"We just went with what seemed the best supported": Lessons on Cryptographic Practices in Securing Microcontrollers from Embedded CTFs

Zheyuan Ma
CactiLab, University at Buffalo
Buffalo, NY, USA
zheyuanm@buffalo.edu

Xi Tan
SUNRISE Lab, University of Colorado Colorado Springs
Colorado Springs, CO, USA
xtan4@uccs.edu

Gaoxiang Liu
CactiLab, University at Buffalo
Buffalo, NY, USA
gliu25@buffalo.edu

Alex Eastman
CactiLab, University at Buffalo
Buffalo, NY, USA
aeastman12@proton.me

Md Armanuzzaman
T3SLab, University of Texas at El Paso
El Paso, TX, USA
marmanuzzaman@utep.edu

Ziming Zhao
CactiLab, Northeastern University
Boston, MA, USA
z.zhao@northeastern.edu

Abstract

Microcontroller-based embedded systems support safety- and security-critical applications yet provide insufficient resources for modern cryptography. While numerous lightweight primitives and libraries have been proposed, little is known about how practitioners select and deploy them. We bridge this gap with a mixed-methods study of the 2023 and 2024 MITRE Embedded Capture-the-Flag (eCTF) competitions. We review 47 working firmware submissions and hold 22 semi-structured interviews with team members.

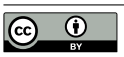
Our analysis reveals two classes of findings. 1) General lessons echo broader usability and human-factors challenges observed in other domains: Developers favor speed and integration convenience over stronger security guarantees, and unclear documentation poses greater barriers to implementing cryptographic functionality than raw resource limits. 2) Embedded-specific lessons highlight unique challenges in resource-constrained environments: Side-channel hardening receives disproportionate attention compared to protecting data at rest, platform heterogeneity complicates portable privacy-preserving implementations, and limited debugging and verification support make it difficult to validate security properties at runtime. We group these findings into *implementation practices*, *participant insights*, and *practical challenges* while giving concrete advice for researchers, educators, device vendors, and the open-source community. Our results show how cryptography is applied in today's security-oriented embedded projects and outline near-term steps to make these systems safer and easier to build.

CCS Concepts

• **Computer systems organization** → *Embedded systems.*

Keywords

Cryptography, Embedded Systems Security, Capture the Flag (CTF)



CCS '26, The Hague, Netherlands

ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3832681

ACM Reference Format:
Zheyuan Ma, Xi Tan, Gaoxiang Liu, Alex Eastman, Md Armanuzzaman, and Ziming Zhao. 2026. "We just went with what seemed the best supported": Lessons on Cryptographic Practices in Securing Microcontrollers from Embedded CTFs. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3832681

1 Introduction

The embedded systems family comprises both *microprocessor-* and *microcontroller-based* platforms, each characterized by distinct architectures and hardware capabilities. Microprocessor-based embedded systems, though more constrained than general-purpose computers, often retain advanced features such as Memory Management Units (MMUs), Real-Time Clocks (RTCs), and Linux-based OSs. In contrast, microcontroller (MCU) systems typically operate on bare-metal firmware or Real-Time Operating Systems (RTOSs) with minimal processing power, memory, and peripherals. Because they frequently operate unattended for long periods and without adequate physical safeguards, these devices remain exposed to attacks that can endanger system safety, disrupt functionality, and leak private data [67]. Deploying cryptographic mechanisms in MCU systems, therefore, demands algorithms that are both computationally efficient and resilient against physical compromise [66]. These design pressures force developers to navigate difficult trade-offs between efficiency, usability, and the strength of privacy protections that their systems can realistically achieve.

Although numerous lightweight cryptographic primitives and libraries tailored for constrained environments exist [49, 60], they are typically designed for microprocessor-based embedded systems, assuming advanced features that are not always present in MCU environments, such as heap memory allocation for runtime memory management, a file system for secret storage, and reliable entropy sources for randomness. Bridging this gap requires additional developer effort and configuration, leading to inconsistent experiences and challenges in adoption on microcontroller platforms. Therefore, it is important to study and categorize these challenges in

microcontroller-based systems, the most resource-constrained subset of embedded devices, where lessons often generalize to the broader embedded family.

Prior studies primarily survey microcontroller hardware features and static firmware flaws [54, 76, 80] or analyze the comparative performance of lightweight algorithms [47]. However, much less is understood about how practitioners work with microcontroller platforms select primitives, integrate them into resource-constrained devices, and evaluate their implementations in practice. Without this understanding, it is difficult to assess whether today's tools, libraries, and research efforts adequately address the practical needs of efficient, secure, and privacy-preserving cryptographic deployment on microcontrollers, or where they fall short.

To bridge this gap, we conduct an in-depth analysis of the 2023 and 2024 MITRE Embedded Capture-the-Flag (eCTF) competitions. This annual, months-long competition involves teams designing, implementing, and attacking MCU systems intended to be secure. The competition unfolds in two phases: teams first build systems to meet predefined functionality and security requirements in the design phase, and then attempt to compromise the designs of others in the attack phase. Cryptography is central to this process, requiring participants to select primitives, adapt algorithms or libraries, and ensure correct operation under resource constraints, while also defending against attacks. Because the robustness and usability of these implementations directly affect both system security and data privacy, cryptographic challenges often dominate the competition, mirroring their central role in real-world embedded development. The analysis of MITRE eCTF competitions helps us answer the following research questions.

- *Q1 - How do participants design and apply cryptographic mechanisms to security-oriented embedded projects?* This question explores the central role of cryptography in real-world embedded development and how participant choices affect privacy and security outcomes.
- *Q2 - What factors influence participants' selection, adaptation, and integration of cryptographic primitives and libraries on MCU systems?* Answering this question helps uncover the practical and cognitive considerations, such as performance, familiarity, and perceived risk, that shape cryptographic design strategies.
- *Q3 - What challenges do participants encounter when implementing, testing, and maintaining cryptographic functionality under resource constraints?* This question identifies the technical, usability, and environmental barriers that hinder secure and privacy-preserving cryptographic deployment in MCU systems.

To answer the first question (Q1), we analyzed 47 firmware submissions that passed the organizers' functionality tests. Our analysis focuses specifically on the cryptographic implementations within these submissions, emphasizing the use of cryptographic primitives and algorithms. These foundational building blocks provide insights into the teams' adaptation and implementation challenges. However, technical analysis alone cannot explain the human factors driving these design decisions or the difficulties developers faced in practice, which are key aspects of the second and third research questions (Q2 and Q3). We further conducted 22 semi-structured interviews with eCTF participants. The interviews explored participants' cryptographic knowledge, the motivations behind their decisions, and the practical obstacles they faced throughout development and testing.

Our findings, which collectively address the three research questions, are organized into three overarching categories: *implementation practices*, *participant insights*, and *practical challenges*. The *implementation practices* reveal consistent patterns in how teams approached cryptography, which answer Q1. Most teams rely on well-established suites that compile with minimal modification. This pragmatic emphasis on performance and ease of integration influences nearly every design decision observed. The *participant insights* explain the motivations behind the observed patterns and answer Q2. When selecting cryptographic primitives, teams prefer algorithms and libraries that are fast, lightweight, and straightforward to integrate, while giving less priority to theoretical optimality or novel designs. Participants often viewed cryptography as a component to be made functional rather than an area for innovation, leading them to reuse proven algorithms and adopt conservative implementation strategies. Finally, the *practical challenges* highlight systemic barriers to secure cryptographic deployment, which answer Q3. Participants struggle more with limited tooling, inconsistent or incomplete documentation, and board-specific idiosyncrasies than with absolute performance or memory constraints. Such issues hinder porting, disrupt builds, and reduce the time available for comprehensive testing. Consequently, systematic evaluations, such as validating fault tolerance, are often rare and ad hoc, leaving potential vulnerabilities unresolved.

The contributions of this paper are as follows:

- We conduct a detailed code-level analysis of 47 validated designs from the 2023 and 2024 MITRE eCTF competitions, characterizing the cryptographic primitives, libraries, and implementation patterns.
- We augment this code-level analysis with 22 in-depth participant interviews, synthesizing their experiences into a unified taxonomy that captures developer insights and distinct classes of challenges faced when porting cryptography onto resource-constrained embedded platforms.
- We translate these empirical findings into actionable guidance, outlining best-practice recommendations for different stakeholders that can immediately strengthen the usability and privacy assurance of cryptography in embedded systems while informing future research directions.

2 Background and Threat Model

In this section, we provide cryptographic background for embedded systems and outline the threat model.

2.1 Embedded Cryptography

Embedded platforms must satisfy the same core security goals as larger systems—confidentiality, integrity, authenticity (CIA)—while operating under far tighter code—size, RAM, and energy budgets. These constraints have motivated decades of research on *lightweight cryptography*: primitives, protocols, and libraries that retain conventional security margins while fitting within a few kilobytes of memory and completing within milliwatts of power [10, 13, 75]. We will explain the common choices and trade-offs for each major class of cryptographic operation on microcontrollers.

Symmetric Operations. Early embedded systems reused desktop ciphers like AES-128, but table lookups and large S-box logic can increase memory footprint and exacerbate side-channel leakage [65]. This motivated "ultra-low-area" designs based on simple permutations or ARX (Add-Rotate-XOR) networks [26]. In 2023, NIST concluded a five-year competition, selecting *Ascon* as the first U.S. lightweight-crypto standard [56]. Stream-cipher-based Authenticated Encryption with Associated Data (AEAD) schemes such as ChaCha20-Poly1305 offer an alternative that maps efficiently to small registers and resists timing attacks [8]. Yet even these require careful, constant-time coding to survive power analysis.

Asymmetric Operations. RSA remains widespread in middleware libraries, yet a 1024-bit modular exponentiation can exceed the cycle budget of entry-level ARM Cortex-M0 MCUs. Elliptic-curve cryptography (ECC) offers equivalent security with far smaller keys, and curves with chosen arithmetic (e.g., Curve25519/Ed25519 [73]) have been optimized to run in a few milliseconds even on 8-bit AVR parts [24]. Comparative studies show ECC outperforms RSA by orders of magnitude in energy per handshake [46]. Open-source libraries such as micro-ecc [45] and nano-ecc [35] achieve footprints under 20 kB, making public-key cryptography practical for firmware updates, key agreement, and digital signatures on single-chip devices. Still, side-channel hardening is mandatory.

