July 18, 2026

Course Instructor
ENG 2003 – Effective Engineering Communication
Lassonde School of Engineering

Subject: Submission of draft technical report on zero trust IoT firmware and secure boot

Dear Instructor,

Please find enclosed the draft technical report prepared for Term Project 1. The report addresses a grand challenge identified by security researcher Bruce Schneier in the BBC feature on the great challenges of the twenty-first century: because the Internet of Things gives computers direct control over physical objects, ordinary software vulnerabilities are no longer confined to data. They can now cause catastrophic physical failures in systems such as vehicles, medical devices, and appliances.

The report examines this challenge through the discipline of software security engineering and proposes zero trust IoT firmware, anchored by secure boot protocols, as a direct technological response. It explains the underlying cause of physical IoT failures, describes what the proposed technology is and how it is currently being developed, evaluates its strengths and limitations, and justifies its potential to address the challenge. Evidence is drawn from five recent peer-reviewed sources spanning healthcare IoT, general IoT security, safety-critical communication, smart grid cyber-physical systems, and over the air firmware updates, supported by two figures and two tables.

This document is a draft submitted for peer review. Feedback on the clarity of the argument, the integration of evidence, and the treatment of the technology's trade-offs would be especially valuable and will guide the revised final report.

Respectfully submitted,
Mahan Ghafarian
Student Number 222066005
ENG 2003 – Term Project 1

Securing the Physical World: Mitigating Catastrophic IoT Failures through Zero Trust Firmware and Secure Boot

A Technical Report and Proposal

Grand Challenge (Bruce Schneier): catastrophic physical failures caused by software vulnerabilities in the Internet of Things

Proposed Technology: Zero Trust IoT Firmware and Secure Boot Protocols

Discipline: Software Security Engineering

Prepared by Mahan Ghafarian

Student Number 222066005

Prepared for ENG 2003 – Effective Engineering Communication

July 18, 2026

Executive Summary

The Internet of Things (IoT) has dissolved the boundary between software and the physical world: embedded devices now steer vehicles, regulate medical therapy, and balance electrical grids. Security researcher Bruce Schneier warns that once computers can act directly on physical objects, ordinary software vulnerabilities stop being data privacy problems and become causes of catastrophic physical failure [6]. This report examines that challenge from the perspective of software security engineering and proposes zero trust IoT firmware, anchored by secure boot protocols, as a direct mitigation.

Recent peer-reviewed literature is used to establish that the root cause of physical IoT failures is an implicit trust model in which a device is trusted by default once it joins a network or completes boot. Layer-wise threat surveys of healthcare IoT [1] and general IoT [2], an empirical safety-critical communication study [3], and a smart grid cyber-physical review [4] each document how this assumption enables false data injection, device impersonation, and compromised firmware updates that translate into physical harm.

The proposed solution applies the zero trust principle of never trust, always verify at the firmware level. A hardware root of trust cryptographically verifies each firmware image before it executes (secure boot), continuous runtime attestation reverifies device integrity rather than trusting a device indefinitely, and every over the air update is signature verified with rollback protection [5]. The report shows that these primitives already exist and have been validated in industrial practice, evaluates their strengths and trade-offs, most notably the compute and energy overhead on resource-constrained devices, and concludes that zero trust firmware is a feasible and necessary foundation for trustworthy cyber-physical IoT.

Table of Contents

List of Figures

List of Tables

1 Introduction and Background

1.1 The Challenge: When Software Failures Become Physical

The Internet of Things refers to the growing population of everyday objects, including cars, infusion pumps, pacemakers, thermostats, and industrial actuators, that contain sensors, processors, and network connections. Security researcher Bruce Schneier argues that this shift is dangerous precisely because it gives computers the ability to affect the physical world directly [6]. Where a software bug in a spreadsheet corrupts data, the same class of bug in a connected vehicle or a networked medical device can injure or kill. For the patient whose infusion pump accepts a tampered update, or the driver whose vehicle trusts a forged command, this is not an abstract data breach but an immediate threat to human safety. In Schneier's framing, the defining risk of the twenty-first century internet is that familiar software vulnerabilities now translate into catastrophic physical failures in systems people depend on for safety.

