National Technical University of Athens

School of Electrical and Computer Engineering

Division of Computer Science
Computing Systems Lab

Implementation of a Collective Remote Attestation Framework for IoT Devices

DIPLOMA THESIS

of

Faidon Cornelis Kourounakis

Supervisor: Dionisios Pnevmatikatos
Professor, NTUA

Athens, October 2025

Εθνικο Μετσοβιο Πολυτεχνειο
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Implementation of a Collective Remote Attestation Framework for IoT Devices

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Φαίδων Κορνέλις Κουρουνάκη

Επιβλέπων: Διονύσιος Πνευματικάτος
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 10 Οκτωβρίου 2025.

.....	.....	.....
Διονύσιος Πνευματικάτος	Δημήτριος Σούντρης	Γεώργιος Γκούμας
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ

Αθήνα, Οκτώβριος 2025.

.....

Φαίδων Κορνέλις Κουρουνάκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η μαζική υιοθέτηση των συσκευών IoT απαιτεί ισχυρά μέτρα ασφαλείας για τη διασφάλιση της ακεραιότητάς τους και την αποτροπή διαταραχών. Αντιμετωπίζοντας αυτή την πρόκληση, η παρούσα εργασία παρουσιάζει ένα ελαφρύ, πλαίσιο Συλλογικής Απομακρυσμένης Επαλήθευσης (Collective Remote Attestation) σχεδιασμένο για συσκευές με περιορισμένους πόρους. Στοχεύει στην εύκολη υιοθέτηση, αξιοποιώντας τον μηχανισμό Physical Memory Protection (PMP) της αρχιτεκτονικής RISC-V ως Ρίζα Εμπιστοσύνης υλικού. Το προτεινόμενο αποκεντρωμένο πρωτόκολλο διεξάγει αμοιβαία ετερογενή επαλήθευση με τοπική επαλήθευση της ακεραιότητας της μνήμης flash. Η χρήση nonces και HMAC διασφαλίζει τη φρεσκάδα και την αυθεντικότητα των μηνυμάτων, μετριάζοντας τις επιθέσεις επανάληψης. Το πλαίσιο υλοποιείται χρησιμοποιώντας το FreeRTOS και αξιολογείται στο χαμηλού κόστους, ευρέως διαθέσιμο μικροελεγκτή ESP32-C3, το οποίο υποστηρίζει τα αναγκαία χαρακτηριστικά ασφαλείας. Η αξιολόγηση απόδοσης επιβεβαίωσε ότι η χρήση της CPU κλιμακώνεται γραμμικά ανάλογα με τον αριθμό των συσκευών και τη συχνότητα των επαληθεύσεων. Το πλαίσιο παρέχει μια βάση για μελλοντική έρευνα και επεκτάσεις σε ασφαλείς αποκεντρωμένους μηχανισμούς εμπιστοσύνης σε δίκτυα IoT με περιορισμένους πόρους.

Λέξεις Κλειδιά

Συλλογική Απομακρυσμένη Επαλήθευση, Ασφάλεια IoT, ESP32, FreeRTOS, RISC-V

Abstract

The mass adoption of Internet of Things devices necessitates robust security measures to ensure their integrity and prevent widespread disruption. Addressing this challenge in IoT networks, this work presents a lightweight Collective Remote Attestation framework designed for resource-constrained devices. It targets real-world adoption by leveraging the RISC-V Physical Memory Protection mechanism as the Hardware Root of Trust. The proposed decentralized protocol conducts mutual heterogeneous attestation with local verification of flash integrity. The use of nonces and HMAC ensures message freshness and authenticity, mitigating replay attacks. The framework is implemented using FreeRTOS and evaluated on the low-cost, readily available ESP32-C3 chip that supports the necessary security features. Performance testing confirmed CPU utilization scaling linearly with device count and attestation frequency. The framework provides a basis for future research and extensions on secure, decentralized trust mechanisms in resource-constrained IoT networks.

Keywords

Collective Remote Attestation, IoT Security, ESP32-C3, FreeRTOS, RISC-V

Acknowledgements

I would like to sincerely thank Professor Dionisios Pnevmatikatos for providing me with the opportunity to undertake this research and my advisor, Foivos Iliadis, for his guidance, constructive feedback, and encouragement, which were essential for the completion of this thesis.

I would also like to acknowledge my friends and university colleagues advice, and most importantly moral support enriched this experience. Lastly, I wish to extend my heartfelt appreciation to my parents and my brother for their constant encouragement and understanding, which sustained me through the most demanding stages of this work.

Contents

List of Figures

List of Tables

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Εισαγωγή

Το Διαδίκτυο των Πραγμάτων (IoT) αναφέρεται σε ένα δίκτυο διασυνδεδεμένων συσκευών, αισθητήρων, ανθρώπων και υπηρεσιών, οι οποίες επικοινωνούν και ανταλλάσσουν δεδομένα με σκοπό την επίτευξη κοινών στόχων σε ποικίλα πεδία εφαρμογής. Η ευρεία διάδοση χαμηλού κόστους μικροελεγκτών και αισθητήρων έχει καταστήσει δυνατή την ενσωμάτωση του IoT σε πλήθος τομέων, όπως η υγειονομική περίθαλψη, όπου χρησιμοποιείται για απομακρυσμένη παρακολούθηση ασθενών και καταπιώμενες ιατρικές συσκευές· τα έξυπνα δίκτυα ενέργειας, όπου συμβάλλει στην ορθολογική διαχείριση ανανεώσιμων πηγών και την εξισορρόπηση της παραγωγής και κατανάλωσης· ο αυτοματισμός κατοικιών, για τον έλεγχο κλίματος, φωτισμού, ασφάλειας και ενεργειακής διαχείρισης· η βιομηχανία αυτοκινήτων, μέσω συστημάτων ασφαλείας, παρακολούθησης και άνεσης· καθώς και οι έξυπνες πόλεις, με εφαρμογές στη διαχείριση απορριμμάτων, τη βελτιστοποίηση της κυκλοφορίας και της δημόσιας συγκοινωνίας. Παρά τα σημαντικά οφέλη, η ραγδαία εξάπλωση των συσκευών IoT συνεπάγεται σοβαρές προκλήσεις ασφάλειας, καθώς οι ευπάθειες σε δικτυωμένα συστήματα μπορούν να εκμεταλλευτούν σε μεγάλη κλίμακα, με ενδεχόμενες συνέπειες την παραβίαση της ιδιωτικότητας των ασθενών, τον κίνδυνο για τη ζωή των ανθρώπων στις μεταφορές, την υπονόμευση της δημόσιας υποδομής ή τη χρήση τους σε εκτεταμένες κυβερνοεπιθέσεις, όπως οι επιθέσεις DDoS. Επομένως, καθίσταται αναγκαία η εφαρμογή αυστηρών προτύπων ασφάλειας και στρατηγικών μετριασμού των κινδύνων. Στο πλαίσιο αυτό, η Απομακρυσμένη Επαλήθευση (Remote Attestation) αποτελεί κρίσιμη τεχνική για την επαλήθευση της ακεραιότητας των συσκευών, ενώ η Συλλογική Απομακρυσμένη Επαλήθευση (Collective Remote Attestation) αντιμετωπίζει επιπλέον προκλήσεις σχετικές με τη διάδοση εμπιστοσύνης και την αποτελεσματική επικοινωνία σε μεγάλα δίκτυα IoT.

Η παρούσα διπλωματική εργασία συνεισφέρει με τον σχεδιασμό και την υλοποίηση ενός πλαισίου Συλλογικής Απομακρυσμένης Επαλήθευσης, το οποίο εκτελεί αμοιβαία και ετερογενή αυτο-επαλήθευση σε δίκτυα του Διαδικτύου των Πραγμάτων (IoT), χωρίς να απαιτείται η ύπαρξη κεντρικού Επαληθευτή. Το πλαίσιο έχει σχεδιαστεί για να συνοδεύεται από εγγυήσεις σε υλικό: secure boot, secure storage και Προστασία

Φυσικής Μνήμης (PMP) της αρχιτεκτονικής RISC-V. Για την υλοποίηση επιλέχθηκε ο μικροελεγκτής ESP32-C3, με σκοπό το πλαίσιο να είναι άμεσα προσαρμόσιμο στη βιομηχανία και να αποτελέσει τη βάση για περαιτέρω έρευνα στο πεδίο της Απομακρυσμένης Επαλήθευσης.

1.2 Υπόβαθρο

Αυτή η ενότητα παρουσιάζει τα βασικά τεχνικά θεμέλια για την κατανόηση της ασφάλειας σε ενσωματωμένα συστήματα IoT, καλύπτοντας ενσωματωμένα συστήματα, IoT, κρυπτογραφία και απομακρυσμένη επαλήθευση.

1.2.1 Ενσωματωμένα Συστήματα και Διαδίκτυο των Πραγμάτων

Τα ενσωματωμένα συστήματα είναι εξειδικευμένοι υπολογιστές σχεδιασμένοι για συγκεκριμένες λειτουργίες μέσα σε μεγαλύτερα συστήματα. Διαφέρουν από τους υπολογιστές γενικής χρήσης καθώς εκτελούν προκαθορισμένες εργασίες υπό συνθήκες πραγματικού χρόνου. Η δικτύωση είναι κρίσιμη για την επικοινωνία: Δικτυώνονται τόσο με ενσύρματα πρωτόκολλα (I2C, SPI και CAN) όσο και με ασύρματα (Wi-Fi, Bluetooth/BLE, Zigbee, Z-Wave, Thread) για την συνεργασία και επικοινωνία με άλλες συσκευές. Εξοπλίζονται τυπικά με Λειτουργικά Συστήματα Πραγματικού Χρόνου (RTOS) εξασφαλίζουν χρονικά προβλέψιμη εκτέλεση εργασιών μέσω αλγορίθμων χρονοπρογραμματισμού (pre-emptive, round-robin). Παραδείγματα είναι το FreeRTOS, Zephyr και VxWorks.

Το IoT αναφέρεται σε δίκτυα συσκευών που επιτρέπουν απομακρυσμένη παρακολούθηση ή έλεγχο.Τα τυπικά επίπεδα μιας συσκευής IoT είναι το επίπεδο αισθητήρων, το επίπεδο δικτύου, το επίδπεδο επεξεργασίας δεδομένων και το επίπεδο εφαρμογής.

Στρώμα Αισθητήρων Συσκευές που συλλέγουν ή ενεργούν στα δεδομένα.

Δικτυακό Στρώμα Μεταφορά δεδομένων μέσω πρωτοκόλλων όπως Wi-Fi, LoRaWAN ή κινητά.

Στρώμα Επεξεργασίας Δεδομένων Αποθήκευση και ανάλυση στο cloud ή σε edge servers.

Στρώμα Εφαρμογής Ορατό στους χρήστες, για dashboards και έλεγχο συσκευών [20].

1.2.2 RISC-V και Physical Memory Protection

Το RISC-V είναι ανοιχτού κώδικα ISA, που έχει εισχωρήσει στις IoT συσκευές χαμηλού κόστους [17]. Η Προστασία Φυσικής Μνήμης (PMP), μια δημοφιλή επέκταση του RISC-V επιτρέπει απομόνωση μνήμης με έλεγχο πρόσβασης ανά περιοχή [19] μέσω ειδικών καταχωρητών όπως φαίνεται στο Σχήμα 1.1, απλούστερη από TDX/AMD SEV ή TrustZone για συστήματα χωρίς MMU.

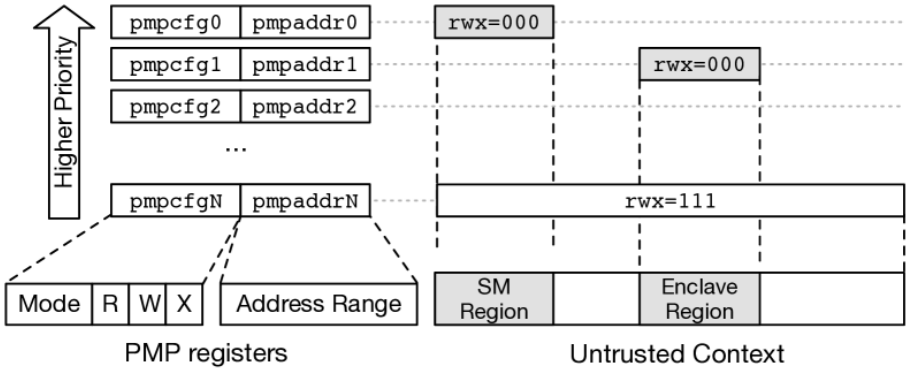

Figure 1.1: Παράδειγμα διαμόρφωσης PMP [19]

1.2.3 Κρυπτογραφία

Η κρυπτογραφία εξασφαλίζει εμπιστευτικότητα, ακεραιότητα και αυθεντικότητα μέσω των μηχανισμών:

Κατακερματισμός Μετατρέπει δεδομένα οποιουδήποτε μεγέθους σε σταθερού μέγεθους περίληψη, π.χ., SHA-256, για έλεγχο ακεραιότητας.

Ασύμμετρη Κρυπτογραφία Ζεύγη δημόσιου/ιδιωτικού κλειδιού για ασφαλή κρυπτογράφηση και ψηφιακές υπογραφές (RSA, ECC).

Κώδικα Αυθεντικοποίησης Μηνύματος MACs και HMACs πιστοποιούν μηνύματα με συμμετρικά κλειδιά.

1.2.4 Εμπιστοσύνη και Secure Boot

Η Root of Trust (RoT) είναι έμπιστο υλικό που επιτρέπει την αλυσίδα εμπιστοσύνης (CoT) για επαλήθευση συστατικών συστήματος. Το Secure Boot επαληθεύει διαδοχικά το λογισμικό προς εκτέλεση χρησιμοποιώντας κρυπτογραφικούς ελέγχους ξεκινώντας από μια Ρίζα Εμπιστοσύνης (RoT).

1.2.5 Απομακρυσμένη Επαλήθευση

Η Απομακρυσμένη Επαλήθευση (RA) επιτρέπει σε έναν επαληθευτή να ελέγξει την ακεραιότητα μιας απομακρυσμένης συσκευής, βασιζόμενη σε μια Root of Trust (RoT) που προστατεύει κρίσιμες λειτουργίες. Ο prover μετρά (hash) τη διαμόρφωση ή τη συμπεριφορά του σε πραγματικό χρόνο (PM, DM, EM, CF, SF) [23] και στέλνει αναφορά (attestation report) μαζί με έναν nonce και υπογραφή ή MAC για να να διασφαλιστεί η αυθεντικότητα. Ο επαληθευτής ελέγχει την εγκυρότητα συγκρίνοντας τον hash ή επαληθεύοντας την υπογραφή/MAC. Σε μεγάλες αποκεντρωμένες υλοποιήσεις, η επαλήθευση κατανέμεται είτε με Self πρωτόκολλα (Self-V, Self-L, Self-LC) είτε με External πρωτόκολλα (Neighbor, Jury). Υλικές RoT, όπως TPM, DICE, TrustZone, Intel SGX/TDX και RISC-V PMP, εξασφαλίζουν ότι το λογισμικό επαλήθευσης και τα μυστικά κλειδιά παραμένουν ασφαλή [2], [13], [14], [18], [25]. Τέλος, η Συλλογική RA (CRA) μειώνει το υπολογιστικό και δικτυακό φορτίο συγκεντρώνοντας επαληθεύσεις ομάδων συσκευών, ενώ τα μοντέλα απειλών περιλαμβάνουν λογισμικούς αντιπάλους,

κινητούς αντιπάλους, μη-παρεμβατικούς και παρεμβατικούς φυσικούς επιτιθέμενους, καθώς και αντιπάλους τύπου Dolev-Yao με πλήρη έλεγχο του δικτύου [9], [23].

1.3 Σχετική Βιβλιογραφία

1.3.1 PMP σε RTOS και Απομακρυσμένη Βεβαίωση

Στα πλαίσια της διπλωματικής του, ο Larmann [24] τροποποίησε τον πυρήνα του FreeRTOS ώστε να ρυθμίζει δυναμικά τις εγγραφές PMP σε κάθε εναλλαγή διεργασίας, απομονώνοντας τις και τον πυρήνα. Οι περιοχές μνήμης ορίστηκαν μέσω linker scripts και χαρτογραφήθηκαν με τη λειτουργία Top-of-Range ενώ υποστηρίχθηκε και η ασφαλής δυναμική κατανομή μνήμης μέσω PMP. Τέλος ενσωματώθηκε το σχήμα κρυπτογραφήσης PRINCE για την μνήμη, ελεγχόμενο από δεσμευμένα bits στις εγγραφές PMP.

Το LIRA-V [16] παρουσιάζει το PMP ως RoT για απομακρυσμένη επαλήθευση σε μικροελεγκτές RISC-V. Ο κώδικας ROM φιλοξενεί κλειδιά και τις κρυπτογραφικές ρουτίνες, ενώ η PMP δημιουργεί περιοχές μόνο-εκτέλεσης για την προστασία των κλειδιών υπογραφής. Μετά την επανεκκίνηση, η PMP ρυθμίζεται και κλειδώνεται. Κατόπιν αιτήματος, μετράται η μνήμη και υπογράφεται για την παραγωγή απόδειξης βεβαίωσης. Το προτεινόμενο πρωτόκολλο αμοιβαίας επαλήθευσης δημιουργεί στη συνέχεια ασφαλές κανάλι με χρήση ECDH και κρυπτογραφημένων αποδείξεων.

1.3.2 Ρίζες Εμπιστοσύνης από την TCG

Πέρα από την PMP, η Trusted Computing Group (TCG) έχει ορίσει εναλλακτικές ρίζες εμπιστοσύνης υλικού. Το Trusted Platform Module (TPM) [3] παρέχει ασφαλή δημιουργία, αποθήκευση και χρήση κρυπτογραφικών κλειδιών, διατηρώντας μετρήσεις firmware και λειτουργικού συστήματος σε Platform Configuration Registers. Συστήματα όπως το TRAP [2] χρησιμοποιούν TPM για ανίχνευση αλλοίωσης κώδικα και επαλήθευση κόμβων.

Η Device Identifier Composition Engine (DICE) είναι μια ελαφρύτερη ρίζα εμπιστοσύνης που παράγει εφήμερες κρυπτογραφικές ταυτότητες κατά την εκκίνηση συνδυάζοντας ένα μοναδικό μυστικό συσκευής με μετρήσεις firmware. Το MATCH-IN αξιοποιεί DICE για αμοιβαία βεβαίωση σε IoT δίκτυα, εκδίδοντας πιστοποιημένες ταυτότητες συσκευών και επιτρέποντας αμοιβαία αυθεντικοποιημένα TLS κανάλια που αποδεικνύουν την αξιοπιστία των συσκευών.

1.3.3 Εναλλακτικές Ρίζες Εμπιστοσύνης της Ακαδημαϊκής Κοινότητας

Πολλές δημοσιεύσεις προτείνουν ελαφρυές, προσαρμοσμένες ρίζες εμπιστοσύνης για μικροελεγκτές. Το GAROTA εισάγει ένα ελάχιστο, τυπικά επαληθευμένο υλικό που τρέχει παράλληλα με την CPU, διασφαλίζοντας την εκτέλεση κρίσιμων ενεργειών ακόμη και υπό παραβιασμένου λογισμικού. Το PRoM στοχεύει κακόβουλο λογισμικό που μετακινείται δυναμικά, μετρώντας τυχαία επιλεγμένα τμήματα μνήμης παράλληλα σε

πολλαπλούς πυρήνες, μειώνοντας τον διαθέσιμο χρόνο απόκρυψης και διατηρώντας υψηλή διαθεσιμότητα. Τέλος, το RATA αντιμετωπίζει το πρόβλημα TOCTOU καταγράφοντας με ασφάλεια τον χρόνο τελευταίας τροποποίησης της μνήμης προγράμματος, επιτρέποντας εκ των υστέρων ανίχνευση αλλαγών και μειώνοντας το υπολογιστικό κόστος όταν η μνήμη είναι γνωστό ότι είναι καθαρή.

1.3.4 Συλλογικά Πρωτόκολλα Απομακρυσμένης Επαλήθευσης

Η επεκτασιμότητα είναι κρίσιμη για μαζικά δίκτυα συσκευών. Το SEDA οργανώνει τις συσκευές σε επικαλυπτόμενο δένδρο ώστε να βεβαιώνουν αναδρομικά τους γείτονες και να συγκεντρώνουν αποτελέσματα, επιτρέποντας στον επαληθευτή να επιβεβαιώσει την ακεραιότητα ολόκληρου του σμήνους με μία υπογεγραμμένη αναφορά. Το HEALED επεκτείνει τη βεβαίωση προσθέτοντας αποκατάσταση: εντοπίζει αλλοιωμένα τμήματα μνήμης με χρήση Merkle Hash Tree, αντλεί σωστό κώδικα από αξιόπιστο κόμβο και επιδιορθώνει τη συσκευή, επαναφέροντάς την σε αξιόπιστη κατάσταση.

1.4 Το Πλαίσιο Απομακρυσμένης Επαλήθευσής μας

1.4.1 Κίνητρο και Απαιτήσεις

Κίνητρο

Πολλές υπάρχουσες λύσεις απομακρυσμένης επαλήθευσης απαιτούν εξειδικευμένο υλικό ή μη τυποποιημένα εξαρτήματα, περιορίζοντας την υιοθέτησή τους. Η εργασία μας στοχεύει στη δημιουργία ενός πρακτικού πλαισίου RA για πραγματικές, χαμηλού κόστους πλατφόρμες IoT, αξιοποιώντας το RISC-V PMP ως ρίζα εμπιστοσύνης (RoT), παρέχοντας μια βάση για μελλοντική έρευνα και υιοθέτηση.

Σύστημα, Μοντέλο Απειλών και Απαιτήσεις

Το πλαίσιο μας υποθέτει ανάπτυξη με έναν μόνο ιδιοκτήτη, ο οποίος εκτελεί μια φάση εκτός σύνδεσης, κατά την οποία οι συσκευές προγραμματίζονται με κώδικα, κλειδιά και υπογραφές. Οι συσκευές λειτουργούν τόσο ως αποδείκτες όσο και ως επαληθευτές, πραγματοποιώντας αμοιβαία αυτο-επαλήθευση ώστε να επιτυγχάνεται αποκεντρωμένη παρακολούθηση ακεραιότητας. Υιοθετούμε το μοντέλο κινητού αντιπάλου λογισμικού, υποθέτοντας ότι ο επιτιθέμενος μπορεί να έχει πλήρη έλεγχο των ευάλωτων εφαρμογών και συμμετέχει στο δίκτυο, αλλά δεν μπορεί να παρακάμψει τη RoT, το secure boot ή να σπάσει τα κρυπτογραφικά σχήματα. Η RoT είναι η απομόνωση μνήμης μέσω PMP, ενώ το secure boot και η ασφαλής αποθήκευση κλειδιών διασφαλίζουν ότι οι συσκευές ξεκινούν σε επιτρεπόμενη κατάσταση και προστατεύουν το κρυπτογραφικό υλικό, παρέχοντας ένα ισχυρό και επεκτάσιμο θεμέλιο εμπιστοσύνης. Χρησιμοποιούνται nonces για την αποτροπή επιθέσεων επανάληψης και τυχαίες περίοδοι επαλήθευσης για τη μείωση κινδύνων TOCTOU. Η επαλήθευση ακεραιότητας ακολουθεί την προσέγγιση Program Memory Attestation σε Harvard αρχιτεκτονικές, υπ-

ολογίζοντας hash της μνήμης προγράμματος για την ανίχνευση αλλοιώσεων, αντί για επαλήθευση της ροής εκτέλεσης (control-flow attestation) ή εναλλακτική μεθόδου.

Εκτός Πεδίου Εφαρμογής

Φυσικές επιθέσεις, κακόβουλος κώδικας που παραμένει μόνο στη μνήμη χωρίς μόνιμα ίχνη, επιθέσεις πλευρικού καναλιού και DDoS εξαιρούνται από το μοντέλο μας. Η επαλήθευση ροής εκτέλεσης δεν υλοποιείται λόγω πολυπλοκότητας και απαιτήσεων πόρων. Αν και το secure boot, η ασφαλής αποθήκευση και η απομόνωση μνήμης μέσω PMP δεν υλοποιούνται στο πλαίσιο αυτής της εργασίας, το πλαίσιο έχει σχεδιαστεί ώστε να μπορεί να τα ενσωματώσει σε μελλοντικές επεκτάσεις, διατηρώντας την έμφαση στη δημιουργία μιας επεκτάσιμης και κλιμακούμενης βάσης απομακρυσμένης επαλήθευσης.

1.4.2 Σχεδίαση Πρωτοκόλλου

Προκαταρκτικές Σκέψεις Σχεδιασμού

Μία αφελής προσέγγιση, όπου ο Αποδεικτής απλώς επιστρέφει την τιμή $Hash(Flash)$, είναι ευάλωτη σε επιθέσεις επανάληψης. Η εισαγωγή ενός τυχαίου N_{once} από τον Επαληθευτή μειώνει τον κίνδυνο καθώς επιβάλλει φρεσκάδα στην απόδειξη, αλλά η θέση του στο hash έχει σημασία. Οι τρεις υποψήφιες μορφές είναι $Hash(N_{once}||Flash)$, $Hash(Flash||N_{once})$, και $Hash(Hash(Flash), N_{once})$. Η πρώτη απαιτεί από τον Επαληθευτή να γνωρίζει ολόκληρο το περιεχόμενο του flash για την επαληθευση, κάτι που δεν κλιμακώνεται εκτός αν όλες οι συσκευές είναι ίδιες. Οι δύο τελευταίες απαιτούν μόνο την αποθήκευση του $Hash(Flash)$ ή της ενδιάμεσης κατάστασης του κατακερματισμού, υποθέτοντας πως χρησιμοποιείται κατασκευή Merkle–Damgård, επομένως είναι ευάλωτες σε επιθέσεις τύπου TOCTOU εάν ένας επιτιθέμενος μπορεί να επέμβει μετά από μερικό υπολογισμό. Σε όλες τις περιπτώσεις απαιτείται εγγύηση αυθεντικότητας επομένως, η οποία μπορεί να επιτευχθεί μέσω HMAC με κοινό μυστικό κλειδί που προκύπτει από ECDH ή RSA key exchange. Τα δημόσια κλειδιά πρέπει να είναι υπογεγραμμένα από τον κατασκευαστή, ενώ τόσο τα κλειδιά όσο και η ρουτίνα επαλήθευσης πρέπει να αποθηκεύονται σε ασφαλή μνήμη και να προστατεύονται από secure boot. Με αυτές τις εγγυήσεις, ο Αποδεικτής μπορεί τοπικά να συγκρίνει το hash του flash με το *Golden Hash* και να στείλει μόνο ένα λογικό αποτέλεσμα (επιτυχία/αποτυχία) χωρίς να εκθέτει η ασφάλεια της Απομακρυσμένης Επαλήθευσης.

