Κατανεμημένο Σύστημα Απομακρυσμένης Επιβεβαίωσης Εμπιστεύσιμου Υλικολογισμικού

Κωνσταντίνος Τσαμπίρας

Διπλωματική Εργασία

Επιβλέπων: Βασίλειος Τενέντες

Ιωάννινα, Μάρτιος 2024

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Ευχαριστίες

Η παρούσα διπλωματική εργασία πραγματοποιήθηκε στο Πανεπιστήμιο Ιωαννίνων, στο τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής. Η ολοκλήρωση αυτής θα ήταν αδύνατη χωρίς την υποστήριξη του καθηγητή μου, Επίκουρου Καθηγητή του Π.Ι., Κου Βασίλειου Τενέντε. Του εκφράζω τις θερμές ευχαριστίες μου για την πολύτιμη βοήθειά του και την καλή συνεργασία μας.

Θα ήθελα επίσης να ευχαριστήσω την οικογένεια μου, που είναι δίπλα μου σε κάθε μου βήμα, και χωρίς αυτούς δεν θα ήμουν στο σημείο που βρίσκομαι τώρα καθώς επίσης και τους φίλους μου για την αμέριστη συμπαράσταση και βοήθεια όλα αυτά τα χρόνια.

3/3/2024

Τσαμπίρας Κωνσταντίνος

Περίληψη

Αυτή η διπλωματική αντιμετωπίζει την πρόκληση της απομακρυσμένης επιβεβαίωσης εμπιστεύσιμου υλικολογισμικού για συσκευές του διαδικτύου των πραγμάτων (IoT) χωρίς την ανάγκη μιας κεντρικής αρχής. Οι υπάρχουσες λύσεις έχουν περιοριστεί λόγω της εξάρτησής τους από κεντρικά συστήματα, τα οποία μπορεί να είναι ευάλωτα σε μεμονωμένα σημεία αποτυχίας και επιθέσεων. Προτείνεται ένα νέο Κατανεμημένο Σύστημα Απομακρυσμένης Επιβεβαίωσης Εμπιστεύσιμου Υλικολογισμικού. Τα ευρήματα μας δείχνουν ότι με την κατανεμημένη προσέγγιση επιτυγχάνεται η απομακρυσμένη επιβεβαίωση περιορίζοντας στο ελάχιστο την ανάγκη για κάποια κεντρική αρχή. Υλοποιώντας το προτεινόμενο κατανεμημένο σύστημα απομακρυσμένης επιβεβαίωσης σε έμπιστες συσκευές διαφόρων αρχιτεκτονικών CPU όπως ARM, x86 και RISCV, τόσο σε πραγματικές συσκευές όπως Raspberry Pi 3B (RPI3B) και Raspberry Pi Zero W, όσο και σε εικονικές συσκευές όπως Azure VM, WSL2 σε PC και QEMU RISCV Emulator σε PC καθώς και με την χρήση μίας πραγματικής μονάδας TPM (Infineon OPTIGA TPM2.0) για την μη έμπιστη συσκευή και εικονικών μονάδων TPM (IBM Software TPM) για τις έμπιστες συσκευές, προκύπτουν χρόνοι CPU για την επιβεβαίωση μίας μη έμπιστης συσκευής κάτω των 70 ms στις x86 αρχιτεκτονικές, κάτω των 625 ms για τις ARM αρχιτεκτονικές, και 2400 ms για την RISCV αρχιτεκτονική, ενώ η μη έμπιστη συσκευή (RPI3B) απαιτεί έναν μέσο χρόνο CPU που ανέρχεται στα 600 ms ανά επιβεβαίωση με μία έμπιστη συσκευή. Συγκρίνοντας την κατανεμημένη υλοποίηση με μία αντίστοιχη κεντροποιήμενη στο Raspberry Pi 3B προκύπτει ότι η κεντροποιημένη υλοποίηση απαιτεί μέχρι και τρεις φορές περισσότερο χρόνο CPU για να ολοκληρώσει την διαδικασία σε σχέση με την κατανεμημένη.

Λέξεις Κλειδιά: Κατανεμημένη Απομακρυσμένη Επιβεβαίωση, TPM, IoT

Abstract

This thesis addresses the challenge of remote attestation for trustworthy software on Internet of Things (IoT) devices without the need for a central authority. Existing solutions have been limited by their dependence on central systems, which may be vulnerable to single points of failure and attacks. A new Distributed Remote Attestation System for Trustworthy Firmware is proposed. The findings show us that with a distributed approach, remote attestation is achieved by minimizing the need for any central authority. By implementing the proposed distributed attestation system on trusted devices of various CPU architectures such as ARM, x86, and RISCV, both on real devices like Raspberry Pi 3B (RPI3B) and Raspberry Pi Zero W, and on virtual devices like Azure VM, WSL2 on PC, and QEMU RISCV Emulator on PC, using a real TPM unit (Infineon OPTIGA TPM2.0) for the untrusted device and virtual TPM units (IBM Software TPM) for the trusted devices, CPU times for attesting an untrusted device are below 70 ms on x86 architectures, below 625 ms for ARM architectures, and 2400 ms for RISCV architecture, while the untrusted device (RPI3B) requires an average CPU time of 600 ms per attestation with a trusted device. Comparing the distributed implementation with a corresponding centralized one on Raspberry Pi 3B reveals that the centralized implementation requires up to three times more CPU time to complete the process compared to the distributed one.

Keywords: Distributed Remote Attestation, TPM, IoT

Πίνακας Περιεχομένων

ΚΑΤΆΛΟΓΟΣ ΕΙΚΌΝΩΝ

Κεφάλαιο 1. Εισαγωγή

1.1 Internet of Things

Το Internet of Things (IoT), ορίζεται [1] ως ένα κατανεμημένο δίκτυο ετερογενών συσκευών (αισθητήρες, ενεργοποιητές, επεξεργαστές) όπου ανταλλάσσουν μηνύματα μεταξύ τους χωρίς να είναι απαραίτητη η ανθρώπινη παρέμβαση. Αποτελεί μία ταχύτατα αναπτυσσόμενη τεχνολογική εξέλιξη, με νέες εφαρμογές να εμφανίζονται συνεχώς. Οι συσκευές αυτές χρησιμοποιούνται στην βιομηχανία αλλά και καθημερινότητα μας μέσα από απλές συσκευές που χειριζόμαστε απομακρυσμένα, είτε εντός του τοπικού δικτύου μας είτε μέσω διαδικτύου. Η άμεση πρόσβαση σε νέες τέτοιες τεχνολογίες έχει δώσει έδαφος σε ένα πλήθος απλών συσκευών να γίνουν «έξυπνες» με την ενσωμάτωση τεχνολογιών όπως το WiFi και το Bluetooth, με στόχο τον απομακρυσμένο χειρισμό τους.

Οι δυνατότητες είναι απεριόριστες, με νέες ιδέες και υλοποιήσεις να παρουσιάζονται συνεχώς, όλες με στόχο να βελτιώσουν την ζωή μας, την καθημερινότητα μας αλλά και σε βιομηχανικά και επαγγελματικά περιβάλλοντα να ελαχιστοποιήσουν το κόστος και να μεγιστοποιήσουν το κέρδος συγκεντρώνοντας δεδομένα για την βελτιστοποίηση διαδικασιών και την πρόβλεψη μελλοντικών αναγκών.

Μερικές από τις εφαρμογές συσκευών IoT περιλαμβάνουν, έξυπνους αισθητήρες παρακολούθησης υγείας, αισθητήρες για έξυπνα σπίτια και έξυπνες πόλεις καθώς και για διασυνδεδεμένα οχήματα.

1.2 Edge Computing και εφαρμογές IoT

Με τον όρο Edge Computing αναφερόμαστε [2] στην επεξεργασία δεδομένων κοντά στο σημείο συλλογής τους. Αποτελεί ένα σχετικά πρόσφατο φαινόμενο στον χώρο της πληροφορικής και διακρίνεται για την γρήγορη επεξεργασία και τον άμεσο χρόνο απόκρισης προσφέροντας πολλά πλεονεκτήματα όπως ταχύτερη και αποδοτικότερη επεξεργασία δεδομένων, ασφάλεια και λιγότερη συμφόρηση στα ήδη υπάρχοντα δίκτυα. Ταυτόχρονα, εφαρμογές ευαίσθητες στις καθυστερήσεις μπορούν να επωφεληθούν από τον χαμηλό λανθάνων χρόνο και την ευελιξία του edge computing.

Με τον όλο και αυξανόμενο αριθμό υπηρεσιών που μας προσφέρουν οι συσκευές IoT, παρουσιάζονται και άλλες ανάγκες, μερικές εκ των οποίων είναι ο χαμηλός λανθάνων χρόνος, η αποτελεσματική χρήση των διαθέσιμων πόρων και ο τρόπος επεξεργασίας των δεδομένων. Οι ανάγκες αυτές εκπληρώνονται από τεχνολογίες όπως το edge computing, με συσκευές που βρίσκονται κοντά στις IoT συσκευές και αναλαμβάνουν το ρόλο του επεξεργαστή των δεδομένων και του αναμεταδότη της πληροφορίας στο δίκτυο.

Οι συσκευές που αποτελούν μέρος του edge computing για ένα υποσύνολο συσκευών IoT, είναι επί της ουσίας πύλες για την επικοινωνία και την πληροφορία που παρέχουν οι IoT συσκευές. Για τον λόγο αυτό είναι υψίστης σημασίας η ασφάλεια και η προστασία των συσκευών αυτών τόσο από εξωτερικές παρεμβάσεις όσο και από διαχειριστές με αυξημένα δικαιώματα επάνω στην edge computing συσκευή. Ένας από τους τρόπους που μπορούμε να το πετύχουμε αυτό αποτελεί και η Απομακρυσμένη Επιβεβαίωση (Remote Attestation) και το Έμπιστο Υλικολογισμικό.

1.3 Έμπιστο Υλικολογισμικό

Το έμπιστο υλικολογισμικό αποτελεί ένα κρίσιμο σημείο για την ασφάλεια και την ακεραιότητα ενσωματωμένων συστημάτων. Πιο συγκεκριμένα, το έμπιστο υλικολογισμικό, αναφέρεται σε υλικολογισμικό, του οποίου ο κώδικας, έχει δημιουργηθεί από μία αξιόπιστη πηγή και δεν έχει υποστεί αλλοιώσεις ή μη εξουσιοδοτημένες τροποποιήσεις. Ο κώδικας αυτός υπογράφεται ψηφιακά και επιβεβαιώνεται εύκολα πως είναι έμπιστος μέσω κρυπτογραφικών διαδικασιών καθιστώντας αδύνατη την πλαστογραφία. Με επιβεβαίωση της ψηφιακής υπογραφής στο στάδιο της εκκίνησης [3], μπορούμε να ενημερώσουμε τον χρήστη για τυχόν μη επαλήθευση της υπογραφής με προτροπή για να μην εκκινηθεί καθόλου ο κώδικας αυτός ή ακόμα και με ρητή άρνηση εκκίνησης μη έμπιστου υλικολογισμικού.

1.4 Απομακρυσμένη Επιβεβαίωση (Remote Attestation)

Η απομακρυσμένη επιβεβαίωση [15] είναι ένας μηχανισμός που επιτρέπει σε τρίτα εξουσιοδοτημένα άτομα ή συσκευές να επιβεβαιώσουν την ακεραιότητα και την εμπιστοσύνη της συσκευής και του λογισμικού απομακρυσμένα, συνήθως με χρήση μίας Μονάδας Έμπιστης Πλατφόρμας (Trusted Platform Module, TPM).

Ένα από τα βασικότερα χαρακτηριστικά της TPM είναι το κλειδί επικύρωσης (Endorsement Key, EK), ένα ζεύγος δημόσιου-ιδιωτικού κλειδιού που δημιουργήθηκε κατά την κατασκευή της TPM. Το ιδιωτικό κλειδί παραμένει ασφαλές εντός της TPM, καθώς δεν εκτίθεται ποτέ σε κανένα άλλο στοιχείο, λογισμικό ή χρήστη, και το δημόσιο κλειδί χρησιμοποιείται για την επιβεβαίωση της ψηφιακής υπογραφής των δεδομένων.

Για την απομακρυσμένη επιβεβαίωση, έχουμε δύο κύριους ρόλους συσκευών: τον attester, αυτού που προσπαθεί να επιβεβαιώσει την εμπιστευσιμότητά του παρουσιάζοντας αποδείξεις για τον ίδιο και τον verifier/relying party, ο οποίος εξετάζει και επιβεβαιώνει την εμπιστευσιμότητα του attester βασιζόμενος σε κάποια πολιτική επιβεβαίωσης. Ο attester δημιουργεί αποδείξεις για την συμπεριφορά, τις ρυθμίσεις και το λογισμικό του, χρησιμοποιώντας την TPM ως ρίζα εμπιστοσύνης για τις μετρήσεις που συλλέγει και τις υπογράφει με το ιδιωτικό κλειδί του. Τις παρουσιάζει στον verifier/relying party και με βάση τα δεδομένα που διαθέτει για τον

attester ο verifier/relying party επιβεβαιώνει την εγκυρότητα της απόδειξης αυτής, συνήθως με την χρήση του κλειδιού επιβεβαίωσης (Attestation Key, AK), ενός κλειδιού που έχει παραχθεί με βάση το EK και μπορεί να επαληθεύσει ότι τα δεδομένα που παρουσίασε ο attester είναι αυθεντικά και έμπιστα. Εφόσον ο verifier/ relying party επιβεβαιώσει τα στοιχεία του attester, θεωρεί τον attester έμπιστο.

Οι χρήσεις του συγκεκριμένου μηχανισμού, εντοπίζονται στην ακεραιότητα κώδικα από μη εξουσιοδοτημένες αλλαγές, την διαχείριση πνευματικών δικαιωμάτων (Digital Rights Management, DRM) θέτοντας περιορισμούς κατά της μη ενδεδειγμένης χρήσης, την ακεραιότητα εικονικών μηχανών και container σε περιβάλλοντα νέφους καθώς και την επιβεβαίωση της εμπιστοσύνης IoT συσκευών πριν την εκχώρηση δικαιωμάτων για συμμετοχή σε ένα έμπιστο δίκτυο συσκευών. Η απομακρυσμένη επιβεβαίωση μπορεί να υλοποιηθεί με κεντροποιημένους τρόπους αλλά και κατανεμημένους. Οι κεντροποιημένοι τρόποι βασίζονται στην φιλοσοφία πως [4], [16] υπάρχει ένας έμπιστος ελεγκτής (verifier), που είναι υπεύθυνος για την επιβεβαίωση μίας συσκευής του IoT δικτύου (attester/prover). Ενώ οι κατανεμημένοι τρόποι, απαιτούν επιβεβαίωση από περισσότερες από μία συσκευές, συνήθως αντίστοιχων δυνατοτήτων με την συσκευή που ζητά την επιβεβαίωση και χωρίς την ύπαρξη κάποιας κεντρικής αρχής, με την χρήση μοναδικών χαρακτηριστικών της συσκευής ως ρίζα εμπιστοσύνης [5], [6].

1.5 Στόχος και δομή της εργασίας

Ενώ υπάρχουν Κεντροποιημένα Συστήματα Απομακρυσμένης Επιβεβαίωσης Εμπιστεύσιμου Υλικολογισμικού (ΑΕΕΥ), αυτά εναποθέτουν απόλυτη εμπιστοσύνη σε έναν κεντρικό επαληθευτή, δημιουργώντας προβλήματα αν αυτός παραβιαστεί ή αποτύχει. Αυτό δημιουργεί την ανάγκη για έναν κατανεμημένο τρόπο ΑΕΕΥ. Στην παρούσα εργασία αναπτύσσεται ένα πειραματικό Κατανεμημένο Σύστημα ΑΕΕΥ, για συστήματα ακροδικτυακής υπολογιστικής (edge computing) βασισμένο σε Python. Πιο συγκεκριμένα, ακολουθεί η δομή της εργασίας.

Στο Κεφάλαιο 2, γίνεται μία εισαγωγή σχετικά με τον τρόπο λειτουργίας της απομακρυσμένης επιβεβαίωσης, αναλύονται οι τεχνολογίες που χρησιμοποιούνται στην εργασία αυτή, όπως η TPM που συνιστά την ρίζα εμπιστοσύνης για κάθε συσκευή και η ασφαλής-μετρήσιμη εκκίνηση. Αναλύεται επίσης το Κεντροποιημένο Σύστημα Απομακρυσμένης Επιβεβαίωσης Εμπιστεύσιμου Υλικολογισμικού (ΑΕΕΥ) και το Κατανεμημένο Σύστημα ΑΕΕΥ, ο τρόπος λειτουργίας τους και υπάρχουσες υλοποιήσεις τους καθώς και ιδιαιτερότητες των κατανεμημένων συστημάτων.

Στο Κεφάλαιο 3, αναλύεται περισσότερο το Κατανεμημένο Σύστημα ΑΕΕΥ, συγκεκριμένα θα παρουσιαστούν: η αρχιτεκτονική του πρωτότυπου Κατανεμημένου Συστήματος ΑΕΕΥ, οι τεχνικές λεπτομέρειες της υλοποίησης του πρωτότυπου, το πρωτόκολλο επικοινωνίας που χρησιμοποιεί το πρωτότυπο και σε μεγαλύτερο βάθος τα στάδια του πρωτόκολλου καθώς και περαιτέρω επεκτάσεις που μπορούν να γίνουν στο συγκεκριμένο πρωτότυπο.

Στο Κεφάλαιο 4, παρουσιάζονται πειραματικές μετρήσεις χρόνου προκειμένου να αξιολογηθεί το παραπάνω πρωτότυπο μέσα από διάφορα σενάρια και πειραματικές διατάξεις.

Και στο Κεφάλαιο 5, εκφράζονται τα τελικά συμπεράσματα.

Κεφάλαιο 2. Υπόβαθρο

Στο κεφάλαιο αυτό, γίνεται μία εισαγωγή σχετικά με τον τρόπο λειτουργίας της απομακρυσμένης επιβεβαίωσης, αναλύονται οι τεχνολογίες που χρησιμοποιούνται στην εργασία αυτή, όπως η TPM που συνιστά την ρίζα εμπιστοσύνης για κάθε συσκευή και η ασφαλής-μετρήσιμη εκκίνηση. Αναλύεται επίσης το Κεντροποιημένο Σύστημα Απομακρυσμένης Επιβεβαίωσης Εμπιστεύσιμου Υλικολογισμικού (ΑΕΕΥ) και το Κατανεμημένο Σύστημα ΑΕΕΥ, ο τρόπος λειτουργίας τους και υπάρχουσες υλοποιήσεις τους καθώς και ιδιαιτερότητες των κατανεμημένων συστημάτων.