This challenge sits at the intersection of the BBC themes of health and humanity and the future of the internet, and it is urgent because IoT deployment is both large and safety-critical. In healthcare alone, patient wearables and networked instruments increasingly share data over wireless links in open environments, which exposes them to a wide range of security threats and makes a high standard of secure communication a prerequisite for safe clinical use [1]. General reviews of IoT report that the threat surface has expanded faster than defences, and that traditional authentication and access control mechanisms frequently fail in resource-constrained IoT settings [2]. The consequences are not hypothetical: in smart grid cyber-physical systems, unauthorized access and false data injection can move directly from the software layer to physical outcomes such as equipment damage and regional blackouts [4].

The current status of the field is therefore paradoxical. Adoption is accelerating across healthcare, energy, and industrial automation, yet the security literature remains dominated by taxonomy and survey work that maps where attacks occur without validating the controls that would stop them [1], [2]. This report addresses that gap by proposing and evaluating a concrete, firmware level defence.

1.2 Background: The Implicit Trust Problem

The underlying weakness that connects these failures is a design assumption rather than a single defect. Conventional IoT devices operate on an implicit trust model: once a device authenticates when it joins a network, or once its firmware has completed boot, the rest of the system treats it as trustworthy for the remainder of its operation. Nothing continuously checks whether the code that is running is the code that was authorized, and nothing reverifies a device after it has been admitted. This is the assumption that allows a compromised node to impersonate a legitimate one, inject valid-looking but fraudulent data, or execute tampered firmware without detection [2], [4].

Software security engineering responds to this with the zero trust principle of never trust, always verify, which replaces one-time, perimeter based trust with continuous verification and least privilege access. Applying this principle at the firmware level depends on three building blocks that recur throughout the recent literature. A root of trust is an immutable, hardware anchored cryptographic key that cannot be altered by software. Secure boot uses that root of trust to verify the cryptographic signature of each stage of code before it is allowed to execute, so unauthorized firmware never runs. Attestation is the

process by which a device proves, on demand and at runtime, that its software is in a known good state [5]. Together these mechanisms reframe integrity, authenticity, and availability as engineered properties of the device rather than assumptions about the network.

2 Zero Trust IoT Firmware and Secure Boot

2.1 What the Technology Is

Zero trust IoT firmware is an architecture that enforces the zero trust principle at the lowest software layer of a device. Rather than trusting firmware because it is already installed, the device treats every image, every reboot, and every update as untrusted until it has been cryptographically verified. The foundation is a chain of trust: an immutable hardware root of trust verifies the digital signature of the bootloader; the verified bootloader in turn verifies the signature of the firmware; and only firmware whose signature matches an authorized key is permitted to execute. This staged verification is what the literature calls secure boot, and it ensures that tampered or unauthorized code never reaches execution [5].

Figure 1 illustrates the complete architecture as three connected parts. The verification chain is secure boot itself: starting from the hardware root of trust, each stage checks the next, so the root of trust verifies the bootloader, the verified bootloader verifies the firmware, and only firmware carrying an authorized signature is allowed to run. The attestation loop shows why verification does not stop at boot. Because a device can be compromised after it has started, whether through a runtime exploit or physical interference, the architecture adds continuous runtime attestation, which in plain terms means the device is made to prove at regular intervals that it is still running only trusted code, reanchoring trust to the hardware root rather than trusting the device indefinitely. The update path completes the picture: the same root of trust governs new firmware, so an over the air update is accepted only if its signature and version manifest verify, which prevents both malicious firmware injection and rollback to a known-vulnerable version.