Επισκόπηση Πρωτοκόλλου

Το προτεινόμενο πρωτόκολλο πραγματοποιεί Αποκεντρωμένη Αμοιβαία Απομακρυσμένη Επαλήθευση με Τοπικό Έλεγχο Ακεραιότητας flash και περιλαμβάνει:

- **Φάση Εκτός Σύνδεσης:** Προμήθεια συσκευών με firmware, ιδιωτικά/δημόσια κλειδιά, υπογραφές κατασκευαστή και ασφαλή αποθήκευση κλειδιών.
- **Στάδιο Εκκίνησης:** Το secure boot επαληθεύει το firmware, αποκρυπτογραφεί τα κλειδιά, υπολογίζει και αποθηκεύει το έγκυρο *Golden Hash* της flash, και προστατεύει περιοχές μνήμης και κλειδιών.

- **Φάση Εξερεύνησης:** Οι συσκευές ανακαλύπτουν γείτονες μέσω τριπλής χειραψίας (ανακοίνωση, αναγνώριση, επιβεβαίωση), επαληθεύουν υπογεγραμμένα κλειδιά και υπολογίζουν κοινό κλειδί HMAC μέσω ECDH+HKDF (Σχήμα 1.2).

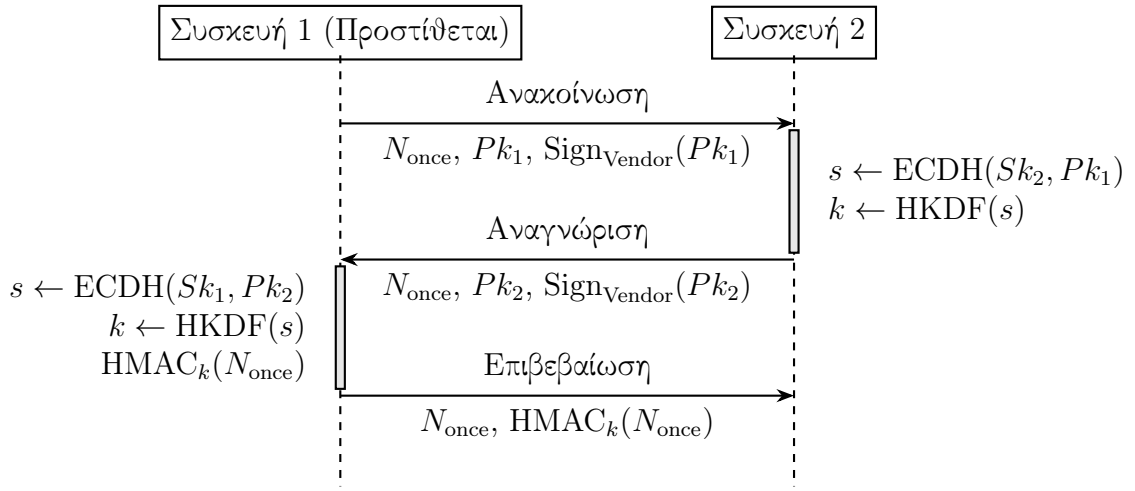

Figure 1.2: Διάγραμμα ακολουθίας της *Φάσης Εξερεύνησης* με 2 συσκευές

- **Γύροι Επαλήθευσης:** Κάθε συσκευή δρα ως Επαληθευτής στέλνοντας αιτήματα με φρέσκα nonces σε όλους τους γείτονες, ελέγχει τις απαντήσεις με HMAC εντός χρονικού ορίου και καταγράφει το αποτέλεσμα. Ως Αποδεικτής, η συσκευή εκτελεί αυτο-επαλήθευση συγκρίνοντας $Hash(Flash)$ με το *Golden Hash* και στέλνει υπογεγραμμένη ανταπόκριση με το αποτέλεσμα (Σχήμα 1.3).

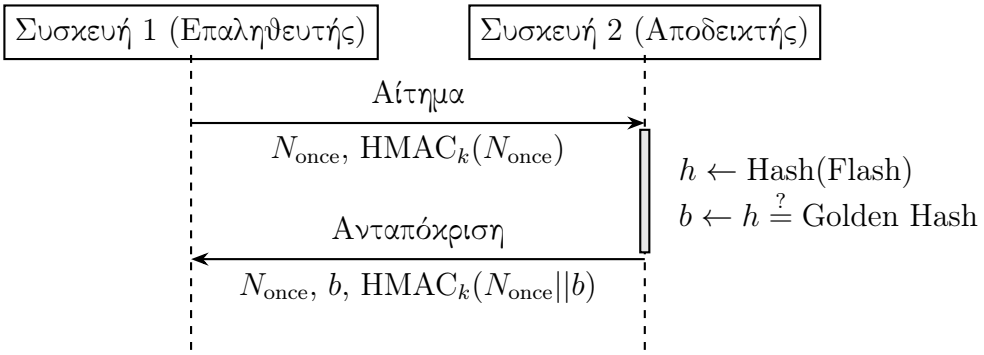

Figure 1.3: Διάγραμμα ακολουθίας ενός Γύρου Επαλήθευσης με 2 συσκευές

Όλες οι συσκευές αλληλο-επαληθεύονται με όσες βρίσκονται εντός εμβέλειας, σχηματίζοντας πλήρως συνδεδεμένο γράφο εμπιστοσύνης στην κοινή περίπτωση.

Figure 1.4: Τοπολογία δικτύου: Κόμβοι $1, 2, 3, 4, ..., N$ είναι όλοι συνδεδεμένοι

1.4.3 Υλοποίηση

Υλικό Πλατφόρμας

Για την ανάπτυξη του Πλαισίου Απομακρυσμένης Επαλήθευσης επιλέχθηκε το τσιπ ESP32-C3 της Espressif λόγω χαμηλού κόστους, μικρών διαστάσεων και πλούσιων

δυνατοτήτων.

Table 1.1: Specifications of ESP32-C3

Χαρακτηριστικό	Τιμή/Πληροφορίες
Αρχιτεκτονική	32-bit RISC-V Μονοπύρηνος
Pipeline	4-stage, in-order, scalar
Μέγιστη Συχνότητα	160 MHz
Wi-Fi Συνδεσιμότητα	2.4 GHz Wi-Fi (802.11b/g/n)
Bluetooth Συνδεσιμότητα	Bluetooth 5 (LE)
Εσωτερική ROM	384 KB
Εσωτερική SRAM	400 KB
Μνήμη RTC (Χαμηλής Ισχύος)	8 KB
Μέγιστη Υποστηριζόμενη Εξωτερική Μνήμη	Up to 16 MB
Πλήθος GPIO	16 or 22
Τάση Εισόδου	3.0 V to 3.6 V
Κρυπτογραφικοί Επιταχυντές	AES, SHA, RSA, HMAC, RNG

Στον πίνακα 1.1 παρατίθονται κάποια από τα χαρακτηριστικά του. Πρόκειται για έναν 32-bit single-core επεξεργαστή RISC-V με συχνότητα έως 160 MHz, 4 MB flash, 400 KB SRAM, Wi-Fi 2,4 GHz, Bluetooth 5 LE, καθώς και επιταχυντές υλικού για RSA, SHA, HMAC, AES και ασφαλές RNG. Υποστηρίζει secure boot, κρυπτογραφημένη αποθήκευση flash και PMP, ικανοποιώντας τις απαιτήσεις μας.

Figure 1.5: Πλακέτα ESP32-C3 Supermini

Στο Σχήμα 1.5 εμφανίζεται η πλακέτα ESP32-C3 Supermini που χρησιμοποιήθηκε στην ανάπτυξη. Αποτελεί μια πλέον οικονομική επιλογή, καθώς είναι διαθέσιμη σε τιμή μικρότερη των 2 ευρώ [30].

Λογισμικό

Το πλαίσιο υλοποιήθηκε σε C για την ευκολότερη ενσωμάτωση με τις υπόλοιπες βιβλιοθήκες που αξιοποιήθηκαν:

FreeRTOS Το FreeRTOS είναι ένας πυρήνας λειτουργικού συστήματος πραγματικού χρόνου (RTOS) ανοιχτού κώδικα από την AWS, σχεδιασμένος για ενσωματωμένα

συστήματα με περιορισμένους πόρους. Διαθέτοντας πολυδιεργασία (multitasking), επικοινωνία μεταξύ εργασιών και μικρό αποτύπωμα μνήμης, το FreeRTOS είναι εξαιρετικά φορητό και έχει υιοθετηθεί ευρέως στο IoT, στα συστήματα αυτοκινήτων και στον βιομηχανικό αυτοματισμό, όπου η αποδοτική και προβλέψιμη απόδοση είναι κρίσιμης σημασίας.

ESP-IDF Το Espressif IoT Development Framework (ESP-IDF) είναι ένα ολοκληρωμένο, ανοικτού κώδικα SDK για τους μικροελεγκτές της Espressif, δομημένο πάνω στο FreeRTOS. Ενσωματώνει πληθώρα βιβλιοθηκών, συμπεριλαμβανομένων πρωτοκόλλων δικτύωσης, επιπέδων αφαίρεσης υλικού. Η σχεδίαση του framework και το σύστημα μεταγλώττισης που βασίζεται στο CMake διευκολύνει την ανάπτυξη επεκτάσιμων και φορητών συστημάτων. Η απόφαση για την υιοθέτηση του ESP-IDF επηρεάστηκε σημαντικά από τον εγγενή ανοικτό του χαρακτήρα, ο οποίος επιτρέπει τη βαθιά παραμετροποίηση βασικών λειτουργιών—από τη διαχείριση μνήμης του FreeRTOS και την πιθανή υλοποίηση απομόνωσης εργασιών μέσω της Προστασίας Φυσικής Μνήμης (Physical Memory Protection - PMP), έως την τροποποίηση της διαδικασίας Ασφαλούς Εκκίνησης (Secure Boot) για την ενίσχυση της Αλυσίδας Εμπιστοσύνης (Chain of Trust) του συστήματος.

MbedTLS Η MbedTLS, μια ελαφριά, ανοιχτού κώδικα κρυπτογραφική βιβλιοθήκη που αναπτύχθηκε από την ARM. Παρέχει μια ολοκληρωμένη σουίτα κρυπτογραφικών πρωτογενών συναρτήσεων και πρωτοκόλλων, οι οποίες αξιοποιήθηκαν στην παρούσα εργασία για την υλοποίηση αυθεντικότητα και γνησιότητα στην επικοινωνία. Επιπλέον, οι κρυπτογραφικές της συναρτήσεις χρησιμοποιήθηκαν για την επαλήθευση της ακεραιότητας της μνήμης flash έναντι μη εξουσιοδοτημένης τροποποίησης και αλλοίωσης δεδομένων. Η ενσωμάτωση της στο ESP-IDF παρέχει εύκολη χρήση των επιταχυντών υλικού.

Δομή του Έργου

Το έργο ακολουθεί μια τυπική δομή ESP-IDF, επεκτεταμένη για να υποστηρίζει benchmarking, αρθρωτά στοιχεία και αυτοματισμό.

Πηγαίος Κώδικας

`components` Περιέχει αρθρωτό πηγαίο κώδικα: `my_attest` για τη βασική λογική attestation, `my_benchmarking` για τη μέτρηση απόδοσης, `my_crypto` για κρυπτογραφικές λειτουργίες, `my_flash_hash` για τη μέτρηση ακεραιότητας της flash, και `my_net` για τη δικτύωση.

`scripts` Μια συλλογή από scripts για δοκιμές, αποσφαλμάτωση (debugging) και αυτοματοποίηση.

Παραγόμενα Αρχεία

`benchmark` Περιέχει όλα τα αυτόματα παραγόμενα αποτελέσματα του benchmarking, συμπεριλαμβανομένων των δεδομένων CSV (`cpu_usage_time_series`, `results.csv`), των παραγόμενων `plots` και των debug logs.

`build` Περιέχει τα παραγόμενα από το ESP-IDF εκτελέσιμα (binaries) και αρχεία μεταγλώττισης (build files) για το flashing των συσκευών.

`keys` Αποθηκεύει τα παραγόμενα κρυπτογραφικά κλειδιά τόσο για τον `vendor` όσο και για τις μεμονωμένες `devices`, συμπεριλαμβανομένων των υπογραφών.

`logs` Περιέχει αρχεία καταγραφής (logs) από τη μεταγλώττιση και την παρακολούθηση της συσκευής.

Αρχεία Ρυθμίσεων

`CMakeLists.txt` Το κύριο script μεταγλώττισης για το σύστημα build του ESP-IDF.

`partitions.csv` & `.in` Ο πίνακας κατάτμησης της μνήμης flash της συσκευής και το πρότυπό του.

`sdkconfig` & `.defaults` Παραγόμενες και προκαθορισμένες από τον προγραμματιστή τιμές ρυθμίσεων για το framework και το ESP-IDF.

`.last_build_time` Ένα αρχείο χρονοσφραγίδας (timestamp) που χρησιμοποιείται για τη βελτιστοποίηση της επανατοποθέτησης (redeployment).

Αυτοματοποίηση Ανάπτυξης

Για την επιτάχυνση του κύκλου ανάπτυξης, δημιουργήθηκε μια σειρά από προγράμματα bash για την αυτοματοποίηση της μεταγλώττισης, του προγραμματισμού και της παρακολούθησης πολλαπλών συσκευών ταυτόχρονα. Χρησιμοποιώντας τα `Tmux`, `Openssl` και την εργαλειοθήκη `ESP-IDF`, αυτά τα scripts διαχειρίζονται τα πάντα, από τη δημιουργία κλειδιών έως τη μαζική ανάπτυξη και την ανάλυση αρχείων καταγραφής (logs).

Ανάθεση Κλειδιών Σενάρια για τη δημιουργία και διαχείριση κρυπτογραφικών κλειδιών για τον προμηθευτή και τις συσκευές.

- `generate_vendor_keys.sh`: Αυτοματοποιεί τη δημιουργία ζευγών κρυπτογραφικών κλειδιών του προμηθευτή (vendor).
- `generate_sign_device_keys.sh`: Δημιουργεί ένα ζεύγος κλειδιών για μια συσκευή και υπογράφει το δημόσιο κλειδί με το ιδιωτικό κλειδί του προμηθευτή.
- `generate_device_partition_binary.sh`: Δημιουργεί ένα δυαδικό partition που περιέχει τη διεύθυνση MAC της συσκευής, όλα τα κλειδιά και τις υπογραφές.
- `flash_attestation_partition.sh`: Δημιουργεί το τελικό αρχείο `partitions.csv`, υπολογίζοντας το μέγεθος και τη θέση του attestation partition.

Ανάπτυξη Σενάρια για τον προγραμματισμό του firmware και την παρακολούθηση πολλαπλών συσκευών.

- `find_esp32_dev.sh`: Σαρώνει τις συσκευές του συστήματος για να βρει και να εμφανίσει όλες τις συνδεδεμένες πλακέτες ESP32.
- `run_device.sh`: Προγραμματίζει μία μόνο συσκευή με τα απαιτούμενα κλειδιά και δυαδικά αρχεία, και στη συνέχεια ανοίγει έναν σειριακό monitor.

- `run_all.sh`: Κάνει build το project και ενορχηστρώνει τον προγραμματισμό και την παρακολούθηση όλων των συνδεδεμένων συσκευών ταυτόχρονα, χρησιμοποιώντας το `tmux`.

Μέτρηση Επιδόσεων Σενάρια για την αυτοματοποίηση δοκιμών απόδοσης και την οπτικοποίηση των αποτελεσμάτων.

- `benchmark.sh`: Εκτελεί αυτοματοποιημένες δοκιμές και συλλέγει μετρήσεις (CPU, RAM, κ.λπ.) σε ένα αρχείο CSV.
- `plot.py`: Διαβάζει αρχεία CSV από τα benchmarks και δημιουργεί γραφήματα με τα δεδομένα απόδοσης.

Διάφορα Βοηθητικά scripts για παραμετροποίηση και αποσφαλμάτωση.

- `update_sdkconfig_from_defaults.sh`: Ενημερώνει ένα υπάρχον αρχείο `sdkconfig` από ένα αρχείο προεπιλογών (defaults).
- `generate_partitions.sh`: Δημιουργεί το τελικό αρχείο `partitions.csv` για τη διάταξη της μνήμης flash της συσκευής.
- `find_errors.sh`: Σαρώνει αρχεία καταγραφής (logs) για να βρει και να εξάγει μηνύματα 'Guru Meditation Error'.
- `print_keys.sh`: Εμφανίζει τα κλειδιά από ένα δυαδικό αρχείο partition σε αναγνώσιμη από άνθρωπο δεκαεξαδική μορφή.

Αρχιτεκτονική Πλαισίου

Το πλαίσιο σχεδιάστηκε γύρω από tasks και queues του FreeRTOS για modularity και real-time απόκριση. Κατά την εκκίνηση φορτώνονται τα credentials της συσκευής, ξεκινά η δικτύωση και οι tasks για Απομακρυσμένη Επαλήθευση και εφαρμογή.

Τα κύρια Tasks όπως φαίνονται και στο Σχήμα 1.6.

- **Network Sending Task:** Αποστέλλει πακέτα (ESP-NOW) από τις διεργασίες της Απομακρυσμένης Επαλήθευσης ή της εφαρμογής στο δίκτυο, προσθέτοντας ανάλογο αναγνωριστικό..
- **Network Receiving Callback:** Κατευθύνει εισερχόμενα πακέτα στις σωστές ουρές ανά τύπο μηνύματος.
- **Attestation Sending Task:** Ξεκινά ανακοινώσεις και αιτήματα επαλήθευσης.
- **Attestation Receiving Task:** Αντιδρά σε αιτήματα επαλήθευσης και καταγράφει αποτελέσματα από ανταποκρίσεις.

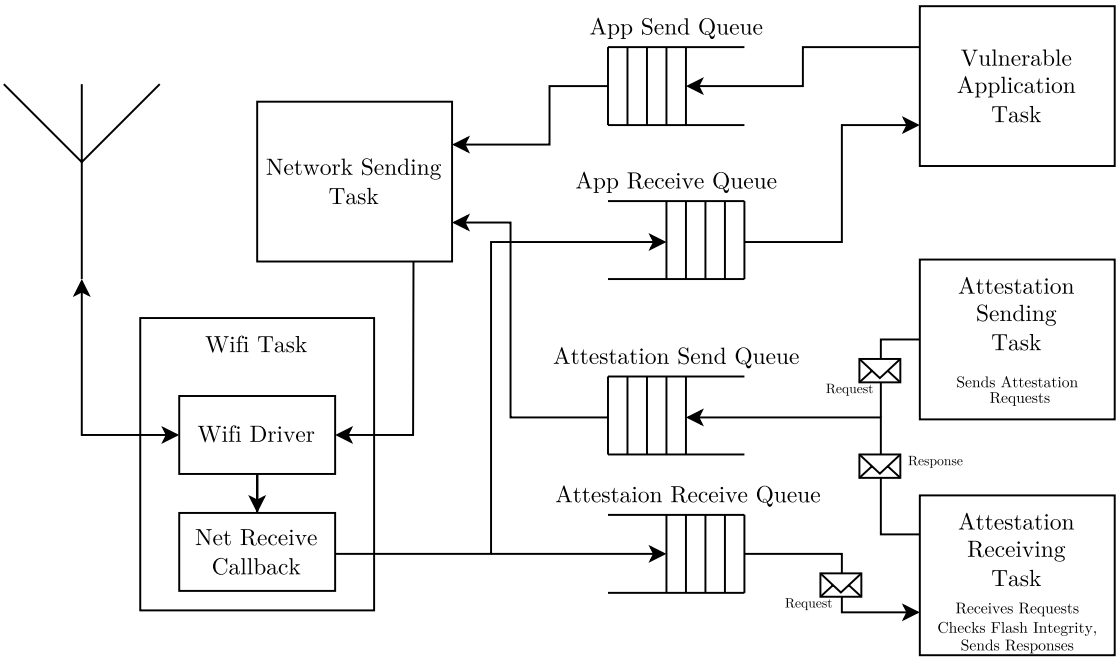

Figure 1.6: Διάγραμμα Tasks RTOS για το Πλαίσιο Απομακρυσμένης Επαλήθευσης

Επιλογές Ρύθμισης του Πλαισίου

Το πλαίσιο έχει παραμετρικοποιηθεί με σταθερές χρόνου μεταγλώττισης που ρυθμίζονται στο αρχείο `sdkconfig.defaults` ή μέσω του `menuconfig`. Οι επιλογές κατηγοριοποιημένες:

Ρύθμιση Συσκευών Ορίζεται ο αριθμός των συσκευών (`CONFIG_MY_NUM_DEVICES`), η παυση μετά την ανακάλυψη γειτόνων και timeout για επανασύνδεση γείτονα (`CONFIG_MY_NEIGHBOURS_FOUND_WAIT_MS`, `CONFIG_MY_TIME_FROM_LAST_ATT_RES_TO_RERECOC`

Ρύθμιση Δικτύωσης Καθορίζει το μέγεθος πακέτου (`CONFIG_MY_PACKET_DATA_MAX_LENGTH`), τη λογική επαναποστολής μηνυμάτων (`CONFIG_MY_OFFLINE_RESEND_MAX_COUNT`, `CONFIG_MY_RESEND_DELAY_MS`) και την περίοδο ανακοινώσεων (`CONFIG_MY_MSG_ANNOUNCE_SEND`

Περίοδο Επαλήθευσης Οι σταθερές `CONFIG_MY_ATTESTATION_INTERVAL_MIN_MS` και `CONFIG_MY_ATTESTATION_INTERVAL_MAX_MS` ορίζουν το τυχαίο χρονικό εύρος για την έναρξη νέων γύρων επαλήθευσης (attestation rounds).

Επιλογές Καταγραφής Μια σειρά από σημαίες (flags) (`CONFIG_MY_LOG_...`) για την ενεργοποίηση/απενεργοποίηση της αναλυτικής καταγραφής για συμβάντα δικτύου, μηνύματα, στατιστικά γειτόνων και γύρους πιστοποίησης, συν την περίοδο καταγραφής στατιστικών της CPU (`CONFIG_MY_LOG_CURRENT_STATS_PERIOD_MS`).

Επιλογές Αξιολόγησης Απόδοσης Η επιλογή `CONFIG_MY_BENCHMARKING` ενεργοποιεί τη συλλογή μετρήσεων απόδοσης για διάρκεια που ορίζεται από την `CONFIG_MY_BENCHMARK_RUNTIME_MS`.

Αναγνώριση Μηνυμάτων Οι `CONFIG_MY_ATTESTATION_MAGIC` και `CONFIG_MY_APP_MAGIC` είναι «μαγικοί αριθμοί» (magic numbers) που χρησιμοποιούνται για τη δρο-

μολόγηση εισερχόμενων πακέτων είτε στη μονάδα επαλήθευσης είτε στην εφαρμογή.

Επιλογές Κρυπτογραφίας Σημαίες όπως η `CONFIG_MY_PK_USE_EC_SECP192R1` επιλέγουν την ελλειπτική καμπύλη που χρησιμοποιείται για όλες τις λειτουργίες δημόσιου κλειδιού.

Προκλήσεις Υλοποίησης και Παραλείψεις

Κατά την ανάπτυξη αντιμετωπίστηκαν προβλήματα όπως μη προγραμματιζόμενες συσκευές, ασταθείς USB hubs, σφάλματα χρονισμού δικτύου, συγκρούσεις MAC διευθύνσεων, ασυμφωνίες κατακερματισμού flash και buffer overflows, τα οποία αντιμετωπίστηκαν με επανασχεδιασμό κώδικα και έλεγχο μνήμης. Ορισμένες λειτουργίες ασφάλειας του υλικού (secure boot, κρυπτογραφημένη αποθήκευση, απομόνωση tasks μέσω PMP) έχουν εφαρμοστεί ανεξάρτητα της εργασίας αυτής στο παρελθόν και δεν υλοποιήθηκαν στο πλαίσιο αυτής της διπλωματικής αλλά η ενσωμάτωσή τους αποτελούν μελλοντική εργασία.

1.5 Αξιολόγηση

Αυτή η ενότητα παρουσιάζει την αξιολόγηση του συστήματός μας σε διαφορετικές διαμορφώσεις και φορτία, εστιάζοντας στη χρήση CPU, στην κατανάλωση μνήμης, στους χρόνους απόκρισης και στους χρόνους ολοκλήρωσης κύκλων απομακρυσμένης επαλήθευσης. Τα δεδομένα συλλέχθηκαν χρησιμοποιώντας το αυτοματοποιημένο script `benchmark.sh` σε 10 ESP32-C3 Supermini Boards, με εκτελέσεις που διαφοροποιούν τόσο τον αριθμό των συσκευών (2–10) όσο και την περίοδο T των γύρων απομακρυσμένης επαλήθευσης (2–20s). T είναι η μέση τιμή της περιόδου καθώς για την αποφυγή επιθέσεων χρονισμού και συμφόρηση του δικτύου, η πραγματική περίοδος κάθε γύρου κάθε συσκευής επιλγόταν στο $T \pm 0.5s$. Κάθε εκτέλεση διήρκησε 100s, με καταγραφή της CPU κάθε 100ms και συλλογή μέσων στατιστικών ανά συσκευή. Οι χρονικές υπερβάσεις (timeouts) συμβαίνουν αν η απόκριση υπερβεί $T - 1$s, ενώ οι συγκεκριμένες ανταποκρίσεις δεν λήφθηκαν υπόψιν στους υπολογισμούς χρόνων απόκρισης.

1.5.1 Χρήση CPU με την πάροδο του χρόνου

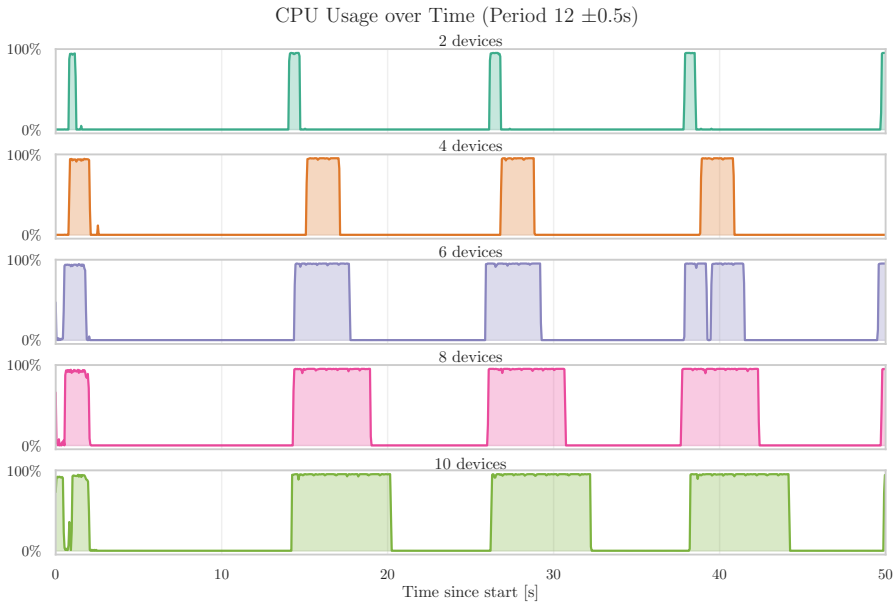

Figure 1.7: Χρήση CPU σε βάθος χρόνου με περίοδο 12s.