Hash Functions. Microcontroller firmware commonly needs fast message digests for secure boot and for deriving keys from limited entropy. Many deployments use the SHA family, with SHA-256 as a common baseline due to broad support and a conservative security margin, but its 64-round compression and working state can impose noticeable cycle cost and RAM pressure on MCUs. The BLAKE2 family was explicitly tuned for short messages and small working sets. BLAKE2s, in particular, targets 8- to 32-bit CPUs and has become a practical choice in embedded TLS stacks [6]. Ascon-Hash, standardized alongside Ascon-AEAD, further reduces RAM by processing only 8-byte lanes [56].

Password Hashing. Many MCU applications (e.g., locks, fobs, sensor hubs) store local authentication material such as PIN verifiers, pairing secrets, or cryptographic keys. Memory-hard schemes like scrypt [62] and Argon2 [61] can be difficult to deploy on RAM-limited MCUs. In contrast, bcrypt [64] mainly increases computation time (rather than memory usage), so it is often easier to run on MCUs. Comparative studies recommend selecting the slowest parameter set that fits the target MCU's RAM and latency budget [3].

Entropy Sources. Robust cryptography presumes a good source of entropy, yet many microcontrollers ship without a true hardware RNG. Developers often fall back to uninitialized SRAM or oscillator jitter, neither of which provides provable entropy [34]. Research shows that relying on a single weak source can leave keys guessable after power-on [32]. NIST SP 800-90B recommends mixing multiple noisy sources and hashing them before seeding a DRBG [74]; implementing that guidance remains non-trivial on devices without file systems or hardware counters.

2.2 MITRE eCTF Competition and Threat Model

The eCTF competition is an annual semester-long event in which teams design, implement, and attack embedded software for a specified MCU platform. In the 2023 competition, teams emulated a vehicle manufacturer that develops firmware for cars and key fobs equipped with remote keyless entry systems. In 2024, the challenge focused on securing an insulin-pump system composed of three interconnected microcontroller boards sharing an I^2C bus. The details of the MITRE eCTF competition and its cryptographic usage scenarios are presented in the supplemental materials [44].

The eCTF competition adopts the strongest realistic assumptions for microcontroller deployments. An adversary has access to the complete *source code* with embedded secrets and flags removed, unfettered *physical* possession of every board, complete *visibility* into all on-board buses and external links, and *sufficient time* to probe, modify, and clone hardware. In effect, every adversary works in a white-box setting. Thus, static analysis can reveal protocol logic, key schedules, RNG seeding, and error paths. In cryptographic terms, those assumptions extend the capabilities of the classical attacker in several important ways.

Side-channel observation. The adversary can mount power-analysis, electromagnetic, or timing attacks on symmetric and asymmetric operations. Even constant-time software on a desktop may leak on an 8- or 32-bit MCU if table look-ups correlate with Hamming weight [17, 42]; similarly, an unbalanced scalar multiplication loop can reveal halves of an ECC private key within minutes of power-trace collection [27].

Fault injection and glitching. Clock, voltage, laser, or electromagnetic glitches allow attackers to skip instructions, flip result bits, or force peripheral registers to unsafe states. Typical goals include bypassing signature checks in a secure-boot routine, forcing an RSA implementation to emit faulty ciphertext for key recovery, or rolling back a monotonic nonce counter to enable replay [11, 20].

Bus eavesdropping and manipulation. Unencrypted I^2C, SPI, UART, or CAN traffic can be recorded and replayed. Message counters, MACs, or AEAD tags are therefore required not merely for remote adversaries but for anyone able to attach logic probes to board headers.

Key-storage attacks. MCUs seldom feature a secure enclave or MMU. Secrets share the same address space as application code, so a single memory-safety error can expose long-term keys. This presents MCUs with a unique attack surface, which is absent, or far harder to reach, on general-purpose computers protected by file-system permissions and tamper-evident enclosures.

Entropy degradation. Hardware random-number generators, when present, can be slowed or biased by controlling voltage or temperature, while software PRNGs can be reproduced if an attacker resets the device to a known seed state.

3 Research Methodology

In this section, we describe the research methodology employed in our study, which includes a comprehensive analysis of team submissions from the eCTF competitions, semi-structured interviews with participants, and threats to validity.

Ethical considerations. To ensure compliance with ethical standards, we submitted our study protocol to the Institutional Review Board (IRB) at our institution. The study adhered to guidelines ensuring informed consent, submission analysis, the right of participants to withdraw at any time, and the anonymization of Personally Identifiable Information (PII) to maintain privacy and

confidentiality. Our IRB approved the study under an IRB exemption. In addition, we obtained MITRE's written approval to allow us to analyze the submitted materials. More ethical considerations are discussed in the supplementary materials [44].

3.1 Submission Analysis

We conducted a detailed analysis of 47 unique team submissions that advanced to the attack phase in the 2023 and 2024 competitions. The authors' team submissions were included in the statistical analysis for completeness. However, to avoid potential bias, no examples or case studies in the paper originate from the authors' own submissions unless explicitly stated. Each submission included source code, documentation, build instructions, and supplementary materials such as posters and presentations. These artifacts were redistributed to all attack-phase participants and released under the terms of the MITRE Participant Agreement [78], which authorizes their use for research purposes.

Our analysis focused on identifying the cryptographic primitives and libraries used in each submission, including whether teams embedded library source code or relied on precompiled components. Beyond static inspection, we compiled and debugged the firmware to confirm that the identified cryptographic components were exercised at runtime. Three study members independently reviewed each submission and resolved discrepancies through discussion. We present the detailed criteria and results of the analysis in §4.

3.2 Interviews with Participants

We conducted semi-structured interviews with 22 participants, comprising undergraduate, Master's, and Ph.D. students. Of these participants, 15 reported medium or higher prior cryptographic experience, indicating at least a basic understanding of cryptographic concepts. Additionally, 14 participants had some experience working with embedded systems. Supplemental materials [44] summarize the demographic details of the participants.

Participant recruitment. We conducted two rounds of participant recruitment to gather interviewees. Both rounds targeted individuals who participated in the attack phase of the eCTF competitions at least once between 2021 and 2024. The first recruitment round took place during the 2024 eCTF award ceremony in April, where we recruited the 10 participants. The second round occurred in December through direct email outreach (81 participants) and competition channels with MITRE-supported advertising (reaching over 200 past competitors), resulting in 12 additional participants. In total, we got responses from 25 individuals, of whom 22 completed the interviews. We were able to recruit only one participant from before 2023 (P13), who led teams from 2018 to 2021 and served as an advisor in 2022 and 2023. After identifying prospective participants, we provided detailed study information, including a consent information sheet outlining the purpose of the study, its voluntary nature, and the steps taken to maintain data confidentiality.

Interview procedure. The interviews were conducted remotely via Zoom from May to June 2024 for the first round and from January to March 2025 for the second round. They lasted with an average duration of 25 minutes with the crypto-related questions. Prior to each session, participants were asked to complete an online demographic survey form (supplemental materials [44]), which gathered information about their academic background, prior experience with cryptography and embedded systems, and their specific roles and contributions during the eCTF competition. The interviews focused on questions related to the usage and implementation of cryptographic algorithms, their experiences with library support, and challenges specific to embedded systems. The full list of interview questions is provided in the supplemental materials [44]. Each participant received a $50 gift card as compensation.

Data collection. Interviews were recorded through Zoom, and the audio recordings were transcribed using Otter AI [58]. We contacted all participants after the interview to inform them of this third-party service usage and affirm its privacy practices [59] comply with our IRB protocol with proper data de-identification. To ensure the accuracy of the transcripts, all study team members cross-reviewed and proofread each transcription against the original recordings, addressing any discrepancies to maintain the semantic integrity of the responses.

Data analysis. Data analysis started with creating an initial coding scheme based on established qualitative and thematic methods [16, 69]. Three study members independently coded a single transcript to generate a diverse set of initial codes. These codes were then discussed collaboratively, resulting in a refined codebook that categorized the challenges participants faced, the strategies they employed, and the insights they provided, following an inductive, reflexive workflow. All three coders are Ph.D.-level embedded-security researchers with prior eCTF experience. Using this codebook, all three coders independently coded each subsequent transcript before any discussion. After each round of coding, they met to resolve discrepancies, iteratively refine the codebook, and proceed to the next transcript. Weekly discussions were held to ensure consistency and consensus throughout the process. This iterative process negated the need for formal inter-rater reliability measures, as coding decisions were made collectively [50]. The final analysis created a codebook with 6 themes, 32 subthemes, and 109 codes.

3.3 Stakeholders

The findings of this study are relevant to multiple stakeholders. *Researchers* can identify open problems in lightweight cryptography and microcontroller security. *Educators* can use the insights to design training that better reflects the practical challenges of embedded development. *Device vendors* gain guidance on integrating cryptographic protections into resource-constrained products. *The open-source community* can refine libraries and tooling to reduce integration burdens for developers.

3.4 Threat to Validity

While the MITRE eCTF competition offers a realistic and structured environment for examining cryptographic development on embedded systems, it remains a simulated competition environment. All participants were university students. As a result, the findings may not fully generalize to professional or industry contexts, where priorities, experience levels, and development workflows differ. Nevertheless, the eCTF reproduces many aspects of real-world embedded development, including strict functionality requirements, constrained hardware, and adversarial evaluation, which lend ecological realism to our observations.

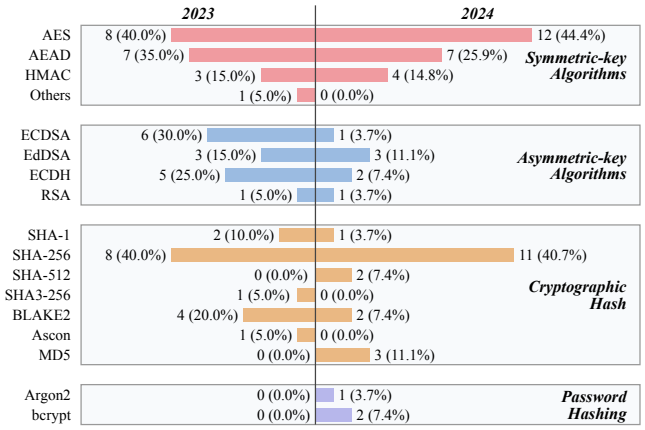


Figure 1: The cryptographic algorithms used in different teams in both years.

