TPMs verwaltet. Erhält ein Anwender eine verschlüsselte eMail, so kann er diese vom TPM nach Eingabe seiner Passphrase entschlüsseln lassen. Im Interesse des Unternehmens soll aber auch ein Zugriff auf die eMails ohne Kenntnis dieser Passphrase möglich sein, etwa nach dem Ausscheiden des Mitarbeiters oder bei plötzlicher Erkrankung. Für diesen Fall werden Passwörter für Betriebsrat, Geschäftsführer und Administrator hinterlegt. Mittels einer Policy kann nun geregelt werden, dass der Zugriff auf das Schlüsselmaterial und damit auf die eMails ebenfalls möglich ist, wenn mindestens zwei dieser drei Passwörter korrekt eingegeben wurden. Damit sind sowohl die Interessen des Unternehmens als auch die Privatsphäre des Mitarbeiters gewahrt.

Funktionsweise

Policies attestieren einen bestimmten Systemzustand, eine bestimmte Eigenschaft oder ein bestimmtes Ereignis. Dies kann z.B. das Vorhandensein eines bestimmten Wertes oder Wertebereichs in einem PCR oder im nichtflüchtigen Speicher des TPMs sein. Aber auch komplexere Bedingungen, wie ein bestimmter Wert in einem zweiten TPM, das in einem Fingerabduckleser verbaut ist.

Der Zweck einer Policy ist dabei immer die Freigabe des Zugriffs aufs eine *entity* des TPMs, meist ein geheimer Schlüssel oder ein Bereich des nichtflüchtigen Speichers. Eine Policy ist also ein Satz von Restriktionen, die den Zugriff beschränken.

Zur Überprüfung der Restriktionen wird ein spezielles *policyDigest*-Register verwendet. Ist die Policy erfüllt, wird das *policyDigest*-Register mittels *extend*-Funktion und einem speziellen Fingerabdruck der Policy (und der Bedingung) aktualisiert. Die Freigabe der *entity* ist dann an einen bestimmten Wert des *policyDigest*-Registers gebunden. Dieser Wert muss bei der Erzeugung der *entity* bekannt sein.

Es sind dabei zwei Arten von Policies zu unterscheiden: Policies mit unmittelbarer (*immediate assertion*) Überprüfung der Bedingungen und Policies mit zurückgestellter (*deferred assertion*) Überprüfung der Bedingungen. Während erstere wie bereits beschrieben funktioniert, findet bei letzterer die Überprüfung der Bedingungen bei Zugriff auf die durch die Policy geschützte *entity* statt. Das *policyDigest* wird aber bereits zum Zeitpunkt der Vormerkung mit einem speziellen Wert aktualisiert. Die Überprüfung der Bedingungen unmittelbar beim Zugriff auf die *entity* wird dann durch das TPM sichergestellt.

Policies mit zurückgestellter Überprüfung ermöglichen das Vorverarbeiten komplexer Policy-Kombinationen, die statische Bedingungen (z.B. bestimmte Hardwarekonfigurationen) und externe/dynamische Ereignisse (z.B. bestimmte Zeiträume) koppeln. Das typischste Beispiel ist aber die einfache Autorisierung per Passwort. Hierbei fließt in den *policyDigest* lediglich ein, dass eine solche Autorisierung beim Zugriff auf die entsprechende entity zu prüfen ist, nicht das Passwort selbst. (Das TPM stellt die Überprüfung beim tatsächlichen Zugriff sicher.)

Generell ist es möglich, den Referenz-*policyDigest* offline, d.h. ohne TPM und rein in Software zu berechnen, da nie Geheimnisse in diesen Wert einfließen. In der Praxis verwendet man aber meist eine sogenannten *trial policy session*. Dabei

werden die Policies auf dem TPM in gleicher Weise wie bei der Überprüfung angewandt, allerdings wird stets die Erfüllung der Policy angenommen. Am Ende kann der benötigte *policyDigest-Wert* ausgelesen oder direkt zur Erzeugung einer entity verwendet werden.