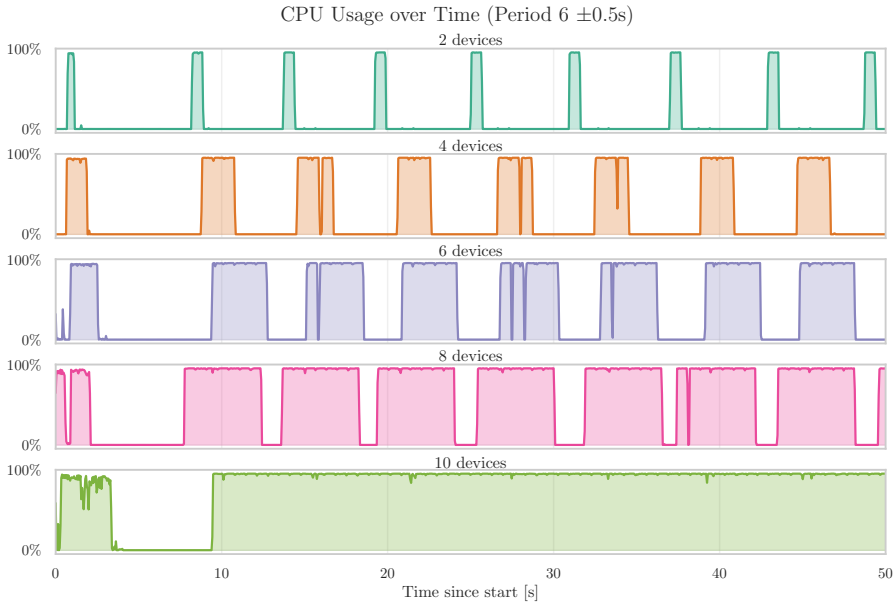

Figure 1.8: Χρήση CPU σε βάθος χρόνου με περίοδο 6s.

Όπως φαίνεται στα Σχήματα 1.7,1.8, η CPU παρουσιάζει περιοδικές κορυφές που ευθυγραμμίζονται με τους κύκλους RA. Η διάρκεια των κορυφών αυξάνεται με τον αριθμό των συσκευών, καθώς αυξάνεται ο αριθμός των αιτημάτων προς απάντηση και οι κατακερματισμοί της μνήμης. Κατά την διάρκεια του κατακερματισμού η χρήση είναι 100%.

1.5.2 Ανάλυση Κλιμάκωσης Αριθμού Συσκευών

Μετρήσεις συναρτήσει του αριθμού συσκευών που συμμετέχουν στο δίκτυο.

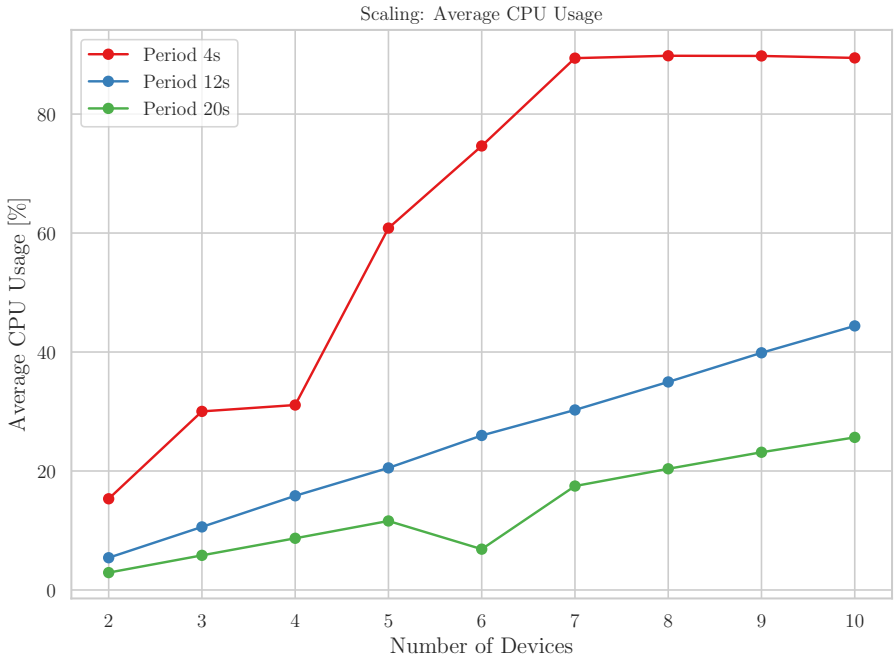

Figure 1.9: Μέση χρήση CPU vs αριθμός συσκευών.

Στο Σχήμα 1.9 γίνεται ξεκάθαρο πως η μέση χρήση του επεξεργαστή, από τις ρουτίνες της Απομακρυσμένης Επαλήθευσης, αυξάνεται σχεδόν γραμμικά και παρατηρείται κορεσμός σε περιπτώσεις πολλών συσκευών με υψηλή συχνότητα.

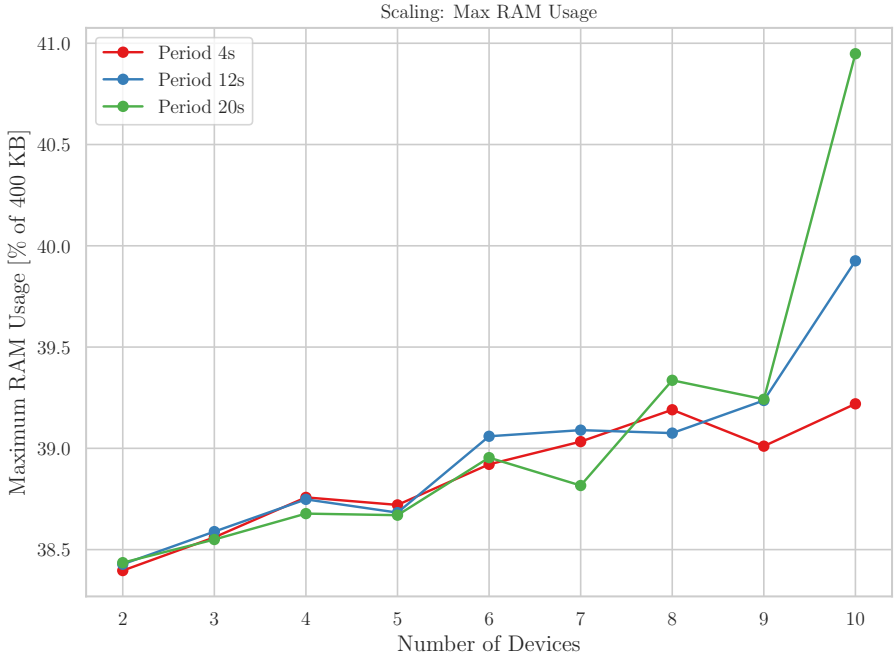

Figure 1.10: Μέγιστη χρήση RAM vs αριθμός συσκευών.

Όπως αναμένουμε η χρήση μνήμης (ολόκληρου του συστήματος) μένει περίπου σταθερή κατά την αύξηση του πληθυσμού, καθώς το μεγαλύτερο αποτύπωμα οφείλεται στις υπόλοιπες λειτουργίες του μικροϋπολογιστή.

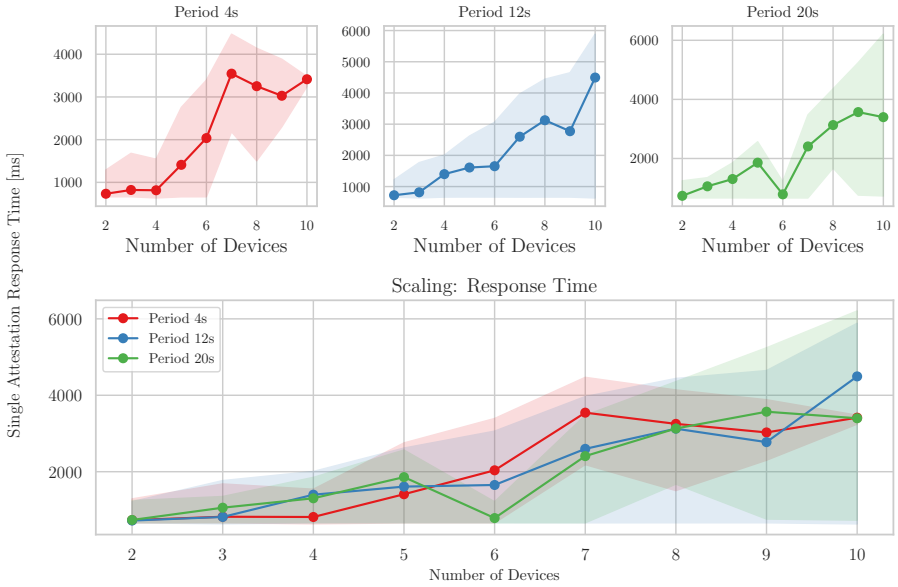

Figure 1.11: Χρόνος απόκρισης μίας αίτησης RA vs αριθμός συσκευών.

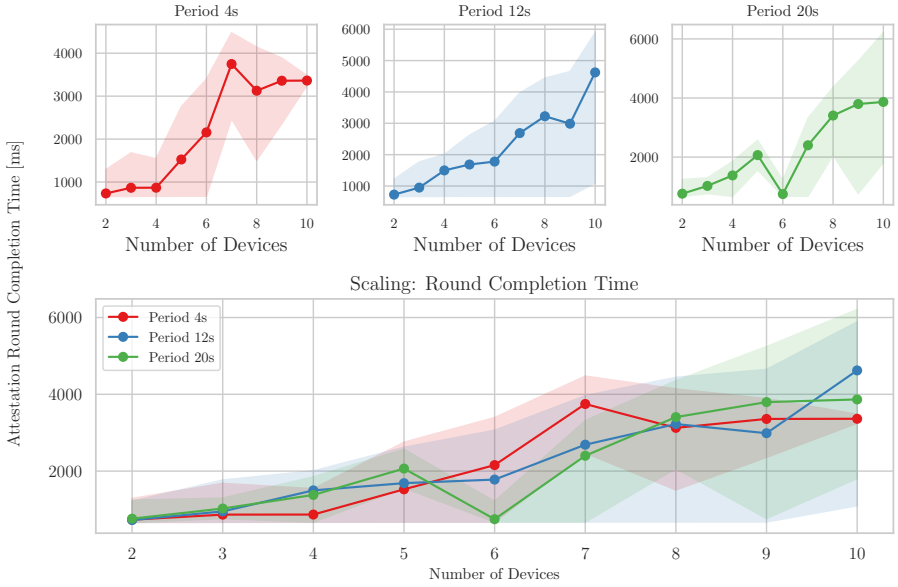

Figure 1.12: Χρόνος ολοκλήρωσης κύκλου RA vs αριθμός συσκευών.

Οι χρόνοι απόκρισης αιτημάτων και χρόνοι ολοκλήρωσης γύρου επαληθεύσεων από κάθε συσκευή αυξάνονται κατά την προσθήκη συσκευών στο δίκτυο όπως φαίνεται στα Σχήματα 1.11, 1.12. Αυτό οφείλεται στο ότι η διασπορά μεταξύ των περιόδων των αιτημάτων από όλες τις συσκευές είναι μικρή, με αποτέλεσμα τα αιτήματα κατά μέσο όρο να συγχρονίζονται και να προκαλούν συμφόρηση στις συσκευές.

1.5.3 Ανάλυση Συχνότητας Γύρων Επαλήθευσης

Μετρήσεις συναρτήσει της μέσης περιόδου μεταξύ διακριτών Γύρων Επαλήθευσης.

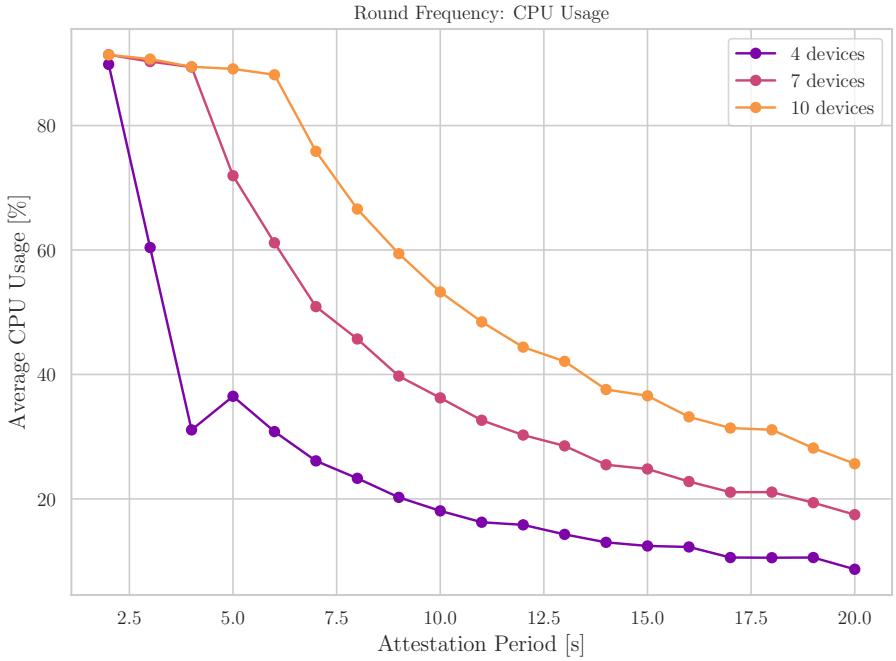

Figure 1.13: Μέση χρήση CPU vs περίοδος RA. Η CPU μειώνεται με αύξηση της περιόδου.

Το σχήμα 1.13 επιβεβαιώνει πως η μέση χρήση του επεξεργαστή για την επαλήθευση είναι αντιστρόφος ανάλογη της περιόδου, δηλαδή ανάλογη της συχνότητας των γύρων απομακρυσμένης επαλήθευσης που εκτελούνται.

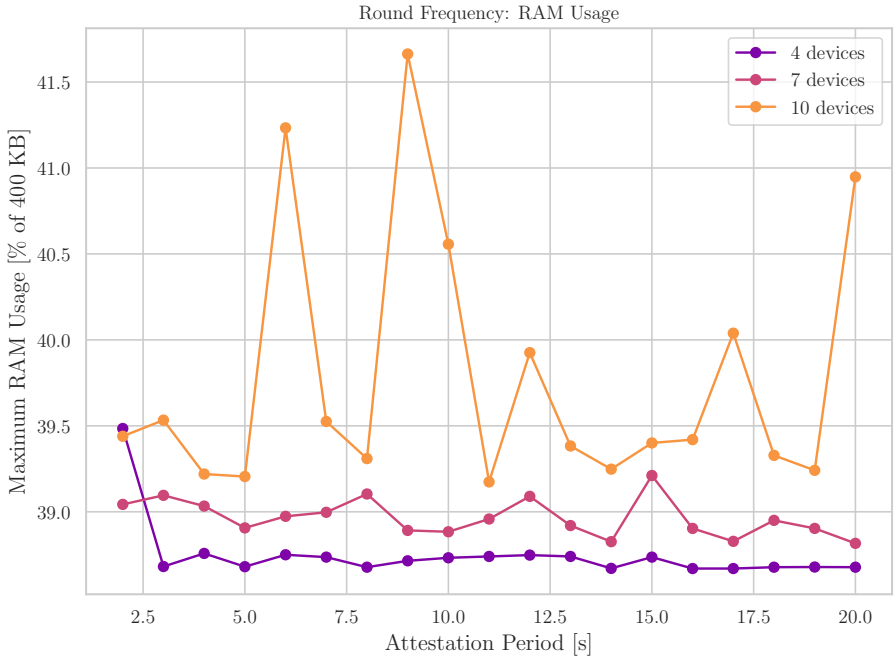

Figure 1.14: Μέγιστη χρήση RAM vs περίοδος RA. Το αποτύπωμα μνήμης παραμένει σχεδόν ανεπηρέαστο.

Από το σχήμα 1.14 επίσης επιβεβαιώνεται πως το αποτύπωμα μνήμης του πρωτοκόλου δεν επηρεάζεται από την περίοδο. Η αυξομειώσεις που παρατηρούνται οφείλονται στον τυχαίο ταυτοχρονισμό των διαφορετικών χρήσεων της σωρού, και σε σχέση με την συνολική κατανάλωση μνήμης είναι αμελητέα.

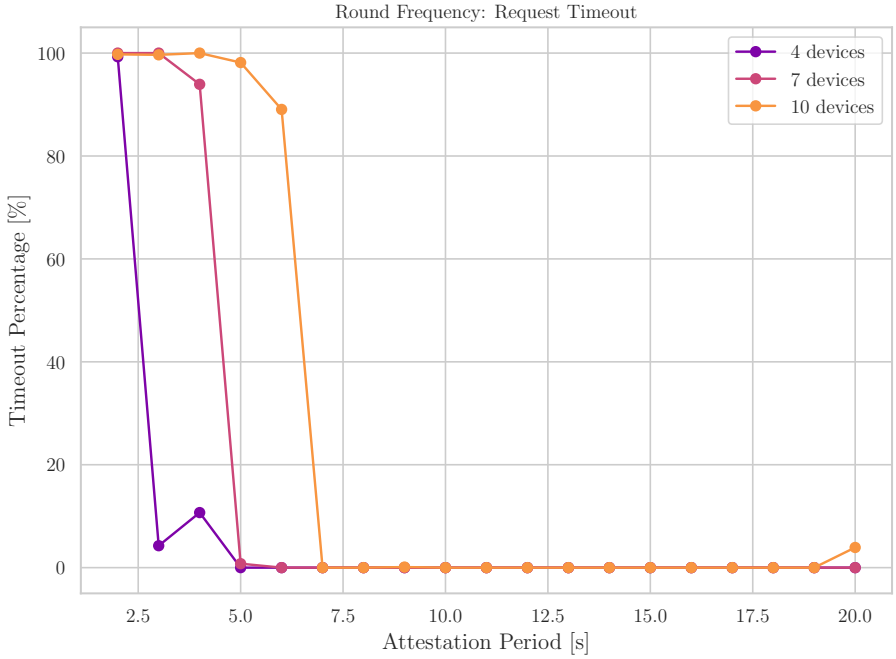

Figure 1.15: Ποσοστό χρονικών υπερβάσεων vs περίοδος RA. Μικρές περίοδοι αυξάνουν τις υπερβάσεις.

Το Σχήμα 1.15 φανερώνει αυτό που παρατηρήθηκε και στα προηγούμενα γραφήματα σε περιπτώσεις πολλών συσκευών και μικρής περιόδου. Ο συνοστισμός αιτημάτων σε αυτές τις καταστάσεις προκαλεί υπερβάσεις χρόνου (timeouts) με αποτέλεσμα το επαληθευτής να μην αποκτά γνώση της κατάστασης του αποδείκτη. Παρατηρούμε επιπροσθέτως πως για κάθε αριθμό συσκευών, υπάρχει μια οριακή τιμή της περιόδου, άνω της οποίας το σύστημα είναι σταθερό και λειτουργικό.

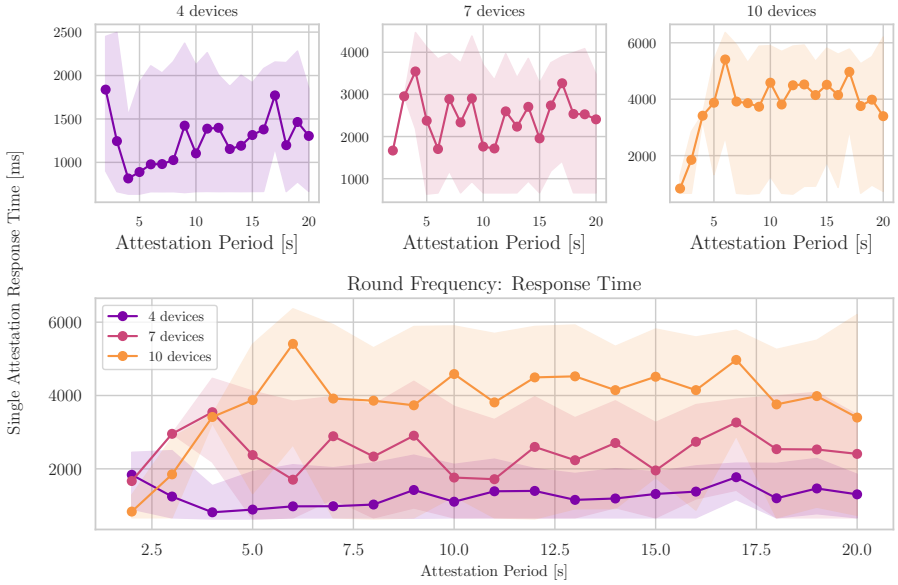

Figure 1.16: Χρόνος απόκρισης μίας αίτησης RA vs περίοδος. Σταθεροποιείται πάνω από κρίσιμη περίοδο.

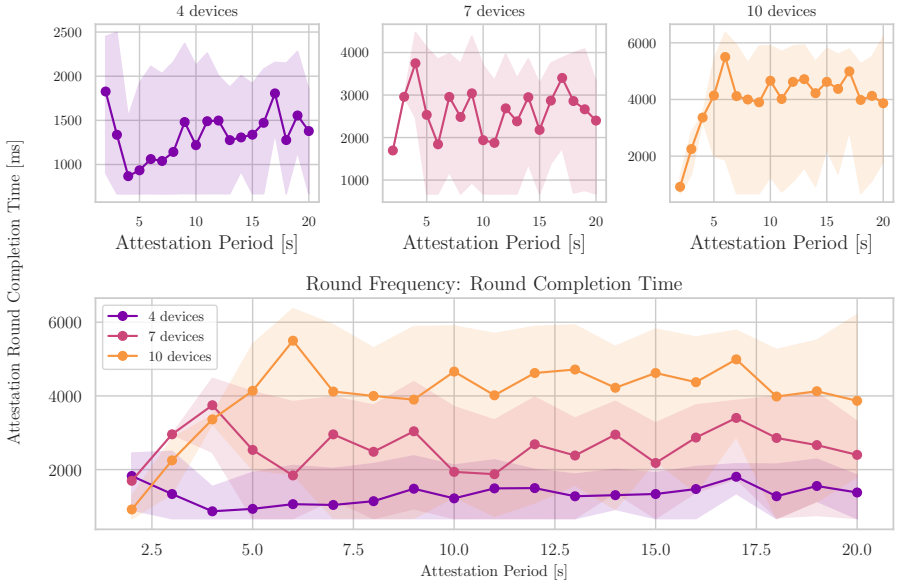

Figure 1.17: Χρόνος ολοκλήρωσης κύκλου RA vs περίοδος. Σταθεροποιείται όταν η περίοδος ξεπερνά το κρίσιμο όριο.

Στα Σχήματα 1.16, 1.17 παρατηρούμε πως η ανταπόκριση των συσκευών, τόσο

ατομικά όσο και συλλογικά, είναι ανεξάρτητη της μέσης περιόδου, δεδομένου πως αυτή ξεπερνά την οριακή τιμή για σταθερότητα του συστήματος.

1.6 Συμπεράσματα και Μελλοντική Εργασία

1.6.1 Συμπεράσματα

Η παρούσα εργασία παρουσίασε ένα απλό και επεκτάσιμο Πλαίσιο Απομακρυσμένης Επαλήθευσης για συσκευές IoT, αξιοποιώντας την Προστασία Φυσικής Μνήμης του RISC-V ως υλική Ρίζα Εμπιστοσύνης. Το πλαίσιο συνδυάζει μηχανισμούς για προστασία από επαναλήψεις, εγγυήσεις φρεσκάδας με nonces και αποκεντρωμένη RA peer-to-peer, επιτρέποντας συλλογικό έλεγχο ακεραιότητας χωρίς μοναδικό σημείο εμπιστοσύνης.

Η υλοποίηση στην πλατφόρμα ESP32-C3 απέδειξε αξιόπιστη λειτουργία σε δίκτυα διαφορετικών μεγεθών και συχνότητες επαλήθευσης. Η αξιολόγηση έδειξε ότι η χρήση CPU είναι ο κυριότερος παράγοντας απόδοσης, ενώ η κατανάλωση μνήμης παραμένει σταθερή. Ο σωστός καθορισμός των διαστημάτων επαλήθευσης εξασφαλίζει προβλέψιμους χρόνους ολοκλήρωσης και χαμηλά ποσοστά χρονικών εξαιρέσεων. Συνολικά, το πλαίσιο γεφυρώνει το χάσμα μεταξύ θεωρητικών συλλογικών σχεδίων RA και πρακτικής υλοποίησης σε συσκευές IoT περιορισμένων πόρων της βιομηχανίας.

1.6.2 Μελλοντική Εργασία

Δυνατές επεκτάσεις για βελτίωση του Πλαισίου Απομακρυσμένης Επαλήθευσής μας περιλαμβάνουν:

- **Ρίζα Εμπιστοσύνης στο Υλικό:** Ενσωμάτωση secure boot και ασφαλούς αποθήκευσης κλειδιών, σε συνδυασμό με PMP για διαχωρισμό εργασιών στο FreeRTOS στην υλοποίηση [24].
- **Εξυπνότερη Επαλήθευση:** Ενσωμάτωση περιοδικού ελέγχου ακεραιότητας αντί ελέγχου μετά από κάθε εισερχόμενο αίτημα επαλήθευσης.
- **Βελτιστοποίηση Απόδοσης:** Μείωση χρήσης μνήμης και CPU μέσω εναλλακτικών αλγορίθμων κατακερματισμού και πιθανοτικών μετρήσεων ακεραιότητας.
- **Κρυπτογραφία:** Ολοκλήρωση υποστήριξης RSA για δημόσια επαλήθευση, παράλληλα με την υπάρχουσα υλοποίηση με ECC.
- **Αυτοΐαση:** Δυνατότητα ασφαλούς και αυτοματοποιημένης αποκατάστασης μολυσμένων συσκευών εμπνευσμένη από το HEALED [8].
- **Κλιμάκωση:** Υποστήριξη σχηματισμού αποδοτικότερων τοπολογιών (π.χ. δενρικών) για Συλλογική Απομακρυσμένη Επαλήθευση όπως το SEDA [4].
- **Προστασία από DDoS:** Αξιοποίηση challenge–response και φίλτρων για αντίσταση σε επιθέσεις υπερφόρτωσης.