The study is also limited by the relatively small sample size of interview participants. While this is common in qualitative research, we do not claim that thematic saturation was formally reached, and the findings should be interpreted as capturing a range of perspectives rather than an exhaustive set. Our recruitment aimed to maximize diversity by reaching a broad pool of past participants; however, participation was voluntary, and the final sample reflects those who responded and completed the interviews. Due to this limited sample size, we do not draw conclusions about subgroup differences (e.g., participants with prior cryptography experience versus CTF-only experience) or treat developer experience as an independent analytical dimension. Our analysis focuses on technical and integration challenges in embedded cryptographic engineering.

Additionally, the interviews relied on self-reported data, introducing the potential for social desirability bias, where participants may present themselves in a favorable light. Participants also reflected on events that occurred one to two years earlier, introducing potential recall bias. While efforts were made to design comprehensive and unbiased questions, the interviews may not have captured all aspects of cryptographic implementation challenges.

Furthermore, the submission analysis focused on identifying the cryptographic primitives and their use in the teams' designs. This approach may not fully capture large modifications made to primitives or cases where teams used different components within libraries to achieve similar functionality. Similarly, the categorization of insights and challenges may overlap or extend beyond the defined categories due to the complexity and interconnected nature of cryptographic implementation issues. To mitigate these limitations, observations from the submission analysis were triangulated with documentation and interview findings to verify consistency.

4 Implementation Practices

Figure 1 visually summarizes the usage of cryptographic primitives. We have additional tables in the supplemental materials [44] to present the cryptographic primitives for each year. We focused on cryptographic implementations within each submission, including symmetric-key algorithms, asymmetric-key algorithms, and cryptographic hash functions. We concentrated on the lowest-level functions available in selected libraries, as these best reflected the adaptation and implementation challenges faced by teams. Because the participating teams differ between years, we counted the number of teams that used each algorithm.

While the 2023 and 2024 competitions differ in application theme, they share a similar physical threat model and rely on comparable MCU-based platforms. Both settings require teams to deploy cryptographic mechanisms to ensure CIA under resource constraints. As a result, we expect these differences in application context to have limited influence on high-level cryptographic implementation patterns, though some variation may arise from year-specific protocol requirements and system interactions.

Symmetric-key Algorithms and Schemes. The AES was the most commonly used symmetric-key algorithm in both years, with 8 out of 20 teams choosing it in 2023, and 12 out of 27 teams in 2024. Despite its popularity, a notable number of teams (11 out of 20) used the basic electronic codebook (ECB) mode of AES, which is generally discouraged due to its vulnerability to pattern leakage when identical plaintext blocks are encrypted into identical ciphertext blocks [51]. AEAD was the second most popular scheme, implemented by 7 teams each year. Teams typically chose AEAD schemes such as XChaCha20-Poly1305 and Ascon. The use of Hash-based Message Authentication Code (HMAC) remains consistent each year, with 3 and 4, respectively.

Asymmetric-key Algorithms and Schemes. In contrast to symmetric algorithms, asymmetric-key algorithms were less commonly used, with 11 teams in 2023 and only 6 teams in 2024 adopting them. ECC was the most frequently used asymmetric primitive. Teams used ECC-based schemes such as the Elliptic Curve Digital Signature Algorithm (ECDSA), Edwards-Curve Digital Signature Algorithm (EdDSA), and Elliptic-curve Diffie-Hellman (ECDH) for secure key exchange and digital signatures. 2 teams used public-key protocols typical of certificate-based authentication, including RSA (Team A4) and a full SSL/TLS stack (Team B12).

Cryptographic Hash Functions. Cryptographic hash functions were widely used in both years. 10 and 17 teams in 2023 and 2024 used the SHA family, while others used alternatives such as BLAKE2 and Ascon. In 2024, 3 teams used MD5. However, prior work has demonstrated practical collision attacks and MD5 is widely discouraged for security-critical use [14].

Password Hashing Schemes. Additionally, 3 teams incorporated password hashing schemes, such as Argon2 [9] and bcrypt [64]. These schemes are specifically designed to securely hash passwords, which are intentionally designed to be slow and resource-intensive, thereby mitigating the risk of brute-force attacks. However, in MCU settings, teams configured these schemes with reduced iteration counts and/or lower memory parameters to fit available resources.

Open-Source Software. In 2023, teams chose from a range of open-source cryptographic libraries, as the default library provided with the reference design, tiny-AES [40], offered only AES encryption support. In 2024, 17 out of 27 teams opted to use the default cryptographic library, wolfSSL [60], which included a comprehensive set of cryptographic primitives suited to embedded systems. However, as discussed in Section 6, many teams encountered challenges with wolfSSL's complexity, which required substantial effort to integrate effectively into their designs.

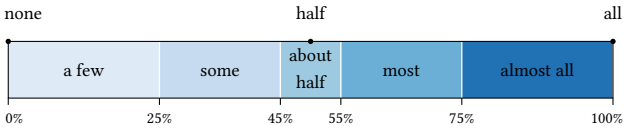


Figure 2: Terminology used to quantify participant counts in our study.

Observation 1: Teams predominantly used well-established primitives such as AES, HMAC, and SHA-256, often relying on default modes and configurations provided by libraries. Asymmetric cryptography was less common and was typically ECC-based, while password hashing schemes appeared only rarely.

5 Participant Insights

In this section, we present a qualitative analysis of the interviews with the 22 eCTF participants. The findings provide insight into participants' design choices, including algorithm selection, decision factors, and strategies for mitigating cryptographic attacks. To avoid overly precise quantitative interpretations and confusion between participant and team counts, we use an alternative quantifier scale [25] to report participant frequencies, as shown in Figure 2.

5.1 Selection of Cryptographic Algorithms

Participants chose their cryptographic algorithms due to a combination of factors, including performance efficiency, simplicity, widespread acknowledgment, and security features. Additionally, they tend not to prioritize the secure storage of sensitive data when time is limited.

5.1.1 Performance. **When selecting a cryptographic algorithm, speed was the primary consideration and the most influential factor.** Most participants discussed how the performance efficiency of a crypto algorithm influenced their design.

For symmetric-key algorithms, about half of the participants favored AES, and none of them reported performance issues with AES. Participant P13's team primarily used ChaCha20 with Salsa for symmetric encryption over the years, as it was *"extremely efficient"* and at some point even *"faster than AES and some other things"* through their internal benchmarks.

In terms of asymmetric-key algorithms, participant P7 chose Curve25519 for EdDSA because it is efficient in signature verification and is generally recommended for performance, especially for operations like Diffie-Hellman and digital signatures. In addition, participant P3 told us that they selected algorithms and implementations mainly based on performance efficiency, which led to the selection of the Salted Challenge Response Authentication Mechanism (SCRAM) for mutual authentication:

> *"The mutual authentication step ... the primary motivating concern was performance. Namely, that asymmetric public key crypto algorithms take a lot longer to run than symmetric key algorithms. So I looked for and found the SCRAM protocol, which was a mutual authentication protocol that only required hashes."*

Some participants preferred ECC over traditional RSA. P9 noted that ECC *"is simply faster and requires smaller key sizes"* compared to RSA, while P12 mentioned that, while using smaller key sizes, ECC has an equivalent security strength to AES with larger key sizes. P12 also highlighted that ECC provided a 30 or 40 percent reduction in key size, which has a huge impact on performance.

5.1.2 Simplicity. **The simplicity of the algorithm and the easy adaptation process were the major influential factors, in addition to the speed.** Some participants explicitly favor it.

P2 told us that besides efficiency, the ease of integration within the system and availability in existing libraries were other factors when choosing the algorithm: *"Did wolfcrypt [wolfSSL] have it? Because we already have wolfcrypt ... that's what we've done instead of like creating our own, for example."*

Similarly, P3's team opted for the best-supported digital signature algorithm available to avoid unnecessary complexity in adaptation, as the algorithms *"are all about the same"* in theory. P3 explained that they chose what seemed to be the best-supported algorithm, and there would be no point in adding extra complexity. P5's team noted that they decided not to use asymmetric encryption in their design, as *"it was making the code a lot more complex in terms of the crypto part."*

Participant P19 emphasized that the audibility and minimalism were key criteria in their cryptographic library selection, as they aimed to *"be auditing as much as we can"* on their software stack. And since the wolfSSL library was comparably large and complex, they decided to use the more auditable Monocypher library instead. In contrast, P9's team focused more on the integration aspect of the cryptographic library. Their team was looking for a cryptographic library that they could *"rapidly compile tested, and then actually working with our design."*

P13 mentioned that since the ChaCha20 algorithm supports the AEAD scheme, it allowed their team to include additional metadata, such as transaction IDs, directly into the encryption process. This feature simplified their protocol design and reduced the need for separate integrity mechanisms such as the HMAC:

> *"You don't need to think too much about ... two separate things where you encrypt and then you create your own HMAC or integrity check on top of that."*

5.1.3 Symmetric vs. Asymmetric Cryptography. **Participants had mixed opinions on the necessity of asymmetric cryptography in embedded systems.**

P19's team chose symmetric cryptography over asymmetric methods for simplicity and because symmetric primitives fulfilled all their security requirements. They thought the symmetric cryptography was *"remarkably easier"* than asymmetric methods, and thus they *"didn't see any benefit"* in using the latter for their design. Participant P11 strongly favored symmetric cryptography, particularly AES, for embedded systems due to its efficiency and performance advantages over asymmetric algorithms. They questioned the need for asymmetric encryption given that the competition's thread model can securely distribute the keys by stating, *"if you have a block cipher primitive, why do you need an asymmetric one?"* P11 emphasized that cryptographic choices should be based

on necessity and rigorous justification rather than novelty or perceived sophistication. P11 also warned that using a cryptographic algorithm without a mathematical proof of security could lead to vulnerabilities, no matter how *"cool"* the algorithm might seem.

On the other hand, P13 recognized that symmetric algorithms are *"obviously faster"* but highlighted that their use introduces risks when key material is compromised, as all communicating devices share the same secret. They noted that asymmetric algorithms can offer *"particular different security guarantees"* in cases where device compromise or impersonation is part of the threat model. Similarly, P1 acknowledged that while symmetric encryption is optimal for embedded systems, asymmetric crypto is still necessary for key validation and other use cases. As a result, several teams decided to utilize an asymmetric key exchange algorithm for generating the symmetric keys and then employ symmetric encryption to secure the communication. P2 explained the advantage of such a design:

> *"First, write symmetric in the end was just more effective, quicker communication, so it's easier to hit the timing goals ... If the symmetric key gets discovered for some reason, maybe it doesn't work, because you have to recreate the keys."*

5.1.4 Other Factors. **Participants relied on research literature, course and competition experiences, or academic expertise to determine the cryptographic algorithm to employ.**

Participant P1 told us that in deciding to use Ascon [22], their team reviewed the existing research literature and could not find any published attacks that could be sufficient against the hardware. Thus, they justified that Ascon was secure enough for their use.