2.1 Εισαγωγή

Με την Απομακρυσμένη Επιβεβαίωση [15] (Remote Attestation) επιβεβαιώνεται η αξιοπιστία μιας απομακρυσμένης συσκευής χρησιμοποιώντας κρυπτογραφικές αποδείξεις που μπορεί να δημιουργήσει η απομακρυσμένη συσκευή μόνο αν δεν έχει υποστεί αλλοιώσεις στο υλικό/λογισμικό της. Ο βασικός τρόπος λειτουργίας της απομακρυσμένης επιβεβαίωσης είναι ο εξής:

- Κατά την διάρκεια εκκίνησης ενός συστήματος, λαμβάνονται μετρήσεις για τον κώδικα, το λειτουργικό σύστημα και άλλα κρίσιμα υποσυστήματα, Αυτές οι μετρήσεις μπορούν να ταυτοποιήσουν μοναδικά μία πλατφόρμα. Η Αρχιτεκτονική Μετρήσιμης Ακεραιότητας (Integrity Measurement Architecture, IMA) χρησιμοποιεί αυτές τις μετρήσεις και τις «επεκτείνει» σε έναν Καταχωρητή Διαμόρφωσης Πλατφόρμας (Platform Configuration Register, PCR) που βρίσκεται εντός της μονάδας TPM και οποιαδήποτε αλλοίωση στα μετρούμενα δεδομένα θα προκαλέσει και αλλαγή στην τιμή αυτού του καταχωρητή.

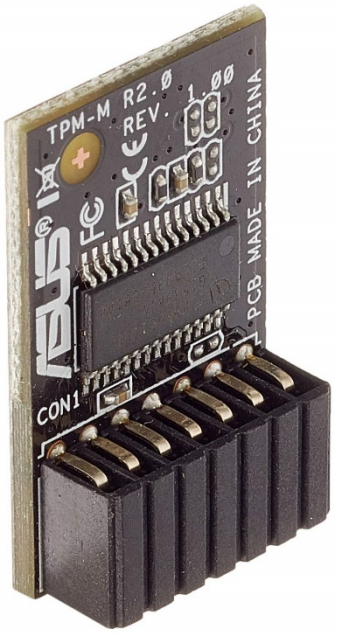
- Ο remote verifier ζητά τις μετρήσεις ενός συστήματος για να επαληθεύσει την ακεραιότητα του συστήματος ή λαμβάνει αίτημα από ένα σύστημα για να πραγματοποιήσει σε εκείνο απομακρυσμένη επιβεβαίωση.
- Ο remote verifier λαμβάνει τα αποτελέσματα των μετρήσεων αυτών μέσω κρυπτογραφημένου καναλιού και τις αναλύει προκειμένου να επιβεβαιώσει την ακεραιότητα του απομακρυσμένου συστήματος.
- Αν ο attester επιβεβαιωθεί επιτυχώς, τότε μπορεί να αποκτήσει πρόσβαση σε υπηρεσίες και δικαιώματα πάνω στο δίκτυο στο οποίο βρίσκεται, ενώ σε διαφορετική περίπτωση δεν δίνονται δικαιώματα και ο remote verifier μπορεί να ενημερώσει το δίκτυο ότι μία συσκευή δεν πέρασε την διαδικασία επιβεβαίωσης.

Περιβάλλοντα στα οποία είναι απαραίτητο ένα υψηλό επίπεδο εμπιστοσύνης, όπως δίκτυα συσκευών IoT, υπολογιστικά νέφη και υπηρεσίες που διαχειρίζονται ευαίσθητα ή/και εμπιστευτικά δεδομένα, χρησιμοποιούν εξιδεικευμένες τεχνικές ασφαλείας όπως είναι η απομακρυσμένη επιβεβαίωση. Παρακάτω αναλύονται μερικές από τις βασικές έννοιες οι οποίες αποτελούν αναπόσπαστο κομμάτι της απομακρυσμένης επιβεβαίωσης και βάση για το πρωτότυπο.

2.1.1 Μονάδα Έμπιστης Πλατφόρμας

Η Μονάδα Έμπιστης Πλατφόρμας (Trusted Platform Module, TPM), ή μονάδα TPM είναι μία συσκευή ή δομή εντός του επεξεργαστή, η οποία ενσωματώνεται σε όλο ένα και περισσότερα σύγχρονα συστήματα υπολογιστών. Κύρια αρμοδιότητα της είναι η δημιουργία και η ασφαλής αποθήκευση κρυπτογραφικών κλειδιών, η επιβεβαίωση της ακεραιότητας του λειτουργικού συστήματος και του υλικολογισμικού κατά την εκκίνηση και η παροχή ενός έμπιστου περιβάλλοντος για την προστασία εμπιστευτικών πληροφοριών και κρυπτογραφικών διαδικασιών.

Εικόνα 1: Infineon OPTIGA SLB9670 για Raspberry Pi



Εικόνα 2: dTPM για υπολογιστή

Η TPM αποτελεί την βάση για πολλές τεχνολογίες και μηχανισμούς σχετικών με την ασφάλεια και την εμπιστοσύνη υπολογιστών και ενσωματωμένων συσκευών, και αποτελεί και κρίσιμο στοιχείο για την αρχιτεκτονική του πρωτοτύπου όπου εκεί η TPM είναι υπεύθυνη για την Μετρήσιμη Εκκίνηση του συστήματος και για την ΑΕΕΥ.

Οι [7] Καταχωρητές Διαμόρφωσης Πλατφόρμας (Platform Configuration Registers, PCR) είναι ένα κρίσιμο στοιχείο της TPM. Οι PCR είναι μία μνήμη που υπάρχει εντός της TPM, με κάποιες ιδιαίτερες ιδιότητες που την καθιστούν ιδανική για χρήσεις που έχουν να κάνουν με την ασφάλεια ενός συστήματος. Η κυριότερη χρήση τους είναι να παρέχουν μία μέθοδο για καταγραφή της κατάστασης του λογισμικού (εκτελέσιμων αρχείων και ρυθμίσεων που χρησιμοποιούν) χρησιμοποιώντας την κρυπτογραφία. Η ενημέρωση ενός PCR καταχωρητή απαιτεί την «επέκταση», δηλαδή το πέρασμα από μία συνάρτηση κατακερματισμού, έτσι ώστε να μην είναι δυνατή η απομάκρυνση κάποιας προηγούμενης μέτρησης. Υπάρχουν συνήθως πολλαπλά σύνολα καταχωρητών (banks) όπου κάθε σύνολο υποστηρίζει μία συγκεκριμένη συνάρτηση κατακερματισμού.

Η «επέκταση» ενός PCR [8] γίνεται με την ένωση (concatenate) της τρέχουσας τιμής του PCR με τα δεδομένα που πρέπει να «επεκταθούν» και από αυτό να παραχθεί το digest τους, (συνήθως αντί να ενωθούν απευθείας τα δεδομένα, υπολογίζεται πρώτα το digest των δεδομένων προς επέκταση και ενώνεται αυτό με την προηγούμενη τιμή του PCR).

$$PCR_{new} = \boldsymbol{H}_{alg}(PCR_{old}||digest)$$

Όπου

$\boldsymbol{H}_{alg}$	Η συνάρτηση κατακερματισμού που χρησιμοποιεί ο PCR
PCR_{new}	Η νέα τιμή που θα πάρει ο καταχωρητής
PCR_{old}	Η προηγούμενη τιμή του καταχωρητή
$digest$	Τα δεδομένα προς επέκταση (μετά από το πέρασμά τους από συνάρτηση κατακερματισμού)

Οι PCR μπορούν στην συνέχεια να αναγνωστούν και να αναφέρουν την τρέχουσα τιμή τους ή για υπογραφθούν κατάλληλα και να επιστρέψουν μία πιο ασφαλή αναφορά, ένα quote, που μπορεί να χρησιμοποιηθεί για την Απομακρυσμένη Επιβεβαίωση της Πλατφόρμας.

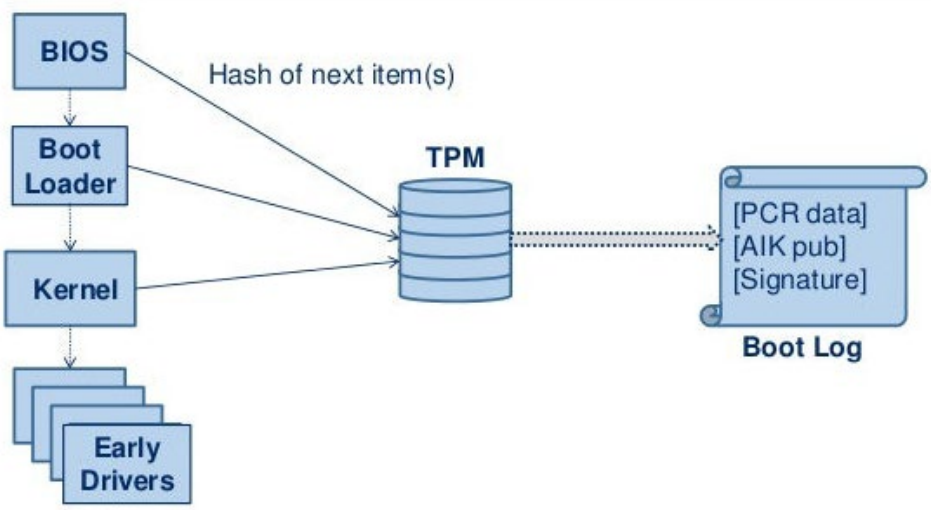
2.1.2 Ασφαλής - Μετρήσιμη Εκκίνηση

Οι τεχνολογίες ασφαλούς εκκίνησης και μετρήσιμης εκκίνησης συμβάλλουν στην ασφάλεια της διαδικασίας εκκίνησης του bootloader και του λειτουργικού συστήματος ενός υπολογιστή/πλατφόρμας, είτε αποτρέποντας είτε εντοπίζοντας προσπάθειες επίθεσης που θα μπορούσαν να λάβουν χώρα προτού εκκινήσει ο bootloader.

Η Ασφαλής Εκκίνηση έχει ως βασικό στόχο να εξασφαλίσει πως ο κώδικας ο οποίος πρόκειται να εκτελεστεί κατά την διαδικασία εκκίνησης του υπολογιστή είναι υπογεγραμμένος και έμπιστος. Η λειτουργία του βασίζεται στην επιβεβαίωση ψηφιακών υπογραφών του κώδικα που επρόκειτο να εκτελεστεί (όπως του bootloader για την εκκίνηση του λειτουργικού συστήματος) με δημόσια κλειδιά που βρίσκονται προεγκατεστημένα στον εκκινητή, εξασφαλίζοντας με αυτόν τον τρόπο ότι ο κώδικας είναι αυθεντικός και χωρίς τροποποιήσεις.

Η Μετρήσιμη Εκκίνηση, αποτελεί έναν διαφορετικό τρόπο ασφαλούς εκκίνησης, όπου το σύστημα μετρά την ακεραιότητα του λειτουργικού συστήματος κατά την διάρκεια της εκκίνησης λαμβάνοντας μετρήσεις για να τις χρησιμοποιήσει σε μελλοντικές αναφορές. Οι μετρήσεις αυτές αποτελούνται από τα αποτελέσματα μίας συνάρτησης κατακερματισμού για διάφορες δομές του λειτουργικού συστήματος που εκκινήθηκε, παρέχοντας έτσι ένα αντικειμενικό ιστορικό για τον τρόπο που εκκινήθηκε και έτσι οποιαδήποτε απόπειρα τροποποίησης του κώδικα εκκίνησης ή του

λειτουργικού συστήματος κατά την διαδικασία της εκκίνησης θα είναι εύκολο να εντοπιστεί αργότερα.



Εικόνα 3: Τρόπος Λειτουργίας Μετρήσιμης Εκκίνησης

Πολλά συστήματα υποστηρίζουν ένα συνδυασμό των παραπάνω τρόπων, διασφαλίζοντας ότι εκκινεί ένα έμπιστο λειτουργικό σύστημα και ένας έμπιστος bootloader καθώς και ότι δεν έχει υπάρξει κάποια απόπειρα τροποποίησης κατά την εκκίνηση, πράγμα που εντοπίζεται εύκολα κοιτώντας τις μετρήσεις εκκίνησης για μη εγκεκριμένες καταχωρήσεις.

2.1.2.1 Αρχιτεκτονική Μετρήσιμης Ακεραιότητας

Η Αρχιτεκτονική Μετρήσιμης Ακεραιότητας (Integrity Measurement Architecture, IMA) αποτελεί ένα βασικό χαρακτηριστικό ασφαλείας σε συστήματα Linux με βασική αρμοδιότητα [9] την συλλογή hash (μετρήσεων) από αρχεία συστήματος και την διασφάλιση της ακεραιότητας τους. Λειτουργεί σε επίπεδο πυρήνα και δεν μπορεί να αλλοιωθεί από το επίπεδο χρήστη. Ο τρόπος λειτουργίας του βασίζεται στη συλλογή των hash από αρχεία συστήματος, την τοποθέτηση τους στη μνήμη πυρήνα και την ανάγνωση από τρίτους, όταν αυτό ζητηθεί, των τιμών καταγραφής για την επιβεβαίωση των τιμών που μετρήθηκαν κατά την εκκίνηση.

```
10 d4db[…] ima-sig sha1:9797[…] boot_aggregate
10 7ec8[…] ima-sig sha256:f485[…] /lib/systemd/systemd
10 c55e[…] ima-sig sha256:b426[…] /lib/arm-linux-gnueabihf/ld-2.28.so
10 574a[…] ima-sig sha256:7201[…] /lib/[…]/systemd-bless-boot-generator
10 14a7[…] ima-sig sha256:929b[…] /lib/[…]/systemd-debug-generator
```

Υπόδειγμα αρχείου καταγραφής IMA από ένα σύστημα Linux.

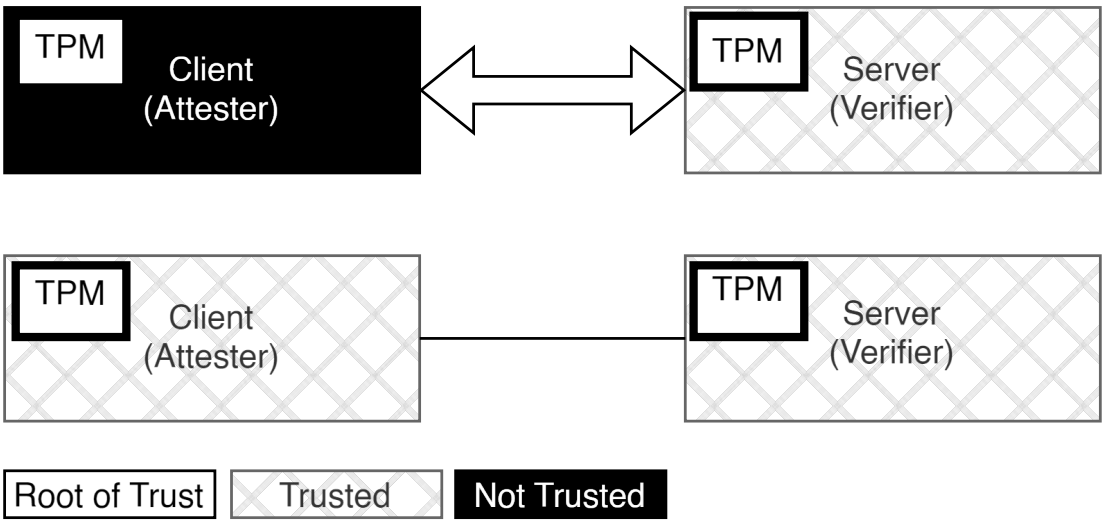
Σε συνδυασμό με μία TPM, τα hashes που συγκεντρώνει η IMA μπορούν να «επεκταθούν» σε έναν από τους PCR της TPM, κάνοντας έτσι δύσκολη οποιαδήποτε επίθεση/αλλοίωση στην λίστα μετρήσεων χωρίς αυτό να γίνει αντιληπτό. Έτσι, όταν ένα έμπιστο σύστημα εκκινήσει, χρησιμοποιώντας τις μετρήσεις IMA και την τρέχουσα τιμή του PCR μπορεί να επιβεβαιωθεί η ακεραιότητα του συστήματος.

2.2 Απομακρυσμένη Επιβεβαίωση Εμπιστεύσιμου Υλικολογισμικού (ΑΕΕΥ)

2.2.1 Κεντροποιημένο Σύστημα ΑΕΕΥ

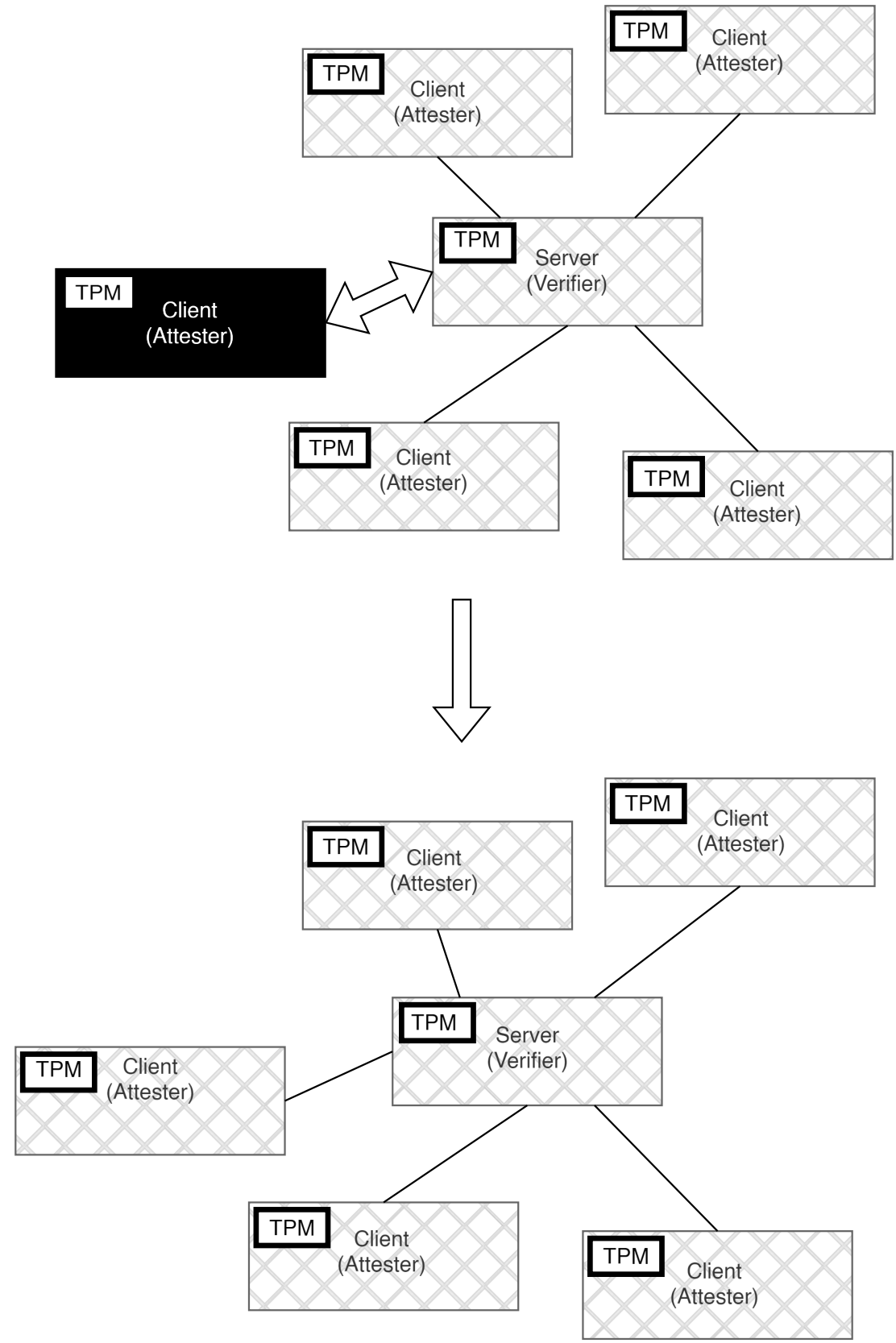
Ένας τρόπος υλοποίησης της ΑΕΕΥ βασίζεται στο μοντέλο server-client όπου ένας (ή περισσότεροι) client-attester ζητά από έναν server-verifier να πραγματοποιήσει επιβεβαίωση σε αυτόν απομακρυσμένα.