Αυτές οι βελτιώσεις θα ενισχύσουν περαιτέρω την ανθεκτικότητα, την αποδοτικότητα και τη δυνατότητα υιοθέτησης συλλογικών πλαισίων RA σε μεγάλης κλίμακας δίκτυα

IoT περιορισμένων πόρων.

Chapter 2

Introduction

2.1 IoT

Humans have always used technology as a tool, to improve and optimize our society. Internet of Things is no different case than that. Internet of things (IoT) is a collection of many interconnected objects, services, humans, and devices that can communicate, share data, and information to achieve a common goal in different areas and applications. These devices are in essence tiny computers, embedded in our society and lives.

2.1.1 Applications of IoT

The recent emergence of low cost interconnected chips, sensors has allowed many industries to adopt IoT and improve their operations. Common examples are [11]:

Healthcare IoT devices are used for remote health monitoring, such as with heart beat and respiration sensors, fall detection for the elderly, accelerometers, and pedometers. Devices have even been used inside the body, with pill-sized devices to examine the digestive system, and the acquired data is then transmitted to other devices for processing, filtering, and visualization.

Smart Grids and Metering IoT plays a crucial role in the transition to a green grid, where renewables make the majority of energy sources. All sources have to be able to monitor their potential output power, communicate it to the grid administrators which meter all consumer and industry needs concurrently. Admins dispatch quotas of power output distributed sources have to meet, and all that in a matter of milliseconds, in order to keep the grid stable, with minimal voltage and frequency deviations, or in the case of failure, to bring parts of the grid back up to speed quickly [20].

Home Automation Most modern homes are equipped with interconnected AC, fridges, heating, dehumidifiers, lighting, EV chargers and of course a variety of humidity, brightness, temperature, and air composition sensors. These can all be organized into smart systems, to provide a comfortable home e.g. with

ambient lighting, constant humidity and temperature, and smart charging during cheap electricity hours. Even more, intrusion or smoke detectors connected to the internet can notify absent home owners in critical moments.

Automotive Industry A plethora of IoT is used in the cars of the 21st century, from temperature, acceleration, wear, oil level sensors to critical safety systems like the ABS, tranction control, the airbags and to the more optional and comfort features like climate control, seat heating, infotainment, navigation [22].

Smart Cities Modern cities can be equipped with trash and recycling bins that have weight sensors and notify the waste management authorities which have the information available to optimize waste retrieval only where and when it's needed, reducing the necessary man-hours and fuel of their trucks. At the same time smart traffic lights that can sense the number of cars in each intersection and any incoming buses or trams can optimize their schedule to prioritize public vehicles and optimize car flow. This can improve or even solve congestion, and, together with bus/trolley/tram GPS tracking and monitoring through the internet [6], they can drastically increase the quality of the public transportation resulting in reduced travel times for all!

2.1.2 The Challenge of Security

n recent years, the proliferation of Internet of Things (IoT) devices has increased dramatically due to their widespread availability, low cost, and diverse applications. However, this rapid growth introduces significant challenges, particularly in the domain of security. The responsibility of ensuring the security and integrity of these devices becomes critical as their numbers continue to escalate.

Experience in the field of computer engineering has consistently demonstrated that any computational system inherently possesses vulnerabilities. When such systems are network-connected, these vulnerabilities become considerably more exposed and accessible to malicious actors. The security of IoT devices is of particular concern given their massive scale of deployment. A single exploited vulnerability or compromised entry point has the potential to affect millions of devices simultaneously, leading to widespread disruption.

Hijacked IoT devices can pose a big threat, dependent on the industry the are applied in. Compromised health monitors pose a huge privacy concern for all patients while the can result in life or death situations for much of the elderly clientele of this industry. Furthermore, in the automotive industry attacks can lead to deadly accidents and rogue cars, while the same goes for the rest of the transportation sector; Smart traffic lights are credible attack vectors for adversaries to sabotage one's roads [15] while cyberattacks on rail, public transportation systems and airports are far and wide in the age of IoT (e.g. [21], [26]). A compromised electric grid is also a major strategic liability for any country as it can be a very useful asset in the hands of a rival state [5]. Finally a network of compromised IoT devices can be used as a botnet for DDoS attacks or to infect an even wider range of devices.

It is for this reason crucial to not only have rigorous standards and security best practices in the development of such IoT Solutions, but also greater mitigation strategies. Such a mitigation technique could be the detection of intrusion in IoT during their operation.

Such a method, of ensuring security and trust in networks of interconnected IoT devices is Remote Attestation. It enables a single verifier device, or multiple verifiers, to be sure at a given time that their peers, are not compromised and working as meant. Scale this to tens or thousands of devices and Collective Remote Attestation brings challenges of it's own, for efficient communication, aggregation, and propagation of this trust.

This thesis contributes to the field by designing and implementing a Collective Remote Attestation framework, performing mutual heterogenous self-verifying attestation in IoT networks without a central Verifier. It is designed to be accompanied by hardware guarantees through secure boot, secure storage and RISC-V PMP. The implementation targeted the ESP32-C3 microcontroller, in order for the framework to be easily adaptable to industry and serve a basis for future research in the field of Remote Attestation.

Chapter 3

Background

To understand the security challenges at the heart of this work, we must first establish a technical foundation. This chapter introduces the core technological domains involved: the specialized nature of embedded systems, the interconnected world of the Internet of Things (IoT), the fundamental cybersecurity principles that protect them, and the critical process of remote attestation.

3.1 Embedded Systems

An embedded system is a specialized computer—a combination of a processor, memory, and I/O peripherals—designed for a dedicated function within a larger electronic or mechanical system. Unlike a general-purpose PC, an embedded system performs predefined tasks, often with strict real-time computing requirements. These systems are ubiquitous, powering everything from consumer electronics like digital watches to industrial machinery and automotive systems.

3.1.1 Networking

Networking is crucial for modern embedded systems, enabling them to communicate with other devices and central servers. The choice of technology hinges on the application's requirements for range, bandwidth, and power consumption.

Wired Protocols For on-board communication, protocols like I2C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface) connect microcontrollers to peripherals [1]. In robust industrial and automotive applications, CAN (Controller Area Network) bus is prevalent due to its noise resistance .

Wireless Protocols The rise of IoT has spurred wireless adoption. **Wi-Fi (IEEE 802.11)** is used for high-bandwidth applications where power is readily available. For low-power, short-range communication, several protocols compete: **Bluetooth** and its low-energy variant, **BLE**, are standard for point-to-point connections in wearables and personal gadgets. For creating robust mesh networks, particularly in smart home automation, protocols like **Zigbee**, **Z-Wave**, and the IP-based **Thread** are common choices. In long-range, low-

power (LPWAN) scenarios, cellular technologies like **NB-IoT** and **LTE-M**, alongside protocols like **LoRaWAN**, allow battery-powered devices to send small amounts of data over several kilometers [7].

3.1.2 Real Time Operating Systems

A Real Time Operating System (RTOS) is an operating system intended to serve real-time applications that process data and events as they come in, typically without buffering delays. The key characteristic of an RTOS is not high speed, but determinism: the ability to process and respond to events within a predictable and guaranteed time frame. This is critical in systems where a delayed response could lead to a system failure, such as in an automobile's braking system or a medical device.

An RTOS manages the system's resources (CPU time, memory) through a scheduler. Common scheduling algorithms include pre-emptive scheduling, where a higher-priority task can interrupt a lower-priority one, ensuring that critical tasks are always executed first. When multiple equal-priority tasks have to be scheduled, the scheduler can allow any task to run uninterrupted, until it yields, which can lead to starvation. Alternatively it can adopt a round-robin approach, where every task is given a time slot to process in

Popular examples of RTOS include FreeRTOS, Zephyr, and VxWorks.

3.1.3 Internet of Things

As mentioned before, the Internet of Things (IoT) refers to the vast network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these objects to connect and exchange data. The goal of IoT is to create a seamless fabric of interconnected objects that can be monitored, controlled, and optimized remotely, leading to improvements in efficiency, accuracy, and economic benefit.

An IoT architecture is typically composed of four main layers [20]:

Sensing Layer This consists of the "things" themselves—the physical devices with sensors and actuators that collect data from their environment (e.g., temperature, motion) or act upon it (e.g., turning on a light).

Network Layer This layer is responsible for transmitting the data collected by the sensing layer to a central processing platform. It includes gateways and the communication protocols (e.g., Wi-Fi, LoRaWAN, cellular).

Data Processing Layer This layer, often located in the cloud or on an edge server, is where the data is stored, processed, and analyzed.

Application Layer This is the user-facing layer, where the processed data is presented in a meaningful way, such as through a dashboard, and where users can control the IoT devices.

3.1.4 RISC-V

RISC-V [17] is an open standard instruction set architecture based on the principles of reduced instruction set computing (RISC). Unlike proprietary ISAs like x86 or ARM, RISC-V is free and open-source, allowing researchers, companies, and hobbyists to design and implement processors without licensing fees. Its modular design enables a small, simple base ISA that can be extended with optional features such as integer multiplication, floating-point operations, or vector processing. Its openness has spurred innovation, transparency, and a rapidly growing ecosystem of hardware, software, and development tools.

RISC-V has become especially popular in IoT because of its flexibility, scalability, and cost-effectiveness. Manufacturers can customize the ISA to include only the instructions they need, minimizing power consumption and silicon area—critical factors for low-cost, battery-powered devices. Its open-source nature removes licensing costs, making it attractive for startups and small companies developing IoT solutions. Additionally, the growing ecosystem of RISC-V development boards, compilers, and operating system support (like FreeRTOS and Zephyr) makes it easier for engineers to adopt. As a result, RISC-V is increasingly being used in microcontrollers, smart sensors, and edge devices where efficiency, affordability, and security are top priorities.

Physical Memory Protection

Physical Memory Protection (PMP) is a standard feature in RISC-V architecture designed for memory isolation in security-critical systems. In essence it consists a simple Memory Protection Unit (MPU) as it enables firmware to define specific physical memory regions and control the associated access permissions. PMP plays a crucial role in protecting memory for high-privilege binaries, such as firmware, and in trusted execution environments, where it isolates enclaves and manages shared memory regions. [19]

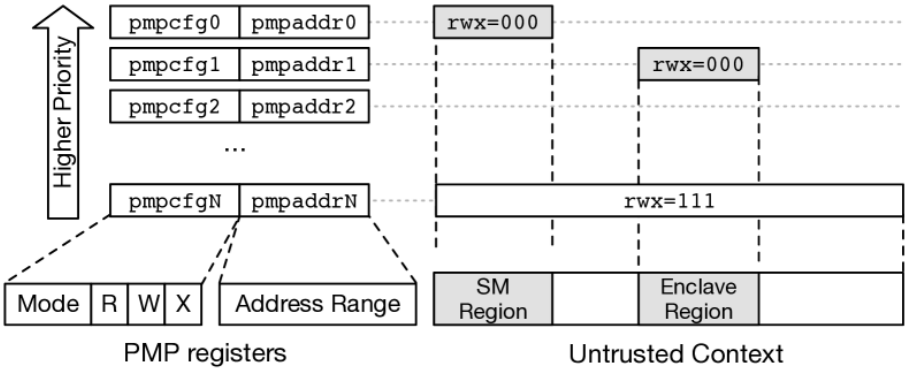

Figure 3.1: PMP Configuration Example [19]

PMP is configured using per-hardware-thread machine-mode control registers, with each core potentially having up to 16 PMP registers. Each PMP entry comprises a configuration register (pmpcfg) and an address register (pmpaddr). The

pmpcfg register specifies the addressing mode and permission bits (read, write, execute), while the pmpaddr defines the address range using one of three addressing modes: 4-byte aligned word (NA4), naturally-aligned power-of-two (NAPOT), or top-of-range (TOR). PMP entries operate as a whitelist, meaning memory is inaccessible by default unless explicitly defined by an entry. Entries are statically prioritized, where the lowest-numbered entry that matches a memory access (including the privilege mode) dictates whether the access succeeds or fails. The PMPChecker, a core hardware module, implements these rules, taking the memory address, access size, PMP register states (including lock bits and permissions), and the current privilege mode as inputs to determine the access permissions. Exceptions include denying permissions for partial matches within a region and granting default full access for high privilege modes or no access for low privilege modes when an address is not covered by any PMP region [19].

Analogous technologies in rival architectures are TDX/AMD SEV for x86 and TrustZone for Arm, which are richer in capabilities, able to provide TEEs but more costly. PMP provides a simpler solution that them, for systems without a Memory Management Unit (MMU).

3.2 Cryptography

Cryptography provides the essential tools to achieve confidentiality, integrity, and authenticity in communications and data storage.

3.2.1 Hashing

A cryptographic hash function is an algorithm that takes an input of arbitrary size (e.g., a file, a message) and produces a fixed-size string of characters, which is called a hash value or digest. A key property of these functions is that they are one-way; it is computationally infeasible to reverse the process and derive the original input from its hash. They are also designed to be collision-resistant, meaning it is extremely difficult to find two different inputs that produce the same hash value. Common hashing algorithms include SHA-256 (Secure Hash Algorithm 256-bit) and SHA-3.

SHA-256 first pads the input to a multiple of 512 bits by appending a '1' bit, followed by enough '0' bits, and then the 64-bit message length; it then processes the message in 512-bit blocks using the Merkle–Damgård construction, chaining a compression function to produce the final 256-bit hash. Each block is combined with the hash of the previous block, ensuring that the output depends on the entire message.

Hashing is widely used to verify data integrity; if the hash of a received file matches the hash computed by the sender, the receiver can be confident the file was not altered in transit.

3.2.2 Asymmetric Cryptography

Also known as public key cryptography, this system uses pairs of keys: a public key K_{pub}, which may be disseminated widely, and a private key K_{priv}, which is known only to the owner. The two keys are mathematically linked.

If the public key is used to encrypt data, only the corresponding private key can decrypt it. Formally, if M is a message:

$$C = E_{K_{\text{pub}}}(M)$$

where E is the encryption function and C is the ciphertext. Decryption is then performed using the private key:

$$M = D_{K_{\text{priv}}}(C)$$

where D is the decryption function. This process provides **confidentiality**.

Conversely, a message can be "signed" using the private key to create a digital signature:

$$S = \text{Sign}_{K_{\text{priv}}}(M)$$

Anyone with the public key can then verify this signature:

$$\text{Verify}_{K_{\text{pub}}}(M, S) \Rightarrow \text{True}$$

This confirms both the **authenticity of the sender** and the **integrity of the message**.

Prominent algorithms include RSA (Rivest-Shamir-Adleman) and Elliptic Curve Cryptography (ECC).

3.2.3 Message Authentication

Message authentication is the process of verifying that a received message comes from the alleged source and has not been altered. While digital signatures (using asymmetric cryptography) provide this, a more lightweight method is a Message Authentication Code (MAC). A MAC is generated using a symmetric key (a single key shared between the sender and receiver) and the message content. The sender computes the MAC and sends it along with the message. The receiver performs the same computation on the received message using the shared key and compares their result to the received MAC. If they match, the message is authenticated.

A common implementation is HMAC (Hash-based Message Authentication Code), which enhances the security of basic MACs by combining a secret key with a cryptographic hash function in a carefully structured way that resists common attacks. It works by hashing the message together with the key twice; once with an "inner"

key and once with an "outer" key, providing strong integrity guarantees even if the underlying hash function has some weaknesses. Formally, HMAC is defined as

$$\mathrm{HMAC}_K(M) = H\big((K \oplus \mathrm{opad}) \parallel H((K \oplus \mathrm{ipad}) \parallel M)\big)$$

where H is the cryptographic hash function, K is the secret key (padded to the block size), M is the message, and opad and ipad are fixed outer and inner padding constants. This double application of the hash function ensures that even if part of the message is modified, the computed HMAC will no longer match, providing message authentication.

3.3 Trust

In cybersecurity, trust is not an assumption but a property that must be established through verifiable mechanisms. It is the foundation upon which secure operations are built.

3.3.1 Root of Trust

The Root of Trust (RoT) is a component or set of components within a system that is inherently trusted and cannot be compromised. All other secure operations depend on it. An RoT is typically implemented in hardware to make it immutable. For example, a hardware module containing an unchangeable cryptographic key can serve as an RoT. All subsequent software and operations rely on this initial anchor of trust. Building upon the RoT, a Chain of Trust (CoT) is a sequential process that verifies the integrity of a system's components, one after another.

Secure Boot

A typical Chain of Trust for computers can be described by the Secure Boot process. It describes the method through which a single computer can boot securely and verifiably the correct software stack. The process starts with the RoT, which is some type of ROM with boot code, which is guaranteed the first piece of machine code that is run during boot, and an immutable public key of the computer's vendor. These two guarantees are set in hardware and thus trusted. The ROM code (e.g. 1st boot stage) verifies the integrity of the next component in the boot sequence, by taking its hash and verifying the vendor's signature of that hash. If the bootloader is valid, execution is handed over to it and it is itself trusted to verify the next component (e.g. the operating system kernel), and so on. This ensures that every piece of software loaded onto the system has been authenticated and is in a known, trusted state. If any link in the chain is broken (i.e., a component's integrity check fails), the boot process can be halted to prevent the execution of compromised code.

3.3.2 Remote Attestation

Remote Attestation (RA) is an effective solution to detect software compromises on remote devices. RA schemes allow a verifier to assess the integrity of the code and

configuration of a remote prover. In a typical RA protocol the prover, upon request delivers a proof of its configuration to the verifier who compares it against an a priori defined set of correct configurations [9]. It is essentially a method of building out the Chain of Trust across a network of devices, and ensure their integrity.

This section will elaborate on the mechanics of this process, including hash acquisition, delivery, authenticity, alternative verification models and the foundational Root of Trust in such systems.

Acquisition of Configuration Hash

Typically a proof takes the form of a hash, which summarises the configuration of the device. The acquisition of the hash (measurement) of a device's configuration is typically performed by the prover itself, often within a trusted environment. The target of attestation can vary, including program memory (PM), which generates a measurement for the entire region where program instructions reside; data memory (DM), aimed at protecting against compromise of runtime data; or entire memory (EM), which measures all memory, including data, program, and free space, to detect malicious code injection. Some advanced protocols also perform control-flow (CF) attestation to protect the runtime control-flow of instructions against tampering without altering binaries, or service-flow (SF) attestation to verify the flow of data communicated between devices, ensuring actions are not based on malicious services [23].

Delivery of Configuration Hash

Once acquired, the hash is delivered as an "attestation report" from the prover to the verifier through an interactive protocol. In a widely used method known as Type 1 Interactive Remote Attestation, the verifier sends a challenge (a nonce N) to the prover's RoT. The RoT responds with a proof, which is usually a hash of the prover's configuration (c), concatenated with the nonce N. This proof is then signed using public key cryptography or tagged with a Message Authentication Code (MAC) in symmetric cryptography.

Authenticity and Trust Verification

The authenticity of the delivered hash is paramount and is established through cryptographic means, relying on a RoT within the prover device. The verifier proves authenticity by recomputing and comparing the expected hash or by verifying the cryptographic signature/MAC. For symmetric cryptography, the verifier uses a shared key to recompute the MAC over the expected configuration and nonce and compares it with the received value. In public key cryptography, the verifier uses the prover's public key to verify the signature on the report. This process ensures that the measurement genuinely originated from the attested prover and has not been tampered with. CRA protocols assume that provers are equipped with a Root of Trust to protect against software attacks, often implemented through hybrid architectures combining minimal hardware (like Read-Only Memory (ROM) and Memory Protection Unit (MPU)) with software. The RoT ensures that the

attestation code itself cannot be altered and that secret keys are accessible only by genuine attestation code.

Decentralized Verification Models

When the verifier is not a central entity, CRA protocols adopt various decentralized approaches to distribute the verification task. In Self Protocols the measurement is validated by the prover device itself. This can be Self-Verifier (Self-V), where a verifier sends expected measurements to provers, or Self-Local (Self-L), where expected measurements are stored locally within the prover. Some protocols also employ Self-Local Consensus (Self-LC), where all provers share attestation results and converge to a collective decision, allowing any prover to be queried by a relying party. On the other hand, provers in External Protocols send their measurements to a third party for validation. This can involve Neighbor protocols, where measurements are validated by a prover's neighboring devices, or Jury protocols, where a group of provers acts as a jury.

Root of Trust in Remote Attestation

The Root of Trust (RoT) is a fundamental component for establishing and maintaining security in RA systems. In IoT devices, the RoT is typically realized by leveraging hardware providing minimal security capabilities, such as code and memory isolation. Examples include commercial solutions like TPM [2], DICE [25] ARM TrustZone, Intel SGX/TDX and of course the case of RISC-V PMP, but research platforms too like GAROTA [18], RATA [14], PRoM [13] and many more. This hardware-based RoT is assumed to be trusted and serves as the endpoint of the attestation protocol. It protects critical functions and secrets from software-based attacks.

Collective Remote Attestation

While traditional remote attestation works well for a single device, it presents significant scalability challenges in large-scale IoT environments, such as smart grids or swarms of drones, which may consist of thousands or millions of devices. Attesting each device individually would generate a massive amount of network traffic and computational overhead for the central verifier.

Collective Remote Attestation (CRA), also known as Swarm Attestation, refers to a class of protocols designed to efficiently verify the integrity of a group of devices simultaneously. The objective is to produce a single, aggregated attestation result for the entire group, or large subgroups, without requiring the verifier to interact with each device individually. This approach aims to achieve scalability and efficiency in more complex networks.

Various architectural approaches to CRA have been proposed and will be discussed in the next chapter.

Threat Models

Threat models are essential for systematically comparing and assessing the security guarantees of Collective Remote Attestation (CRA) protocols. The overall goal of adversaries is to compromise IoT network services to execute attacks against service integrity, network availability, and authentication while evading detection by CRA mechanisms. The literature on CRA [9][23]) identifies specific kinds of attackers based on their level of access and capability: Software Adversary, Mobile Software Adversary, Physical Non-Intrusive Adversary, Physical Intrusive Adversary, Stealthy Physical Intrusive Adversary and Dolve-Yao Adversary.

A **Software Adversary** ($\mathrm{Adv}_{\mathrm{SW}}$) can execute malicious code or firmware on a device and modify any memory in the untrusted environment, though they cannot restore it to the original, expected state. A stronger software threat is the **Mobile Software Adversary** ($\mathrm{Adv}_{\mathrm{MSW}}$), who shares the capabilities of $\mathrm{Adv}_{\mathrm{SW}}$ but can restore the device's memory to its original state, often attempting to eliminate any traces of its presence (e.g., erasing malware) to evade detection between successive attestation periods. Moving beyond software, Physical Adversaries are categorized into two types: the **Physical Non-Intrusive Adversary** ($\mathrm{Adv}_{\mathrm{PNI}}$), who operates in proximity to infer information, perhaps through side-channel attacks; and the intrusive variant, the **Physical Intrusive Adversary** ($\mathrm{Adv}_{\mathrm{PI}}$), who is capable of capturing a device, introducing external hardware, and reading or writing to any memory location, even in the trusted environment, by taking the device offline for a non-negligible duration. A related type is the **Stealthy Physical Intrusive Adversary** ($\mathrm{Adv}_{\mathrm{SPI}}$), who attempts to exfiltrate information after capturing the device without leaving traces behind [9]. Finally, the **Dolev-Yao Adversary** ($\mathrm{Adv}_{\mathrm{DY}}$) has full control over the communication network, enabling network attacks such as dropping reports, delay attacks, and recording and replaying healthy results [23].

Chapter 4

Related Work

4.1 Previous Uses of PMP in RTOS and Remote Attestation

PMP in RISC-V processors provides a lightweight mechanism for enforcing memory access control and isolation in resource-constrained systems. By enabling fine-grained permissions on memory regions, PMP enhances the security of RTOS and supports trusted computing functionalities, such as remote attestation, without requiring a full MMMU or dedicated security hardware. In RTOS environments, where tasks typically execute in a flat memory model, PMP can isolate tasks from one another and from the kernel, preventing unauthorized read, write, or execute operations. Similarly, in remote attestation, PMP can protect sensitive cryptographic keys and measurement routines, establishing a minimal hardware root of trust on constrained devices. The following subsections review representative approaches that exploit PMP in these contexts, highlighting its use for task isolation in FreeRTOS and for lightweight remote attestation on RISC-V microcontrollers.

4.1.1 Secure Task Management in FreeRTOS: A RISC-V Core Approach with Physical Memory Protection

The thesis by Christian Larmann [24] addressed the inherent lack of security in the flat memory model of FreeRTOS when running on RISC-V platforms, particularly against malicious third-party applications. The primary contribution was the enhancement of FreeRTOS security by dynamically managing memory access rights using the PMP feature of RISC-V. This mechanism serves to isolate user tasks from one another and from the FreeRTOS kernel itself, preventing unauthorized read, write, or execute operations between tasks and into critical kernel memory regions. This implementation leveraged the CV32E40S RISC-V core, chosen specifically for its PMP capabilities, and aimed to drastically improve security without resorting to a full Memory Management Unit, which is typically too resource-intensive for embedded systems.

Task isolation relied on fundamental modifications to the FreeRTOS kernel and

toolchain usage. To define protected memory spaces, the approach ensured that a task's executable code, data, and stack resided in contiguous memory regions through the use of linker scripts and compiler attributes. This memory encapsulation allowed the entire region to be defined and controlled using a minimal number of PMP entries, primarily leveraging the Top-of-Range addressing mode to handle arbitrary memory sizes. The crucial management step occurs during a task context switch: the kernel is modified to save the PMP configuration of the interrupted task and dynamically load the new PMP settings associated with the next scheduled task from its Task Control Block. This dynamic reconfiguration ensures the PMP hardware restricts memory access only to the memory and peripherals explicitly designated for the currently running task.

To further solidify the security architecture, the implementation included support for secure dynamic memory allocation and selective memory encryption. When a task allocates memory on the heap, the system dynamically assigns unused PMP regions to the newly allocated range, ensuring that this memory is also protected and exclusive to the task. Furthermore, the platform integrated an external memory encryption unit using the Prince encryption scheme. This hardware feature was cleverly combined with PMP control by utilizing reserved bits within the PMP configuration registers to selectively enable the encryption or decryption logic for external memory accesses pertaining only to certain tasks. The kernel was also protected from unauthorized privilege escalation attempts by malicious user tasks by implementing robust checks in the trap handler to verify that environment calls, used for kernel service requests, originated exclusively from the legitimate system call address space.

LIRA-V: Lightweight Remote Attestation for Constrained RISC-V Devices

LIRA-V [16] is a lightweight system designed for performing remote attestation between constrained devices that use the RISC-V architecture. The system is specifically tailored for environments such as microcontroller units (MCUs) which possess limited computing power, may lack a MMU, and may only operate using a single CPU protection mode (M mode). The overall goal is to assess the platform operating state of a proving device by a remote verifier, and to use this process to establish trusted communication between devices.