Participant P2 mentioned that their choice of ChaCha20-Poly1305 was supported by its historical use in stream cipher contexts and the credibility of the designer, who designed both ChaCha20 and the elliptic curve Curve25519. Participant P1 also noted that their decision to use ASCON was influenced by the previous years of the competition, as they observed that top teams had been using it.

P5 credited one of their teammates for suggesting the use of SHA3, who *"did a cryptography course"* and was *"a bit more in touch with crypto"* at the time:

> *"So he suggested us to use SHA3, since we don't have any timing attacks existing on SHA3. So then, ... we used HMAC SHA3 to have the [key hashing] part."*

Additionally, their Key Derivation Function (KDF) design, which derives new keys by hashing the previous keys, was introduced based on a suggestion from their professor that specializes in hardware security. They also noted the inspiration from existing well-known applications, as they were suggested that *"WhatsApp and Signal [also] use key ratcheting algorithm."*

P11's team had designed a custom three-party key exchange protocol named "Mask-On" in 2024, which was to *"make sure the session key is fresh every single time."* They explained that the design was derived from the participant's own research work and had been proved *"mathematically secure."*

The widespread acknowledgment of a cryptographic algorithm contributes to its selection as well. Participant P2 told us that in choosing their cryptographic algorithms, *"we just went with a package that other people do,"* referring to the ChaCha20-Poly1305, ECC, and ephemeral Diffie-Hellman. They believed that the widespread usage and perceived reliability of these basic algorithms would not add unnecessary complexity to their crypto operations.

On the other hand, P4 had chosen Ascon in 2023 due to its winner standing in NIST lightweight cryptography competition [57] and *"they're explicitly designed to be used for embedded systems"*. Participant P8 admitted that their knowledge of bcrypt and its features was acquired through internet research aiming to identify a hashing function that could provide substantial computational load against potential attacks:

> *"We had basically just looked at the internet to ask it: What was the most complicated [computational intensive] hashing function to implement?"*

Similarly, participant P7 explained their reason of choosing Curve25519 for digital signature, as it was *"recommended online"* and *"generally known as performant, and good enough for implementations of ... signatures and Diffie-Hellman."* Another participant, P8, selected AES for its robustness and *"gold standard"* acceptance as a secure encryption method: *"And everybody told us don't worry about AES like, AES in itself is theoretically safe. It's just how do you use it, that it's going to break it."*

Observation 2: The speed and ease of adoption of the algorithms played the most significant roles in participants' selection of MCU systems. While they have different opinions on the necessity of asymmetric cryptography, the existing research literature, course experience, or academic expertise helped them make the decision.

5.2 Mitigating Crypto-Related Attacks

Crypto-related attacks target the design, implementation, or usage of cryptographic algorithms and protocols to undermine data confidentiality, integrity, or authenticity. The crypto-related attacks that participants were concerned about include side-channel attacks, brute-force attacks, and arbitrary memory access.

5.2.1 Side-Channel Attacks. Side-channel attacks exploit indirect information leakage from a computing device rather than targeting its algorithm or implementation directly. Embedded systems are often more susceptible to side-channel attacks due to their resource constraints, physical accessibility, simpler designs, and higher likelihood of emitting measurable side-channel signals like power consumption or electromagnetic emissions [29]. As a result, cryptographic algorithms or libraries that specifically target embedded systems often include hardening features to mitigate side-channel vulnerabilities. **To protect against side-channel attacks, participants used the built-in hardening features of cryptographic libraries or chose algorithms specifically designed to withstand side-channel vulnerabilities.**

P1 and P18's team used Rust and chose Ascon for their crypto implementation, but they had concerns about the side-channel protections in the existing Rust crate that implements Ascon [31, 68]. As mentioned by P1:

> *"We were looking at those [Rust Ascon] libraries, the source and crate, we didn't see sufficient protections against side channel attacks. So we wanted to use the original Ascon C implementation because that is tested against side-channel attacks."*

Eventually, they took *"a protected implementation of Ascon that uses like masking to avoid power side-channel analysis"* in C and then linked it into their Rust code. P18 admitted that despite the use of a protected implementation, the side-channel analysis in general *"is a little bit more difficult even on a non-protected implementation."*

Participant P5's team was set to look for a crypto algorithm free from existing side-channel attacks, which led to the selection of SHA3. They also adopted KDF to thwart key replay and side-channel attacks by ensuring that the keys are constantly changing and become obsolete after every use. They explained: *"In that case, you avoid ... power and timing side-channel attacks since your keys differ on every interaction, also it's ... very hard to replay the key since your key gets expired every time you use it."*

In terms of the crypto libraries, P5 valued the wolfSSL library's inbuilt hardening options as *"probably people have dedicated much time in hardening that library for microcontrollers,"* which provide an added layer of security without the need for extensive custom development by the team. Participant P4 mentioned that they initially considered AES-GCM during design but ultimately did not use it because *"it's relatively prone to side-channel attacks."* They instead chose EdDSA for its robustness against physical attacks that might leak bits of secret information, and noted that one of the threat models to consider on embedded systems was *"physical-related attacks of like trying to leak bits on the wire."*

In contrast, P2 believed that the security of the crypto algorithm was less of a concern than its efficiency on embedded devices, as fewer people would *"try to break the encryption itself"*. They thought most attacks would originate from *"bad implementation"* instead of the algorithm itself and that the algorithm choice should be based more on efficiency.

Recommendation 1: Cryptographic **library maintainers** should clearly document whether their libraries include side-channel mitigations and under what conditions they are effective.

5.2.2 Brute-Force Attacks. Brute-force attacks involve trying all possible combinations of a password or key until the correct one is found. These attacks are computationally intensive and time-consuming, making them less effective against algorithms that are inherently slow. **Participants utilized the slow nature of the password hashing algorithms to prevent brute-force attacks.**

Participant P4 said that their team explored memory-hard hashing functions like bcrypt for password hashing to prevent brute-force attacks, detailed as *"password hash that essentially makes the hash goes through like multiple iterations,"* which makes the procedure computationally intensive and thus more difficult to brute-force. Similarly, P8's team was also looking for a hashing function that would *"take a substantial amount of time or as much time as we possibly could"* to prevent brute-force attacks. They initially considered using SHA512 and other complex hash functions while "*"hashing twice or making it more convoluted than it used to be,"* but eventually they chose bcrypt via online research.

5.2.3 Arbitrary Memory Access. Arbitrary memory access attacks involve unauthorized access to sensitive data stored in a device's memory, including flash memory, SRAM, or EEPROM, which can expose cryptographic keys or other confidential information.

A few participants assumed attackers would have full access to the memory and designed their systems accordingly. Participant P19's team implemented a "Russian-doll-style" protocol, *"where we had blobs that were encrypted with keys that were distributed on different components"* to ensure that all the necessary components in the system need to be present at the same time to decrypt the data. This design was intended to be resilient even if the full memory was compromised: *"If it's just a bunch of random data, you really can't do anything, so even if there's a bug."* Their approach deliberately avoided relying on memory protection features, instead designing the protocol to be resilient even in the presence of software or logic bugs:

> *"Designing protocols where you assume nothing, and ... as long as the crypto is good and achieves its purpose ... then you can make a lot of assumptions about the security design, such as the entire memory address space is readable."*

About half of the participants chose to store the sensitive data directly in the header files without encryption or hashing. The development board utilized in the 2023 eCTF featured a dedicated EEPROM region, distinct from the flash memory address space, suitable for storing sensitive data. Conversely, the board in 2024 included only flash memory for data storage. Despite this, about half of the teams in 2023 opted to embed secrets directly within C header files, resulting in their storage in flash memory. Additionally, our review revealed that many teams, across both competitions, stored secrets in C headers without employing encryption or hashing for protection. Participant P2 admitted that their decision to put plaintext secrets in the header files was a mix of *"lazy and overconfident"* in their security measures. They believed that their design was secure enough to withstand attacks and thus did not prioritize the protection of secrets in memory.

In contrast, P4's team used the EEPROM to store cryptographic secrets in 2023 due to the concern of potential data leaks of the flash memory. Participant P8 thought that *"leaving things in plain text is just a bad idea"* and explained their motivation behind storing secrets in cryptographic secure hashes: *"If it [the hashes] used to be like MD5 or some other truly simple hash to crack, you basically just leave it in plain text. So we picked bcrypt for that."*

Recommendation 2: Developers should assume that data stored in flash memory is potentially accessible to attackers and take appropriate measures to protect it. More importantly, **device vendors** should always provide alternative storage options for sensitive data, such as dedicated EEPROM regions, to help developers protect their secrets.

5.2.4 Security Issues with AES-ECB. AES in ECB mode is considered insecure as it deterministically encrypts identical plaintext blocks into identical ciphertext blocks. This lack of diffusion leaks patterns in the data, making it vulnerable to analysis and revealing structural information about the original content, especially in cases with repetitive data such as images or formatted data structures.

Participants were unaware of the security issues associated with AES-ECB mode. A few participants who utilized the AES-ECB mode in their designs lacked awareness of its security

vulnerabilities. For instance, participants P6, P15, and P16 demonstrated gaps in their understanding of AES modes, which impacted their choice of cryptographic solutions. P16 admitted:

> *"But to my knowledge, I didn't know that the ECB algorithm has this inherent vulnerability in them, like you can break the encryption."*

However, we also observed that one participant, P11, was aware of the security issues associated with AES-ECB mode and still chose to use it in their design. They acknowledged that *"ECB is the most ... unsafe thing ever"*, but justified their choice based on their specific use case. P11 explained that their application only encrypts a single 16-byte block, which effectively avoids the pattern leakage issue characteristic of ECB mode. In their view, this scenario is *"nothing like it's the same as a one time pad"* and using ECB for a one-block encryption was comparable in effect to a simple XOR operation. Moreover, they had already incorporated additional protections, such as nonces and ephemeral keys, elsewhere in the protocol, which they argued mitigated the core vulnerabilities of ECB.