Πιο συγκεκριμένα [10], βασίζεται στην συλλογή και επιβεβαίωση μετρήσεων που συλλέχθηκαν από τους clients κατά την διαδικασία της εκκίνησης, με τους clients να παραδίδουν τις αρχικές μετρήσεις τους στον server. Ο server αποθηκεύει αυτές τις μετρήσεις και τις χρησιμοποιεί αργότερα όταν ο εκάστοτε client ζητά να επιβεβαιωθεί. Όταν συμβεί αυτό, ο server επαληθεύει αν τα στοιχεία που ανέβασε ο client προκειμένου να επιβεβαιωθεί, αντιστοιχούν σε αυτά που είχαν παραδοθεί νωρίτερα από εκείνον. Αν υπάρχει αντιστοιχία, τότε η συσκευή επιβεβαιώνεται και αναλόγως την υλοποίηση, η συσκευή απόκτα και πρόσβαση σε δεδομένα/υπηρεσίες που μόνο επιβεβαιωμένες συσκευές έχουν. Διαφορετικά, η συσκευή παραμένει μη επιβεβαιωμένη.



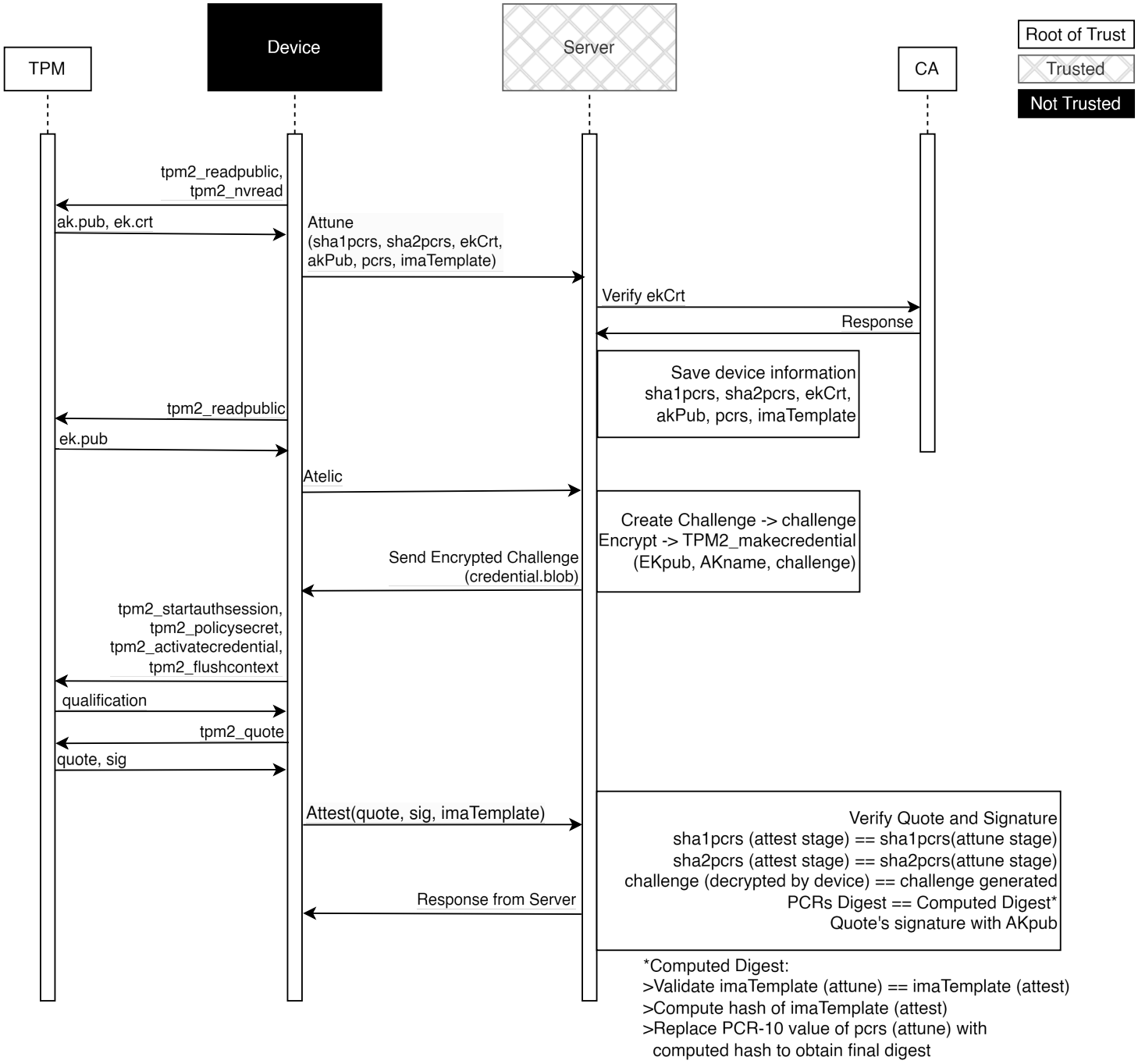
Εικόνα 4: Μία μη έμπιστη συσκευή πραγματοποιεί απομακρυσμένη επιβεβαίωση σε έναν έμπιστο εξυπηρετητή

Αυτός ο τρόπος υλοποίησης εντοπίζεται κυρίως σε υπηρεσίες cloud computing για την επαλήθευση της ακεραιότητας εικονικών μηχανών, σε datacenters για επιβεβαίωση σε κατανεμημένες συσκευές αλλά και σε δίκτυα IoT συσκευών για την επιβεβαίωση της εμπιστοσύνης τους. Με αυτόν το τρόπο επιτυγχάνεται επεκτασιμότητα μεγάλης κλίμακας, συνέπεια με τακτικούς ελέγχους σε όλες τις συσκευές και εύκολη παρακολούθηση όλων των επιβεβαιώσεων που πραγματοποιήθηκαν με τον συγκεκριμένο server.



Εικόνα 5: Παράδειγμα Κεντροποιημένου Συστήματος ΑΕΕΥ, μία μη έμπιστη συσκευή πραγματοποιεί απομακρυσμένη επιβεβαίωση σε έναν έμπιστο εξυπηρετητή και μετά από επιτυχή επιβεβαίωση γίνεται και αυτή έμπιστη.

Το Application Note της Infineon [4] υλοποιεί το πρωτόκολλο της εικόνας 6, στην βασική εκδοχή που περιγράφεται, Device (Attester) και Server (Verifier) βρίσκονται στην ίδια συσκευή (με την επικοινωνία να γίνεται μέσω του localhost), ωστόσο με μικρές τροποποιήσεις μπορεί να γίνει η επικοινωνία και σε ξεχωριστές συσκευές.



Εικόνα 6: Πρωτόκολλο επικοινωνίας μεταξύ TPM, Device, Server και Certification Authority. Σημείωση, το CA είναι υπεύθυνο αποκλειστικά για τον έλεγχο της αυθεντικότητας της TPM, επιβεβαιώνοντας αν το endorsement key certificate είναι του κατασκευαστή.

Γενικότερα, ένα από τα προβλήματα αυτής της υλοποίησης είναι η απόλυτη εμπιστοσύνη στον κεντρικό server υπεύθυνο για την επιβεβαίωση. Αυτό μπορεί να δημιουργήσει προβλήματα κατά την διάρκεια κάποιας αστοχίας σε αυτόν ή αν δεν είναι σε θέση να επιβεβαιώσει τις client συσκευές. Ταυτόχρονα, ο server μπορεί να γίνει στόχος επίθεσης επηρεάζοντας την σταθερότητα και την εμπιστοσύνη σε όλο το δίκτυο των client συσκευών. Ενώ επίσης, σε πολύ μεγάλη κλίμακα, με πολύ μεγάλο πλήθος συσκευών θα μπορούσε ο συγκεκριμένος server να μην μπορεί να ανταποκριθεί σε τόσο μεγάλο πλήθος συσκευών ή/και να δημιουργούσε πολύ μεγάλη συμφόρηση στο δίκτυο. Ένας τρόπος αντιμετώπισης των παραπάνω προβλημάτων αποτελεί και το Κατανεμημένο Σύστημα ΑΕΕΥ το οποίο αναλύεται στην συνέχεια.

2.2.2 Κατανεμημένο Σύστημα ΑΕΕΥ

Ένας διαφορετικός τρόπος υλοποίησης της ΑΕΕΥ βασίζεται στην peer-to-peer επικοινωνία μεταξύ των συσκευών, όπου οι ήδη έμπιστες συσκευές στο δίκτυο επιβεβαιώνουν άλλες μη έμπιστες συσκευές που θέλουν να γίνουν έμπιστες, και εφόσον επιβεβαιωθούν επιτυχώς, μπορούν και αυτές να επιβεβαιώσουν με την σειρά τους, άλλες νέες συσκευές. Ταυτόχρονα, με την χρήση τεχνολογιών blockchain, είναι εφικτή και η επαλήθευση των πληροφοριών που λαμβάνονται από μη έμπιστες συσκευές.

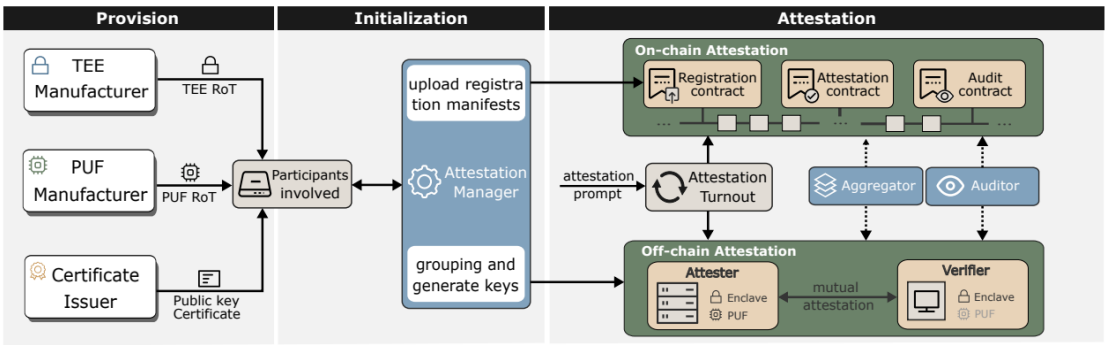
Ωστόσο, και εδώ υπάρχουν κάποιες σημαντικές προκλήσεις [11], ο χωρικός περιορισμός των blockchain λόγω περιορισμών κλιμάκωσης και η ελλιπής υποστήριξη σε κατανεμημένα πρωτόκολλα που υποστηρίζουν επιβεβαιώσεις σε blockchain. Για τον λόγο αυτό, έχουν προταθεί νέες ιδέες ώστε να λυθούν τα παραπάνω ζητήματα. Συνοπτικά:

- Συμβατότητα με πολλαπλά πρωτόκολλα, για μεγαλύτερη ευελιξία και ένα πιο αξιόπιστο σύστημα
- Επιβεβαίωση μέσω Blockchain, έτσι ώστε να μην είναι εφικτή η αλλοίωση των δεδομένων
- Κατανεμημένα πρωτόκολλα, για μεγαλύτερη διαθεσιμότητα και αξιοπιστία πχ μέσω του InterPlanetary File System (IPFS)
- Εφαρμογές, για περισσότερες υπηρεσίες όπως πιστοποίηση και χρονοσήμανση πληροφοριών.

Άλλες αντίστοιχες προσεγγίσεις [5] αξιοποιούν τα Έμπιστα Περιβάλλοντα Εκτέλεσης (Trusted Execution Environments, TEE) εντός των επεξεργαστών καθώς και την Φυσική Μη-Κλωνοποιήσιμη Συνάρτηση (Physically Unclonable Function, PUF), δηλαδή μίας δυνατότητας εντός του υλικού που δημιουργεί μοναδικές και μη προβλέψιμες εξόδους για κάθε συσκευή με βάση τα φυσικά χαρακτηριστικά της, ως

ρίζα εμπιστοσύνης για το TEE και ταυτόχρονα μέσω blockchain τεχνολογιών και έξυπνων συμβολαίων (smart contract) επιτυγχάνουν την κατανεμημένη επιβεβαίωση, με τα δεδομένα να καταγράφονται στο blockchain και να εκτελούνται οι επιθυμητές ενέργειες σύμφωνα με αυτό που ορίζει το έξυπνο συμβόλαιο.

Σε αυτή την περίπτωση, το πρωτόκολλο JANUS βασίζεται σε 3 στάδια, Προμήθεια (Provision), Αρχικοποίηση (Initialization), Επιβεβαίωση (Attestation). Αρχικά, στο στάδιο της προμήθειας, το TEE προμηθεύεται με μοναδικά για την συσκευή ιδιωτικά κλειδιά, ενώ οι PUF στην συσκευή προσφέρουν την ρίζα εμπιστοσύνης στο σύστημα. Στην αρχικοποίηση, ο Attester, ο Verifier και οι υπόλοιπες συσκευές δημιουργούν μία κοινή αλυσίδα. Ο Attestation Manager βοηθά στην δημιουργία κλειδιών για αυτές και καταχωρεί τις πρώτες εγγραφές στην αλυσίδα, ολοκληρώνοντας την αρχικοποίηση. Στην επιβεβαίωση, ο Verifier και ο Attester επικοινωνούν για το πώς θα πραγματοποιήσουν την επιβεβαίωση, έπειτα ο Verifier αποστέλλει μία πρόκληση για να ξεκινήσει επιβεβαίωση εκτός αλυσίδας ή εντός αλυσίδας και το τελικό αποτέλεσμα αποφασίζεται και μπορεί να εξεταστεί μελλοντικά.



Εικόνα 7: Αρχιτεκτονική και τρόπος λειτουργίας του JANUS, επανεκτύπωση από [5]

Στην [6], οι συγγραφείς παρουσιάζουν ένα υβριδικό σύστημα που βασίζεται στις PUF ετερογενών συσκευών για την δημιουργία μίας σύνθετης αλυσίδας. Οι PUF, μέσα από τις ιδιότητες να δημιουργούν μοναδικά αποτυπώματα για την πλατφόρμα στην οποία βρίσκονται, μπορούν να δημιουργήσουν και challenge-response ζεύγη (Challenge Response Pairs, CRP) για Industrial IoT συσκευές και μέσα από την χρήση blockchain για την αποθήκευση και τον συγχρονισμό των CRP αυτών για να αποφευχθούν προβλήματα αποτυχίας/αστοχίας σε ένα σημείο. Το σύστημα αυτό υποστηρίζει την δημιουργία διαφορετικών ειδών PUF από διαφορετικά είδη συσκευών, ενώ υποστηρίζει και πολλαπλούς επαληθευτές.

2.2.2.1 Ιδιαιτερότητες Κατανεμημένων Υλοποιήσεων

Παρά τα πλεονεκτήματα των κατανεμημένων υλοποιήσεων, υπάρχουν και μερικές ιδιαιτερότητες που μπορούν να κάνουν δύσκολη την αποτελεσματική εκμετάλλευση τους [11], [12]. Ορισμένα από αυτά είναι τα παρακάτω:

- Προβλήματα επεκτασιμότητας: Με όλο και περισσότερες συσκευές που συμμετέχουν στο δίκτυο, μπορούν να παρουσιαστούν προβλήματα επεκτασιμότητας. Θα πρέπει η κατανεμημένη υλοποίηση να μπορεί να διαχειριστεί μεγάλο αριθμό συσκευών αποτελεσματικά χωρίς να επηρεάζεται αρνητικά η απόδοση.
- Χωρητικότητα αποθήκευσης: Προσεγγίσεις που χρησιμοποιούν την blockchain τεχνολογία, έχουν περιορισμούς ως προς την χωρητικότητα σε κάθε block της αλυσίδας για λόγους επεκτασιμότητας.
- Πλαστοπροσωπία: Χωρίς τα κατάλληλα μέτρα, κακόβουλες συσκευές μπορούν να προσποιηθούν πως είναι απλά μη επιβεβαιωμένες συσκευές και να εμπλακούν στην διαδικασία απομακρυσμένης επιβεβαίωσης με έμπιστες συσκευές.
- Επαλήθευση: Τα δεδομένα που λαμβάνονται από μία συσκευή, θα πρέπει να μπορούν να επαληθευτούν πως είναι έγκυρα και έμπιστα, χωρίς όμως την ανάγκη ύπαρξης κάποιας κεντρικής αρχής.
- Μοναδικό Σημείο Αποτυχίας: Ενώ κατανεμημένα συστήματα προσπαθούν να αντιμετωπίσουν αυτό το πρόβλημα, ενδέχεται να υπάρχουν σημεία με κεντροποιημένες προσεγγίσεις, τα οποία θα πρέπει θωρακισμένα έναντι αποτυχιών ή αστοχιών.
- Διαλειτουργικότητα: Ανάλογα με την υλοποίηση, μπορεί να είναι απαραίτητη η διαλειτουργικότητα μεταξύ διαφορετικών συστημάτων.

2.2.2.2 Επαλήθευση κλειδιών σε κατανεμημένα περιβάλλοντα

Ένα από τα προβλήματα που αναφέρθηκε παραπάνω είναι η πλαστοπροσωπία, καθώς αν είναι απαραίτητα όλα τα στοιχεία μίας μη έμπιστης συσκευής κατά την εγγραφή, πως θα ελεγχθεί πως πρόκειται για μία νόμιμη συσκευή που θέλει να εισέλθει στο δίκτυο και όχι μία κακόβουλη συσκευή; Πως μπορεί να ελεγχθεί ότι τα κλειδιά που παρέδωσε η μη έμπιστη συσκευή είναι από εγκεκριμένη διάταξη και όχι ένα γενικό κλειδί;

Σε αυτό το σημείο, είναι απαραίτητο να γίνει επαλήθευση της προέλευσης των κλειδιών με κάποια κεντρική αρχή και συγκεκριμένα με την αρχή έκδοσης των πιστοποιητικών, δηλαδή τον κατασκευαστή της TPM αποκλειστικά και μόνο για να επιβεβαιώσει πως το κλειδί προέρχεται από TPM που κατασκεύασε και δεν είναι ένα γενικό κλειδί. Από εκεί, εφόσον είναι γνωστό πως το κλειδί προέρχεται από μία αυθεντική TPM που χρησιμοποιείται για τις συσκευές του συγκεκριμένου δικτύου, επιτρέπεται η εγγραφή των δεδομένων που παρέχονται από αυτή τη συσκευή και συνεχίζεται η διαδικασία απομακρυσμένης επιβεβαίωσης χρησιμοποιώντας τα στοιχεία της εγγραφής και σε κάθε μελλοντική προσπάθεια απομακρυσμένης επιβεβαίωσης ως σημείο αναφοράς για την συγκεκριμένη συσκευή.

Τα στοιχεία της εγγραφής αυτής μπορούν να αποθηκευτούν στο Διαπλανητικό Σύστημα Αρχείων (InterPlanetary File System), το οποίο δημιουργεί ένα hash για τα δεδομένα που αποθηκεύονται και όταν χρειαστούν τα δεδομένα αυτά, μπορούν να ανακτηθούν χρησιμοποιώντας το hash αυτό.