Hierarchien
Die logische und-Verknüpfung mehrerer Policies ergibt sich trivial: Die Policies müssen lediglich nacheinander abgeprüft werden. Wichtig ist hierbei die Einhaltung der Reihenfolge. Der TPM 2.0-Standard ermöglicht aber auch die Verbindung von Policies mit einer (nicht-)exklusiven oder- Verknüpfung.

Dadurch lassen sich komplexe Hierarchien in Form von Policy-Bäumen erzeugen. Noch mehr Flexibilität ergibt sich durch die Verwendung von digitalen Signaturen. Innerhalb einer Policy-Hierarchie lässt sich die Bedingung „Policy A ist erfüllt“ ersetzen durch „Policy X ist erfüllt und es liegt eine gültige Signatur für Policy X vor.“ Damit lassen sich Policies dynamisch austauschen oder ergänzen, insbesondere wenn die konkreten Werte zur Erfüllung von Policy X bei der Erstellung der Hierarchie noch nicht bekannt sind.

Was in der theoretischen Beschreibung noch kompliziert klingt, wird schnell klar wenn ein tatsächlich implementiertes Beispiel betrachtet wird.

Beispiel: Rechte-Management
Zur Veranschaulichung der Konzepte sei nun folgendes Szenario gegeben: Für ein Fahrzeug (etwa einen Mietwagen), sollen verschiedene Benutzerrollen (Gruppen von Fahrern) definiert werden. Neben dem normalen Fahrer, der das Analogon eines Schlüssels besitzt, soll es eine alternative Möglichkeit zur Autorisierung für einen Hotelportier geben. Dieser darf das Fahrzeug zum Parken nur mit reduzierter Geschwindigkeit bewegen. Die dritte Rolle bildet einen Servicetechniker ab, der Zugriff auf erweiterte Funktonalität hat, sich dafür aber mittels Zweifaktor-Authentifizierung legitimieren muss. Unabhängig von der Rolle soll die gesamte Policy an ein Fahrzeugmerkmal gebunden sein. Dazu wird im nichtflüchtigen Speicher des TPMs das Kennzeichen oder die Fahrgestellnummern hinterlegt.

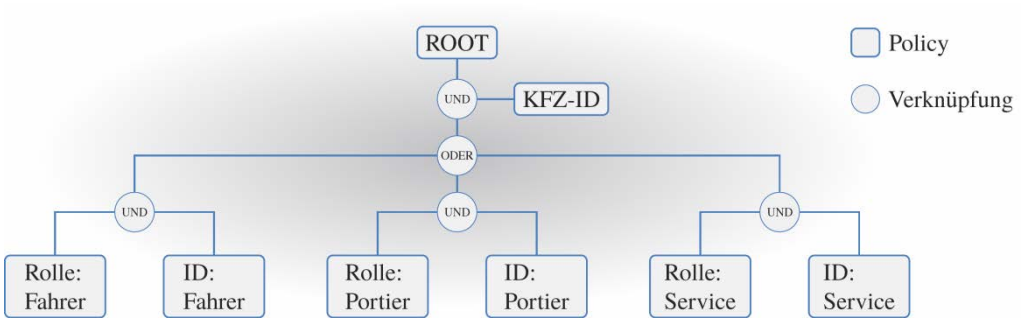

Abb. 1: Initiale Policy-Hierarchie

Policy-Hierarchie
Aus diesen Anforderungen ergibt sich initial die in Abbildung 1 dargestellte Policy-Hierarchie. Die KFZ-ID wird bei der Provisionierung des Systems im nichtflüchtigen Speicher des TPMs hinterlegt. Sei für den Moment angenommen, alle darunterliegenden Policies seien erfüllt, wird über das SAPI-Kommando *TPM_PolicyNV* geprüft ob der angegebene Bereich im NV-RAM des TPMs den gewünschten Wert enthält.