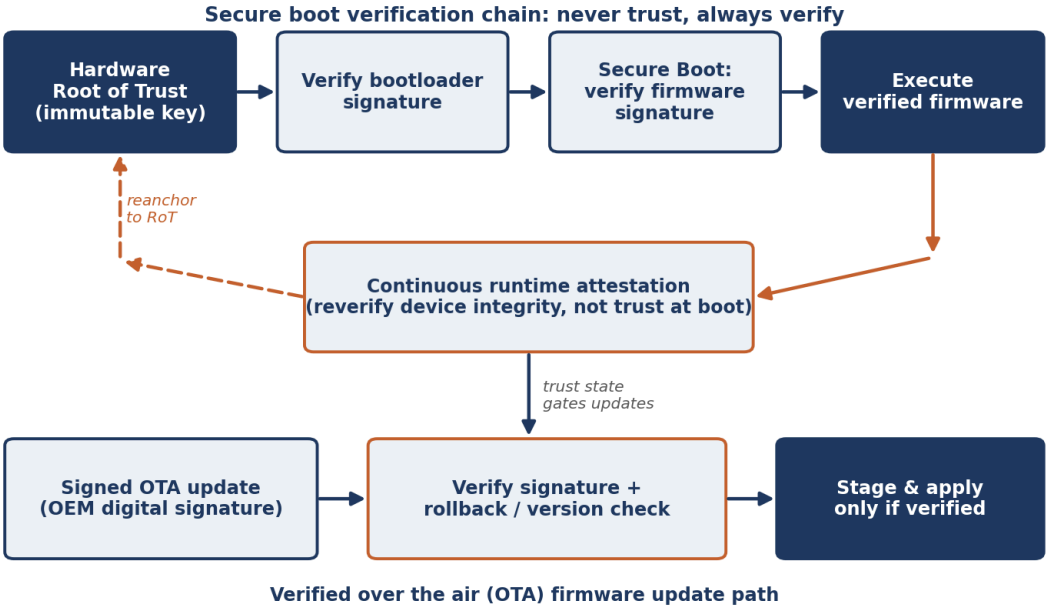

Figure 1. Zero trust secure boot verification chain, continuous runtime attestation, and the verified over the air update path. Trust is reestablished from the hardware root of trust rather than assumed after boot. Diagram compiled from the secure boot, attestation, and firmware update mechanisms described in [5] and [4].

The cryptographic primitives that make this possible are well established. Firmware integrity is confirmed with cryptographic hashes such as SHA-256 and digital signatures such as ECDSA or RSA; secure boot is implemented with hardware features such as ARM TrustZone; and a persistent root of trust can be maintained with certificates, physically unclonable functions, and short-lived tokens so that only authenticated firmware remains active across reboots and update cycles [5]. Elliptic-curve cryptography with timestamps and nonces provides the same guarantees for low-power components while limiting computational cost [4].

2.2 Current Use and Development

The most direct evidence that this technology is practical, rather than aspirational, comes from a recent catalog of techniques for designing over the air update systems for IoT [5]. That study consolidated thirty-four techniques into six mechanisms and validated the catalog in a controlled industrial experiment with ten engineers.

Its first mechanism, secure and safe updates, assembles precisely the building blocks of zero trust firmware. These include firmware integrity and authenticity verification, secure boot before execution, encrypted payloads packaged in signed formats and delivered over secure transport protocols such as TLS, DTLS, or OSCORE, trusted-execution isolation during installation, and a persistent root of trust that resists post-installation tampering [5].

To close the remaining gaps, update requests are themselves authenticated and validated, which defeats replay attacks, unauthorized firmware injection, and misuse of debug ports [5].

Parallel work in other domains shows the same direction of travel. A conceptual security framework for smart grid cyber-physical systems proposes device level cryptographic verification, combining a lightweight, privacy-preserving, quantum-resistant protocol with machine-learning anomaly detection, so that no grid node is implicitly trusted [4]. A comprehensive healthcare IoT survey, after narrowing the literature from 2015 to 2023 to 243 core studies, identifies robust device to device and edge node authentication as the key open requirement for preventing life-threatening failures [1]. Table 1 maps the layered attack surface documented across these surveys to the specific zero trust and secure boot controls that address each layer.

Table 1. IoT threat layers mapped to zero trust and secure boot controls. Attacks compiled from [1] and [2]; controls from [5].