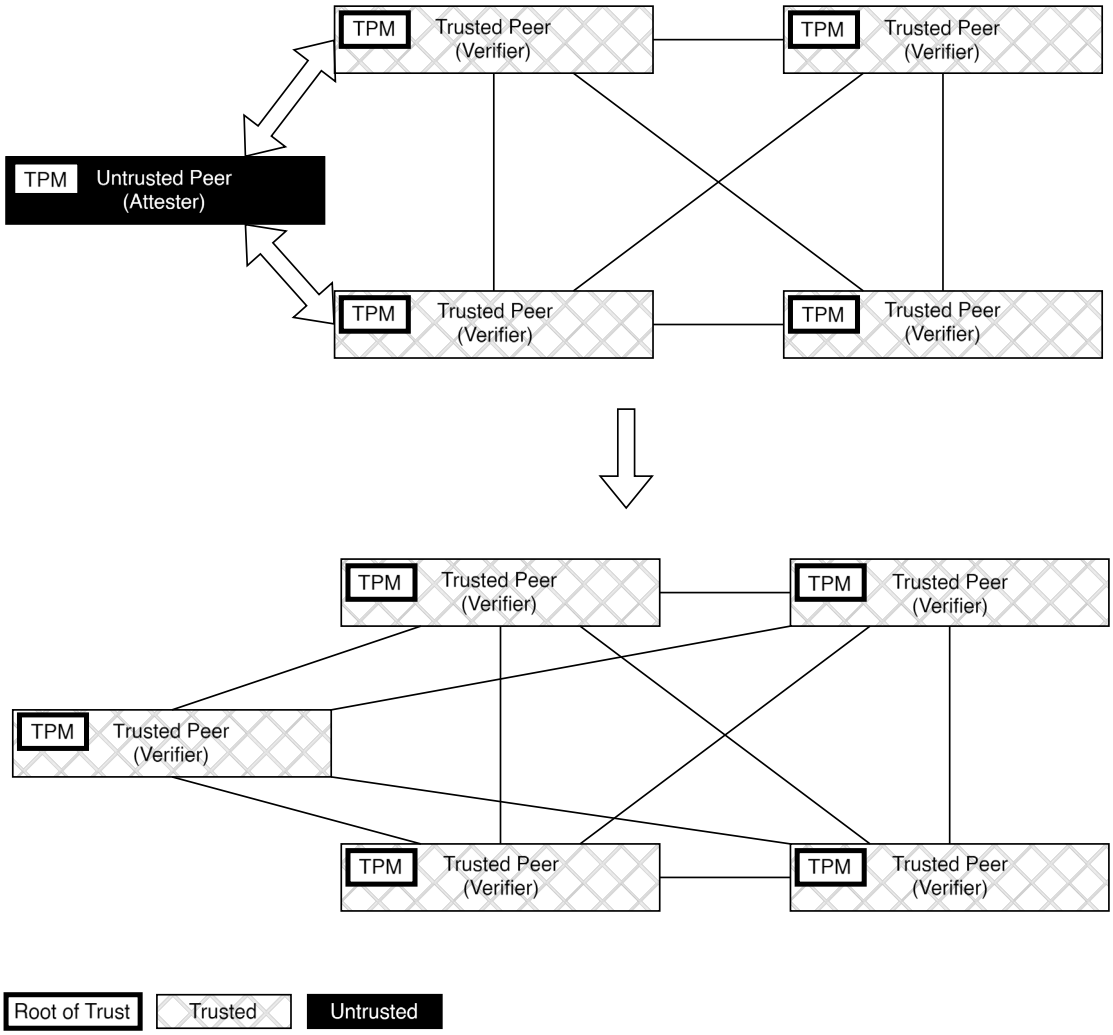
Κεφάλαιο 3. Πρωτότυπο Κατανεμημένο Σύστημα Επιβεβαίωσης

Στο κεφάλαιο αυτό, αναλύεται περισσότερο το Κατανεμημένο Σύστημα ΑΕΕΥ, συγκεκριμένα θα παρουσιαστούν: η αρχιτεκτονική του πρωτότυπου Κατανεμημένου Συστήματος ΑΕΕΥ, οι τεχνικές λεπτομέρειες της υλοποίησης του πρωτότυπου, το πρωτόκολλο επικοινωνίας που χρησιμοποιεί το πρωτότυπο και σε μεγαλύτερο βάθος τα στάδια του πρωτόκολλου καθώς και περαιτέρω επεκτάσεις που μπορούν να γίνουν στο συγκεκριμένο πρωτότυπο.

3.1 Βασική Ιδέα

Όπως και σε Κεντροποιημένα Συστήματα ΑΕΕΥ, έτσι και σε κατανεμημένα συστήματα, οι μη επιβεβαιωμένοι (untrusted) peers ζητούν από έναν ή περισσότερους επιβεβαιωμένους (trusted) peers, να τους επιβεβαιώσουν. Οι untrusted peers συλλέγουν και παρέχουν τις μετρήσεις τους στους trusted peers, οι οποίοι στην συνέχεια επαληθεύουν τις μετρήσεις αυτές και εφόσον αυτό γίνει με επιτυχία, οι untrusted peers γίνονται trusted, αντί αυτό να γίνεται αποκλειστικά μέσω κάποιας κεντρικής αρχή.

Ταυτόχρονα, είναι εφικτό σε δίκτυα κατανεμημένων συσκευών να έχουμε και πολιτικές κοινής συναίνεσης (consensus) όπου θα απαιτείται επιβεβαίωση με πολλαπλό αριθμό από trusted peer και μόνο εφόσον επιτευχθεί συναίνεση μεταξύ των trusted peer να γίνει η συσκευή trusted.



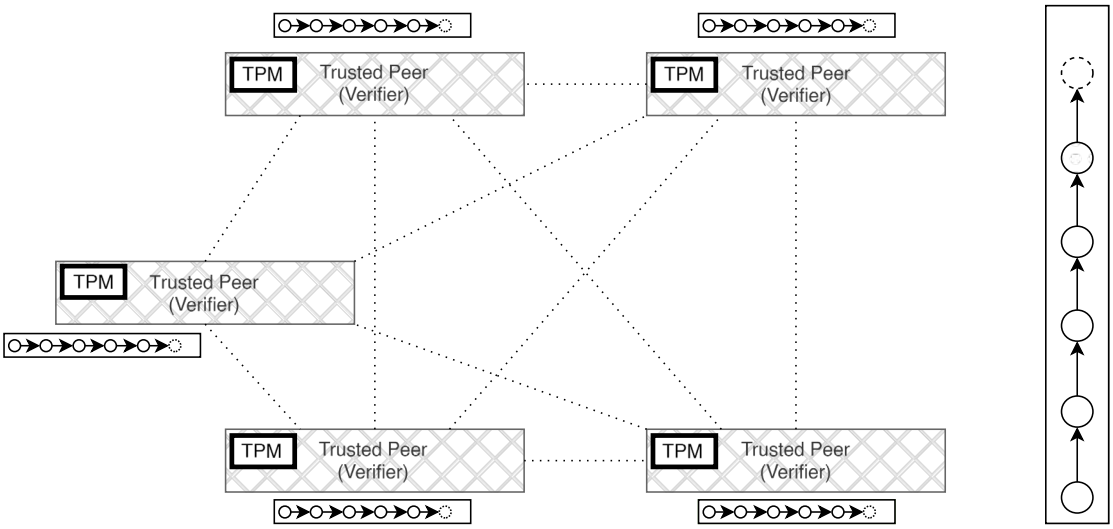
Εικόνα 8: Ένα δίκτυο επιβεβαιωμένων συσκευών και μία μη επιβεβαιωμένη συσκευή που πραγματοποιεί attestation με δύο έμπιστες συσκευές του δικτύου και μετά από επιτυχές attestation και με τις δύο εντάσσεται στο δίκτυο και οι υπόλοιπες συσκευές ενημερώνονται για την νέα προσθήκη στο δίκτυο.

Πλεονεκτήματα μίας κατανεμημένης υλοποίησης είναι η υψηλή ευελιξία, καθώς δεν υπάρχει μοναδικό σημείο αποτυχίας, ο διαμοιρασμός της εμπιστοσύνης, αφού δεν υπάρχει εξάρτηση από μία κεντρική αρχή και η διαφάνεια, διότι όλα τα δεδομένα μπορούν να επιβεβαιωθούν από όλους τους συμμετέχοντες του δικτύου (εφόσον βέβαια έχει οριστεί κάποια μέθοδος καταγραφής).

Μία υλοποίηση του Κατανεμημένου Συστήματος ΑΕΕΥ, μπορεί να βασιστεί στην ανεξάρτητη διατήρηση από κάθε peer, μίας λίστας με έμπιστες συσκευές. Κατά την επιβεβαίωση μίας νέας συσκευής και εφόσον αυτή ολοκληρωθεί με επιτυχία, η συσκευή αυτή προστίθεται στην λίστα των έμπιστων συσκευών των peer που πραγματοποίησαν

την επιβεβαίωση και αυτοί με την σειρά τους φροντίζουν για την ενημέρωση των υπόλοιπων έμπιστων συσκευών του δικτύου, ώστε και αυτές να ενημερώσουν την λίστα που διατηρούν. Αυτή η λίστα μπορεί να είναι μία εσωτερική δομή μέσα στην εφαρμογή, ένα αρχείο καταγραφής ή ακόμα και μία τοπική βάση δεδομένων. Ένα πρόβλημα που διακρίνεται εδώ είναι και η ανάγκη για πολύ μεγάλο βαθμό διασύνδεσης του κάθε peer μαζί με όλους τους υπόλοιπους, όπου μία ενημέρωση στην λίστα, όσο αυξάνονται οι έμπιστες συσκευές θα απαιτεί έναν αρκετά μεγάλο αριθμό ενημερώσεων, μεγαλύτερο ή ίσο με τον αριθμό των trusted peers.

Σε μία διαφορετική προσέγγιση του προβλήματος, θα μπορούσαμε αντί για τον παραπάνω τρόπο διαχείρισης των έμπιστων συσκευών, να αξιοποιήσουμε τις τεχνολογίες blockchain, όπου αντί κάθε peer να διατηρεί μία λίστα την όποια θα πρέπει να ενημερώνει κατόπιν ενημέρωσης από άλλους έμπιστους peers, να μπορεί κάθε peer να δει την προσθήκη ενός νέου έμπιστου peer μέσα από μία ενημέρωση στο blockchain. Αυτό μπορεί να γίνεται μέσα από έλεγχο σε τακτικά χρονικά διαστήματα από κάθε peer ή όταν θα χρειαστεί ο trusted peer να αλληλοεπιδράσει με έναν άλλον που δεν γνωρίζει πως είναι έμπιστος πλέον, αλλά μπορεί να το επαληθεύσει κοιτώντας στο blockchain για νέες προσθήκες ή αλλαγές.



Εικόνα 9: Παράδειγμα δικτύου με trusted peers, όπου κάθε peer διατηρεί αντίγραφο του blockchain.

Ένα παράδειγμα τεχνολογίας blockchain που θα μπορούσε να χρησιμοποιηθεί αποτελεί και το Hyperledger Fabric. Μίας πλατφόρμας για δημιουργία κατανεμημένων αρχείων καταγραφής [13] με αρθρωτή αρχιτεκτονική και μεγάλα επίπεδα ευελιξίας, εμπιστευτικότητας, ανθεκτικότητας και επεκτασιμότητας χωρίς όμως να εξαρτάται από κάποιο κρυπτονόμισμα ή ειδικού σκοπού γλώσσα προγραμματισμού.

Αν και αυτή η προσέγγιση ίσως αποτελούσε την καλύτερη επιλογή, στο πλαίσιο αυτής της διπλωματικής, θα παρουσιαστεί μία πιο απλοϊκή εκδοχή της πρώτης

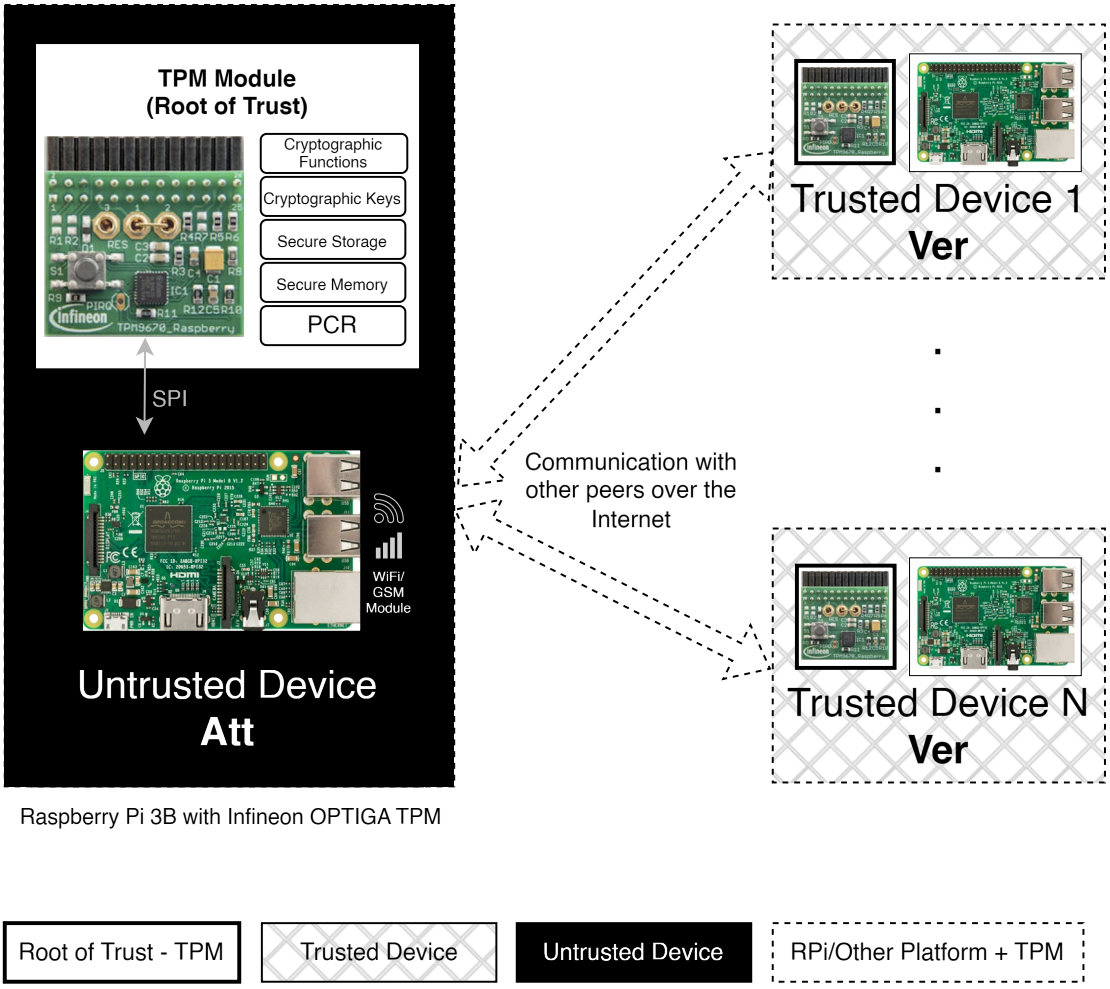
προσέγγισης, όπου βέβαια με τις κατάλληλες επεκτάσεις θα μπορούσε να εξελιχθεί ώστε να αξιοποιεί και blockchain τεχνολογίες σαν την προαναφερθείσα.

3.2 Πρωτότυπο σύστημα

Για την υλοποίηση αυτή έχει δημιουργηθεί ένα πρωτότυπο στο όποιο υπάρχει μία μη έμπιστη συσκευή η οποία διαθέτει μία φυσική TPM η οποία επιθυμεί να ενταχθεί σε ένα δίκτυο έμπιστων συσκευών.

Οι έμπιστες συσκευές, έχουν ήδη ενταχθεί στο δίκτυο και περιμένουν για τυχόν νέες συσκευές που θα θελήσουν να επιβεβαιωθούν για να ενταχθούν στο δίκτυο ως έμπιστες συσκευές. Μία νέα συσκευή, ξεκινά την διαδικασία επιβεβαίωσης με τις έμπιστες συσκευές (αυτή την στιγμή, οι διευθύνσεις IP των έμπιστων συσκευών έχουν δοθεί εντός του κώδικα, αλλά θα μπορούσε να υλοποιηθεί με κάποια διαδικασία εύρεσης εντός του δικτύου ή αναζητώντας για έμπιστες συσκευές στο blockchain). Η έμπιστη συσκευή αποδέχεται την σύνδεση από την μη έμπιστη συσκευή και μετά από μία ανταλλαγή πληροφοριών μεταξύ τους, η έμπιστη συσκευή δίνει (ή όχι) την έγκρισή της για να ενταχθεί η συσκευή στο δίκτυο απαντώντας θετικά (ή αρνητικά) στο αίτημα της μη έμπιστης συσκευής. Κατά τον ίδιο τρόπο, η διαδικασία επαναλαμβάνεται και με άλλες έμπιστες συσκευές και εφόσον έχει λάβει εξίσου θετικές απαντήσεις, η συσκευή εντάσσεται στο δίκτυο ως έμπιστη και αναμένει και αυτή για νέες συνδέσεις επιβεβαίωση.

Η αρχιτεκτονική του πρωτοτύπου βασίζεται πάνω στο πρωτόκολλο της υλοποίησης της Infineon [4], έχοντας βέβαια αλλαγές ως στον τρόπο λειτουργίας προκειμένου να μπορεί να πραγματοποιηθεί απομακρυσμένη επιβεβαίωση σε ένα κατανεμημένο δίκτυο, χωρίς την ανάγκη ύπαρξης κάποιου κεντρικού εξυπηρετητή.



Εικόνα 10: Αρχιτεκτονική Συστήματος

Η επικοινωνία μεταξύ της μη έμπιστης συσκευής με μία έμπιστη συσκευή μπορεί να χωριστεί σε 3 στάδια (Attune, Atelic, Attest). Η διαδικασία αυτή επαναλαμβάνεται από την αρχή μέχρι η μη έμπιστη συσκευή να λάβει τον απαιτούμενο αριθμό εγκρίσεων προκειμένου να γίνει έμπιστη. Μόλις λάβει έγκριση από μία έμπιστη συσκευή, επικοινωνεί με την επόμενη έμπιστη συσκευή με την ίδια ακριβώς λογική.

Σημειογραφία	
Att	Attester – η μη έμπιστη συσκευή η οποία επιθυμεί να γίνει έμπιστη
Ver	Verifier – η έμπιστη συσκευή η οποία επιβεβαιώνει την μη έμπιστη συσκευή

3.2.1 Attune – 1ο Στάδιο

Στο στάδιο του attune, η **Att** παρέχει κάποιες παραμέτρους του στην **Ver** με την οποία επικοινωνεί, παραμέτρους που θα χρειαστεί αργότερα στο στάδιο του Attest, συγκεκριμένα είναι:

- Register Indexes από το PCR bank 1 (SHA1)
- Register Indexes από το PCR bank 2 (SHA256)
- Το Endorsement Key πιστοποιητικό της Infineon
- Το δημόσιο Attestation Key
- Τις τιμές των PCR
- Το IMA template, ένα αρχείο καταγραφής με τα hashes εκτελέσιμων αρχείων κατά το boot. Το IMA template που αποστέλλεται στον server είναι το binary_runtime_measurements.
- Το δημόσιο Endorsement Key
- Το όνομα του AK και
- AK και EK πιστοποιητικά σε μορφή .pem

Κάθε hash από κάθε καταχώριση στο ascii_runtime_measurements γίνεται κατά την εκκίνηση extend στο PCR-10 (από προεπιλογή της αρχιτεκτονικής IMA). Λόγω της μη ντετερμινιστικής εκκίνησης του Linux, το αρχείο καταγραφής αλλάζει σε κάθε εκκίνηση του συστήματος, γι' αυτό και θα πρέπει να γνωστοποιηθεί στην **Ver** πρώτα.