Die darunterliegende ODER-Verknüpfung kann mit dem Kommando *TPM2_PolicyOR* realisiert werden. Hierbei prüft das TPM ob der aktuelle *policyDigest* einem der in der Policy hinterlegten Werte entspricht. (Im Standard ist die Zahl der möglichen Werte auf acht begrenzt. Es lassen sich aber beliebig viele *TPM2_PolicyOR*-Bedingungen verketten.) Ist die Policy erfüllt, d.h. der aktuelle *policyDigest* entspricht einem gültigen Wert, wird die Policy *konvertiert*: Es wird ein Hashwert über die Policy berechnet und dieser in (einen frischen) *policyDigest* erweitert. Damit ergibt sich stets der gleiche Folgewert für den *policyDigest*, unabhängig davon welche Teilbedingung erfüllt ist.

Die unterste Ebene der Hierarchie kann beispielsweise durch sequentielles Ausführen von *TPM_PolicyNV* und *TPM2_PolicyAuthValue* realisiert werden.

Für den praktischen Einsatz erweist sich diese Hierarchie aber als noch nicht ausreichend flexibel, denn jede Rolle ist starr mit einer einzigen ID verknüpft. Darüber hinaus sind alternative Methoden zur Authentifizierung noch nicht explizit abgebildet.

Die Lösung findet sich in Konzept der Wildcard-Policies. Konkret wird in der initialen Hierarchie jede Benutzer-ID durch eine solche Wildcard-Policy ersetzt. Mittels *TPM2_PolicyAuthorize* wird ein (Fingerabruck) des öffentlichen Teils eines asymmetrischen Schlüsselpaares in den *policyDigest* erweitert. An dieser Stelle kann nun jede Policy eingesetzt werden, zu derem abschließenden Wert eine gültige Signatur mit diesem Schlüsselpaar vorliegt. Die Überprüfung der Policy-Hierarchie erfolgt dann in zwei Schritten, zuerst wird die Signatur durch das TPM überprüft. Das TPM erzeugt daraufhin ein Ticket mit dem es sich selbst gegenüber die erfolgreiche Signaturprüfung attestiert. Damit wird, bildlich gesprochen, die konkrete Policy anstelle der Wildcard-Policy in die Policy-Hierarchie eingehängt. Schlussendlich ergibt sich die Hierarchie aus Abbildung 2. Jede der dort gestrichelt markierten Sub-Hierarchien kann mehrfach existieren. Damit können dynamisch Benutzer-IDs für die verschiedenen Rollen aktiviert und deaktiviert werden.