The design of LIRA-V relies on leveraging two specific hardware components already present in many RISC-V systems to build its trust anchor: ROM and the RISC-V PMP primitive. The ROM is utilized to host the Core RoT for Measurement and associated cryptographic algorithms, ensuring the integrity of the measurement and reporting procedures. The PMP primitive is used to create execute-only memory regions as a lightweight mechanism for protecting the secrecy of the quote signing key. This approach allows the system to operate without requiring dedicated security hardware, such as Trusted Platform Modules, or separate CPU privilege modes associated with fully-fledged Trusted Execution Environments.

The measurement procedure begins immediately after a device reset when the ROM code executes. This code performs the PMP configuration, locking a PMP

entry to assign X-only permissions to the region containing the signing key and its signing method. After an attestation request is received, the RoT measures the target memory region by computing an aggregated hash of the memory contents. This aggregated measurement is then signed using the ROM signing key to produce an attestation quote. LIRA-V further supports secure device-to-device communication through a mutual attestation protocol, which uses the quotes to bootstrap a secure channel. This protocol involves a three-message exchange that incorporates ECDH key exchange to derive an ephemeral session key, which is then used to encrypt the signed attestation quotes securely.

4.2 Alternative Roots of Trust by TCG

Except for the hardware guarantees that RISC-V PMP provides, the Trusted Computing Group (TCG) has defined 2 alternatives RoTs which some publications have adopted to achieve remote attestation.

4.2.1 Trusted Platform Module as RoT

A TPM (Trusted Platform Module) as defined by the Trusted Computing Group (TCG) [3] is a secure hardware chip that generates, stores, and uses cryptographic keys entirely inside the chip, protecting them from the OS or software attacks. It can encrypt, decrypt, sign, and verify data, and it includes a hardware random number generator for secure operations. During boot, the TPM measures the firmware, bootloader, and OS by hashing each component and storing the results in Platform Configuration Registers (PCRs). It can then seal data so it can only be decrypted if the system is in a trusted state and attest to other systems that the device is secure without exposing secrets. When a request is made—like decrypting a disk or signing data—the TPM first checks the PCRs to ensure the system hasn't been tampered with; only if the measurements match expected values does it release keys or perform cryptographic operations. In essence, the TPM acts as a secure vault and trust anchor, enforcing hardware-based security, verifying system integrity, and protecting sensitive operations.

TRAP [2], a hardware-based security solution, uses a Trusted Platform Module (TPM) on each sensor node in a Wireless Sensor Network (WSN) to detect unauthorized changes in the running application code. The verification is prepared during the node's bootloader stage as explained previously. During the application stage, a Challenger node initiates the process by sending an encrypted, random message as a challenge to the Attestor node. The Attestor's TPM retrieves the stored fingerprint and uses the challenge to generate a specific, signed response. This response includes the integrity fingerprint and other security checks, and is digitally signed by the Attestor's TPM using its private key. Finally, the Challenger verifies this signature and confirms that the fingerprint in the response matches the expected value for the application, ensuring that the remote node's program code has not been tampered with.

4.2.2 Device Identifier Composition Engine as RoT

A DICE (Device Identifier Composition Engine) by TCG [12] is a lightweight, hardware-rooted security module embedded in an SoC or microcontroller that establishes a chain of trust from the very first instruction at boot. Unlike a TPM, which is a general-purpose security chip providing persistent keys, sealed storage, and broad attestation services, DICE focuses on boot-time device identity and measurement-based, ephemeral key derivation. It contains an immutable Root of Trust, device-unique secrets stored in fuses or secure memory, cryptographic accelerators, and measurement/hash logic. During boot, the DICE engine generates a device-specific identity by combining the device secret with firmware measurements, deriving ephemeral keys for secure boot, encryption, and attestation. Each subsequent software stage is measured and verified, ensuring only trusted code executes, and the device can cryptographically prove its integrity to remote verifiers without exposing permanent secrets, providing a minimal, firmware-bound, hardware-backed security foundation.

A simple implementation of DICE for a RA is proposed in MATCH-IN (Mutual Attestation for Trusted Collaboration in Heterogeneous IoT Networks) [25]. It involves implicit mutual attestation in dynamic networks of IoT devices. The process begins with device bootstrapping, where the immutable Boot ROM and DICE Core enforce secure boot and utilize a Unique Device Secret (UDS) to generate layered cryptographic identities. Through measurement and derivation steps, the device creates Compound Device Identifiers (CDIs) and ultimately generates a Local Device Identity Key (LDevID), which represents the cryptographic identity of the running application. Next, an external application provider runs an "attestation provisioning service" to verify the trustworthiness of the device's hardware and firmware by checking the full attestation certificate chain against stored reference measurements; if validated, the service issues an externally certified LDevID certificate to the device. Finally, when two unknown IoT devices need to communicate, they establish a mutually authenticated TLS channel using their certified LDevID keys and certificates. The successful establishment of this channel provides implicit evidence of the trustworthiness of the application and the underlying device; if any component has been compromised, the LDevID key will not match the certified key, preventing the connection and seamlessly isolating the untrusted device.

4.3 Alternative Roots of Trust in Academia

A selection of alternative hardware RoTs for academia is presented which are lightweight although non standard in industry.

4.3.1 GAROTA: Generalized Active Root-Of-Trust Architecture

GAROTA [18] is the first clean-slate Generalized Active Root-Of-Trust Architecture designed for low-end micro-controller units, intended to guarantee that a desired action will be performed, even under full software compromise. Its mechanism relies

on a minimal, formally verified hardware component that runs in parallel with the CPU, monitoring signals to detect security violations. The core security features are "Guaranteed Triggering," ensuring the Trusted Computing Base (TCB) executes based on arbitrary hardware peripherals like timers or network events, and "Re-Triggering on Failure," which immediately resets the MCU if TCB execution is illegally interrupted, guaranteeing the RoT is the first component to run upon re-initialization. While primarily active, GAROTA enables security services such as enforcing memory integrity through timer-based triggering of TCB functions.

4.3.2 PRoM: Passive Remote Attestation against Roving Malware

PRoM [13], a Passive Remote Attestation technique designed for multi-core IoT devices, aims to detect roving malware while ensuring high device availability by relying on minimal secure hardware for secret key storage and True Random Number generation. The attestation mechanism is lightweight and avoids calculating a hash digest of the entire memory, instead using a randomized approach where the prover's memory is divided into blocks. PRoM exploits the multi-core architecture by scheduling a batch of blocks (equal to the number of cores μ) to be measured in parallel using a random permutation sequence Δ determined by secure seeds. This parallelism minimizes the adversary's evasion time. Crucially, the non-interruptibility requirement is relaxed, applying only when measuring a single block, thus enabling high availability for safety-critical applications.

4.3.3 RATA: Remote Attestation with TOCTOU Avoidance

Remote Attestation with TOCTOU Avoidance (RATA) [14] addresses the Time-Of-Check-Time-Of-Use problem, preventing transient malware from hiding memory modifications made between attestation measurements. RATA integrates a formally verified hardware module that provides historical context by monitoring program memory modifications and securely logging the event of the Latest Modification Time (LMT) into a protected memory region, which itself is covered by the attestation integrity check. RATA offers two techniques: $RATA_A$ (RTC-based), which logs a synchronized Real-Time Clock timestamp, and $RATA_B$ (Clockless), which logs the authenticated attestation challenge. This logging mechanism allows the Verifier to retrospectively detect unauthorized changes. Furthermore, if the attested region is known to be clean, RATA enables constant-time remote attestation by verifying only the small LMT region, drastically reducing runtime overhead.

4.4 Established Collective Remote Attestation Protocols

Till this point RoTs were discussed that ensure that Remote Attestation is guaranteed to complete correctly between a single prover and verifier. In this section two established protocols are discussed, which tackle the challenges of massive networks of devices that need to be attested and managed safely.

4.4.1 SEDA

SEDA (Scalable Embedded Device Attestation) [4] was the first attestation scheme for large-scale device swarms to efficiently verify their overall software integrity. The process is divided into an offline and online phase. In the offline phase the swarm operator (OP) initializes each device (D_i) with its software configuration (c_i), identity certificates, and signing keys; devices then execute a `join` protocol with neighbours to establish shared symmetric attestation keys (k_{ij}) and verify each other's code certificates. The on-line phase begins when a verifier (V) contacts an arbitrary initiator device (D_1). Using a global session identifier (q), the swarm constructs a spanning tree, and devices recursively execute the `attdev` protocol. Each device attests its children, accumulates the results, including the number of successfully attested devices (β) and the total devices (τ) in its subtree, and reports this accumulated outcome along with its own attestation status to its parent. The initiator (D_1) eventually receives the aggregated report for the entire swarm, digitally signs it, and sends the result to V. V authenticates the report using the OP's public key and accepts the attestation if the signature verifies and the accumulated counts (β and τ) confirm that all devices ($s-1$ devices besides D_1) were successfully attested.

4.4.2 HEALED

HEALED (HEaling & Attestation for Low-end Embedded Devices) [8] introduced a mechanism for not only detecting, also disinfecting software compromises on embedded devices. The foundation of HEALED's software measurement is a Merkle Hash Tree (MHT) construction, which allows the verifier to pinpoint the exact software blocks that have been modified. In the attestation protocol, a verifier device (D_v) periodically sends a request containing a random nonce (N_p) to a prover device (D_p). D_p measures its software state by generating the root (c'_p) of its MHT and computes a MAC over c'_p and N_p using a shared symmetric key, sending the MAC back to D_v for verification against the expected benign software configuration (c_p). If the MAC verification fails, D_v deduces that D_p is compromised and initiates the healing protocol. This requires identifying a benign healer device (D_h) that has an identical reference software configuration to the compromised device (D_c). Once D_h is identified and its trustworthiness is proven, it compares D_c's software configuration (c'_c) with its own, recursively requesting child hash nodes from D_c down the MHT until the specific compromised memory regions (leaf nodes) are identified. D_h then compiles a patch (L) containing the correct code segments for the modified regions, authenticates it with a MAC based on a shared key (k_{hc}), and sends it to D_c, which verifies and installs the patch to restore its software to a benign state.

Chapter 5

Our Framework

5.1 Motivation

It is no question that enabling mechanisms that can verify the integrity of our interconnected devices is crucial, in order to avoid bad actors in a world that is only getting more interconnected. This challenge is tall, and although there has been a lot of research, while studying the aforementioned publications and background theory, some worrying trends emerged. It is our opinion that these trends are serious limiting factors in the application of Remote Attestation in the IoT industry.

5.1.1 Need for Special Purpose Hardware

One of the key design decisions of a Remote Attestation protocol is where to put the Root of Trust (RoT). Software RoTs as discussed before rely inherently on weak assumptions but does however also provide limited security and subsequently Hardware RoTs are the preferred way. Still, in order to offer the security that is sought, many solutions rely on new specialized hardware modules ([25], [2], [18], [13]) or non-standard Memory Protection Units (MPU) ([14]). This in turn too limits adoption since the IoT industry relies heavily on chips that are already developed, readily available, and most importantly, low-cost.

5.1.2 Addressing RTOS Configuration

Proposed RA protocols do not address systems in which a Real Time Operating System is in use. Mono or multicore that run concurrent or parallel tasks have a much more complex runtime and memory structure that poses α new challenge in the field of Remote Attestation.

Conclusion

Thus we decided to create a framework for Remote Attestation, targeted at a platform with real adoption in the IoT, leveraging among others RISC-V PMP as our root of trust, in the context of a RTOS, similarly to most low-end IoT applications.

This would form a basis for future research and expansions of remote attestation protocols.

5.2 Requirements

5.2.1 System and Security Model

A clear definition of the system and attacker model is necessary to scope the guarantees provided by our remote attestation framework. This section outlines the operational assumptions of the system as well as the adversarial capabilities considered in our design.

System Model

We consider a deployment managed by a single Owner, who performs an offline setup phase in which devices are flashed with firmware, cryptographic keys, and signatures. Once deployed, devices form a network where each node acts as both Prover and verifier, engaging in mutual self-attestation. As a Prover, each device measures its own integrity on demand and produces a signed attestation report that is verified by its peers. As a verifier, each device can make attestation request to its peers and become aware of their state. This decentralized approach removes the need for a central verifier and allows scalable integrity monitoring across the network. We assume devices possess a hardware Root of Trust and that secure channels can be established after successful attestation.

Threat Model

The attacker is assumed to adhere to the **Mobile Software Adversary** Threat Model, and can have full control over vulnerable application code running on the device. In addition he is allowed network access, allowing arbitrary message injection, modification, delay, and replay. Physical tampering that bypasses the hardware root of trust, secure boot, or key storage is out of scope, as are attacks that break underlying cryptographic primitives. This model focuses on logical compromise and network-level threats, ensuring the framework primarily defends against software-level attacks with stronger hardware protections.

5.2.2 General Requirements

The remote attestation framework should target a microcontroller that is low-cost, widely available, and supported by a mature open ecosystem, ensuring flexibility, extensibility and ease of adoption by the industry. Critical hardware security features, including secure boot, secure storage, hardware random number generation, and cryptographic accelerators, are required to strengthen integrity of the Attestation Framework. Furthermore, the framework must work along side a RTOS and implement mechanisms to mitigate replay attacks, such as nonces, to prevent unauthorized message reuse. Furthermore, in order to combat Time of Check - Time of

Use (TOCTOU) attacks, randomness should be embedded in the Attestation Period. Collectively, these requirements ensure the framework is secure, practical, and accessible for both research and industrial applications.

5.2.3 Hardware Root of Trust

A fundamental component of any attestation protocol is the Root of Trust (RoT), which establishes the foundation for system security. To provide strong guarantees, the RoT must be implemented in hardware rather than software, as software-based solutions are inherently vulnerable to compromise. Hardware-based RoTs ensure that critical operations, such as cryptographic routines and key management, can be trusted even if other parts of the device are compromised. This hardware foundation is essential for the integrity and trustworthiness of the remote attestation framework.

We choose the RISC-V architecture for our framework due to its open-source nature and widespread interest in both academia and industry. RISC-V's open design allows for transparent verification and modification, which is crucial for security-critical applications. As part of our RoT, we leverage the Physical Memory Protection (PMP) mechanism in RISC-V to enforce strict access controls on memory regions. PMP ensures that the operations of the remote attestation framework remain isolated from potentially compromised components of the IoT device such as the main vulnerable application.

In addition to RISC-V and PMP, the hardware RoT must include components that support secure boot. This requires immutable storage elements, such as on-chip ROM and efuses, to hold the vendor's public key and track version revisions. These mechanisms guarantee that only authenticated and untampered firmware is executed during device startup, preventing unauthorized code from compromising the attestation process. Secure boot forms a critical anchor for the system's trust chain and ensures that the device starts from a known good state.

Finally, the hardware RoT must support secure storage for the encryption key of the device's flash storage. This key, accessible only by code either in ROM or that has been verified will used to encrypt other sensitive cryptographic keys and data.

All these characteristics together will form a strong foundation of trust.

5.2.4 Targeting Harvard Architectures

Remote Attestation Protocols are categorised by the way integrity is verified. Protocols that belong to the category of Control Flow Attestation, attest follow the program through its execution, either with checkpoints or graph based schemes. Control Flow Attestation protocols are few, since they typically can not be applied to complex applications without a huge overhead in computing resources, hardware, or power consumption.

Unlike the dynamic nature of Control Flow Attestation, Program Memory Attestation is static hardware security, which focuses on integrity measurement of the program memory. This implies checking if the program memory has been tampered

with or not, e.g. by measuring the hash of the memory and comparing it against a set of known allowed values. This is the more popular among publications and it is the approach chosen for the proposed Remote Attestation Framework: integrity measurement will assume a Harvard Architecture, where mutable data and code are not intertwined in flash.

5.2.5 Outside of our Scope

Although the proposed Remote Attestation Framework is designed to provide a robust foundation for verifying the integrity of IoT devices according to the aforementioned Threat Model. Classes of attacks and security mechanisms that are outside the capabilities of the Threat Model include physical attacks, such as directly flashing the device memory or tampering with hardware components. Attacks that manipulate data in memory without leaving a footprint on persistent storage, such as certain forms of in-memory malware or transient fault injections, are also excluded. Additionally, the framework does not address distributed denial-of-service (DDoS) attacks or side-channel attacks, including power analysis or electromagnetic monitoring, which require specialized hardware defences and mitigation strategies beyond the focus of this study. As such, Runtime and Control Flow attestation, which would involve monitoring program execution paths in real time to detect deviations are not considered.

In terms of hardware security features, mechanisms such as secure boot, secure storage, and RISC-V Physical Memory Protection (PMP) for memory isolation will not be implemented within the scope of this work. Nonetheless, the framework and underlying platform are explicitly designed to be compatible with these mechanisms. This ensures that, while they are not part of the current implementation, secure boot processes, tamper-resistant key storage, and memory isolation can be incorporated in future extensions. By explicitly delineating these boundaries, the framework maintains a clear focus on attestation operations while remaining extensible, providing a foundation that can accommodate more comprehensive security measures in subsequent work.

5.3 Preliminary Cryptographic Design Considerations

In the suggested Remote Attestation Protocol, it was concluded that the Integrity Measurement would consist of the hash of the device's Program Memory or Flash.

5.3.1 Basic Hash-Based Attestation

Suppose a naive Remote Attestation Protocol, in which upon request the Prover calculates the hash of his flash and transmits that. An attacker can, of course, intercept this message, and thus commit replay attacks, where he resends the same attestation response and convinces the verifier of the Provers integrity. In conclusion such a protocol is not secure and needs a replay attack mitigation mechanism.

5.3.2 Incorporating Nonces for Replay Attack Mitigation

Consider the case of one verifier and one prover device, without any authenticity requirements for now, but with the need for replay attack mitigation. To achieve this, the protocol has to be interactive, i.e. the verifier has to send a unique, uniformly randomly picked number -a N_{once}- to the prover, which has to include this N_{once} in the calculation of his proof of integrity, or in other words, his attestation response/report. The attestation routine on the Prover's side is responsible for conducting the integrity measurement, which is sent there after as proof to the Verifier.

With these requirements in mind, let us analyze three possible ways of incorporating the N_{once} in the Attestation Responses. Specifically the Proof could take the form of either:

- $Hash(N_{once}||Flash)$ or
- $Hash(Flash||N_{once})$ or
- $Hash(Hash(Flash), N_{once})$

Implications of Concatenation of Flash to Nonce

Consider the case when the proof has the form:

$$Hash(N_{once}||Flash)$$

The diagram of this hash calculation is visible in Figure 5.1.

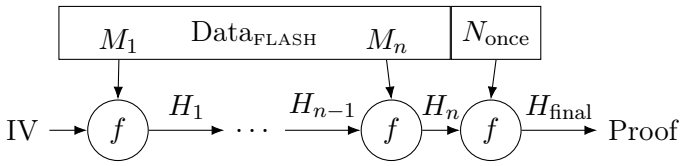

Figure 5.1: Hash Diagram of $Hash(N_{once}||Flash)$

Suppose an attacker is capable of running rogue code inside the Prover's device. This of course would cause a modification in the flash's contents and a correct Attestation Response would not be able to be constructed from the compromised device. In this case, the bad actor could only forge a correct Attestation Response if he had access to the entirety of the Flash's (or Program Memory's) contents. This is indeed very difficult, since, excluding the scenario of a supply chain attack, in order to extract all the flash's contents one would have to introduce code into the device, responsible for extracting and sending the flash's contents. However, this action itself would corrupt the flash's contents and thus they would not be recoverable.

Nonetheless, the Verifier would only be able to verify an Attestation Response if he himself had access to the entirety of the Prover's flash's contents, in order to be able to compute $Hash(N_{once}||Flash)$ for an arbitrary N_{once}. This limits us to networks where a central Verifier is present, capable of holding all possible flashes' contents of every Prover device, or network of identical devices with identical code. Hence this method is not desirable.

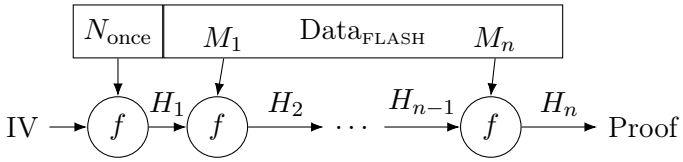

Figure 5.2: Hash Diagram of $Hash(Flash||N_{once})$

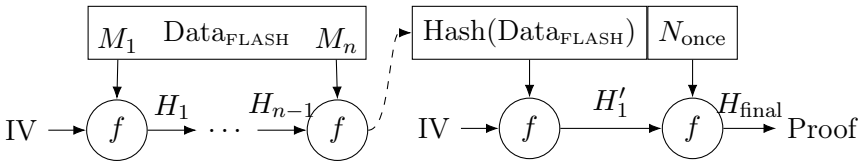

Figure 5.3: Hash Diagram of $Hash(Hash(Flash), N_{once})$

Implications of Concatenation of Nonce to Flash

Consider the cases when the proof has either one of the below forms:

$$Hash(Flash||N_{once})$$

or

$$Hash(Hash(Flash), N_{once})$$

In these 2 cases, the verifier is assumed to hold either the entire contents of the Flash, or the Hash(Flash) for the 1st case, or the or the internal state of the hash algorithm during the calculation of Hash(Flash || X) before including X (X is a placeholder for any value). This is necessary in order to be able to verify the proof received from a Prover, by combining the aforementioned with the fresh N_{once}. The calculation of the hashes and intermediate states are visible on the Figures 5.2 and 5.3.

It is however crucial to note that the attacker too with access to the hash of the flash or the internal state during the hashing, could forge a correct Proof. This could be achieved through compromising an application on the device at the right moment, after most of the flash has been hashed during a normal Attestation Response computation. This is a type of Time of Check - Time of Use (TOCTOU) attack.

5.3.3 Concluding Design Remarks

In conclusion, any use of N_{once} is not enough on its own. The verifier must be able to verify the authenticity of the Attestation Response as much as its freshness.

In order to verify authenticity, a Hash Message Authentication Code (HMAC) can be used. The creation and verification of the HMAC requires the two parties, Prover and Verifier, share a common secret key. This can be achieved by a public key cryptography, either through Elliptic Curve Diffie Hellman Key Exchange or through a RSA Key Transfer. As such, both devices need to be equiped with a private and public key.

In addition, the inclusion of the Vendor's signature of each devices public key ensures the verifiable authenticity of these public keys that are used for the key exchange.

Another requirement would be for both the share common secret and the device's private key be kept in sort of secure memory/storage. If it can be guaranteed that the Attestation Routine has exclusive access to the aforementioned keys, then any message accompanied by such an HMAC can be trusted to originate from the Attestation Routine.

The Attestation Routine itself must operated without any corruption, and thus it would have to be included in a Secure Boot chain, and it's program memory protected from any modifications, similarly to the keys in memory.

Assuming all the above, trust is based on authenticity the message and the correct operation of the Attestation Routine. Additionally, the Prover can himself store the expected hash or integrity measurement and conclude its state. It is thus redundant for the integrity measurement to be transmitted to the Verifier and to externally verify the Integrity Measurement, i.e. that the configuration hash belongs to an allowed configuration. It suffices for the Integrity Measurement to be compared to a correct measurement internally to the Prover. The final Attestation Response must of course contain the result of this comparison in the form of a boolean value that indicates success or failure of attestation.

5.4 Protocol Overview

Our Remote Attestation Protocol does mutual heterogenous attestation with local verification of flash integrity. The protocol consists the following phases.

5.4.1 Offline Phase

The offline phase is defined as the period of setting up the device before final deployment in a controlled environment. During this stage the Vendor assigns private and public key pairs to each device, signs the public keys. The firmware, with the application and attestation code, together with the encrypted keys are flashed on each device. The hash of the Vendor's public key is written in a One Time Programmable (OTP) memory, while the encryption key is stored in secure storage, that is only accessible by ROM boot code. Then the devices are deployed in the outside world.

5.4.2 Online Phase

The online phase is defined as the operation phase after deployment and consists of the *Boot Stage*, the *Exploration Phase* and the Attestation Rounds

Boot Stage

The device follows the Secure Boot process, ensuring the validity of the entire firmware. It then decrypts the device's keys from flash in memory and boots the RTOS kernel as well as the attestation task. The attestation task takes the hash of the flash, before any application code has executed. This is stored in memory as the *Golden Hash*. The *Golden Hash*, the device keys, the kernel and attestation code and their data memory are all protected against reads and writes from the vulnerable application's code through a Memory Protection Unit. Finally the application is invoked in order to enable the intended operation of the IoT device.

$$\text{Golden Hash} \leftarrow \text{Hash}(\text{Flash}_{t=0})$$

Exploration Phase

After the *Boot Stage*, and thereafter in predefined intervals, the device seeks new neighbours. Essentially a triple handshake occurs:

1. **Announcement:** The device announces its presence by a broadcast message. The announcement contains the sender's public key Pk and the vendor's signature
2. **Recognition:** A neighbour who has not registered the device as neighbouring and who receives the announcement verifies the authenticity of the public key and replies a unicast message, recognizing his presence. The recognition contains the sender's public key and vendor's signature. The neighbour can complete the ECDH Key Exchange and compute the shared secret through a Hash Key Derivation Function (HKDF) compute the HMAC key.
3. **Acknowledgment:** The device, upon receiving the recognition and verifying its public key's authenticity, computes in turn the shared HMAC key. Finally it replies with an Acknowledgment, which is accompanied by an HMAC.

All the above messages are accompanied by a nonce to identify the session. Recognition messages may have to be resent if the appropriate Acknowledgements are not received in time. After the mutual recognition, peers add each other to their respective neighbour lists for future attestation requests. The sequence diagram of Figure 5.4 shows the triple handshake.

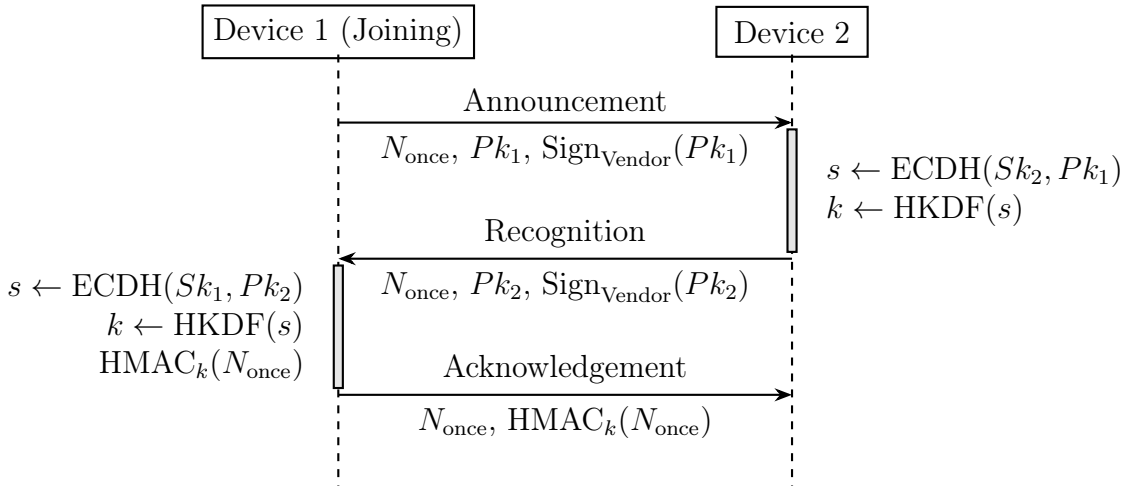

Figure 5.4: Sequence diagram of *Exploration Phase* with 2 devices

Attestation Rounds

The protocol is decentralised and conducts mutual attestation. As such every device acts both as a Prover and a Verifier.