Μετά από την επιτυχή ανάγνωση των αρχείων από την **Ver**, η **Ver** ενημερώνει την **Att** πως το στάδιο αυτό ολοκληρώθηκε.

3.2.2 Atelic – 2ο Στάδιο

Κατά το στάδιο του atelic, η **Att** ζητά μία πρόκληση από την **Ver**, η οποία αρχικά δημιουργεί μία τυχαία τιμή, για να χρησιμοποιηθεί στην πρόκληση για την **Att**. Η πρόκληση αυτή δημιουργείται με την tpm2_makecredential με την χρήση του ak.name, του ek.pem και της τυχαίας τιμής που δημιουργήθηκε νωρίτερα. Με την εντολή αυτή, δημιουργείται ένα νέο αρχείο, το credential.blob, το οποίο εμπεριέχει την τυχαία τιμή σε κρυπτογραφημένη μορφή, με τέτοιο τρόπο όπου μόνο η συγκεκριμένη TPM μπορεί να χρησιμοποιηθεί για την αποκρυπτογράφηση του. Στη συνέχεια η **Ver** απαντά στην **Att** με το challenge αυτό και με μία ενημέρωση πως η διαδικασία για το 2ο στάδιο ολοκληρώθηκε.

Η **Att** λαμβάνει το challenge, και το αποκρυπτογραφεί εκτελώντας την εντολή tpm2_activatecredential ώστε να ανακτήσει την πρόκληση (την τυχαία τιμή που δημιούργησε η **Ver**) την οποία και θα χρησιμοποιήσει για να δημιουργήσει ένα quote και ένα αρχείο υπογραφής sig. Το quote είναι μία ασφαλή αναφορά, κρυπτογραφικά υπογεγραμμένη και παραγμένη από το TPM και επιβεβαιώνει την τρέχουσα κατάσταση των PCR. Κατά την δημιουργία του AK, ένα σύνθετο hash από επιλεγμένα PCR υπολογίστηκε (στην περίπτωσή αυτή, μόνο ένα, το sha256:10 γιατί αυτό χρησιμοποιείται από την IMA αρχιτεκτονική) και παραδόθηκε στο 1ο στάδιο στην **Ver**

(στο αρχείο pcrs). Το quote αυτό παράγει εσωτερικά ένα σύνθετο hash, από τα ίδια επιλεγμένα PCR, το οποίο μπορεί να συγκριθεί με αυτό που παραδόθηκε στο 1ο στάδιο.

3.2.3 Attest – 3ο Στάδιο

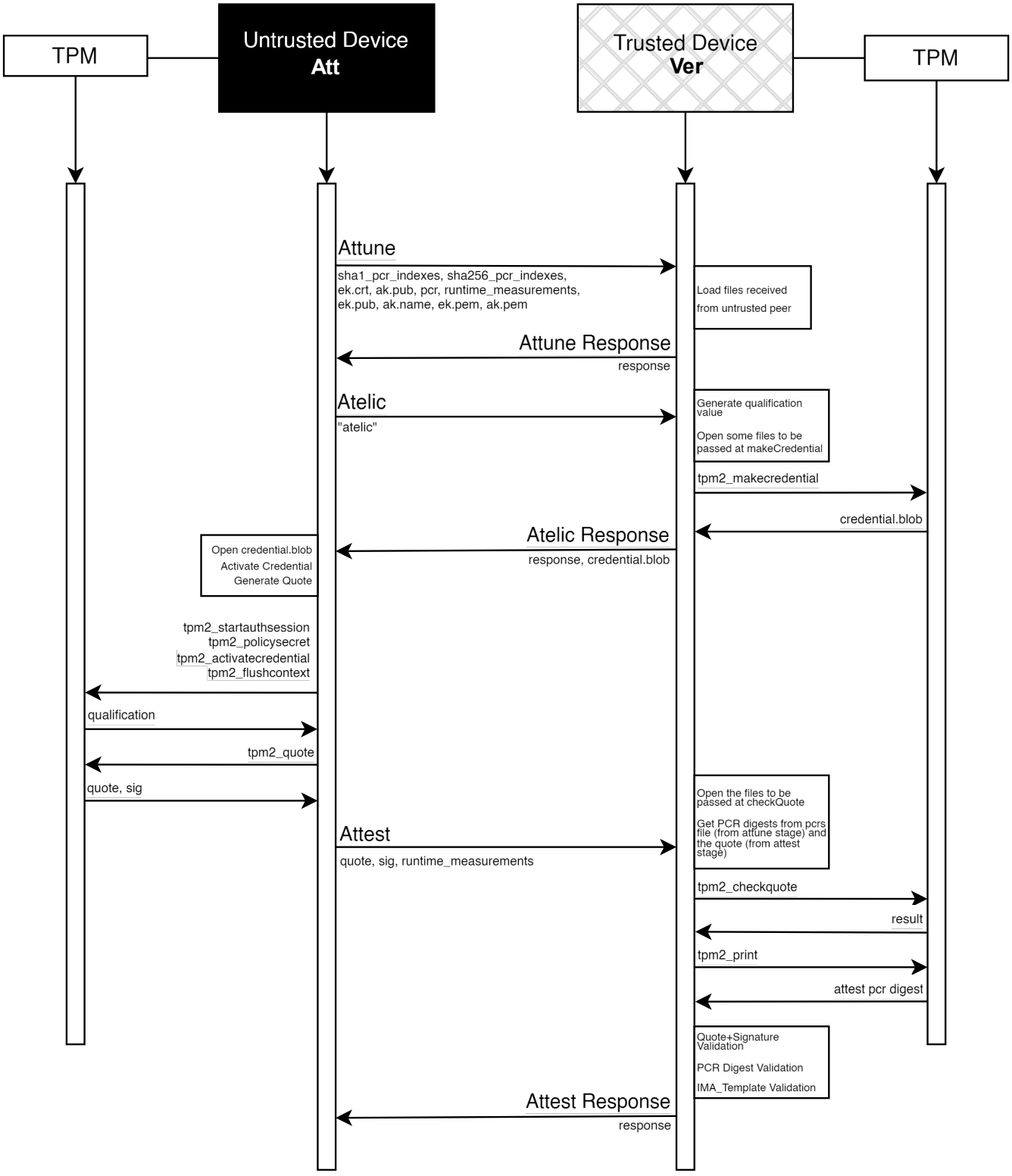
Στο στάδιο του attest, η **Att** στέλνει στην **Ver**, το quote και το sig που δημιούργησε προηγουμένως και το πιο πρόσφατο αντίγραφο των μετρήσεων IMA. Η **Ver** λαμβάνει τα αρχεία αυτά και τα περνά από την tpm2_checkquote, η οποία αναλαμβάνει την επιβεβαίωση του quote που έστειλε η **Att**. Αυτό το κάνει επιβεβαιώνοντας αν το δημόσιο κλειδί που δόθηκε σαν παράμετρος επιβεβαιώνει την υπογραφή του quote και η τυχαία τιμή που είχε δημιουργηθεί στο 2ο στάδιο αντιστοιχεί με αυτή που υπάρχει εντός του quote.

Στη συνέχεια, ελέγχεται αν το σύνθετο hash από το pcrs είναι το ίδιο με αυτό που υπάρχει μέσα στο quote και αν το IMA template που δόθηκε στο 1ο στάδιο είναι το ίδιο με αυτό που λήφθηκε σε αυτό το στάδιο.

Εφόσον ισχύουν οι τρεις παραπάνω προϋποθέσεις, η **Ver** δίνει την έγκρισή της για να γίνει έμπιστη, και στέλνει την απάντησή της στην **Att**.

3.3 Πρωτόκολλο Επικοινωνίας Untrusted-Trusted Peer

Παρακάτω παρουσιάζεται το πρωτόκολλο επικοινωνίας μεταξύ **Att** και **Ver** με βάση την αρχιτεκτονική Attune-Atelic-Attest καθώς και η επικοινωνία κάθε συσκευής με την μονάδα TPM που διαθέτει. Κάθε στάδιο ξεκινά από την **Att** η οποία παρέχει πληροφορίες σχετικά με την πλατφόρμα και την TPM στην **Ver**, στη συνέχεια ζητά την πρόκληση από την **Ver**, η οποία της την αποστέλλει και η **Att** την αποκρυπτογραφεί, έπειτα η **Att** δημιουργεί μία ασφαλή αναφορά, την υπογράφει και την αποστέλλει μαζί με την τρέχουσα κατάσταση της πλατφόρμας στην **Ver**. Η **Ver** ελέγχει τα στοιχεία που έλαβε και εφόσον όλοι οι έλεγχοι ολοκληρώνονται με επιτυχία και η **Att** επιβεβαιώνεται, δίνει την έγκριση της για να γίνει trusted.



*Εικόνα 11: Πρωτόκολλο επικοινωνίας μεταξύ **Att** και **Ver***

3.4 Επεκτάσεις της υλοποίησης

Μερικές επεκτάσεις της παραπάνω υλοποίησης, με στόχο να την αποτροπή ζητημάτων πλαστοπροσωπίας που είχαν αναφερθεί στο Κεφ. 2.2.2.2 και βελτιώσεις για το πως θα μπορούσε να γίνει το ανωτέρω σύστημα πιο ασφαλές, καθώς το τρέχον πρωτότυπο δεν χρησιμοποιεί τεχνολογίες blockchain και decentralized storage. Συγκεκριμένα, στην πρώτη εμφάνιση μίας συσκευής στο δίκτυο, κρίνεται απαραίτητο να πραγματοποιηθεί μία επαλήθευση με την αρχή έκδοσης των πιστοποιητικών της TPM, ώστε να επαληθευτεί ότι το δημόσιο κλειδί που χρησιμοποιεί προέρχεται από μία αυθεντική TPM που χρησιμοποιούν οι συσκευές του συγκεκριμένου δικτύου. Αυτό θα είναι το μοναδικό σημείο που θα απαιτείται η χρήση μιας κεντροποιημένης προσέγγισης και αποκλειστικά και μόνο για την επαλήθευση αυτή.

Ταυτόχρονα, με την επαλήθευση από την αρχή έκδοσης πιστοποιητικών της TPM, η έμπιστη συσκευή μπορεί να εγγράψει (enrollment) τα δεδομένα του 1ου σταδίου (attune) που απέστειλε η μη έμπιστη συσκευή στο IPFS, και έτσι κάθε μελλοντική επαλήθευση θα χρειάζεται απλά το hash από αυτά τα δεδομένα ώστε να βρεθεί εντός του IPFS και να μην χρειάζεται να τα αποστείλει ξανά η μη έμπιστη συσκευή. Αυτά τα δεδομένα θα είναι και το σημείο αναφοράς για την κατάσταση της συγκεκριμένης συσκευής, με την οποία θα γίνεται σύγκριση σε μελλοντικές προσπάθειες απομακρυσμένης επιβεβαίωσης.

Αυτό είναι χρήσιμο καθώς μπορεί να ενσωματωθεί η δυνατότητα λήξης της επιβεβαίωσης μίας συσκευής μετά από ένα χρονικό διάστημα και να απαιτείται η ανανέωση της επιβεβαίωσης της συσκευής αυτής ώστε να εντοπισθεί τυχόν παραβίαση ή αλλαγή στην κατάσταση της συσκευής και να αποκλειστεί η πρόσβαση στο δίκτυο έμπιστων συσκευών.

Επίσης, αντί η μη έμπιστη συσκευή να επικοινωνεί με όλες τις έμπιστες συσκευές που χρειάζεται για να εγγραφεί/ανανεώσει την επιβεβαίωση της, θα μπορούσε να επικοινωνεί αποκλειστικά με μία συσκευή, την συσκευή που έκανε την εγγραφή στο IPFS και να αναλαμβάνει αυτή η έμπιστη συσκευή να επικοινωνήσει με άλλες έμπιστες συσκευές για να πραγματοποιήσουν την απομακρυσμένη επιβεβαίωση από κοινού με διαφορετικές προκλήσεις η κάθε μία. Έτσι η έμπιστη συσκευή που έκανε την εγγραφή, γίνεται και ο διαμεσολαβητής μεταξύ της μη έμπιστης συσκευής και άλλων έμπιστων συσκευών του δικτύου, με τις έμπιστες συσκευές να εκτελούν τα αντίστοιχα βήματα και να τα προωθούν μέσω της έμπιστης συσκευής που έχει αναλάβει την διαμεσολάβηση. Για την περίπτωση που έχουμε ανανέωση της εμπιστοσύνης, τα δεδομένα αναφοράς

για αυτή την συσκευή μπορούν να συγκριθούν με αυτά υπάρχουν στο IPFS, χωρίς να χρειαστεί να γίνει το 1ο στάδιο (attune) ξανά.

Και τέλος, το blockchain μπορεί να είναι το μέσο στο οποίο οι έμπιστες συσκευές εγγράφουν γεγονότα που συμβαίνουν σε αυτές ή εντός του δικτύου, π.χ. επιτυχημένες και αποτυχημένες προσπάθειες επιβεβαίωσης, νέες εγγραφές συσκευών στο IPFS, καταγραφή δεδομένων από αισθητήρες κ.λπ.

Ενισχύοντας το παραπάνω πρωτότυπο με τις παραπάνω επεκτάσεις καθίσταται εφικτή η χρήση του σε περιβάλλοντα ακροδικτυακής υπολογιστικής, όπου συσκευές που υλοποιούν το παραπάνω πρωτότυπο θα είναι σε θέση να είναι ασφαλείς από εξωτερικές παρεμβάσεις καθώς η οποιαδήποτε παραβίαση ή ανωμαλία θα είναι σε θέση να εντοπιστεί και να αντιμετωπιστεί με την ανεπιτυχή επιβεβαίωση της συσκευής αυτής και με την ελάχιστη δυνατή επιβάρυνση.

Κεφάλαιο 4. Πειραματικά Αποτελέσματα

Στο κεφάλαιο αυτό, παρουσιάζονται πειραματικές μετρήσεις χρόνου προκειμένου να αξιολογηθεί το παραπάνω πρωτότυπο μέσα από διάφορα σενάρια και πειραματικές διατάξεις.

4.1 Βασική Διάταξη Αξιολόγησης

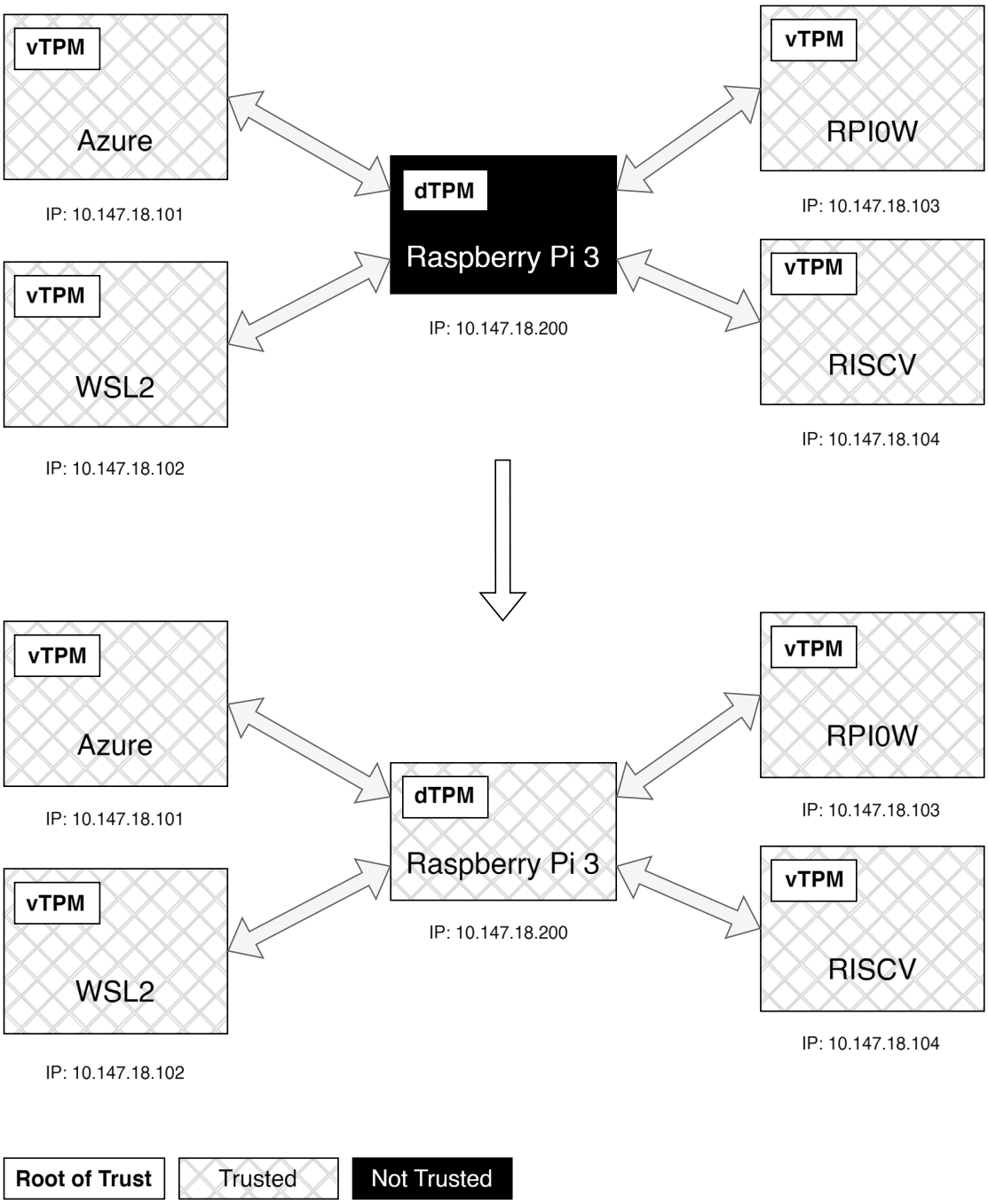
Οι μετρήσεις που συγκεντρώθηκαν βασίζονται στην ακόλουθη διάταξη, όπου υπάρχει μία μη έμπιστη συσκευή η οποία πραγματοποιεί την διαδικασία απομακρυσμένης επιβεβαίωσης σε τέσσερις άλλες έμπιστες συσκευές στο δίκτυο. Η μη έμπιστη συσκευή είναι η Raspberry Pi 3B (**RPI3B**) αρχιτεκτονικής ARMv7 η οποία έχει συνδεδεμένο το Infineon OPTIGA™ TPM SLI 9670, ενώ όλες οι υπόλοιπες συσκευές είναι ήδη έμπιστες συσκευές του δικτύου και διαθέτουν ένα εικονικό TPM (IBM Software TPM). Η μη έμπιστη συσκευή επικοινωνεί με τις έμπιστες συσκευές μία-μία και εφόσον επιβεβαιωθεί από όλες, γίνεται και εκείνη έμπιστη. Για τις μετρήσεις χρησιμοποιήθηκε η time για τα script που κλήθηκαν από το πρωτότυπο και η time.process_time() της Python για τις εσωτερικές συναρτήσεις του πρωτότυπου.