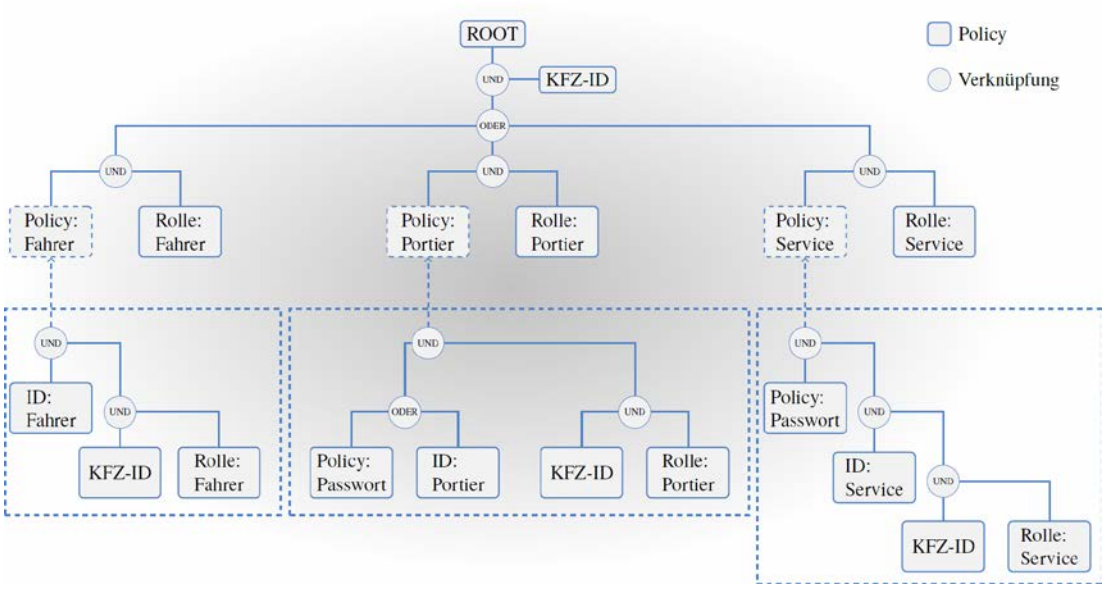

Abb. 2: Vollständige Policy-Hierarchie

Umsetzung
Die Erzeugung und Überprüfung der vorgestellten Policy-Hierarchie wurde bei MixedMode sowohl mit Skripten auf einem RaspberryPi, als auch für die hauseigene M2Control Demonstratorplattform als QT-App umgesetzt. Als TPM-Modul kam dabei in beiden Fällen ein Infineon SLB 9670-Baustein zum Einsatz, der per SPI angebunden wurde.

Die Policies werden durch die Demo direkt per *trial-session* auf dem TPM generiert (Abbildung 3), das Schlüsselpaar für die Wildcard-Policies wird mit OpenSSL erzeugt. Über die QT-Oberfläche lassen sich schließlich User-IDs registrieren und Authentifizierungen durchführen.