As a Verifier, the device conducts "Attestation Rounds", who's frequency is random, in order to mitigate TOCTOU attacks. At the start of an Attestation Round the Verifier sends an Attestation Request to each of its neighbours, containing a fresh N_{once}. He expects Attestation Responses within a predefined timeout period. Once the response arrives, the HMAC and N_{once} are verified, and the result -successfull attestation or failed attestation or request timeout- is logged by the verifier.

As a Prover, the device listens to Attestation Requests. Upon receiving a request the Prover from a known neighbour conducts Self Attestation: It hashes the contents of its flash and compares the resulting measurement to the *Golden Hash*. An Attestation Response is sent back, indicating success if the flash has stayed intact and failure otherwise, including an HMAC of the message.

Figure 5.5 contains the sequence diagram of an Attestation Round with 2 devices.

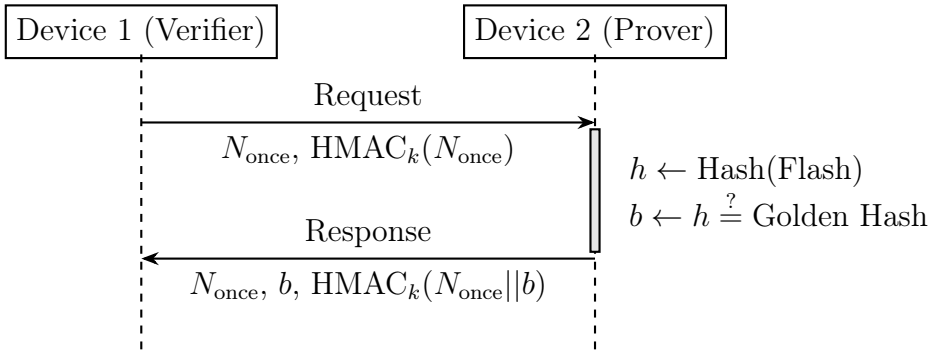

Figure 5.5: Sequence diagram of an Attestion Round with 2 devices

5.4.3 Network Topology

Since there is no central Verifier, every device in the network attests all its available neighbours within communication range and thus trust is established between them.

The network topology is arbitrary and in the simplest case of N devices, they are all within range of each other and neighbouring as shown in Figure 5.6. We evaluated with this common case topology, but the framework is also orthogonal to any other node topology.

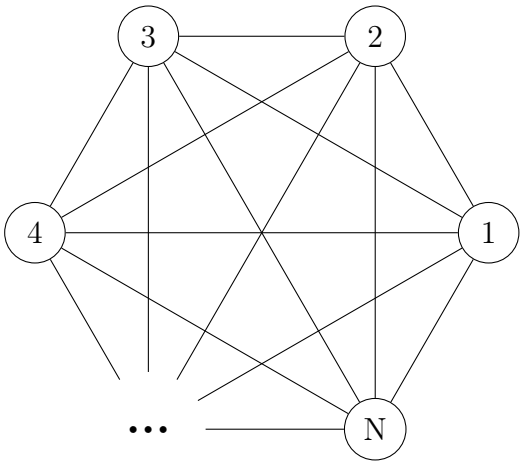

Figure 5.6: Network Topology: Nodes $1, 2, 3, 4, ..., N$ are all connected

5.5 Implementation

5.5.1 Hardware Platform

After careful consideration of the goals of this work, it was concluded that a fitting platform to develop the Remote Attestation Framework would be the ESP32-C3 chip by Espressif.

Table 5.1: Specifications of ESP32-C3

Specification	Value/Detail
Architecture	32-bit RISC-V Single-Core
Pipeline	4-stage, in-order, scalar
Max Speed	Up to 160 MHz
Wi-Fi Standard	2.4 GHz Wi-Fi (802.11b/g/n)
Bluetooth Standard	Bluetooth 5 (LE)
Internal ROM	384 KB
Internal SRAM	400 KB
RTC Memory	8 KB
Max External Flash Support	Up to 16 MB
GPIO Count	16 or 22
Recommended Input Voltage	3.0 V to 3.6 V
Cryptography Hardware	AES, SHA, RSA, HMAC,

Table 5.1 provides the major specifications of this chip. Most importantly in our context, the chip is conforming to the 32 bit RISC-V architecture, has a single pipelined core that can be clocked up to 160 MHz. It contains an antenna in it's tiny package and the hardware to support 2.4 GHz Wi-Fi (802.11b/g/n) and Bluetooth 5 (LE). It includes 4MB of flash, 400KB or SRAM, as well as a 4096 bit efuse memory, of which 1792 can be used by the developer. This device is suitable for ultra low power applications, as it consumes $<5\mu A$ in sleep mode. It can be flashed, monitored and powered by a USB Type C port, through which the device can communicate debug information through JTAG.

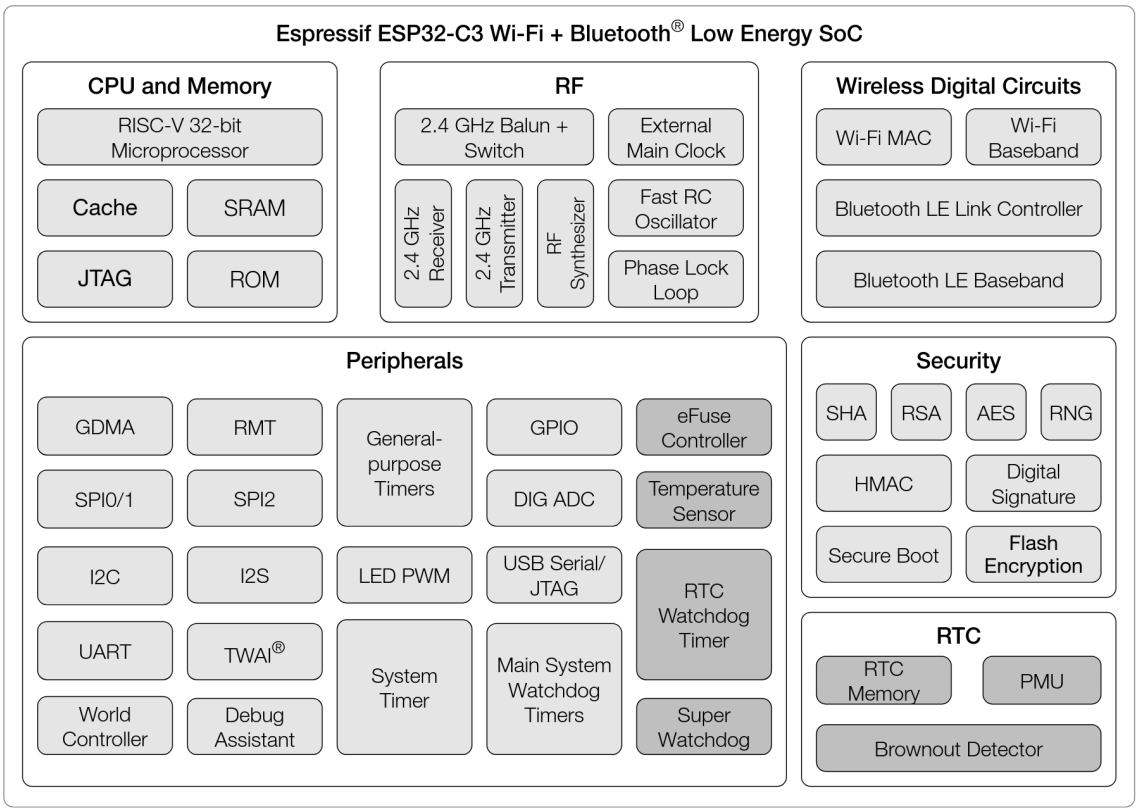

Figure 5.7: ESP32-C3 Functional Block Diagram [10]

Crucially for our application, except for its exceptional networking capabilities, it is also equipped with cryptographic accelerators for RSA, SHA256, HMAC, AES and a secure RNG, as well as 16 regions for the RISC-V Physical Memory Protection Extension. Finally It has support for secure storage (encrypted flash) and secure boot [10].

Figure 5.8: ESP32-C3 Supermini Board

All these features add up in a board that is very low cost, readily available at less than $2 per piece [30] in a small package, like the development board used

for evaluation, visible in Figure 5.8. This makes it a very attractive option for any IoT developer and a prime subject for this work, whose goal was to build the foundations of a solution that can be adopted in the real world, not just in the laboratory. In conclusion, a total of 10 chips were acquired for the development, testing and evaluation.

5.5.2 Software Stack

All the code for this project was written in the C language since it gives the most control to the developer and integrates the best with the rest of the software stack. Specifically:

FreeRTOS

FreeRTOS is an open-source, real-time operating system (RTOS) kernel designed for resource-constrained embedded systems and microcontrollers. Developed and maintained by Amazon Web Services (AWS), FreeRTOS provides deterministic task scheduling using a preemptive, priority-based scheduler, ensuring that high-priority tasks meet strict timing requirements. It supports multitasking, inter-task communication, synchronization primitives (queues, semaphores, and mutexes), and memory management, thereby enabling efficient utilization of limited hardware resources. FreeRTOS is highly portable and modular, supporting a wide range of processor architectures and toolchains, which makes it suitable for diverse embedded applications. Due to its small footprint, real-time performance, and reliability, FreeRTOS has become a widely adopted kernel in industrial automation, consumer electronics, automotive systems, and IoT applications, where predictable task execution and efficient resource management are critical.

Espressif IoT Development Framework

The Espressif IoT Development Framework (ESP-IDF) is the official open-source Software Development Kit (SDK) for Espressif's series of microcontrollers, including the ESP32, ESP32-S, and ESP32-C families. ESP-IDF is built upon FreeRTOS, providing a real-time operating system environment that supports multitasking, inter-process communication, and deterministic execution essential for time-critical embedded applications. The framework incorporates a wide range of software components such as network protocol stacks (TCP/IP, HTTP, MQTT, Bluetooth Classic, and BLE), hardware abstraction layers (HAL), peripheral drivers, and security libraries (TLS/SSL, cryptography, secure boot, and flash encryption). Its modular architecture facilitates scalable system design, enabling the integration of custom components and third-party libraries while maintaining code portability across hardware variants. Furthermore, ESP-IDF utilizes a CMake-based build system and Python-based project configuration tools, ensuring cross-platform development support. With comprehensive debugging capabilities, continuous integration support, and adherence to industry standards, ESP-IDF is widely employed in both academic research and industrial IoT solutions, where reliable wireless communication, low-power operation, and secure edge processing are critical.

The openness of the framework with which applications for the ESP32-C3 chip played a big part in our decision, as it allows us to not only develop using established libraries, but to customize everything, from the memory allocation of FreeRTOS, with the ability of possibly adding task isolation through PMP, to the Secure Boot mechanism that can progress our Chain of Trust.

MbedTLS

MbedTLS is a lightweight, open-source cryptographic library developed by ARM, specifically designed for embedded systems with constrained computational resources. It provides a comprehensive set of cryptographic primitives and protocols, including symmetric encryption algorithms, asymmetric encryption and digital signature schemes, hashing algorithms, message authentication codes, and key derivation functions. Its modular and portable architecture allows it to be easily integrated into microcontroller-based platforms, and Espressif has ported MbedTLS to the ESP32-C3 and related SoCs as part of the ESP-IDF framework, providing seamless access to hardware-accelerated cryptographic functions when available.

In this work, MbedTLS was employed to implement custom secure communication protocols, ensuring integrity, and authenticity of transmitted data. Additionally, its cryptographic primitives were used to verify the integrity of flash storage, enabling detection of unauthorized modifications and data corruption. The library's lightweight design, compliance with established cryptographic standards, and ease of integration make it particularly suitable for embedded IoT applications, where both security and computational efficiency are critical.

Networking

For the physical layer of our Remote Attestation Protocol WiFi. In fact, for reasons of development speed and power efficiency the open-source protocol ESP-NOW developed by Espressif in the ESP-IDF was used [29]. This consists simply of custom vendor WiFi frames. It is important to note that any other physical layer could be used in place of ESP-NOW like Bluetooth, Zigbee or other custom WiFi frames, and in fact our framework allows for this change.

5.5.3 Project Structure

The Attestation Framework project, follows the standard ESP-IDF structure but extends it to support benchmarking, key management, and modular attestation components, custom configuration options and deployment automation. The following is a hierarchical overview of the folder and file organization:

Source Code

`components` Part of the ESP-IDF project structure, it contains different software modules that interact with each other, allows for compartmentalizing code.

`my_attest` Contains all the core logic of the Attestation Framework. It makes use of all its sibling components.

`my_benchmarking` Contains device-side code for benchmarking, i.e. collecting and formatting CPU and RAM usage and various other statistics about the performance of the Remote Attestation Protocol.

`my_crypto` Contains functions for signing/verifying digital signatures, Elliptic Curve Diffie Hellman Key Exchange, Hash Message Authentication Code etc., utilizing the MbedTLS API of the ESP32-C3.

`my_flash_hash` Abstraction layer for flash operations, such as integrity measurement of device i.e. taking the hash of the entire flash and retrieval of vendor assigned keys.

`my_net` Here in the tasks and interface for networking is defined, that is used by the (vulnerable) application as well as the attestation protocol to communicate with peers.

`scripts` Contains a variety of scripts for testing, debugging, showcasing and benchmarking. We will expand in a following section.

Generated Artifacts

`benchmark` This folder holds data about benchmarks, that were used for the Evaluation chapter of this thesis. Automatically generated by our scripts.

`cpu_usage_time_series` Contains CSV files that hold data about the CPU usage of the attestation related tasks over time, in different configurations and devices.

`fail_logs` This is mainly used for debugging as it contains logs of benchmark runs that did not produce results successfully.

`plots` Contains all plots drawn from the benchmark data.

`errors.txt` Contains any logged errors of benchmark runs.

`results.csv` All benchmark results (CPU usage, RAM usage, request/response delays, timeouts).

`build` Contains all build files, binaries of ESP-IDF that will be used to flash the chips. This is automatically generated by the ESP-IDF toolchain.

`keys` Contains all generated keys:

`device` Public, private keys of each device, as well as the vendor's signatures proving the validity and authenticity of the devices' public keys.

`vendor` Public and private keys of the vendor (issuer of the devices).

`logs` Contains all logs, including the output of the compilation toolchain of ESP-IDF and the output log of every device that is flashed and monitored.

Configuration Files

`.last_build_time` Used for redeployment optimization and speedup.

`CMakeLists.txt` Instructs CMake build system integrated into ESP-IDF.

`partitions.csv` Generated partition table for device, after appropriate allocation for device cryptographic keys.

`partitions.csv.in` Template for the above partition table.

`sdkconfig` Generated configuration setting values, managed by ESP-IDF.

`sdkconfig.defaults` Holds the ESP-IDF configuration setting values that are of interest to the Attestation Framework, as well as the values to all our custom compile-time settings, managed by the developer.

This modular structure allows for maintainability, clear separation of concerns, and automated benchmarking.

5.5.4 Deployment Automation and Scripts

During the framework's development a lot of challenges arose, and it became apparent that a quick development cycle had to be adopted. This is why in order to flash and monitor multiple devices concurrently multiple USB hubs were used to connect all the devices to the deploying computer.

A series of bash scripts were used in order to automate the build cycle, including compilation, flashing of the program binaries to the devices, and creation and population of a custom partition on the devices' memories with the unique MAC address, cryptographic keys and the vendor's signature assigned to each device. In addition Tmux was used in these scripts in order to be able to monitor all logs and outputs of all 10 devices. Finally Openssl was used in order to create the public/private key pairs that would be assigned to each device and sign them using one vendor key.

Following are the scripts developed for the suggested Attestation Framework grouped by operation, in greater detail. Note some scripts depend on others and the framework user does not have to interact with all of them.

Key Assignment

Script to assist with key assignment to the devices.

`generate_vendor_keys.sh` This script automates the generation of cryptographic vendor key pairs based on the selected option in `sdkconfig.defaults`, ensuring consistent key management for the project. It safely handles existing keys by renaming old ones with timestamps, generates new private and public keys in DER format, and cleans up outdated device keys.

`generate_sign_device_keys.sh` Generates a cryptographic key pair for a device based on the algorithm specified in `sdkconfig.defaults` and outputs them in DER format. It then signs the device's public key with the vendor's private key, producing a signature file that can be used to verify the device's authenticity.

`generate_device_partition_binary.sh` Creates a binary package containing a device's assigned MAC address, private key, public key, its signature, and the

vendor's public key, all serialized with length-prefixed fields in little-endian format. The resulting binary file can be directly flashed or used by firmware to provision secure device identity and authentication data.

`flash_attestation_partition.sh.sh` Generates a final `partitions.csv` file contianing the partition table that will be flashed on the device, by calculating the attestation partition's offset and size according to `sdkconfig.defaults` and filling them into a template CSV. It ensures the partition size is aligned to 4 KB and correctly positioned at the end of the flash memory

Deployment

Script to assist with deployment and monitoring.

`find_esp32_dev.sh` Scans all `/dev/ttyUSB*` and `/dev/ttyACM*` devices, walks their sysfs paths to find USB vendor and product IDs, and checks them against known ESP32 chipsets. If a match is found, it prints the device path (e.g., `/dev/ttyUSB0`) for easy identification of connected ESP32 boards.

`run_device.sh` Prepares and flashes an ESP32 device with attestation keys and partition binaries, generating missing device or vendor keys if necessary. After flashing, it opens the monitor on the specified `PORT` and allows the user to interact with the device.

`run_all.sh` Orchestrates building the project, regenerating partitions if needed, and launching one process per connected ESP32 device using either `tmux` windows or silent background processes. It verifies the expected number of devices, logs output for each device, and provides a control pane or signal handler to terminate all running instances at once.

Benchmarking

Script to assist with benchmarking of the framework.

`benchmark.sh` This Bash script automates running benchmarking tests across multiple device counts and period configurations, collecting CPU, RAM, timeout, and response metrics. It manages log collection, retries on failures, updates results in a CSV file, and gracefully handles cleanup on errors or interruptions.

`plot.py` This Python script reads benchmark result CSVs and generates plots for CPU usage, RAM usage, timeout rates, response times, and round completion times across different device counts and periods. It automatically creates organized output folders and saves time series, scaling, and round frequency visualizations as PNG files.

Miscellaneous

Other scripts.

`update_sdkconfig_from_defaults.sh` Updates an existing `sdkconfig` file by merging and applying configuration values from a given defaults file, reporting any

changes made. A dependency of `run_all.sh`.

`generate_partitions.sh` Generates a final `partitions.csv` file containing the partition table that will be flashed on the device, calculating the offset and size of the attestation partition according to `sdkconfig.defaults` and filling them into a template CSV. It ensures the partition size is aligned to 4 KB and correctly positioned at the end of the flash memory.

`find_errors.sh` Scans log files for 'Guru Meditation Error' entries and extracts each error with its surrounding context into a consolidated output file.

`print_keys.sh` Extracts and displays device and vendor key sections from a binary partition file and corresponding DER/signature files in a human-readable hexadecimal format.

5.5.5 Framework Architecture

The architecture of the Attestation Framework was designed in order to be simple and extensible. It is designed around FreeRTOS, utilizing its Task and Queue constructions among others. Tasks are similar to processes as defined in Unix, and are switched by the RTOS scheduler, although because of the lack of memory translation in low-end IoT chips, they lack memory isolation by default. Queues are FIFO structures through which different tasks can send and receive items to and from each other.

Once booted, the Framework's startup sequence will complete the following operations:

1. Read the MAC address, private key, public key and the vendor's signature assigned to the device.
2. Register the Network Receiving Callback and start the Network Sending Tasks.
3. Start the Attestation Sending and Receiving Tasks.
4. Start the Vulnerable Application Task.

The main framework consists of teh following four tasks: Network Sending Task, Network Receiving Callback, Attestation Receiving Task and Attestation Sending Task. Figure 5.9 shows the architecture of the framework and how its tasks interact with each other, the WiFi Driver and the Application Task. All the tasks are run concurrently by the RTOS scheduler, be it with different priority values. Network related tasks have the highest priority of all, in order to be able to respond to incoming/outgoing messages, while attestation related tasks have higher priority than the Application Task, in order to ensure that in a case of a rogue application, it can never disallow the Attestation to operate.

The **Network Sending Task** is essentially an abstraction over the networking implementation, which in our case is ESP-NOW (Custom WiFi frames) but can be replaced with any other backend. It receives packets to be sent from the Attestation and Application Task through queues. Depending on the source of the packet, whether it was sent from the Application Task or an Attestation Task, it adds

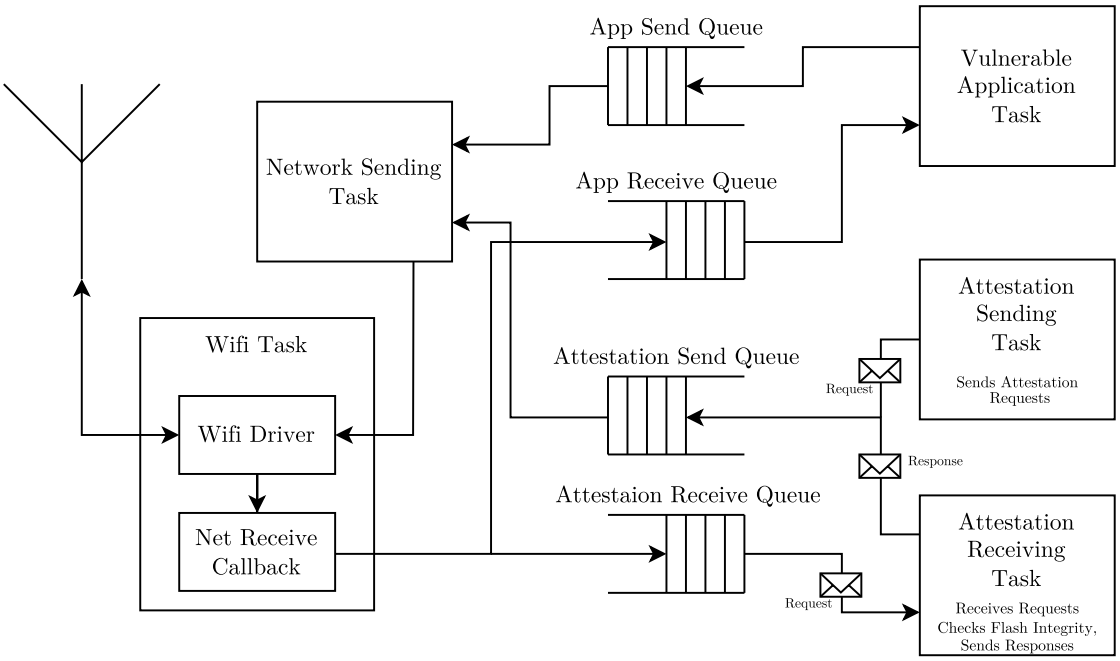

Figure 5.9: RTOS Task Diagram for the Attestation Framework [10]

a suiting magic number to the packet for differentiation. Finally it forwards the packets to the Wifi Driver via the API provided by ESP-IDF for the specific chip.

The **Network Receiving Callback** is registered with the Wifi Driver of the chip and is called similarly to an interrupt, anytime the device receives a WiFi frame. It's responsibility is to only check the packet's magic number, and forward the received packet to the correct queue, either for consumption by the Attestation Receiving Task or the Application Task.

The **Attestation Sending Task** is responsible for the initiation of all attestation related communication with neighbours. It will send the announcements during the *Exploration Phase*, start attestation rounds and send all attestation request.

The **Attestation Receiving Task** is responsible for receiving any attestation related messages and responding accordingly. As such it responds to announcement messages with recognition messages, to recognition messages with acknowledgments and to attestation requests with attestation responses, after completing the integrity check. It will also log any incoming attestation responses to the global state.

5.5.6 Framework Configuration Options

As noted before, the user can tweak many compile time constants that affect the timing, the behaviour and more specific technical details of the Attestation Framework. Many options concern the configuration of ESP-IDF and FreeRTOS, while others were defined just for this Framework. These options lie and should be configured in the `sdkconfig.defaults`. Additionally they can be configured through the command line graphical interface of `menuconfig`, but the user must opt to save the changes to `sdkconfig.defaults`. Some notable configuration options:

Device Configuration

`CONFIG_MY_NUM_DEVICES` The number of devices. Used in `run_all.sh`. In the current version of the framework, each device, after boot, attempts to find the according number neighbours. The framework can support more dynamic topologies with changing number of neighbours per device however the current behaviour is desirable for testing.

`CONFIG_MY_NEIGHBOURS_FOUND_WAIT_MS` How much time (ms) to wait after the device's neighbours have been found before starting Attestation Rounds. This induced delay proved useful to combat timing related issues.

`CONFIG_MY_TIME_FROM_LAST_ATT_RES_TO_RERECOGNIZE_MS` How much time must pass from the last successful attestation response from a neighbour to conclude that they have restarted and that their announcement needs to be replied.

Networking Configuration

`CONFIG_MY_PACKET_DATA_MAX_LENGTH` Maximum size in bytes of a packet sent through our custom networking interface that can be sent or received by either the (vulnerable) app or the attestation routines.

`CONFIG_MY_OFFLINE_RESEND_MAX_COUNT` The number of times a Recognition message is (re)sent while an Acknowledgement has not been received.

`CONFIG_MY_RESEND_DELAY_MS` Delay before resending a message.

`CONFIG_MY_MSG_ANNOUNCE_SEND_PERIOD_MS` Period of new broadcast announcements.

Attestation Timing

`CONFIG_MY_ATTESTATION_INTERVAL_MIN_MS, CONFIG_MY_ATTESTATION_INTERVAL_MAX_MS` The time period before the next Attestation Round is initiated by the device is chosen uniformly randomly in this range.

Logging Options

`CONFIG_MY_LOG_RECV_CB` Enables logging from the Network Receive Callback.

`CONFIG_MY_LOG_MESSAGES` Enables logging of all received and sent messages.

`CONFIG_MY_LOG_NEIGHBOUR_INFO` Enables logging of neighbours' info and attestation stats.

`CONFIG_MY_LOG_ATT_ROUNDS` Enables logging of finishing and new Attestation Rounds.