ΣΥΣΚΕΥΕΣ	ΛΕΙΤ. ΣΥΣ.	ΕΠΕΞΕΡΓΑΣΤΗΣ (ΠΥΡΗΝΕΣ) @ ΧΡΟΝΙΣΜΟΣ	ΑΡΧΙΤ.
Azure	Ubuntu 20.04.6	Intel Xeon Platinum 8272CL (1) @ 2.593Ghz	x86_64
WSL2	Ubuntu 22.04.4	AMD Ryzen 7 4800U (16) @ 1.796Ghz	x86_64
RPI0W	Raspbian Buster	BCM2835 – ARM1176JZF-S (1) @ 1.000GHz	armv6l
RISCV	Ubuntu 22.04.3	(Generic virt QEMU machine) (4) @ N/A	riscv64
RPI3B	Raspbian Buster	BCM2837 – ARM Cortex A53 (4) @ 1.200GHz	armv7l

Πίνακας 1: Χαρακτηριστικά των συσκευών της διάταξης αξιολόγησης

Οι έμπιστες συσκευές αποτελούνται από μία εικονική μηχανή με Ubuntu στο Azure (**Azure**) αρχιτεκτονικής x86, μία εγκατάσταση Ubuntu μέσω WSL2 σε Windows (**WSL2**) αρχιτεκτονικής x86, μία φυσική εγκατάσταση Raspbian 10 σε ένα Raspberry Pi Zero W (**RPI0W**) αρχιτεκτονικής ARMv6 και μία εγκατάσταση Ubuntu στον προσομοιωτή QEMU (**RISCV**) αρχιτεκτονικής RISCV.

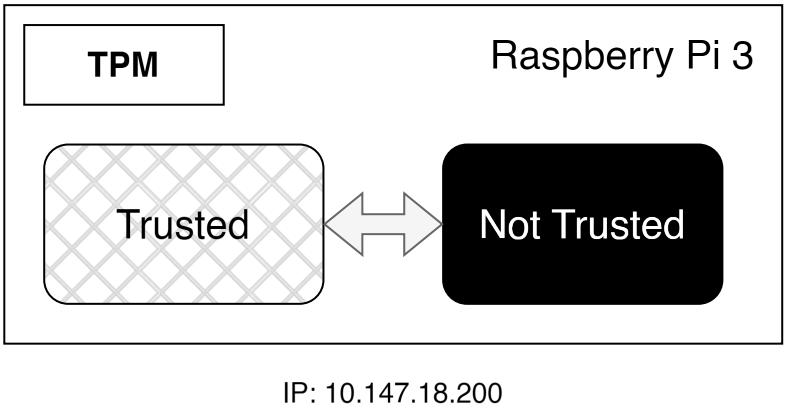
Εικόνα 12: Το πειραματικό setup



Εικόνα 13: Διάταξη Αξιολόγησης

Η επικοινωνία γίνεται σειριακά, ξεκινώντας από την **Azure**, και στην συνέχεια ακολουθούν **WSL2**, **RPI0W** και **RISCV**. Οι συσκευές αποτελούν nodes σε ένα Software Defined Network και επικοινωνούν μέσω αυτού, διαθέτοντας διαφορετική φυσική IP διεύθυνση για να βγαίνουν στο διαδίκτυο.

Με την ολοκλήρωση της παραπάνω διαδικασίας, εξετάστηκε με την πλέον έμπιστη συσκευή **RPI3B** η συλλογή μετρήσεων σχετικών με την απόδοση της ίδιας όταν είναι έμπιστη και καλείται να επιβεβαιώσει μία μη επιβεβαιωμένη συσκευή.

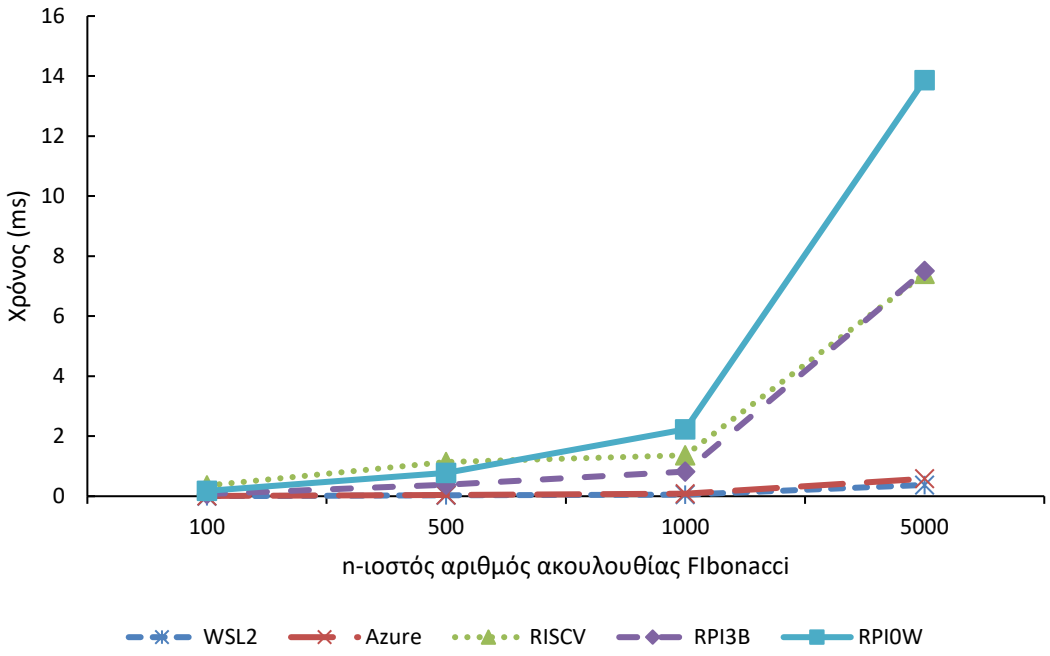


Εικόνα 14: Αφού επιβεβαιώθηκε η συσκευή, ένα νέο instance της εφαρμογής ξεκινά και ζητά επιβεβαίωση από την ίδια την συσκευή (χρησιμοποιώντας την IP διεύθυνση 10.147.18.200)

Για αυτό το σενάριο, με μία ελάχιστα τροποποιημένη έκδοση της εφαρμογής, εκκινήθηκε εκ νέου διαδικασία επιβεβαίωσης επάνω στην ίδια την έμπιστη συσκευή. Συγκεκριμένα, αφού έχει επιβεβαιωθεί η συσκευή και είναι έμπιστη και αφήνοντάς την στην έμπιστη κατάσταση σε αναμονή για νέες συνδέσεις, τότε εκκινήθηκε από ένα νέο τερματικό με την τροποποιημένη έκδοση της εφαρμογής, η διαδικασία επιβεβαίωσης με την έμπιστη συσκευή, απαιτώντας απλά μία επιβεβαίωση για να ολοκληρωθεί η διαδικασία.

4.2 Μέτρηση της απόδοσης των συσκευών

Πριν την έναρξη των παραπάνω ελέγχων, δοκιμάστηκε η απόδοση των παραπάνω συσκευών με μία απλή εφαρμογή η οποία υπολογίζει τον n-ιοστό αριθμό της ακολουθίας Fibonacci και πόσο χρόνο CPU χρειάστηκε η κάθε μία για να υπολογίσει αυτόν τον αριθμό. Όπως διακρίνεται στο παρακάτω διάγραμμα, για μικρό n δεν υπάρχουν σημαντικές διαφορές μεταξύ των συσκευών, αλλά όσο αυξάνεται το n, η συσκευή **RPI0W**, καθυστερεί περισσότερο σε σχέση με τις υπόλοιπες συσκευές, με σχεδόν διπλάσιο χρόνο σε σχέση με τα **RPI3B** και **RISCV**, ενώ οι συσκευές **Azure** και **WSL2** απαιτούν χρόνο κάτω από 1 ms μέχρι και για n = 5000.



Διάγραμμα 1: Χρόνος CPU (σε ms) που απαιτείται για την εύρεση του n-ιοστού αριθμού της ακολουθίας Fibonacci στις συσκευές που εξετάζουμε

	n =100	**n = 500**	**n = 1000**	**n = 5000**
WSL2	0.00058 ms	0.0276 ms	0.055 ms	0.377 ms
Azure	0.00848 ms	0.0375 ms	0.081 ms	0.583 ms
RISCV	0.358 ms	1.139 ms	1.362 ms	7.412 ms
RPI3B	0.0713 ms	0.381 ms	0.825 ms	7.511 ms
RPI0W	0.178 ms	0.783 ms	2.226 ms	13.868 ms

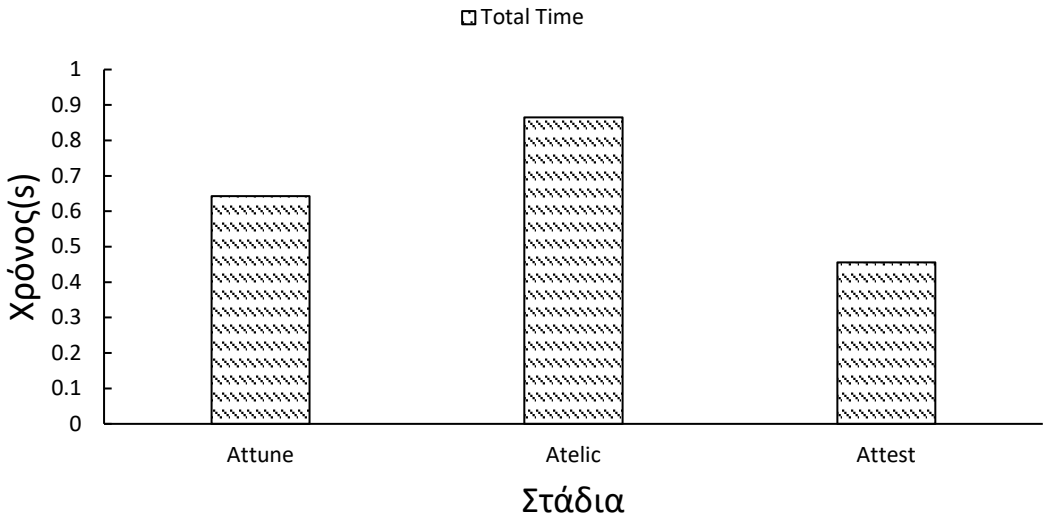
Πίνακας 2: Δεδομένα διαγράμματος 1

4.3 Χρόνοι CPU για κεντροποιημένη υλοποίηση

Για να ληφθούν πληροφορίες και για την απόδοση κεντροποιημένων υλοποιήσεων, εξετάστηκαν δύο σενάρια, ένα με την απόδοση που έχει η μη έμπιστη συσκευή (**RPI3W** με το Infineon OPTIGA™ TPM SLI 9670) για να επιβεβαιωθεί από μία άλλη έμπιστη οντότητα (**Azure**), όπως φαίνεται στην εικόνα 4, χρησιμοποιώντας την υλοποίηση [4] της Infineon, και ένα άλλο το οποίο χρησιμοποιεί το πρωτότυπο χωρίς όμως να εμπλέκει άλλες έμπιστες συσκευές.

4.3.1 Με βάση την υλοποίηση της Infineon

Οι μετρήσεις για την υλοποίηση αυτή έγιναν με χρήση της συνάρτησης time πριν την εκτέλεση κάθε script που χρησιμοποιεί η μη έμπιστη συσκευή, σε αναλογία με τα στάδια του πρωτότυπου, ο χρόνος CPU είναι το άθροισμα του χρόνου user (χρόνος CPU εκτός πυρήνα ΛΣ) με τον χρόνο sys (χρόνος CPU εντός πυρήνα ΛΣ). Παρατηρούμε πως για την συγκεκριμένη υλοποίηση, το στάδιο Atelic, που περιλαμβάνει και την ανάκτηση της πρόκλησης του Server και την δημιουργία Quote, ήταν και το πιο χρονοβόρο με μέσο χρόνο CPU 0.865 s, ακολουθεί στη συνέχεια το στάδιο Attune με χρόνο 0.643 s και πιο γρήγορο ήταν το στάδιο Attest με χρόνο 0.456 s. Ο συνολικός χρόνος CPU για την ολοκλήρωση της διαδικασίας ανέρχεται στα 1.964 s.



Διάγραμμα 2: Ο συνολικός χρόνος CPU για κάθε στάδιο της διαδικασίας επιβεβαίωσης στην μη έμπιστη συσκευή στην υλοποίηση της Infineon.

	Attune	**Atelic**	**Attest**
Συν. Χρόνος CPU	0.643 s	0.865 s	0.456 s

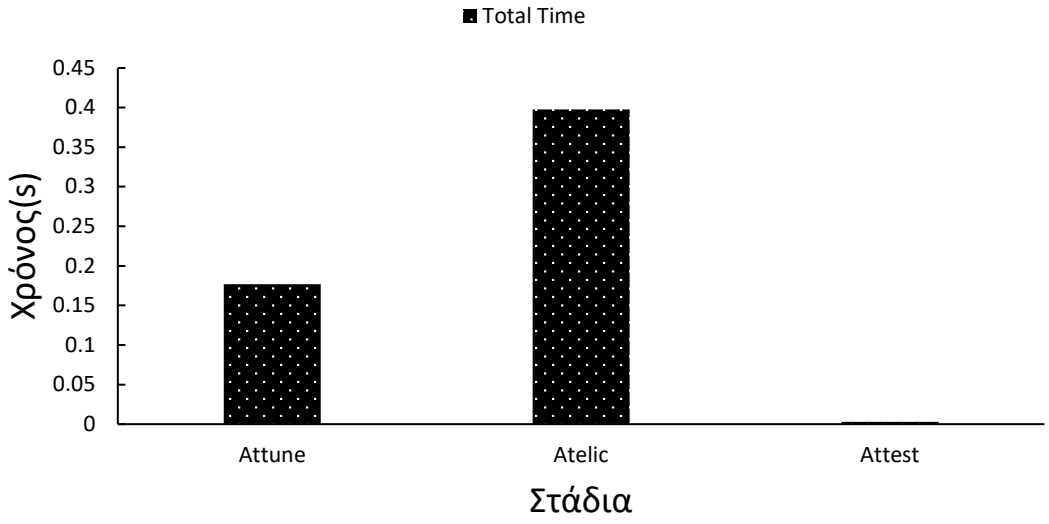
Πίνακας 3: Δεδομένα διαγράμματος 2

Από το παραπάνω διάγραμμα εξάγεται το συμπέρασμα πως το στάδιο Atelic είναι το πιο χρονοβόρο με χρόνο CPU που πλησιάζει τα 0.9 s, ενώ συνολικά για την ολοκλήρωση της επιβεβαίωσης, απαιτείται χρόνος CPU που ανέρχεται κοντά στα 2 s.

4.3.2 Με βάση το πρωτότυπο

Με τον ίδιο τρόπο δημιουργήθηκε και μία κεντροποιημένη υλοποίηση με το πρωτότυπο, χωρίς την χρήση άλλων έμπιστων συσκευών για να πραγματοποιήσουν απομακρυσμένη επιβεβαίωση στην μη έμπιστη συσκευή. Όπως φαίνεται από το διάγραμμα, η Atelic ήταν και πάλι η πιο χρονοβόρα συνάρτηση, πράγμα αναμενόμενο

καθώς εκεί γίνεται η αποκρυπτογράφηση της πρόκλησης του Server και η δημιουργία του Quote για το επόμενο στάδιο. Οι διαφορές με την υλοποίηση της Infineon εντοπίζονται κυρίως στο ότι η επικοινωνία γίνεται απευθείας μεταξύ έμπιστης και μη έμπιστης συσκευής χωρίς την ανάγκη μορφοποίησης των δεδομένων στο πρωτότυπο, ενώ αντίθετα η Infineon στην υλοποίησή της είναι απαραίτητο να μορφοποιήσει τα δεδομένα σε μορφή json πριν τα αποστείλει στον Server.



Διάγραμμα 3: Ο συνολικός χρόνος CPU για κάθε στάδιο της διαδικασίας επιβεβαίωσης στην μη έμπιστη συσκευή για την υλοποίησή μας.

	Attune	**Atelic**	**Attest**
Συν. Χρόνος CPU	0.177 s	0.398 s	0.003 s

Πίνακας 4: Δεδομένα διαγράμματος 3

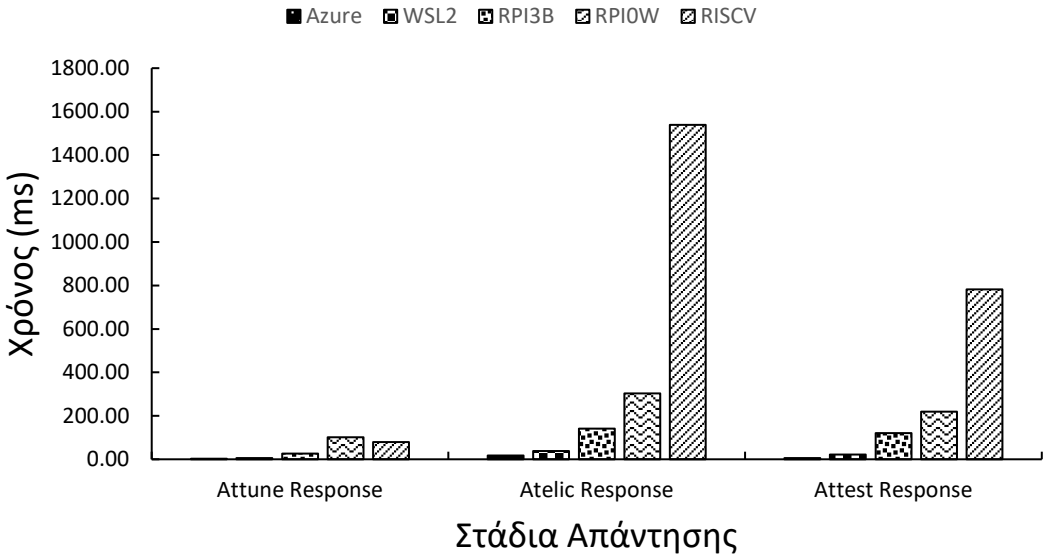
Από το παραπάνω διάγραμμα εξάγεται το συμπέρασμα πως το στάδιο Atelic είναι το πιο χρονοβόρο, ενώ το στάδιο Attest είναι το πιο γρήγορο, με την όλη διαδικασία επιβεβαίωσης να ολοκληρώνεται σε χρόνο μικρότερο των 0.7 s.

4.4 Χρόνοι CPU για κατανεμημένη υλοποίηση

Ο χρόνος CPU είναι ο χρόνος που χρειάζεται η CPU για να επεξεργαστεί τις εντολές ενός προγράμματος και δεν περιλαμβάνει χρόνο που ο επεξεργαστής περιμένει για είσοδο ή αναστέλλεται για να επιτρέψει σε άλλα προγράμματα να τρέξουν.