Recommendation 3: Cryptographic **library maintainers** should provide clear warnings about the security implications of using insecure modes like AES-ECB or disable them by default to prevent inadvertent misuse.

5.3 Longitudinal Perspectives

We further present a qualitative analysis of multi-year participants to explore longitudinal perspectives. In total, we have 6 multi-year participants. The analysis results suggest that repeated participation did not simply lead teams to reuse the same primitives or designs. Instead, participants described adapting their choices based on concrete lessons from prior practices. We discuss those changes in the context of symmetric cryptography, asymmetric cryptography, cryptographic implementation, and security concerns.

5.3.1 Symmetric Cryptography Usage. Some multi-year participants described an evolution **from choosing familiar or default primitives to selecting algorithms that better fit the embedded platform**. For example, participant P2 explained that in both years, AES was their first option because it was the most familiar primitive. However, in 2024, they switched to ChaCha20-Poly1305 because AES-GCM mode did not fit their buffer constraints:

> *"This year. We use ChaCha Polly 1305. And the reason why was because of the buffer size... And then to fit it onto the AES buffer, it seemed too much work. So we were like, Let's use a stream cipher. "*

Similarly, participant P7 noted that they switched to ChaCha20-Poly1305 due to performance and timing issues, as AES-GCM was too slow to meet the strict 3-second boot and test time limits in 2024. P7 further explained that ChaCha20-Poly1305 was useful because it avoided block-cipher padding under I^2C packet-size limits: *"it's a stream cipher, and it's not a block cipher. So we don't need to pad the messages ... messages over I^2C were limited to 255 bytes."*

5.3.2 Asymmetric Cryptography Usage. Half of the multi-year participants changed their use of asymmetric cryptography as **their understanding of the system's security needs evolved**. P4's team moved from a purely symmetric design in 2023 to a design that included asymmetric cryptography in 2024. Participant P4 explained that the 2024 design needed a way to uniquely identify components and prevent one from impersonating another:

> *"we ended up including a digital signature algorithm ... to make sure that the messages weren't tampered with."*

P7's team also described moving from standard ECC choices to Ed25519 and X25519 because these curves better fit their needs: *"ED25519 is faster for signature verification"* and X25519 was used for Diffie-Hellman key exchange.

In contrast, P18's team moved in the opposite direction. While their 2023 design used asymmetric cryptography, their 2024 design relied only on symmetric cryptography. Their re-evaluation suggested that, for the 2024 challenge structure and attack scenarios, the complexity of asymmetric cryptography did not provide enough additional benefits.

5.3.3 Cryptographic Implementation. Participants also shifted from treating cryptography choices as a **primitive-selection problem** to treating it as a **system-integration problem**. In 2024, P2's team moved to the woflcrypt library. They had previously tried wolfSSL but found it to be a *"black box"* that consumed too much memory. Using primitives from wolfcrypt gave them better control:

> *"...the huge problem was the size, like memory that you needed from wolfSSL, amount of communication side, like we didn't have control. So we didn't know if things were working really well, because it was kind of a black box happening. And then we abandon the border."*

Participant P7 similarly described wolfSSL integration as difficult because the library was not only for embedded systems but also provided a full TLS suite for servers. Their team had to determine which functions could run in their environment and which build flags were needed for constraints such as no dynamic memory and no `/dev/random`. At final, P7's team thought *"it would be easier to just write the crypto ourselves and create the handshake."*

Participant P13 reinforces this system-level view of cryptographic implementation choices. Although P13 did not describe a specific year-by-year library migration, he emphasized that implementations had to be efficient, small, and free of unnecessary dependencies: *"we were just choosing get, like, a set of like algorithms which we were experienced with... we run efficient enough, we're small enough... We didn't have dependencies, external dependencies. We didn't need, like, some special like float point, like FPU... And so we were basically choose based on that..."*

5.3.4 Security Concerns. There is **increased attention to physical and implementation-level attacks**. Some multi-year participants explicitly contrasted earlier designs, where they were less careful about side-channel or hardware attacks, with later designs where they chose masked ASCON implementations, avoided certain AES uses, preferred EdDSA over ECDSA, or considered whether hardware accelerators and flash storage could leak secrets.

P1's team chose to create Rust-C bindings for the ASCON C library rather than using a native Rust implementation. This was a strategic move to ensure they were using a version of the algorithm that had been specifically tested against side-channel attacks:

> *"And by doing that, we ensure that you know, we're not using an implementation that is vulnerable to such*

> *[side-channel] attacks. So that was our reasoning for specifically doing a rust-C bindings for ASCON."*

P4 gave another example when discussing their choice of bcrypt for password hashing in 2024. In 2023, P4's team used the ASCON hash, which they later realized was *"too fast"*, making it easier to brute-force: *"With it being lightweight, it's easier to brute force and little bits."* In 2024, they switched to bcrypt, a *"slow hash"* specifically designed to make brute-forcing computationally expensive.

5.3.5 Uneven Learning. At the same time, the longitudinal data also show that learning was uneven. Participants improved their awareness of algorithm choice, protocol design, and implementation risks, but **sensitive-data storage remained a recurring weakness**. Half of the multi-year participants explained their teams' decisions to store secrets in plaintext. Participant P2 showed that the team stored sensitive data in C header files, *"not encrypted or anything. No hash, pure in header."* P2 later characterized this as *"in between lazy and overconfident"* and contrasted the protocol design with the implementation:

> *"We did zero trust on the protocol ... But when we went through the code implementation, we thought it would be fine just to have it in the header."*

Participant P7 described a similar issue. P7's team built a certificate-based system, but left both generated eCTF key files and their own secrets in plaintext header files.

Observation 3: Multi-year experience helped teams reason more carefully about communication security, but did not translate into systematic protection of sensitive data, e.g., stored secrets.

6 Practical Challenges

In this section, we discuss the challenges faced by participants in implementing cryptographic algorithms on embedded systems, and provide best practices to address these challenges. We use the same quantifier scale as in §5, shown in Figure 2.

6.1 Limited Resources

The limited resources presented challenges to developers when implementing cryptographic algorithms on the microcontroller system. Compared to the microprocessors employed in smartphones, tablets, and desktop computers, microcontrollers operate at lower frequencies and possess smaller memory capacities. These limitations present noticeable challenges in implementing computationally and memory-intensive cryptographic algorithms.

Lower operating frequencies of microcontrollers lead to reduced processing speeds, a challenge that participants commonly identified as impacting the implementation of cryptographic algorithms. Participant P7 indicated a switch in encryption protocols from 2023 to 2024: from AES-GCM to ChaCha20-Poly1305 for symmetric encryption, and from traditional ECC curves (secp256r1) to Curve25519 for digital signatures. This decision was made because *"with testing, we were getting speed problems because it was taking multiple seconds."* These delays were exceeding the permitted booting time requirements.

P3's team opted for ECDSA for their public key cryptography but encountered performance challenges. They resolved them by *"wrangling the clock speed on the device"*: *"We had to eventually bump up the clock speed in order to get public key signatures verified quickly enough."* Similarly, to solve the performance issues, P9's team overclocked the device and applied *"some unrolling of the algorithm"* to speed up the cryptographic algorithms: *"It would take up more a bit more memory, but in exchange should run a little faster."* Participant P4 considered Argon2 in design to slow down potential brute-force attacks, utilizing its slow-computing nature to hinder attackers. However, they eventually failed to integrate it: *"But in both years, we weren't able to really meet the timing requirements, with the actual parameters that are suggested by Argon."*

Participants also highlighted difficulties in integrating full SSL modules into embedded systems, attributing these challenges to the limited memory capacities of microcontrollers. The SSL modules in wolfSSL provide an implementation of the SSL/TLS protocol, and the library was originally developed for embedded systems, IoT devices, and memory-constrained devices [60]. Despite its small footprint by design, some participants have complained about having technical issues during the integration process on the microcontroller.

Participant P2 encountered difficulties with wolfSSL due to its memory requirements and the high abstraction of its SSL modules. They eventually decided to implement their cryptographic design with basic primitives instead of using the comprehensive SSL solutions. They said: *"The huge problem was the size, like memory that you needed from wolfSSL ... we [also] didn't know if things were working really well, because it was kind of a black box happening."*

Comparably, participant P7 initially considered leveraging the TLS functionality of wolfSSL to facilitate secure communications between boards, akin to a typical TLS handshake between servers and clients. However, they encountered difficulties adapting the TLS library for the I^2C communication channel, as wolfSSL's TLS implementation is primarily designed for socket-based communications, not I^2C used by the boards. They instead opted to implement cryptographic operations manually: *"... at the end, we thought it would be easier to just write the crypto ourselves and create the handshake."*

Participant P5 highlighted a general perception that TLS might be too resource-intensive for less powerful microcontroller systems: *"One thing which was totally opposite to us ... was usage of TLS on embedded devices. I guess few teams used TLS. And we were actually wondering: it's heavy, right? And [usually] you're using it in a network. [Use in] embedded devices, that was ... opposite of the performance."*

Recommendation 4: Developers must account for the performance and memory constraints of microcontrollers, where full SSL/TLS stacks may be unsuitable, and instead use lightweight libraries or basic cryptographic primitives tailored to their needs.

6.2 Embedded Adaptation

The adaptation of cryptographic libraries and tools to embedded systems often requires significant modifications, as standard implementations may not function effectively within the constrained environments typical of microcontrollers. **Participants frequently encountered challenges adapting the cryptographic libraries to suit their specific hardware and software configurations on the microcontroller, requiring extensive customization and troubleshooting.**

Participant P3 had to make a *"customized version"* of the scram-SHA-256 algorithm for a `no_std` Rust environment [12], such as *"stripped out the unnecessary steps like base64 encoding,"* which added additional complexity to the adaptation process. Participant P4 mentioned that even though the Ascon cipher won the NIST lightweight cryptography competition, they still encountered difficulties with deploying Ascon on the microcontroller: *"We ran into a lot of challenges trying to get that to actually work on the board properly, to actually audit to make sure that it was doing things correctly and ended up missing some things there."*

Participant P8 complained about using bcrypt in their design, as *"the compiler wouldn't play nice"* and *"there was a special way that it needed to be compiled"* on the microcontroller, which affected the implementation progress. Participant P3 also noted complications with hardware AES accelerators due to their complex dependencies within the vendor-supplied libraries:

> *"The way the hardware AES accelerator worked was there was a header that led another header, something like 13 or 14 different linkages, that we had to go and hunt down ... And fishing that out was a nightmare."*

Participant P2 encountered difficulties with I^2C communication, specifically with buffer management during multistep communication processes. They noticed an issue where sending multiple commands in quick succession could lead to I^2C buffer overflow, which affects the integrity of transmitted data: *"... whenever we were sending many commands ... if we didn't pause to make some ACK, it would go crazy ... maybe we were overflowing the buffer without noticing."* P13's team encountered performance issues with some cryptographic libraries that relied on floating-point operations, which were not supported by the target MCU: *"... it was basically emulated in software. It was like super, super slow. We just replace it with something which didn't need to use that."*

Recommendation 5: Library maintainers should ensure cryptographic primitives are modular, easily configurable to exclude non-essential dependencies. They should also proactively validate their libraries against common MCU architectures and popular embedded toolchains to streamline the integration processes.