`CONFIG_MY_LOG_OK_RESP_INFO` Enables logging of successful Attestation Responses. Failed and timed out responses are always logged.

`CONFIG_MY_LOG_CURRENT_STATS_PERIOD_MS` Period (resolution) of CPU usage logging.

Benchmarking Options

`CONFIG_MY_BENCHMARKING` Enable benchmarking, with logging of CPU and RAM usage, and other statistics.

`CONFIG_MY_BENCHMARK_RUNTIME_MS` Duration of benchmark, after which stats will be logged by the devices.

Message Identification

`CONFIG_MY_ATTESTATION_MAGIC, CONFIG_MY_APP_MAGIC` Magic numbers that are appended to the start of network messages, depending on their type/origin (application or attestation module). Based on these the network module forwards each message to either the attestation module or the application.

Cryptography Options

`CONFIG_MY_PK_USE_EC_SECP192R1/SECP224R1/SECP384R1/SECP521R1` Which elliptic curve to use for the public key cryptography of the Framework (including vendor and device keys).

5.5.7 Implementation Challenges

The implementation of this framework required addressing several technical obstacles through extensive troubleshooting and iterative refinement. This section summarizes the primary challenges encountered during development, along with the strategies employed to resolve them.

Unflashable Chips

This issue is common with low-cost microcontrollers and was observed with the first batch of ESP32-C3 boards used in this project. These boards could not initially be flashed with new firmware using the ESP-IDF framework. The problem was resolved by consulting multiple troubleshooting guides and performing a carefully timed sequence of button presses on the boards, synchronized with flashing attempts from the host computer.

Unreliable USB Hubs

The USB hubs used to control, flash, and monitor all ten ESP32-C3 boards were sourced from a low-cost supplier. Their performance proved unreliable, frequently causing intermittent failures and delays during development. The hubs' inconsistent functionality introduced significant overhead to both deployment and testing cycles.

Network Timing Bugs

It was observed that the retransmission of unacknowledged high-frequency messages, combined with heavy incoming traffic during the *Exploration Phase*, occasionally prevented some devices from completing the three-way handshake with their neighbors. This caused a situation where some devices performed attestation rounds as

expected, while others remained stuck in the *Exploration Phase*. This was mitigated by allowing devices to join the network dynamically, even after neighboring devices had exited the *Exploration Phase*, and by optimizing network-related code and increasing message queue sizes to handle traffic bursts more effectively.

Messages Looping Back

An unexpected issue encountered during initial development was that devices were unable to receive one another's unicast messages. When Device A received a broadcast from Device B and attempted to send a unicast reply, the message failed to be delivered. This was traced to the fact that all devices were configured with identical default MAC addresses. As a result, when Device A attempted transmission, the Wi-Fi driver interpreted the destination MAC address as its own and suppressed the packet. The problem was resolved by assigning unique vendor-specific MAC addresses to each device.

Unexpected Integrity Check Failures

During testing of an alternative mutual attestation protocol, devices running identical firmware produced inconsistent attestation results, incorrectly indicating flash hash mismatches. Further investigation revealed that flashing new firmware did not erase unused sections of flash memory, resulting in differing unused byte sequences across devices. The issue was resolved by performing a complete flash erase on all devices prior to testing. This protocol was later abandoned in favour of the final attestation approach described in this thesis.

Memory Safety

The entire codebase was implemented in C, the language used by the ESP-IDF libraries, which does not provide memory safety guarantees. The most significant memory safety issue encountered was a buffer overflow discovered in an early development version, where certain devices failed their integrity check during attestation. Upon inspection, it was determined that the flash contents and their computed hashes were unchanged during runtime, but the reference *Golden Hash* stored in memory had been partially overwritten, with only its first byte altered. This was traced to a buffer overflow that occurred when reading ECC keys from flash into global arrays that were located adjacent to the *Golden Hash* in memory. The maximum key length calculation was corrected, and explicit bounds checks were introduced to prevent recurrence.

Iterations and Rewrites

Early versions of the framework were deliberately simplified to allow the development team to build familiarity with the platform. When asymmetric encryption and HMAC functionality were introduced, it became clear that a complete rewrite of the codebase was necessary. The final implementation was redesigned to meet cryptographic requirements while also providing greater modularity and flexibility for future extensions.

5.5.8 Implementation Omissions

The set goal of this Thesis was to build a working basis would easily be extended to incorporate more optimizations, more complex topologies and other features, and this project delivers by implementing all the core components and logic of the proposed Framework. The hardware guarantees that ensure the integrity of the attestation system—such as secure boot, secure storage, and FreeRTOS kernel and task isolation via RISC-V Physical Memory Protection—have been implemented previously and independently of the attestation context [27], [28], [24]. As such, they are left out of this Diploma Thesis and constitute future work for the Attestation Framework.

Chapter 6

Evaluation

This chapter presents the evaluation of our system under different configurations and workloads. The evaluation is divided into three main aspects: Time Series Graphs, Scaling Graphs, and Round Frequency Graphs. Each of the latter sections provides an analysis of system performance metrics such as CPU usage, RAM usage, response time, and attestation round completion time.

6.1 Methodology

For the collection of the data, the automated benchmark in `benchmark.sh` was used. The setup consisted of 10 purchased ESP32-C3 Supermini Boards, connected via multiple USB hubs to a PC where the benchmark was initiated from. The benchmarks explores every possible combination of number of devices (from 2 devices to at most 10) and average attestation round period (from 2 seconds to 20 seconds).

Every combination is run for 100 seconds on the devices. The devices are constantly logging their CPU usage attributed to the attestation related tasks, every 100ms, while at the end of the run their timeout percentage, their average CPU usage, RAM footprint, minimum, maximum, average round and response time. These are collected by the benchmarking script, compiled into CSV files in order to be processed into figures by a python script.

Note that for the purposes of benchmarking, during a run with e.g. an attestation round period (average) of T seconds, the actual time between two consecutive attestation rounds initiated by a device is picked randomly and uniformly from the range $(T-0.5s, T+0.5s)$. For the benchmark, the maximum time allowed to elapse between an attestation request and an attestation response before before it is regarded as timed out, is $T-1s$. If the response takes more than that time to arrive then it counts as a timeout, and it's response time is not taken into account in the statistics collection of response time and attestation round time.

6.2 CPU Usage Over Time

The following time series graphs illustrate the CPU usage over time for different numbers of devices with fixed attestation round periods. The usage shown originated from the measurements on a randomly selected device of the network. The attestation periods are configurable in the evaluation scripts. All period labels indicate (±0.5s) to reflect the uniform random variation applied to the round period selection.

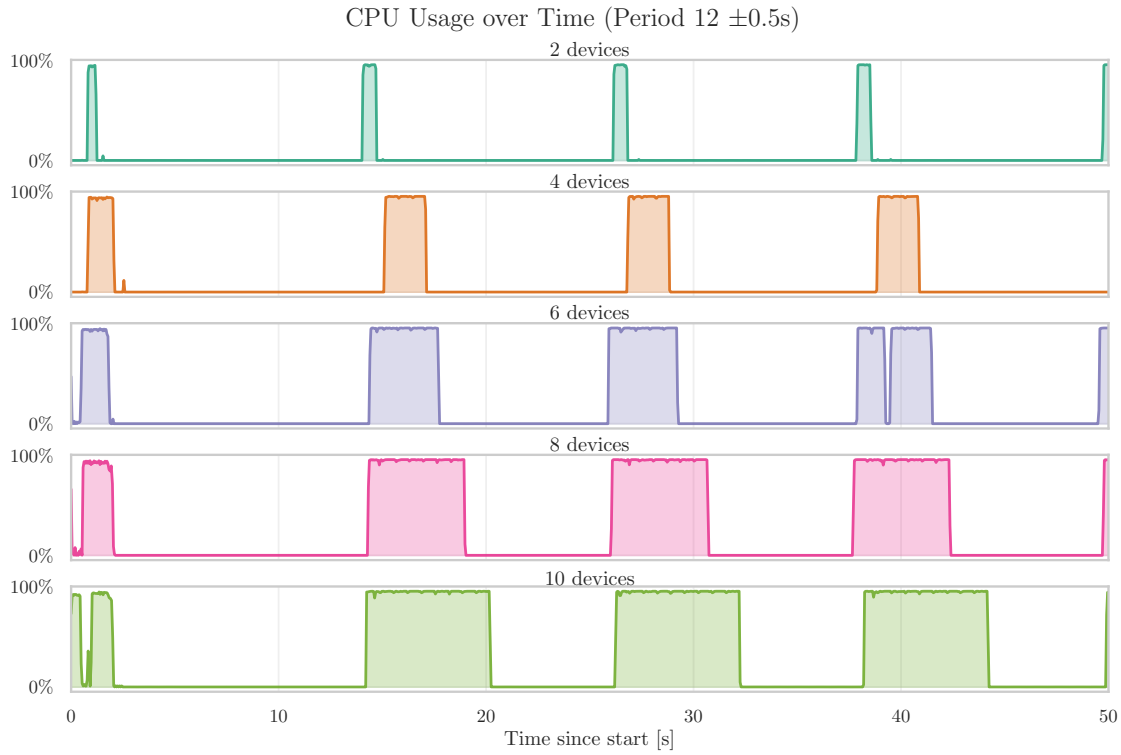

Figure 6.1: CPU usage over time for a round period of 12 seconds. Lines correspond to 2, 4, 6, 8 and 10 devices.

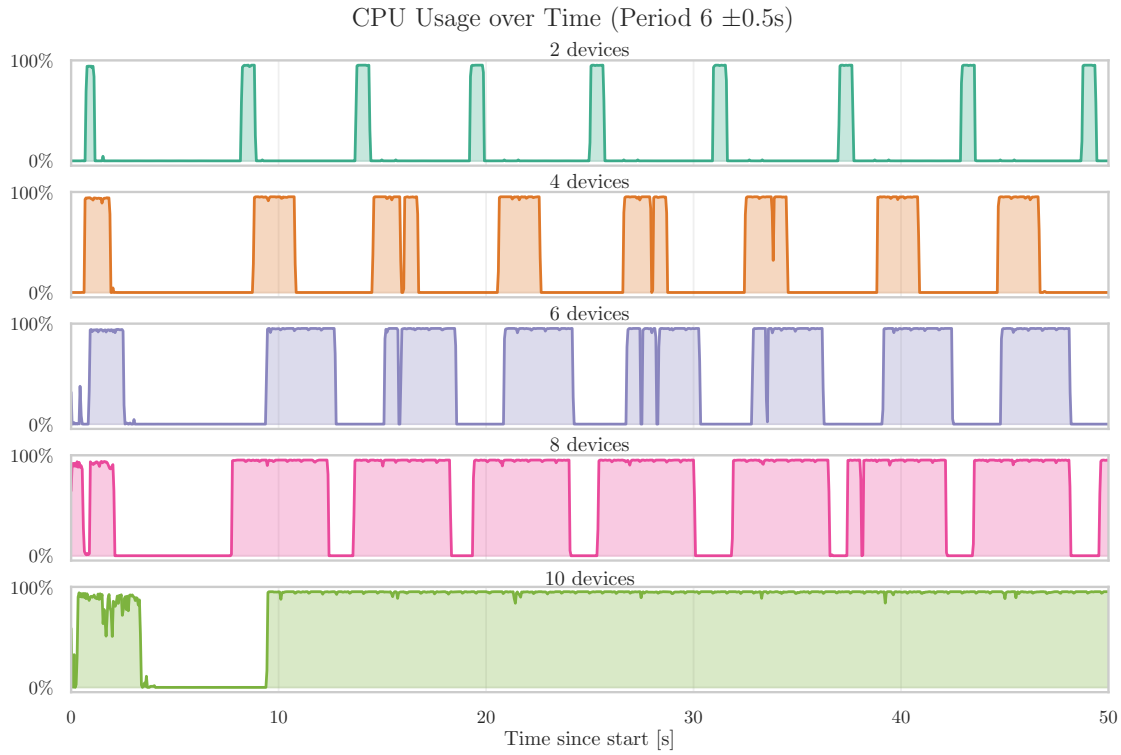

Figure 6.2: CPU usage over time for a round period of 6 seconds. Lines correspond to 2, 4, 6, 8 and 10 devices.

As depicted in Figures 6.1 and 6.2, the CPU usage exhibits distinct, periodic spikes. These spikes directly reflect the configured attestation period, as each spike corresponds to a group of closely timed attestation rounds of peers. The high-CPU usage phases are caused by the system having to respond to concentrated attestation requests, which include performing costly integrity measurements. Specifically, the integrity check involves hashing the entire 4 MB flash memory for every single incoming request, a computationally intensive task, even with the hardware acceleration of SHA256 in the chip. During the intensive phases, the attestation is responsible for effectively 100% CPU usage. It is also observable that as the number of devices increases, the duration of these spikes increase, indicating that the system requires more processing time to handle larger sets of devices concurrently.

Conclusions

These figures showcase the type of load the operation of the attestation framework that is put on the devices. Since all devices initiate attestation rounds with periods selected uniformly from a small range, all the requests the neighbours receive are concentrated and thus the CPU usage of the attestation tasks follow this pattern, of very high and enduring spikes, followed by near zero use.

6.3 Scaling Graphs

Scaling graphs analyse system performance as the number of devices increases. We tested with a variable number of devices in order to simulate small networks, incrementally increasing their size and observing the system's overall responsiveness. Different constant attestation round periods are compared.

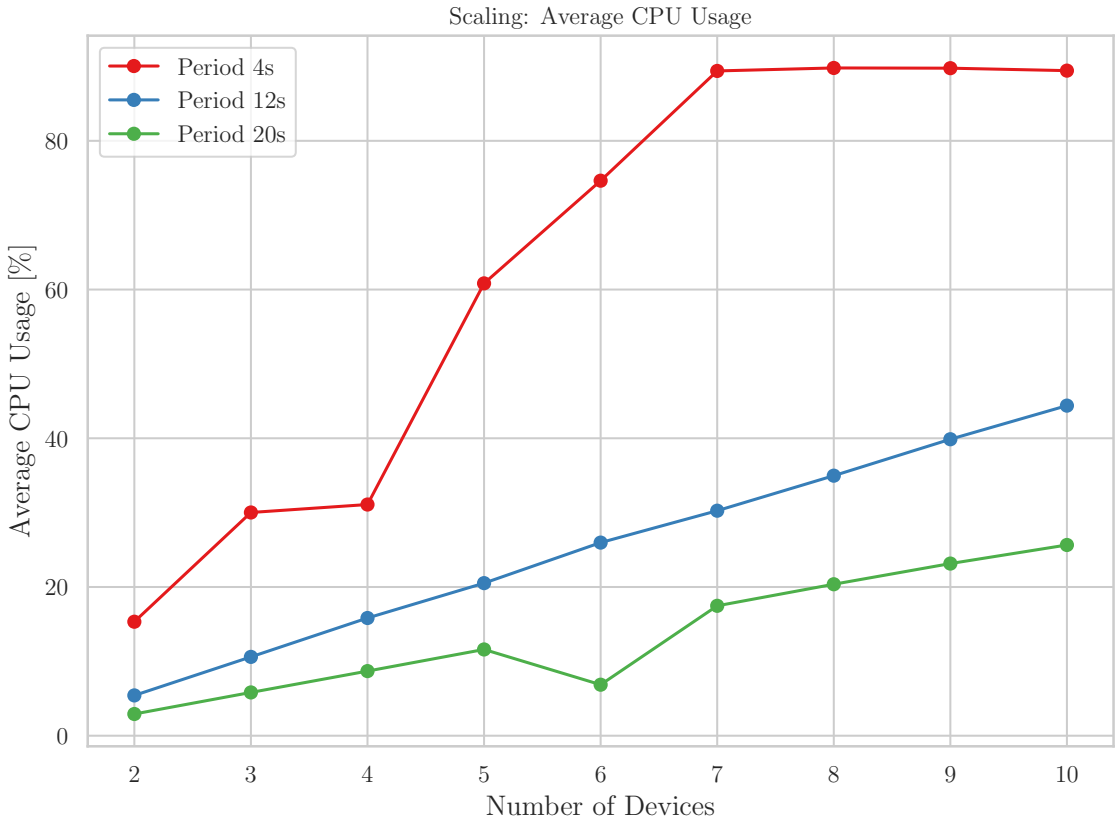

Figure 6.3: Average CPU usage versus number of devices for three different attestation periods. As expected, CPU usage increases with the number of devices and higher frequency rounds.

Figure 6.3 illustrates that average CPU usage scales approximately linearly with the number of participating devices. The slope is steeper for shorter attestation periods, reflecting the higher overall system load due to more frequent rounds. For the shortest period of 4 seconds, CPU usage begins to plateau near saturation around 7-8 devices, indicating a potential performance bottleneck under high load.

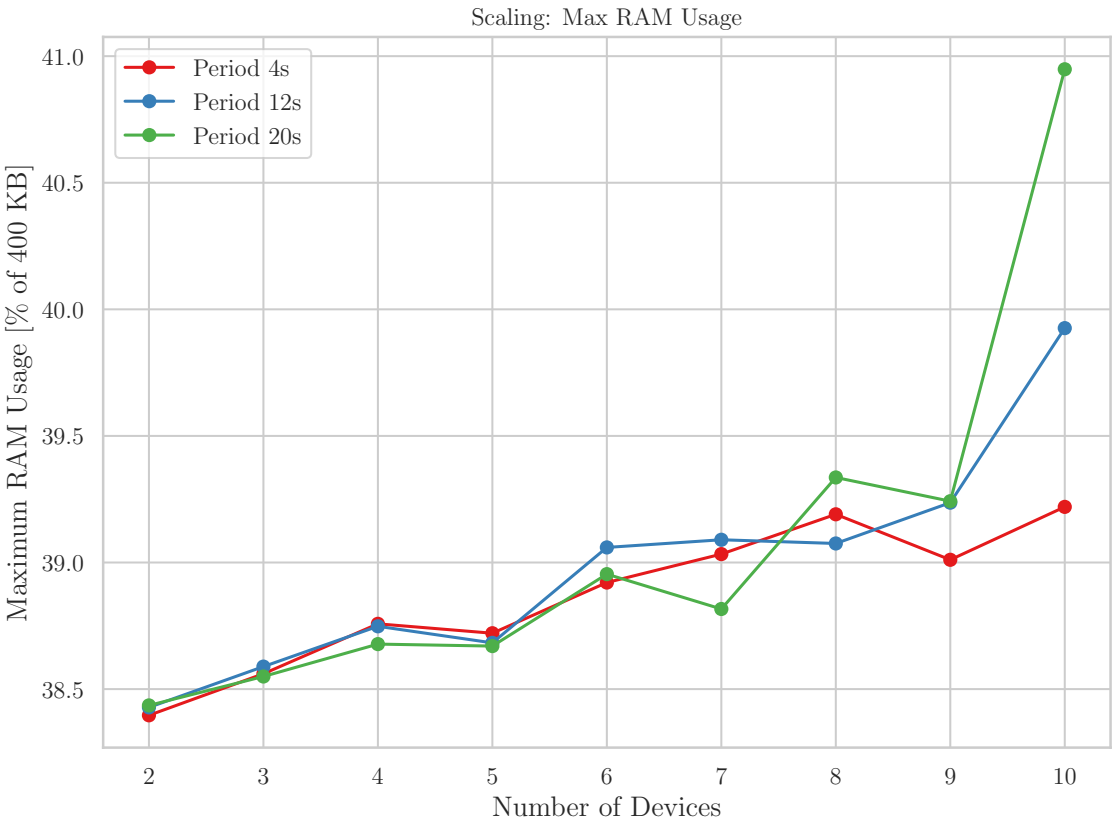

Figure 6.4: Maximum RAM usage (stack + total heap) versus number of devices. Three different attestation periods are shown. Memory usage remains relatively stable with increasing devices.

The analysis of maximum RAM usage (Figure 6.4) reveals that memory consumption is relatively insensitive to the number of devices. This suggests that the system's memory footprint is dominated by baseline requirements such as the operating system kernel and device drivers, rather than per-device overhead, which is a positive indicator for scalability.

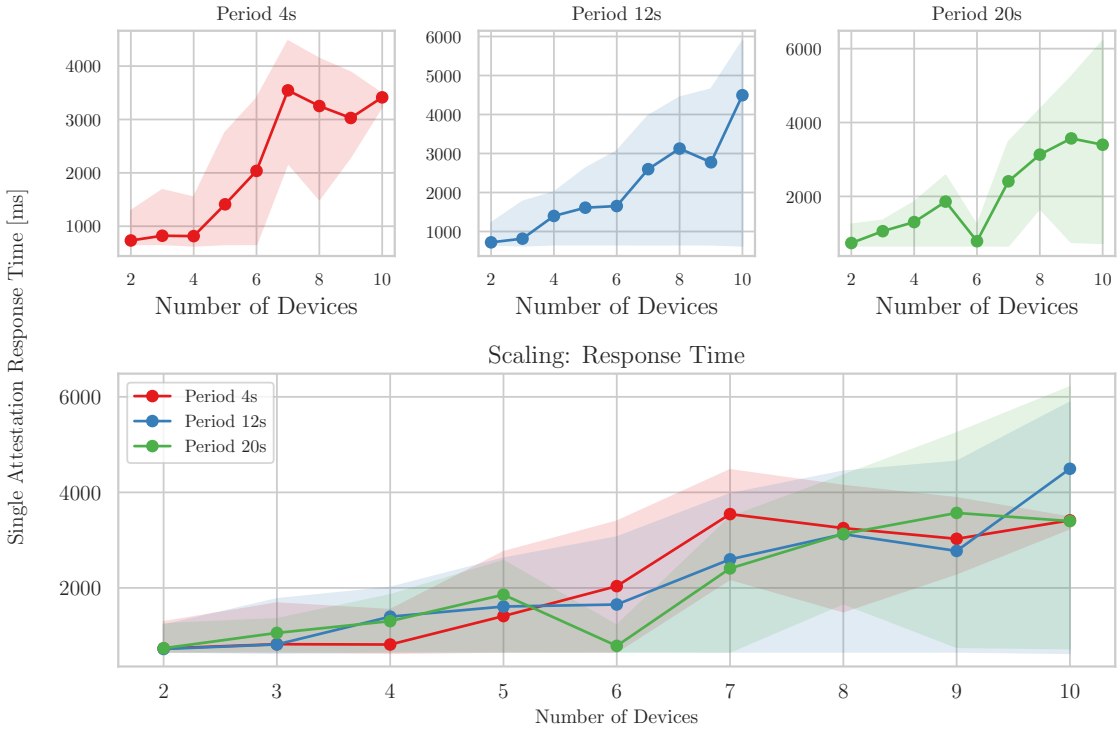

Figure 6.5: Response time statistics (average, minimum, maximum) of single attestation requests versus number of devices. The plot highlights how response time grows with higher device counts.

Figure 6.5 demonstrates that both average and maximum response times increase with the number of devices. The widening gap between minimum and maximum response times, particularly for shorter attestation periods, reflects growing contention and variability in system performance under increased load.

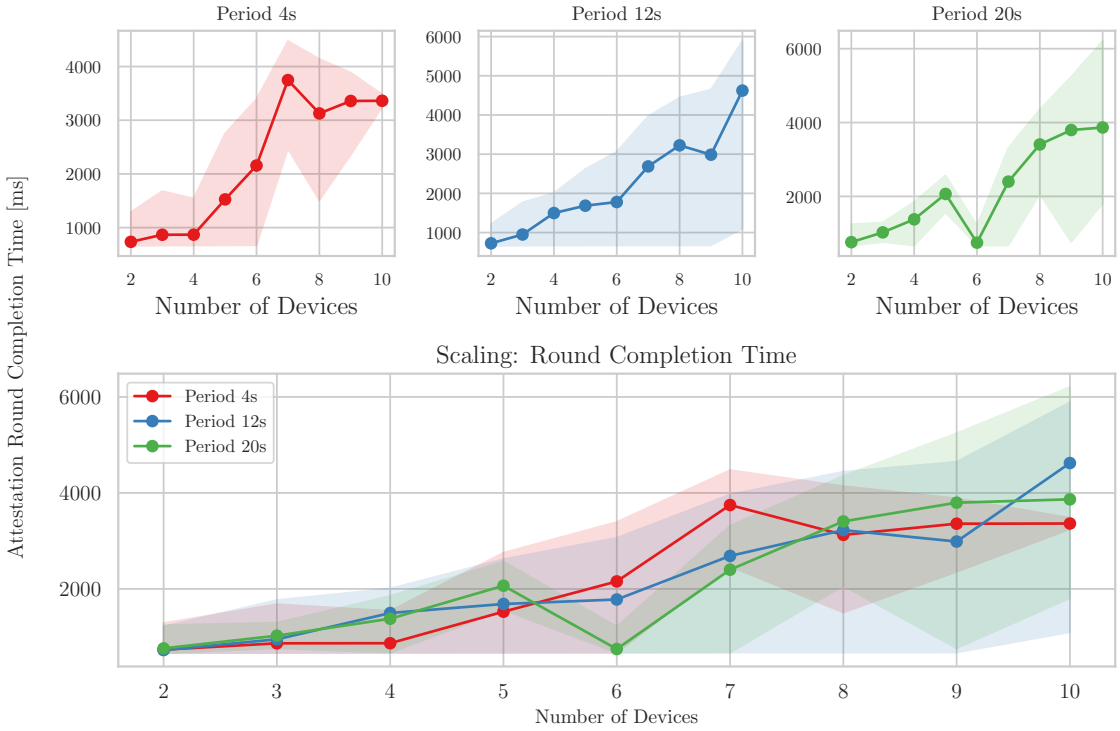

Figure 6.6: Attestation round completion time (average, minimum, maximum) versus number of devices. Completion time scales with device count and attestation frequency.

Figure 6.6 shows that the total attestation round completion time increases with device count and attestation frequency. Timed out responses are excluded from this metric, but round completion times approaching the configured period indicate a strain on the system and potential instability.

Conclusions

The scaling analysis indicates that CPU usage grows approximately linearly with the number of devices and the frequency of attestation rounds, while memory usage remains relatively stable. High-frequency attestations combined with a large number of devices can saturate the CPU and increase response times significantly, potentially creating bottlenecks. The widening variability in response time under high load highlights the limits of scalability and the importance of balancing device count with attestation frequency in the current topology.

6.4 Round Frequency Graphs

Round frequency graphs explore system behaviour with varying attestation periods. Period labels include (± 0.5s) to reflect the randomized selection within the range.