Στο παρακάτω διάγραμμα παρουσιάζεται ο χρόνος σε ms που χρειάζεται ο επεξεργαστής κάθε έμπιστης συσκευής προκειμένου να απαντήσει στην μη έμπιστη συσκευή. Παρατηρείται πως τα στάδια Atelic Response και Attest Response είναι αυτά που έχουν την μεγαλύτερη καθυστέρηση. Στο 2ο στάδιο (Atelic Response) και πιο συγκεκριμένα στην ρουτίνα makeCredential, συναντάται η μεγαλύτερη καθυστέρηση. Η

μέθοδος αυτή καλεί την αντίστοιχη μέθοδο της TPM για να δημιουργήσει την πρόκληση που θα δώσει στην μη έμπιστη συσκευή, για την μέθοδο αυτή, η **RISCV** συσκευή απαιτεί έως και 1560 ms για να την ολοκληρώσει, όταν την ίδια στιγμή η **RPI0W** απαιτεί 290 ms, με τις **RPI3B**, **WSL2** και **Azure** να ακολουθούν με 140 ms, 30 ms και 10 ms αντίστοιχα. Στο 3ο στάδιο υπάρχουν και εκεί αντίστοιχες κλήσεις συναρτήσεων όπως της tpm2_checkquote και της tpm2_print. Και πάλι η **RISCV** συσκευή χρειάζεται για αυτές τις μεθόδους 390 ms και 210 ms αντίστοιχα για να τις ολοκληρώσει, ενώ την ίδια στιγμή όλες οι υπόλοιπες συσκευές χρειάζονται λιγότερο από 100 ms για κάθε μέθοδο.



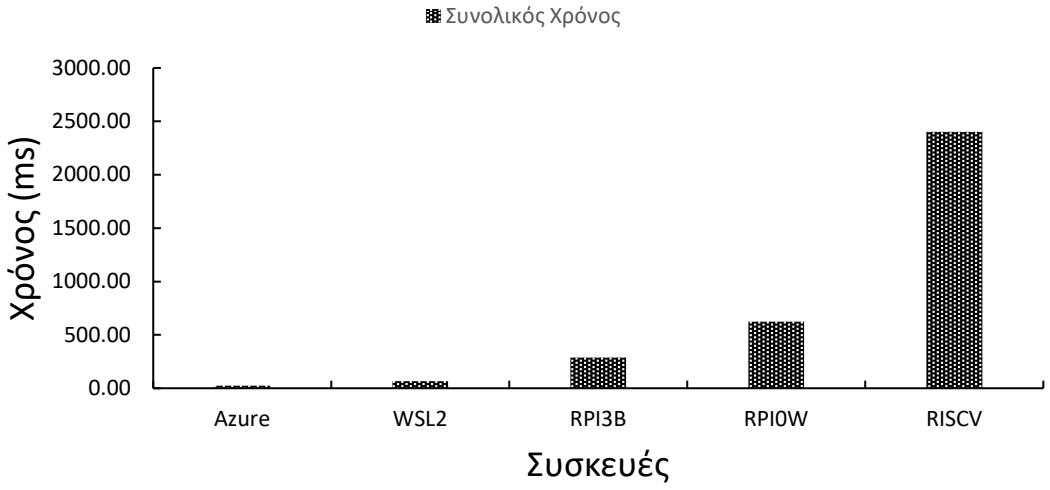
Διάγραμμα 4: Χρόνος CPU για κάθε στάδιο απάντησης ανά έμπιστη συσκευή

	Attune Response	**Atelic Response**	**Attest Response**
Azure	2.77 ms	17.43 ms	5.11 ms
WSL2	5.71 ms	38.60 ms	21.27 ms
RPI3B	26.79 ms	141.77 ms	119.88 ms
RPI0W	101.68 ms	302.99 ms	219.44 ms
RISCV	79.18 ms	1539.69 ms	781.63 ms

Πίνακας 5: Δεδομένα διαγράμματος 4

Συμπερασματικά, από το διάγραμμα 4, προκύπτει πως το πιο χρονοβόρο στάδιο απάντησης είναι το Atelic Response, κάτι το οποίο ισχύει για κάθε έμπιστη συσκευή, ενώ ακολουθούν τα στάδια Attest Response και Attune Response.

Στο παρακάτω διάγραμμα παρουσιάζεται ο συνολικός χρόνος CPU που απαιτείται από κάθε έμπιστη συσκευή προκειμένου να επιβεβαιώθει μία μη έμπιστη συσκευή. Όπως διαπιστώθηκε και στο διάγραμμα 1, ο χρόνος CPU που χρειάζονται **Azure** και **WSL2** είναι λιγότερος από 100 ms. Αμέσως μετά ακολουθούν το **RPI3B** και το **RPI0W** με μέγιστο χρόνο στα 288.44 ms και 624.11 ms αντίστοιχα, ενώ με την μεγαλύτερη διαφορά το **RISCV** είναι τελευταίο με χρόνο που φτάνει τα 2400.49 ms.



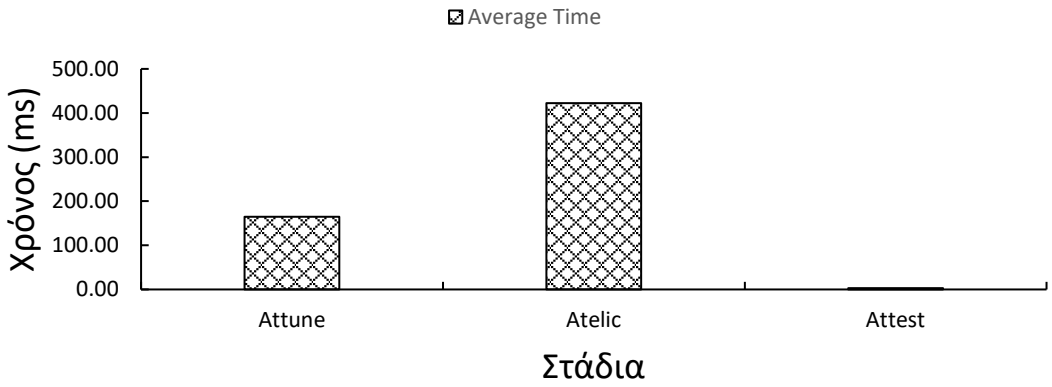
Διάγραμμα 5: Συνολικός Χρόνος CPU για επιβεβαίωση μίας μη έμπιστης συσκευής

	Azure	**WSL2**	**RPI3B**	**RPI0W**	**RISCV**
Συν. Χρόνος CPU	25.31 ms	65.58 ms	288.44 ms	624.11 ms	2400.49 ms

Πίνακας 6: Δεδομένα διαγράμματος 5

Από το παραπάνω διάγραμμα εξάγεται το συμπέρασμα πως η **RISCV** συσκευή είναι αυτή με την μεγαλύτερη καθυστέρηση έναντι όλων των υπόλοιπων έμπιστων συσκευών με συνολικό χρόνο CPU που φτάνει τα 2400 ms, ενώ ακολουθούν οι ARM συσκευές με χρόνο μικρότερο των 650 ms και με την λιγότερη καθυστέρηση οι συσκευές με επεξεργαστές x86 με χρόνο μικρότερο των 66 ms.

Αντίστοιχα, αναδεικνέται και ο μέσος χρόνος CPU που χρειάζεται η μη έμπιστη συσκευή (**RPI3B**) ώστε να πραγματοποιήσει όλα τα στάδια της απομακρυσμένης επιβεβαίωσης. Εδώ παρατηρείται ότι το πιο αργό στάδιο είναι το Atelic (2ο στάδιο), με μέσο χρόνο περίπου 422.07 ms, εκεί η μη έμπιστη συσκευή αποκρυπτογραφεί το challenge που έδωσε η έμπιστη συσκευή και δημιουργεί το quote, την ασφαλή αναφορά που θα στείλει στο επόμενο στάδιο. Γενικότερα, διακρίνεται ότι ο μέσος χρόνος CPU για να ολοκληρωθεί η απομακρυσμένη επιβεβαίωση με κάθε έμπιστη συσκευή ανέρχεται στα 589.93 ms.



Διάγραμμα 6: Μέσος Χρόνος CPU για κάθε στάδιο στην μη έμπιστη συσκευή

	Attune	**Atelic**	**Attest**
RPI3B	165.12 ms	422.07 ms	2.74 ms

Πίνακας 7: Δεδομένα διαγράμματος 6

Από το παραπάνω διάγραμμα, συμπερασματικά προκύπτει πως η πιο χρονοβόρα διαδικασία είναι το στάδιο Atelic με χρόνο CPU μεγαλύτερο των 400 ms, με την αμέσως ταχύτερη να είναι το στάδιο Attune, ενώ η πιο γρήγορη και άμεση να είναι το στάδιο Attest με χρόνο μικρότερο των 3 ms, και συνολικό μέσο χρόνο CPU μικρότερο των 600 ms για κάθε έμπιστη συσκευή.

Κεφάλαιο 5. Συμπεράσματα

Με αυτή την διπλωματική αντιμετωπίστηκε η πρόκληση της απομακρυσμένης επιβεβαίωσης εμπιστεύσιμου υλικολογισμικού για συσκευές του διαδικτύου των πραγμάτων (IoT) χωρίς την ανάγκη μιας κεντρικής αρχής. Δόθηκε μία εισαγωγή σχετικά με τα στοιχεία που απαιτούνται για την απομακρυσμένη επιβεβαίωση και έγινε μία επισκόπηση των τεχνικών που χρησιμοποιούνται σε κεντροποιημένα και κατανεμημένα συστήματα (Κεφάλαιο 2). Παρουσιάστηκε ένα πρωτότυπο κατανεμημένο σύστημα επιβεβαίωσης (Κεφάλαιο 3), η αρχιτεκτονική του (Εικόνα 10), ο τρόπος λειτουργίας του και πιθανές μελλοντικές επεκτάσεις για το πρωτότυπο αυτό. Επίσης, αξιολογήθηκε το πρωτότυπο κατανεμημένο σύστημα επιβεβαίωσης σε έμπιστες συσκευές διαφόρων αρχιτεκτονικών CPU όπως ARM, x86 και RISCV, τόσο σε πραγματικές συσκευές όπως Raspberry Pi 3B (RPI3B) και Raspberry Pi Zero W, όσο και σε εικονικές συσκευές όπως Azure VM, WSL2 σε PC και QEMU RISCV Emulator σε PC καθώς και με την χρήση μίας πραγματικής μονάδας TPM (Infineon OPTIGA TPM2.0) για την μη έμπιστη συσκευή και εικονικών μονάδων TPM (IBM Software TPM) για τις έμπιστες συσκευές (Κεφάλαιο 4), όπου προέκυψαν χρόνοι CPU για την επιβεβαίωση μίας μη έμπιστης συσκευής κάτω των 70 ms στις x86 αρχιτεκτονικές, κάτω των 625 ms για τις ARM αρχιτεκτονικές, και 2400 ms για την RISCV αρχιτεκτονική (Διάγραμμα 5), ενώ η μη έμπιστη συσκευή (RPI3B) απαίτησε έναν μέσο χρόνο CPU που ανερχόταν στα 600 ms ανά επιβεβαίωση με μία έμπιστη συσκευή (Διάγραμμα 6). Συγκριτικά, η κατανεμημένη υλοποίηση απαίτησε μέχρι και 3 φορές λιγότερο χρόνο CPU για την ολοκλήρωση της διαδικασίας σε σχέση με μία αντίστοιχη κεντροποιήμενη υλοποίηση στο Raspberry Pi 3B (Διαγράμματα 2 και 6).

Βιβλιογραφία

[1] Ρ. Μπαμπατσίκου, “ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ ΓΙΑ ΤΟ ΜΑΘΗΜΑ ΤΗΛΕΜΑΤΙΚΗ ΚΑΙ ΝΕΕΣ ΥΠΗΡΕΣΙΕΣ ΠΑΝΕΠΙΣΤΉΜΙΟ ΠΑΤΡΏΝ ΠΟΛΥΤΕΧΝΙΚΉ ΣΧΟΛΉ ΤΜΉΜΑ ΜΗΧΑΝΙΚΏΝ ΗΛΕΚΤΡΟΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΏΝ & ΠΛΗΡΟΦΟΡΙΚΉΣ,” 2020. Available: https://telematics.upatras.gr/telematics/system/files/bouras_site/ergasies_foithtwn/Babatsikou_thlematiki.pdf

[2] K. Gagandeep and B. Ranbir Singh, “Edge Computing: Classification, Applications, and Challenges,” in 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), IEEE, Apr. 2021, pp. 254–259.

[3] F. Baldassari, “Secure firmware updates with code signing,” Interrupt, Sep. 08, 2020. https://interrupt.memfault.com/blog/secure-firmware-updates-with-code-signing

[4] Infineon, “OPTIGATM TPM Application Note,” github.com, Nov. 26, 2020. https://raw.githubusercontent.com/Infineon/remote-attestation-optiga-tpm/master/documents/tpm-appnote-ra.pdf

[5] X. Zhang, K. Qin, S. Qu, T. Wang, C. Zhang, and D. Gu, “Teamwork Makes TEE Work: Open and Resilient Remote Attestation on Decentralized Trust,” arXiv.org, Feb. 13, 2024. https://arxiv.org/abs/2402.08908

[6] K. Qian, Y. Liu, X. He, M. Du, S. Zhang, and K. Wang, “HPCchain: A Consortium Blockchain System Based on CPU-FPGA Hybrid-PUF for Industrial Internet of Things,” IEEE Transactions on Industrial Informatics, vol. 19, no. 11, pp. 11205–11215, Nov. 2023, doi: https://doi.org/10.1109/tii.2023.3244339.

[7] W. Arthur, D. Challener, and K. Goldman, A Practical Guide to TPM 2.0 Using the New Trusted Platform Module in the New Age of Security. Berkeley, Ca Apress, 2015.

[8] Trusted Computing Group, “TCG Trusted Platform Module Library Part 1: Architecture Family ‘2.0’ Level 00 Revision 01.59 TCG Published,” Nov. 2019. Accessed:

Jan. 24, 2024. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf

[9] H. Sidhpurwala, “How to use the Linux kernel’s Integrity Measurement Architecture,” www.redhat.com, Oct. 22, 2020. https://www.redhat.com/en/blog/how-use-linux-kernels-integrity-measurement-architecture (accessed Jan. 14, 2024).

[10] L. Jain and J. Vyas, “Security Analysis of Remote Attestation,” Stanford University. Available: https://seclab.stanford.edu/pcl/cs259/projects/cs259_final_lavina_jayesh/CS259_report_lavina_jayesh.pdf

[11] F. Harer and H.-G. Fill, “Decentralized Attestation and Distribution of Information Using Blockchains and Multi-Protocol Storage,” IEEE Access, vol. 10, pp. 18035–18054, 2022, doi: https://doi.org/10.1109/access.2022.3150356.

[12] Y. Li, Q. Zhou, B. Li, and Y. Zhuang, “CFRV: A Decentralized Control-Flow Attestation Schema Using Mutual Secret Sharing,” Sensors, vol. 22, no. 16, p. 6044, Aug. 2022, doi: https://doi.org/10.3390/s22166044.

[13] Hyperledger Foundation, “White Papers - Hyperledger Foundation,” Hyperledger Foundation, Aug. 2018. Available: https://8112310.fs1.hubspotusercontent-na1.net/hubfs/8112310/Hyperledger/Offers/HL_Whitepaper_IntroductiontoHyperledger.pdf

[14] F. Lampayan, “How to setup TPM-simulator in Ubuntu 20.04,” Medium, Sep. 07, 2021. https://francislampayan.medium.com/how-to-setup-tpm-simulator-in-ubuntu-20-04-25ec673b88dc (accessed Feb. 21, 2024).

[15] J. Ménétrey et all., "Attestation mechanisms for trusted execution environments demystified," in Distributed Applications and Interoperable Systems, D. Eyers and S. Voulgaris, Eds. Cham: Springer International Publishing, 2022, pp. 95-113

[16] S. W. Smith, Attestation. Boston, MA: Springer US, 2011, pp 50-51. [Online] Available: https://doi.org/10.1007/978-1-4419-5906-5_697

Παράρτημα Α

Οδηγίες ρύθμισης της Απομακρυσμένης Επιβεβαίωσης

Οι οδηγίες που παραθέτουμε βασίζονται στις πληροφορίες που δίνει το Application Note της Infineon [4] και για όποια προβλήματα αντιμετωπίσαμε, αναφέρουμε εδώ τα βήματα για την επίλυσή τους.

Α.1 Προϋποθέσεις

Έχοντας ρυθμίσει τις παραπάνω συσκευές όπως θα περιγράψουμε παρακάτω, θα παρουσιάσουμε και το σύστημα κεντρικής απομακρυσμένης επιβεβαίωσης.

Αρχικά θα χρειαστούμε

1. Ένα από τα Raspberry Pi παραπάνω
2. Μία microSD (≥8GB) με το Raspberry Pi OS (raspbian-2020-02-14)
3. Μία host συσκευή με Ubuntu 18.04 LTS (στην υλοποίηση μας χρησιμοποιήσαμε μία εικονική μηχανή)
4. Το Infineon OPTIGA TPM 2.0

Α.2 Μεταγλώττιση πυρήνα Linux

Για αρχή θα χρειαστούμε τα παρακάτω προγράμματα στην host συσκευή

```
001 sudo apt install git bc bison flex libssl-dev make libc6-dev
    libncurses5-dev libncurses5-dev
```

Τα παρακάτω toolchains για μεταγλώττιση σε ARM

```
001 git clone https://github.com/raspberrypi/tools ~/tools
002 export PATH=$PATH:~/tools/arm-bcm2708/arm-linux-gnueabihf/bin
```

Και το Linux Kernel

```
001 git clone -b rpi-4.19.y https://github.com/raspberrypi/linux
002 cd linux
003 git checkout raspberrypi-kernel_1.20200601-1
```

Για το Raspberry Pi 3, θα πρέπει να κάνουμε τα παρακάτω.

```
001 KERNEL=kernel7
002 makeARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
    bcm2709_defconfig
003 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

Ενώ για το Raspberry Pi Zero τα παρακάτω.

```
001 KERNEL=kernel
002 makeARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
    bcmrpi_defconfig
003 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

Στο βήμα 003 θα χρειαστεί να κάνουμε κάποιες αλλαγές στον πυρήνα πριν τον μεταγλωττίσουμε, συγκεκριμένα να ενεργοποιήσουμε το IMA module ενεργοποιώντας τις παρακάτω επιλογές (στο Raspberry Pi Zero, δεν υπάρχει η δυνατότητα για χρήση του αλγόριθμου κατακερματισμού SHA256, παρά μόνο SHA1).

```
Security options --->
[*] Enable different security models
[*] Integrity subsystem
[*]   Integrity Measurement Architecture(IMA)
        Default template (ima-sig) --->
        Default integrity hash algorithm (SHA256) --->
[*]     Enable multiple writes to the IMA policy
[*]     Enable reading back the current IMA policy
```

Στη συνέχεια θα κάνουμε μερικές αλλαγές για να ενεργοποιηθούν τα SPI και TPM πριν την ενεργοποίηση του IMA.

Ρύθμιση του TPM ως ενσωματωμένο module:

```
Device Drivers --->
Character devices --->
-*-  TPM Hardware Support --->
<*>    TPM Interface Specification 1.3 Interface / TPM 2.0 FIFO
       Interface - (SPI)
```

Ρύθμιση του SPI ως ενσωματωμένο module:

```
Device Drivers --->
[*] SPI support --->
<*>     BCM2835 SPI controller
```

Μετά την ολοκλήρωση των παραπάνω, αποθηκεύουμε και βγαίνουμε, από αυτό το μενού.