IoT layer	Representative attacks [1], [2]	Zero trust / secure boot control [5]
Perception / device (physical)	Physical tampering, node capture, unauthorized firmware injection, firmware modification	Hardware root of trust; secure boot verifies firmware signature before execution; trusted execution environments and physically unclonable functions
Network	Eavesdropping, man-in-the-middle, replay, Sybil, jamming, distributed denial of service	Mutual authentication with short-lived credentials; encrypted transport (TLS / DTLS / OSCORE); signed, sequence-checked messages
Application / update	Malicious or unauthorized over the air updates, rollback to vulnerable versions, service impersonation	Signed updates with manifest-driven rollback and version control; continuous

IoT layer	Representative attacks [1], [2]	Zero trust / secure boot control [5]
		runtime attestation; least privilege, role-based update access

2.3 Strengths

The central strength of zero trust firmware is that it attacks the root cause rather than a symptom. By verifying integrity and authenticity at execution and reverifying them continuously at runtime, it removes the implicit trust assumption that converts a software vulnerability into a physical failure. Its further advantages are concrete:

- **Mature, standardized primitives.** The verification depends on established components such as SHA-256 hashing, ECDSA and RSA signatures, and ARM TrustZone secure boot, all of which have already been validated in an industrial engineering setting [5], so the approach does not require unproven cryptography.
- **Composability with availability defences.** Zero trust firmware complements, rather than competes with, the reliability techniques that safety-critical systems already use. A multi-protocol redundant architecture achieves 100% packet delivery by transmitting simultaneously over Serial, Wi-Fi, and Bluetooth Low Energy with a dual Wi-Fi and 5G backhaul [3]. However, redundancy guarantees only that a message arrives; it cannot determine whether the sending device is trustworthy. Zero trust firmware supplies exactly that missing guarantee, so the two layers together deliver messages that are both received and trusted.
- **Resistance to post-boot and persistent attacks.** Continuous attestation and a persistent root of trust ensure that only authenticated firmware remains active across reboots and update cycles, providing resistance to post-installation tampering that a one-time boot check cannot offer [5].

Figure 2 reinforces why the availability layer alone is an inadequate defence. Even when every individual protocol delivers 100% of packets in isolation, tail latency varies dramatically: at the 500 Hz clinical sampling rate, Wi-Fi's 99th-percentile latency reaches 129.3 ms against a 42.5 ms average [3]. A defence that focuses only on getting data through says nothing about whether that data is authentic, which underscores the need for the integrity guarantees that zero trust firmware provides.

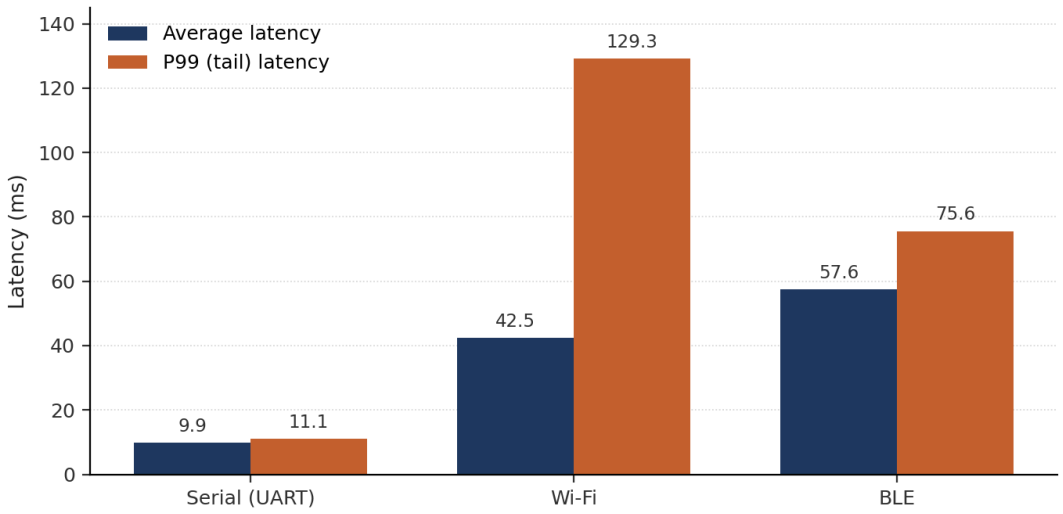

Figure 2. Average versus 99th-percentile (tail) latency for single-protocol transmission at the 500 Hz clinical sampling rate. Data from Doe and Herglotz [3].