6.3 Documentation Complexity

The documentation of cryptographic libraries and tools is crucial for developers to understand the implementation details, configuration options, and potential pitfalls. **Participants reported that the documentation was often complex, poorly structured, or lacking in clarity, which presented configuration challenges during implementation.**

Participant P16 found wolfSSL's documentation sparse, unstructured, and difficult to navigate: *"It was so sparse and not specifically crafted ... it looked random to me. For any library like this, they should have some kind of, like structured documentation."* They were not able to find the specific dependency information from the documentation, and eventually had to *"look for open source platform, like, reddit and ask questions there."* Similarly, to make wolfSSL work, P20's team manually included the entire library codebase into their repository, which bloated their code size:

> *"There wasn't a whole lot of documentation teaching us how to use it exactly. So we just tried different things, and eventually we found that dragging the entire thing just works, so we kind of stuck with that."*

Participant P2 experienced difficulties with wolfSSL and could not find the necessary documentation to understand the library's inner workings, which appeared as a "black box": *"They don't have how, for example, the assembly works behind that ... So it becomes kind of a black box again ... if you want to check specifics, it is a bit harsh."* Participant P5 found wolfSSL *"was pretty straightforward"* in implementation but encumbered by poorly structured documentation and excessive configuration flags: *"The flags you need to enable to make it work ... there were so many unwanted flags, which you can define and make sure your code doesn't compile with those."*

Participant P7 noted that due to wolfSSL being designed with capabilities beyond typical embedded system requirements, they *"had a bit of trouble figuring out which parts of it could run in an embedded environment and which parts couldn't."* They also encountered conflicting information within the library documentation, leading to uncertainty about the functionality's suitability for embedded environments:

> *"... the documentation is conflicting in some part. It says that this function could not be used in an embedded environment, but then we included it, and it worked. ... and different parts had different instructions."*

In contrast, participant P4 reported a smoother experience with cryptographic implementations using Rust in 2024, as *"most of the algorithms worked without issue."* They attributed their success to the use of well-vetted libraries: *"We did mostly use the Rust crypto implementations, which are implemented by the Rust Foundation team ... even on an embedded system, we didn't really have too many problems trying to run those properly."*

Recommendation 6: Library maintainers should provide documentation that clearly outlines dependencies, configuration flags relevant to MCU environments, and highlights which library components are optimized or compatible with embedded targets.

6.4 Testing Cryptographic Design

Testing cryptographic designs on embedded systems posed noticeable challenges for participants, primarily due to the absence of sophisticated tools and the complexity of verifying cryptographic functions. This section discusses three primary testing challenges: the lack of effective testing methods known to participants, the strategy of off-board testing before implementation on the device, and the challenges in using the debuggers.

6.4.1 Lack of Effective Testing Methods. **Participants noted a lack of automatic tools for verifying randomness and the correctness of cryptographic implementations, complicating their testing processes.**

Participant P2 emphasized the need for automatic tools to ensure the correctness of cryptographic implementations, as they mostly *"just read the documentation [and] see if it's sending plaintext or not"* when testing with the libraries. P16 admitted that their primary technical vulnerability in the design, which was an overflow caused

by concatenating two large nonces for HMAC, was due to the lack of systematic debugging or verification infrastructure:

> *"The main reason we were not able to catch that because there were a lack of testing in our work, and we just implemented it and tested whether it's working or not ... We didn't have any specific test setup for the competition at that time."*

Participant P8 employed simple, ad-hoc tests rather than formal unit testing, reflecting a common approach among participants: *"We just ran simple tests ... we basically took the input that we knew ... and we ran it through the embedded system to basically spit it out and check that it was what we were expecting."* P9 mentioned that they conducted testing by *"running the same algorithms in a different library on the PC"* to compare results, and also attempted to attack their own design by *"sending forged messages and see if they would be accepted."*

In contrast, participant P19 developed an automated functional test suite that flashed the boards and verified behavior through a sequence of inputs and expected outputs over UART:

> *"You'd have a JSON file which would describe, like a scenario ... then a sequence of inputs to pass to serial and then expected return value. If those didn't match, this was a failed test case."*

Fuzzing is a common technique for testing software by randomly sending data to the system to observe unexpected behaviors. Participant P2 learned one lesson from the competition, which *"was about redundancy and doing as many things as we can to be paranoid."* They suggested that fuzzing could be a useful technique to test the cryptographic design, and by sending random data to the system, they continued: *"That seems like a good way of going over a broad group of attacks that could be happening, not specific ones."*

However, P1 expressed reluctance to rely on automated fuzzing or code checking tools due to a perceived risk of false negatives and inappropriate confidence levels. They relied on manual debugging and testing instead: *"Honestly, I would not really trust. I feel like automated tooling, will say like, okay, we ran it, we detected nothing. But that kind of gives us false sense of confidence."*

Recommendation 7: Researchers should design lightweight, MCU-compatible frameworks for automating the testing of cryptographic primitives, including unit testing, fuzzing, and conformance checking. These tools should support headless execution and serial/UART interfaces to integrate smoothly with embedded workflows.

6.4.2 Off-Board Testing. **Testing cryptographic primitives often commenced off-board to verify basic functionality and data encryption, ensuring that no plaintext data transmission occurred inadvertently.** This approach allowed participants to refine and debug their cryptographic implementations in a more controlled environment before migrating to embedded systems.

P2 described their team's preliminary off-board testing approach, explaining that they initially focused on *"testing how the primitives worked"* and checking whether any *"plaintext was going through"* before deploying to the actual boards. While this did not guarantee that the cryptographic logic was fully sound, it helped confirm that the output was not trivially interpretable and *"not making gibberish."* Participant P5 described isolating cryptographic functions from I^2C communications by testing off-board to simplify debugging: *"We didn't want to do end-to-end testing in our system, since it's harder to spot bugs."*

P7 developed and tested their handshake implementation in a desktop environment to avoid the overhead of repeated re-flashing during development. They created a standalone repository that ran entirely on Linux, allowing them to implement and verify the full handshake protocol before porting it to the embedded system: *"I got tired of like re-flashing the boards again, and again, just to test small changes ... So I made a separate repository ... just wolfSSL, and the algorithm for the handshake ... it just ran on Linux."*

In contrast, participant P13 performed all testing directly on hardware using a hardware-in-the-loop setup, which allowed them to test the system in a more realistic environment. However, they noted that their efforts in making the automatic testing pipeline were time-consuming and *"super painful"*: *"So it was always hardware in the loop. Even though it probably would have been more efficient to just run in the desktop."*

6.4.3 Using Debuggers. **Participants faced challenges in using the debuggers, making them rely on print statements and ad-hoc testing methods.** About half of the participants reported that they only used print statements to verify the correctness of their implementations, as they either were unaware of the debugging tools or encountered issues with them. The print outputs will be sent to the serial port, which can be monitored using a terminal program on the host computer. As P11 mentioned:

> *"... arbitrary print, like the main way I debug. I'm a pretty rudimentary person ... I use print lines, and... does it break here? Does it break between? ... It dies from between five and six ... let me look at this block of code. What is it doing? That's the way I debug."*

Participant P6 had challenges with the debugger as *"the interface was not very friendly."* They instead relied on basic validation techniques, such as inserting many print statements, to verify the correctness of their implementation: *"When it printed the encrypted stuff, we assumed that was validly encrypted, but no other checks to make sure it was actually encrypted correctly, other than making sure we can decrypt it and get the same message that we had sent."*

P16 used a manual testing approach by flashing the code to the microcontroller and monitoring device communication with a logic analyzer. Although they attempted to set up a debugger using the Cortex-Debug extension in Visual Studio Code [48], they encountered issues with the debugger not working properly, which led them to focus on *"implementing security mechanisms and testing the transactions of messages using the logic analyzer only."*

Recommendation 8: Developers should adopt a layered approach: use simple, off-board tests to validate algorithm correctness before deploying to constrained hardware, and complement these with automated sanity checks once embedded. **Device vendors** could provide better debugging support for UART-based logging frameworks and semi-hosting alternatives to reduce the reliance the need to rely solely on print statements.

7 Discussion and Future Work

7.1 Discussion

7.1.1 Similarities with General-Purpose Systems. Many challenges in embedded cryptographic implementations are similar to those in general-purpose systems, including microprocessor systems. A key practical challenge is the integration of libraries, which often suffers from limited documentation, dependency conflicts, and unclear behaviors. Weaknesses in key management are another critical vulnerability across all systems. Additionally, reliance on informal testing practices, which favor manual checks over automated verification, reveals systemic issues in cryptographic validation. Finally, the trade-off between performance and security is not unique to embedded systems. Developers on general-purpose systems face the challenge of balancing computational overhead with necessary security guarantees when implementing cryptographic techniques.

7.1.2 Embedded-System-Specific Challenges. Distinct constraints in embedded environments amplify specific risks. The limitations of memory and processing power have led development teams to prioritize lightweight algorithms such as ASCON and ChaCha20-Poly1305, whereas general-purpose systems commonly do not have concerns for padding or buffer size. Embedded systems also face hardware constraints, such as dependence on AES accelerators or the absence of floating-point units, and integration challenges that are not typically encountered in microprocessors. The timing constraints imposed by real-time operations further complicate the implementation of asymmetric cryptography, necessitating optimizations such as overclocking or algorithm unrolling. Moreover, physical attack vectors, including side-channel and fault-injection attacks, are considerably more pertinent to microcontrollers than to general-purpose systems, necessitating specialized countermeasures such as masking techniques and ephemeral-key strategies. Lastly, the lack of advanced debugging and profiling tools tailored for embedded systems results in a reliance on UART outputs and logic analyzers, in stark contrast to the more extensive instrumentation available for general-purpose systems.