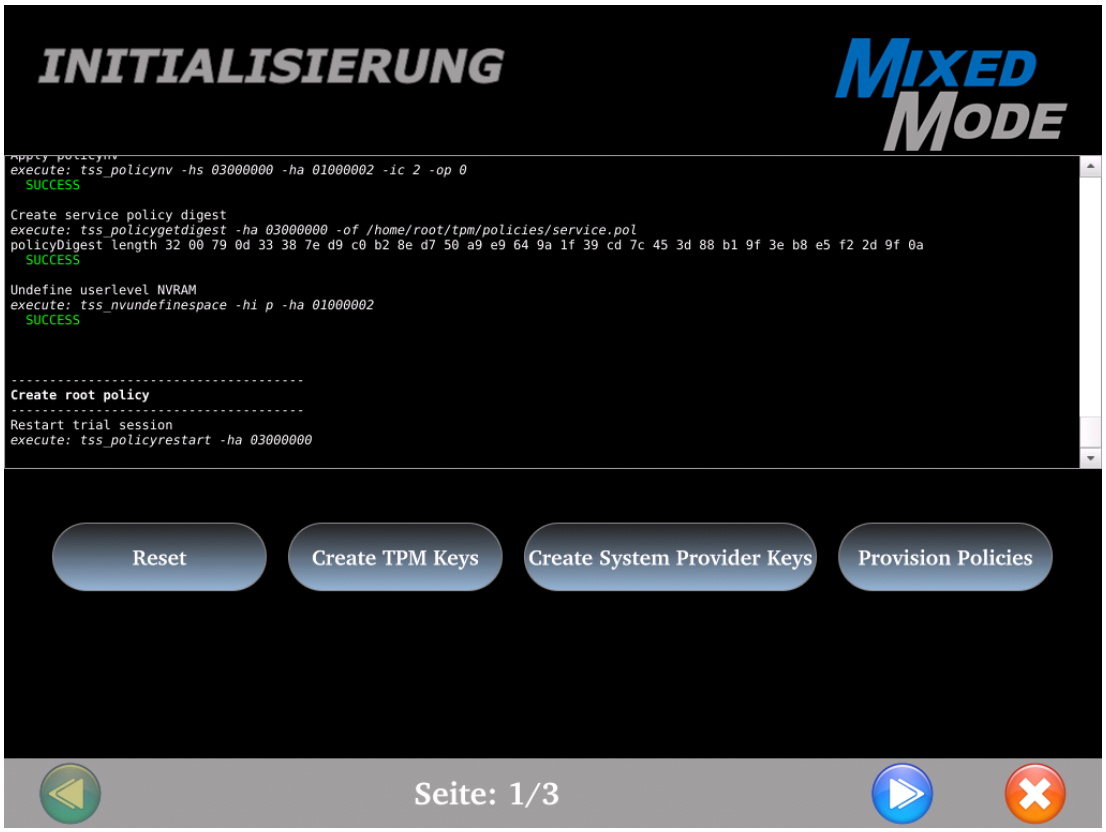

Abb. 3: Erzeugen der Policy-Hierarchie auf dem Demonstrator

Ausblick: Weitere Einsatzszenarien
Das vorgestellte Beispiel lässt sich leicht auf ein allgemeineres Rechtemanagement übertragen. Dies kann für ein reines Software-Lizenzmanagement (Freigabe von Funktionalitäten on-demand und im pay-per-use-Modell) bis zu einem integrierten Produktionsmanagement (Lizensierung von Produktionsdaten, Limitierung produzierter Einheiten, Echtheitszertifizierung etc.) ausgebaut werden, da Policies auch einfache numerische Vergleichsoperationen und über das TPM realisierte irreversible Zähler umfassen können.

Extended Authorizations ermöglichen auch, Policies zu zertifizieren. Damit lässt sich etwa das Vertrauen in Signaturschlüssel erhöhen, da das TPM zusätzlich zur Signatur auch einen kryptographischen Nachweis darüber liefern kann, welche Form der Zugriffskontrolle auf den Signaturschlüssel realisiert ist.

Schlussendlich ermöglichen Wildcard-Policies ein sicheres Systemupdate. Sie liefern die benötigte Indirektion um eine *TPM_PolicyPCR* ohne temporären Bruch der Vertrauenskette zu aktualisieren. Ein auf diese Weise realisiertes SecureBoot/TrustedBoot-Konzept kann dann sogar mit der vorgestellten Lösung kombiniert werden.

Zusammenfassung
Mit der Einführung von Policies wurde ein mächtiges Werkzeug geschaffen, um komplexe und gleichzeitig flexible Rechtemanagement-Architekturen auf Basis von TPM 2.0 Modulen zu realisieren.

Die weite Verbreitung der Module und standardisierte Programmierschnittstellen erlauben eine schnelle, sichere und nachhaltige Umsetzung eines Rechtemanagements in verschiedensten Einsatzszenarien.
Die Verknüpfung verschiedener Arten von Teil-Policies zu einer komplexen Policy wurde am Beispiel des Mixed Mode M2Control Embedded Demonstrators aufgezeigt. Hier wurden an einem (fiktiven) Fahrzeug Funktionalität auf Basis von Benutzerrollen freigegeben. Die Authentifizierung der einzelnen Benutzer war dabei für jede für jede Rolle individuell gestaltet.

Literatur- und Quellenverzeichnis

[1] O. Bünte, „heise online,“ 05 10 2018. [Online]. Available: https://heise.de/-4181988. [Zugriff am 12 10 2018].

[2] Trusted Computing Group, TPM Library Specification 2.0, 1. Oktober 2014.

[3] W. Arthur, D. Challener und K. Goldman, A Practical Guide to TPM 2.0, Apress, 2015.

Danksagung

Ein großer Dank gebührt Benedikt Petschkuhn, der das Praxisbeispiel entworfen und initial für den Raspberry Pi implementiert hat.

Autor

Markus Wamser arbeitet als Systementwickler und Consultant mit dem Schwerpunkt Embedded Security bei Mixed Mode in Gräfelfing bei München. Zuvor hat er den Lehrstuhl für Sicherheit in der Informationstechnik an der Technischen Universität München von seiner Gründung an als Mitarbeiter begleitet.
Markus Wamser war Vortragender auf internationalen Konferenzen von Abu Dhabi bis Verona, hat mehr als zehn Jahre Erfahrung in der universitären Lehre und besitzt je ein Diplom in Mathematik und Informatik.

Kontakt

Internet: https://www.mixed-mode.de/
Email: markus.wamser@mixed-mode.de