2.4 Weaknesses and Limitations

Zero trust firmware is not without cost, and an honest proposal must acknowledge its limitations:

- **Compute and energy overhead.** Continuous cryptographic verification competes for the same scarce resources that constrain IoT devices. General IoT reviews note that traditional authentication and public-key infrastructure often fail in resource-constrained environments [2], and the smart grid framework concedes that quantum-resistant and hash-graph methods are computationally demanding, offering no overhead benchmarks on constrained nodes [4]. Because wireless transmission already draws 570–656 mW at higher sampling rates [3], per-message verification must be implemented with lightweight elliptic-curve cryptography to remain viable.
- **Hardware dependency.** Secure boot requires an immutable hardware root of trust. This adds to the bill of materials and cannot be retrofitted to the large installed base of legacy devices, limiting the near-term reach of the approach to new hardware.
- **Evidence maturity.** The supporting evidence validates the building blocks in pieces rather than as a single integrated stack at scale. The over the air catalog was tested with only ten engineers from one organization [5]; the smart grid framework remains conceptual and unbenchmarked [4]; and the redundancy results were obtained in a laboratory rather than a busy hospital [3]. The primitives are sound, but a full field validation of zero trust firmware as an integrated system is still outstanding.
- **A moving cryptographic target.** The migration toward post-quantum cryptography means that today's signature schemes will need to be replaced, so a zero trust firmware design must be crypto-agile from the outset [4].

3 Discussion

Taken together, the evidence supports a clear claim: zero trust IoT firmware, anchored by secure boot, is an appropriate and feasible response to the challenge Schneier identifies. The justification follows directly from the literature. The healthcare, general IoT, and smart grid surveys agree that the common enabler of physical IoT failures is implicit trust, that is, devices that are believed rather than verified [1], [2], [4]. The over the air techniques catalog demonstrates that the cryptographic controls needed to remove that assumption already exist and can be applied by working engineers [5]. The safety-critical communication study shows that availability defences, however strong, cannot by themselves establish trust [3]. A control that verifies firmware before it acts, and reverifies it while it runs, therefore addresses the precise mechanism by which a software bug becomes a physical catastrophe.

The proposal also carries consequences that a responsible engineer must weigh. The limitations set out in Section 2.4, chiefly the compute and energy overhead of continuous verification and the hardware root of trust it depends on, are joined at deployment by organizational obligations that Section 2.4 does not address, namely secure key management, an original-equipment-manufacturer signing infrastructure, and role-based control over who may authorize updates [5].

What matters for the proposal is how these costs compare with the benefit. They are real trade-offs rather than reasons to reject the solution, because each is bounded by design while the risk it offsets, a software vulnerability acting directly on the physical world, is not.

Lightweight elliptic-curve cryptography and hardware trusted-execution environments hold the overhead to a fixed budget on constrained devices [4], the hardware cost falls only on new devices that can be built with a root of trust from the start, and the key-management burden mirrors practices already familiar to organizations that ship signed software. Set against the catastrophic physical failures the architecture is designed to prevent, these are manageable engineering costs rather than obstacles, which is why the balance favours adoption.

Two objections deserve a direct answer. First, one might argue that existing defences, namely network redundancy, encryption, and attack taxonomies, are sufficient. They are not, because none of them verifies the integrity of the code a device is executing; redundancy ensures delivery [3] and taxonomies map threats [1], [2], but neither prevents a trusted by default device from acting maliciously. Zero trust firmware composes with these defences rather than replacing them.