7.2 Future Work

Future research should enhance the cryptographic robustness and implementation practices in embedded systems. Secure key management and storage mechanisms should replace plain-text storage. It's crucial to implement defenses against side-channel and fault-injection attacks, including masked implementations and lightweight countermeasures for microcontrollers. Automated verification tools for protocol correctness and randomness quality can reduce reliance on manual debugging. Additionally, more precise documentation for cryptographic libraries can help mitigate integration challenges with solutions like wolfSSL. Exploring performance-security trade-offs is essential, particularly for hybrid approaches that use hardware accelerators with efficient primitives such as ChaCha20-Poly1305. Lastly, protocol-level innovations like key ratcheting and formally verified AEAD-based designs can enhance resilience against replay attacks while maintaining efficiency in resource-constrained environments.

8 Related Work

Evolution of cryptography for resource-constrained systems. Lightweight symmetric cryptography has advanced rapidly, with NIST's initial public draft [56] selecting the Ascon family as the first AEAD and hash standard for constrained devices. On the asymmetric side, post-quantum cryptography has become viable on MCUs after years of optimization in the NIST PQC standardization process [2, 55], as demonstrated by the pqm4 framework [36], which measures Kyber-512 key exchange in under 20 ms on a Cortex-M4 with 640 kB RAM, and by recent optimizations that halve Falcon-512 verification time on the same core [15, 63]. Academic research has also emphasized the practical barriers to deployment, with Aumasson et al. [7] detailing hurdles such as limited entropy sources, stateless protocols, and the high integration cost of new primitives, and Manifavas et al. [47] benchmarking lightweight ciphers to guide designers toward appropriate device choices.

While these works highlight algorithmic properties and industrial constraints, our study focuses on the developer experience, showing that most eCTF teams still default to AES because mature libraries and well-documented build recipes outweigh the potential benefits of newer or lighter primitives.

User studies of cryptography usage. Acar et al. [1] asked 53 developers to encrypt a file using five Python crypto APIs, and only nine produced semantically secure code due to confusing defaults, sparse examples, and opaque error messages. Gorski et al. [30] build on and are motivated by findings from Acar. Mindermann et al. [53] report a separate Java-based controlled experiment. Both confirmed that API ergonomics rather than cryptography theory drive correctness, foreshadowing our finding that eCTF teams gravitate toward well-documented AES implementations when time is scarce, even when lighter or stronger primitives are available. Qualitative studies report similar obstacles. Huaman et al. [33] interviewed 20 engineers implementing IETF and NIST standards and found that contradictory RFCs, outdated reference code, and uncertainty about constant-time execution routinely derail projects. Krause et al. [41] interviewed 21 senior developers maintaining cryptographic code in long-lived products and identified legacy interfaces, unclear entropy sources, and fear of breaking deployments as common blockers. Fischer et al. [28] interviewed 18 academic and industrial experts and highlighted socio-technical barriers such as limited funding for hardened implementations and the absence of reproducible test suites.

Our mixed-methods eCTF study extends this line of work by combining artifact mining with developer interviews, providing the first empirical link between the primitives teams claim to prefer, the code they actually submit, their side-channel priorities, and the vendor-specific constraints of resource-constrained firmware.

CTF competitions for education and security analysis. CTF competitions were initially studied mainly as teaching tools: long-running case studies (e.g., UCSB's iCTF [79]) document sustained engagement and improved learning outcomes. More recent studies regard CTF telemetry as an empirical lens on security practice. Savin et al. [70] collect and label low-level attack/defense traces to examine operator behavior in real time. Ma et al. [43] mine firmware submissions and interviews from the 2023-2024 MITRE eCTF to uncover broader microcontroller-security challenges. It

focuses on broader MCU-security challenges or general crypto-API usability, while our work delves into the end-to-end practice of cryptographic engineering on microcontrollers.

Our study builds on this approach but focuses exclusively on cryptographic engineering choices, enabling us to connect player decisions to algorithm footprints, library maturity, and hardware constraints—dimensions largely overlooked in prior CTF studies.

Other hardening mechanisms on microcontrollers. Since cryptography cannot prevent all types of attacks, MCU-specific defenses have emerged. Privilege separation and compartmentalization [18, 19, 21, 37–39, 52, 82] enforce more fine-grained access control. Shadow stack [23, 81], return address integrity [4], and control-flow violation detection [77] are implemented to ensure control flow integrity. Randomization techniques diversify code layout or execution timing to increase resistance against fault-injection and side-channel exploits [71, 72]. Remote-attestation frameworks, exemplified by ENOLA [5], provide post-deployment verification of a device's control flow. Nevertheless, recent surveys [76] find that such defenses rarely appear in production firmware, indicating compiler friction and linker-script complexity—the same tooling and documentation hurdles reported by our interviewees.

9 Conclusion

In this paper, we presented an empirical view of how security-oriented microcontroller design incorporates cryptography by developers in practice. By combining a code-level study of 47 team submissions with 22 interviews from the MITRE eCTF competitions, we identified the cryptographic primitives, libraries, and design choices that developers actually make, and the reasons behind them. We found that speed and library fit drive algorithm choice, side-channel hardening often ranks above secure data storage, and gaps in tooling, documentation, and test support hinder robust deployment. By framing these observations as actionable insights and challenges, we provide a roadmap for researchers, educators, device vendors, and the open-source community to streamline integration, improve testing, and raise overall security on resource-constrained embedded platforms.

Acknowledgments

This material is based upon work supported in part by the National Science Foundation (NSF) under grants 2523436, 2512972, 2508320, 2453496, and by an Amazon Research Award. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of the United States Government, any of its agencies, or Amazon.

This paper was edited for grammar using Grammarly and ChatGPT. All technical content, claims, and final editorial decisions were made and verified by the authors.

References

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2017. Comparing the usability of cryptographic apis. In *IEEE Symposium on Security and Privacy (S&P)*.

[2] Gorjan Alagic, Maxime Bros, Pierre Ciadoux, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, et al. 2025. Status report on the fourth round of the nist post-quantum cryptography standardization process. *National Institute of Standards and Technology* (2025).

[3] Aishah Alfrhan, Tarek Moulahi, and Abdulatif Alabdulatif. 2021. Comparative study on hash functions for lightweight blockchain in Internet of Things (IoT). *Blockchain: Research and Applications* (2021).

[4] Naif Saleh Almakhdhub, Abraham A Clements, Saurabh Bagchi, and Mathias Payer. 2020. μRAI: Securing Embedded Systems with Return Address Integrity. In *Network and Distributed Systems Security (NDSS) Symposium*.

[5] Md Armanuzzaman, Engin Kirda, and Ziming Zhao. 2026. ENOLA: Linear-Space and Low-Overhead Control-Flow Attestation for Microcontroller-based Systems. In *International Conference on Embedded Software (EMSOFT)*.

[6] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. BLAKE2: simpler, smaller, fast as MD5. In *International Conference on Applied Cryptography and Network Security*. Springer.

[7] Jean-Philippe Aumasson, Antony Vennard, and AG Teserakt. 2019. Cryptography in industrial embedded systems: our experience of needs and constraints. In *NIST Lightweight Cryptography Workshop*.

[8] Daniel J Bernstein et al. 2008. ChaCha, a variant of Salsa20. In *Workshop record of SASC*. Citeseer.

[9] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. 2016. Argon2: new generation of memory-hard functions for password hashing and other applications. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.

[10] Alex Biryukov and Leo Perrin. 2017. State of the art in lightweight symmetric cryptography. *Cryptology ePrint Archive* (2017).

[11] Dan Boneh, Richard A DeMillo, and Richard J Lipton. 2001. On the importance of eliminating errors in cryptographic computations. *Journal of cryptology* (2001).

[12] The Embedded Rust Book. 2025. A no_std Rust Environment. https://docs.rust-embedded.org/book/intro/no-std.html.

[13] William J Buchanan, Shancang Li, and Rameez Asif. 2017. Lightweight cryptography methods. *Journal of Cyber Security Technology* (2017).

[14] Chad R Dougherty. 2008. MD5 vulnerable to collision attacks. https://www.kb.cert.org/vuls/id/836068.

[15] Junhyeok Choi, SeungYong Yoon, and Seog Chung Seo. 2025. Optimized Falcon Verify on Cortex-M4 for Post-Quantum secure UAV communications. *ICT Express* (2025).

[16] Victoria Clarke and Virginia Braun. 2014. Thematic analysis. In *Encyclopedia of critical psychology*. Springer.

[17] Christophe Clavier, Damien Marion, and Antoine Wurcker. 2014. Simple power analysis on AES key expansion revisited. In *Cryptographic Hardware and Embedded Systems–CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings 16*. Springer.

[18] Abraham A Clements, Naif Saleh Almakhdhub, Saurabh Bagchi, and Mathias Payer. 2018. ACES: Automatic compartments for embedded systems. In *USENIX Security Symposium*.

[19] Abraham A Clements, Naif Saleh Almakhdhub, Khaled S Saab, Prashast Srivastava, Jinkyu Koo, Saurabh Bagchi, and Mathias Payer. 2017. Protecting bare-metal embedded systems with privilege overlays. In *IEEE Symposium on Security and Privacy (S&P)*.

[20] Ang Cui and Rick Housley. 2017. BADFET: Defeating modern secure boot using Second-Order pulsed electromagnetic fault injection. In *USENIX Workshop on Offensive Technologies (WOOT)*.

[21] Daniel Danner, Rainer Müller, Wolfgang Schröder-Preikschat, Wanja Hofer, and Daniel Lohmann. 2014. Safer Sloth: Efficient, hardware-tailored memory protection. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*.

[22] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. 2021. Ascon v1. 2: Lightweight authenticated encryption and hashing. *Journal of Cryptology* (2021).

[23] Yufei Du, Zhuojia Shen, Komail Dharsee, Jie Zhou, Robert J Walls, and John Criswell. 2022. Holistic Control-Flow protection on Real-Time embedded systems with kage. In *USENIX Security Symposium*.

[24] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, and Peter Schwabe. 2015. High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. *Designs, Codes and Cryptography* (2015).

[25] Pardis Emami-Naeini, Henry Dixon, Yuvraj Agarwal, and Lorrie Faith Cranor. 2019. Exploring how privacy and security factor into IoT device purchase behavior. In *CHI Conference on Human Factors in Computing Systems*.