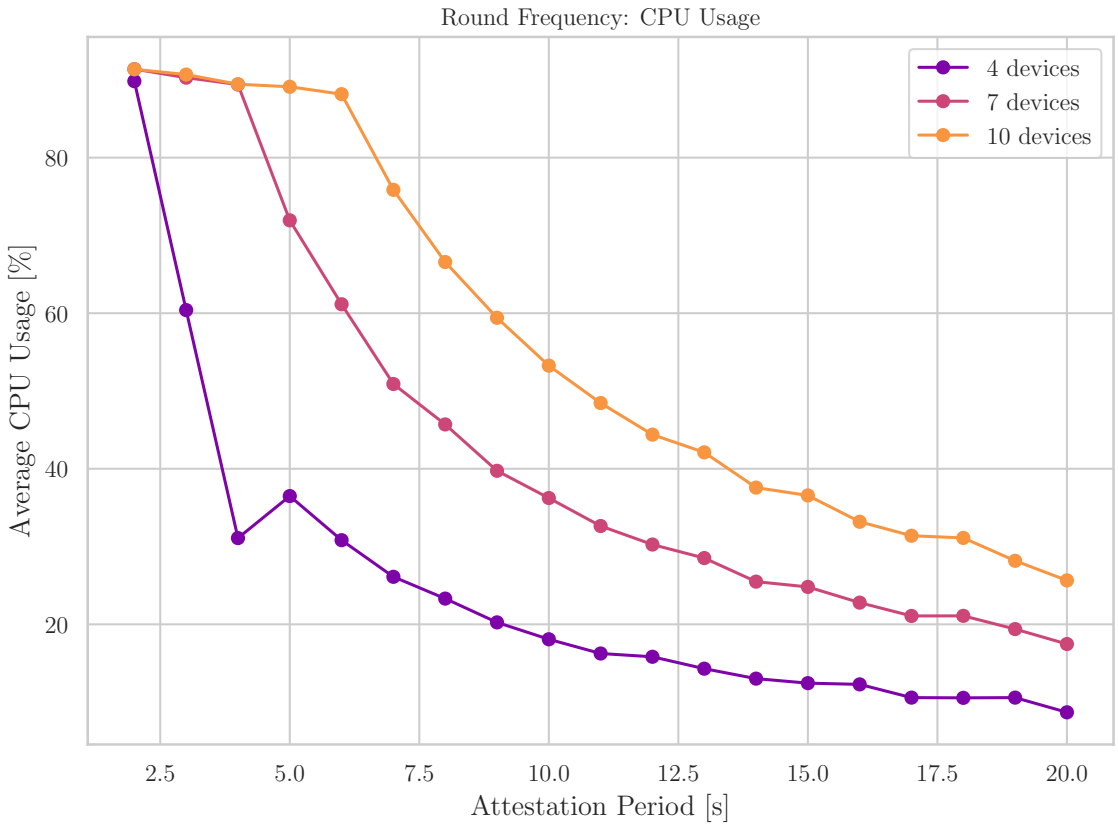

Figure 6.7: Average CPU usage versus attestation period for 4, 7 and 10 devices. CPU usage decreases as the attestation period increases.

Figure 6.7 demonstrates that average CPU usage decreases as the attestation period increases. Longer intervals between attestation rounds allow the CPU to remain idle for longer periods, reducing overall load. Short periods result in frequent, concentrated processing of attestations, creating peaks in CPU usage similar to those seen in the time series graphs.

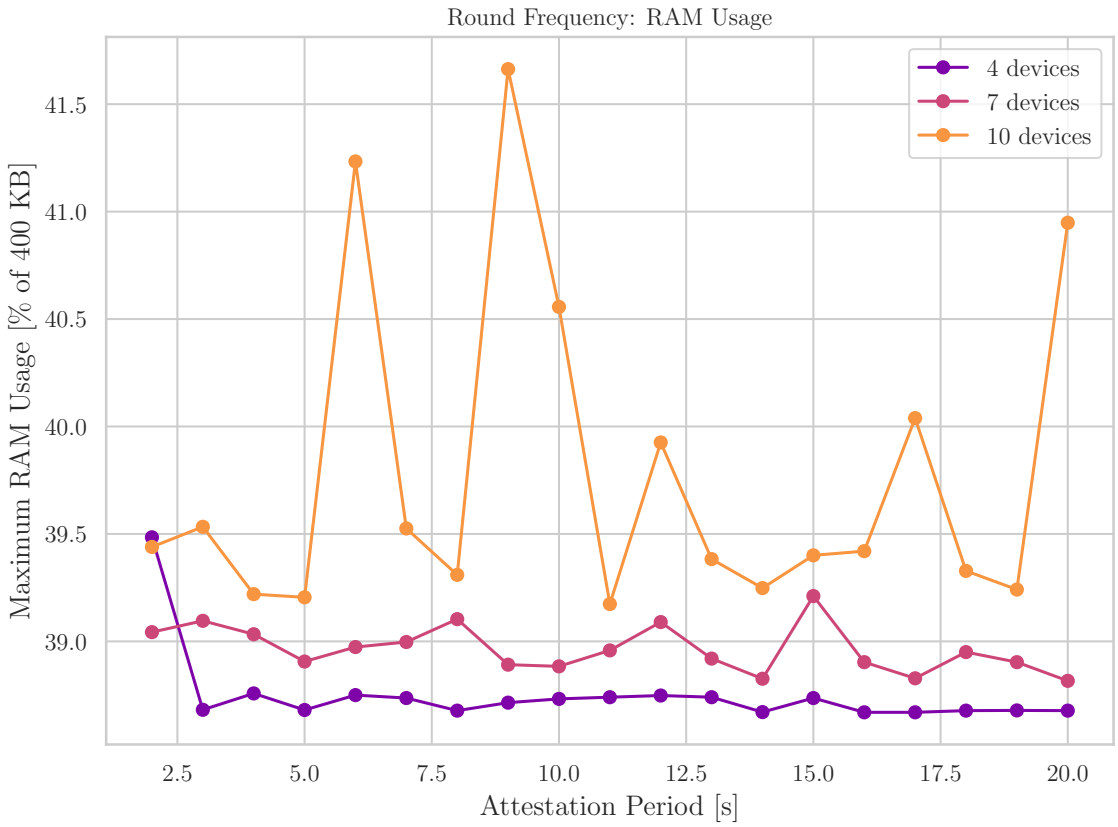

Figure 6.8: Maximum RAM usage versus attestation period for 4, 7 and 10 devices. Memory consumption remains mostly constant across periods.

Maximum RAM usage (Figure 6.8) remains largely unaffected by attestation frequency, confirming that memory is dominated by baseline system requirements rather than per-round allocations. The RAM footprint's peaks visible in the graph are minor in true scale and can be attributed to timing differences between different benchmark runs, that result in fewer or more heap concurrent heap allocations and thus affect the all time maximum RAM usage. It can be safely concluded that the memory footprint of the attestation task is in fact independent independent from the attestation round frequency.

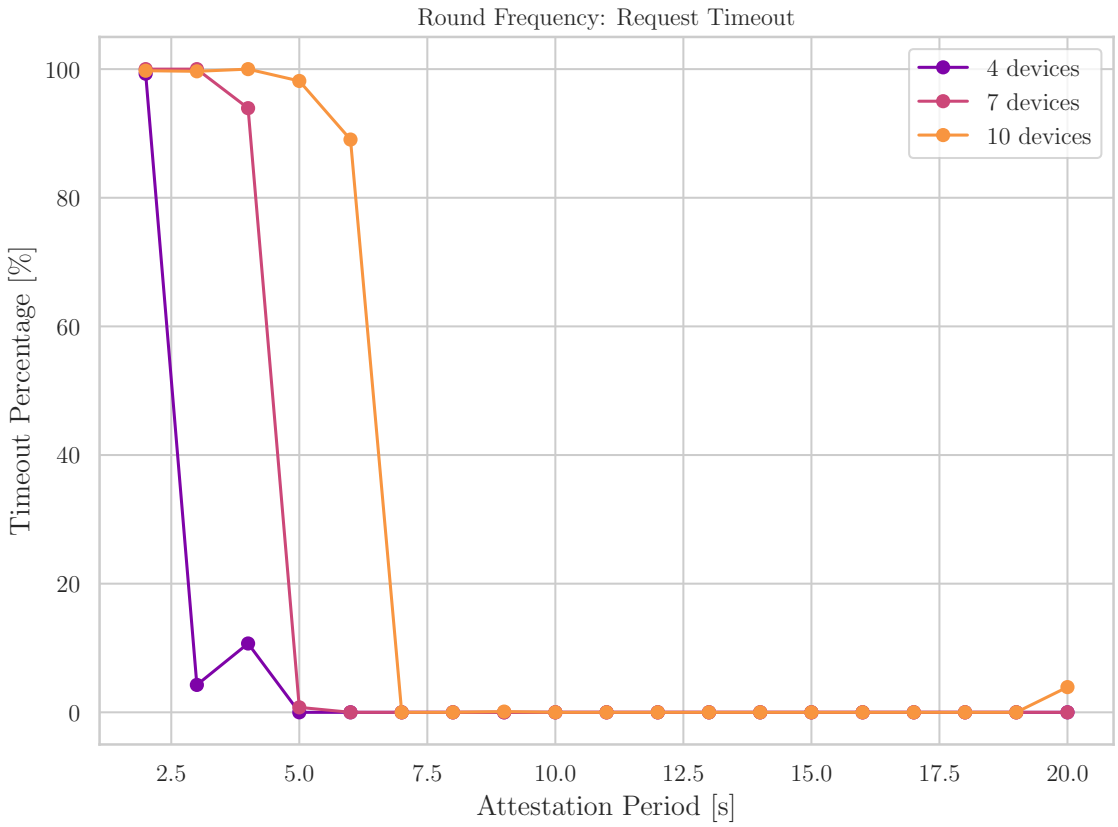

Figure 6.9: Request timeout percentage versus attestation period for 4, 7 and 10 devices. Higher frequency rounds increase the probability of timeouts.

Figure 6.9 reveals that short attestation periods, particularly with many devices, dramatically increase the probability of request timeouts. Beyond a certain critical period, timeouts drop sharply and system performance stabilizes.

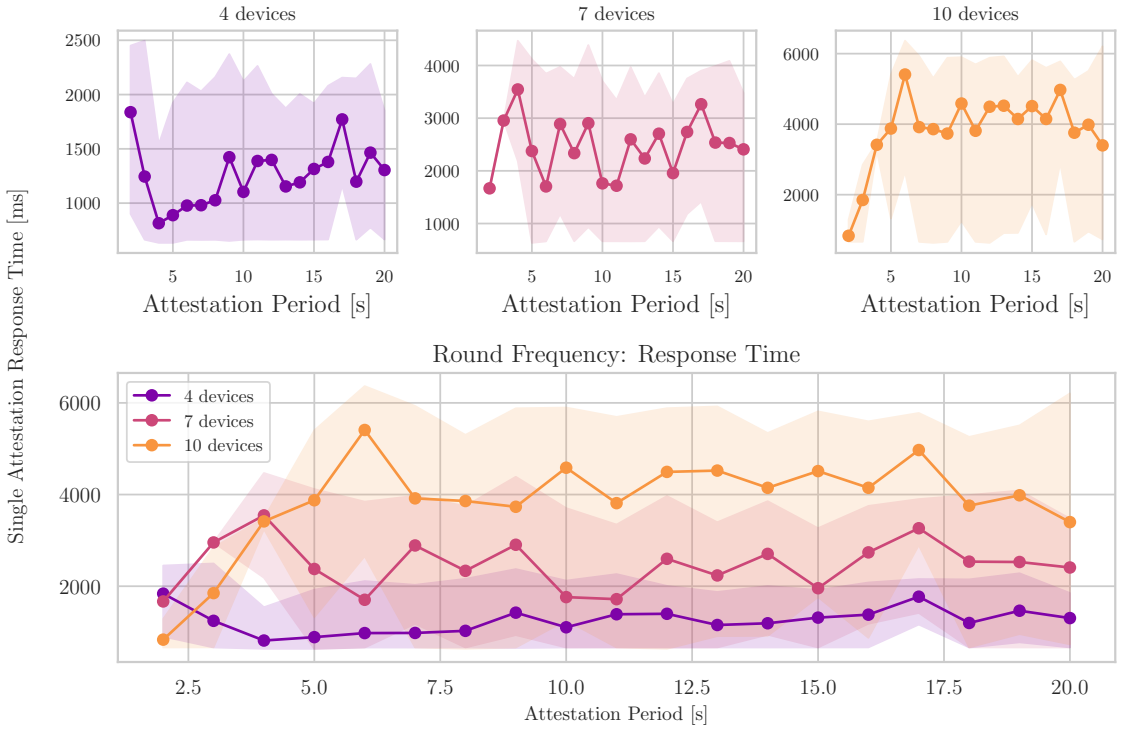

Figure 6.10: Single attestation request response time (average, minimum, maximum) versus attestation period for 4, 7 and 10 devices. Times exceeding the timeout are ignored.

Figure 6.10 critically does not include timed out responses. Thus it becomes clear that once the attestation period is above the critical threshold for each configuration, the average response time for a single request stabilizes.

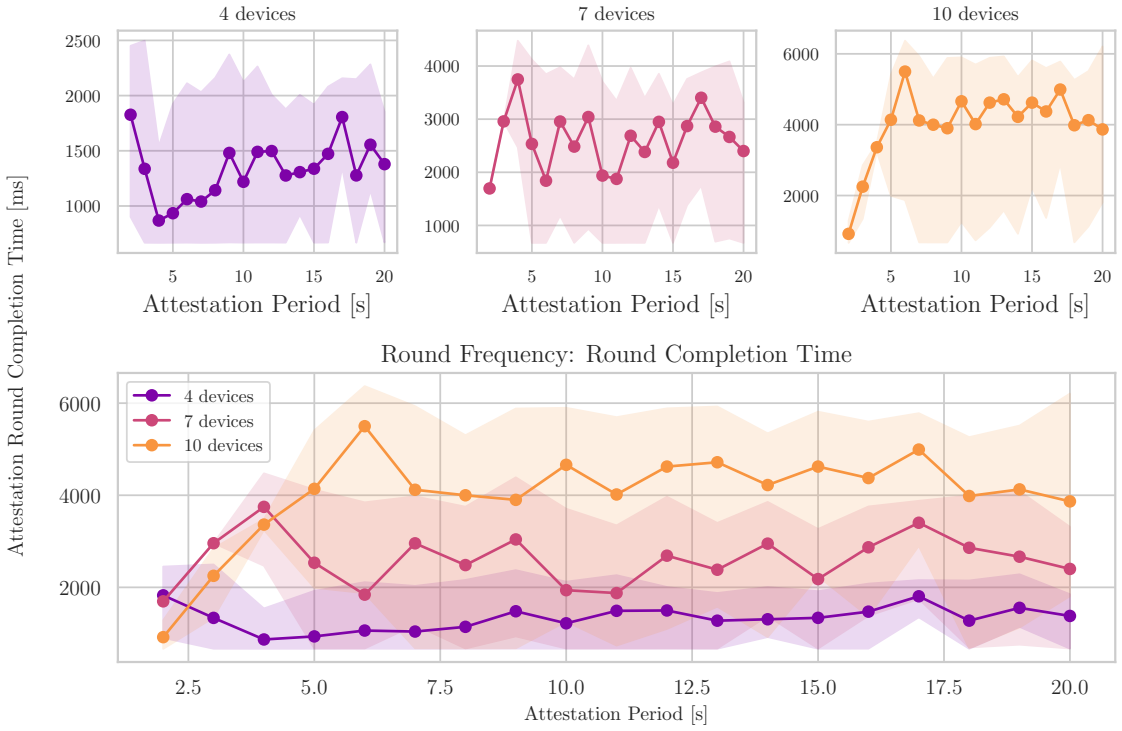

Figure 6.11: Attestation round completion time (average, minimum, maximum) versus attestation period for 4, 7 and 10 devices. Completion times exceeding the timeout are ignored.

Figure 6.11 shows that total round completion time also stabilizes once attestation periods exceed the critical value, further confirming that the system performs reliably only above this threshold.

Conclusions

The round frequency analysis demonstrates that CPU usage is strongly dependent on attestation frequency, while memory consumption remains largely unaffected. Short attestation periods create high computational load, leading to increased request timeouts and longer round completion times. There exists a critical attestation period for each device count, below which the system becomes overloaded and unreliable. Above this critical period, the system stabilizes, with predictable response times and round completion times. These findings emphasize the importance of configuring attestation periods to balance the trade-off between verification frequency and system reliability.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis presented the design, implementation, and evaluation of a lightweight and scalable Remote Attestation framework tailored for IoT devices, leveraging RISC-V Physical Memory Protection (PMP) as the hardware Root of Trust. The core contribution lies not only in the implementation but also in the protocol design, which combines nonce-based freshness guarantees, replay-attack resilience, and decentralized peer-to-peer attestation to enable collective integrity verification without relying on a single point of trust. The framework follows a two-phase approach, with an offline setup for key provisioning and an online phase that performs periodic attestation rounds using a tree-based aggregation model to reduce communication overhead.

Our implementation on the ESP32-C3 platform demonstrated that the framework can operate reliably under varying network sizes and attestation frequencies, providing predictable and reproducible results once the system operates above a critical attestation period. The evaluation confirmed that CPU utilization is the dominant performance factor influenced by attestation frequency, while memory consumption remains relatively constant regardless of network scale. The findings also highlight the importance of choosing appropriate round intervals: too frequent attestations introduce congestion and missed deadlines, whereas appropriately tuned intervals yield stable round completion times and low timeout rates.

By focusing on a design that is targeted on existing low-cost platforms, this work establishes a practical foundation for building secure, decentralized trust mechanisms in resource-constrained IoT networks with Remote Attestation. The protocol and implementation together provide a bridge between purely theoretical collective attestation schemes and deployable solutions, opening the way for further research and extensions to this framework, that can be directly adopted by the IoT industry.

7.2 Future Work

While the implementation and evaluation presented in this thesis demonstrate the feasibility of collective remote attestation on low-cost IoT devices, there are several directions for future work that can enhance security, scalability, and robustness. These improvements span hardware-based isolation mechanisms, cryptographic support, runtime protection, and overall system optimization.

- **Secure Boot and Secure Storage for Keys** Extend the framework's implementation with hardware-backed secure boot and secure storage (utilizing efuses, flash encryption) to verifiably setup the attestation tasks and correctly, and providing exclusive access to encrypted keys to the attestation system.
- **RISC-V PMP for FreeRTOS Task Separation and Benchmarking:** Integrate Physical Memory Protection (PMP) additions to the FreeRTOS kernel to isolate tasks and protect sensitive routines, keys, and memory regions from untrusted code into the framework implementation. This addition, complemented by the Secure Boot and Secure Storage mechanism would complete all hardware guarantees of the protocol.
- **Incorporate Watchdog timer:** Instead of performing an Integrity Measurement on every incoming request.
- **Different Hash Implementaitons:** Add support for SHA-3 and BLAKE3 in addition to the current implementation using SHA2 (256 bit).
- **Finalize RSA Support:** As an alternative to ECC, fully integrate RSA-based asymmetric cryptography for attestation signatures and verifier authentication, providing a robust alternative to symmetric-key MAC-based schemes and enabling public verifiability of attestation results. ECC was prioritized in the current implementation due to it's better performance on the ESP32-C3
- **Healing System:** Implement a self-healing mechanism inspired by HEALED [8], where compromised devices can securely receive and install verified patches from trusted peers or a management node, reducing the need for manual intervention.
- **Improve Protocol Scalability:** Add support for automatic spanning tree formation and other topologies for efficient attestation across massive networks, including but not limited to the mechanism in SEDA [4].
- **DDoS Protection:** Introduce rate limiting, challenge–response mechanisms, and distributed filtering strategies to mitigate flooding attacks that could disrupt attestation rounds or saturate network resources.
- **Optimization:**
  - **Heap/Stack Usage:** Minimize memory footprint through careful allocation strategies and compile-time optimizations, allowing the framework to run on even more constrained devices.
  - **CPU, Hash Calculation, and Probabilistic Integrity Measure-**

ment: Explore faster hashing algorithms and probabilistic or incremental measurement techniques to reduce CPU cycles spent on attestation.

In summary, the work presented here serves as a robust foundation for future research. Extending the framework with stronger isolation guarantees, adaptive topologies, runtime monitoring, and self-healing mechanisms will further increase its resilience against sophisticated adversaries. Additionally, continued performance tuning will enable deployment in even more resource-constrained and large-scale IoT environments, moving closer to practical, industry-ready collective attestation.

Bibliography

[1] P. Myers, "Interfacing using serial protocols using spi and i2c", *Proc. ESP*, vol. 2005, pp. 1–9, 2007.

[2] H. Tan, W. Hu, and S. Jha, "A tpm-enabled remote attestation protocol (trap) in wireless sensor networks", in *Proceedings of the 6th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, 2011, pp. 9–16.

[3] Trusted Computing Group, *Tpm main specification*, Version 1.2, Revision 116, 2011. [Online]. Available: https://trustedcomputinggroup.org/resource/tpm-main-specification/.

[4] N. Asokan et al., "Seda: Scalable embedded device attestation", in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 964–975.

[5] J. Finkle. "U.s. firm blames russian 'sandworm' hackers for ukraine outage". Accessed: 2025-09-10. [Online]. Available: https://www.reuters.com/article/us-ukraine-cybersecurity-sandworm-idUSKBN0UM00N20160108/.

[6] S. Geetha and D. Cicilia, "Iot enabled intelligent bus transportation system", in *2017 2nd International Conference on Communication and Electronics Systems (ICCES)*, 2017, pp. 7–11. DOI: 10.1109/CESYS.2017.8321235.

[7] S. Elhadi, A. Marzak, N. Sael, and S. Merzouk, "Comparative study of iot protocols", *Smart Application and Data Analysis for Smart Cities (SADASC'18)*, 2018.

[8] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, "Healed: Healing & attestation for low-end embedded devices", in *International Conference on Financial Cryptography and Data Security*, Springer, 2019, pp. 627–645.

[9] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, "Collective remote attestation at the internet of things scale: State-of-the-art and future challenges", *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2447–2461, 2020.

[10] Espressif Systems, *Esp32-c3 series datasheet v2.1: Ultra-low-power soc with risc-v single-core cpu*, version 2.1, ESP32-C3: 2.4 GHz Wi-Fi (802.11b/g/n) and Bluetooth 5 (LE), Nov. 2020. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf.

[11] S. J. Ramson, S. Vishnu, and M. Shanmugam, "Applications of internet of things (iot) – an overview", in *2020 5th International Conference on Devices, Circuits and Systems (ICDCS)*, 2020, pp. 92–95. DOI: 10.1109/ICDCS48716.2020.243556.

[12] Trusted Computing Group, "Dice attestation architecture, version 1.00, revision 0.22", Trusted Computing Group (TCG), Technical Report Version 1.00 / Revision 0.22, Sep. 2020, Public Review. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG_DICE_Attestation_Architecture_r22_02dec2020.pdf.

[13] M. N. Aman et al., "Prom: Passive remote attestation against roving malware in multicore iot devices", *IEEE Systems Journal*, vol. 16, no. 1, pp. 789–800, 2021.

[14] I. De Oliveira Nunes, S. Jakkamsetti, N. Rattanavipanon, and G. Tsudik, "On the toctou problem in remote attestation", in *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, 2021, pp. 2921–2936.

[15] C. Özarpa, İ. Avcı, B. F. Kınacı, S. Arapoğlu, and S. A. Kara, "Cyber attacks on scada based traffic light control systems in the smart cities", *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 46, pp. 411–415, 2021.

[16] C. Shepherd, K. Markantonakis, and G.-A. Jaloyan, "Lira-v: Lightweight remote attestation for constrained risc-v devices", in *2021 IEEE Security and Privacy Workshops (SPW)*, 2021, pp. 221–227. DOI: 10.1109/SPW53761.2021.00036.

[17] A. Waterman, K. Asanović, and J. Hauser, "The risc-v instruction set manual, volume ii: Privileged architecture", RISC-V International, Technical Report Version 20211203, Dec. 2021, Document Version 20211203. [Online]. Available: https://www.scs.stanford.edu/~zyedidia/docs/riscv/riscv-privileged.pdf.

[18] E. Aliaj, I. D. O. Nunes, and G. Tsudik, "{Garota}: Generalized active {root-of-trust} architecture (for tiny embedded devices)", in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2243–2260.

[19] K. Cheang, C. Rasmussen, D. Lee, D. W. Kohlbrenner, K. Asanović, and S. A. Seshia, "Verifying risc-v physical memory protection", *arXiv preprint arXiv:2211.02179*, 2022.

[20] A. Goudarzi, F. Ghayoor, M. Waseem, S. Fahad, and I. Traore, "A survey on iot-enabled smart grids: Emerging, applications, challenges, and outlook", *Energies*, vol. 15, no. 19, 2022, ISSN: 1996-1073. DOI: 10.3390/en15196984. [Online]. Available: https://www.mdpi.com/1996-1073/15/19/6984.

[21] S. Soderi, D. Masti, and Y. Z. Lun, "Railway cyber-security in the era of interconnected systems: A survey", *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 7, pp. 6764–6779, 2023.

[22] D. P. T. F. Iliadis, "Accelerating the secure boot process in modern microcontrollers", Master's thesis, National Technical University of Athens, Athens, Greece, 2023. [Online]. Available: http://artemis.cslab.ece.ntua.gr:8080/jspui/handle/123456789/18668.

[23] S. Ahmadi, J. Le-Papin, L. Chen, B. Dongol, S. Radomirovic, and H. Treharne, "On the design and security of collective remote attestation protocols", *arXiv preprint arXiv:2407.09203*, 2024.

[24] C. Larmann, "Secure task management in freertos: A risc-v core approach with physical memory protection", M.S. thesis, Delft University of Technology, Aug.

2024. [Online]. Available: https://resolver.tudelft.nl/uuid:25ac1dc8-7d80-485b-be33-b4f43b6c379e.

[25] S. Sisinni, D. G. Berbecaru, V. Donnini, and A. Lioy, "Match-in: Mutual attestation for trusted collaboration in heterogeneous iot networks", in *2024 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2024, pp. 1–6.

[26] Associated Press. "Beirut airport screens hacked amid intensifying hezbollah-israel clashes". Accessed: 2025-09-10. [Online]. Available: https://apnews.com/article/lebanon-beirut-airport-hack-hezbollah-israel-5d5855c3635b429d63491993f7748a64.

[27] Espressif Systems, *Secure boot v2 - esp32-c3 (esp-idf programming guide v5.4.2)*, Accessed: 2025-09-20, Espressif Systems, 2025.

[28] Espressif Systems, *Security overview — esp32-c3 (esp-idf programming guide* `stable`*)*, Accessed: 2025-09-20, Espressif Systems, 2025.

[29] E. Systems, *Esp-now - esp32-c3*, Accessed: 2025-09-19, 2025. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v5.4.2/esp32c3/api-reference/network/esp_now.html.

[30] *Esp32-c3 development board esp32 super mini wifi bluetooth module*, https://www.alibaba.com/product-detail/ESP32-C3-Development-Board-ESP32-Super_1601082263140.html, Accessed: 2025-09-09.