Second, one might argue that continuous verification is too expensive for constrained hardware. This is a genuine limitation, but it is a matter of engineering the cryptography to the device budget rather than a flaw in the principle, and elliptic-curve schemes designed for low-power components show that the budget can be met [4]. On balance, the approach converts the safety of cyber-physical IoT from an assumption into a property that is continuously and verifiably enforced.

4 Conclusion

The Internet of Things has given software direct authority over the physical world, and with it the power to cause catastrophic physical failures when that software is compromised, which is the challenge that Bruce Schneier places at the centre of twenty-first century internet security [6]. This report has argued that the failures documented across healthcare IoT [1], general IoT [2], safety-critical communication [3], and smart grid cyber-physical systems [4] share a single root cause: an implicit trust model that treats devices as trustworthy once they are admitted or booted.

Zero trust IoT firmware, anchored by secure boot, addresses that root cause directly. By verifying each firmware image against a hardware root of trust before execution, reverifying device integrity continuously through runtime attestation, and accepting only signed, rollback-protected updates, the architecture ensures that only authenticated code ever acts on the physical world [5]. Its primitives are mature and validated, it composes cleanly with the availability defences that safety-critical systems already deploy, and its principal costs, namely compute and energy overhead, a hardware root-of-trust requirement, and a post-quantum migration path, are manageable through lightweight cryptography and hardware trusted-execution support. As a draft proposal, this report recommends zero trust firmware as a feasible and necessary foundation for trustworthy cyber-physical IoT, and it invites peer feedback to sharpen the argument and the treatment of the technology's trade-offs in the revised final report.

References

[1] M. Adil et al., “Healthcare Internet of Things: Security Threats, Challenges, and Future Research Directions,” *IEEE Internet of Things Journal*, vol. 11, no. 11, pp. 19046–19069, Jun. 2024.

[2] M. Aqeel, F. Ali, M. W. Iqbal, T. A. Rana, M. Arif, and M. R. Auwul, “A Review of Security and Privacy Concerns in the Internet of Things (IoT),” *Journal of Sensors*, vol. 2022, pp. 1–20, Sep. 2022.

[3] N. P. Doe and C. Herglotz, “Multi-Protocol Redundancy and Adaptive Sampling for Safety-Critical IoT Communication Using Clinical ECG Monitoring as Case Study,” in *Proc. 19th Int. Joint Conf. Biomedical Engineering Systems and Technologies (BIOSTEC 2026), Vol. 1: BIOSIGNALS*, 2026, pp. 588–597.

[4] M. K. Hasan et al., “Smart grid cyber-physical systems: Components, vulnerabilities, mitigations, opportunities, and conceptual framework,” *Energy Reports*, vol. 15, art. no. 109255, 2026.

[5] M. M. Villegas, M. Solar, F. D. Giraldo, and H. Astudillo, “DeOTA-IoT: A Techniques Catalog for Designing Over the Air (OTA) Update Systems for IoT,” *Sensors*, vol. 26, no. 1, art. no. 193, 2026.

[6] “50 grand challenges for the 21st Century,” *BBC Future*, Mar. 31, 2017. [Online]. Available: https://www.bbc.com/future/article/20170331-50-grand-challenges-for-the-21st-century

Appendix A: Revision Summary

This report was revised from the Phase 2 draft after blind peer review through PeerScholar, in which two classmates scored the draft against the course rubric. Both agreed that the argument is well organized, ties every section back to the challenge, technology, and discipline set out in the introduction, and is well supported by peer reviewed evidence. One reviewer liked that the report introduces the problem before the solution, said Section 1.2's explanation of devices being trusted after boot made the security issue easy to understand, and noted that the report connects that idea forward to secure boot and runtime attestation so the sections flow together. Both praised Table 1 and the two figures as useful and properly referenced before they appear. Their main suggestions were to explain Figure 1 in more depth, to break up some of the longer, term dense paragraphs in Sections 2.2 and 3, to define terms such as runtime attestation in simpler language on first use, and, from a formatting standpoint, to give the table of contents, list of figures, and list of tables each their own page as required by the Technical Writing Handout. The edits below address that feedback.