[26] Nur Fasihah Mohd Esa, Shekh Faisal Abdul-Latip, and Mohd Rizuan Baharon. 2019. A Survey of ARX-based Symmetric-key Primitives. *International Journal of Communication Networks and Information Security* (2019).

[27] Junfeng Fan and Ingrid Verbauwhede. 2012. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. *Cryptography and Security: From Theory to Applications: Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday* (2012).

[28] Konstantin Fischer, Ivana Trummová, Phillip Gajland, Yasemin Acar, Sascha Fahl, and Angela Sasse. 2024. The challenges of bringing cryptography from research papers to products: results from an interview study with experts. In *USENIX Security Symposium*.

[29] Catherine H Gebotys and Catherine H Gebotys. 2010. Side Channel Attacks on the Embedded System. *Security in Embedded Devices* (2010).

[30] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. 2018. Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic API misuse. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS)*.
[31] Hannes Gross, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. 2017. Ascon hardware implementations and side-channel evaluation. *Microprocessors and Microsystems* (2017).
[32] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2012. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*.
[33] Nicolas Huaman, Jacques Suray, Jan H Klemmer, Marcel Fourné, Sabrina Klivan, Ivana Trummová, Yasemin Acar, and Sascha Fahl. 2024. "You have to read 50 different RFCs that contradict each other": An Interview Study on the Experiences of Implementing Cryptographic Standards. In *USENIX Security Symposium*.
[34] James P Hughes and Whitfield Diffie. 2022. The Challenges of IoT, TLS, and Random Number Generators in the Real World: Bad random numbers are still with us and are proliferating in modern systems. *Queue* (2022).
[35] iSEC Partners. 2013. nano-ecc. https://github.com/iSECPartners/nano-ecc.
[36] Matthias J Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. 2019. pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4. *Cryptology ePrint Archive* (2019).
[37] Arslan Khan, Dongyan Xu, and Dave Jing Tian. 2023. Ec: Embedded systems compartmentalization via intra-kernel isolation. In *IEEE Symposium on Security and Privacy (S&P)*.
[38] Arslan Khan, Dongyan Xu, and Dave Jing Tian. 2023. Low-cost privilege separation with compile time compartmentalization for embedded systems. In *IEEE Symposium on Security and Privacy (S&P)*.
[39] Chung Hwan Kim, Taegyu Kim, Hongjun Choi, Zhongshu Gu, Byoungyoung Lee, Xiangyu Zhang, and Dongyan Xu. 2018. Securing Real-Time Microcontroller Systems through Customized Memory View Switching. In *Network and Distributed System Security Symposium (NDSS)*.
[40] kokke. 2012. tiny-AES-c. https://github.com/kokke/tiny-AES-c.
[41] Alexander Krause, Harjot Kaur, Jan H. Klemmer, Oliver Wiese, and Sascha Fahl. 2025. "That's my perspective from 30 years of doing this": An Interview Study on Practices, Experiences, and Challenges of Updating Cryptographic Code. In *USENIX Security Symposium*.
[42] Owen Lo, William J Buchanan, and Douglas Carson. 2017. Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA). *Journal of Cyber Security Technology* (2017).
[43] Zheyuan Ma, Gaoxiang Liu, Alex Eastman, Kai Kaufman, Md Armanuzzaman, Xi Tan, Katherine Jesse, Robert Walls, and Ziming Zhao. 2025. "We just did not have that on the embedded system": Insights and Challenges for Securing Microcontroller Systems from the Embedded CTF Competitions. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.
[44] Zheyuan Ma, Xi Tan, Gaoxiang Liu, Alex Eastman, Md Armanuzzaman, and Ziming Zhao. 2026. Supplemental Materials. https://doi.org/10.1145/3830454.3832681.
[45] Ken MacKay. 2013. micro-ecc. https://github.com/kmackay/micro-ecc.
[46] Dindayal Mahto and Dilip Kumar Yadav. 2017. RSA and ECC: A comparative analysis. *International journal of applied engineering research* (2017).
[47] Charalampos Manifavas, George Hatzivasilis, Konstantinos Fysarakis, and Konstantinos Rantos. 2013. Lightweight cryptography for embedded systems–a comparative analysis. In *International Workshop on Data Privacy Management*. Springer.
[48] marus25. 2023. Cortex-Debug. https://marketplace.visualstudio.com/items?itemName=marus25.cortex-debug.
[49] Mbed-TLS. 2012. mbedtls. https://github.com/Mbed-TLS/mbedtls.
[50] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI practice. *Proceedings of the ACM on human-computer interaction* CSCW (2019).
[51] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. 2018. *Handbook of Applied Cryptography*. CRC Press.
[52] Alejandro Mera, Yi Hui Chen, Ruimin Sun, Engin Kirda, and Long Lu. 2022. D-box: DMA-enabled compartmentalization for embedded applications. *Network and Distributed System Security Symposium (NDSS)* (2022).
[53] Kai Mindermann and Stefan Wagner. 2018. Usability and security effects of code examples on crypto apis. In *IEEE Annual Conference on Privacy, Security and Trust (PST)*.
[54] Nicolas Nino, Ruibo Lu, Wei Zhou, Kyu Hyung Lee, Ziming Zhao, and Le Guan. 2024. Unveiling IoT Security in Reality: A Firmware-Centric Journey. In *USENIX Security Symposium*.
[55] NIST. 2024. Post-Quantum Cryptography Standardization. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization.
[56] The National Institute of Standards and Technology. 2023. Lightweight Cryptography Standardization Process: NIST Selects Ascon. https://csrc.nist.gov/news/2023/lightweight-cryptography-nist-selects-ascon.
[57] The National Institute of Standards and Technology. 2025. Lightweight Cryptography | CSRC. https://csrc.nist.gov/projects/lightweight-cryptography.
[58] Otter.ai. 2025. AI Meeting Note Taker & Real-time AI Transcription. https://otter.ai/.
[59] Otter.ai. 2025. Privacy & Security. https://otter.ai/privacy-security.
[60] Todd Ouska. 2015. wolfSSL Embedded SSL/TLS Library. https://github.com/wolfSSL.
[61] Password Hashing Competition. 2015. Password Hashing Competition. https://www.password-hashing.net/.
[62] Colin Percival and Simon Josefsson. 2016. *The scrypt password-based key derivation function*. Technical Report. Internet Engineering Task Force.
[63] Thomas Pornin. 2025. Falcon on ARM Cortex-M4: an Update. *Cryptology ePrint Archive* (2025).
[64] Niels Provos and David Mazieres. 1999. A future-adaptable password scheme. In *USENIX annual technical conference, FREENIX track*.
[65] Mark Randolph and William Diehl. 2020. Power side-channel attack analysis: A review of 20 years of study for the layman. *Cryptography* (2020).
[66] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. 2004. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)* (2004).
[67] Grand View Research. 2024. Embedded Security Market Size And Share Report, 2030. https://www.grandviewresearch.com/industry-analysis/embedded-security-market-report.
[68] Rust. 2023. Crate ascon. https://docs.rs/ascon/0.4.0/ascon/index.html.
[69] Johnny Saldaña. 2021. *The Coding Manual for Qualitative Researchers*. SAGE publications Ltd.
[70] Georgel M Savin, Ammar Asseri, Josiah Dykstra, Jonathan Goohs, Anthony Melaragno, and William Casey. 2023. Battle ground: Data collection and labeling of ctf games to understand human cyber operators. In *Cyber Security Experimentation and Test Workshop*.
[71] Xinhui Shao, Lan Luo, Zhen Ling, Huaiyu Yan, Yumeng Wei, and Xinwen Fu. 2022. fASLR: Function-based ASLR for resource-constrained IoT systems. In *European Symposium on Research in Computer Security (ESORICS)*.
[72] Jiameng Shi, Le Guan, Wenqiang Li, Dayou Zhang, Ping Chen, and Ning Zhang. 2022. HARM: Hardware-Assisted Continuous Re-randomization for Microcontrollers. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.
[73] Jerome A Solinas. 2000. Efficient arithmetic on Koblitz curves. *Towards a Quarter-Century of Public Key Cryptography: A Special Issue of DESIGNS, CODES AND CRYPTOGRAPHY An International Journal*. (2000).
[74] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry McKay, Mary Baish, and Michael Boyle. 2016. *Recommendation for the entropy sources used for random bit generation*. Technical Report. National Institute of Standards and Technology.
[75] Jesús Soto-Cruz, Erica Ruiz-Ibarra, Javier Vázquez-Castillo, Adolfo Espinoza-Ruiz, Alejandro Castillo-Atoche, and Joaquin Mass-Sanchez. 2024. A Survey of Efficient Lightweight Cryptography for Power-Constrained Microcontrollers. *Technologies* (2024).
[76] Xi Tan, Zheyuan Ma, Sandro Pinto, Le Guan, Ning Zhang, Jun Xu, Zhiqiang Lin, Hongxin Hu, and Ziming Zhao. 2024. SoK: Where's the "up"?! A Comprehensive (bottom-up) Study on the Security of Arm Cortex-M Systems. In *USENIX WOOT Conference on Offensive Technologies (WOOT)*.
[77] Xi Tan and Ziming Zhao. 2023. Sherloc: Secure and holistic control-flow violation detection on embedded systems. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.
[78] The MITRE Corporation. 2024. 2024 Participation Agreement. https://ectf.mitre.org/wp-content/uploads/2023/12/eCTF-Participant-Agreement-2024.pdf.
[79] Giovanni Vigna, Kevin Borgolte, Jacopo Corbetta, Adam Doupe, Yanick Fratantonio, Luca Invernizzi, Dhilung Kirat, and Yan Shoshitaishvili. 2014. Ten Years of iCTF: The Good, The Bad, and The Ugly. In *USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE)*.
[80] Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. 2020. Firmxray: Detecting bluetooth link layer vulnerabilities from bare-metal firmware. In *ACM Conference on Computer and Communications Security (CCS)*.
[81] Jie Zhou, Yufei Du, Zhuojia Shen, Lele Ma, John Criswell, and Robert J Walls. 2020. Silhouette: Efficient protected shadow stacks for embedded systems. In *USENIX Security Symposium*.
[82] Xia Zhou, Jiaqi Li, Wenlong Zhang, Yajin Zhou, Wenbo Shen, and Kui Ren. 2022. OPEC: operation-based security isolation for bare-metal embedded systems. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys)*.