Έπειτα στο αρχείο ***drivers/clk/bcm/clk-bcm2835.c*** αντικαθιστούμε την γραμμή παρακάτω

```
--- postcore_initcall(__bcm2835_clk_driver_init);
+++ subsys_initcall(__bcm2835_clk_driver_init);
```

Και στο αρχείο ***security/integrity/ima/ima_policy.c***, στην γραμμή 122 θα αντικαταστήσουμε την μέθοδο με την παρακάτω. Με αυτόν τον τρόπο η πολιτική IMA καταγράφει την ακεραιότητα των εκτελέσιμων αρχείων που ανήκουν στον χρήστη root όταν αυτά προσπελλάσσονται ή εκτελούνται.

```
static struct ima_rule_entry default_measurement_rules[]
__ro_after_init = {
       {.action = MEASURE, .func = FILE_CHECK, .mask = MAY_EXEC,
       .uid = GLOBAL_ROOT_UID, .uid_op = &uid_eq,
       .fowner = GLOBAL_ROOT_UID, .fowner_op = &uid_eq,
       .flags = IMA_FUNC | IMA_MASK | IMA_UID | IMA_FOWNER},
};
```

Μετά τα παραπάνω βήματα μπορούμε να συνεχίσουμε στο βήμα 004 του τελευταίου μέρους, μεταγλωττίζοντας τον πυρήνα Linux.

```
004 make -j$(nproc) ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
    zImage modules dtbs
```

Περνάμε το Raspbian στην microSD με το Raspberry Pi Imager και μεταφέρουμε τα kernel modules, το kernel image και τα device tree blobs στην microSD (κάνοντας τις απαραίτητες αλλαγές στα sdbX και sdbY).

```
001 mkdir mnt
002 mkdir mnt/fat32
003 mkdir mnt/ext4
004 sudo umount /dev/sdbX
005 sudo umount /dev/sdbY
006 sudo mount /dev/sdbX mnt/fat32
007 sudo mount /dev/sdbY mnt/ext4
008 sudo env PATH=$PATH make ARCH=arm CROSS_COMPILE=arm-linux-
    gnueabihf- INSTALL_MOD_PATH=mnt/ext4 modules_install
009 sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
010 sudo cp arch/arm/boot/zImage mnt/fat32/$KERNEL.img
011 sudo cp arch/arm/boot/dts/*.dtb mnt/fat32/
012 sudo cp arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
013 sudo cp arch/arm/boot/dts/overlays/README mnt/fat32/overlays/
014 sudo umount mnt/fat32
015 sudo umount mnt/ext4
```

A.3 Ενεργοποίηση του TPM και του IMA

Τοποθετούμε την microSD στο Raspberry Pi και το εκκινούμε. Όταν μπούμε στο περιβάλλον εργασίας, κάνουμε τις παρακάτω αλλαγές στα αρχεία config.txt και cmdline.txt. Μετά τις αλλαγές αυτές, επαννεκινούμε το Pi.

- Ανοίγουμε το config.txt αρχείο και εισάγουμε τις παρακάτω γραμμές

```
001 sudo nano /boot/config.txt
+++ dtparam=spi=on
+++ dtoverlay=tpm-slb9670
```

- Ανοίγουμε το cmdline.txt αρχείο και τοποθετούμε στο τέλος το παρακάτω

```
002 sudo nano /boot/cmdline.txt
+++ ima_policy=tcb
```

Ελέγχουμε αν το TPM είναι ενεργοποιημένο

```
001 ls /dev | grep tpm
tpm0
tpmrm0
```

Ελέγχουμε αν το IMA είναι ενεργοποιημένο (πρέπει η εντολή να επιστρέψει τιμή >1)

```
005 sudo cat /sys/kernel/security/ima/runtime_measurements_count
146
```

Ελέγχουμε αν έχει ρυθμιστεί σωστά η πολιτική IMA

```
006 sudo cat /sys/kernel/security/ima/policy
dont_measure fsmagic=0x9fa0
dont_measure fsmagic=0x62656572
dont_measure fsmagic=0x64626720
dont_measure fsmagic=0x1021994
dont_measure fsmagic=0x1cd1
dont_measure fsmagic=0x42494e4d
dont_measure fsmagic=0x73636673
dont_measure fsmagic=0xf97cff8c
dont_measure fsmagic=0x43415d53
dont_measure fsmagic=0x27e0eb
dont_measure fsmagic=0x63677270
dont_measure fsmagic=0x6e736673
measure func=FILE_CHECK mask=MAY_EXEC uid=0
```

Και τέλος, ελέγχουμε αν το πρότυπο IMA (ima-sig) και ο αλγόριθμος (SHA256) είναι σωστά ρυθμισμένα παρατηρώντας το αρχείο ascii_runtime_measurements (στο Raspberry Pi Zero W, θα δείχνει μόνο SHA1 μετρήσεις)

```
007 sudo cat /sys/kernel/security/ima/ascii_runtime_measurements
10 <20 bytes of hash value> ima-sig sha1:<20 bytes of hash value>
boot_aggregate
```

10 <20 bytes of hash value> ima-sig sha256:<32 bytes of hash value> <filename with path>

…

A.4 Εγκατάσταση του λογισμικού για το TPM στην συσκευή

Θα χρειαστεί να εγκαταστήσουμε το παρακάτω λογισμικό

tpm2-tss	https://github.com/tpm2-software/tpm2-tss
tpm2-tools	https://github.com/tpm2-software/tpm2-tools

A.4.1 Ρύθμιση του TPM2 Software Stack

Θα χρειαστεί να εκτελέσουμε στο Pi τις ακόλουθες εντολές

```
001 sudo useradd --system --user-group tss
002 sudo apt update
003 sudo apt -y install autoconf-archive libcmocka0 libcmocka-dev
    procps iproute2 build-essential git pkg-config gcc libtool
    automake libssl-dev uthash-dev autoconf doxygen libgcrypt-dev
    libjson-c-dev libcurl4-gnutls-dev uuid-dev pandoc
```

Για την εγκατάσταση θα χρειαστεί να έχουμε εγκαταστήσει πρώτα τα παρακάτω

```
004 sudo apt -y install autoconf-archive libcmocka0 libcmocka-dev
    procps iproute2 build-essential git pkg-config gcc libtool
    automake libssl-dev uthash-dev autoconf doxygen libjson-c-dev
    libgcrypt20-dev libcurl4-gnutls-dev acl
```

Στη συνέχεια μπορούμε να προχωρήσουμε στην εγκατάσταση του tpm2-tss, με την έκδοση 2.4.0 είχαμε προβλήματα, γι' αυτό χρησιμοποιήσαμε την έκδοση 3.0.x

```
005 git clone -b 3.0.x https://github.com/tpm2-software/tpm2-tss.git
006 cd tpm2-tss
007 ./bootstrap
008 ./configure --with-udevrulesdir=/etc/udev/rules.d --with-
    udevrulesprefix=70- --sysconfdir=/etc --localstatedir=/var
    --runstatedir=/run
009 make
010 sudo make install
```

Και με την ολοκλήρωση της εγκατάστασης πρέπει να τρέξουμε τις εντολές παρακάτω

```
011 sudo udevadm control --reload-rules && sudo udevadm trigger
012 sudo ldconfig
```

A.4.2 Ρύθμιση του TPM2 Tools

Για την εγκατάσταση θα χρειαστεί να έχουμε εγκαταστήσει πρώτα τα παρακάτω

```
013 sudo apt -y install autoconf automake libtool pkg-config gcc
    libssl-dev libcurl4-gnutls-dev pandoc python-yaml expect
```

Και στη συνέχεια μπορούμε να προχωρήσουμε στην εγκατάσταση του tpm2-tools

```
014 git clone -b 4.2.x https://github.com/tpm2-software/tpm2-
    tools.git
015 cd tpm2-tools/
016 ./bootstrap
017 ./configure
018 make
019 sudo make install
020 sudo ldconfig
```

A.5 Εγκατάσταση του λογισμικού για την απομακρυσμένη επιβεβαίωση στην συσκευή

Το λογισμικό για την απομακρυσμένη επιβεβαίωση στην συσκευή περιλαμβάνει τις εφαρμογές για την επικοινωνία με τον server καθώς και τις εντολές για την πραγματοποίηση απομακρυσμένης επιβεβαίωσης βήμα-βήμα.

Αρχικά θα χρειαστεί να έχουμε εγκαταστήσει τα παρακάτω πακέτα

```
001 sudo apt update
002 sudo apt install libconfig-dev libjson-c-dev libcurl4-gnutls-dev
```

Στην συνέχεια θα χρειαστούμε το λογισμικό απομακρυσμένης επιβεβαίωσης

```
003 git clone https://github.com/infineon/remote-attestation-optiga-
    tpm -b device
004 cd remote-attestation-optiga-tpm
```

Θα μεταβάλλουμε το αρχείο config.cfg ώστε να δείχνει στον Server, και στην συνέχεια μεταγλωττίζουμε

```
005 nano config.cfg
006 make
```

A.6 Εγκατάσταση λογισμικού για τον Server

Το ακόλουθο λογισμικό θα χρειαστεί να εγκατασταθεί στον Server, ο οποίος θα είναι υπεύθυνος για την προβολή ενός web περιβάλλοντος όπου θα δείχνει αν η συσκευή έκανε επιτυχώς απομακρυσμένη επιβεβαίωση ή όχι καθώς και για να δημιουργεί την πρόκληση που θα έχει να λύσει η συσκευή προκειμένου η γίνει η επιβεβαίωση. Στην περίπτωση μας το λογισμικό αυτό το έχουμε σε μία cloud εικονική μηχανή (Azure) με Ubuntu 20.04.6 LTS.

```
001 git clone https://github.com/infineon/remote-attestation-optiga-
    tpm -b server
002 cd remote-attestation-optiga-tpm
003 wget https://download.java.net/java/GA/jdk9/9.0.4/binaries/
    openjdk-9.0.4_linux- x64_bin.tar.gz
004 tar -zxvf openjdk-9.0.4_linux-x64_bin.tar.gz
005 sudo mv jdk-9.0.4 /opt/
006 sudo update-alternatives --install /usr/bin/java java /opt/jdk
    -9.0.4/bin/java 1
007 sudo update-alternatives --install /usr/bin/javac javac
    /opt/jdk-9.0.4/bin/javac 1
008 sudo update-alternatives --config java
009 sudo update-alternatives --config javac
010 sudo apt install maven
011 mvn package
```

A.7 Εκκίνηση Server

Στον Server θα πάμε στην παρακάτω διαδρομή και θα εκτελέσουμε το .jar αρχείο.

```
001 cd remote-attestation-optiga-tpm/server/target
002 sudo java -jar server-0.0.1-SNAPSHOT.jar
```

Ο Server θα εκτελείται κανονικά μετά την εμφάνιση των παρακάτω γραμμών

```
...
2020-06-10 22:37:51.856 INFO 12828 --- [ main]
o.s.m.s.b.SimpleBrokerMessageHandler : Started.
2020-06-10 22:37:52.414 INFO 12828 --- [ main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 443
(https)
80 (http) with context path ''
2020-06-10 22:37:52.418 INFO 12828 --- [ main]
com.ifx.server.ServerApplication : Started ServerApplication in 91.269
seconds (JVM running for 98.966)
```

Για εμφάνιση του web interface πηγαίνουμε στην διεύθυνση https://10.147.18.101. Θα μας εμφανίσει μία προειδοποίηση για self-signed πιστοποιητικό την οποία θα αγνοήσουμε. Στο επάνω μενού, πατάμε «start» και βάζουμε τα παρακάτω στοιχεία για να συνδεθούμε.

Username	infineon
Password	password

A.8 Προετοιμασία TPM

Τα ακόλουθα βήματα θα γίνουν στην συσκευή με το TPM.

Εκκαθάριση ιεραρχίας TPM:

```
001 sudo chmod a+rw /dev/tpm0
002 sudo chmod a+rw /dev/tpmrm0
003 tpm2_clear -c p
```

Παραγωγή ενός TCG συμβεβλημένου κλειδιού επικύρωσης EK (endorsement key) και αποθήκευση στα διατηρητέα κλειδιά:

```
004 tpm2_createek -G rsa -u ek.pub -c ek.ctx
005 tpm2_evictcontrol -C o -c ek.ctx 0x81010001
```

Παραγωγή ενός κλειδιού επιβεβαίωσης AK (attestation key) και αποθήκευση στα διατηρητέα κλειδιά:

```
006 tpm2_createak -C 0x81010001 -c ak.ctx -G rsa -g sha256 -s rsassa
    -u ak.pub -n ak.name
007 tpm2_evictcontrol -C o -c ak.ctx 0x81000002
```

Επαλήθευση των παραγόμενων κλειδιών με ανάγνωση των handles για τα διατηρητέα κλειδιά στο TPM:

```
008 tpm2_getcap handles-persistent
0x81000002
0x81010001
```

A.9 Εκτέλεση των script για απομακρυσμένη επιβεβαίωση

Πηγαίνουμε στον παρακάτω κατάλογο

```
001 cd remote-attestation-optiga-tpm/bin
```

Και εκτελούμε διαδοχικά τα παρακάτω script:

0_prep.sh	Έγκριση μη-προνομιακής πρόσβασης στο TPM.
1_cleanup.sh	Διαγραφή περιττών αρχείων και επαναφορά αρχείου ρύθμισης.
2_pcr.sh	Ανάγνωση των τιμών PCR του TPM και του αρχείου καταγραφής IMA.
3_attune.sh	Εγγραφή έγκυρων τιμών PCR στον Server.
4_atelic.sh	Αίτηση κρυπτογραφημένης πρόκλησης.
5_credential.sh	Αποκρυπτογράφηση της πρόκλησης με το tpm2_activatecredential.
6_quote.sh	Δημιουργία quote και υπογραφής με το tpm2_quote. Αγνοείστε το 6_quote_bad.sh
6_quote-bad.sh	Για δημιουργία απόρριψης με άκυρη πρόκληση. Αγνοείστε το 6_quote.sh
7_attest.sh	Αποστολή του quote, της υπογραφής και του τελευταίου αρχείου καταγραφής IMA στον server για απομακρυσμένη επιβεβαίωση.

Παράρτημα Β

Οδηγίες ρύθμισης εικονικών μονάδων TPM

Οι οδηγίες που παραθέτουμε αφορούν την ρύθμιση της εικονικής μονάδας TPM που χρησιμοποιήσαμε στις έμπιστες συσκευές μας [14].

Β.1 Εγκατάσταση Εικονικής TPM

Πρώτα θα χρειαστεί να εγκαταστήσουμε τα παρακάτω προγράμματα

```
sudo apt-get install lcov acl \
pandoc autoconf-archive liburiparser-dev \
libdbus-1-dev libglib2.0-dev dbus-x11 \
libssl-dev autoconf automake \
libtool pkg-config gcc libjson-c-dev \
libcurl4-gnutls-dev libgcrypt20-dev libcmocka-dev uthash-dev \
```

Κατεβάζουμε την Εικονική TPM και την αποσυμπιέζουμε σε έναν φάκελο

```
wget https://jaist.dl.sourceforge.net/project/ibmswtpm2/ibmtpm1682.tar.gz
mkdir ibmtpm1682
cd ibmtpm1682
tar -xzvf ../ibmtpm1682.tar.gz
```

Κάνουμε compile την εφαρμογή

```
cd src/
sudo make
```

Αντιγράφουμε το εκτελέσιμο στον φάκελο bin

```
sudo cp tpm_server /usr/local/bin
```

Δημιουργούμε μία υπηρεσία για εικονική TPM

```
sudo nano /lib/systemd/system/tpm-server.service
```

Προσθέτουμε τις παρακάτω γραμμές στο αρχείο

```
[Unit]
Description=TPM2.0 Simulator Server daemon
Before=tpm2-abrmd.service
[Service]
ExecStart=/usr/local/bin/tpm_server
Restart=always
Environment=PATH=/usr/bin:/usr/local/bin
[Install]
WantedBy=multi-user.target
```

Εκκινούμε την υπηρεσία

```
systemctl daemon-reload
systemctl start tpm-server.service
```

Ελέγχουμε αν η υπηρεσία ενεργοποιήθηκε

```
service tpm-server status
```

Β.2 Εγκατάσταση tpm2-tss

Κατεβάζουμε και κάνουμε compile το tpm2-tss (έκδοση 3.1.0)

```
wget https://github.com/tpm2-software/tpm2-
tss/releases/download/3.1.0/tpm2-tss-3.1.0.tar.gz
tar -xzvf tpm2-tss-3.1.0.tar.gz
cd tpm2-tss-3.1.0/
./configure
sudo make install
sudo ldconfig
```

Β.3 Εγκατάσταση tpm2-abrmd

Κατεβάζουμε και κάνουμε compile το tpm2-abrmd (έκδοση 2.3.1)

```
wget https://github.com/tpm2-software/tpm2-
abrmd/releases/download/2.3.1/tpm2-abrmd-2.3.1.tar.gz
tar -xzvf tpm2-abrmd-2.3.1.tar.gz
cd tpm2-abrmd-2.3.1
sudo ldconfig
./configure --with-dbuspolicydir=/etc/dbus-1/system.d --with-
    systemdsystemunitdir=/usr/lib/systemd/system
sudo make install
sudo cp /usr/local/share/dbus-1/system-
    services/com.intel.tss2.Tabrmd.service /usr/share/dbus-1/system-
    services/
```

Επανεκκινούμε το DBUS

```
sudo pkill -HUP dbus-daemon
```

Ανοίγουμε το αρχείο με τις ρυθμίσεις της υπηρεσίας tpm2-abrmd.service

```
sudo nano /lib/systemd/system/tpm2-abrmd.service
```

Γράφουμε τα παρακάτω και αποθηκεύουμε

```
[Unit]
Descript=TPM2 Access Broker and Resource Management Daemon

[Service]
Type=dbus
Restart=always
RestartSec=5
BusName=com.intel.tss2.Tabrmd
StandardOutput=syslog
ExecStart=/usr/local/sbin/tpm2-abrmd --tcti="libtss2-tcti-
mssim.so.0:host=127.0.0.1,port=2321"
User=tss

[Install]
WantedBy=multi-user.target
```

Εκκινούμε την υπηρεσία και ελέγχουμε την κατάσταση της

```
systemctl daemon-reload
systemctl start tpm2-abrmd.service
service tpm2-abrmd status
```

Β.4 Εγκατάσταση tpm2-tss-engine

Κατεβάζουμε και κάνουμε compile το tpm2-tss-engine (έκδοση 1.1.0)

```
wget https://github.com/tpm2-software/tpm2-tss-
    engine/releases/download/v1.1.0/tpm2-tss-engine-1.1.0.tar.gz
tar -xzvf tpm2-tss-engine-1.1.0.tar.gz
cd tpm2-tss-engine-1.1.0/
./configure
sudo make install
sudo ldconfig
```

Δοκιμή με openssl, πρέπει να βγει μία δεκαεξαδική τιμή, διαφορετικά υπάρχει πρόβλημα με την εγκατάσταση

```
openssl rand -engine tpm2tss -hex 10
```

Β.5 Εγκατάσταση tpm2-tools

Κατεβάζουμε και κάνουμε compile το tpm2-tools

(τελευταία έκδοση που χρησιμοποιήσαμε, έκδοση 5.6)

```
git clone https://github.com/tpm2-software/tpm2-tools.git
cd tpm2-tools
./bootstrap
./configure
make
sudo make install
```

Δοκιμή

```
tpm2_pcrread
```