At the proofreading level, I identified the report for the non blind final submission by adding the author name and student number to the title page and letter of transmittal, removed every em dash, and reduced discretionary hyphenated compounds. I confirmed the IEEE reference list italicizes all journal and conference titles and applies the first author et al. convention to the two sources with more than six authors, [1] and [4]. Following the formatting note both reviewers raised, I split the table of contents, list of figures, and list of tables back onto three separate pages, which shifted the page numbers throughout and required the table of contents to be corrected again. At the sentence level, I strengthened the audience centered framing in the introduction by naming the human stakes of a tampered medical or vehicular update directly. At the paragraph and flow level, I split the longest paragraphs in Sections 2.2 and 3 so each covers a narrower point.

The two reviews pointed in a similar direction rather than a contradictory one, so reconciling them mainly meant deciding how far to go without diluting the technical content. On checking Figure 1 against the suggestion, I found the surrounding paragraph already walks through its three parts, the verification chain, the attestation loop, and the update path, in the order a reader would trace them on the diagram, and that runtime attestation is already glossed in plain language where it first appears in Section 1.2, so I left both passages as is rather than add length that would repeat what the text already does. I was more conservative with splitting every long paragraph in Sections 2.2 and 3, since some of that density comes from citing multiple sources for a single claim, so I only split paragraphs carrying two distinct ideas, one in Section 2.2 and one in the Discussion, and left single, well supported claims intact. The core argument in Sections 2 and 3, both figures, and Table 1 were retained without substantive change, because the reviewers rated the use of evidence highly and each claim is directly supported by a cited source. Compared with the first draft, the final report is properly identified, presents accurate front matter with each list on its own page, and is more readable while preserving the technical substance peer review validated.

AI Use Disclosure Statement

In this preparation of Phase 1 for Term my source review process, I used AI algorithms to create short and customized summaries of different articles in order to properly filter out the amount of available academic material. This made it possible for me to quickly assess their applicability to my focus on software security engineering and the prevention of catastrophic physical failures in Internet of Things devices. I carefully limited my Google Scholar queries before reaching out to the AI to make sure the first pool of materials satisfied the assignment's fundamental requirements, concentrating especially on peer-reviewed articles released in the last five years. However, as an added layer of verification, I provided the AI with these exact assignment requirements to double check my selections, ensuring that no outdated or unscholarly materials bypassed my initial screening.

Additionally, time management was challenging due to the length and density of the source material. To efficiently deal with this, I fed the AI the larger texts and asked it to recognize and highlight key passages that would be most useful for my specific research requirements. Instead of getting weighed down in unimportant details, this focused approach enabled me to concentrate my in depth reading and critical analysis on the most important parts of the texts. I personally checked all of the AI's section recommendations against the original texts to verify contextual accuracy, even though the AI helped find and summarize data. All essential revisions and critical assessments were developed totally on my own, and the final review, reflection, and summary contained in the annotated bibliography were carried out independently.

Prompts Used

Prompt 1: Used for double checking sources

Act as an expert academic research assistant evaluating sources for my software security engineering term project, which focuses on mitigating catastrophic physical failures in IoT systems. For every article title, publication detail, and abstract/text I provide, complete three tasks in a single response: first, verify constraints by confirming if it is peer reviewed and published within the last 5 years, strictly flagging any failures; second, provide a concise, fluff free technical summary of its core methodology, findings, and conclusions; and third, evaluate its relevance by explaining exactly how it addresses the intersection of software security and physical/hardware failure prevention in IoT environments.

Prompt 2: Used for optimizing selection process

Act as an expert engineering research assistant to pinpoint specific, important sections of the provided technically dense academic text that directly serve my focus on Software Security Engineering, specifically securing the physical world and mitigating catastrophic physical failures in IoT systems through Zero Trust IoT Firmware and Secure Boot protocols. Instead of summarizing the entire paper, provide a targeted breakdown of the most relevant segments highlighting their methodologies and critical findings, include the exact section titles or headings to direct my reading, and briefly explain why each identified section is directly applicable to IoT software security and physical failure mitigation.